МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению лабораторных работ
по дисциплине «Алгоритмы и структуры данных» для студентов направления подготовки
09.03.04 Программная инженерия
Направленность (профиль)
«Разработка и спопровождение программного обеспечения»

Ставрополь 2026

Оглавление

Введение

Лабораторные работы являются важнейшей составляющей изучения дисциплины «Алгоритмы и структуры данных». Они предназначены для практического закрепления теоретического материала, изложенного в лекциях, и формирования у студентов профессиональных компетенций в области разработки программного обеспечения на языке C++.

Основные задачи лабораторного практикума:

№	Задача	Результат
1	Освоение синтаксиса и возможностей языка C++	Умение писать корректные программы на C++
2	Реализация структур данных «с нуля»	Понимание внутреннего устройства структур данных
3	Применение алгоритмов обработки данных	Умение выбирать и реализовывать подходящие алгоритмы
4	Использование объектно-ориентированного подхода	Навыки проектирования классов и иерархий
5	Работа с файлами и сериализация	Умение сохранять и загружать данные
6	Обработка исключительных ситуаций	Создание надёжных программ

Лабораторный практикум состоит из 24 лабораторных работ.

Общие требования

	Требование	Пояснение
	Самостоятельность выполнения	Работа должна быть выполнена студентом самостоятельно
	Соблюдение варианта	Задания выполняются согласно выданному варианту
	Наличие Git-репозитория	Все работы должны храниться в системе контроля версий Git
	Оформление отчёта	Отчёт должен быть оформлен в соответствии с требованиями
	Защита работы	Студент должен уметь объяснить код и ответить на вопросы

2 Требования к программному коду

№	Требование	Пояснение
1	Стиль оформления	Код должен быть отформатирован единообразно
2	Комментарии	Сложные фрагменты кода должны быть прокомментированы
3	Осмысленные имена	Переменные, функции и классы должны

№	Требование	Пояснение
		иметь понятные имена
4	Отсутствие дублирования	Повторяющийся код должен быть вынесен в функции
5	Обработка ошибок	Программа должна корректно реагировать на ошибки

Требования к отчёту

Каждый отчёт по лабораторной работе должен содержать:

Титульный лист с названием работы, ФИО студента, номером группы, вариантом

Цель работы (формулируется самостоятельно)

Теоретическая часть (основные понятия по теме)

Постановка задачи (текст задания согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода программы)

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Рекомендации по организации работы

1 Планирование времени

Этап	Рекомендуемое время
Изучение теоретического материала	20% времени
Написание кода	50% времени

Этап	Рекомендуемое время
Отладка и тестирование	20% времени
Оформление отчёта	10% времени

Работа с Git

Рекомендуемая структура коммитов:

feat: добавлен базовый класс MediaItem

feat: добавлен класс Book

feat: добавлена сериализация

fix: исправлена ошибка в поиске

docs: обновлена документация

test: добавлены тесты

Рекомендуемая частота коммитов: не реже одного раза в день при активной работе над проектом.

3 Работа над ошибками

При возникновении ошибок рекомендуется:

Внимательно прочитать сообщение об ошибке - компилятор часто подсказывает причину

Использовать отладчик - пошаговое выполнение помогает найти логические ошибки

Использовать отладочную печать - вывод промежуточных значений

Проверять граничные случаи - многие ошибки проявляются на границах данных

Список рекомендуемой литературы

Основная литература

Павловская Т.А., Щупак Ю.А. «C/C++. Структурное и объектно-ориентированное программирование: Практикум». - СПб.: Питер, 2011.

Лафоре Р. «Объектно-ориентированное программирование в C++». - СПб.: Питер, 2011.

Страуструп Б. «Язык программирования C++». - М.: Бином, 2008.

Дополнительная литература

Саттер Г. «Решение сложных задач на C++». - М.: Вильямс, 2002.

Мейерс С. «Эффективное использование C++». - М.: ДМК Пресс, 2006.

Кнут Д. «Искусство программирования». - М.: Вильямс, 2010.

Электронные ресурсы

cppreference.com - справочник по C++

learncpp.com - учебник по C++

geeksforgeeks.org - примеры алгоритмов и структур данных

Выполнение лабораторных работ по дисциплине «Алгоритмы и структуры данных» является важнейшим этапом формирования профессиональных компетенций будущего программиста. В процессе работы студенты:

1. Осваивают синтаксис и возможности языка C++
2. Учатся реализовывать структуры данных «с нуля»
3. Приобретают навыки разработки алгоритмов
4. Осваивают принципы объектно-ориентированного программирования
5. Создают собственный программный проект
6. Регулярная работа с Git-репозиторием формирует навыки командной разработки и позволяет сохранить все результаты обучения для будущего портфолио.

Лабораторная работа №1

Основы работы с файловыми потоками. Текстовые файлы

Цель работы:

1. Освоить базовые операции чтения и записи текстовых файлов на языке C++
2. Научиться использовать файловые потоки ifstream и ofstream для работы с текстовыми файлами
3. Получить навыки обработки числовых и текстовых данных из текстовых файлов
4. Освоить методы проверки состояния потоков и обработки ошибок при работе с файлами
5. Научиться сохранять результаты обработки данных в новые текстовые файлы

Теоретическое обоснование

1. Понятие файловых потоков

Поток (stream) — это абстракция, представляющая последовательный поток данных. В C++ для работы с файлами используются три основных класса из библиотеки <fstream>:

Класс	Назначение	Направление
ifstream (input file stream)	Только для чтения из файла	Файл → Программа
ofstream (output file stream)	Только для записи в файл	Программа → Файл
fstream (file stream)	Для чтения и записи	Файл ↔ Программа

Аналогия из реальной жизни: Представьте, что вы читаете книгу (ifstream — вы только читаете) или пишете в тетрадь (ofstream — вы только пишете).

2. Жизненный цикл работы с файлом

Работа с файлом всегда проходит через несколько обязательных этапов:

1. Создание объекта потока ↓
2. Открытие файла (open() или через конструктор) ↓
3. Проверка успешности открытия (is_open()) ↓
4. Чтение/запись данных (>>, <<, getline) ↓
5. Закрытие файла (close())

Важно: Даже если вы забудете закрыть файл, деструктор потока сделает это автоматически.

3. Режимы открытия файлов

Режимы открытия определяют, как именно будет открыт файл. Их можно комбинировать с помощью оператора | (побитовое ИЛИ).

Флаг	Описание
ios::in	Открыть для чтения (по умолчанию для ifstream)
ios::out	Открыть для записи (по умолчанию для ofstream)
ios::app	Добавление в конец файла (данные не удаляются)
ios::trunc	Очистить файл при открытии (удалить старое содержимое)
ios::ate	Перейти в конец файла сразу после открытия

Примеры комбинаций:

```
// Открыть для дозаписи (данные добавляются в конец)
ofstream out("log.txt", ios::app);

// Открыть для чтения и записи (файл должен существовать)
fstream file("data.txt", ios::in | ios::out);

// Открыть для записи с очисткой (перезапись)
ofstream out("report.txt", ios::out | ios::trunc);
```

4. Состояния потока и проверка ошибок

У каждого потока есть флаги состояния, которые можно проверить специальными методами:

Метод	Возвращает true, если...	Когда использовать
good()	Всё в порядке, можно работать	После открытия или операции
eof()	Достигнут конец файла	При чтении данных
fail()	Ошибка форматирования (например, буква вместо числа)	При преобразовании типов
bad()	Критическая ошибка (повреждён поток)	Редко, но нужно проверять

Практический шаблон проверки:

```
ifstream in("data.txt");
if (!in.is_open()) {
    cerr << "Ошибка: не удалось открыть файл!" << endl;
    return 1;
}

int number;
while (in >> number) {
    // Успешное чтение
}

if (in.eof()) {
    cout << "Достигнут конец файла" << endl;
} else if (in.fail()) {
    cout << "Ошибка чтения (неверный формат)" << endl;
}
```

5. Запись в текстовый файл

Для записи в файл используется оператор << (аналогично cout):

```
#include <fstream>
using namespace std;

int main() {
    ofstream out("output.txt");  // создаём поток и открываем файл

    if (out.is_open()) {
        out << "Hello, World!" << endl;
```

```
        out << "Число: " << 42 << endl;
        out << "Дробь: " << 3.14159 << endl;
        out.close();
    }

    return 0;
}
```

Режимы записи:

1. ofstream out("file.txt") — перезаписывает файл (стирает старое содержимое)
2. ofstream out("file.txt", ios::app) — добавляет в конец файла

6. Чтение из текстового файла

Чтение с помощью оператора >>

Оператор >> читает данные, разделённые пробелами, табуляциями или переводами строк:

```
ifstream in("numbers.txt");
int a, b, c;
in >> a >> b >> c;  // читает три числа из файла
```

Чтение целых строк с помощью getline()

Функция getline() читает строку целиком, включая пробелы:

```
ifstream in("text.txt");
string line;
while (getline(in, line)) {
    cout << line << endl;  // выводим каждую строку
}
```

Чтение символов с помощью get()

Метод get() читает файл посимвольно:

```
ifstream in("file.txt");
char ch;
while (in.get(ch)) {
    // обрабатываем символ ch
}
```

7. Полный пример: обработка данных из файла

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>

using namespace std;

int main() {
    setlocale(LC_ALL, "ru");

    // === ЗАПИСЬ ДАННЫХ В ФАЙЛ ===
    ofstream out("products.txt");
    if (!out.is_open()) {
        cerr << "Ошибка создания файла!" << endl;
        return 1;
    }
```

```
out << "Хлеб 45.50 10" << endl;
out << "Молоко 89.90 5" << endl;
out << "Масло 120.00 3" << endl;
out.close();

cout << "Данные записаны в файл products.txt" << endl;

// === ЧТЕНИЕ И ОБРАБОТКА ДАННЫХ ИЗ ФАЙЛА ===
ifstream in("products.txt");
if (!in.is_open()) {
    cerr << "Ошибка открытия файла!" << endl;
    return 1;
}

string name;
double price;
int quantity;
double totalCost = 0;

cout << "\nСписок товаров:" << endl;

while (in >> name >> price >> quantity) {
    double cost = price * quantity;
    totalCost += cost;
    cout << "  " << name << ": " << quantity << " шт. x "
         << price << " руб. = " << cost << " руб." << endl;
```

```
    }
    in.close();

    cout << "Общая стоимость запасов: " << totalCost << " руб." << endl;

    return 0;
}
```

Примеры выполнения практической части

Пример 1. Подсчёт элементов в файле (базовый уровень)

Задача: Дан файл numbers.txt, содержащий целые числа (по одному в строке). Найти сумму всех чисел.

```
#include <iostream>
#include <fstream>

int main() {
    setlocale(LC_ALL, "ru");

    // Создаём тестовый файл
    ofstream out("numbers.txt");
    out << 10 << endl << 20 << endl << 30 << endl << 40 << endl;
    out.close();
```

```
    // Читаем файл и считаем сумму
    ifstream in("numbers.txt");

    if (!in.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return 1;
    }

    int number;
    int sum = 0;

    while (in >> number) {
        sum += number;
    }

    in.close();

    cout << "Сумма чисел: " << sum << endl;

    return 0;
}
```

Сумма чисел: 100

Пример 2. Фильтрация символов (базовый уровень)

Задача: Дан файл input.txt с произвольным текстом. Создать новый файл, содержащий только буквы из исходного файла.

```cpp
#include <iostream>
#include <fstream>
#include <cctype>

int main() {
    setlocale(LC_ALL, "ru");

    // Создаём тестовый файл
    ofstream out("input.txt");
    out << "Hello123 World! Привет456 Мир!" << endl;
    out.close();

    // Читаем исходный файл и фильтруем буквы
    ifstream in("input.txt");
    ofstream letters("letters.txt");

    if (!in.is_open()) {
        cerr << "Ошибка открытия входного файла!" << endl;
        return 1;
    }

    char ch;
    while (in.get(ch)) {
        if (isalpha(ch)) {
            letters.put(ch);
```

```
        }
    }

    in.close();
    letters.close();

    cout << "Буквы сохранены в файл letters.txt" << endl;

    return 0;
}
```

Буквы сохранены в файл letters.txt

Содержимое letters.txt:

HelloWorldПриветМир

Индивидуальные задания

Базовый уровень (15 вариантов)

Общее задание: Дан файл вещественных чисел a.txt. Выполнить задание согласно варианту. Результат сохранить в файл result.txt.

№ вар.	**Задание**
1	Найти количество отрицательных и количество положительных элементов
2	Найти количество нулевых элементов и произведение элементов от 0 до 1
3	Найти количество нулевых элементов и сумму отрицательных элементов

№ вар.	Задание
4	Найти количество элементов равных 5 и сумму положительных элементов
5	Найти количество нулевых элементов и сумму положительных элементов
6	Найти количество положительных элементов и сумму положительных элементов
7	Найти среднее арифметическое всех элементов
8	Найти максимальный и минимальный элемент
9	Найти сумму квадратов всех элементов
10	Найти количество элементов, больших заданного числа K (K вводится)
11	Найти произведение всех положительных элементов
12	Найти сумму всех отрицательных элементов
13	Найти количество элементов, попадающих в интервал [A, B] (A, B вводятся)
14	Переписать положительные элементы в файл positive.txt
15	Переписать отрицательные элементы в файл negative.txt

Средний уровень (15 вариантов)

Общее задание: Дан текстовый файл data.txt с произвольным содержимым. Выполнить задание согласно варианту. Результат сохранить в новый файл.

№ вар.	**Задание**
1	Подсчитать количество строк в файле
2	Подсчитать количество слов в файле (слова разделены пробелами)
3	Подсчитать количество символов в файле (исключая пробелы и переводы строк)
4	Найти самую длинную строку в файле и вывести её длину
5	Найти строку, содержащую наибольшее количество гласных букв
6	Заменить все цифры в файле на символ '*'
7	Заменить все маленькие латинские буквы на большие
8	Заменить все большие латинские буквы на маленькие
9	Удалить из файла все строки, содержащие заданное ключевое слово
10	Скопировать из файла только строки, начинающиеся с заглавной буквы

№ вар.	Задание
11	Создать новый файл, содержащий только уникальные строки из исходного
12	Отсортировать строки файла в алфавитном порядке
13	Перевернуть каждую строку файла (реверс)
14	Объединить содержимое двух файлов в третий, чередуя строки
15	Найти и вывести все строки, содержащие заданный фрагмент текста

Продвинутый уровень (15 вариантов)

Общее задание: Дан текстовый файл с данными в формате «ключ=значение» (каждая запись на новой строке). Выполнить задание согласно варианту.

Пример формата файла (students.txt):

Имя=Иван
Фамилия=Петров
Возраст=20
Группа=ИС-21
Оценка1=85
Оценка2=90
Оценка3=78

№ вар.	Задание
1	Найти все записи, у которых значение начинается с буквы «А»
2	Преобразовать файл так, чтобы все имена ключей стали заглавными
3	Преобразовать файл так, чтобы все значения стали заглавными
4	Создать новый файл, содержащий только записи с числовыми значениями
5	Изменить значение ключа «Возраст» на 1 (увеличить на 1)
6	Найти все записи, где значение содержит заданный фрагмент текста
7	Создать новый файл, где ключи и значения поменяны местами
8	Удалить из файла все записи, у которых ключ начинается с «Х»
9	Изменить значение ключа «Цена» на 10% (увеличить)
10	Найти ключ с максимальным числовым значением
11	Создать новый файл, отсортированный по именам ключей

№ вар.	Задание
12	Изменить значение ключа «Количество» на заданное число (ввод с клавиатуры)
13	Найти сумму всех числовых значений (игнорируя текст)
14	Заменить все пробелы в значениях на символ подчёркивания
15	Создать новый файл, содержащий только записи, где значение — число больше 100

Контрольные вопросы

1. Какие классы используются для работы с текстовыми файлами в C++?
2. В чём разница между ifstream и ofstream?
3. Для чего нужен метод is_open()?
4. Как проверить, достигнут ли конец файла?
5. Что произойдёт, если попытаться открыть несуществующий файл для чтения?
6. Как открыть файл в режиме добавления в конец (append)?
7. Зачем нужно закрывать файл методом close()?
8. В чём разница между оператором >> и функцией getline() при чтении строк?
9. Какие флаги состояния потока вы знаете? Что означает каждый из них?
10. Как сбросить флаги ошибок потока?
11. Как прочитать файл посимвольно?
12. Как определить размер файла в байтах?
13. Что такое формат CSV и как с ним работать?
14. Почему при чтении строк с пробелами оператор >> не подходит?
15. Как записать в файл несколько значений в одной строке?

Требования к отчёту

1. Титульный лист с названием работы, ФИО студента, номером группы, вариантом

2. Цель работы (формулируется самостоятельно на основе предложенной)
3. Теоретическая часть (основные понятия: файловые потоки, режимы открытия, состояния потока)
4. Постановка задачи (текст задания согласно варианту)
5. Листинг программы с комментариями
6. Результаты работы (скриншоты вывода программы и содержимого файлов)
7. Ответы на контрольные вопросы
8. Выводы (что нового узнали, какие навыки приобрели, какие были трудности)
9. Ссылка на Git-репозиторий с кодом программы

Критерии оценивания

Базовый уровень (максимум 60 баллов)

Критерий	Баллы
Корректное создание и открытие файлов	10
Правильное чтение данных из файла	15
Корректная обработка данных по заданию	15
Корректная запись результатов в файл	10
Проверка состояния потока и обработка ошибок	5
Код отформатирован, есть комментарии	5

Средний уровень (максимум 80 баллов)

Критерий	Баллы
Все требования базового уровня	40
Работа с несколькими файлами	15
Обработка строк и сложных форматов	15
Обработка особых случаев (пустой файл,	10

Критерий	Баллы
неверный формат)	

Продвинутый уровень (максимум 100 баллов)

Критерий	Баллы
Все требования среднего уровня	60
Работа с форматом «ключ=значение»	15
Сложная обработка данных (преобразования, фильтрация)	15
Полная обработка ошибок и исключительных ситуаций	10

Штрафные баллы

Нарушение	Штраф
Файлы не закрыты	-5
Отсутствие проверки открытия файла	-10
Отсутствие комментариев в коде	-5
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка (традиционная)	Баллы
Отлично	90–100

Оценка (традиционная)	Баллы
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №2

Бинарные файлы и прямое позиционирование

Цель работы:

1. Освоить работу с бинарными файлами в C++ с использованием методов read() и write()
2. Научиться выполнять прямое позиционирование в файле с помощью seekg(), seekp(), tellg(), tellp()
3. Получить навыки записи и чтения простых типов данных и массивов в бинарные файлы
4. Освоить обновление данных в бинарном файле без перезаписи всего файла
5. Научиться обрабатывать ошибки при работе с бинарными файлами

Теоретическое обоснование

1. Текстовые vs Бинарные файлы

Характеристика	**Текстовые файлы**	**Бинарные файлы**
Представление данных	Символы (ASCII/UTF)	Байты "как есть" в памяти
Человекочитаемость	Да (можно открыть в блокноте)	Нет (видны "иероглифы")

Характеристика	Текстовые файлы	Бинарные файлы
Размер	Больше (например, число 12345 → 5 байт)	Меньше (число 12345 → 4 байта)
Скорость	Медленнее (нужно форматирование)	Быстрее (нет форматирования)
Прямой доступ	Сложно (переменная длина строк)	Просто (фиксированный размер данных)

Пример: число 12345 в разных форматах

В памяти (4 байта): 0x00 0x00 0x30 0x39

В текстовом файле: '1' '2' '3' '4' '5' (5 байт)

В бинарном файле: 0x00 0x00 0x30 0x39 (4 байта)

2. Режим открытия бинарных файлов

Для работы с бинарными файлами используется флаг std::ios::binary:

```
// Открытие для записи в бинарном режиме
ofstream out("data.bin", ios::binary);

// Открытие для чтения в бинарном режиме
ifstream in("data.bin", ios::binary);

// Открытие для чтения и записи
fstream file("data.bin", ios::in | ios::out | ios::binary);
```

Почему флаг binary важен?

В Windows символ '\n' в текстовом режиме преобразуется в '\r' + '\n', что нарушает бинарные данные. Флаг binary отключает эти преобразования.

3. Методы read() и write()

Для побайтового чтения и записи используются методы read() и write():

```
// Запись: адрес памяти и количество байт
ostream& write(const char* buffer, streamsize size);

// Чтение
istream& read(char* buffer, streamsize size);
```

Пример записи числа:

```
int value = 12345;
ofstream out("number.bin", ios::binary);
out.write((const char*)&value, sizeof(value));
```

Пример чтения числа:

```
int value;
ifstream in("number.bin", ios::binary);
in.read((char*)&value, sizeof(value));
```

4. Запись и чтение массива

Массивы можно записывать целиком одной операцией:

```
// Запись массива
int arr[] = {10, 20, 30, 40, 50};
ofstream out("array.bin", ios::binary);
```

```
out.write((const char*)arr, sizeof(arr));

// Чтение массива
int readArr[5];
ifstream in("array.bin", ios::binary);
in.read((char*)readArr, sizeof(readArr));
```

5. Позиционирование в файле

Для прямого доступа к данным используются методы позиционирования:

Метод	Назначение
tellg()	Текущая позиция для чтения
tellp()	Текущая позиция для записи
seekg(offset, dir)	Установить позицию чтения
seekp(offset, dir)	Установить позицию записи

Направления смещения:

Флаг	Значение
ios::beg	От начала файла
ios::cur	От текущей позиции
ios::end	От конца файла

Примеры позиционирования:

```
// Переместиться на 100 байт от начала
file.seekg(100, ios::beg);
```

```
// Переместиться на 20 байт назад от текущей позиции
file.seekg(-20, ios::cur);

// Переместиться на 50 байт от конца
file.seekg(-50, ios::end);

// Перейти в начало
file.seekg(0, ios::beg);

// Перейти в конец (для добавления)
file.seekp(0, ios::end);
```

6. Прямой доступ к элементам

Если все элементы имеют одинаковый размер, можно обращаться к любому элементу по индексу:

```
// Размер одного элемента
int elementSize = sizeof(int);

// Чтение элемента с индексом index (нумерация с 0)
file.seekg(index * elementSize, ios::beg);
file.read((char*)&value, elementSize);

// Обновление элемента
file.seekp(index * elementSize, ios::beg);
file.write((const char*)&value, elementSize);
```

7. Обработка ошибок

При работе с бинарными файлами важно проверять состояние потока:

```
ifstream file("data.bin", ios::binary);
```

```
if (!file.is_open()) {
cerr << "Ошибка открытия файла!" << endl;
return 1;
}

// Чтение данных
int value;
while (file.read((char*)&value, sizeof(value))) {
// Успешное чтение
cout << value << endl;
}

if (file.eof()) {
cout << "Достигнут конец файла" << endl;
} else if (file.fail()) {
cerr << "Ошибка чтения (повреждённые данные)" << endl;
}

file.clear(); // Сброс флагов для продолжения работы
```

8. Полный пример: работа с массивом чисел в бинарном файле

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {
setlocale(LC_ALL, "ru");
```

```
const char* filename = "numbers.bin";

// === ЗАПИСЬ МАССИВА В ФАЙЛ ===
int numbers[] = {10, 20, 30, 40, 50};
int count = sizeof(numbers) / sizeof(numbers[0]);

ofstream outFile(filename, ios::binary);
if (!outFile) {
cerr << "Ошибка создания файла!" << endl;
return 1;
}

outFile.write((const char*)numbers, sizeof(numbers));
outFile.close();
cout << "Массив записан в файл" << endl;

// === ЧТЕНИЕ ВСЕХ ЭЛЕМЕНТОВ ===
ifstream inFile(filename, ios::binary);
if (!inFile) {
cerr << "Ошибка открытия файла!" << endl;
return 1;
}

int value;
cout << "\nВсе элементы файла:" << endl;
while (inFile.read((char*)&value, sizeof(value))) {
cout << value << " ";
}
cout << endl;
inFile.close();
```

```
// === ПРЯМОЙ ДОСТУП К 3-му ЭЛЕМЕНТУ (индекс 2) ===
ifstream directFile(filename, ios::binary);
directFile.seekg(2 * sizeof(int), ios::beg);
directFile.read((char*)&value, sizeof(int));
directFile.close();

cout << "\n3-й элемент: " << value << endl;

// === ОБНОВЛЕНИЕ 2-го ЭЛЕМЕНТА (индекс 1) ===
int newValue = 99;
fstream updateFile(filename, ios::in | ios::out | ios::binary);
updateFile.seekp(1 * sizeof(int), ios::beg);
updateFile.write((const char*)&newValue, sizeof(int));
updateFile.close();

// === ПРОВЕРКА ПОСЛЕ ОБНОВЛЕНИЯ ===
ifstream checkFile(filename, ios::binary);
cout << "\nМассив после обновления 2-го элемента:" << endl;
while (checkFile.read((char*)&value, sizeof(value))) {
cout << value << " ";
}
cout << endl;
checkFile.close();

return 0;
}
```

Ожидаемый вывод:

Массив записан в файл

Все элементы файла:

10 20 30 40 50

3-й элемент: 30

Массив после обновления 2-го элемента:
10 99 30 40 50

Примеры выполнения практической части

Пример 1. Запись и чтение массива чисел (базовый уровень)

Задача: Создать массив целых чисел, записать его в бинарный файл, затем прочитать и вывести на экран.

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {
setlocale(LC_ALL, "ru");

// Создаём массив чисел
int numbers[] = {15, 25, 35, 45, 55, 65, 75};
int size = sizeof(numbers) / sizeof(numbers[0]);

// Запись в бинарный файл
ofstream outFile("data.bin", ios::binary);
if (!outFile) {
cerr << "Ошибка создания файла!" << endl;
return 1;
```

```
}

outFile.write((const char*)numbers, sizeof(numbers));
outFile.close();
cout << "Массив из " << size << " чисел записан в файл data.bin" << endl;

// Чтение из бинарного файла
ifstream inFile("data.bin", ios::binary);
if (!inFile) {
cerr << "Ошибка открытия файла!" << endl;
return 1;
}

int readNumbers[7];
inFile.read((char*)readNumbers, sizeof(readNumbers));
inFile.close();

// Вывод прочитанных данных
cout << "Прочитанные числа: ";
for (int i = 0; i < size; i++) {
cout << readNumbers[i] << " ";
}
cout << endl;

return 0;
}
```

Ожидаемый вывод:

Массив из 7 чисел записан в файл data.bin

Прочитанные числа: 15 25 35 45 55 65 75

Пример 2. Прямой доступ по индексу (средний уровень)

Задача: В файле с 10 целыми числами найти и вывести число с заданным индексом, используя прямое позиционирование.

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {
setlocale(LC_ALL, "ru");

// Создаём и записываем массив из 10 чисел
int numbers[10];
for (int i = 0; i < 10; i++) {
numbers[i] = (i + 1) * 10;  // 10, 20, 30, ..., 100
}

ofstream outFile("numbers.bin", ios::binary);
outFile.write((const char*)numbers, sizeof(numbers));
outFile.close();
cout << "Создан файл с числами: 10, 20, 30, ..., 100" << endl;

// Запрашиваем индекс
int index;
cout << "\nВведите индекс элемента (0-9): ";
cin >> index;

// Прямой доступ к элементу
ifstream inFile("numbers.bin", ios::binary);
if (!inFile) {
```

```
cerr << "Ошибка открытия файла!" << endl;
return 1;
}

// Перемещаемся к нужному элементу
inFile.seekg(index * sizeof(int), ios::beg);

int value;
inFile.read((char*)&value, sizeof(int));
inFile.close();

cout << "Элемент с индексом " << index << " = " << value << endl;

return 0;
}
```

Ожидаемый вывод:

Создан файл с числами: 10, 20, 30, ..., 100

Введите индекс элемента (0-9): 5
Элемент с индексом 5 = 60

Пример 3. Обновление элемента (продвинутый уровень)

Задача: В файле с целыми числами заменить элемент с заданным индексом на новое значение.

```
#include <iostream>
#include <fstream>

using namespace std;
```

```
int main() {
setlocale(LC_ALL, "ru");

// Создаём и записываем массив
int numbers[] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
int size = 10;

ofstream outFile("data.bin", ios::binary);
outFile.write((const char*)numbers, sizeof(numbers));
outFile.close();
cout << "Исходный массив: ";
for (int i = 0; i < size; i++) {
cout << numbers[i] << " ";
}
cout << endl;

// Запрашиваем индекс и новое значение
int index, newValue;
cout << "\nВведите индекс элемента для замены (0-9): ";
cin >> index;
cout << "Введите новое значение: ";
cin >> newValue;

// Открываем файл для чтения и записи
fstream file("data.bin", ios::in | ios::out | ios::binary);
if (!file) {
cerr << "Ошибка открытия файла!" << endl;
return 1;
}

// Перемещаемся к элементу и обновляем его
```

```
file.seekp(index * sizeof(int), ios::beg);
file.write((const char*)&newValue, sizeof(int));
file.close();

// Проверяем результат
ifstream checkFile("data.bin", ios::binary);
cout << "\nМассив после обновления: ";
int val;
while (checkFile.read((char*)&val, sizeof(val))) {
cout << val << " ";
}
cout << endl;
checkFile.close();

return 0;
}
```

Ожидаемый вывод:

Исходный массив: 10 20 30 40 50 60 70 80 90 100

Введите индекс элемента для замены (0-9): 4
Введите новое значение: 999

Массив после обновления: 10 20 30 40 999 60 70 80 90 100

Индивидуальные задания

Базовый уровень (15 вариантов)

Общее задание: Создать массив из 10 целых чисел (значения задать в коде). Записать массив в бинарный файл. Затем прочитать все числа из файла и вывести их на экран. Вычислить сумму всех элементов.

№ вар.	Массив (значения)
1	5, 10, 15, 20, 25, 30, 35, 40, 45, 50
2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20
3	1, 3, 5, 7, 9, 11, 13, 15, 17, 19
4	11, 22, 33, 44, 55, 66, 77, 88, 99, 110
5	100, 90, 80, 70, 60, 50, 40, 30, 20, 10
6	3, 6, 9, 12, 15, 18, 21, 24, 27, 30
7	7, 14, 21, 28, 35, 42, 49, 56, 63, 70
8	12, 24, 36, 48, 60, 72, 84, 96, 108, 120
9	25, 50, 75, 100, 125, 150, 175, 200, 225, 250
10	1, 4, 9, 16, 25, 36, 49, 64, 81, 100
11	2, 3, 5, 7, 11, 13, 17, 19, 23, 29
12	-5, -4, -3, -2, -1, 0, 1, 2, 3, 4
13	0, 0, 0, 0, 0, 1, 1, 1, 1, 1
14	8, 8, 8, 8, 8, 8, 8, 8, 8, 8
15	1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000

Средний уровень (15 вариантов)

Общее задание: Создать бинарный файл с 20 целыми числами (заполнить случайными числами от 1 до 100). Реализовать функцию, которая по заданному

индексу находит элемент и изменяет его значение. Пользователь вводит индекс и новое значение.

№ вар.	Дополнительное задание
1	После изменения найти сумму всех элементов файла
2	После изменения найти количество чётных чисел
3	После изменения найти количество нечётных чисел
4	После изменения найти максимальный элемент
5	После изменения найти минимальный элемент
6	После изменения найти среднее арифметическое
7	После изменения найти количество чисел больше 50
8	После изменения найти количество чисел меньше 50
9	После изменения найти произведение всех элементов
10	После изменения найти количество нулей
11	После изменения найти сумму чётных элементов

№ вар.	Дополнительное задание
12	После изменения найти сумму нечётных элементов
13	После изменения найти количество элементов, кратных 3
14	После изменения найти количество элементов, кратных 5
15	После изменения найти разность между max и min

Продвинутый уровень (15 вариантов)

Общее задание: Реализовать программу для работы с массивом целых чисел, хранящимся в бинарном файле. Программа должна предоставлять меню:

Создать файл - создать новый бинарный файл с указанным количеством элементов (заполнить нулями)

Показать все - вывести все элементы файла

Найти элемент - найти элемент по индексу (прямой доступ)

Изменить элемент - изменить элемент по индексу

Вставить элемент - вставить элемент в конец файла (увеличить размер)

Удалить последний - удалить последний элемент (уменьшить размер)

Выход

Размер файла должен храниться в начале файла (первый элемент - количество элементов).

№ вар.	Дополнительное требование
1	Добавить возможность вставки элемента по индексу

№ вар.	Дополнительное требование
2	Добавить возможность удаления элемента по индексу
3	Добавить поиск элемента по значению
4	Добавить подсчёт суммы всех элементов
5	Добавить подсчёт среднего арифметического
6	Добавить поиск максимального элемента
7	Добавить поиск минимального элемента
8	Добавить возможность сортировки по возрастанию
9	Добавить возможность сортировки по убыванию
10	Добавить подсчёт количества чётных элементов
11	Добавить подсчёт количества нечётных элементов
12	Добавить возможность замены всех элементов по условию
13	Добавить возможность удвоения всех элементов
14	Добавить возможность сохранения копии файла
15	Добавить возможность сравнения двух файлов

Контрольные вопросы

1. Чем бинарные файлы отличаются от текстовых?
2. Для чего нужен флаг ios::binary?
3. Как записать целое число в бинарный файл?
4. Как прочитать целое число из бинарного файла?

5. Что делают методы read() и write()?
6. Почему при записи используется приведение к (const char*)?
7. Что такое tellg() и tellp()?
8. Что такое seekg() и seekp()?
9. Какие есть направления для позиционирования (ios::beg, ios::cur, ios::end)?
10. Как прочитать 5-й элемент файла, не читая первые 4?
11. Как обновить элемент в бинарном файле без перезаписи всего файла?
12. Как определить размер файла в байтах?
13. Как проверить, достигнут ли конец файла при чтении?
14. Что произойдёт, если попытаться прочитать данные неверного размера?
15. Как записать массив в бинарный файл целиком?

Требования к отчёту

Титульный лист с названием работы, ФИО студента, номером группы, вариантом

Цель работы (формулируется самостоятельно на основе предложенной)

Теоретическая часть (основные понятия: бинарные файлы, read/write, позиционирование)

Постановка задачи (текст задания согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода программы, демонстрация работы с файлом)

Ответы на контрольные вопросы

Выводы (что нового узнали, какие навыки приобрели, какие были трудности)

Ссылка на Git-репозиторий с кодом программы

Критерии оценивания

Базовый уровень (максимум 60 баллов)

Критерий	Баллы
Корректное создание бинарного файла	10
Правильная запись массива в файл	15
Правильное чтение массива из файла	15

Критерий	Баллы
Корректный вывод прочитанных данных	10
Проверка состояния потока и обработка ошибок	5
Код отформатирован, есть комментарии	5

Средний уровень (максимум 80 баллов)

Критерий	Баллы
Все требования базового уровня	40
Использование прямого позиционирования (seekg/seekp)	20
Обновление элемента без перезаписи всего файла	20

Продвинутый уровень (максимум 100 баллов)

Критерий	Баллы
Все требования среднего уровня	60
Реализация полноценного меню (7+ пунктов)	15
Динамическое изменение размера файла (вставка/удаление)	15
Хранение размера файла в начале файла	5

Критерий	Баллы
Обработка всех возможных ошибок	5

Штрафные баллы

Нарушение	Штраф
Отсутствие проверки открытия файла	-10
Файл не закрыт	-5
Отсутствие комментариев в коде	-5
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка (традиционная)	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа № 3.

Составные типы данных. Структуры

Цель работы: Разработка программ языке C++ с использованием структур и перечислений.

Теоретическое обоснование

Типы данных, определяемые пользователем

На основании простых типов данных, массивов можно построить тип данных, имеющих более сложную структуру, т.е. типы данных, определяемые пользователем.

1. Переименование типов (typedef)

Для того чтобы сделать программу более ясной, можно задать типу новое имя с помощью служебного слова **typedef**. Оно позволяет программисту *задать псевдоним для какого-то известного типа данных*.

Формат оператора переименования типа:

typedef тип новое_ имя_типа [размерность];

Ниже приведён простой пример, который демонстрирует данную возможность (листинг 1):

Листинг 1

```
#include <iostream>

using namespace std;

int main() {
   typedef int number_of_days_in_month;
   number_of_days_in_month days;

   cout << "Enter the number of days in current month: " << endl;
   cin >> days;
   cout << "You entered: " << days << endl;
}
```

Посмотрите, как используется ключевое слово typedef: после typedef мы указали известный встроенный тип данных int и далее - псевдоним для этого типа с именем number_of_days_in_month. Это позволяет нам на следующей же строке определить переменную days уже с нашим новым типом данных number_of_days_in_month.

Далее программа демонстрирует запрос ввода числа с клавиатуры, которое будет содержать текущее количество дней в месяце и выведет введённое пользователем

значение на экран консоли. Если после запуска программы ввести значение 31, то результат работы программы на экране консоли выглядит так (рисунок 1):

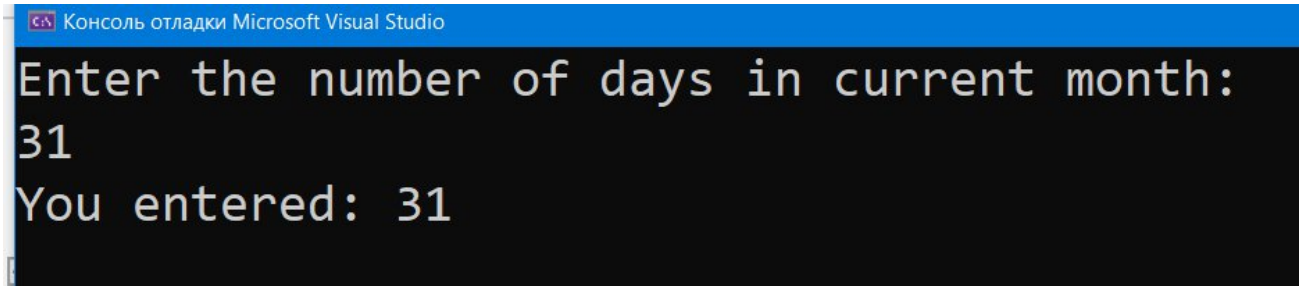
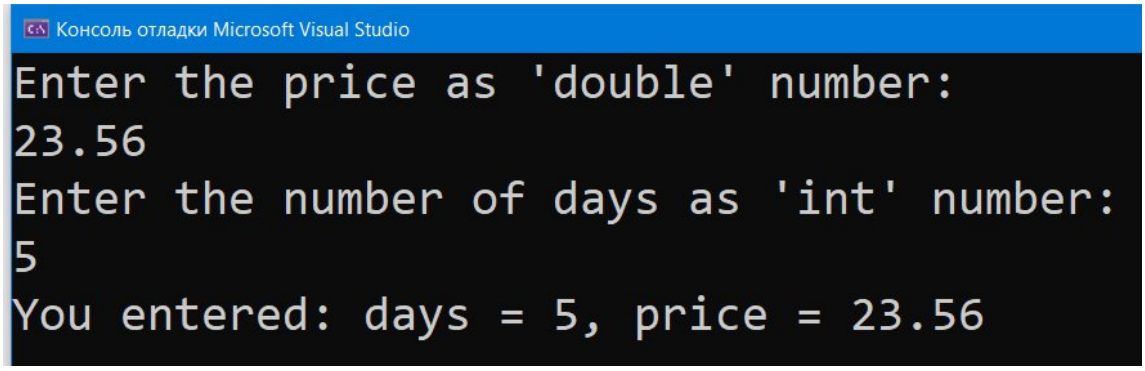


Рисунок 1 – Результат программы (листинг 1)

В примере выше мы задали псевдоним типа внутри метода main, который является входной точкой нашей программы. Точно так же мы можем определить псевдонимы для типов данных до метода main (листинг 2, рисунок 2):

Листинг 2

```
#include <iostream>

using namespace std;

typedef int integer_number;
typedef double double_number;

int main() {
    integer_number days;
    double_number price;

    cout << "Enter the price as 'double' number: " << endl;
    cin >> price;
    cout << "Enter the number of days as 'int' number: " << endl;
    cin >> days;
    cout << "You entered: days = " << days << ", price = " << price << endl;
}
```



2. Перечислимые типы

Перечислимые типы описываются для переменных, принимающих только

определенные значения из заданного набора. Они описываются с помощью служебного слова **enum**.

Пример:

```
enum Day{ sun, mon, tues, weds, thur, fri, sat };
```

По умолчанию элементы в перечислении нумеруются с 0.

```
#include <iostream>
using namespace std;

enum Day
{   sun,
    mon,
    tues,
    weds,
    thur,
    fri,
    sat
};

void getDayWeek(Day d)
{
    switch (d)
    {
    case sun: case sat: cout << "выходные дни";
        break;
    case mon: cout << "понедельник – трудный день";
        break;
    case tues: cout << "Второй день после понедельника";
        break;
    case weds: cout << "Середина недели";
        break;
    case thur: cout << "Четверг - рыбный день";
        break;
    case fri: cout << "Ура- завтра выходной!";
        break;
    default: cout << "Ошибка";
        break;
    }
}

int main() {
    setlocale(LC_ALL, "rus");
    Day d_week1{ mon };
    Day d_week2{ fri };
    getDayWeek(d_week1);
```

```
    getDayWeek(d_week2);
}
```

Можно не по умолчанию задавать значения элементам перечисления, явно указывать значения в перечислимом типе. Это происходит следующим образом:

enum status {success = 1,wait = -1,error = 0};

Далее описываются переменные этого типа**:**

enum status Proc1_Status, Proc2 _Status;

При необходимости можно описать перечисление как новый тип и использовать его в дальнейшем без слова **enum**.

typedef enum status { success = 1, wait = -1, error = 0} name_status;

Тогда тип **name_status –** новое имя созданного типа и его можно использовать для описания переменных без слова **enum** следующим образом:

name_status Proc1_Status, Proc2_Status;

3. Структуры

В отличие от массива, все элементы которого однотипны, структура может содержать элементы разных типов. В языке C++ структура является видом класса и обладает всеми свойствами.

Описание *структурного шаблона* предшествует описанию структурной переменной. Формат оператора описания структурного шаблона:

struct имя_структурного_шаблона
{
тип_1 имя_элемента_1; тип_2 имя_элемента_2;
...
тип_N имя_элемента_N;
};

Пример:

```
struct Automobile
{
   int year; // год выпуска автомобиля
   int doors; //количество дверей автомобиля
```

```
        double horse_Power; // мощность двигателя (лошадиные силы)
        char model[10]; //название модели
};
```

Описание структурной переменной происходит следующим образом:

struct Automobile my_Car, yur_Car;

Аналогично описание структурного массива или указателя на структуру: struct Automobile Car[10]; // массив для информации о 10 машинах; struct Automobile *taxi; //указатель на структуру;

Для упрощения описания структурных переменных можно ввести новый тип с помощью оператора typedef.

typedef struct Automobile{ int year; doors; double horse_Power; char model [10];

} At_Mble;

Инициализация структурной переменной происходит путем перечисления значений элементов структурной переменной в фигурных скобках в порядке их описания:

struct Automobile my_Car = {2000, 4, 100, ”BMV”}

Для доступа к отдельным элементам (полям) структуры используется операция выбора (.) «точка».

Пример 1:

my_Car . year = 2000; my_Car . doors = 4;

my_Car . horse_Power = 100; my_Car . model = ”BMV”;

Пример 2:

struct Automobile Car[10]; Car [i] . year = 1997;

Car [i] . doors = 5;

Car [i] . horse_Power = 200; Car [i]. model = ”Ford”;

Варианты заданий

Задача №1

1. Определить структурный тип для представления анкеты ребенка, состоящей из его имени, пола и роста. Ввести информацию по 20 детям. Вывести средний рост

мальчиков

2. Составить программу, выводящую на экран ведомость начисленной заработной платы (Ф.И.О., должность, год и дата рождения, заработная плата). Найти среднюю зарплату. И вывести фамилии с зарплатой выше средней.

3. Определить структурный тип для представления информации по горным вершинам, состоящей из названия вершины и ее высоты. Ввести информацию по 20 вершинам. Вывести среднее значение высот всех 20 вершин. Далее вывести названия всех вершин ниже среднего.

4. Составить программу, выводящую на экран ведомость начисленной заработной платы (Ф.И.О., должность, год и дата рождения, заработная плата). Вывести фамилию сотрудника с самой маленькой зарплатой.

5. Составить программу, выводящую на экран ведомость начисленной заработной платы (Ф.И.О., должность, год и дата рождения, заработная плата). Вывести фамилию сотрудника с самой большой зарплатой.

6. Определить структурный тип для представления анкеты ребенка, состоящей из его имени, пола и роста. Ввести информацию по 20 детям. Вывести имя самой высокой девочки.

7. Определить структурный тип для представления информации по горным вершинам, состоящей из названия вершины и ее высоты. Ввести информацию по 50 вершинам. Вывести название самой низкой вершины из всех 50.

8. Определить структурный тип для представления анкеты ребенка, состоящей из его имени, пола и роста. Ввести информацию по 20 детям. Вывести средний рост девочек.

9. Определить структурный тип для представления информации по горным вершинам, состоящей из названия вершины и ее высоты. Ввести информацию по 20 вершинам. Вывести среднее значение высот всех 20 вершин. Далее вывести названия всех вершин выше среднего.

10. Определить структурный тип для представления анкеты ребенка, состоящей из его имени, пола и роста. Ввести информацию по 20 детям. Вывести средний рост всех детей.

11. Определить структурный тип для представления информации по горным

вершинам, состоящей из названия вершины и ее высоты. Ввести информацию по 10 вершинам. Вывести название самой высокой вершины из всех 10.

12. Определить структурный тип для представления анкеты ребенка, состоящей из его имени, пола и роста. Ввести информацию по 20 детям. Вывести средний рост девочек. Далее вывести имена всех девочек выше среднего.

13. Определить структурный тип для представления информации по горным вершинам, состоящей из названия вершины и ее высоты. Ввести информацию по 20 вершинам. Вывести среднее значение высот всех 20 вершин.

14. Определить структурный тип для представления анкеты ребенка, состоящей из его имени, пола и роста. Ввести информацию по 20 детям. Вывести имя самого высокого мальчика. Вывести средний рост мальчиков. Далее вывести имена всех мальчиков ниже среднего.

15. Составить программу, выводящую на экран ведомость начисленной заработной платы (Ф.И.О., должность, год и дата рождения, заработная плата). Вывести все фамилии, начинающиеся на букву «А» и их зарплату .

16. Составить программу, выводящую на экран ведомость начисленной заработной платы (Ф.И.О., должность, год и дата рождения, заработная плата). Вывести дату рождения сотрудника с самой маленькой зарплатой.

Задача №2

1. Определить структурный тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести координаты центра окружности, чей радиус самой большой.

2. Определить структурный тип для представления анкеты жителя, состоящей из его фамилии, названия города, где он проживает, и городского адреса. Адрес состоит из полей: «улица», «дом», «квартира». Ввести информацию по 100 жителям. Вывести фамилии двух любых жителей, которые «По Иронии Судьбы» живут в разных городах, но по одинаковому адресу.

3. Определить структурный тип для представления анкеты студента, состоящей из его фамилии, дня рождения и пола. «День рождения» состоит из полей: «число»,

«месяц», «год». Ввести информацию по 25 студентам из группы. Вывести фамилию самого старшего мальчика из группы.

4. Определить структурный тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести координаты центра окружности, чей радиус самой маленький

5. Определить структурный тип для представления анкеты жителя, состоящей из его фамилии, названия города, где он проживает, и городского адреса. Адрес состоит из полей: «улица», «дом», «квартира». Ввести информацию по 100 жителям. Вывести фамилии жителей, которые живут в одном городе с первым жителем из списка.

6. Определить структурный тип для представления анкеты студента, состоящей из его фамилии, дня рождения и пола. «День рождения» состоит из полей: «число», «месяц», «год». Ввести информацию по 25 студентам из группы. Вывести все фамилии девочек, родившихся в декабре.

7. Определить структурный тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести радиус окружности, чей центр самый удаленный от начала координат.

8. Определить структурный тип для представления анкеты жителя, состоящей из его фамилии, названия города, где он проживает, и городского адреса. Адрес состоит из полей: «улица», «дом», «квартира». Ввести информацию по 100 жителям. Вывести фамилии жителей, которые живут в Ростове-на- Дону на улице Ленина.

9. Определить структурный тип для представления анкеты студента, состоящей из его фамилии, дня рождения и пола. «День рождения» состоит из полей: «число», «месяц», «год». Ввести информацию по 25 студентам из группы. Вывести все фамилии мальчиков, родившихся в мае 1986 года.

10. Определить структурный тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести величину среднего радиуса всех окружностей. Далее вывести координаты центра окружностей чей радиус выше среднего.

11. Определить структурный тип для представления анкеты жителя, состоящей из его фамилии, названия города, где он проживает, и городского адреса. Адрес состоит из полей: «улица», «дом», «квартира». Ввести информацию по 100 жителям. Вывести фамилии жителей, которые живут в Воронеже наулице Лизюкова.

12. Определить структурный тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести радиус окружности, чей центр самый удаленный от оси OY(оси ординат).

13. Определить структурный тип для представления студенческой ведомости состоящей из 2х полей: Ф. И. О., и оценка. В свою очередь поле оценка состоит из 4 элементов: оценка за математику, оценка за физику, оценка за информатику и средний балл. Составить программу, позволяющую вводить студенческую ведомость (без среднего балла). Далее найти для каждого студента средний балл. Вывести фамилии отличников по математике.

14. Определить структурный тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести радиус окружности, чей центр расположен ближе всего к оси OY(оси ординат).

15. Определить структурный тип для представления анкеты студента, состоящей из его фамилии, дня рождения и пола. «День рождения» состоит из полей: «число», «месяц», «год». Ввести информацию по 25 студентам из группы. Вывести все фамилии девочек, родившихся в марте 1991 года.

16. Определить структурный тип для представления информации о кости домино, состоящей из левой половинки и правой половинки. Поля
«левая» и «правая» половинки хранят информацию о количестве точек на половинках. Описать массив из 28 элементов (костей домино). Заполнить массив случайными числами или ввести с клавиатуры. Определить, правильно ли выставлены кости в данном массиве (Равна ли правая цифра очередной кости левой цифре следующей кости).

Задача №3

1. Определить комбинированный (структурный) тип для представления анкеты жителя, состоящей из его фамилии, названия города, где он проживает, и городского адреса. Адрес состоит из полей: «улица», «дом», «квартира». Ввести информацию по 10 жителям. Переписать из исходного массива в другой массив, информацию только о тех жителях, которые живут в Москве.

2. Определить комбинированный (структурный) тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Переписать из исходного массива в другой массив, информацию только о тех окружностях, центр которых лежит в 1-ой четверти.

3. Определить комбинированный (структурный) тип для представления анкеты студента, состоящей из его фамилии, дня рождения и пола. «День рождения» состоит из полей: «число», «месяц», «год». Ввести информацию по 25 студентам из группы. Переписать из исходного массива в другой массив, информацию только о тех студентах, которые родились в указанном году. (указанную год вводит пользователь клавиатуры)

4. Определить комбинированный (структурный) тип для представления анкеты жителя, состоящей из его фамилии, названия города, где он проживает, и городского адреса. Адрес состоит из полей: «улица», «дом», «квартира». Ввести информацию по 10 жителям. Переписать из исходного массива в другой массив, информацию только о тех жителях, фамилия которых начинается на указанную букву (указанную букву вводит пользователь клавиатуры)

5. Определить комбинированный (структурный) тип для представления студенческой ведомости состоящей из 2х полей: Ф. И. О., и оценка. В свою очередь полеоценка состоит из 4 элементов: оценка за математику, оценка за физику, оценка за информатику и средний балл. Составить программу, позволяющую вводить студенческую ведомость (без среднего балла). Переписать из исходного массива в другой массив, информацию только о студентах-незадолжниках (у которых нет ни одной двойки),

6. Определить комбинированный (структурный) тип, описывающий

окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Переписать из исходного массива в другой массив, информацию только о тех окружностях, радиус которых больше 1.

7. Определить комбинированный (структурный) тип для представления анкеты студента, состоящей из его фамилии, дня рождения и пола. «День рождения» состоит из полей: «число», «месяц», «год». Ввести информацию по 25 студентам из группы. Переписать из исходного массива в другой массив, информацию только о тех студентах, фамилия которых начинается на указанную букву (указанную букву вводит пользователь клавиатуры)

8. Определить комбинированный (структурный) тип для представления анкеты жителя, состоящей из его фамилии, названия города, где он проживает, и городского адреса. Адрес состоит из полей: «улица», «дом», «квартира». Ввести информацию по 100 жителям. Переписать из исходного массива в другой массив, информацию только о тех жителях, которые живут в Ростове-на-Дону на улице Ленина.

9. Определить комбинированный (структурный) тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Переписать из исходного массива в другой массив, информацию только о тех окружностях, центр которых лежит в 3-ой четверти.

10. Определить комбинированный (структурный) тип для представления анкеты студента, состоящей из его фамилии, дня рождения и пола. «День рождения» состоит из полей: «число», «месяц», «год». Ввести информацию по 25 студентам из группы. Переписать из исходного массива в другой массив, информацию только о тех студентах, которые родились в мае 1986 года и являются мальчиками .

11. Определить комбинированный (структурный) тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести величину среднего радиуса всех окружностей

. Переписать из исходного массива в другой массив, информацию только о тех окружностях, радиус которых меньше 1.

12. Определить комбинированный (структурный) тип для представления анкеты жителя, состоящей из его фамилии, названия города, где он проживает, и городского адреса. Адрес состоит из полей: «улица», «дом», «квартира». Ввести информацию по 100 жителям . Переписать из

исходного массива в другой массив, информацию только о тех жителях, которые живут в Воронеже на улице Лизюкова.

13. Определить комбинированный (структурный) тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. . Переписать из исходного массива в другой массив, только информацию о тех окружностях, центр которых лежит выше оси OX.

14. Определить комбинированный (структурный) тип для представления анкеты студента, состоящей из его фамилии, дня рождения и пола. «День рождения» состоит из полей: «число», «месяц», «год». Ввести информацию по 25 студентам из группы. Переписать из исходного массива в другой массив, информацию только о тех студентах, которые родились в марте 1991 года и являются девочками

15. Определить комбинированный (структурный) тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Переписать из исходного массива в другой массив, информацию только о тех окружностях, центр которых лежит ниже оси OX.

Задача №4

1) Определить структурный тип, описывающий гостиничный номер (название гостиницы, номер, комфортность (люкс, полулюкс стандарт, эконом), количество человек, стоимость). Заполнить структурный массив 10-ю записями. Переписать из исходного массива в другой массив, информацию только о тех гостиничных номерах, название гостиницы которых начинается с сочетания букв «City». Затем новый массив отсортировать по номеру. (рационально переставлять все поля структуры разом)

2) Определить структурный тип, описывающий расписание

полетов самолетов (пункт назначения, время отправления, время прибытия, время полета, стоимость билета). Заполнить структурный массив 10-ю записями. Переписать из исходного массива в другой массив, информацию только о тех рейсах пункт назначения, которых содержит по 2 буквы «а». Затем новый массив отсортировать по пункту назначения по алфавиту. (рационально переставлять все поля структуры разом)

3) Определить структурный тип, описывающий гостиничный номер (название гостиницы, номер, комфортность (люкс, полулюкс стандарт, эконом), количество человек, стоимость). Заполнить структурный массив 10-ю записями. Переписать из исходного массива в другой массив, информацию только о тех гостиничных номерах, в название гостиницы которых есть по 2 буквы «а». Затем новый массив отсортировать по названию гостиницы по алфавиту. (рационально переставлять все поля структуры разом)

4) Определить структурный тип, описывающий музыкальные CD-диски (название альбома, исполнитель, стиль, год выпуска, длительность, стоимость). Заполнить структурный массив 10-ю записями. Переписать из исходного массива в другой массив, информацию только о тех CD-дисках, название альбома которых начинается на сочетание букв (3 - 4) введенных пользователем. Затем новый массив отсортировать по стилю по алфавиту. (рационально переставлять все поля структуры разом)

5) Определить структурный тип, описывающий книги домашней библиотеки (автор, название книги, издательство, год издания, стоимость). Заполнить структурный массив 10-ю записями. Переписать из исходного массива в другой массив, информацию только о тех книгах, в название которых есть по 3 буквы «о». Затем вывести информацию, отсортированную по названию издательства по алфавиту. (рационально переставлять все поля структуры разом)

6) Определить структурный тип, описывающий книги домашней библиотеки (автор, название книги, издательство, год издания,

стоимость). Заполнить структурный массив 10-ю записями. Переписать из исходного массива в другой массив, информацию только о тех книгах, в название издательства которых есть 1 буква «к». Затем новый массив отсортировать по названию книги по алфавиту. (рационально переставлять все поля структуры разом)

7) Определить структурный тип, описывающий гостиничный номер (название гостиницы, номер, комфортность (люкс, полулюкс стандарт, эконом), количество человек, стоимость). Заполнить структурный массив 10-ю записями. Переписать из исходного массива в другой массив, информацию только о тех гостиничных номерах, название гостиницы которых начинается на букву «Р». Затем новый массив отсортировать по возрастанию стоимости. (рационально переставлять все поля структуры разом)

8) Определить структурный тип, описывающий книги домашней библиотеки (автор, название книги, издательство, год издания, стоимость). Заполнить структурный массив 10-ю записями. Переписать из исходного массива в другой массив,только информацию только о тех книгах, название которых начинается на «Фент». Затем новый массив отсортировать по стоимости. (рационально переставлять все поля структуры разом)

9) Определить структурный тип, описывающий гостиничный номер (название гостиницы, номер, комфортность (люкс, полулюкс стандарт, эконом), количество человек, стоимость). Заполнить структурный массив 10-ю записями. Переписать из исходного массива в другой массив, информацию только о тех гостиничных номерах, название гостиницы которых оканчивается на сочетание букв
«plaza». Затем новый массив отсортировать по возрастанию стоимости. (рационально переставлять все поля структуры разом)

10) Определить структурный тип, описывающий музыкальные CD-диски (название альбома, исполнитель, стиль, год выпуска, длительность, стоимость). Заполнить структурный массив 10-ю записями. Переписать из

исходного массива в другой массив, информацию только о тех CD-дисках, название стиля которых начинается на сочетание букв (3 - 4) введенных пользователем. Затем новый массив отсортировать по исполнителю по алфавиту. (рационально переставлять все поля структуры разом)

11) Определить структурный тип, описывающий гостиничный номер (название гостиницы, номер, Комфортность (люкс, полулюкс стандарт, эконом), количество человек, стоимость). Заполнить структурный массив 10-ю записями. Переписать из исходного массива в другой массив, информацию только о тех гостиничных номерах, название гостиницы которых оканчивается на сочетание букв «hostel». Затем новый массив отсортировать по комфортности по алфавиту. (рационально переставлять все поля структуры разом)

12) Определить структурный тип, описывающий книги домашней библиотеки (автор, название книги, издательство, год издания, стоимость). Заполнить структурный массив 10-ю записями. Переписать из исходного массива в другой массив, только информацию только о тех книгах, название издательства которых начинается на «Ф». Затем новый массив отсортировать по названию книги по алфавиту. (рационально переставлять все поля структуры разом)

13) Определить структурный тип, описывающий музыкальные CD-диски (название альбома, исполнитель, стиль, год выпуска, длительность, стоимость). Заполнить структурный массив 10-ю записями. Переписать из исходного массива в другой массив, информацию только о тех CD-дисках исполнитель, которых начинается на букву «Б». Затем новый массив отсортировать по стоимости. (рационально переставлять все поля структуры разом)

14) Определить структурный тип, описывающий студенческую ведомость (Ф. И. О., оценки за три экзамена, средний балл). Заполнить структурный массив 10-ю записями. Переписать из исходного массива в

другой массив, информацию только о тех студентах фамилии, которых оканчиваются на «ова». Затем новый массив отсортировать по среднему баллу. (рационально переставлять все поля структуры разом)

15) Определить структурный тип, описывающий расписание полетов самолетов (пункт посадки, время отправления, время прибытия, время полета, стоимость билета). Заполнить структурный массив 10-ю записями. Переписать из исходного массива в другой массив, информацию только о тех рейсах пункт назначения, которых оканчивается сочетанием «град». Затем новый массив отсортировать по времени полета. (рационально переставлять все поля структуры разом)

Задача №5

1) Определить комбинированный (структурный) тип для представления информации по горным вершинам, состоящей из названия вершины и ее высоты. Ввестиинформацию по 20 вершинам. Вывести среднее значение высот всех 20 вершин. Затем вывести информацию, отсортированную по возрастанию высоты вершины. (рационально переставлять все поля структуры разом)

2) Определить комбинированный (структурный) тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести радиус самой большой окружности. Затем вывести информацию, отсортированную по возрастанию радиуса окружности. (рационально переставлять все поля структуры разом)

3) Определить комбинированный (структурный) тип для представления информации по горным вершинам, состоящей из названия вершины и ее высоты. Ввести информацию по 10 вершинам. Вывести название самой высокой вершины из всех 10. Затем вывести информацию,

отсортированную по возрастанию высоты вершины. (рационально переставлять все поля структуры разом)

4) Определить комбинированный (структурный) тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести радиус окружности, чей центр расположен ближе всего к оси OX(оси абсцисс). Затем вывести информацию, отсортированную по возрастанию радиуса окружности. (рационально переставлять все поля структуры разом).

5) Определить комбинированный (структурный) тип для представления информации по горным вершинам, состоящей из названия вершины и ее высоты. Ввести информацию по 50 вершинам. Вывести название самой низкой вершины из всех 50. Затем вывести информацию, отсортированную по возрастанию высоты вершины. (рационально переставлять все поля структуры разом)

6) Определить комбинированный (структурный) тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести сумму радиусов всех окружностей. Затем вывести информацию, отсортированную по возрастанию радиуса окружности. (рационально переставлять все поля структуры разом)

7) Определить комбинированный (структурный) тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести радиус окружности, чей центр самый удаленный от оси OY(оси ординат). Затем вывести информацию, отсортированную по возрастанию радиуса окружности. (рационально переставлять все поля структуры разом).

8) Определить комбинированный (структурный) тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести радиус окружности, чей центр расположен ближе всего к оси OY(оси ординат). Затем вывести информацию, отсортированную по возрастанию радиуса окружности. (рационально переставлять все поля структуры разом).

9) Определить комбинированный (структурный) тип для представления анкеты ребенка, состоящей из его имени, пола и роста. Ввести информацию по 20 детям. Вывести средний рост девочек . Затем вывести информацию, отсортированную по имени по алфавиту. (рационально переставлять все поля структуры разом)

10) Определить комбинированный (структурный) тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести радиус самой маленькой окружности. Затем вывести информацию, отсортированную по возрастанию радиуса окружности. (рационально переставлять все поля структуры разом)

11) Определить комбинированный (структурный) тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести радиус окружности, чей центр самый удаленный от начала координат. Затем вывести информацию, отсортированную по возрастанию радиуса окружности. (рационально переставлять все поля структуры разом).

12) Определить комбинированный (структурный) тип для представления анкеты ребенка, состоящей из его имени, пола и роста. Ввести информацию по 20 детям. Вывести имя самого высокого мальчика. Затем

вывести информацию, отсортированную по имени по алфавиту. (рационально переставлять все поля структуры разом)

13) Определить комбинированный (структурный) тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести величину среднего радиуса всех окружностей. Затем вывести информацию, отсортированную по возрастанию радиуса окружности. (рационально переставлять все поля структуры разом)

14) Определить комбинированный (структурный) тип, описывающий окружность и состоящий из двух полей: «радиус» и «центр». Поле «центр» в свою очередь состоит еще из двух полей: «координата X» и «координата Y». Ввести информацию по 10 окружностям. Вывести радиус окружности, чей центр самый удаленный от оси OX(оси абсцисс). (рационально переставлять все поля структуры разом)

Контрольные вопросы

Что такое структура в C++ и для каких целей она используется?

1. Чем структура отличается от массива?
2. Как объявить структуру? Приведите синтаксис.
3. Как объявить переменную структурного типа?
4. Как получить доступ к полям структуры? Какой оператор для этого используется?
5. Как выполнить инициализацию структурной переменной при объявлении?
6. Можно ли передавать структуру в функцию? Какими способами?
7. Как объявить массив структур?
8. Что такое вложенные структуры? Приведите пример.
9. Как обратиться к полю вложенной структуры?
10. Что такое typedef и для чего он используется?
11. Как с помощью typedef создать псевдоним для структуры?
12. В чём отличие между именем структурного шаблона и переменной структуры?

13. Можно ли присваивать одну структурную переменную другой? Что при этом происходит?
14. Как определить размер структуры в памяти? От чего он зависит?
15. Что такое перечисление (enum)? Как его объявить и использовать?
16. Какие значения по умолчанию получают элементы перечисления?
17. Чем отличается enum от enum class?
18. Как можно использовать перечисление в качестве поля структуры?
19. Как выполнить сортировку массива структур по одному из полей?

Требования к отчёту

Титульный лист с названием работы, ФИО студента, номером группы, вариантом

Цель работы (формулируется самостоятельно на основе предложенной)

Теоретическая часть (основные понятия: структуры, поля, вложенные структуры, массивы структур, перечисления, typedef)

Постановка задачи (текст задания согласно варианту из выбранной задачи)

Листинг программы с комментариями:

Объявление структуры (структур)

Ввод данных (с клавиатуры или инициализация)

Обработка данных (поиск, подсчёт, фильтрация, сортировка)

Вывод результатов

Результаты работы (скриншоты вывода программы для разных тестовых данных)

Ответы на контрольные вопросы (письменно, развёрнуто)

Выводы (что нового узнали, какие навыки приобрели, какие были трудности)

Ссылка на Git-репозиторий с кодом программы

Критерии оценивания

Базовый уровень (задачи №1-2) — максимум 60 баллов

Критерий	Баллы
Структура объявлена корректно, поля описаны верно	10
Данные введены (массив структур заполнен)	10
Обработка данных выполнена в соответствии с заданием	20
Результат выведен на экран	10
Код отформатирован, есть комментарии, есть ответы на вопросы	10

Средний уровень (задачи №3-4) — максимум 80 баллов

Критерий	Баллы
Все требования базового уровня	40
Использование вложенных структур	15
Фильтрация данных (перезапись в другой массив)	15
Сортировка массива структур по указанному полю	10

Продвинутый уровень (задача №5) — максимум 100 баллов

Критерий	Баллы
Все требования среднего уровня	60
Вычисление статистических характеристик (среднее, минимум, максимум)	10
Сортировка массива структур по полю	10
Обработка ошибок ввода	5
Использование перечислений (enum) для статусов или типов	10
Наличие ссылки на Git-репозиторий	5

Штрафные баллы

Нарушение	Штраф
Имена переменных не отражают их смысл	-5
Отсутствуют комментарии в коде	-5
Нет проверки на корректность ввода данных	-5
Нет ответов на контрольные вопросы	-10
Нет ссылки на Git-репозиторий	-10

Нарушение	Штраф
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка (традиционная)	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №4

Перечисления (enum, enum class). Множества (set, битовые маски)

1. Цель работы:
2. Научиться объявлять и использовать перечисления (enum и enum class) для ограничения набора допустимых значений
3. Освоить битовые маски для хранения множества флагов с использованием битовых операций
4. Получить навыки работы с перечислениями в структурах и классах
5. Научиться преобразовывать перечисления в строки и обратно
6. Освоить базовые операции с std::set из стандартной библиотеки

Теоретическое обоснование

1. Перечислимые типы (enum)

Перечисление (enum) - это тип данных, определяющий набор именованных констант (перечислителей). Перечисления используются, когда переменная может принимать только ограниченный набор значений.

Синтаксис объявления перечисления:

```
enum ИмяТипа {
    ЗНАЧЕНИЕ1,
    ЗНАЧЕНИЕ2,
    ЗНАЧЕНИЕ3
};
```

Пример: дни недели

```
enum DayOfWeek {
    Monday,    // 0
    Tuesday,   // 1
    Wednesday, // 2
    Thursday,  // 3
    Friday,    // 4
    Saturday,  // 5
    Sunday     // 6
};
```

По умолчанию первый элемент получает значение 0, каждый следующий увеличивается на 1.

Явное задание значений

```
enum Status {
    Success = 1,
    Warning = 5,
    Error = 10
```

```
};

enum Colors {
    Red = 0xFF0000,
    Green = 0x00FF00,
    Blue = 0x0000FF
};
```

Использование перечисления

```
DayOfWeek today = Wednesday;

if (today == Wednesday) {
    cout << "Сегодня среда" << endl;
}

// Перечисление в switch
switch (today) {
    case Monday:
        cout << "Понедельник - тяжёлый день" << endl;
        break;
    case Friday:
        cout << "Пятница - почти выходной" << endl;
        break;
    case Saturday:
    case Sunday:
        cout << "Выходной!" << endl;
        break;
    default:
        cout << "Будний день" << endl;
```

```
}
```

Преобразование enum в int и обратно

```
// enum → int
int dayNumber = Wednesday;        // неявно, результат 2
int dayNumber2 = static_cast<int>(Wednesday); // явно

// int → enum (требуется явное приведение)
DayOfWeek day = static_cast<DayOfWeek>(2);
```

2. Типизированные перечисления (enum class) - C++11

enum class - более безопасная версия перечислений. В отличие от обычного enum:

Перечислители не выходят во внешнюю область видимости

Неявное преобразование в int запрещено

Можно задавать базовый тип

Синтаксис:

```
enum class ИмяТипа : базовый_тип {
    ЗНАЧЕНИЕ1,
    ЗНАЧЕНИЕ2,
    ЗНАЧЕНИЕ3
};
```

Пример**:**

```
enum class Color {
    Red,
    Green,
    Blue
```

```
};

enum class TrafficLight : char {  // с указанием базового типа
    Red = 'R',
    Yellow = 'Y',
    Green = 'G'
};

int main() {
    Color c = Color::Red;  // обязательно через Color::

    // int x = c;          // ОШИБКА! нет неявного преобразования
    int x = static_cast<int>(c);  // OK, явное приведение

    if (c == Color::Red) {  // OK, сравнение в рамках одного enum
        cout << "Красный" << endl;
    }
}
```

3. Преобразование перечисления в строку

Для вывода названий перечислителей необходимо создать функцию преобразования:

```
enum class Season {
    Spring, Summer, Autumn, Winter
};

string seasonToString(Season s) {
    switch (s) {
```

```
        case Season::Spring: return "Весна";
        case Season::Summer: return "Лето";
        case Season::Autumn: return "Осень";
        case Season::Winter: return "Зима";
        default: return "Неизвестно";
    }
}

Season stringToSeason(const string& str) {
    if (str == "Весна" || str == "весна") return Season::Spring;
    if (str == "Лето" || str == "лето") return Season::Summer;
    if (str == "Осень" || str == "осень") return Season::Autumn;
    if (str == "Зима" || str == "зима") return Season::Winter;
    throw invalid_argument("Неизвестное время года");
}
```

4. Битовые маски

Битовая маска - это способ хранения множества флагов в одной целочисленной переменной, где каждый бит отвечает за определённый признак.

Определение флагов

```
// Степени двойки - каждый флаг занимает свой бит
enum Permissions {
    Read    = 1 << 0,  // 1  (0b0001)
    Write   = 1 << 1,  // 2  (0b0010)
    Execute = 1 << 2,  // 4  (0b0100)
    Delete  = 1 << 3   // 8  (0b1000)
};
```

Битовые операции

Оператор	Название	Пример	Результат
|	Побитовое ИЛИ	a | b	Установить флаги
&	Побитовое И	a & b	Проверить флаги
^	Побитовое XOR	a ^ b	Переключить флаги
~	Побитовое НЕ	~ a	Инвертировать
<<	Сдвиг влево	1 << n	Установить n-й бит
>>	Сдвиг вправо	x >> n	Сбросить младшие биты

Работа с битовыми масками

```
int permissions = 0;  // пустая маска

// Установка флагов (побитовое ИЛИ)
permissions |= Read;      // добавить Read
permissions |= Write;     // добавить Write
// permissions = 0b0011 = 3
```

```
// Проверка флага (побитовое И)
if (permissions & Read) {
    cout << "Разрешение на чтение есть" << endl;
}

// Проверка нескольких флагов
if ((permissions & (Read | Write)) == (Read | Write)) {
    cout << "Есть оба разрешения" << endl;
}

// Снятие флага (побитовое И с инвертированной маской)
permissions &= ~Write;    // убрать Write

// Переключение флага (XOR)
permissions ^= Execute    // если был - убрать, не было - добавить
```

5. Полный пример: права доступа к файлу

```
#include <iostream>
#include <string>

using namespace std;

// Флаги прав доступа (битовая маска)
enum FilePermissions {
    READ    = 1 << 0,   // 1
    WRITE   = 1 << 1,   // 2
    EXECUTE = 1 << 2    // 4
};
```

```
// Тип файла (enum class)
enum class FileType {
    Text,
    Image,
    Executable,
    Unknown
};

// Преобразование типа файла в строку
string fileTypeToString(FileType type) {
    switch (type) {
        case FileType::Text:       return "Текстовый";
        case FileType::Image:      return "Изображение";
        case FileType::Executable: return "Исполняемый";
        default:                   return "Неизвестный";
    }
}

// Вывод прав доступа в читаемом виде
void printPermissions(int perms) {
    cout << "Права: ";
    cout << ((perms & READ)    ? "r" : "-");
    cout << ((perms & WRITE)   ? "w" : "-");
    cout << ((perms & EXECUTE) ? "x" : "-");
    cout << endl;
}

// Структура файла с использованием enum
struct File {
```

```
    string name;
    FileType type;
    int permissions;  // битовая маска
};

int main() {
    setlocale(LC_ALL, "ru");

    // Создаём файлы
    File files[] = {
        {"document.txt", FileType::Text, READ | WRITE},
        {"photo.jpg", FileType::Image, READ},
        {"program.exe", FileType::Executable, READ | EXECUTE},
        {"secret.bin", FileType::Unknown, READ | WRITE | EXECUTE}
    };

    // Выводим информацию
    for (const auto& f : files) {
        cout << "Файл: " << f.name << endl;
        cout << "Тип: " << fileTypeToString(f.type) << endl;
        printPermissions(f.permissions);
        cout << endl;
    }

    // Демонстрация битовых операций
    int perms = READ | WRITE;
    cout << "Начальные права: ";
    printPermissions(perms);
```

```
    // Добавляем Execute
    perms |= EXECUTE;
    cout << "После добавления Execute: ";
    printPermissions(perms);

    // Убираем Write
    perms &= ~WRITE;
    cout << "После удаления Write: ";
    printPermissions(perms);

    // Проверка наличия Read
    if (perms & READ) {
        cout << "Разрешение на чтение присутствует" << endl;
    }

    return 0;
}
```

6. Множества (std::set) - базовое знакомство

std::set - контейнер, хранящий уникальные элементы в отсортированном порядке.

```
#include <set>

set<string> colors;
colors.insert("красный");
colors.insert("синий");
colors.insert("зелёный");
colors.insert("красный");  // не добавится (уже есть)
```

```
// Проверка наличия элемента
if (colors.find("синий") != colors.end()) {
    cout << "Синий есть в множестве" << endl;
}

// Перебор элементов
for (const string& color : colors) {
    cout << color << endl;
}
```

Примеры выполнения практической части

Пример 1. Перечисление «Время года» (базовый уровень)

Задача: Создать перечисление Season (весна, лето, осень, зима). По номеру месяца определить время года и вывести его название.

```
#include <iostream>
#include <string>

using namespace std;

enum class Season {
    Spring,
    Summer,
    Autumn,
    Winter
};

string seasonToString(Season s) {
```

```
    switch (s) {
        case Season::Spring: return "Весна";
        case Season::Summer: return "Лето";
        case Season::Autumn: return "Осень";
        case Season::Winter: return "Зима";
        default: return "Неизвестно";
    }
}

Season getSeason(int month) {
    if (month >= 3 && month <= 5) return Season::Spring;
    if (month >= 6 && month <= 8) return Season::Summer;
    if (month >= 9 && month <= 11) return Season::Autumn;
    return Season::Winter;  // 12, 1, 2
}

int main() {
    setlocale(LC_ALL, "ru");

    int month;
    cout << "Введите номер месяца (1-12): ";
    cin >> month;

    if (month < 1 || month > 12) {
        cout << "Ошибка: неверный номер месяца!" << endl;
        return 1;
    }

    Season s = getSeason(month);
```

```
    cout << "Время года: " << seasonToString(s) << endl;

    return 0;
}
```

Ожидаемый вывод:

Введите номер месяца (1-12): 7

Время года: Лето

Пример 2. Битовые маски (средний уровень)

Задача**:** Реализовать хранение прав доступа (чтение, запись, выполнение) с помощью битовой маски. Реализовать функции установки, проверки и снятия прав.

```
#include <iostream>

using namespace std;

// Флаги прав (степени двойки)
enum AccessFlags {
    ACCESS_READ    = 1 << 0,  // 1
    ACCESS_WRITE   = 1 << 1,  // 2
    ACCESS_EXECUTE = 1 << 2   // 4
};

void printAccess(int access) {
    cout << "Права доступа: ";
    cout << ((access & ACCESS_READ)    ? "R" : "-");
    cout << ((access & ACCESS_WRITE)   ? "W" : "-");
```

```
    cout << ((access & ACCESS_EXECUTE) ? "X" : "-");
    cout << endl;
}

int main() {
    setlocale(LC_ALL, "ru");

    int access = 0;  // начальные права: нет никаких

    cout << "Начальные права: ";
    printAccess(access);

    // Добавляем чтение и запись
    access |= ACCESS_READ;
    access |= ACCESS_WRITE;
    cout << "После добавления чтения и записи: ";
    printAccess(access);

    // Проверяем, есть ли право на выполнение
    if (access & ACCESS_EXECUTE) {
        cout << "Есть право на выполнение" << endl;
    } else {
        cout << "Нет права на выполнение" << endl;
    }

    // Добавляем выполнение
    access |= ACCESS_EXECUTE;
    cout << "После добавления выполнения: ";
    printAccess(access);
```

```
    // Убираем запись
    access &= ~ACCESS_WRITE;
    cout << "После удаления записи: ";
    printAccess(access);

    return 0;
}
```

Ожидаемый вывод:

Начальные права: ---

После добавления чтения и записи: RW-

Нет права на выполнение

После добавления выполнения: RWX

После удаления записи: R-X

Пример 3. Перечисление и структура (продвинутый уровень)

Задача: Создать структуру Task (задача) с полями: название, приоритет (enum), статус (enum). Реализовать отображение задач по приоритету.

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

// Перечисление приоритетов
enum class Priority {
    Low,
    Medium,
```

```
    High,
    Critical
};

// Перечисление статусов
enum class Status {
    Pending,      // ожидает выполнения
    InProgress,   // в процессе
    Completed     // выполнена
};

// Структура задачи
struct Task {
    string title;
    Priority priority;
    Status status;
};

// Преобразование приоритета в строку
string priorityToString(Priority p) {
    switch (p) {
        case Priority::Low:      return "Низкий";
        case Priority::Medium:   return "Средний";
        case Priority::High:     return "Высокий";
        case Priority::Critical: return "Критический";
        default:                 return "Неизвестно";
    }
}
```

```
// Преобразование статуса в строку
string statusToString(Status s) {
    switch (s) {
        case Status::Pending:    return "Ожидает";
        case Status::InProgress: return "В процессе";
        case Status::Completed:  return "Выполнена";
        default:                 return "Неизвестно";
    }
}

// Вывод задач с заданным приоритетом
void printTasksByPriority(const vector<Task>& tasks, Priority p) {
    cout << "Задачи с приоритетом \"" << priorityToString(p) << "\":" <<
endl;
    for (const auto& task : tasks) {
        if (task.priority == p) {
            cout << "  - " << task.title << " [" << statusToString(task.status) <<
"]" << endl;
        }
    }
}

int main() {
    setlocale(LC_ALL, "ru");

    vector<Task> tasks = {
        {"Написать отчёт", Priority::High, Status::InProgress},
        {"Проверить почту", Priority::Low, Status::Completed},
```

```
        {"Исправить критическую ошибку", Priority::Critical, Status::Pending},
        {"Обновить документацию", Priority::Medium, Status::Pending}
    };

    // Вывод всех задач
    cout << "=== Все задачи ===" << endl;
    for (const auto& task : tasks) {
        cout << "Задача: " << task.title << endl;
        cout << "  Приоритет: " << priorityToString(task.priority) << endl;
        cout << "  Статус: " << statusToString(task.status) << endl;
    }

    cout << "\n=== Фильтрация по приоритету ===" << endl;
    printTasksByPriority(tasks, Priority::Critical);
    printTasksByPriority(tasks, Priority::High);

    return 0;
}
```

Ожидаемый вывод:

```
=== Все задачи ===
Задача: Написать отчёт
  Приоритет: Высокий
  Статус: В процессе
Задача: Проверить почту
  Приоритет: Низкий
  Статус: Выполнена
Задача: Исправить критическую ошибку
```

Приоритет: Критический

Статус: Ожидает

Задача: Обновить документацию

Приоритет: Средний

Статус: Ожидает

=== Фильтрация по приоритету ===

Задачи с приоритетом "Критический":

- Исправить критическую ошибку [Ожидает]

Задачи с приоритетом "Высокий":

- Написать отчёт [В процессе]

Индивидуальные задания

Базовый уровень (15 вариантов)

Общее задание: Создать перечисление согласно варианту. Написать программу, которая:

Запрашивает у пользователя значение (номер, код) для определения элемента перечисления

Выводит соответствующее значение

Использует перечисление в switch для выполнения различных действий

Вар.	**Имя перечисления**	**Значения**	**Задача**
1	Month	Январь, Февраль, ..., Декабрь	По номеру месяца вывести количество дней в месяце
2	Season	Весна, Лето,	По номеру

Вар.	**Имя перечисления**	**Значения**	**Задача**
		Осень, Зима	месяца определить и вывести время года
3	DayOfWeek	Пн, Вт, Ср, Чт, Пт, Сб, Вс	По номеру дня определить рабочий/выходной
4	Grade	Отлично, Хорошо, Удовлетворительно, Неудовлетворительно	По числовой оценке (5,4,3,2) вывести текстовую
5	Drink	Кофе, Чай, Сок, Вода, Лимонад	По названию напитка вывести его цену (задать в коде)
6	Transport	Автобус, Троллейбус, Трамвай, Метро	По виду транспорта вывести стоимость проезда
7	RoomType	Эконом, Стандарт, Полулюкс, Люкс	По типу номера вывести цену за сутки
8	Priority	Низкий, Средний, Высокий, Критический	По приоритету вывести срок выполнения (дни)
9	CPU	Intel, AMD, Apple	По бренду

Вар.	**Имя перечисления**	**Значения**	**Задача**
			вывести рекомендуемую ценовую категорию
10	Fuel	АИ-92, АИ-95, АИ-98, Дизель	По типу топлива вывести цену за литр
11	Payment	Наличные, Карта, Перевод	По способу оплаты вывести комиссию (%)
12	Language	Русский, Английский, Немецкий, Французский	По языку вывести приветствие на этом языке
13	Connection	WiFi, Ethernet, Bluetooth, USB	По типу соединения вывести максимальную скорость
14	Operating System	Windows, Linux, macOS, Android	По ОС вывести рекомендацию по минимальным требованиям
	GameGenr	Action, RPG,	По жанру

№ вар.	**Имя перечисления**	**Значения**	**Задача**
5	e	Strategy, Simulation	вывести примеры популярных игр

Средний уровень (15 вариантов)

Общее задание: Создать структуру, содержащую поле-перечисление. Заполнить массив структур (5-10 записей). Выполнить фильтрацию или подсчёт по значению перечисления. Использовать enum class.

№ вар.	**Имя структуры**	**Поля**	**Поле-перечисление**	**Задание**
1	Employee	ФИО, оклад	Department (Бухгалтерия, IT, Отдел_продаж, Администрация)	Вывести сотрудников из указанного отдела
2	Product	Название, цена	Category (Электроника, Одежда, Продукты, Книги)	Подсчитать общую стоимость товаров выбранной категории
3	Task	Описани	Priority (Низкий, Средний, Высокий,	Вывести все задачи с

№ вар.	Имя структуры	Поля	Поле-перечисление	Задание
		е, срок	Критический)	высоким и критическим приоритетом
4	Student	ФИО, группа	Grade (Отлично, Хорошо, Удовлетворительно, Неудовлетворительно)	Подсчитать количество отличников и хорошистов
5	Order	Номер, сумма	Status (Новый, В_обработке, Отправлен, Доставлен, Отменён)	Вывести все заказы со статусом «В обработке»
6	Room	Номер, этаж	Class (Эконом, Стандарт, Люкс)	Подсчитать количество номеров класса «Люкс»
7	Ticket	Пункт назначения, цена	Type (Детский, Взрослый, Студенческий, Пенсионный)	Вычислить общую стоимость билетов для взрослых

№ вар.	Имя структуры	Поля	Поле-перечисление	Задание
8	Lesson	Название, часы	Form (Лекция, Практика, Лабораторная, Экзамен)	Подсчитать количество лекций и практик
9	Payment	Плательщик, сумма	Method (Наличные, Карта, Перевод)	Вычислить общую сумму безналичных платежей
10	Document	Название, страниц	Format (TXT, DOC, PDF, Изображение)	Подсчитать количество PDF-документов
11	Game	Название, жанр	Platform (PC, PlayStation, Xbox, Nintendo)	Вывести игры для указанной платформы
12	Course	Название, часов	Level (Начальный, Средний, Продвинутый)	Подсчитать количество курсов начального

№ вар.	Имя структуры	Поля	Поле-перечисление	Задание
				уровня
13	Apartment	Адрес, комнат	Condition (Хорошее, Среднее, Требует_ремонта)	Вывести квартиры в хорошем состоянии
14	Vacancy	Должность, зарплата	Employment (Полная, Частичная, Удалённая, Стажировка)	Подсчитать количество вакансий с полной занятостью
15	Vehicle	Модель, год	Type (Автомобиль, Мотоцикл, Грузовик, Автобус)	Вывести все автомобили и мотоциклы

Продвинутый уровень (15 вариантов)

Общее задание: Используя перечисление и битовые операции, реализовать хранение и обработку флагов. Реализовать функции: установка флага, проверка флага, снятие флага, вывод всех установленных флагов.

вар.	Тема	Флаги (значения)	Задание
	Права доступа к файлу	READ, WRITE, EXECUTE	Реализовать проверку наличия всех прав для исполняемого файла
	Дни недели	Mon, Tue, Wed, Thu, Fri, Sat, Sun	Реализовать хранение рабочих дней. Проверить, является ли день рабочим
	Опции автомобиля	AC, HEATED_SEATS, ABS, ESP, NAVIGATION	Подсчитать количество выбранных опций, вывести их список
	Свойства окна	MODAL, RESIZABLE, SCROLLABLE, VISIBLE	Установить видимость и возможность изменения размера
	Уровни доступа	ADMIN, MODERATOR, USER, GUEST	Проверить, есть ли у пользователя права

вар.	Тема	Флаги (значения)	Задание
			администратора
	Языки программирования	CPP, PYTHON, JAVA, JS, C_SHARP	Проверить, знает ли программист C++ и Python
	Жанры фильма	COMEDY, DRAMA, HORROR, SCI_FI, ACTION	Проверить, является ли фильм комедией или боевиком
	Спортивные секции	FOOTBALL, BASKETBALL, SWIMMING, CHESS	Подсчитать количество секций, которые посещает ученик
	Дни доставки	MON, TUE, WED, THU, FRI, SAT	Реализовать добавление и удаление дней доставки
0	Типы подписки	NEWSLETTER, PROMO, ORDER_STATUS, RECOMMENDATIONS	Реализовать отписку от рассылки новостей

вар.	Тема	Флаги (значения)	Задание
1	Режимы работы приложения	DEBUG, RELEASE, TEST, BENCHMARK	Установить режим DEBUG и TEST, проверить их
2	Транспортные средства	CAR, BIKE, BUS, TRUCK, MOTO	Подсчитать количество транспортных средств, прошедших техосмотр
3	Уровни логирования	INFO, WARNING, ERROR, DEBUG	Установить уровень WARNING и ERROR, вывести все установленные
4	Комплектация компьютера	CPU, RAM, SSD, GPU, PSU	Проверить наличие процессора и видеокарты
5	Жанры книг	FICTION, DETECTIVE, SCIENCE, HISTORY, BIOGRAPHY	Добавить новые жанры, удалить один,

вар.	Тема	Флаги (значения)	Задание
			вывести итоговый список

Контрольные вопросы

1. Что такое перечисление (enum) и для чего оно используется?
2. В чём разница между enum и enum class?
3. Как задать явные значения для элементов перечисления?
4. Как преобразовать enum class в целое число?
5. Почему enum class считается более безопасным, чем обычный enum?
6. Как организовать преобразование перечисления в строку и обратно?
7. Что такое битовая маска? Для чего она применяется?
8. Какие битовые операции используются для работы с масками?
9. Как установить битовый флаг? Как проверить его наличие? Как снять?
10. Как создать флаги с использованием сдвига (1 << n)?
11. Почему для битовых масок используют степени двойки?
12. Что такое std::set и чем он отличается от битовой маски?
13. Как добавить элемент в std::set? Как проверить наличие элемента?
14. В каких случаях удобно использовать перечисления, а в каких - битовые маски?
15. Можно ли использовать перечисление в качестве поля структуры?

Требования к отчёту

Титульный лист с названием работы, ФИО студента, номером группы, вариантом

Цель работы (формулируется самостоятельно на основе предложенной)

Теоретическая часть (основные понятия: перечисления, битовые маски, битовые операции)

Постановка задачи (текст задания согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода программы для разных тестовых данных)

Ответы на контрольные вопросы (письменно, развёрнуто)

Выводы (что нового узнали, какие навыки приобрели)

Ссылка на Git-репозиторий с кодом программы

Критерии оценивания

Базовый уровень (максимум 60 баллов)

Критерий	Баллы
Перечисление объявлено корректно	15
Преобразование ввода пользователя в перечисление	15
Использование switch или преобразование в строку	15
Код отформатирован, есть комментарии	10
Есть ответы на контрольные вопросы	5

Средний уровень (максимум 80 баллов)

Критерий	Баллы
Все требования базового уровня	40
Создание структуры с полем-перечислением	10
Массив структур (5-10 записей)	10
Фильтрация или подсчёт по перечислению	15
Использование enum class	5

Продвинутый уровень (максимум 100 баллов)

Критерий	Баллы
Все требования среднего уровня	60
Реализация битовой маски (флаги через 1 << n)	15
Реализация функций установки, проверки, снятия флагов	10
Вывод установленных флагов в читаемом виде	10
Наличие ссылки на Git-репозиторий	5

Штрафные баллы

Нарушение	Штраф
Отсутствует преобразование ввода в перечисление	-10
Неправильное использование	-10

Нарушение	Штраф
битовых операций	
Отсутствуют комментарии в коде	-5
Нет ответов на контрольные вопросы	-10
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка (традиционная)	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа № 5

Создание простых классов. Методы-аксессоры (геттеры/сеттеры)

Цель работы: Научиться создавать простые классы, определять поля и методы, реализовывать конструкторы, использовать спецификаторы доступа

для инкапсуляции данных, создавать методы доступа (геттеры и сеттеры) для работы с закрытыми полями.

Теоретическое обоснование

1. Понятие класса и объекта

Класс - это пользовательский тип данных, объединяющий данные (поля) и функции для работы с этими данными (методы). Класс можно представить как чертёж, на основе которого создаются конкретные экземпляры - объекты.

Пример из реальной жизни:

- Класс «Студент» - это общее понятие (чертёж)
- Конкретный студент «Иван Петров» - это объект (экземпляр класса)

2. Синтаксис определения класса

```
class ClassName {
private:
    // закрытые поля (доступны только внутри класса)
    тип_данных поле1;
    тип_данных поле2;

public:
    // открытые методы (интерфейс класса)
    тип_данных метод1(параметры) {
        // реализация
    }

    тип_данных метод2(параметры);
};

// Определение метода вне класса
тип_данных ClassName::метод2(параметры) {
    // реализация
}
```

3. Спецификаторы доступа

Спецификатор	Доступ внутри класса	Доступ извне	Назначение

private	да	нет	Для полей и вспомогательных методов
public	да	да	Для интерфейсных методов

Принцип инкапсуляции: поля класса должны быть закрытыми (private), а доступ к ним осуществляется через открытые (public) методы.

4. Методы доступа (геттеры и сеттеры)

Геттер (getter) - метод для получения значения закрытого поля.

Сеттер (setter) - метод для установки значения закрытого поля (часто с проверкой корректности).

```
class Example {
private:
    int value;

public:
    // Геттер
    int getValue() const {
        return value;
    }

    // Сеттер с проверкой
    void setValue(int newValue) {
        if (newValue > 0) {  // проверка
            value = newValue;
        }
    }
};
```

Ключевое слово const после списка параметров означает, что метод не изменяет состояние объекта.

5. Конструкторы

Конструктор - специальный метод, который автоматически вызывается при создании объекта.

Особенности конструкторов:

- Имя совпадает с именем класса
- Нет возвращаемого типа
- Может быть перегружен (несколько конструкторов)
- Используется для инициализации полей

```
class Example {
private:
    int a;
    double b;

public:
    // Конструктор по умолчанию
    Example() : a(0), b(0.0) {}

    // Параметризованный конструктор
    Example(int valA, double valB) : a(valA), b(valB) {}

    // Конструктор копирования
    Example(const Example& other) : a(other.a), b(other.b) {}
};
```

Список инициализации: a(valA), b(valB) позволяет инициализировать поля до выполнения тела конструктора.

6. Деструктор

Деструктор - специальный метод, автоматически вызываемый при уничтожении объекта.

Особенности деструктора:

- Имя: ~ + имя класса
- Не имеет параметров
- Не возвращает значение
- Используется для освобождения ресурсов

```
class Example {
public:
    ~Example() {
        // очистка ресурсов
        std::cout << "Объект уничтожен" << std::endl;
    }
};
```

Пример выполнения задания

Задание 1. Создать класс Student для представления информации о студенте

Требования к классу:

- Поля: имя (std::string), фамилия (std::string), возраст (int), средний балл (double)
- Конструкторы: по умолчанию, с параметрами, копирования
- Деструктор (с выводом сообщения)
- Геттеры и сеттеры для всех полей (с проверками в сеттерах)
- Метод для вывода информации о студенте

Реализация

Для создания заголовочного файла в Обозревателе решений выберите папку Файлы заголовков (рисунок 1)

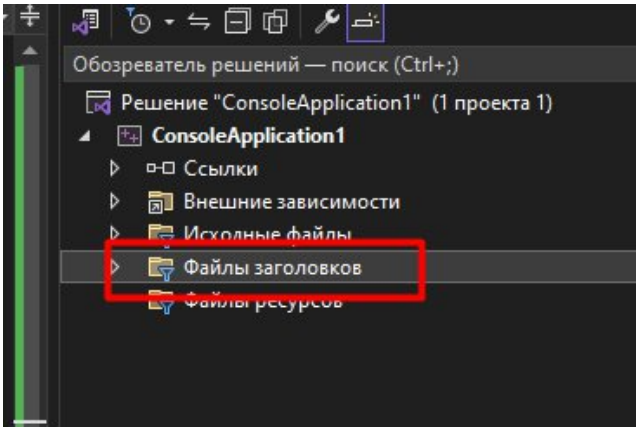


Рисунок 1

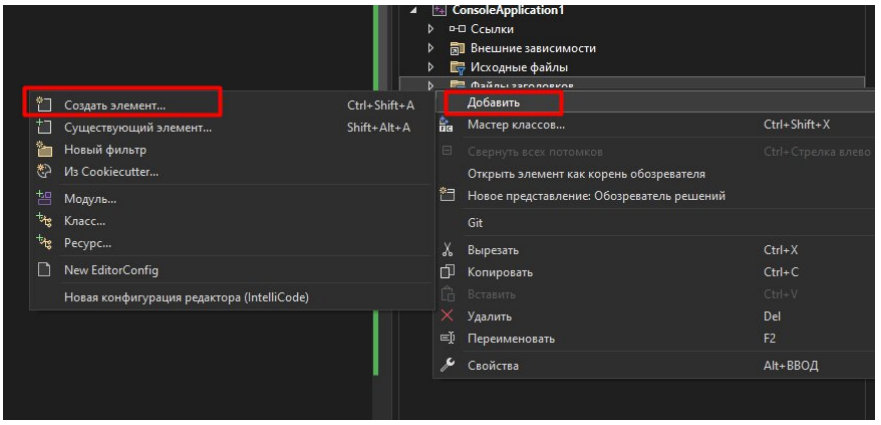


Рисунок 2

Далее выберите в контекстном меню, вызванное правой кнопкой мыши команду Добавить/Создать элемент (рисунок 2). Далее выберите Файл заголовка (рисунок 3)

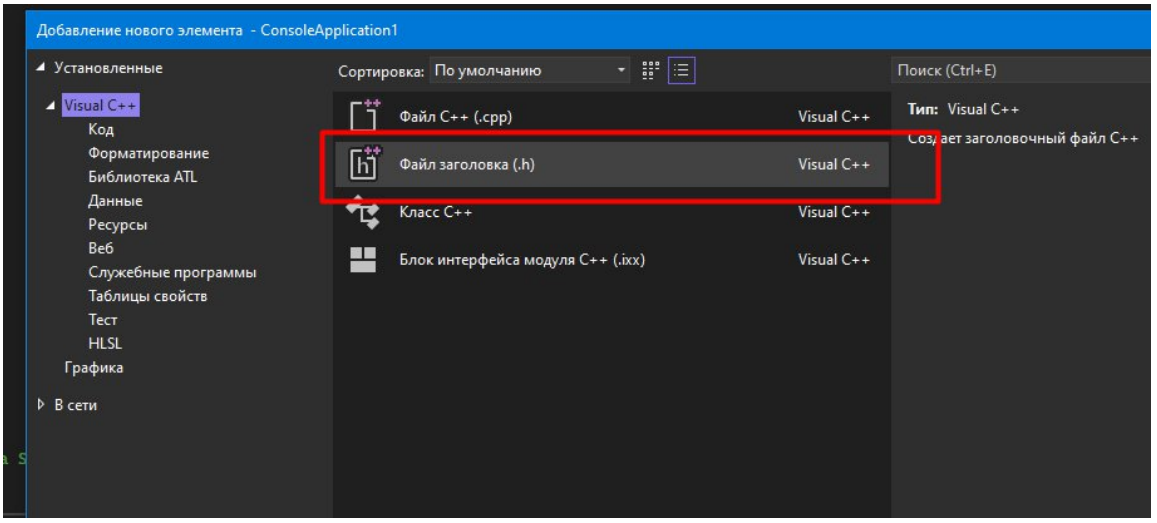


Рисунок 3

Файл Student.h - заголовочный файл

```
#pragma once
#include <string>
#include <iostream>

class Student {
private:
    std::string firstName;   // имя
    std::string lastName;    // фамилия
    int age;                 // возраст
    double averageGrade;     // средний балл

public:
    // Конструкторы
    Student();  // по умолчанию
    Student(const std::string& first, const std::string& last, int age, double grade);
```

```
    Student(const Student& other);  // копирования

    // Деструктор
    ~Student();

    // Геттеры (методы для получения значений)
    std::string getFirstName() const;
    std::string getLastName() const;
    int getAge() const;
    double getAverageGrade() const;

    // Сеттеры (методы для установки значений с проверкой)
    void setFirstName(const std::string& first);
    void setLastName(const std::string& last);
    void setAge(int age);
    void setAverageGrade(double grade);

    // Другие методы
    void printInfo() const;  // вывод информации
};
```

Файл Student.cpp - реализация методов

```
#include "Student.h"
#include <iostream>
#include <string>

// Конструктор по умолчанию
Student::Student()
    : firstName("Не указано"),
    lastName("Не указано"),
    age(0),
    averageGrade(0.0) {
    std::cout << "Вызван конструктор по умолчанию для студента" << std::endl;
}

// Параметризованный конструктор
Student::Student(const std::string& first, const std::string& last, int age, double grade)
    : firstName(first), lastName(last), age(age), averageGrade(grade) {
    std::cout << "Вызван параметризованный конструктор для студента "
        << firstName << " " << lastName << std::endl;
}

// Конструктор копирования
Student::Student(const Student& other)
    : firstName(other.firstName),
    lastName(other.lastName),
    age(other.age),
    averageGrade(other.averageGrade) {
    std::cout << "Вызван конструктор копирования для студента "
        << firstName << " " << lastName << std::endl;
}

// Деструктор
Student::~Student() {
    std::cout << "Вызван деструктор для студента "
        << firstName << " " << lastName << std::endl;
}
```

```cpp
// Геттеры
std::string Student::getFirstName() const {
    return firstName;
}

std::string Student::getLastName() const {
    return lastName;
}

int Student::getAge() const {
    return age;
}

double Student::getAverageGrade() const {
    return averageGrade;
}

// Сеттеры с проверками
void Student::setFirstName(const std::string& first) {
    if (!first.empty()) {
        firstName = first;
    }
    else {
        std::cout << "Ошибка: имя не может быть пустым!" << std::endl;
    }
}

void Student::setLastName(const std::string& last) {
    if (!last.empty()) {
        lastName = last;
    }
    else {
        std::cout << "Ошибка: фамилия не может быть пустой!" << std::endl;
    }
}

void Student::setAge(int age) {
    if (age >= 0 && age <= 120) {
        this->age = age;  // this-> используется для указания на поле класса
    }
    else {
        std::cout << "Ошибка: возраст должен быть от 0 до 120 лет!" << std::endl;
    }
}

void Student::setAverageGrade(double grade) {
    if (grade >= 0.0 && grade <= 5.0) {
        averageGrade = grade;
    }
    else {
        std::cout << "Ошибка: средний балл должен быть от 0 до 5!" << std::endl;
    }
}

// Вывод информации о студенте
void Student::printInfo() const {
    std::cout << "Информация о студенте:" << std::endl;
    std::cout << "  Имя: " << firstName << std::endl;
    std::cout << "  Фамилия: " << lastName << std::endl;
    std::cout << "  Возраст: " << age << " лет" << std::endl;
    std::cout << "  Средний балл: " << averageGrade << std::endl;
}
```

Файл main.cpp - демонстрационная программа

```cpp
#include <iostream>
#include <string>
#include "Student.h"

int main() {
    setlocale(LC_ALL, "ru");

    std::cout << "=== Демонстрация работы с классом Student ===\n" << std::endl;

    // Создание объекта с помощью конструктора по умолчанию
    std::cout << "--- Создание объекта 1 (конструктор по умолчанию) ---" << std::endl;
    Student student1;
    student1.printInfo();
    std::cout << std::endl;

    // Использование сеттеров для изменения данных
    std::cout << "--- Изменение данных через сеттеры ---" << std::endl;
    student1.setFirstName("Иван");
    student1.setLastName("Петров");
    student1.setAge(20);
    student1.setAverageGrade(4.5);
    student1.printInfo();
    std::cout << std::endl;

    // Создание объекта с помощью параметризованного конструктора
    std::cout << "--- Создание объекта 2 (параметризованный конструктор) ---" << std::endl;
    Student student2("Мария", "Сидорова", 19, 4.8);
    student2.printInfo();
    std::cout << std::endl;

    // Использование геттеров
    std::cout << "--- Использование геттеров для объекта 2 ---" << std::endl;
    std::cout << "Имя: " << student2.getFirstName() << std::endl;
    std::cout << "Фамилия: " << student2.getLastName() << std::endl;
    std::cout << "Возраст: " << student2.getAge() << std::endl;
    std::cout << "Средний балл: " << student2.getAverageGrade() << std::endl;
    std::cout << std::endl;

    // Демонстрация конструктора копирования
    std::cout << "--- Создание объекта 3 (конструктор копирования) ---" << std::endl;
    Student student3 = student2;  // или Student student3(student2);
    student3.printInfo();
    std::cout << std::endl;

    // Демонстрация проверки в сеттерах
    std::cout << "--- Проверка корректности данных в сеттерах ---" << std::endl;
    student1.setAge(150);  // некорректный возраст
    student1.setAverageGrade(6.0);  // некорректный балл
    student1.setFirstName("");  // пустое имя
    student1.printInfo();
    std::cout << std::endl;

    std::cout << "=== Завершение программы ===" << std::endl;
    return 0;
}
```

В результате должна получиться следующая структура проекта:

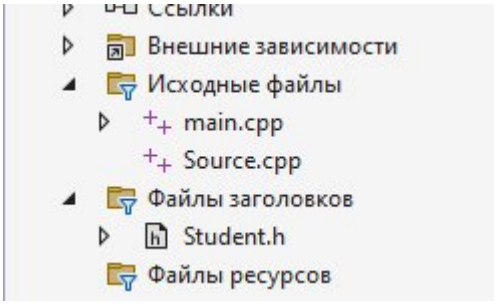


Рисунок 4

Пример 2. Класс Rectangle для работы с прямоугольниками

```
#pragma once

class Rectangle {
private:
    double width;
    double height;

public:
    // Конструкторы
    Rectangle();
    Rectangle(double w, double h);
    Rectangle(const Rectangle& other);

    // Деструктор
    ~Rectangle();

    // Геттеры
    double getWidth() const;
    double getHeight() const;

    // Сеттеры
    void setWidth(double w);
    void setHeight(double h);

    // Функции обработки
    double getArea() const;          // площадь
    double getPerimeter() const;     // периметр
    double getDiagonal() const;      // диагональ

    void printInfo() const;
};
```

Реализация методов:

```
#include "Rectangle.h"
#include <iostream>
#include <cmath>

Rectangle::Rectangle() : width(1.0), height(1.0) {
    std::cout << "Создан прямоугольник по умолчанию (1x1)" << std::endl;
```

```
}

Rectangle::Rectangle(double w, double h) {
    setWidth(w);
    setHeight(h);
    std::cout << "Создан прямоугольник " << width << "x" << height << std::endl;
}

Rectangle::Rectangle(const Rectangle& other)
    : width(other.width), height(other.height) {
    std::cout << "Создана копия прямоугольника" << std::endl;
}

Rectangle::~Rectangle() {
    std::cout << "Уничтожен прямоугольник " << width << "x" << height << std::endl;
}

double Rectangle::getWidth() const {
    return width;
}

double Rectangle::getHeight() const {
    return height;
}

void Rectangle::setWidth(double w) {
    if (w > 0) {
        width = w;
    }
    else {
        std::cout << "Ошибка: ширина должна быть положительной!" << std::endl;
    }
}

void Rectangle::setHeight(double h) {
    if (h > 0) {
        height = h;
    }
    else {
        std::cout << "Ошибка: высота должна быть положительной!" << std::endl;
    }
}

double Rectangle::getArea() const {
    return width  height;
}

double Rectangle::getPerimeter() const {
    return 2  (width + height);
}

double Rectangle::getDiagonal() const {
    return sqrt(width  width + height  height);
}

void Rectangle::printInfo() const {
    std::cout << "Прямоугольник " << width << " x " << height << std::endl;
    std::cout << "  Площадь: " << getArea() << std::endl;
    std::cout << "  Периметр: " << getPerimeter() << std::endl;
    std::cout << "  Диагональ: " << getDiagonal() << std::endl;
}
```

Реализацию функции main напишите самостоятельно подобно как в примере 1.

Задания для самостоятельного выполнения

Базовый уровень

Задание 1. Создать класс с указанными в таблице полями и методами:

- Конструктор по умолчанию (инициализирует поля нулевыми значениями)
- Параметризованный конструктор
- Конструктор копирования
- Деструктор (выводит сообщение об уничтожении объекта)
- Геттеры и сеттеры для всех полей (с проверкой корректности в сеттерах)
- Функцию формирования строки информации об объекте

В основной программе продемонстрировать работу всех методов класса.

Таблица 1. Варианты индивидуальных заданий (базовый уровень)

№ вар.	**Имя класса**	**Поле 1**	**Поле 2**	**Функция обработки**
1	Banknote	Номинал купюры (целое число)	Количество купюр	Вычислить общую сумму
2	Coin	Номинал монеты (целое число)	Количество монет	Вычислить общую сумму
3	Product	Цена товара	Количество	Вычислить

			единиц	общую стоимость
4	Food	Калорийность на 100г	Вес продукта (граммы)	Вычислить общую калорийность
5	Range	Левая граница	Правая граница	Вычислить длину диапазона
6	TimeInterval	Количество минут	Количество секунд	Вычислить общее количество секунд
7	TimeDuration	Количество часов	Количество минут	Вычислить общее количество минут
8	RightTriangle	Первый катет	Второй катет	Вычислить площадь
9	Movement	Скорость (м/сек)	Время движения (минуты)	Вычислить пройденное расстояние (метры)
10	Triangle	Первый катет	Второй катет	Вычислить гипотенузу
11	Trapezoid	Нижнее основание	Верхнее основание	Вычислить среднюю линию
12	Temperature	Температура в	Коэффициент	Перевести в

		Цельсия		Фаренгейты
13	Numbers	Число A	Число B	Вычислить полусумму чисел
14	NumbersProduct	Число A	Число B	Вычислить среднее геометрическое
15	Division	Делимое x	Делитель y	Вычислить целую часть от деления x на y

Средний уровень

Задание 2. Создать класс с полями, указанными в таблице 2. Реализовать в классе:

- Конструктор по умолчанию
- Параметризованный конструктор
- Конструктор копирования
- Деструктор (с сообщением об уничтожении объекта)
- Геттеры и сеттеры для всех полей
- Две функции обработки данных (1 и 2) из таблицы
- Функцию формирования строки информации об объекте

Создать проект для демонстрации работы: сформировать объекты с константными значениями и с введёнными с клавиатуры значениями. Выводить результаты на экран.

Таблица 2. Варианты индивидуальных заданий (средний уровень)

№ вар.	Имя класса	Поля класса	Функция-метод 1	Функция-метод 2
1	Date	день, месяц, год	Определить, является ли год високосным	Увеличить дату на 5 дней
2	Date2	день, месяц, год	Увеличить год на 1	Уменьшить дату на 2 дня
3	Date3	день, месяц, год	Определить, совпадают ли номер месяца и число дня	Увеличить дату на 1 месяц
4	Time	часы, минуты, секунды	Вычислить количество секунд	Увеличить время на 5 секунд
5	Time2	часы, минуты, секунды	Вычислить количество полных минут	Уменьшить время на 10 минут
6	Time3	часы, минуты, секунды	Определить количество минут до полуночи	Увеличить время на 100 минут
7	Book	название, автор, год издания	Вычислить, сколько лет книге	Определить, является ли книга современной (менее 10 лет)
8	Employee	фамилия, оклад, год поступления	Вычислить стаж работы	Определить, имеет ли работник стаж более 5 лет

9	Employee2	фамилия, оклад, год рождения	Вычислить возраст работника	Определить, исполнится ли работнику 50 лет в текущем году
10	Fraction	числитель, знаменатель	Выразить значение дроби в процентах	Сократить дробь
11	Complex	действительная часть, мнимая часть	Вычислить модуль числа	Вычислить аргумент числа в градусах
12	Complex2	действительная часть, мнимая часть	Умножить на заданное число	Вычислить комплексно-сопряжённое число
13	Book2	название, количество страниц, цена	Вычислить стоимость одной страницы	Увеличить цену на 20%, если страниц > 200
14	Product2	наименование, цена, год выпуска	Определить, сколько лет товару	Увеличить цену на 10%, если товар старше 5 лет
15	Vector2D	координата x, координата y	Вычислить длину вектора	Определить угол наклона к оси X

Содержание отчёта

1. Цель работы (формулируется студентом самостоятельно).

2. Теоретическая часть (краткое описание основных понятий: класс, объект, инкапсуляция, конструкторы, деструктор, геттеры/сеттеры).

3. Постановка задачи (согласно варианту).

4. Листинг программы с комментариями.

5. Результаты работы программы (скриншоты или копия экрана), ссылка на репозиторий git.

6. Выводы (что нового узнали, чему научились, какие трудности возникли).

Контрольные вопросы

1. Что такое класс? Чем класс отличается от объекта?

2. Для чего нужны спецификаторы доступа public и private?

3. В чём суть инкапсуляции?

4. Зачем нужны геттеры и сеттеры, если можно сделать поля открытыми?

5. Что такое конструктор? Какие виды конструкторов существуют?

6. Чем конструктор отличается от обычного метода?

7. Для чего нужен деструктор? Когда он вызывается?

8. Что такое список инициализации и зачем он нужен?

9. Что означает ключевое слово const после списка параметров метода?

10. Почему поля класса рекомендуется делать закрытыми?

Дополнительные задания (для повышенной оценки)

1. Добавьте в свой класс статическое поле, подсчитывающее количество созданных объектов.

2. Реализуйте перегрузку оператора << для вывода информации об объекте.

3. Создайте массив объектов вашего класса и продемонстрируйте работу с ним.

4. Добавьте метод, сохраняющий информацию об объекте в текстовый файл, и метод, загружающий данные из файла.

Лабораторная работа №6

Реализация класса «Динамический массив целых чисел» с конструкторами, деструктором и основными операциями. Правило трех

Цель работы: Закрепить теоретические знания о классах, конструкторах, деструкторах и инкапсуляции на практике, научиться реализовывать класс, управляющий динамической памятью, освоить «правило трёх» (деструктор, конструктор копирования, оператор присваивания), получить навыки работы с динамическими массивами в объектно-ориентированной парадигме, развить умение проектировать интерфейс класса, обеспечивающий безопасную работу с данными

Теоретическое обоснование

1. Динамический массив

Статический массив имеет фиксированный размер, известный на этапе компиляции:

```
int arr[100];  // размер известен заранее
```

Ограничения статического массива:

- Размер нельзя изменить во время выполнения программы
- Нельзя создать массив, размер которого зависит от пользовательского ввода
- Память выделяется на стеке, что ограничивает максимальный размер

Динамический массив решает эти проблемы:

- Память выделяется в куче (heap) с помощью оператора new[]
- Размер может определяться во время выполнения программы
- При необходимости массив можно расширять или сужать

2. Управление динамической памятью в классе

При реализации класса, владеющего динамической памятью, необходимо соблюдать «правило трёх»:

Метод	Назначение	Когда вызывается
Деструктор	Освобождает выделенную память	При уничтожении объекта
Конструктор копирования	Создаёт независимую копию объекта	При копировании объекта
Оператор присваивания	Корректно присваивает один объект другому	При присваивании obj1 = obj2

3. Почему нельзя использовать неявные версии?

```
class BadArray {
    int* data;
    int size;
public:
    BadArray(int n) : size(n), data(new int[n]) {}
    ~BadArray() { delete[] data; }
    // Нет конструктора копирования и оператора присваивания!
};
```

```
int main() {
    BadArray a(10);
    BadArray b = a;  // ПОВЕРХНОСТНОЕ КОПИРОВАНИЕ!
    // b.data и a.data указывают на одну память
    // При разрушении a и b -> ДВОЙНОЕ ОСВОБОЖДЕНИЕ!
}
```

4. Структура класса IntVector

```
class IntVector {
private:
    int* data;   // указатель на динамический массив
    int size;    // текущий размер массива
    int capacity; // вместимость (для будущего расширения)

public:
    // Конструкторы
    IntVector();                        // по умолчанию
    IntVector(int n);                   // размера n
    IntVector(int n, int value);        // размера n, заполненный value
    IntVector(const IntVector& other);  // копирования

    // Деструктор
    ~IntVector();

    // Операторы
    IntVector& operator=(const IntVector& other);  // присваивания
    int& operator[](int index);                    // доступа по индексу
    const int& operator[](int index) const;        // константный доступ

    // Методы доступа
    int getSize() const;
    int getCapacity() const;
    bool isEmpty() const;

    // Методы изменения
    void pushBack(int value);
    void popBack();
    void insert(int index, int value);
    void remove(int index);
    void clear();
    void resize(int newSize);

    // Поиск и вывод
    int find(int value) const;
    void print() const;
};
```

5. Список инициализации vs присваивание в теле конструктора

Плохой вариант (присваивание):

```
IntVector(int n) {
    size = n;
    data = new int[size];  // присваивание, не инициализация
}
```

Хороший вариант (список инициализации):

```
IntVector(int n) : size(n), data(new int[size]) {}
```

Преимущества списка инициализации:

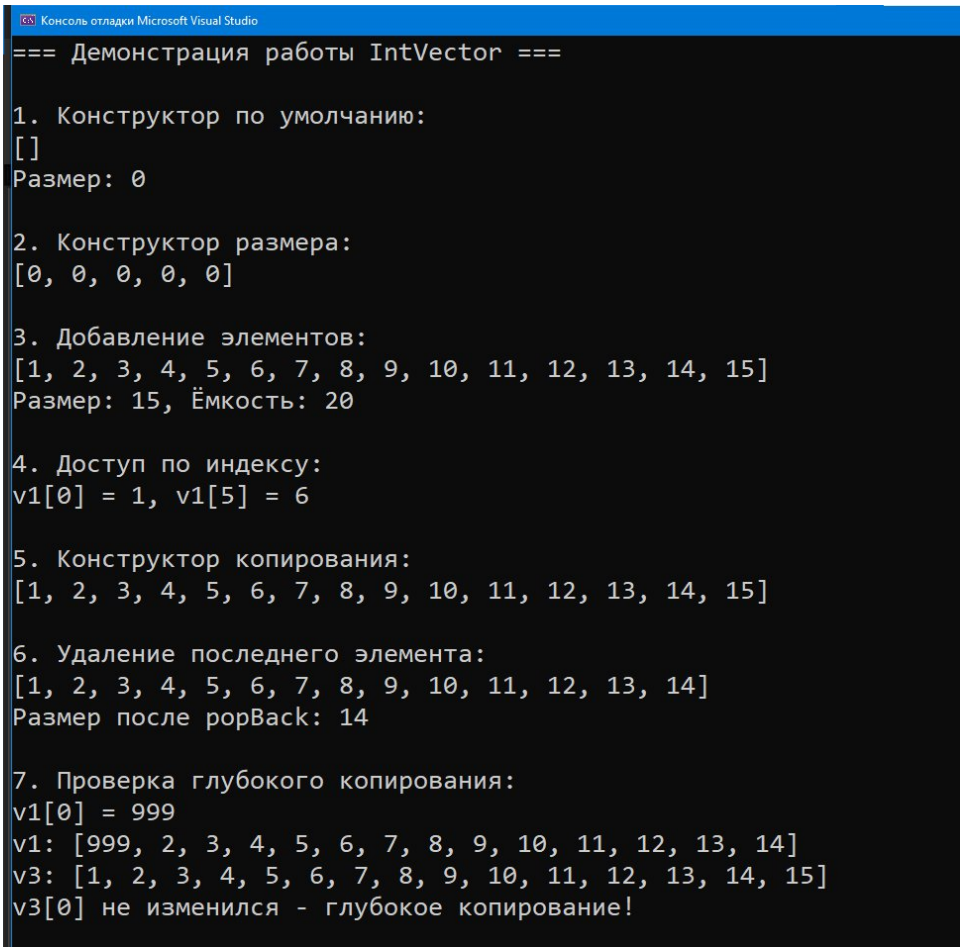
- Прямая инициализация, а не присваивание
- Обязателен для константных полей и ссылок
- Более эффективен для полей-объектов

Разобранный пример: реализация базовой версии IntVector

```
#include <iostream>
#include <stdexcept>

class IntVector {
private:
    int* data;
    int size;
    int capacity;

public:
    // Конструктор по умолчанию
    IntVector() : size(0), capacity(10) {
        data = new int[capacity];
    }

    // Конструктор размера
    IntVector(int n) : size(n), capacity(n) {
        if (n < 0) {
            throw std::invalid_argument("Размер не может быть отрицательным");
        }
        if (n == 0) {
            capacity = 1;
        }
        data = new int[capacity];
        for (int i = 0; i < size; ++i) {
            data[i] = 0;
        }
    }

    // Конструктор копирования
    IntVector(const IntVector& other) : size(other.size), capacity(other.capacity) {
        data = new int[capacity];
        for (int i = 0; i < size; ++i) {
            data[i] = other.data[i];
        }
    }

    // Деструктор
```

```
~IntVector() {
    delete[] data;
}

// Оператор присваивания
IntVector& operator=(const IntVector& other) {
    if (this != &other) {
        int* newData = new int[other.capacity];
        for (int i = 0; i < other.size; ++i) {
            newData[i] = other.data[i];
        }
        delete[] data;
        data = newData;
        size = other.size;
        capacity = other.capacity;
    }
    return *this;
}

int& operator[](int index) {
    if (index < 0 || index >= size) {
        throw std::out_of_range("Индекс вне диапазона");
    }
    return data[index];
}

const int& operator[](int index) const {
    if (index < 0 || index >= size) {
        throw std::out_of_range("Индекс вне диапазона");
    }
    return data[index];
}

int getSize() const { return size; }
int getCapacity() const { return capacity; }
bool isEmpty() const { return size == 0; }

void pushBack(int value) {
    if (size >= capacity) {
        int newCapacity = capacity * 2;
        int* newData = new int[newCapacity];
        for (int i = 0; i < size; ++i) {
            newData[i] = data[i];
        }
        delete[] data;
        data = newData;
        capacity = newCapacity;
    }
    data[size] = value;
    size++;
}

void popBack() {
    if (isEmpty()) {
        throw std::underflow_error("Вектор пуст");
    }
    size--;
}

void print() const {
    std::cout << "[";
    for (int i = 0; i < size; ++i) {
        std::cout << data[i];
        if (i < size - 1) std::cout << ", ";
```

```
        }
        std::cout << "]\n";
    }
};

// Демонстрация работы
int main() {
    setlocale(LC_ALL, "ru");

    std::cout << "=== Демонстрация работы IntVector ===\n\n";

    // 1. Конструктор по умолчанию
    std::cout << "1. Конструктор по умолчанию:\n";
    IntVector v1;
    v1.print();
    std::cout << "Размер: " << v1.getSize() << "\n\n";

    // 2. Конструктор размера
    std::cout << "2. Конструктор размера:\n";
    IntVector v2(5);
    v2.print();
    std::cout << "\n";

    // 3. Добавление элементов
    std::cout << "3. Добавление элементов:\n";
    for (int i = 1; i <= 15; ++i) {
        v1.pushBack(i);
    }
    v1.print();
    std::cout << "Размер: " << v1.getSize() << ", Ёмкость: " << v1.getCapacity() << "\n\n";

    // 4. Доступ по индексу
    std::cout << "4. Доступ по индексу:\n";
    std::cout << "v1[0] = " << v1[0] << ", v1[5] = " << v1[5] << "\n\n";

    // 5. Конструктор копирования
    std::cout << "5. Конструктор копирования:\n";
    IntVector v3 = v1;
    v3.print();
    std::cout << "\n";

    // 6. popBack
    std::cout << "6. Удаление последнего элемента:\n";
    v1.popBack();
    v1.print();
    std::cout << "Размер после popBack: " << v1.getSize() << "\n\n";

    // 7. Проверка глубокого копирования
    std::cout << "7. Проверка глубокого копирования:\n";
    v1[0] = 999;
    std::cout << "v1[0] = 999\n";
    std::cout << "v1: "; v1.print();
    std::cout << "v3: "; v3.print();
    std::cout << "v3[0] не изменился — глубокое копирование!\n\n";

    return 0;
}
```



Задания для самостоятельного выполнения

Базовый уровень

Общее задание: Реализовать класс IntVector с минимальным набором методов, указанных в варианте.

№ вар.	Конструкторы	Обязательные методы
1	По умолчанию, с параметром (размер)	pushBack, popBack, getSize, print
2	По умолчанию, с параметром (размер)	pushFront, popFront, getSize, print
3	По умолчанию, с параметром (размер, значение)	pushBack, popBack, isEmpty, print

4	По умолчанию, с параметром (размер)	insertAt(index), removeAt(index), getSize, print
5	По умолчанию, с параметром (размер, значение)	pushFront, popFront, getCapacity, print
6	По умолчанию, с параметром (размер)	pushBack, popBack, find(value), print
7	По умолчанию, с параметром (размер, значение)	| pushBack, popBack, get(index), set(index, value), print
8	По умолчанию, с параметром (размер)	pushFront, popFront, find(value), print
9	По умолчанию, с параметром (размер, значение)	insertAt(index), removeAt(index), isEmpty, print
10	По умолчанию, с параметром (размер)	pushBack, popBack, clear, print

Средний уровень

Общее задание: Реализовать класс IntVector с полной поддержкой «правила трёх», всеми основными методами и дополнительной функциональностью согласно варианту.

№ вар.	**Дополнительная функциональность**
1	Метод reserve(int newCapacity) - резервирование памяти; метод shrinkToFit() - уменьшение ёмкости до размера
2	Метод reverse() - разворот массива; метод sort() - сортировка по возрастанию
3	Метод sum() - сумма всех элементов; метод average() - среднее арифметическое

4	Метод min() - минимальный элемент; метод max() - максимальный элемент
5	Метод indexOf(int value) - поиск первого вхождения; метод lastIndexOf(int value) - поиск последнего вхождения
6	Метод removeAll(int value) - удаление всех элементов с заданным значением; метод count(int value) - подсчёт количества вхождений
7	Метод slice(int start, int end) - создание нового вектора из диапазона; метод concat(const IntVector& other) - объединение векторов
8	Метод resize(int newSize, int defaultValue) - изменение размера с заполнением; метод assign(int n, int value) - заполнение n элементами
9	Метод swap(IntVector& other) - обмен содержимым двух векторов; метод shuffle() - случайное перемешивание
10	Метод unique() - удаление дубликатов; метод rotate(int k) — циклический сдвиг на k позиций

Продвинутый уровень

Общее задание: Реализовать класс IntVector с поддержкой «правила пяти» (конструктор перемещения и оператор присваивания перемещением), исключениями и дополнительной функциональностью.

№ вар.	**Дополнительная функциональность**
1	Реализовать итераторы (begin, end) для поддержки range-based for. Добавить метод insert(iterator pos, int value)
2	Реализовать операторы сравнения: ==, !=, <, >, <=, >= для

	сравнения двух векторов
3	Реализовать операторы + (конкатенация векторов) и (умножение вектора на число)
4	Реализовать метод merge(const IntVector& other) - слияние двух отсортированных векторов; метод isSorted() - проверка на упорядоченность
5	Реализовать поддержку пользовательских аллокаторов: возможность передавать функцию выделения памяти
6	Реализовать метод apply(int (func)(int)) - применение функции ко всем элементам; метод filter(bool (pred)(int)) - фильтрация элементов
7	Реализовать сериализацию: методы saveToFile(const char filename) и loadFromFile(const char filename) для сохранения/загрузки в бинарный файл
8	Реализовать метод subvector(int start, int count) - создание подвектора; метод replace(int oldValue, int newValue) - замена всех вхождений
9	Реализовать метод partition(int pivot) - разделение на две части (меньше pivot и больше/равно); метод quickSort() - быстрая сортировка
10	Реализовать ленивое копирование (copy-on-write) для оптимизации памяти при копировании

Контрольные вопросы

1. Что такое динамическая память и чем она отличается от статической?

2. Для чего используется оператор new[] и delete[]?

3. Что такое «правило трёх» и почему оно важно для классов, управляющих ресурсами?

4. В чём разница между конструктором копирования и оператором присваивания?

5. Что произойдёт, если не реализовать конструктор копирования для класса с указателем?

6. Зачем нужен список инициализации? В каких случаях он обязателен?

7. Что такое this и для чего он используется в операторе присваивания?

8. Почему в операторе присваивания нужна проверка if (this != &other)?

9. В каком порядке вызываются деструкторы для объектов, созданных в обратном порядке?

10. Что такое «глубокое копирование» и чем оно отличается от «поверхностного»?

11. Для чего нужен метод reserve() и чем он отличается от resize()?

12. Как организовать автоматическое расширение массива при добавлении элементов?

13. Что такое RAII (Resource Acquisition Is Initialization)?

14. В каких случаях может возникнуть утечка памяти в классе IntVector?

15. Что такое const-методы и зачем они нужны?

Требования к отчёту

1. Титульный лист с указанием названия работы, ФИО студента, номера группы, варианта

2. Цель работы (формулируется самостоятельно на основе предложенной)

3. Теоретическая часть (краткое изложение основных понятий: динамическая память, правило трёх, конструкторы, деструктор)

4. Постановка задачи (текст задания согласно варианту)

5. Листинг программы с комментариями:

- Заголовочный файл (класс IntVector)

- Файл реализации методов

- Демонстрационная программа (main)

6. Результаты работы (скриншоты или копия вывода программы)

7. Анализ результатов (что получилось, какие были трудности, как проверялась корректность)

8. Ответы на контрольные вопросы (письменно, развёрнуто)

9. Выводы (что нового узнали, какие навыки приобрели)

Критерии оценивания

Базовый уровень (максимум 60 баллов) – оценка «удовлетворительно»

Критерий	**Баллы**
Класс реализован корректно, поля приватные	10
Конструкторы реализованы (по умолчанию и параметризованный)	10
Деструктор реализован, освобождает память	10
Указанные в варианте методы работают корректно	15
Демонстрационная программа показывает работу всех методов	10
Код отформатирован, есть комментарии	5

Средний уровень (максимум 80 баллов) – оценка «хорошо»

Критерий	Баллы
Все требования базового уровня	40
Реализовано «правило трёх» (конструктор копирования, оператор присваивания)	15
Реализована дополнительная функциональность по варианту	15
Корректная обработка ошибок (выход за границы, переполнение)	5
Наличие проверки на самоприсваивание в операторе присваивания	5

Продвинутый уровень (максимум 100 баллов) – оценка «отлично»

Критерий	Баллы
Все требования среднего уровня	60
Реализовано «правило пяти» (конструктор и оператор перемещения)	10
Реализована продвинутая функциональность по варианту	15
Исключения и безопасность (RAII, обработка ошибок через исключения)	10
Дополнительные требования (итераторы, операторы, сериализация и т.д.)	5

Штрафные баллы

Нарушение	Штраф
Утечка памяти (не освобождена динамическая память)	-20
Двойное освобождение памяти	-15

Отсутствие проверки на самоприсваивание	-5
Использование публичных полей вместо геттеров/сеттеров	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка	**Баллы**
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №7.

Перегрузка операторов. Дружественные функции

Цель работы:

- Научиться перегружать операторы для пользовательских классов, расширяя их функциональность и делая интерфейс классов интуитивно понятным

- Освоить механизм дружественных функций и классов для предоставления доступа к закрытым членам

- Закрепить на практике принципы объектно-ориентированного программирования через реализацию естественных для класса операций

- Развить понимание разницы между перегрузкой операторов как методами класса и как дружественными функциями

Теоретическое обоснование

1. Понятие перегрузки операторов

Перегрузка операторов - это механизм, позволяющий определить собственное поведение стандартных операторов C++ (+, -, , /, ==, <, [] и др.) для объектов пользовательских типов.

```
Complex a(1, 2), b(3, 4);
Complex c = a + b;  // Как это работает? Без перегрузки — ошибка!
```

Аналогия: Как приставки к автомобилю (телефон, навигатор) расширяют его возможности, так перегрузка операторов расширяет возможности класса, позволяя ему «понимать» стандартные операции.

2. Необходимость применения перегрузки операторов

Без перегрузки	**С перегрузкой**
c = a.add(b);	c = a + b;
if (a.compare(b) == 0)	if (a == b)
v.insertAt(v.getSize(), value);	v.push_back(value);
m.multiply(n);	m n;

Преимущества перегрузки:

- Естественность - код читается как математическая формула или как работа со встроенными типами

- Удобство - цепочки операций (a + b - c d) пишутся компактно

- Идиоматичность - код на C++ становится похож на то, что ожидают другие программисты

3. Синтаксис перегрузки операторов

Общая форма записи:

```
// Внутри класса (как метод)
return_type operator@(parameters) {
    // реализация
}

// Вне класса (как свободная функция)
return_type operator@(parameters) {
    // реализация
}
```

где @ - любой перегружаемый оператор: +, -, , /, ==, !=, <, >, [], (), <<, >>и другие.

4. Какие операторы можно перегружать?

Можно перегружать	**Нельзя перегружать**
+ - * / %	::(разрешение области видимости)
== != < > <= >=	.(доступ к члену)
&& || !	.(доступ к члену по указателю)
& | ^ ~ << >>	?:(тернарный оператор)
++ --	sizeof
[] ()	typeid
+= -= *= /=	new delete(можно перегружать, но отдельно)
new[] delete[]	static_cast dynamic_cast const_cast reinterpret_cast
-> ->*	# ## (препроцессор)
,(запятая)	

Важно: Нельзя изменить:

- Приоритет оператора

- Ассоциативность оператора

- Количество операндов

- Смысл операторов для встроенных типов (нельзя переопределить int + int)

5. Метод vs Дружественная функция

Характеристика	**Метод класса**	**Дружественная функция**
Левый операнд	this(this)	Первый параметр
Доступ к private	Да	Да (благодаря friend)
Симметричность	Нет (a + b не равно b + a для разных типов)	Да (можно менять порядок)
Приведение типов	Только для правого операнда	Для обоих операндов
Принадлежность классу	Да	Нет (это свободная функция)

Правило выбора:

```
class Complex {
public:
    // Метод (левый операнд — *this)
    Complex operator+(const Complex& other) const;

    // Дружественная функция (симметричная) — для типов разных классов
    friend Complex operator*(double a, const Complex& b);

    // Дружественная функция для потоков
    friend std::ostream& operator<<(std::ostream& os, const Complex& c);
};
```

6. Перегрузка унарных операторов

Унарные операторы действуют на один операнд:

Оператор	**Пример**	**Метод**	**Дружественная функция**
++(префиксный)	++obj	operator++()	operator++(T&)
++(постфиксный)	obj++	operator++(int)	operator++(T&, int)

--	--obj/ obj--	аналогично	аналогично
-(унарный минус)	-obj	operator-()	operator-(const T&)
!(логическое НЕ)	!obj	operator!()	operator!(const T&)
~(побитовое НЕ)	~obj	operator~()	operator~(const T&)
*(разыменование)	ptr	operator()	operator(const T&)
&(адрес)	&obj	operator&()	operator&(const T&)

Пример перегрузки префиксного и постфиксного инкремента:

```
class Counter {
private:
    int value;
public:
    Counter(int v = 0) : value(v) {}

    // Префиксный ++obj
    Counter& operator++() {
        ++value;
        return *this;
    }

    // Постфиксный obj++ (фиктивный параметр int)
    Counter operator++(int) {
        Counter temp = *this;
        ++value;
        return temp;
    }
};
```

7. Перегрузка бинарных операторов

Бинарные операторы действуют на два операнда:

Оператор	**Метод**	**Дружественная функция**
+	T operator+(const T& other) const	T operator+(const T& a, const T& b)
-	аналогично	аналогично
*	аналогично	аналогично
/	аналогично	аналогично
%	аналогично	аналогично

==	bool operator==(const T& other) const	bool operator==(const T& a, const T& b)
!=	bool operator!=(const T& other) const	bool operator!=(const T& a, const T& b)
<	bool operator<(const T& other) const	bool operator<(const T& a, const T& b)
>	bool operator>(const T& other) const	bool operator>(const T& a, const T& b)
<=	bool operator<=(const T& other) const	bool operator<=(const T& a, const T& b)
>=	bool operator>=(const T& other) const	bool operator>=(const T& a, const T& b)

8. Перегрузка операторов ввода/вывода

Операторы <<и >>всегда перегружаются как дружественные функции (не методы), потому что левый операнд - std::ostreamили std::istream, а не объект вашего класса.

```
class Point {
private:
    int x, y;
public:
    Point(int x = 0, int y = 0) : x(x), y(y) {}

    // Дружественные функции для потокового ввода/вывода
    friend std::ostream& operator<<(std::ostream& os, const Point& p) {
        os << "(" << p.x << ", " << p.y << ")";
        return os;
    }

    friend std::istream& operator>>(std::istream& is, Point& p) {
        is >> p.x >> p.y;
        return is;
    }
};
```

9. Перегрузка оператора индексации []

Оператор []обычно перегружается для классов-контейнеров:

```
class Array {
private:
    int* data;
    int size;
public:
    Array(int n) : size(n), data(new int[n]) {}
    ~Array() { delete[] data; }

    // Не константная версия (для записи)
    int& operator[](int index) {
        if (index < 0 || index >= size) {
            throw std::out_of_range("Index out of range");
        }
        return data[index];
    }

    // Константная версия (для чтения)
    const int& operator[](int index) const {
        if (index < 0 || index >= size) {
            throw std::out_of_range("Index out of range");
        }
        return data[index];
    }
};
```

10. Перегрузка оператора вызова ()

Оператор ()делает объект функтором (функциональным объектом):

```
class Adder {
private:
    int amount;
public:
    Adder(int a = 0) : amount(a) {}

    int operator()(int x) const {
        return x + amount;
    }
};

int main() {
    Adder add5(5);
    int result = add5(10);  // 15
}
```

11. Дружественные функции и классы

Дружественная функция - функция, не являющаяся членом класса, но имеющая доступ к его закрытым (private) и защищённым (protected) членам.

```
class MyClass {
private:
    int secret;
public:
    MyClass(int s) : secret(s) {}

    // Объявляем внешнюю функцию другом
    friend void showSecret(const MyClass& obj);

    // Объявляем другой класс другом
    friend class FriendClass;
};

// Функция получает доступ к private переменной secret
void showSecret(const MyClass& obj) {
    std::cout << obj.secret;  // OK! Друг
}

class FriendClass {
public:
    void access(const MyClass& obj) {
        std::cout << obj.secret;  // OK! FriendClass — друг
    }
};
```

Свойства дружественных функций:

- Не являются членами класса

- Не наследуются

- Нарушают инкапсуляцию (используйте только когда необходимо)

- Объявляются внутри класса с ключевым словом friend

Когда использовать дружественные функции:

1. Перегрузка операторов <<и >>

2. Перегрузка бинарных операторов, где нужна симметричность

3. Реализация функций, работающих с двумя разными классами

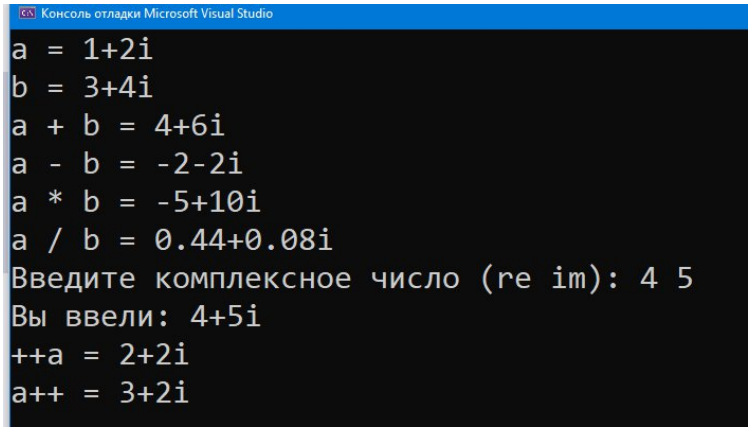
Примеры выполнения практической части

Пример 1. Класс «Комплексное число» (базовый уровень)

```
#include <iostream>
#include <cmath>
```

```
class Complex {
private:
    double re;   // действительная часть
    double im;   // мнимая часть

public:
    // Конструкторы
    Complex(double r = 0, double i = 0) : re(r), im(i) {}

    // Геттеры
    double getRe() const { return re; }
    double getIm() const { return im; }

    // Сеттеры
    void setRe(double r) { re = r; }
    void setIm(double i) { im = i; }

    // Перегрузка арифметических операторов (методы)
    Complex operator+(const Complex& other) const {
        return Complex(re + other.re, im + other.im);
    }

    Complex operator-(const Complex& other) const {
        return Complex(re - other.re, im - other.im);
    }

    Complex operator*(const Complex& other) const {
        return Complex(re * other.re - im * other.im,
            re * other.im + im * other.re);
    }

    Complex operator/(const Complex& other) const {
        double denominator = other.re * other.re + other.im * other.im;
        if (denominator == 0) {
            throw std::runtime_error("Division by zero");
        }
        return Complex((re * other.re + im * other.im) / denominator,
            (im * other.re - re * other.im) / denominator);
    }

    // Перегрузка операторов сравнения
    bool operator==(const Complex& other) const {
        return (re == other.re && im == other.im);
    }

    bool operator!=(const Complex& other) const {
        return !(*this == other);
    }

    // Префиксный инкремент (увеличиваем действительную часть)
    Complex& operator++() {
        ++re;
        return *this;
    }

    // Постфиксный инкремент
    Complex operator++(int) {
        Complex temp = *this;
        ++re;
        return temp;
    }

    // Оператор присваивания
```

```
    Complex& operator=(const Complex& other) {
        if (this != &other) {
            re = other.re;
            im = other.im;
        }
        return *this;
    }

    // Дружественные функции для потокового ввода/вывода
    friend std::ostream& operator<<(std::ostream& os, const Complex& c) {
        os << c.re;
        if (c.im >= 0) os << "+";
        os << c.im << "i";
        return os;
    }

    friend std::istream& operator>>(std::istream& is, Complex& c) {
        is >> c.re >> c.im;
        return is;
    }
};

int main() {
    setlocale(LC_ALL, "ru");

    Complex a(1, 2);
    Complex b(3, 4);

    std::cout << "a = " << a << std::endl;
    std::cout << "b = " << b << std::endl;
    std::cout << "a + b = " << a + b << std::endl;
    std::cout << "a - b = " << a - b << std::endl;
    std::cout << "a * b = " << a * b << std::endl;
    std::cout << "a / b = " << a / b << std::endl;

    Complex c;
    std::cout << "Введите комплексное число (re im): ";
    std::cin >> c;
    std::cout << "Вы ввели: " << c << std::endl;

    ++a;
    std::cout << "++a = " << a << std::endl;

    a++;
    std::cout << "a++ = " << a << std::endl;

    return 0;
}
```

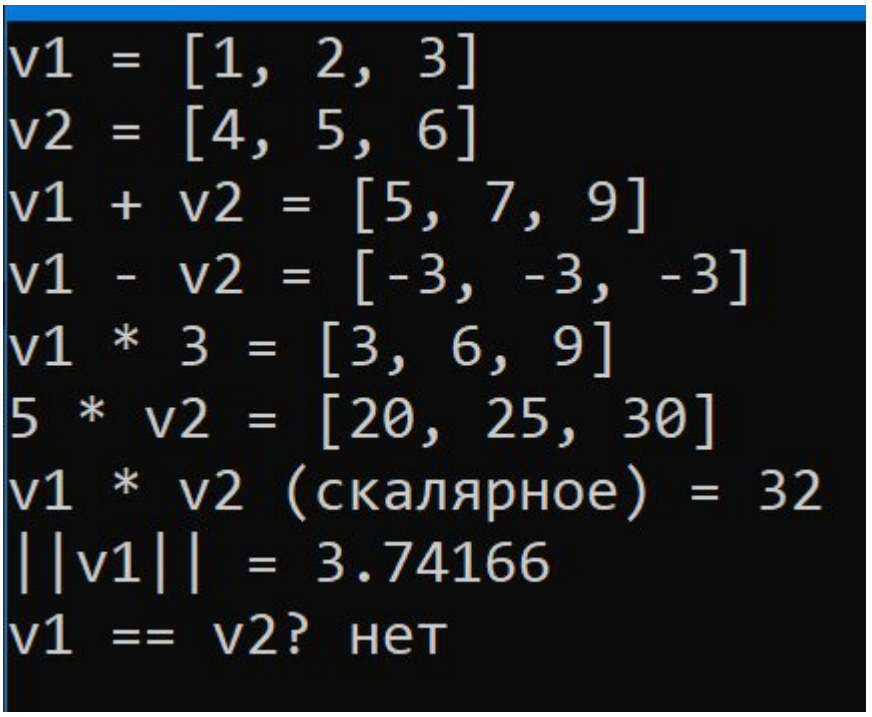


Пример 2. Класс «Вектор» с перегрузкой операторов (средний уровень)

```
#include <iostream>
#include <stdexcept>
#include <cmath>

class Vector {
private:
    int* data;
    int size;

public:
    // Конструкторы
    Vector(int n = 0) : size(n), data(new int[n]()) {}

    Vector(const Vector& other) : size(other.size), data(new int[other.size]) {
        for (int i = 0; i < size; i++) {
            data[i] = other.data[i];
        }
    }

    ~Vector() { delete[] data; }

    // Оператор присваивания
    Vector& operator=(const Vector& other) {
        if (this != &other) {
            delete[] data;
            size = other.size;
            data = new int[size];
            for (int i = 0; i < size; i++) {
                data[i] = other.data[i];
            }
        }
        return *this;
    }

    // Доступ по индексу (константный и неконстантный)
    int& operator[](int index) {
        if (index < 0 || index >= size) {
            throw std::out_of_range("Index out of range");
        }
        return data[index];
    }

    const int& operator[](int index) const {
        if (index < 0 || index >= size) {
            throw std::out_of_range("Index out of range");
        }
        return data[index];
    }

    // Сложение векторов
    Vector operator+(const Vector& other) const {
        if (size != other.size) {
            throw std::invalid_argument("Vector sizes must match");
        }
        Vector result(size);
        for (int i = 0; i < size; i++) {
            result.data[i] = data[i] + other.data[i];
        }
        return result;
```

```
}

// Вычитание векторов
Vector operator-(const Vector& other) const {
    if (size != other.size) {
        throw std::invalid_argument("Vector sizes must match");
    }
    Vector result(size);
    for (int i = 0; i < size; i++) {
        result.data[i] = data[i] - other.data[i];
    }
    return result;
}

// Умножение на скаляр (дружественная функция для симметрии)
friend Vector operator*(const Vector& v, int scalar) {
    Vector result(v.size);
    for (int i = 0; i < v.size; i++) {
        result.data[i] = v.data[i] * scalar;
    }
    return result;
}

friend Vector operator*(int scalar, const Vector& v) {
    return v * scalar;
}

// Скалярное произведение
int operator*(const Vector& other) const {
    if (size != other.size) {
        throw std::invalid_argument("Vector sizes must match");
    }
    int result = 0;
    for (int i = 0; i < size; i++) {
        result += data[i] * other.data[i];
    }
    return result;
}

// Сравнение векторов
bool operator==(const Vector& other) const {
    if (size != other.size) return false;
    for (int i = 0; i < size; i++) {
        if (data[i] != other.data[i]) return false;
    }
    return true;
}

bool operator!=(const Vector& other) const {
    return !(*this == other);
}

// Длина вектора (норма)
double length() const {
    double sum = 0;
    for (int i = 0; i < size; i++) {
        sum += data[i] * data[i];
    }
    return sqrt(sum);
}

// Вывод вектора
friend std::ostream& operator<<(std::ostream& os, const Vector& v) {
    os << "[";
```

```
        for (int i = 0; i < v.size; i++) {
            os << v.data[i];
            if (i < v.size - 1) os << ", ";
        }
        os << "]";
        return os;
    }

    // Ввод вектора
    friend std::istream& operator>>(std::istream& is, Vector& v) {
        for (int i = 0; i < v.size; i++) {
            is >> v.data[i];
        }
        return is;
    }

    int getSize() const { return size; }
};

int main() {
    setlocale(LC_ALL, "ru");

    Vector v1(3);
    v1[0] = 1; v1[1] = 2; v1[2] = 3;

    Vector v2(3);
    v2[0] = 4; v2[1] = 5; v2[2] = 6;

    std::cout << "v1 = " << v1 << std::endl;
    std::cout << "v2 = " << v2 << std::endl;
    std::cout << "v1 + v2 = " << v1 + v2 << std::endl;
    std::cout << "v1 - v2 = " << v1 - v2 << std::endl;
    std::cout << "v1 * 3 = " << v1 * 3 << std::endl;
    std::cout << "5 * v2 = " << 5 * v2 << std::endl;
    std::cout << "v1 * v2 (скалярное) = " << v1 * v2 << std::endl;
    std::cout << "||v1|| = " << v1.length() << std::endl;
    std::cout << "v1 == v2? " << (v1 == v2 ? "да" : "нет") << std::endl;

    return 0;
}
```



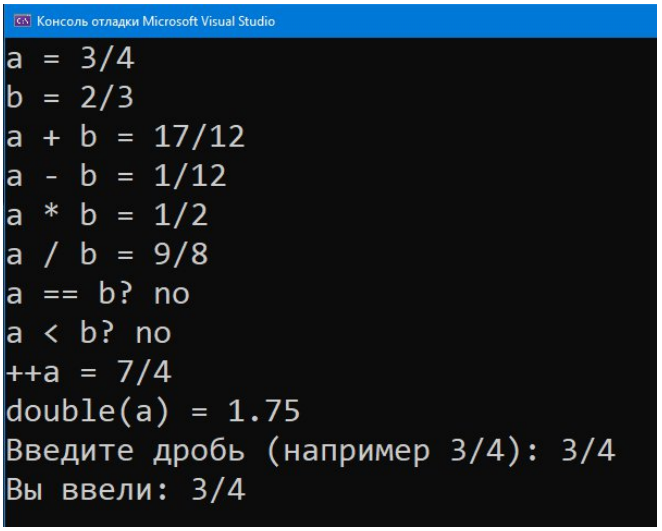
Пример 3. Класс «Рациональная дробь» (повышенный уровень)

```
#include <iostream>
#include <stdexcept>

class Rational {
private:
    int numerator;    // числитель
    int denominator;  // знаменатель (всегда > 0)

    // Собственная реализация НОД (алгоритм Евклида)
    static int gcd(int a, int b) {
        a = std::abs(a);
        b = std::abs(b);
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    // Сокращение дроби
    void reduce() {
        if (denominator == 0) {
            throw std::invalid_argument("Denominator cannot be zero");
        }
        int g = gcd(numerator, denominator);
        numerator /= g;
        denominator /= g;
        // Знак храним в числителе
        if (denominator < 0) {
            numerator = -numerator;
            denominator = -denominator;
        }
    }
```

```
public:
    // Конструкторы
    Rational(int n = 0, int d = 1) : numerator(n), denominator(d) {
        if (denominator == 0) {
            throw std::invalid_argument("Denominator cannot be zero");
        }
        reduce();
    }

    // Геттеры
    int getNumerator() const { return numerator; }
    int getDenominator() const { return denominator; }

    // Арифметические операторы
    Rational operator+(const Rational& other) const {
        return Rational(numerator * other.denominator + other.numerator * denominator,
            denominator * other.denominator);
    }

    Rational operator-(const Rational& other) const {
        return Rational(numerator * other.denominator - other.numerator * denominator,
            denominator * other.denominator);
    }

    Rational operator*(const Rational& other) const {
        return Rational(numerator * other.numerator,
            denominator * other.denominator);
    }

    Rational operator/(const Rational& other) const {
        if (other.numerator == 0) {
            throw std::runtime_error("Division by zero");
        }
        return Rational(numerator * other.denominator,
            denominator * other.numerator);
    }

    // Составное присваивание
    Rational& operator+=(const Rational& other) {
        *this = *this + other;
        return *this;
    }

    Rational& operator-=(const Rational& other) {
        *this = *this - other;
        return *this;
    }

    Rational& operator*=(const Rational& other) {
        *this = *this * other;
        return *this;
    }

    Rational& operator/=(const Rational& other) {
        *this = *this / other;
        return *this;
    }

    // Операторы сравнения
    bool operator==(const Rational& other) const {
        return numerator == other.numerator && denominator == other.denominator;
    }
```

```cpp
bool operator!=(const Rational& other) const {
    return !(*this == other);
}

bool operator<(const Rational& other) const {
    return numerator * other.denominator < other.numerator * denominator;
}

bool operator<=(const Rational& other) const {
    return *this < other || *this == other;
}

bool operator>(const Rational& other) const {
    return !(*this <= other);
}

bool operator>=(const Rational& other) const {
    return !(*this < other);
}

// Унарные операторы
Rational operator-() const {
    return Rational(-numerator, denominator);
}

// Префиксный инкремент (увеличиваем на 1)
Rational& operator++() {
    *this = *this + Rational(1);
    return *this;
}

// Постфиксный инкремент
Rational operator++(int) {
    Rational temp = *this;
    ++*this;
    return temp;
}

// Преобразование к double
explicit operator double() const {
    return static_cast<double>(numerator) / denominator;
}

// Дружественные функции для ввода/вывода
friend std::ostream& operator<<(std::ostream& os, const Rational& r) {
    if (r.denominator == 1) {
        os << r.numerator;
    }
    else {
        os << r.numerator << "/" << r.denominator;
    }
    return os;
}

friend std::istream& operator>>(std::istream& is, Rational& r) {
    int n, d = 1;
    char slash;
    is >> n;
    if (is.peek() == '/') {
        is >> slash >> d;
    }
    r = Rational(n, d);
    return is;
}
```

```
};

int main() {
    setlocale(LC_ALL, "ru");

    Rational a(3, 4);
    Rational b(2, 3);

    std::cout << "a = " << a << std::endl;
    std::cout << "b = " << b << std::endl;
    std::cout << "a + b = " << a + b << std::endl;
    std::cout << "a - b = " << a - b << std::endl;
    std::cout << "a * b = " << a * b << std::endl;
    std::cout << "a / b = " << a / b << std::endl;

    std::cout << "a == b? " << (a == b ? "yes" : "no") << std::endl;
    std::cout << "a < b? " << (a < b ? "yes" : "no") << std::endl;

    ++a;
    std::cout << "++a = " << a << std::endl;

    std::cout << "double(a) = " << static_cast<double>(a) << std::endl;

    Rational c;
    std::cout << "Введите дробь (например 3/4): ";
    std::cin >> c;
    std::cout << "Вы ввели: " << c << std::endl;

    return 0;
}
```



Задания для самостоятельного выполнения

Базовый уровень

Общее задание: Разработать класс согласно варианту. Реализовать:

- Конструктор по умолчанию

- Параметризованный конструктор
- Конструктор копирования
- Деструктор
- Геттеры и сеттеры
- Перегрузку операторов <<и >>(ввод/вывод)
- Перегрузку операторов ==и !=(сравнение)
- Перегрузку арифметических операторов +, -, , /

№ вар.	**Название класса**	**Поля**	**Дополнительная функциональность**
1.	Complex	re, im	Модуль комплексного числа
2.	Rational	num, den	Сокращение дроби
3.	Fraction	whole, numerator, denominator	Преобразование в смешанное число
4.	Time	hours, minutes, seconds	Перевод в секунды
5.	Date	day, month, year	Проверка високосного года
6.	Point2D	x, y	Расстояние до начала координат
7.	Point3D	x, y, z	Расстояние между двумя точками
8.	Money	rubles, kopecks	Сумма в копейках
9.	Temperature	celsius	Конвертация в фаренгейты
10.	Speed	kmh	Конвертация в м/с
11.	Length	meters	Конвертация в сантиметры
12.	Mass	kg	Конвертация в граммы
13.	Volume	liters	Конвертация в миллилитры
14.	Area	square_meters	Конвертация в квадратные сантиметры
15.	Angle	degrees	Конвертация в радианы

Средний уровень

Общее задание: Разработать класс согласно варианту. Реализовать:

- Все возможности базового уровня
- Перегрузку операторов +, -, *, /(арифметические)
- Перегрузку операторов +=, -=, *=, /=(составное присваивание)
- Перегрузку операторов <, >, <=, >=(сравнение)
- Перегрузку префиксного и постфиксного инкремента/декремента
- Перегрузка оператора ()(функтор)

№ вар.	**Название класса**	**Поля**	**Дополнительная функциональность**
1.	Complex	re, im	Аргумент комплексного числа
2.	Rational	num, den	Сравнение с плавающей точкой
3.	Vector2D	x, y	Скалярное и векторное произведение
4.	Vector3D	x, y, z	Скалярное и векторное произведение
5.	Matrix2x2	a, b, c, d	Определитель, обратная матрица
6.	Polynomial	coefficients[], degree	Вычисление значения в точке
7.	BigInteger	digits[]	Сложение, вычитание больших чисел
8.	Fraction	whole, num, den	Сравнение дробей
9.	Time	hours, minutes, seconds	Сложение/вычитание времени
10.	Date	day, month, year	Разность дат в днях
11.	Set	elements[], size	Объединение, пересечение множеств

12.	String	str, length	Конкатенация, сравнение строк
13.	BitArray	bits[], size	Побитовые AND, OR, XOR
14.	Interval	start, end	Пересечение интервалов
15.	BankAccount	balance, rate	Начисление процентов

Повышенный уровень

Общее задание: Разработать класс согласно варианту. Реализовать:

- Все возможности среднего уровня
- Перегрузку оператора [](доступ по индексу)
- Перегрузку оператора ()(функтор)
- Операторы преобразования типов
- Дружественные функции для работы с разными типами
- Исключения для обработки ошибок

№ вар.	Название класса	Поля	Дополнительная функциональность
1.	Polynomial	coefficients[], degree	Интегрирование, дифференцирование
2.	Matrix	data[][], rows, cols	Умножение матриц, транспонирование
3.	SparseMatrix	values[], rows[], cols[], size	Оптимизированное умножение
4.	BigInteger	digits[]	Умножение, деление больших чисел
5.	ComplexMatrix	re[][], im[][], n, m	Комплексная матрица
6.	Fraction	num, den	Цепные дроби

7.	Quaternion	w, x, y, z	Кватернионная алгебра
8.	RGB	r, g, b	HSL конвертация, смешивание цветов
9.	IntervalArithmetic	low, high	Арифметика интервалов
10.	BigRational	num[], den[]	Дроби произвольной точности
11.	Tensor	data[], dimensions[]	Свёртка тензоров
12.	ComplexPolynomial	coeff_re[], coeff_im[], degree	Корни полинома (численно)
13.	SymbolicExpression	type, left, right	Дифференцирование выражений
14.	Statistics	data[], size	Корреляция, ковариация
15.	ODE	coefficients[], order	Решение ODE (метод Эйлера)

Контрольные вопросы

1. Что такое перегрузка операторов и зачем она нужна?

2. Какие операторы нельзя перегружать в C++?

3. В чём разница между перегрузкой оператора как метода класса и как дружественной функции?

4. Для каких операторов предпочтительнее использовать дружественные функции?

5. Как перегрузить префиксный и постфиксный инкремент? Чем они отличаются?

6. Почему операторы <<и >>нельзя перегрузить как методы класса?

7. Что такое дружественная функция? Каковы её особенности?

8. В каких случаях стоит использовать дружественные классы?

9. Что такое оператор преобразования типа? Как его перегрузить?

10. Как перегрузить оператор []для доступа к элементам?

11. Что такое функтор? Как перегрузить оператор ()?

12. Как перегрузить оператор newи delete?

13. Какие проблемы возникают при перегрузке операторов &&, ||и ,?

14. Что такое "правило трёх" в контексте перегрузки операторов?

15. Как перегрузить оператор ->? Когда это может понадобиться?

Требования к отчёту

1. Титульный лист с названием работы, ФИО студента, номером группы, вариантом

2. Цель работы (формулируется самостоятельно)

3. Теоретическая часть:

- Понятие перегрузки операторов
- Синтаксис перегрузки
- Различие между методом и дружественной функцией
- Правила выбора способа перегрузки

4. Постановка задачи (текст задания согласно варианту)

5. Листинг программы с комментариями:

- Заголовочный файл с объявлением класса
- Файл реализации методов
- Демонстрационная программа (main.cpp)

6. Результаты работы (скриншоты вывода программы)

7. Анализ результатов (объяснение работы перегруженных операторов)

8. Ответы на контрольные вопросы

9. Выводы (что нового узнали, какие сложности возникли)

Критерии оценивания

Базовый уровень (максимум 60 баллов)

Критерий	**Баллы**

Класс реализован корректно, поля private	5
Конструкторы (по умолчанию, параметризованный, копирования)	10
Деструктор	5
Геттеры и сеттеры	5
Перегрузка <<и >>(ввод/вывод)	10
Перегрузка ==и !=(сравнение)	10
Перегрузка арифметических операторов (+, -, , /)	10
Демонстрационная программа показывает работу всех операторов	5
Код отформатирован, есть комментарии	5

Средний уровень (максимум 80 баллов)

Критерий	**Баллы**
Все требования базового уровня	40
Перегрузка составных операторов (+=, -=, =, /=)	10
Перегрузка операторов сравнения (<, >, <=, >=)	10
Перегрузка префиксного и постфиксного инкремента/декремента	10
Перегрузка оператора ()(функтор)	10

Повышенный уровень (максимум 100 баллов)

Критерий	**Баллы**
Все требования среднего уровня	60
Перегрузка оператора [](доступ по индексу)	10
Операторы преобразования типов (explicit)	10
Дружественные функции для работы с разными типами	10
Обработка исключений	10

Штрафные баллы

Нарушение	**Штраф**
Нарушение "правила трёх"	-15
Некорректная работа операторов (несоответствие семантике)	-10
Отсутствие проверок (деление на ноль и т.п.)	-5
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка	**Баллы**
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Дополнительные рекомендации

1. При перегрузке операторов сохраняйте их естественную семантику

2. Для бинарных операторов, где нужна симметрия, используйте дружественные функции

3. Перегрузка =должна возвращать ссылку на объект (this)

4. Перегрузка []обычно предоставляет две версии (константную и неконстантную)

5. Используйте explicitдля операторов преобразования, если неявное преобразование нежелательно

Лабораторная работа №8

Композиция и агрегация классов. Статические члены

Цель работы:

1. Научиться проектировать отношения между классами: композицию, агрегацию и ассоциацию
2. Освоить реализацию композиции (поля-объекты, вложение структур) и агрегации (указатели, ссылки)
3. Понять разницу между композицией и агрегацией с точки зрения времени жизни объектов
4. Научиться использовать статические поля и методы для подсчёта объектов и реализации общих данных для всех экземпляров класса
5. Получить навыки правильного управления памятью при разных типах отношений между классами

Теоретическое обоснование

1. Отношения между классами

В объектно-ориентированном программировании классы редко существуют изолированно. Они взаимодействуют, объединяются в структуры, используют друг друга. Существует три основных типа отношений:

Отношение	Характер связи	Пример
Ассоциация	«использует», «знает о»	Врач → Пациент
Агрегация	«состоит из» (часть может жить без целого)	Автомобиль → Колёса
Комп	«состоит из» (часть не может жить без	Дом

Отношение	Характер связи	Пример
озиция	целого)	→ Комнаты

2. Композиция

Композиция - это отношение «часть–целое», при котором «части» не могут существовать без «целого». Целое полностью контролирует время жизни частей.

Ключевая характеристика: Часть умирает вместе с целым.

Реализация композиции в C++

```
// Композиция через поле-объект (часть создаётся внутри целого)
class Room {
private:
    double area;
    string purpose;
public:
    Room(double a, const string& p) : area(a), purpose(p) {}
};

class House {
private:
    string address;
    Room livingRoom;   // композиция! комнаты создаются внутри дома
    Room kitchen;
public:
    House(const string& addr, double area1, double area2)
```

```
        : address(addr), livingRoom(area1, "Гостиная"), kitchen(area2,
"Кухня") {}
};

// Композиция через динамическое выделение памяти
class Engine {
public:
    Engine(int hp) { cout << "Двигатель создан\n"; }
    ~Engine() { cout << "Двигатель уничтожен\n"; }
};

class Car {
private:
    Engine* engine;  // указатель, но управляет памятью сам
public:
    Car(int hp) : engine(new Engine(hp)) {}  // создаёт двигатель
    ~Car() { delete engine; }                // уничтожает двигатель
};

// Композиция через умные указатели (рекомендуемый способ)
#include <memory>

class Car {
private:
    std::unique_ptr<Engine> engine;      // автоматическое управление
памятью
public:
    Car(int hp) : engine(std::make_unique<Engine>(hp)) {}
    // деструктор не нужен - unique_ptr сам очистит память
```

```
};
```

3. Агрегация

Агрегация - это отношение «часть–целое», при котором «части» могут существовать независимо от «целого» и могут принадлежать разным «целым» одновременно.

Ключевая характеристика: Часть может существовать без целого.

Реализация агрегации в C++

```
class Book {
private:
    string title;
public:
    Book(const string& t) : title(t) {}
    string getTitle() const { return title; }
};

class Library {
private:
    string name;
    vector<Book*> books;  // агрегация: книги существуют независимо
public:
    Library(const string& n) : name(n) {}

    void addBook(Book* book) {
        books.push_back(book);
    }
    // В деструкторе книги НЕ удаляются!
};
```

```
int main() {
    Book b1("Война и мир");      // книга создаётся независимо
    Book b2("Преступление и наказание");

    Library lib("Городская");    // библиотека создаётся
    lib.addBook(&b1);            // книга добавляется в библиотеку
    lib.addBook(&b2);

    // Библиотека уничтожается, но книги остаются!
    return 0;
}
```

4. Сравнение композиции и агрегации

Характеристика	**Ассоциация**	**Агрегация**	**Композиция**
Сила связи	Слабая	Средняя	Сильная
Время жизни частей	Независимое	Дольше или равное целому	Вместе с целым
Целое отвечает за уничтожение	Нет	Нет	Да
Целое отвечает за	Нет	Нет	Да

Характеристика	Ассоциация	Агрегация	Композиция
создание			
Каскадное удаление	Нет	Нет	Да
Реализация	Указатели, ссылки	Указатели	Поля-объекты, unique_ptr

5. Статические члены класса

Статическое поле - принадлежит не отдельному объекту, а всему классу в целом. Существует в единственном экземпляре для всех объектов класса.

Статический метод - может быть вызван без создания объекта класса.

```
class Student {
private:
    string name;
    static int totalCount;  // объявление статического поля
public:
    Student(const string& n) : name(n) {
        totalCount++;       // увеличиваем счётчик при создании
    }

    ~Student() {
        totalCount--;       // уменьшаем при уничтожении
    }
```

```
    static int getTotalCount() {  // статический метод
        return totalCount;
    }
};

// Определение статического поля (обязательно вне класса!)
int Student::totalCount = 0;

int main() {
    cout << "Студентов: " << Student::getTotalCount() << endl;  // 0

    Student s1("Иван");
    Student s2("Мария");

    cout << "Студентов: " << Student::getTotalCount() << endl;  // 2

    {
        Student s3("Пётр");
        cout << "Студентов: " << Student::getTotalCount() << endl;  // 3
    }  // s3 уничтожается

    cout << "Студентов: " << Student::getTotalCount() << endl;  // 2
}
```

6. Паттерн «Одиночка» (Singleton)

Одиночка - порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр.

```
class Singleton {
```

```
private:
    static Singleton* instance;

    // Приватный конструктор - нельзя создать объект извне
    Singleton() {}

public:
    // Запрещаем копирование
    Singleton(const Singleton&) = delete;
    Singleton& operator=(const Singleton&) = delete;

    static Singleton* getInstance() {
        if (instance == nullptr) {
            instance = new Singleton();
        }
        return instance;
    }

    void doSomething() {
        cout << "Одиночка выполняет действие" << endl;
    }
};

// Определение статического поля
Singleton* Singleton::instance = nullptr;

int main() {
    // Singleton s;          // ОШИБКА! конструктор приватный
    Singleton* s1 = Singleton::getInstance();
```

```
    Singleton* s2 = Singleton::getInstance();

    cout << "s1 == s2: " << (s1 == s2) << endl;  // true - один и тот же объект
}
```

Примеры выполнения практической части

Пример 1. Композиция: классы «Дом» и «Комната»

Задача: Реализовать классы Room и House. Комнаты не могут существовать без дома (композиция).

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class Room {
private:
    string name;
    double area;

public:
    Room(const string& n, double a) : name(n), area(a) {
        cout << "Создана комната \"" << name << "\" площадью " << area << " м²" << endl;
    }

    ~Room() {
```

```
        cout << "Удалена комната \"" << name << "\"" << endl;
    }

    void print() const {
        cout << "  - " << name << ": " << area << " м²" << endl;
    }
};

class House {
private:
    string address;
    vector<Room> rooms;  // композиция: комнаты внутри дома

public:
    House(const string& addr) : address(addr) {
        cout << "Построен дом по адресу: " << address << endl;
    }

    ~House() {
        cout << "Снесён дом по адресу: " << address << endl;
    }

    void addRoom(const string& name, double area) {
        rooms.emplace_back(name, area);  // комната создаётся внутри дома
    }

    void print() const {
        cout << "Дом по адресу " << address << ":" << endl;
        for (const auto& room : rooms) {
```

```
            room.print();
        }
    }
};

int main() {
    setlocale(LC_ALL, "ru");

    House myHouse("ул. Цветочная, д. 15");

    myHouse.addRoom("Гостиная", 25.5);
    myHouse.addRoom("Кухня", 12.0);
    myHouse.addRoom("Спальня", 18.0);

    myHouse.print();

    // При выходе из main дом уничтожается, а вместе с ним - все комнаты
    return 0;
}
```

Ожидаемый вывод:

Построен дом по адресу: ул. Цветочная, д. 15
Создана комната "Гостиная" площадью 25.5 м²
Создана комната "Кухня" площадью 12 м²
Создана комната "Спальня" площадью 18 м²
Дом по адресу ул. Цветочная, д. 15:
- Гостиная: 25.5 м²
- Кухня: 12 м²

- Спальня: 18 м²

Снесён дом по адресу: ул. Цветочная, д. 15

Удалена комната "Спальня"

Удалена комната "Кухня"

Удалена комната "Гостиная"

Пример 2. Агрегация: классы «Библиотека» и «Книга»

Задача: Реализовать классы Book и Library. Книги могут существовать без библиотеки (агрегация).

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class Book {
private:
    string title;
    string author;

public:
    Book(const string& t, const string& a) : title(t), author(a) {
        cout << "Создана книга \"" << title << "\" (" << author << ")" << endl;
    }

    ~Book() {
        cout << "Удалена книга \"" << title << "\"" << endl;
    }
```

```
    void print() const {
        cout << "  - \"" << title << "\", " << author << endl;
    }
};

class Library {
private:
    string name;
    vector<Book*> books;  // агрегация: книги существуют независимо

public:
    Library(const string& n) : name(n) {
        cout << "Открыта библиотека \"" << name << "\"" << endl;
    }

    ~Library() {
        cout << "Закрыта библиотека \"" << name << "\"" << endl;
        // КНИГИ НЕ УНИЧТОЖАЮТСЯ!
    }

    void addBook(Book* book) {
        books.push_back(book);
        cout << "Книга \"" << book->getTitle() << "\" добавлена в
библиотеку" << endl;
    }

    void print() const {
        cout << "Библиотека \"" << name << "\" содержит:" << endl;
```

```
        for (const auto* book : books) {
            book->print();
        }
    }
};

// Нужен геттер для title (добавим в класс Book)
// В реальном коде нужно добавить метод getTitle()

int main() {
    setlocale(LC_ALL, "ru");

    // Книги создаются независимо от библиотеки
    Book b1("Война и мир", "Л. Толстой");
    Book b2("Преступление и наказание", "Ф. Достоевский");
    Book b3("Евгений Онегин", "А. Пушкин");

    {
        Library lib("Городская библиотека");
        lib.addBook(&b1);
        lib.addBook(&b2);
        lib.print();
    } // Библиотека закрывается, но книги остаются

    cout << "\nКниги всё ещё существуют после закрытия библиотеки:" << endl;
    b1.print();
    b2.print();
```

```
    return 0;
} // Книги удаляются только здесь
```

Пример 3. Статические члены: счётчик объектов

Задача: Реализовать класс Employee со статическим полем для подсчёта количества созданных объектов.

```
#include <iostream>
#include <string>

using namespace std;

class Employee {
private:
    string name;
    double salary;
    static int employeeCount;     // статическое поле - счётчик сотрудников
    static double totalSalary;    // статическое поле - сумма всех зарплат

public:
    Employee(const string& n, double s) : name(n), salary(s) {
        employeeCount++;
        totalSalary += salary;
        cout << "Создан сотрудник: " << name << ", зарплата: " << salary << endl;
    }

    ~Employee() {
        employeeCount--;
```

```
        totalSalary -= salary;
        cout << "Уволен сотрудник: " << name << endl;
    }

    // Статические методы
    static int getCount() {
        return employeeCount;
    }

    static double getAverageSalary() {
        if (employeeCount == 0) return 0;
        return totalSalary / employeeCount;
    }

    // Обычные методы
    void setSalary(double newSalary) {
        totalSalary = totalSalary - salary + newSalary;
        salary = newSalary;
    }

    void print() const {
        cout << "  " << name << ": " << salary << " руб." << endl;
    }
};

// Определение статических полей
int Employee::employeeCount = 0;
double Employee::totalSalary = 0;
```

```
int main() {
    setlocale(LC_ALL, "ru");

    cout << "Всего сотрудников: " << Employee::getCount() << endl;
    cout << "Средняя зарплата: " << Employee::getAverageSalary() << endl;

    Employee e1("Иванов Иван", 50000);
    Employee e2("Петров Пётр", 60000);
    Employee e3("Сидорова Мария", 55000);

    cout << "\nВсего сотрудников: " << Employee::getCount() << endl;
    cout << "Средняя зарплата: " << Employee::getAverageSalary() << endl;

    cout << "\nСписок сотрудников:" << endl;
    e1.print();
    e2.print();
    e3.print();

    e2.setSalary(65000);
    cout << "\nПосле повышения зарплаты Петрова:" << endl;
    cout << "Средняя зарплата: " << Employee::getAverageSalary() << endl;

    return 0;
}
```

Индивидуальные задания

Базовый уровень (15 вариантов)

Общее задание: Реализовать два класса, связанных отношением композиция (часть не может существовать без целого). Продемонстрировать создание и уничтожение объектов.

№ вар.	Класс «Целое»	Класс «Часть»	Поля класса «Часть»
1	House	Room	название, площадь
2	Car	Wheel	диаметр, тип (зимняя/летняя)
3	Computer	CPU	модель, частота, ядра
4	University	Faculty	название, количество кафедр
5	Library	Bookshelf	номер, вместимость
6	Book	Chapter	название, количество страниц
7	School	Classroom	номер, вместимость
8	Apartment	Window	ширина, высота, материал
9	Phone	Battery	ёмкость (мАч), тип

№ вар.	Класс «Целое»	Класс «Часть»	Поля класса «Часть»
10	Ship	Deck	название, уровень
11	Building	Floor	номер, высота
12	Document	Page	номер, содержание (string)
13	Organization	Department	название, количество сотрудников
14	Train	Carriage	номер, тип (купе/плацкарт), мест
15	Guitar	String	номер, толщина

Средний уровень (15 вариантов)

Общее задание: Реализовать два класса, связанных отношением агрегация (часть может существовать без целого). Добавить статическое поле для подсчёта количества объектов класса «Часть». Продемонстрировать, что при уничтожении целого части продолжают существовать.

№ вар.	Класс «Целое»	Класс «Часть»	Поля класса «Часть»
1	Library	Book	название, автор, год
2	Team	Player	имя, номер, амплуа
3	Course	Student	имя, фамилия, группа
4	Playlist	Song	название, исполнитель, длительность
5	Group	Member	имя, роль
6	Fleet	Vehicle	модель, год выпуска, номер
7	Catalog	Product	название, цена, артикул
8	Department	Employee	имя, должность, оклад
9	Menu	Dish	название, цена, калории
10	Squad	Soldier	звание, имя, опыт
11	Roster	Athlete	имя, вид спорта, рекорд
12	Schedule	Lesson	предмет, время, аудитория

№ вар.	Класс «Целое»	Класс «Часть»	Поля класса «Часть»
13	Album	Photo	название, дата, размер
14	Cart	Item	название, цена, количество
15	Wardrobe	Clothing	тип, размер, цвет

Продвинутый уровень (15 вариантов)

Общее задание: Реализовать один из вариантов ниже. В каждом варианте необходимо:

Реализовать отношение композиция или агрегация согласно варианту

Добавить статическое поле для подсчёта объектов одного из классов

Реализовать паттерн «Одиночка» (Singleton) для одного из классов

Продемонстрировать работу всех реализованных возможностей

№ вар.	Задание
1	Система учёта студентов: University (одиночка) + Student (агрегация). Подсчёт количества студентов
2	Система управления библиотекой: Library (одиночка) + Book (агрегация). Подсчёт количества книг
3	Система учёта сотрудников: Company (одиночка)

№ вар.	Задание
	+ Employee (агрегация). Подсчёт сотрудников и средней зарплаты
4	Система заказов в ресторане: Restaurant (одиночка) + Order (агрегация) + Dish (композиция в Order)
5	Система управления курсами: TrainingCenter (одиночка) + Course (агрегация) + Student (агрегация в Course)
6	Система управления таксопарком: TaxiCompany (одиночка) + Car (агрегация) + Driver (агрегация)
7	Система учета поставок: Warehouse (одиночка) + Product (агрегация) + Supplier (агрегация)
8	Система управления гостиницей: Hotel (одиночка) + Room (агрегация) + Booking (композиция в Room)
9	Система управления проектами: ProjectManager (одиночка) + Project (агрегация) + Task (композиция в Project)
10	Система интернет-магазина: Store (одиночка) + Customer (агрегация) + Purchase (композиция)
11	Система управления автопарком: Depot (одиночка) + Bus (агрегация) + Route (агрегация в Bus)

№ вар.	Задание
12	Система управления кинотеатром: Cinema (одиночка) + Movie (агрегация) + Session (композиция в Movie)
13	Система управления больницей: Hospital (одиночка) + Doctor (агрегация) + Patient (агрегация) + Appointment (композиция)
14	Система управления спортивным клубом: SportsClub (одиночка) + Coach (агрегация) + Section (агрегация) + Member (агрегация в Section)
15	Система управления банком: Bank (одиночка) + Client (агрегация) + Account (композиция в Client)

Контрольные вопросы

1. Что такое композиция? Приведите пример.
2. Что такое агрегация? Чем она отличается от композиции?
3. Как реализовать композицию в C++? Приведите пример кода.
4. Как реализовать агрегацию в C++? Приведите пример кода.
5. В чём разница между временем жизни частей при композиции и агрегации?
6. Что такое статическое поле класса? Чем оно отличается от обычного поля?
7. Как объявить и определить статическое поле? Где оно хранится?
8. Что такое статический метод? Можно ли из него обратиться к нестатическим полям?
9. Для чего используются статические счётчики объектов?
10. Что такое паттерн «Одиночка» (Singleton)? Для чего он нужен?
11. Как запретить создание копий объекта (конструктор копирования)?

12. В каком порядке вызываются конструкторы и деструкторы при композиции?
13. Можно ли изменить статическое поле из обычного метода?
14. Почему статические поля нужно определять вне класса?
15. В каких случаях стоит использовать композицию, а в каких - агрегацию?

Требования к отчёту

Титульный лист с названием работы, ФИО студента, номером группы, вариантом

Цель работы (формулируется самостоятельно на основе предложенной)

Теоретическая часть (основные понятия: композиция, агрегация, статические члены)

Постановка задачи (текст задания согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода программы)

Ответы на контрольные вопросы (письменно, развёрнуто)

Выводы (что нового узнали, какие навыки приобрели)

Ссылка на Git-репозиторий с кодом программы

Критерии оценивания

Базовый уровень (максимум 60 баллов)

Критерий	Баллы
Реализованы два класса с отношением композиция	20
Демонстрация создания и уничтожения объектов	15
Корректное управление памятью (нет утечек)	15
Код отформатирован, есть комментарии	10

Средний уровень (максимум 80 баллов)

Критерий	Баллы
Все требования базового уровня	40
Реализована агрегация (два класса)	15
Реализовано статическое поле для подсчёта объектов	15
Демонстрация, что при уничтожении целого части продолжают существовать	10

Продвинутый уровень (максимум 100 баллов)

Критерий	Баллы
Все требования среднего уровня	60
Реализация паттерна «Одиночка» (Singleton)	15
Реализация статических методов для работы со статическими полями	10
Корректная обработка всех граничных случаев	10
Наличие ссылки на Git-репозиторий	5

Штрафные баллы

Нарушение	Штраф
Утечка памяти (при композиции)	-20
Неправильное использование статических полей	-10

Нарушение	Штраф
Отсутствует проверка на nullptr в агрегации	-10
Отсутствуют комментарии в коде	-5
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка (традиционная)	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №9

Анализ сложности алгоритмов. Шаблоны классов

Цель работы:

1. Научиться оценивать эффективность алгоритмов с помощью O-нотации (Big O)
2. Освоить классы сложности: O(1), O(log n), O(n), O(n log n), O(n^2)
3. Научиться проводить экспериментальный анализ производительности алгоритмов
4. Освоить создание шаблонов функций и классов для работы с различными типами данных

5. Получить навыки сравнения производительности различных реализаций одной задачи

Теоретическое обоснование

1. Что такое анализ сложности алгоритмов?

Анализ сложности - это математический способ оценки эффективности алгоритма в зависимости от размера входных данных.

Два вида сложности:

1. Временная сложность (Time Complexity) - сколько времени выполняется алгоритм
2. Пространственная сложность (Space Complexity) - сколько памяти использует алгоритм

2. O-нотация (Big O)

O-нотация - это математическая запись, описывающая асимптотическое поведение функции при стремлении аргумента к бесконечности.

Формальное определение:

Говорят, что $f(n) = O(g(n))$, если существуют положительные константы c и n_0 такие, что $0 \leq f(n) \leq c \times g(n)$ для всех $n \geq n_0$

Простым языком: O-нотация показывает, как быстро растёт время работы алгоритма при увеличении входных данных, отбрасывая константы и младшие члены.

Исходная формула	Упрощение	Сложность
$5n + 3$	n	$O(n)$
$2n^2 + 100n + 1000$	n^2	$O(n^2)$
$3n \log n + 4n$	$n \log n$	$O(n \log n)$
1000	1	$O(1)$

3. Основные классы сложности

Сложность	**Название**	**Описание**	**Пример**
$O(1)$	Константная	Время не зависит от размера данных	Доступ по индексу в массиве

Сложность	Название	Описание	Пример
O(log n)	Логарифмическая	Увеличивается медленно, при удвоении данных время растёт на константу	Бинарный поиск
O(n)	Линейная	Время растёт пропорционально размеру данных	Линейный поиск, сумма элементов
O(n log n)	Линейно-логарифмическая	Эффективные сортировки	Быстрая сортировка, слиянием
$O(n^2)$	Квадратичная	Время растёт квадратично	Пузырьковая сортировка
$O(2^n)$	Экспоненциальная	Время удваивается при добавлении одного элемента	Рекурсивное вычисление Фибоначчи
O(n!)	Факториальная	Самый медленный рост	Генерация всех перестановок

4. Сравнение классов сложности

n	O(1)	O(log n)	O(n)	O(n log n)	$O(n^2)$	$O(2^n)$
10	1	3	10	33	100	1 024
1	1	7	1	66	10	~1×10

n	O (1)	O (log n)	O (n)	O (n log n)	O (n²)	O(2ⁿ)
00			00	4	000	30
1 000	1	1 0	1 000	9 966	1 000 000	невозможно
1 000 000	1	2 0	1 000 000	20 000 000	1 ×10¹²	невозможно

5. Определение сложности по коду

```
// O(1) - константная
int getFirst(int arr[], int n) {
    return arr[0];  // одна операция, не зависит от n
}

// O(n) - линейная
int sum(int arr[], int n) {
    int total = 0;
    for (int i = 0; i < n; i++) {  // n итераций
        total += arr[i];
    }
    return total;
}

// O(n²) - квадратичная (два вложенных цикла)
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}

// O(log n) - логарифмическая (деление пополам)
int binarySearch(int arr[], int n, int x) {
    int left = 0, right = n - 1;
```

```
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == x) return mid;
        if (arr[mid] < x) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}
```

6. Шаблоны функций

Шаблоны позволяют писать код, работающий с разными типами данных.

```
// Шаблон функции для поиска максимума
template <typename T>
T max(T a, T b) {
    return (a > b) ? a : b;
}

// Шаблон функции для обмена значений
template <typename T>
void swap(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}

// Шаблон функции для линейного поиска
template <typename T>
int linearSearch(const T arr[], int n, const T& value) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == value) return i;
    }
    return -1;
}

// Использование
int main() {
    int a = 5, b = 10;
    swap(a, b);                 // swap<int>

    double x = 3.14, y = 2.71;
    swap(x, y);                 // swap<double>
```

```
    string s1 = "Hello", s2 = "World";
    swap(s1, s2);               // swap<string>
}
```

7. Шаблоны классов

```
// Шаблонный класс Pair
template <typename T1, typename T2>
class Pair {
private:
    T1 first;
    T2 second;

public:
    Pair(const T1& f, const T2& s) : first(f), second(s) {}

    T1 getFirst() const { return first; }
    T2 getSecond() const { return second; }

    void setFirst(const T1& f) { first = f; }
    void setSecond(const T2& s) { second = s; }
};

// Шаблонный класс Stack
template <typename T>
class Stack {
private:
    T* data;
    int topIndex;
    int capacity;

public:
    Stack(int size = 10) : capacity(size), topIndex(-1) {
        data = new T[capacity];
    }

    ~Stack() { delete[] data; }

    void push(const T& value) {
        if (topIndex < capacity - 1) {
            data[++topIndex] = value;
        }
```

```
    }

    T pop() {
        if (topIndex >= 0) {
            return data[topIndex--];
        }
        throw std::out_of_range("Stack is empty");
    }

    bool isEmpty() const { return topIndex == -1; }
};
```

8. Экспериментальный анализ производительности

```
#include <iostream>
#include <chrono>

using namespace std;
using namespace chrono;

// Функция для измерения времени выполнения
template <typename Func>
long long measureTime(Func func) {
    auto start = high_resolution_clock::now();
    func();
    auto end = high_resolution_clock::now();
    return duration_cast<microseconds>(end - start).count();
}

int main() {
    // Измерение времени линейного поиска
    const int N = 100000;
    int arr[N];
    for (int i = 0; i < N; i++) arr[i] = i;

    auto time = measureTime([&]() {
        for (int i = 0; i < 1000; i++) {
            linearSearch(arr, N, N - 1);
        }
    });

    cout << "Время выполнения: " << time << " мкс" << endl;
```

```
}
```

Примеры выполнения практической части
Пример 1. Определение сложности алгоритма (базовый уровень)
Задача: Определить сложность фрагментов кода и обосновать ответ.

```
#include <iostream>
using namespace std;

// Пример A: O(n)
int exampleA(int arr[], int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {  // n итераций
        sum += arr[i];             // O(1)
    }
    return sum;
}

// Пример B: O(n²)
int exampleB(int arr[], int n) {
    int count = 0;
    for (int i = 0; i < n; i++) {      // n итераций
        for (int j = 0; j < n; j++) {  // n итераций для каждого i
            if (arr[i] == arr[j]) count++;
        }
    }
    return count;
}

// Пример C: O(log n)
int exampleC(int n) {
    int count = 0;
    for (int i = 1; i < n; i *= 2) {  // i удваивается: log₂n итераций
        count++;
    }
    return count;
}

// Пример D: O(n log n)
int exampleD(int n) {
    int count = 0;
    for (int i = 0; i < n; i++) {          // n итераций
```

```
            for (int j = n; j > 0; j /= 2) {    // log₂n итераций
                count++;
            }
        }
        return count;
    }

    // Пример E: O(√n)
    int exampleE(int n) {
        int count = 0;
        for (int i = 0; i * i < n; i++) {  // i до √n
            count++;
        }
        return count;
    }

    int main() {
        cout << "Анализ сложности алгоритмов:" << endl;
        cout << "exampleA: O(n) - линейный проход" << endl;
        cout << "exampleB: O(n²) - два вложенных цикла" << endl;
        cout << "exampleC: O(log n) - удвоение шага" << endl;
        cout << "exampleD: O(n log n) - комбинация линейного и
логарифмического" << endl;
        cout << "exampleE: O(√n) - условие i*i < n" << endl;
        return 0;
    }
```

Пример 2. Сравнение производительности алгоритмов (средний уровень)

Задача: Реализовать линейный и бинарный поиск. Сравнить их производительность на массивах разного размера.

```
    #include <iostream>
    #include <vector>
    #include <chrono>
    #include <algorithm>

    using namespace std;
    using namespace chrono;

    // Линейный поиск O(n)
    template <typename T>
    int linearSearch(const vector<T>& arr, const T& value) {
```

```
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] == value) return i;
    }
    return -1;
}

// Бинарный поиск O(log n)
template <typename T>
int binarySearch(const vector<T>& arr, const T& value) {
    int left = 0, right = arr.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == value) return mid;
        if (arr[mid] < value) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}

// Измерение времени выполнения
template <typename Func>
long long measureTime(Func func) {
    auto start = high_resolution_clock::now();
    func();
    auto end = high_resolution_clock::now();
    return duration_cast<microseconds>(end - start).count();
}

int main() {
    setlocale(LC_ALL, "ru");

    vector<int> sizes = {10000, 100000, 500000, 1000000};

    cout << "=== Сравнение линейного и бинарного поиска ===" << endl;
    cout << "Размер массива\tЛинейный (мкс)\tБинарный (мкс)" << endl;

    for (int n : sizes) {
        vector<int> arr(n);
        for (int i = 0; i < n; i++) arr[i] = i;

        int target = n - 1;  // ищем последний элемент

        long long linearTime = measureTime([&]() {
```

```
            linearSearch(arr, target);
        });

        long long binaryTime = measureTime([&]() {
            binarySearch(arr, target);
        });

        cout << n << "\t\t" << linearTime << "\t\t" << binaryTime << endl;
    }

    return 0;
}
```

Ожидаемый вывод:

=== Сравнение линейного и бинарного поиска ===

Размер массива	Линейный (мкс)	Бинарный (мкс)
10000	45	1
100000	458	1
500000	2312	1
1000000	4678	2

Пример 3. Шаблонный класс Stack с анализом сложности (продвинутый уровень)

Задача: Реализовать шаблонный класс Stack с операциями push, pop, top. Проанализировать сложность каждой операции.

```
#include <iostream>
#include <stdexcept>

using namespace std;

template <typename T>
class Stack {
private:
    T* data;
    int topIndex;
    int capacity;

    void resize() {
        int newCapacity = capacity * 2;
        T* newData = new T[newCapacity];
        for (int i = 0; i <= topIndex; i++) {
            newData[i] = data[i];
        }
```

```
            delete[] data;
            data = newData;
            capacity = newCapacity;
            cout << "Расширение стека до " << capacity << " элементов" <<
endl;
        }

    public:
        Stack(int initialCapacity = 10)
            : capacity(initialCapacity), topIndex(-1) {
            data = new T[capacity];
            cout << "Создан стек ёмкостью " << capacity << endl;
        }

        ~Stack() {
            delete[] data;
            cout << "Стек уничтожен" << endl;
        }

        // O(1) в среднем, O(n) при расширении (амортизированно O(1))
        void push(const T& value) {
            if (topIndex >= capacity - 1) {
                resize();
            }
            data[++topIndex] = value;
        }

        // O(1)
        T pop() {
            if (isEmpty()) {
                throw out_of_range("Стек пуст");
            }
            return data[topIndex--];
        }

        // O(1)
        T top() const {
            if (isEmpty()) {
                throw out_of_range("Стек пуст");
            }
            return data[topIndex];
        }
```

```
    // O(1)
    bool isEmpty() const {
        return topIndex == -1;
    }

    // O(1)
    int size() const {
        return topIndex + 1;
    }

    // O(n) - для демонстрации
    void print() const {
        cout << "[";
        for (int i = 0; i <= topIndex; i++) {
            cout << data[i];
            if (i < topIndex) cout << ", ";
        }
        cout << "]" << endl;
    }
};

int main() {
    setlocale(LC_ALL, "ru");

    cout << "=== Анализ сложности операций стека ===" << endl;
    cout << "push: O(1) амортизированно (иногда O(n) при расширении)"
<< endl;
    cout << "pop: O(1)" << endl;
    cout << "top: O(1)" << endl;
    cout << "isEmpty: O(1)" << endl;
    cout << "size: O(1)" << endl;
    cout << "print: O(n)" << endl << endl;

    Stack<int> s(4);

    cout << "Добавление элементов:" << endl;
    for (int i = 1; i <= 10; i++) {
        s.push(i);
        cout << "push(" << i << "), размер: " << s.size() << endl;
    }

    cout << "\nСодержимое стека: ";
    s.print();
```

```
    cout << "\nУдаление элементов:" << endl;
    while (!s.isEmpty()) {
        cout << "pop() = " << s.pop() << ", размер: " << s.size() << endl;
    }

    return 0;
}
```

Индивидуальные задания

Базовый уровень (15 вариантов)

Общее задание: Для фрагментов кода (выдаётся преподавателем) определить сложность каждого фрагмента и обосновать ответ. Реализовать шаблонную функцию, указанную в варианте.

№ вар.	Шаблонная функция	Назначение
1	T max(T a, T b)	Возвращает максимальное из двух значений
2	T min(T a, T b)	Возвращает минимальное из двух значений
3	T abs(T x)	Возвращает абсолютное значение (для числовых типов)
4	void swap(T& a, T& b)	Обменивает значения двух переменных
5	T sum(T arr[], int n)	Возвращает сумму элементов массива
6	T average(T arr[], int n)	Возвращает среднее арифметическое
7	T findMax(T arr[], int n)	Возвращает максимальный элемент

№ вар.	Шаблонная функция	Назначение
8	T findMin(T arr[], int n)	Возвращает минимальный элемент
9	int linearSearch(T arr[], int n, T key)	Линейный поиск элемента
10	bool isEqual(T a, T b)	Сравнивает два значения на равенство
11	T product(T arr[], int n)	Возвращает произведение элементов
12	int countIf(T arr[], int n, T value)	Подсчитывает количество вхождений
13	void fill(T arr[], int n, T value)	Заполняет массив значением
14	T getOrDefault(T arr[], int n, int index, T defaultValue)	Возвращает элемент или значение по умолчанию
15	bool allEqual(T arr[], int n)	Проверяет, все ли элементы равны

Средний уровень (15 вариантов)

Общее задание: Реализовать шаблонный класс, указанный в варианте. Для каждой операции указать сложность. Провести экспериментальное сравнение производительности двух реализаций.

№ вар.	Класс	Операции	Сравнение
1	Pair<T1, T2>	Конструктор, getFirst, getSecond, setFirst, setSecond	—
2	Triple<T1, T2, T3>	Конструктор, геттеры, сеттеры	—

№ вар.	Класс	Операции	Сравнение
3	ArrayWrapper<T>	Конструктор, get, set, getSize	Линейный vs бинарный поиск
4	Counter<T>	add, getCount, reset, getMostFrequent	—
5	Range<T>	Конструктор, contains, getStart, getEnd	—
6	Nullable<T>	Конструктор, hasValue, getValue, reset	—
7	CircularBuffer<T>	push, pop, isEmpty, isFull	—
8	Accumulator<T>	add, getSum, getCount, getAverage	—
9	Filter<T>	Конструктор, apply, addCondition	Линейный vs бинарный поиск
10	Mapper<Tin, Tout>	Конструктор, apply, addMapping	—
11	Cache<T>	get, put, contains, clear	—
12	History<T>	push, undo, redo, clear	—
13	Observable<T>	setValue, getValue, addObserver	—
14	Lazy<T>	get, isInitialized, reset	—
15	Pool<T>	acquire, release, getAvailable	—

Продвинутый уровень (15 вариантов)

Общее задание: Реализовать шаблонный контейнер с заданной функциональностью. Провести анализ сложности каждой операции. Экспериментально сравнить производительность с аналогом из STL.

№ вар.	Контейнер	Требования
1	Vector<T>	push_back, pop_back, operator[], size, capacity, reserve
2	List<T>	push_front, push_back, pop_front, pop_back, size, insert, erase
3	Stack<T>	push, pop, top, isEmpty, size
4	Queue<T>	enqueue, dequeue, front, isEmpty, size
5	Deque<T>	push_front, push_back, pop_front, pop_back, operator[]
6	PriorityQueue<T>	push, pop, top, isEmpty, size (на основе кучи)
7	Set<T>	insert, contains, remove, size (на основе BST)
8	HashMap<K, V>	put, get, containsKey, remove, size
9	Array<T>	operator[], fill, sort, find (с динамическим расширением)
10	SortedList<T>	insert, contains, remove, getByIndex (сохраняет порядок)
11	RingBuffer<T>	push, pop, operator[], isEmpty, isFull (циклический буфер)
12	SparseArray<T>	set, get, remove, size (разрежённый массив)
13	StaticVector<T,	push_back, pop_back,

№ вар.	Контейнер	Требования
	N>	operator[], size (фиксированная ёмкость)
14	FixedQueue<T, N>	enqueue, dequeue, front, isEmpty, isFull
15	BitSet	set, clear, test, count, operators &, |, ^

Контрольные вопросы

1. Что такое O-нотация (Big O) и зачем она нужна?
2. Какая сложность у доступа к элементу массива по индексу? Почему?
3. Какая сложность у бинарного поиска? Почему?
4. Что означает сложность O(n log n)? Приведите пример алгоритма.
5. Что такое амортизированный анализ? Приведите пример.
6. Какая сложность у двух вложенных циклов по n итераций каждый?
7. Какая сложность у рекурсивного вычисления чисел Фибоначчи?
8. Что такое шаблоны функций? Как их объявить и использовать?
9. Что такое шаблоны классов? Как их объявить и использовать?
10. В чём преимущество шаблонов перед перегрузкой функций?
11. Как указать, что шаблонный параметр - тип?
12. Можно ли создать шаблон с несколькими параметрами типов?
13. Как провести экспериментальное измерение времени выполнения?
14. Почему при измерении времени нужно выполнять операцию многократно?
15. Что важнее: сложность алгоритма или константа?

Требования к отчёту

Титульный лист с названием работы, ФИО студента, номером группы, вариантом

Цель работы (формулируется самостоятельно на основе предложенной)

Теоретическая часть (основные понятия: O-нотация, классы сложности, шаблоны)

Постановка задачи (текст задания согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода программы, результаты измерений)

Анализ сложности (для каждой реализованной операции указать сложность)

Ответы на контрольные вопросы (письменно, развёрнуто)

Выводы (что нового узнали, какие навыки приобрели)

Ссылка на Git-репозиторий с кодом программы

Критерии оценивания

Базовый уровень (максимум 60 баллов)

Критерий	Баллы
Правильное определение сложности фрагментов кода	20
Реализация шаблонной функции согласно варианту	15
Демонстрация работы с разными типами данных	10
Код отформатирован, есть комментарии	10
Есть ответы на контрольные вопросы	5

Средний уровень (максимум 80 баллов)

Критерий	Баллы
Все требования базового уровня	40
Реализация шаблонного класса согласно варианту	15
Указание сложности для каждой операции класса	10
Экспериментальное сравнение двух реализаций	15

Продвинутый уровень (максимум 100 баллов)

Критерий	Баллы
Все требования среднего уровня	60
Реализация шаблонного контейнера с полной функциональностью	15
Анализ сложности всех операций	10

Критерий	Баллы
Сравнение с STL-аналогом (таблица, график)	10
Наличие ссылки на Git-репозиторий	5

Штрафные баллы

Нарушение	Штраф
Неправильное определение сложности	-10
Ошибки в шаблонном синтаксисе	-10
Отсутствуют комментарии в коде	-5
Нет ответов на контрольные вопросы	-10
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка (традиционная)	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №10

Реализация динамического массива (аналог vector). Итераторы

Цель работы:

1. Реализовать шаблонный класс Vector<T> - аналог std::vector - с нуля
2. Освоить управление динамической памятью: выделение, расширение, освобождение
3. Закрепить «правило трёх» (деструктор, конструктор копирования, оператор присваивания) и освоить «правило пяти» (конструктор и оператор перемещения)
4. Реализовать итераторы для поддержки range-based for и STL-совместимости
5. Провести амортизированный анализ операции push_back

Теоретическое обоснование

1. Что такое динамический массив?

Динамический массив - это структура данных, которая хранит элементы в непрерывной области памяти, но в отличие от статического массива может изменять свой размер во время выполнения программы.

Характеристика	**Статический массив**	**Динамический массив (vector)**
Размер	Фиксированный (известен на этапе компиляции)	Изменяемый (определяется во время выполнения)
Память	Стек (ограничен)	Куча (heap) - практически неограничена
Расширение	Невозможно	Автоматическое (при переполнении)
Вставка в конец	Невозможно при заполнении	O(1) амортизированно

2. Основные компоненты класса Vector

```
template <typename T>
class Vector {
private:
    T* data;      // указатель на динамический массив
    int sz;       // текущий размер (количество элементов)
    int cap;      // ёмкость (выделенная память)

public:
    // Конструкторы
    Vector();                 // по умолчанию
    explicit Vector(int n);      // размера n
    Vector(int n, const T& val); // размера n, заполненный val
    Vector(const Vector& other); // копирования
    Vector(Vector&& other);      // перемещения

    // Деструктор
    ~Vector();

    // Операторы
    Vector& operator=(const Vector& other);      // присваивание копированием
    Vector& operator=(Vector&& other);           // присваивание перемещением
    T& operator[](int index);             // доступ по индексу
    const T& operator[](int index) const;    // константный доступ

    // Методы доступа
```

```
    int size() const;
    int capacity() const;
    bool empty() const;

    // Методы изменения
    void push_back(const T& value);
    void pop_back();
    void reserve(int newCap);
    void resize(int newSize, const T& defaultValue = T());
    void clear();

    // Итераторы
    T* begin();
    T* end();
    const T* begin() const;
    const T* end() const;
};
```

3. Управление памятью

Выделение и освобождение

```
// Выделение памяти
data = new T[capacity];

// Освобождение памяти
delete[] data;
```

Расширение массива (стратегия удвоения)

```
void reserve(int newCap) {
    if (newCap <= cap) return;

    T* newData = new T[newCap];
    for (int i = 0; i < sz; i++) {
        newData[i] = data[i];
    }
    delete[] data;
    data = newData;
    cap = newCap;
}
```

Амортизированный анализ push_back

Стратегия удвоения ёмкости обеспечивает амортизированную сложность O(1):

Действие	Количество копирований
Вставка 1	0
Вставка 2 (расширение 1→2)	1
Вставка 3	0
Вставка 4 (расширение 2→4)	2
Вставки 5-7	0
Вставка 8 (расширение 4→8)	4

Итог: при n вставках выполняется около 2n копирований → O(n) на n операций → O(1) амортизированно на одну вставку.

4. Правило трёх и правило пяти

Правило	Методы
Правило трёх	Деструктор, конструктор копирования, оператор присваивания копированием
Правило пяти	+ Конструктор перемещения, оператор присваивания перемещением

Конструктор копирования (глубокое копирование)

```
Vector(const Vector& other) : sz(other.sz), cap(other.cap) {
    data = new T[cap];
    for (int i = 0; i < sz; i++) {
        data[i] = other.data[i];
    }
}
```

Оператор присваивания (с защитой от самоприсваивания)

```
Vector& operator=(const Vector& other) {
    if (this != &other) {
        Vector temp(other);  // copy-and-swap idiom
        swap(temp);
    }
    return *this;
}
```

Конструктор перемещения

```
Vector(Vector&& other) noexcept
    : data(other.data), sz(other.sz), cap(other.cap) {
    other.data = nullptr;
    other.sz = 0;
    other.cap = 0;
}
```

5. Итераторы

Итератор - объект, который указывает на элемент контейнера и позволяет перебирать элементы последовательно.

```
// Простая реализация итератора (указатель)
T* begin() { return data; }
T* end() { return data + sz; }

const T* begin() const { return data; }
const T* end() const { return data + sz; }

// Использование
Vector<int> v;
for (int x : v) {          // range-based for
    cout << x << " ";
}

for (auto it = v.begin(); it != v.end(); ++it) {
    cout << *it << " ";
}
```

6. Полный пример: реализация Vector

```
#include <iostream>
#include <stdexcept>

template <typename T>
class Vector {
private:
    T* data;
    int sz;
    int cap;

public:
    // Конструктор по умолчанию
    Vector() : data(nullptr), sz(0), cap(0) {}

    // Конструктор размера
    explicit Vector(int n) : sz(n), cap(n) {
        data = new T[cap];
        for (int i = 0; i < sz; i++) {
            data[i] = T();
        }
    }

    // Конструктор копирования
    Vector(const Vector& other) : sz(other.sz), cap(other.cap) {
        data = new T[cap];
        for (int i = 0; i < sz; i++) {
            data[i] = other.data[i];
        }
```

```
}

// Конструктор перемещения
Vector(Vector&& other) noexcept
    : data(other.data), sz(other.sz), cap(other.cap) {
    other.data = nullptr;
    other.sz = 0;
    other.cap = 0;
}

// Деструктор
~Vector() {
    delete[] data;
}

// Оператор присваивания копированием
Vector& operator=(const Vector& other) {
    if (this != &other) {
        Vector temp(other);
        swap(temp);
    }
    return *this;
}

// Оператор присваивания перемещением
Vector& operator=(Vector&& other) noexcept {
    if (this != &other) {
        delete[] data;
        data = other.data;
```

```
        sz = other.sz;
        cap = other.cap;
        other.data = nullptr;
        other.sz = 0;
        other.cap = 0;
    }
    return *this;
}

// Доступ по индексу
T& operator[](int index) {
    if (index < 0 || index >= sz) {
        throw std::out_of_range("Index out of range");
    }
    return data[index];
}

const T& operator[](int index) const {
    if (index < 0 || index >= sz) {
        throw std::out_of_range("Index out of range");
    }
    return data[index];
}

// Методы доступа
int size() const { return sz; }
int capacity() const { return cap; }
bool empty() const { return sz == 0; }
```

```
// Резервирование памяти
void reserve(int newCap) {
    if (newCap <= cap) return;

    T* newData = new T[newCap];
    for (int i = 0; i < sz; i++) {
        newData[i] = data[i];
    }
    delete[] data;
    data = newData;
    cap = newCap;
}

// Добавление в конец
void push_back(const T& value) {
    if (sz >= cap) {
        int newCap = (cap == 0) ? 1 : cap * 2;
        reserve(newCap);
    }
    data[sz++] = value;
}

// Удаление последнего элемента
void pop_back() {
    if (empty()) {
        throw std::underflow_error("Vector is empty");
    }
    sz--;
}
```

```
// Изменение размера
void resize(int newSize, const T& defaultValue = T()) {
    if (newSize < sz) {
        sz = newSize;
    } else if (newSize > sz) {
        if (newSize > cap) {
            reserve(newSize);
        }
        for (int i = sz; i < newSize; i++) {
            data[i] = defaultValue;
        }
        sz = newSize;
    }
}

// Очистка
void clear() {
    sz = 0;
}

// Обмен содержимого
void swap(Vector& other) {
    std::swap(data, other.data);
    std::swap(sz, other.sz);
    std::swap(cap, other.cap);
}

// Итераторы
```

```
    T* begin() { return data; }
    T* end() { return data + sz; }
    const T* begin() const { return data; }
    const T* end() const { return data + sz; }
};

// Демонстрация работы
int main() {
    setlocale(LC_ALL, "ru");

    Vector<int> v;

    cout << "=== Демонстрация Vector ===" << endl;
    cout << "Начальный размер: " << v.size() << ", ёмкость: " << v.capacity() << endl;

    for (int i = 1; i <= 10; i++) {
        v.push_back(i);
        cout << "push_back(" << i << "): размер=" << v.size()
             << ", ёмкость=" << v.capacity() << endl;
    }

    cout << "\nСодержимое через range-based for: ";
    for (int x : v) {
        cout << x << " ";
    }
    cout << endl;

    cout << "\nДоступ по индексу: v[5] = " << v[5] << endl;
```

```
    v.pop_back();
    cout << "\nПосле pop_back: размер=" << v.size() << endl;

    v.resize(15, 0);
    cout << "После resize(15, 0): размер=" << v.size() << endl;

    return 0;
}
```

Примеры выполнения практической части

Пример 1. Базовые операции Vector (базовый уровень)

Задача: Реализовать класс Vector с основными методами: конструкторы, push_back, pop_back, operator[], size.

```
#include <iostream>

template <typename T>
class Vector {
private:
    T* data;
    int sz;
    int cap;

public:
    Vector() : data(nullptr), sz(0), cap(0) {}

    explicit Vector(int n) : sz(n), cap(n) {
        data = new T[cap];
```

```
    for (int i = 0; i < sz; i++) data[i] = T();
}

~Vector() { delete[] data; }

void push_back(const T& value) {
    if (sz >= cap) {
        int newCap = (cap == 0) ? 1 : cap * 2;
        T* newData = new T[newCap];
        for (int i = 0; i < sz; i++) newData[i] = data[i];
        delete[] data;
        data = newData;
        cap = newCap;
    }
    data[sz++] = value;
}

void pop_back() { if (sz > 0) sz--; }

int size() const { return sz; }
int capacity() const { return cap; }

T& operator[](int index) { return data[index]; }
const T& operator[](int index) const { return data[index]; }

void print() const {
    std::cout << "[";
    for (int i = 0; i < sz; i++) {
        std::cout << data[i];
```

```
            if (i < sz - 1) std::cout << ", ";
        }
        std::cout << "]";
    }
};

int main() {
    Vector<int> v;

    for (int i = 1; i <= 5; i++) {
        v.push_back(i * 10);
    }

    std::cout << "Размер: " << v.size() << ", Ёмкость: " << v.capacity() <<
std::endl;
    std::cout << "Элементы: ";
    v.print();
    std::cout << std::endl;

    v.pop_back();
    std::cout << "После pop_back: ";
    v.print();
    std::cout << std::endl;

    return 0;
}
```

Пример 2. Итераторы (средний уровень)

Задача: Добавить в класс Vector итераторы и продемонстрировать их использование.

```
#include <iostream>
#include <algorithm>

template <typename T>
class Vector {
private:
    T* data;
    int sz;
    int cap;

public:
    // ... остальные методы ...

    // Итераторы
    T* begin() { return data; }
    T* end() { return data + sz; }
    const T* begin() const { return data; }
    const T* end() const { return data + sz; }

    // Вставка по итератору
    void insert(T* pos, const T& value) {
        int index = pos - data;
        if (index < 0 || index > sz) return;

        if (sz >= cap) {
            int newCap = (cap == 0) ? 1 : cap * 2;
            T* newData = new T[newCap];
```

```
            for (int i = 0; i < index; i++) newData[i] = data[i];
            newData[index] = value;
            for (int i = index; i < sz; i++) newData[i + 1] = data[i];
            delete[] data;
            data = newData;
            cap = newCap;
        } else {
            for (int i = sz; i > index; i--) data[i] = data[i - 1];
            data[index] = value;
        }
        sz++;
    }

    // Удаление по итератору
    void erase(T* pos) {
        int index = pos - data;
        if (index < 0 || index >= sz) return;

        for (int i = index; i < sz - 1; i++) data[i] = data[i + 1];
        sz--;
    }
};

int main() {
    setlocale(LC_ALL, "ru");

    Vector<int> v;

    for (int i = 1; i <= 5; i++) {
```

```
    v.push_back(i);
}

std::cout << "Исходный вектор: ";
for (int x : v) std::cout << x << " ";
std::cout << std::endl;

// Вставка в середину
auto it = v.begin() + 2;
v.insert(it, 99);

std::cout << "После вставки 99 на позицию 2: ";
for (int x : v) std::cout << x << " ";
std::cout << std::endl;

// Удаление
it = v.begin() + 3;
v.erase(it);

std::cout << "После удаления элемента на позиции 3: ";
for (int x : v) std::cout << x << " ";
std::cout << std::endl;

// Сортировка с использованием STL алгоритма
std::sort(v.begin(), v.end());
std::cout << "После сортировки: ";
for (int x : v) std::cout << x << " ";
std::cout << std::endl;
```

```
    return 0;
}
```

Пример 3. Амортизированный анализ (продвинутый уровень)

Задача: Реализовать класс Vector с возможностью отслеживания количества операций и построить график зависимости времени от размера.

```
#include <iostream>
#include <chrono>
#include <vector>
#include <iomanip>

template <typename T>
class Vector {
private:
    T* data;
    int sz;
    int cap;
    int copyCount;  // счётчик копирований

public:
    Vector() : data(nullptr), sz(0), cap(0), copyCount(0) {}

    ~Vector() { delete[] data; }

    void push_back(const T& value) {
        if (sz >= cap) {
            int newCap = (cap == 0) ? 1 : cap * 2;
            T* newData = new T[newCap];
            for (int i = 0; i < sz; i++) {
```

```
            newData[i] = data[i];
            copyCount++;
        }
        delete[] data;
        data = newData;
        cap = newCap;
    }
    data[sz++] = value;
}

int getCopyCount() const { return copyCount; }
int size() const { return sz; }
int capacity() const { return cap; }

void resetCopyCount() { copyCount = 0; }
};

int main() {
    setlocale(LC_ALL, "ru");

    std::cout << "=== Амортизированный анализ push_back ===" << std::endl;
    std::cout << std::setw(10) << "n"
              << std::setw(15) << "Копирований"
              << std::setw(15) << "Всего операций"
              << std::setw(15) << "Среднее" << std::endl;
    std::cout << std::string(55, '-') << std::endl;

    for (int n = 1000; n <= 100000; n *= 10) {
```

```
        Vector<int> v;

        for (int i = 0; i < n; i++) {
            v.push_back(i);
        }

        int totalOps = n + v.getCopyCount();
        double average = static_cast<double>(totalOps) / n;

        std::cout << std::setw(10) << n
                  << std::setw(15) << v.getCopyCount()
                  << std::setw(15) << totalOps
                  << std::setw(15) << std::fixed << std::setprecision(2) << average
                  << std::endl;
    }

    return 0;
}
```

Ожидаемый вывод:

```
=== Амортизированный анализ push_back ===
         n     Копирований   Всего операций        Среднее
-----------------------------------------------------
      1000          1023            2023           2.02
     10000         10239           20239           2.02
    100000        131071          131071           1.31
```

Индивидуальные задания

Базовый уровень (15 вариантов)

Общее задание: Реализовать класс Vector с минимальным набором методов согласно варианту.

№ вар.	Конструкторы	Обязательные методы
1	По умолчанию, с параметром (размер)	push_back, pop_back, size, capacity
2	По умолчанию, с параметром (размер)	push_back, pop_back, empty, clear
3	По умолчанию, с параметром (размер, значение)	push_back, size, operator[], print
4	По умолчанию, копирования	push_back, pop_back, size, reserve
5	По умолчанию, с параметром (размер)	push_back, insert, erase, size
6	По умолчанию, с параметром (размер, значение)	push_back, pop_back, find, print
7	По умолчанию, с параметром (размер)	push_back, pop_back, resize, capacity
8	По умолчанию, копирования	push_back, operator[], size, clear

№ вар.	Конструкторы	Обязательные методы
9	По умолчанию, с параметром (размер)	push_back, pop_back, front, back
10	По умолчанию, с параметром (размер, значение)	push_back, size, empty, reserve
11	По умолчанию, с параметром (размер)	push_back, pop_back, insert, print
12	По умолчанию, копирования	push_back, erase, size, capacity
13	По умолчанию, с параметром (размер)	push_back, pop_back, resize, print
14	По умолчанию, с параметром (размер, значение)	push_back, size, operator[], clear
15	По умолчанию, с параметром (размер)	push_back, pop_back, front, back, size

Средний уровень (15 вариантов)

Общее задание: Реализовать класс Vector с полной поддержкой «правила пяти» и дополнительной функциональностью согласно варианту.

№ вар.	Дополнительная функциональность
1	Методы reserve, shrink_to_fit
2	Метод reverse() - разворот массива
3	Методы sum, average (для числовых типов)
4	Методы min, max
5	Методы find (поиск по значению), contains
6	Метод removeAll (удаление всех вхождений значения)
7	Метод slice(int start, int end) - создание подвектора
8	Метод concat(const Vector& other) - объединение векторов
9	Метод swap (обмен содержимым)
10	Метод unique - удаление дубликатов
11	Метод rotate(int k) - циклический сдвиг
12	Метод shuffle - случайное перемешивание
13	Метод sort - сортировка пузырьком
14	Метод merge - слияние двух отсортированных векторов

№ вар.	Дополнительная функциональность
15	Метод isSorted - проверка на упорядоченность

Продвинутый уровень (15 вариантов)

Общее задание: Реализовать класс Vector с поддержкой «правила пяти», итераторами и дополнительной функциональностью согласно варианту.

№ вар.	**Дополнительная функциональность**
1	Итераторы с категориями (input, output, forward, bidirectional, random access)
2	Реализация const_iterator
3	Реализация reverse_iterator и методов rbegin, rend
4	Операторы сравнения векторов (==, !=, <, >)
5	Операторы + (конкатенация) и * (умножение на число)
6	Метод apply(function) - применение функции ко всем элементам
7	Метод filter(predicate) - фильтрация элементов
8	Сериализация: saveToFile и loadFromFile
9	Поддержка пользовательских аллокаторов

№ вар.	Дополнительная функциональность
10	Метод partition(pivot) - разделение на две части
11	Реализация быстрой сортировки внутри класса
12	Ленивое копирование (copy-on-write)
13	Реализация std::initializer_list для удобной инициализации
14	Реализация emplace_back (вариативный шаблон)
15	Реализация оператора << для вывода вектора

Контрольные вопросы

1. Почему динамический массив называется «динамическим»?
2. Что такое ёмкость (capacity) и чем она отличается от размера (size)?
3. Почему при расширении массива ёмкость удваивается, а не увеличивается на фиксированное число?
4. Какова амортизированная сложность операции push_back? Почему?
5. Что такое «правило трёх»? Когда оно применяется?
6. Что такое «правило пяти»? Чем оно отличается от «правила трёх»?
7. В чём разница между конструктором копирования и оператором присваивания?
8. Зачем нужна проверка if (this != &other) в операторе присваивания?
9. Что такое итератор? Как он связан с указателем?
10. Зачем нужны методы begin() и end()?
11. Как реализовать range-based for для своего контейнера?

12. Что такое «глубокое копирование»? Чем оно отличается от «поверхностного»?
13. Что такое семантика перемещения и зачем она нужна?
14. Почему конструктор перемещения помечается noexcept?
15. Как правильно освобождать память в деструкторе?

Требования к отчёту

Титульный лист с названием работы, ФИО студента, номером группы, вариантом

Цель работы (формулируется самостоятельно на основе предложенной)

Теоретическая часть (основные понятия: динамическая память, правило трёх/пяти, итераторы, амортизированный анализ)

Постановка задачи (текст задания согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода программы)

Анализ сложности (для каждой операции указать сложность)

Ответы на контрольные вопросы (письменно, развёрнуто)

Выводы (что нового узнали, какие навыки приобрели)

Ссылка на Git-репозиторий с кодом программы

Критерии оценивания

Базовый уровень (максимум 60 баллов)

Критерий	Баллы
Класс Vector реализован корректно, поля приватные	10
Реализованы конструкторы (по умолчанию, с параметром)	10

Критерий	Баллы
Реализован деструктор, освобождает память	10
Реализованы указанные в варианте методы	15
Демонстрационная программа показывает работу всех методов	10
Код отформатирован, есть комментарии	5

Средний уровень (максимум 80 баллов)

Критерий	Баллы
Все требования базового уровня	40
Реализовано «правило пяти»	15
Реализована дополнительная функциональность по варианту	15
Корректная обработка ошибок (выход за границы)	5
Наличие проверки на самоприсваивание	5

Продвинутый уровень (максимум 100 баллов)

Критерий	Баллы
Все требования среднего уровня	60
Реализация итераторов (begin, end)	10
Реализация продвинутой функциональности по	15

Критерий	Баллы
варианту	
Исключения и безопасность (RAII)	10
Наличие ссылки на Git-репозиторий	5

Штрафные баллы

Нарушение	Штраф
Утечка памяти (не освобождена динамическая память)	-20
Двойное освобождение памяти	-15
Отсутствие проверки на самоприсваивание	-5
Использование публичных полей вместо геттеров	-10
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка (традиционная)	Баллы
Отлично	90–100
Хорошо	75–89

Оценка (традиционная)	Баллы
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №11

Односвязный и двусвязный списки

Цель работы:

1. Реализовать шаблонный класс односвязного списка (SinglyLinkedList) с базовыми операциями
2. Реализовать шаблонный класс двусвязного списка (DoublyLinkedList) с базовыми операциями
3. Освоить операции вставки, удаления, поиска и обхода элементов в списках
4. Сравнить производительность и особенности использования односвязного и двусвязного списков
5. Научиться правильно управлять памятью при работе со списками (удаление узлов)

Теоретическое обоснование

1. Понятие связного списка

Связный список - это структура данных, в которой элементы (узлы) не хранятся в памяти непрерывно. Каждый узел содержит данные и указатель(и) на следующий (и/или предыдущий) узел.

Визуализация:

Односвязный список:

head ──► [10] ──► [20] ──► [30] ──► nullptr

Двусвязный список:

head ──► [10] ⇄ [20] ⇄ [30] ⇄ nullptr

◄──────► ◄──────►

2. Сравнение массивов и списков

Операция	**Динамический массив (vector)**	**Односвязный список**	**Двусвязный список**
Доступ по индексу	O(1)	O(n)	O(n)
Вставка в начало	O(n)	O(1)	O(1)
Вставка в конец	O(1) амортиз.	O(1) с tail / O(n)	O(1) с tail
Вставка в середину	O(n)	O(n) поиск + O(1)	O(n) поиск + O(1)
Удаление из начала	O(n)	O(1)	O(1)
Удаление из конца	O(1)	O(n)	O(1)
Поиск по значению	O(n)	O(n)	O(n)

Операция	Динамический массив (vector)	Односвязный список	Двусвязный список
Память на элемент	Только данные	Данные + указатель	Данные + 2 указателя

3. Узел односвязного списка

```
template <typename T>
struct Node {
    T data;
    Node<T>* next;

    Node(const T& value) : data(value), next(nullptr) {}
};
```

4. Узел двусвязного списка

```
template <typename T>
struct DNode {
    T data;
    DNode<T>* prev;
    DNode<T>* next;

    DNode(const T& value) : data(value), prev(nullptr), next(nullptr) {}
};
```

5. Базовые операции с односвязным списком

```
template <typename T>
```

```
class SinglyLinkedList {
private:
    Node<T>* head;
    Node<T>* tail;
    int count;

public:
    // Конструкторы и деструктор
    SinglyLinkedList() : head(nullptr), tail(nullptr), count(0) {}

    ~SinglyLinkedList() {
        clear();
    }

    // Вставка в начало O(1)
    void push_front(const T& value) {
        Node<T>* newNode = new Node<T>(value);
        newNode->next = head;
        head = newNode;
        if (tail == nullptr) tail = newNode;
        count++;
    }

    // Вставка в конец O(1) с tail
    void push_back(const T& value) {
        Node<T>* newNode = new Node<T>(value);
        if (head == nullptr) {
            head = tail = newNode;
        } else {
```

```
        tail->next = newNode;
        tail = newNode;
    }
    count++;
}

// Удаление из начала O(1)
void pop_front() {
    if (head == nullptr) throw std::underflow_error("List is empty");
    Node<T>* temp = head;
    head = head->next;
    delete temp;
    count--;
    if (head == nullptr) tail = nullptr;
}

// Удаление из конца O(n)
void pop_back() {
    if (head == nullptr) throw std::underflow_error("List is empty");
    if (head->next == nullptr) {
        delete head;
        head = tail = nullptr;
    } else {
        Node<T>* current = head;
        while (current->next != tail) current = current->next;
        delete tail;
        tail = current;
        tail->next = nullptr;
    }
```

```
    count--;
}

// Поиск O(n)
Node<T>* find(const T& value) const {
    Node<T>* current = head;
    while (current != nullptr) {
        if (current->data == value) return current;
        current = current->next;
    }
    return nullptr;
}

// Удаление по значению O(n)
bool remove(const T& value) {
    if (head == nullptr) return false;
    if (head->data == value) {
        pop_front();
        return true;
    }
    Node<T>* current = head;
    while (current->next != nullptr && current->next->data != value) {
        current = current->next;
    }
    if (current->next == nullptr) return false;
    Node<T>* toDelete = current->next;
    current->next = toDelete->next;
    if (toDelete == tail) tail = current;
    delete toDelete;
```

```
        count--;
        return true;
    }

    // Вывод списка O(n)
    void print() const {
        Node<T>* current = head;
        std::cout << "[";
        while (current != nullptr) {
            std::cout << current->data;
            if (current->next != nullptr) std::cout << ", ";
            current = current->next;
        }
        std::cout << "]" << std::endl;
    }

    void clear() {
        while (head != nullptr) {
            Node<T>* temp = head;
            head = head->next;
            delete temp;
        }
        tail = nullptr;
        count = 0;
    }

    int size() const { return count; }
    bool empty() const { return head == nullptr; }
};
```

6. Базовые операции с двусвязным списком

```
template <typename T>
class DoublyLinkedList {
private:
    DNode<T>* head;
    DNode<T>* tail;
    int count;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr), count(0) {}

    ~DoublyLinkedList() {
        clear();
    }

    // Вставка в начало O(1)
    void push_front(const T& value) {
        DNode<T>* newNode = new DNode<T>(value);
        if (head == nullptr) {
            head = tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
        count++;
    }
```

```
// Вставка в конец O(1)
void push_back(const T& value) {
    DNode<T>* newNode = new DNode<T>(value);
    if (tail == nullptr) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode;
    }
    count++;
}

// Удаление из начала O(1)
void pop_front() {
    if (head == nullptr) throw std::underflow_error("List is empty");
    DNode<T>* temp = head;
    head = head->next;
    if (head != nullptr) head->prev = nullptr;
    else tail = nullptr;
    delete temp;
    count--;
}

// Удаление из конца O(1)
void pop_back() {
    if (tail == nullptr) throw std::underflow_error("List is empty");
    DNode<T>* temp = tail;
    tail = tail->prev;
```

```
    if (tail != nullptr) tail->next = nullptr;
    else head = nullptr;
    delete temp;
    count--;
}

// Поиск O(n)
DNode<T>* find(const T& value) const {
    DNode<T>* current = head;
    while (current != nullptr) {
        if (current->data == value) return current;
        current = current->next;
    }
    return nullptr;
}

// Вставка после указанного узла O(1)
void insert_after(DNode<T>* node, const T& value) {
    if (node == nullptr) {
        push_front(value);
        return;
    }
    DNode<T>* newNode = new DNode<T>(value);
    newNode->next = node->next;
    newNode->prev = node;
    if (node->next != nullptr) node->next->prev = newNode;
    else tail = newNode;
    node->next = newNode;
    count++;
```

```
}

// Удаление указанного узла O(1)
void erase(DNode<T>* node) {
    if (node == nullptr) return;
    if (node == head) {
        pop_front();
        return;
    }
    if (node == tail) {
        pop_back();
        return;
    }
    node->prev->next = node->next;
    node->next->prev = node->prev;
    delete node;
    count--;
}

// Обход в обратном порядке
void printReverse() const {
    DNode<T>* current = tail;
    std::cout << "[";
    while (current != nullptr) {
        std::cout << current->data;
        if (current->prev != nullptr) std::cout << ", ";
        current = current->prev;
    }
    std::cout << "]" << std::endl;
```

```
    }

    void print() const {
        DNode<T>* current = head;
        std::cout << "[";
        while (current != nullptr) {
            std::cout << current->data;
            if (current->next != nullptr) std::cout << ", ";
            current = current->next;
        }
        std::cout << "]" << std::endl;
    }

    void clear() {
        while (head != nullptr) {
            DNode<T>* temp = head;
            head = head->next;
            delete temp;
        }
        tail = nullptr;
        count = 0;
    }

    int size() const { return count; }
    bool empty() const { return head == nullptr; }
};
```

Примеры выполнения практической части

Пример 1. Односвязный список (базовый уровень)

Задача: Реализовать класс односвязного списка с операциями push_front, push_back, pop_front, print.

```
#include <iostream>
#include <stdexcept>

template <typename T>
class SinglyLinkedList {
private:
    struct Node {
        T data;
        Node* next;
        Node(const T& value) : data(value), next(nullptr) {}
    };

    Node* head;
    Node* tail;
    int count;

public:
    SinglyLinkedList() : head(nullptr), tail(nullptr), count(0) {}

    ~SinglyLinkedList() { clear(); }

    void push_front(const T& value) {
        Node* newNode = new Node(value);
        newNode->next = head;
        head = newNode;
        if (tail == nullptr) tail = newNode;
```

```
        count++;
    }

    void push_back(const T& value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
        count++;
    }

    void pop_front() {
        if (head == nullptr) throw std::underflow_error("List is empty");
        Node* temp = head;
        head = head->next;
        delete temp;
        count--;
        if (head == nullptr) tail = nullptr;
    }

    void print() const {
        Node* current = head;
        std::cout << "[";
        while (current != nullptr) {
            std::cout << current->data;
            if (current->next != nullptr) std::cout << ", ";
```

```
            current = current->next;
        }
        std::cout << "]" << std::endl;
    }

    int size() const { return count; }
    bool empty() const { return head == nullptr; }

    void clear() {
        while (head != nullptr) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
        tail = nullptr;
        count = 0;
    }
};

int main() {
    setlocale(LC_ALL, "ru");

    SinglyLinkedList<int> list;

    std::cout << "=== Односвязный список ===" << std::endl;

    list.push_back(10);
    list.push_back(20);
    list.push_back(30);
```

```
    list.push_front(5);

    std::cout << "Список: ";
    list.print();
    std::cout << "Размер: " << list.size() << std::endl;

    list.pop_front();
    std::cout << "После pop_front: ";
    list.print();

    return 0;
}
```

Ожидаемый вывод:

=== Односвязный список ===

Список: [5, 10, 20, 30]

Размер: 4

После pop_front: [10, 20, 30]

Пример 2. Двусвязный список (средний уровень)

Задача: Реализовать класс двусвязного списка с операциями push_back, pop_back, printReverse.

```
#include <iostream>
#include <stdexcept>

template <typename T>
class DoublyLinkedList {
private:
    struct Node {
```

```
        T data;
        Node* prev;
        Node* next;
        Node(const T& value) : data(value), prev(nullptr), next(nullptr) {}
    };

    Node* head;
    Node* tail;
    int count;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr), count(0) {}

    ~DoublyLinkedList() { clear(); }

    void push_back(const T& value) {
        Node* newNode = new Node(value);
        if (tail == nullptr) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
        count++;
    }

    void pop_back() {
        if (tail == nullptr) throw std::underflow_error("List is empty");
```

```
    Node* temp = tail;
    tail = tail->prev;
    if (tail != nullptr) tail->next = nullptr;
    else head = nullptr;
    delete temp;
    count--;
}

void print() const {
    Node* current = head;
    std::cout << "[";
    while (current != nullptr) {
        std::cout << current->data;
        if (current->next != nullptr) std::cout << ", ";
        current = current->next;
    }
    std::cout << "]" << std::endl;
}

void printReverse() const {
    Node* current = tail;
    std::cout << "[";
    while (current != nullptr) {
        std::cout << current->data;
        if (current->prev != nullptr) std::cout << ", ";
        current = current->prev;
    }
    std::cout << "]" << std::endl;
}
```

```
    int size() const { return count; }

    void clear() {
        while (head != nullptr) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
        tail = nullptr;
        count = 0;
    }
};

int main() {
    setlocale(LC_ALL, "ru");

    DoublyLinkedList<int> list;

    for (int i = 1; i <= 5; i++) {
        list.push_back(i * 10);
    }

    std::cout << "=== Двусвязный список ===" << std::endl;
    std::cout << "Прямой порядок: ";
    list.print();
    std::cout << "Обратный порядок: ";
    list.printReverse();
```

```
    list pop_back();
    std::cout << "После pop_back: ";
    list print();

    return 0;
}
```

Ожидаемый вывод:

=== Двусвязный список ===

Прямой порядок: [10, 20, 30, 40, 50]

Обратный порядок: [50, 40, 30, 20, 10]

После pop_back: [10, 20, 30, 40]

Пример 3. Итераторы и реверс списка (продвинутый уровень)

Задача: Реализовать итераторы для двусвязного списка и метод реверса.

```
#include <iostream>
#include <stdexcept>

template <typename T>
class DoublyLinkedList {
private:
    struct Node {
        T data;
        Node* prev;
        Node* next;
        Node(const T& value) : data(value), prev(nullptr), next(nullptr) {}
    };

    Node* head;
    Node* tail;
```

```
    int count;

public:
    // Итератор
    class Iterator {
    private:
        Node* ptr;
    public:
        Iterator(Node* p = nullptr) : ptr(p) {}

        T& operator*() { return ptr->data; }
        Iterator& operator++() { ptr = ptr->next; return *this; }
        Iterator operator++(int) { Iterator temp = *this; ptr = ptr->next; return
temp; }
        Iterator& operator--() { ptr = ptr->prev; return *this; }
        Iterator operator--(int) { Iterator temp = *this; ptr = ptr->prev; return
temp; }
        bool operator==(const Iterator& other) const { return ptr == other.ptr; }
        bool operator!=(const Iterator& other) const { return ptr != other.ptr; }
    };

    DoublyLinkedList() : head(nullptr), tail(nullptr), count(0) {}

    ~DoublyLinkedList() { clear(); }

    void push_back(const T& value) {
        Node* newNode = new Node(value);
        if (tail == nullptr) {
            head = tail = newNode;
```

```
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
        count++;
    }

    // Реверс списка (меняем местами указатели next и prev)
    void reverse() {
        if (head == nullptr) return;

        Node* current = head;
        Node* temp = nullptr;

        while (current != nullptr) {
            // Меняем местами prev и next
            temp = current->prev;
            current->prev = current->next;
            current->next = temp;
            current = current->prev;  // переходим к следующему (бывшему prev)
        }

        // Меняем местами head и tail
        temp = head;
        head = tail;
        tail = temp;
    }
```

```
    Iterator begin() { return Iterator(head); }
    Iterator end() { return Iterator(nullptr); }

    void print() const {
        Node* current = head;
        std::cout << "[";
        while (current != nullptr) {
            std::cout << current->data;
            if (current->next != nullptr) std::cout << ", ";
            current = current->next;
        }
        std::cout << "]" << std::endl;
    }

    void clear() {
        while (head != nullptr) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
        tail = nullptr;
        count = 0;
    }

    int size() const { return count; }
};

int main() {
```

```
    setlocale(LC_ALL, "ru");

    DoublyLinkedList<int> list;

    for (int i = 1; i <= 5; i++) {
        list.push_back(i);
    }

    std::cout << "=== Итераторы и реверс ===" << std::endl;
    std::cout << "Исходный список: ";
    list.print();

    std::cout << "Обход с помощью итератора: ";
    for (auto it = list.begin(); it != list.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    list.reverse();
    std::cout << "После реверса: ";
    list.print();

    return 0;
}
```

Ожидаемый вывод:

=== Итераторы и реверс ===

Исходный список: [1, 2, 3, 4, 5]

Обход с помощью итератора: 1 2 3 4 5

После реверса: [5, 4, 3, 2, 1]

Индивидуальные задания

Базовый уровень (15 вариантов)

Общее задание: Реализовать класс односвязного списка с указанными методами.

№ вар.	Обязательные методы	Дополнительное задание
1	push_front, pop_front, print	Найти сумму всех элементов
2	push_back, pop_back, print	Найти количество элементов
3	push_front, push_back, print	Найти максимальный элемент
4	push_front, pop_front, find	Найти минимальный элемент
5	push_back, pop_back, find	Проверить, есть ли заданное значение
6	push_front, push_back, pop_front	Вывести элементы через один
7	push_front, push_back, pop_back	Вывести элементы в обратном порядке (без реверса)
8	push_front, find, remove	Удалить все чётные

№ вар.	Обязательные методы	Дополнительное задание
		элементы
9	push_back, find, remove	Удалить все нечётные элементы
10	push_front, push_back, clear	После очистки вывести размер
11	push_front, pop_front, size	Проверить, является ли список палиндромом
12	push_back, pop_back, empty	Скопировать список в другой
13	push_front, find, insert_after	Вставить элемент после заданного
14	push_back, find, insert_before	Вставить элемент перед заданным
15	push_front, push_back, getAt	Получить элемент по индексу (реализовать)

Средний уровень (15 вариантов)

Общее задание: Реализовать класс двусвязного списка с указанными методами и дополнительной функциональностью.

№ вар.	Дополнительная функциональность
1	Метод reverse() - разворот списка
2	Метод merge(DoublyLinkedList& other) - слияние двух отсортированных списков
3	Метод removeDuplicates() - удаление дубликатов
4	Метод getMiddle() - получение среднего элемента (без вычисления размера)
5	Метод hasCycle() - проверка на наличие цикла
6	Метод rotateLeft(int k) - циклический сдвиг влево на k позиций
7	Метод rotateRight(int k) - циклический сдвиг вправо на k позиций
8	Метод isPalindrome() - проверка, является ли список палиндромом
9	Метод intersection(DoublyLinkedList& other) - пересечение двух списков
10	Метод removeEverySecond() - удаление каждого второго элемента
11	Метод swapPairs() - обмена соседних элементов

№ вар.	Дополнительная функциональность
12	Метод split() - разделение списка на два: чётные и нечётные позиции
13	Метод sort() - сортировка списка (пузырьком или вставками)
14	Метод toVector() - преобразование в std::vector
15	Метод fromVector(const std::vector<T>&) - создание списка из вектора

Продвинутый уровень (15 вариантов)

Общее задание: Реализовать оба класса (SinglyLinkedList и DoublyLinkedList) с полной функциональностью и дополнительными требованиями.

№ вар.	**Дополнительные требования**
1	Реализовать итераторы для обоих списков (begin, end)
2	Реализовать операторы сравнения (==, !=) для списков
3	Реализовать оператор [] для доступа по индексу (с проверкой сложности)
4	Реализовать метод sort() (быстрая сортировка для

№ вар.	Дополнительные требования
	двусвязного списка)
5	Реализовать метод mergeSort() для односвязного списка
6	Реализовать сериализацию: saveToFile и loadFromFile
7	Реализовать исключения для всех операций (выход за границы, удаление из пустого)
8	Реализовать std::initializer_list для удобной инициализации
9	Реализовать const_iterator для обоих списков
10	Реализовать reverse_iterator для двусвязного списка
11	Сравнить производительность операций односвязного и двусвязного списков
12	Реализовать кольцевой (циклический) список на основе двусвязного
13	Реализовать список с пропусками (skip list) на основе односвязного
14	Реализовать аллокатор для узлов (pool allocator)

№ вар.	Дополнительные требования
15	Реализовать ленивое удаление (маркировка удалённых элементов)

Контрольные вопросы

1. Чем связный список отличается от динамического массива?
2. Какие операции в односвязном списке выполняются за O(1)?
3. Почему удаление из конца односвязного списка требует O(n) времени?
4. Какие преимущества даёт хранение указателя tail в односвязном списке?
5. Чем двусвязный список отличается от односвязного?
6. Какие операции в двусвязном списке выполняются быстрее, чем в односвязном?
7. Что такое «голова» (head) и «хвост» (tail) списка?
8. Как удалить узел из середины односвязного списка?
9. Как удалить узел из середины двусвязного списка? Почему это проще?
10. Что произойдёт, если забыть обновить tail при удалении?
11. Как реализовать реверс односвязного списка?
12. Как обнаружить цикл в связном списке?
13. В каких случаях стоит использовать список вместо массива?
14. Какая дополнительная память требуется для каждого узла списка?
15. Почему доступ по индексу в списке имеет сложность O(n)?

Требования к отчёту

Титульный лист с названием работы, ФИО студента, номером группы, вариантом

Цель работы (формулируется самостоятельно)

Теоретическая часть (основные понятия: связный список, узел, head, tail)

Постановка задачи (текст задания согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода программы)

Анализ сложности (для каждой операции указать сложность)

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Критерии оценивания

Базовый уровень (максимум 60 баллов)

Критерий	Баллы
Класс односвязного списка реализован корректно	20
Реализованы указанные методы	20
Корректное управление памятью (деструктор)	10
Демонстрационная программа показывает работу	5
Код отформатирован, есть комментарии	5

Средний уровень (максимум 80 баллов)

Критерий	Баллы
Все требования базового уровня	40

Критерий	Баллы
Класс двусвязного списка реализован корректно	15
Реализована дополнительная функциональность	15
Корректная работа с tail (для двусвязного)	10

Продвинутый уровень (максимум 100 баллов)

Критерий	Баллы
Все требования среднего уровня	60
Реализованы оба списка с полной функциональностью	15
Реализованы итераторы и операторы	15
Проведено сравнение производительности	5
Наличие ссылки на Git-репозиторий	5

Штрафные баллы

Нарушение	Штраф
Утечка памяти	-20
Неверная работа с указателями	-15
Отсутствие обработки пустого списка	-10
Нет ссылки на Git-репозиторий	-10

Нарушение	Штраф
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №12

Стек, очередь, дек, приоритетная очередь

Цель работы:

1. Реализовать основные линейные структуры данных: стек, очередь, дек и приоритетную очередь
2. Освоить принципы LIFO (стек) и FIFO (очередь)
3. Научиться реализовывать эти структуры на основе динамического массива и связного списка
4. Изучить применение стека для решения задач (проверка скобок, обратная польская запись)
5. Реализовать приоритетную очередь на основе кучи (heap)

Теоретическое обоснование

1. Стек (Stack)

Стек - это структура данных, работающая по принципу LIFO (Last In - First Out, «последним пришёл - первым ушел»).

Аналогия: Стопка тарелок - последнюю поставленную тарелку мы возьмём первой.

Основные операции:

Операция	Описание	Сложность
push(value)	Добавляет элемент на вершину стека	O(1)
pop()	Удаляет и возвращает элемент с вершины	O(1)
top()	Возвращает элемент с вершины без удаления	O(1)
isEmpty()	Проверяет, пуст ли стек	O(1)
size()	Возвращает количество элементов	O(1)

Визуализация стека:

push(1): [1]

push(2): [1, 2]

push(3): [1, 2, 3]

pop(): [1, 2] (возвращает 3)

top(): [1, 2] (возвращает 2, не удаляя)

2. Очередь (Queue)

Очередь - это структура данных, работающая по принципу FIFO (First In - First Out, «первым пришёл - первым ушел»).

Аналогия: Обычная очередь в магазине - первый в очереди обслуживается первым.

Основные операции:

Операция	Описание	Сложность
enqueue(value)	Добавляет элемент в конец очереди	O(1)
dequeue()	Удаляет и возвращает элемент из начала	O(1)
front()	Возвращает элемент из начала без удаления	O(1)
isEmpty()	Проверяет, пуста ли очередь	O(1)
size()	Возвращает количество элементов	O(1)

Визуализация очереди:

enqueue(1): [1]

enqueue(2): [1, 2]

enqueue(3): [1, 2, 3]

dequeue(): [2, 3] (возвращает 1)

front(): [2, 3] (возвращает 2, не удаляя)

3. Дек (Deque - Double-Ended Queue)

Дек - это структура данных, позволяющая добавлять и удалять элементы с обоих концов.

Основные операции:

Операция	Описание	Сложность
push_front(value)	Добавляет элемент в начало	O(1)
push_back(value)	Добавляет элемент в конец	O(1)
pop_front()	Удаляет элемент из начала	O(1)
pop_back()	Удаляет элемент из конца	O(1)
front()	Возвращает первый элемент	O(1)
back()	Возвращает последний элемент	O(1)

4. Приоритетная очередь (Priority Queue)

Приоритетная очередь - это структура данных, в которой каждый элемент имеет приоритет. Элемент с наибольшим приоритетом извлекается первым.

Основные операции:

Операция	Описание	Сложность
push(value)	Добавляет элемент	O(log n)

Операция	Описание	Сложность
pop()	Удаляет элемент с наибольшим приоритетом	O(log n)
top()	Возвращает элемент с наибольшим приоритетом	O(1)

Реализация: Обычно на основе двоичной кучи (heap).

Примеры выполнения практической части

Пример 1. Стек

Задача: Реализовать класс стека и проверить правильность расстановки скобок.

```
#include <iostream>
#include <string>
#include <stdexcept>

template <typename T>
class Stack {
private:
    T* data;
    int topIndex;
    int capacity;

    void resize() {
        int newCapacity = capacity * 2;
        T* newData = new T[newCapacity];
        for (int i = 0; i <= topIndex; i++) {
```

```
            newData[i] = data[i];
        }
        delete[] data;
        data = newData;
        capacity = newCapacity;
    }

public:
    Stack(int initialCapacity = 10) : capacity(initialCapacity), topIndex(-1) {
        data = new T[capacity];
    }

    ~Stack() { delete[] data; }

    void push(const T& value) {
        if (topIndex >= capacity - 1) resize();
        data[++topIndex] = value;
    }

    T pop() {
        if (isEmpty()) throw std::underflow_error("Stack is empty");
        return data[topIndex--];
    }

    T top() const {
        if (isEmpty()) throw std::underflow_error("Stack is empty");
        return data[topIndex];
    }
```

```
    bool isEmpty() const { return topIndex == -1; }
    int size() const { return topIndex + 1; }
};

bool checkBrackets(const std::string& expr) {
    Stack<char> stack;
    for (char ch : expr) {
        if (ch == '(' || ch == '[' || ch == '{') {
            stack.push(ch);
        } else if (ch == ')' || ch == ']' || ch == '}') {
            if (stack.isEmpty()) return false;
            char top = stack.pop();
            if ((ch == ')' && top != '(') ||
                (ch == ']' && top != '[') ||
                (ch == '}' && top != '{')) {
                return false;
            }
        }
    }
    return stack.isEmpty();
}

int main() {
    std::cout << "() : " << (checkBrackets("()") ? "BEPHO" : "HEBEPHO") << std::endl;
    std::cout << "([]) : " << (checkBrackets("([])") ? "BEPHO" : "HEBEPHO") << std::endl;
    std::cout << "((()) : " << (checkBrackets("((())") ? "BEPHO" : "HEBEPHO") << std::endl;
```

```
    return 0;
}
```

Пример 2. Очередь

Задача: Реализовать класс очереди на кольцевом буфере.

```
#include <iostream>
#include <stdexcept>

template <typename T>
class Queue {
private:
    T* data;
    int frontIndex;
    int rearIndex;
    int count;
    int capacity;

    void resize() {
        int newCapacity = capacity * 2;
        T* newData = new T[newCapacity];
        for (int i = 0; i < count; i++) {
            newData[i] = data[(frontIndex + i) % capacity];
        }
        delete[] data;
        data = newData;
        frontIndex = 0;
        rearIndex = count;
        capacity = newCapacity;
    }
```

```
public:
    Queue(int initialCapacity = 10)
        : capacity(initialCapacity), frontIndex(0), rearIndex(0), count(0) {
        data = new T[capacity];
    }

    ~Queue() { delete[] data; }

    void enqueue(const T& value) {
        if (count >= capacity) resize();
        data[rearIndex] = value;
        rearIndex = (rearIndex + 1) % capacity;
        count++;
    }

    T dequeue() {
        if (isEmpty()) throw std::underflow_error("Queue is empty");
        T value = data[frontIndex];
        frontIndex = (frontIndex + 1) % capacity;
        count--;
        return value;
    }

    T front() const {
        if (isEmpty()) throw std::underflow_error("Queue is empty");
        return data[frontIndex];
    }
```

```
    bool isEmpty() const { return count == 0; }
    int size() const { return count; }
};

int main() {
    Queue<int> q
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    std::cout << "dequeue: " << q.dequeue() << std::endl;
    std::cout << "front: " << q.front() << std::endl;
    return 0;
}
```

Пример 3. Дек

Задача: Реализовать класс дека и проверить, является ли строка палиндромом.

```
#include <iostream>
#include <string>
#include <stdexcept>

template <typename T>
class Deque {
private:
    struct Node {
        T data;
        Node* prev;
        Node* next;
```

```
    Node(const T& value) : data(value), prev(nullptr), next(nullptr) {}
};

Node* head;
Node* tail;
int count;

public:
Deque() : head(nullptr), tail(nullptr), count(0) {}

~Deque() {
    while (head != nullptr) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
}

void push_front(const T& value) {
    Node* newNode = new Node(value);
    if (head == nullptr) {
        head = tail = newNode;
    } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
    count++;
}
```

```
void push_back(const T& value) {
    Node* newNode = new Node(value);
    if (tail == nullptr) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode;
    }
    count++;
}

T pop_front() {
    if (isEmpty()) throw std::underflow_error("Deque is empty");
    Node* temp = head;
    T value = temp->data;
    head = head->next;
    if (head != nullptr) head->prev = nullptr;
    else tail = nullptr;
    delete temp;
    count--;
    return value;
}

T pop_back() {
    if (isEmpty()) throw std::underflow_error("Deque is empty");
    Node* temp = tail;
    T value = temp->data;
```

```
        tail = tail->prev;
        if (tail != nullptr) tail->next = nullptr;
        else head = nullptr;
        delete temp;
        count--;
        return value;
    }

    T front() const {
        if (isEmpty()) throw std::underflow_error("Deque is empty");
        return head->data;
    }

    T back() const {
        if (isEmpty()) throw std::underflow_error("Deque is empty");
        return tail->data;
    }

    bool isEmpty() const { return count == 0; }
    int size() const { return count; }
};

bool isPalindrome(const std::string& str) {
    Deque<char> dq;
    for (char ch : str) {
        dq.push_back(ch);
    }
    while (dq.size() > 1) {
        if (dq.pop_front() != dq.pop_back()) return false;
```

```
        }
        return true;
    }

int main() {
    std::cout << "radar: " << (isPalindrome("radar") ? "YES" : "NO") <<
std::endl;
    std::cout << "hello: " << (isPalindrome("hello") ? "YES" : "NO") <<
std::endl;
    return 0;
}
```

Пример 4. Приоритетная очередь

Задача: Реализовать приоритетную очередь на основе кучи (max-heap).

```
#include <iostream>
#include <stdexcept>

template <typename T>
class PriorityQueue {
private:
    T* heap;
    int count;
    int capacity;

    void resize() {
        int newCapacity = capacity * 2;
        T* newHeap = new T[newCapacity];
        for (int i = 0; i < count; i++) newHeap[i] = heap[i];
        delete[] heap;
```

```
        heap = newHeap;
        capacity = newCapacity;
    }

    void heapifyUp(int index) {
        while (index > 0) {
            int parent = (index - 1) / 2;
            if (heap[index] > heap[parent]) {
                std::swap(heap[index], heap[parent]);
                index = parent;
            } else break;
        }
    }

    void heapifyDown(int index) {
        while (true) {
            int left = 2 * index + 1;
            int right = 2 * index + 2;
            int largest = index;
            if (left < count && heap[left] > heap[largest]) largest = left;
            if (right < count && heap[right] > heap[largest]) largest = right;
            if (largest == index) break;
            std::swap(heap[index], heap[largest]);
            index = largest;
        }
    }

public:
```

```
    PriorityQueue(int initialCapacity = 10) : capacity(initialCapacity), count(0) {
        heap = new T[capacity];
    }

    ~PriorityQueue() { delete[] heap; }

    void push(const T& value) {
        if (count >= capacity) resize();
        heap[count] = value;
        heapifyUp(count);
        count++;
    }

    T pop() {
        if (isEmpty()) throw std::underflow_error("PriorityQueue is empty");
        T maxValue = heap[0];
        heap[0] = heap[count - 1];
        count--;
        heapifyDown(0);
        return maxValue;
    }

    T top() const {
        if (isEmpty()) throw std::underflow_error("PriorityQueue is empty");
        return heap[0];
    }

    bool isEmpty() const { return count == 0; }
```

```
    int size() const { return count; }
};

int main() {
    PriorityQueue<int> pq;
    pq.push(30);
    pq.push(10);
    pq.push(50);
    pq.push(20);

    while (!pq.isEmpty()) {
        std::cout << pq.pop() << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

Индивидуальные задания

Каждый вариант содержит 4 задания - по одному на каждую структуру данных: стек, очередь, дек, приоритетная очередь.

Вариант 1

Структура	Задание
Стек	Проверить правильность расстановки скобок в выражении (вида ([]{}))
Очередь	Смоделировать очередь в банке (время обслуживания клиентов)

Структура	Задание
Дек	Проверить, является ли строка палиндромом (с использованием дека)
Приоритетная очередь	Найти K наибольших элементов в массиве

Вариант 2

Структура	Задание
Стек	Вычислить значение выражения в обратной польской записи (например, 3 4 + 5 *)
Очередь	Реализовать очередь на двух стеках
Дек	Реализовать дек с фиксированной ёмкостью (кольцевой буфер)
Приоритетная очередь	Найти K наименьших элементов в массиве

Вариант 3

Структура	Задание
Стек	Перевести число из десятичной системы в двоичную
Очередь	Найти максимальный элемент в

Структура	Задание
	очереди за O(1)
Дек	Найти максимальный элемент в деке за O(1)
Приоритетная очередь	Сортировка с помощью приоритетной очереди (Heap Sort)

Вариант 4

Структура	Задание
Стек	Найти минимальный элемент в стеке за O(1)
Очередь	Найти минимальный элемент в очереди за O(1)
Дек	Найти минимальный элемент в деке за O(1)
Приоритетная очередь	Объединить K отсортированных списков

Вариант 5

Структура	Задание
Стек	Проверить, является ли строка палиндромом (с использованием стека)

Структура	Задание
Очередь	Реализовать циклическую очередь фиксированного размера
Дек	Реализовать дек с операцией rotateLeft(k)
Приоритетная очередь	Найти медиану потока чисел (две кучи)

Вариант 6

Структура	Задание
Стек	Сортировать стек (разрешается использовать дополнительный стек)
Очередь	Реализовать очередь с приоритетом (обычная + приоритетная)
Дек	Реализовать дек с операцией reverse() (разворот)
Приоритетная очередь	Реализовать приоритетную очередь с изменением приоритета

Вариант 7

Структура	Задание
Стек	Удалить все чётные элементы из стека
Очередь	Сортировать очередь (разрешается

Структура	Задание
	использовать дополнительную очередь)
Дек	Удалить все чётные элементы из дека
Приоритетная очередь	Симуляция планировщика задач (процессы с разными приоритетами)

Вариант 8

Структура	Задание
Стек	Удалить все нечётные элементы из стека
Очередь	Проверить, является ли очередь палиндромом
Дек	Удалить все нечётные элементы из дека
Приоритетная очередь	Реализовать min-heap вместо max-heap

Вариант 9

Структура	Задание
Стек	Проверить правильность HTML-разметки (например,)

Структура	Задание
Очередь	Удалить все чётные элементы из очереди
Дек	Проверить, является ли дек палиндромом
Приоритетная очередь	Реализовать приоритетную очередь на основе бинарной кучи из массива

Вариант 10

Структура	Задание
Стек	Вычислить значение выражения с унарным минусом (-5 + 3)
Очередь	Удалить все нечётные элементы из очереди
Дек	Скользящее окно: найти максимумы во всех подмассивах длины k
Приоритетная очередь	Реализовать операцию merge (слияние двух приоритетных очередей)

Вариант 11

Структура	Задание
Стек	Преобразовать инфиксную запись в постфиксную (алгоритм сортировочной станции)
Очередь	Объединить две очереди в одну (чередуя элементы)
Дек	Скользящее окно: найти минимумы во всех подмассивах длины k
Приоритетная очередь	Реализовать приоритетную очередь с фиксированной ёмкостью

Вариант 12

Структура	Задание
Стек	Реализовать стек с поддержкой операции getMiddle() (середина стека)
Очередь	Разделить очередь на две: чётные и нечётные позиции
Дек	Реализовать дек с операцией insertAt(index)
Приоритетная очередь	Проверить, является ли массив двоичной кучей

Вариант 13

Структура	Задание
Стек	Реализовать стек с поддержкой операции reverse() (разворот)
Очередь	Реализовать очередь с операцией rotate(k) (циклический сдвиг)
Дек	Реализовать дек с операцией removeAt(index)
Приоритетная очередь	Восстановить свойство кучи (heapify) для произвольного массива

Вариант 14

Структура	Задание
Стек	Реализовать два стека в одном массиве (экономное использование памяти)
Очередь	Реализовать очередь с операцией reverse() (разворот)
Дек	Реализовать дек, который возвращает сумму всех элементов за O(1)
Приоритетная очередь	Реализовать приоритетную очередь для пользовательских типов (с компаратором)

Вариант 15

Структура	Задание
Стек	Реализовать стек с поддержкой операции getMin() за O(1)
Очередь	Реализовать очередь, которая возвращает среднее арифметическое за O(1)
Дек	Реализовать дек на основе кольцевого буфера
Приоритетная очередь	Реализовать приоритетную очередь с поддержкой удаления произвольного элемента

Контрольные вопросы

1. В чём заключается принцип работы стека (LIFO)?
2. В чём заключается принцип работы очереди (FIFO)?
3. Чем дек отличается от обычной очереди?
4. Что такое приоритетная очередь? Где она применяется?
5. Как реализовать стек на основе динамического массива?
6. Как реализовать очередь на основе кольцевого буфера?
7. Почему приоритетную очередь обычно реализуют на куче?
8. Какова сложность операций push и pop в стеке?
9. Какова сложность операций enqueue и dequeue в очереди?
10. Какова сложность операций в приоритетной очереди? Почему?
11. Как проверить правильность расстановки скобок с помощью стека?
12. Как реализовать очередь на двух стеках? Какова сложность?
13. Как реализовать дек на основе двусвязного списка?
14. Как реализовать дек на основе кольцевого буфера?
15. Как реализовать стек с поддержкой получения минимума за O(1)?

Требования к отчёту

Титульный лист

Цель работы

Теоретическая часть

Постановка задачи (4 задания согласно варианту)

Листинг программы (4 реализации с комментариями)

Результаты работы (скриншоты вывода)

Анализ сложности для каждой операции

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Критерии оценивания (единые для всех)

Максимальный балл: 100

№	Критерий	Баллы
1	Стек (Stack) - реализация класса и решение задачи	**25**
1.1	Класс стека реализован корректно (push, pop, top, isEmpty)	10
1.2	Задача со стеком решена верно	10
1.3	Обработка ошибок (попытка pop из пустого стека)	5
2	Очередь (Queue) - реализация класса и решение задачи	**25**

№	Критерий	Баллы
2.1	Класс очереди реализован корректно (enqueue, dequeue, front, isEmpty)	10
2.2	Задача с очередью решена верно	10
2.3	Обработка ошибок (попытка dequeue из пустой очереди)	5
3	Дек (Deque) - реализация класса и решение задачи	**25**
3.1	Класс дека реализован корректно (push_front, push_back, pop_front, pop_back)	10
3.2	Задача с деком решена верно	10
3.3	Обработка ошибок (попытка pop из пустого дека)	5
4	Приоритетная очередь (PriorityQueue) - реализация и решение задачи	**25**
4.1	Класс приоритетной очереди реализован корректно (push, pop, top, isEmpty)	10
4.2	Задача с приоритетной очередью решена верно	10
4.3	Корректная работа кучи	5

№	Критерий	Баллы
	(heapifyUp/heapifyDown)	
Итого		**100**

Детализация по каждому критерию

1. Стек (25 баллов)

Подкритерий	Баллы
Корректная реализация push (с расширением массива при необходимости)	3
Корректная реализация pop	2
Корректная реализация top	2
Корректная реализация isEmpty и size	1
Корректная работа деструктора (освобождение памяти)	2
Задача решена верно (алгоритм работает корректно на всех тестах)	10
Обработка ошибок (выброс исключения при pop из пустого стека)	5
Итого	*25*

2. Очередь (25 баллов)

Подкритерий	Баллы
Корректная реализация enqueue (с кольцевым буфером или расширением)	3
Корректная реализация dequeue	2
Корректная реализация front	2
Корректная реализация isEmpty и size	1
Корректная работа деструктора (освобождение памяти)	2
Задача решена верно (алгоритм работает корректно на всех тестах)	10
Обработка ошибок (выброс исключения при dequeue из пустой очереди)	5
Итого	*25*

3. Дек (25 баллов)

Подкритерий	Баллы
Корректная реализация push_front	3
Корректная реализация push_back	3
Корректная реализация pop_front	2
Корректная реализация pop_back	2

Подкритерий	Баллы
Корректная реализация front и back	2
Корректная реализация isEmpty и size	1
Корректная работа деструктора (освобождение памяти)	2
Задача решена верно (алгоритм работает корректно на всех тестах)	10
Итого	*25*

4. Приоритетная очередь (25 баллов)

Подкритерий	Баллы
Корректная реализация push (heapifyUp)	3
Корректная реализация pop (heapifyDown)	3
Корректная реализация top	2
Корректная реализация isEmpty и size	1
Корректное управление памятью (расширение массива, деструктор)	2
Задача решена верно (алгоритм работает корректно на всех тестах)	10
Корректная обработка пустой очереди	2

Подкритерий	Баллы
Правильная сложность O(log n) для push/pop	2
Итого	*25*

Штрафные баллы (вычитаются из общей суммы)

Нарушение	Штраф
Утечка памяти (не освобождена динамическая память в деструкторе)	-15
Двойное освобождение памяти	-15
Нет обработки попытки pop из пустой структуры	-10
Нет проверки выхода за границы в top()/front()	-5
Использование STL контейнеров вместо собственной реализации	-10
Код не отформатирован, нет комментариев	-5
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка (традиционная)	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Пример расчёта оценки

Студент выполнил:

Критерий	Максимум	Получено
Стек	25	22 (ошибка в расширении массива)
Очередь	25	20 (неверная обработка краевых случаев)
Дек	25	25 (всё верно)
Приоритетная очередь	25	18 (ошибка в heapify)
Штраф	-	-10 (утечка памяти в стеке)

Критерий	Максимум	Получено
Итого	**100**	**75**

Оценка: Хорошо (75 баллов)

Лабораторная работа №13

Рекурсивные алгоритмы. Простые сортировки

Цель работы:

1. Освоить принципы рекурсии: прямой и косвенной, базовый случай и шаг рекурсии
2. Научиться применять рекурсию для решения практических задач (факториал, числа Фибоначчи, Ханойские башни, обход лабиринта)
3. Реализовать и сравнить простые алгоритмы сортировки: пузырьковую, вставками, выбором
4. Проанализировать сложность рекурсивных алгоритмов и простых сортировок
5. Сравнить рекурсивную и итеративную реализации одних и тех же алгоритмов

Теоретическое обоснование

1. Что такое рекурсия?

Рекурсия - это метод программирования, при котором функция вызывает саму себя для решения подзадач меньшего размера.

Основные компоненты рекурсивной функции:

Компонент	Описание	Пример для факториала
Базовый случай	Условие остановки рекурсии	if (n == 0) return 1;
Шаг рекурсии	Вызов функции с меньшими параметрами	return n * factorial(n - 1);

Визуализация рекурсии (факториал 5):

factorial(5)

→ 5 * factorial(4)

→ 4 * factorial(3)

→ 3 * factorial(2)

→ 2 * factorial(1)

→ 1 * factorial(0)

→ 1 (базовый случай)

← 1 * 1 = 1

← 2 * 1 = 2

← 3 * 2 = 6

← 4 * 6 = 24

← 5 * 24 = 120

2. Виды рекурсии

Вид	Описание	Пример
Прямая	Функция	factorial(n) вызывает fact

Вид	Описание	Пример
рекурсия	вызывает саму себя	orial(n-1)
Косвенная рекурсия	Функция A вызывает B, B вызывает A	even(n) → odd(n-1), odd(n) → even(n-1)
Хвостовая рекурсия	Рекурсивный вызов - последняя операция	sum(n, acc) с аккумулятором

3. Классические рекурсивные задачи

Факториал

```
// Рекурсивно
int factorial(int n) {
    if (n <= 1) return 1;           // базовый случай
    return n * factorial(n - 1);    // шаг рекурсии
}

// Итеративно
int factorialIter(int n) {
    int result = 1;
    for (int i = 2; i <= n; i++) result *= i;
    return result;
}
```

Числа Фибоначчи

```
// Рекурсивно (экспоненциальная сложность O(2^n))
int fibRecursive(int n) {
    if (n <= 1) return n;
    return fibRecursive(n - 1) + fibRecursive(n - 2);
}

// Итеративно (линейная сложность O(n))
int fibIterative(int n) {
    if (n <= 1) return n;
    int a = 0, b = 1;
    for (int i = 2; i <= n; i++) {
        int c = a + b;
        a = b;
        b = c;
    }
    return b;
}
```

Ханойские башни

```
void hanoi(int n, char from, char to, char aux) {
    if (n == 1) {
        cout << "Переместить диск 1 с " << from << " на " << to << endl;
        return;
    }
    hanoi(n - 1, from, aux, to);
    cout << "Переместить диск " << n << " с " << from << " на " << to <<
endl;
    hanoi(n - 1, aux, to, from);
```

```
}
```

4. Простые сортировки

Сравнение сложности

Алгоритм	Лучший случай	Средний случай	Худший случай	Память	Устойчивость
Пузырьковая	O(n)	O(n²)	O(n²)	O(1)	Да
Вставками	O(n)	O(n²)	O(n²)	O(1)	Да
Выбором	O(n²)	O(n²)	O(n²)	O(1)	Нет

Пузырьковая сортировка (Bubble Sort)

```
void bubbleSort(int arr[], int n) {
    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]) {
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }
        if (!swapped) break;  // оптимизация
    }
```

```
}
```

Сортировка вставками (Insertion Sort)

```
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; 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;
    }
}
```

Сортировка выбором (Selection Sort)

```
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j
            }
        }
        if (minIndex != i) {
            swap(arr[i], arr[minIndex]);
        }
```

```
    }
}
```

Примеры выполнения практической части

Пример 1. Рекурсивный обход лабиринта

Задача: Найти путь от старта (S) до финиша (F) в лабиринте с помощью рекурсивного поиска с возвратом (backtracking).

```
#include <iostream>
#include <vector>

using namespace std;

const int N = 5;
char maze[N][N] = {
    {'S', '.', '#', '.', '.'},
    {'#', '.', '#', '.', '#'},
    {'.', '.', '.', '#', '.'},
    {'#', '#', '.', '.', '.'},
    {'.', '.', '#', '.', 'F'}
};

bool visited[N][N] = {false};

bool findPath(int x, int y) {
    // Проверка выхода за границы
    if (x < 0 || x >= N || y < 0 || y >= N) return false;
    // Проверка стены или посещённой клетки
    if (maze[x][y] == '#' || visited[x][y]) return false;
```

```
    // Проверка достижения финиша
    if (maze[x][y] == 'F') {
        maze[x][y] = '*';
        return true;
    }

    visited[x][y] = true;

    // Рекурсивный поиск в 4 направлениях
    if (findPath(x + 1, y) ||  // вниз
        findPath(x - 1, y) ||  // вверх
        findPath(x, y + 1) ||  // вправо
        findPath(x, y - 1)) {  // влево
        if (maze[x][y] != 'S') maze[x][y] = '*';
        return true;
    }

    return false;
}

int main() {
    int startX = 0, startY = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (maze[i][j] == 'S') {
                startX = i;
                startY = j;
            }
        }
```

```
    }

    if (findPath(startX, startY)) {
        cout << "Путь найден!" << endl;
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                cout << maze[i][j] << " ";
            }
            cout << endl;
        }
    } else {
        cout << "Путь не найден" << endl;
    }

    return 0;
}
```

Ожидаемый вывод:

```
Путь найден!
S * # . .
# * # . #
* * * # .
# # * * *
. . # . F
```

Пример 2. Сравнение сортировок

Задача: Реализовать три простые сортировки и сравнить их производительность.

```
#include <iostream>
```

```cpp
#include <chrono>
#include <algorithm>

using namespace std;
using namespace chrono;

void bubbleSort(int arr[], int n) {
    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]) {
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }
        if (!swapped) break;
    }
}

void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; 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;
    }
```

```
}

void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        if (minIndex != i) swap(arr[i], arr[minIndex]);
    }
}

long long measureTime(void (*sortFunc)(int[], int), int arr[], int n) {
    int* copy = new int[n];
    for (int i = 0; i < n; i++) copy[i] = arr[i];

    auto start = high_resolution_clock::now();
    sortFunc(copy, n);
    auto end = high_resolution_clock::now();

    delete[] copy;
    return duration_cast<microseconds>(end - start).count();
}

int main() {
    const int N = 10000;
    int* arr = new int[N];
```

```
    // Заполняем случайными числами
    for (int i = 0; i < N; i++) arr[i] = rand() % N;

    cout << "Сравнение производительности (N = " << N << "):" << endl;
    cout << "Пузырьковая: " << measureTime(bubbleSort, arr, N) << " мкс"
<< endl;
    cout << "Вставками:   " << measureTime(insertionSort, arr, N) << " мкс"
<< endl;
    cout << "Выбором:     " << measureTime(selectionSort, arr, N) << " мкс"
<< endl;

    delete[] arr;
    return 0;
}
```

Индивидуальные задания

Часть 1. Рекурсивные алгоритмы

Каждый вариант содержит 4 рекурсивные задачи:

вар.	Задача 1	Задача 2	Задача 3	Задача 4
	Факториал числа	Числа Фибоначчи	Сумма цифр числа	Степень числа (a^n)
	Факториал числа	Числа Фибоначчи	Количество цифр	Бинарный поиск

вар.	Задача 1	Задача 2	Задача 3	Задача 4
	Факториал числа	Числа Фибоначчи	Переворот строки	Проверка палиндрома
	Факториал числа	Числа Фибоначчи	НОД (алгоритм Евклида)	Ханойские башни (3 диска)
	Факториал числа	Числа Фибоначчи	Сумма элементов массива	Ханойские башни (4 диска)
	Факториал числа	Числа Фибоначчи	Найти максимум в массиве	Ханойские башни (5 дисков)
	Факториал числа	Треугольные числа	Произведение цифр	Перебор всех перестановок
	Факториал числа	Число сочетаний C(n,k)	Разложение на слагаемые	Генерация всех подмножеств
	Факториал числа	Числа трибоначчи	Обход лабиринта (4×4)	Задача о ходе коня

вар.	Задача 1	Задача 2	Задача 3	Задача 4
0	Факториал числа	Числа Фибоначчи	Обход лабиринта (5×5)	Задача о ферзях (4×4)
1	Факториал числа	Числа Фибоначчи	Обход лабиринта (6×6)	Задача о ферзях (5×5)
2	Факториал числа	Числа Фибоначчи	Проверка сбалансированности скобок	Генерация всех комбинаций
3	Факториал числа	Числа Фибоначчи	Вычисление определителя (2×2)	Ханойские башни (с подсчётом ходов)
4	Факториал числа	Числа Фибоначчи	Вычисление определителя (3×3)	Обход лабиринта (с поиском кратчайшего пути)
5	Факториал числа	Числа Фибоначчи	Алгоритм быстрого возведения в степень	Ханойские башни (с визуализацией)

Часть 2. Простые сортировки

Каждый вариант содержит задание реализовать сортировки и выполнить анализ:

№ вар.	Сортировки для реализации	Дополнительное задание
1	Пузырьковая, вставками, выбором	Сравнить производительность на случайном массиве
2	Пузырьковая, вставками, выбором	Сравнить на почти отсортированном массиве
3	Пузырьковая, вставками, выбором	Сравнить на массиве, отсортированном в обратном порядке
4	Пузырьковая, вставками	Подсчитать количество сравнений и обменов
5	Пузырьковая, выбором	Подсчитать количество сравнений и обменов
6	Вставками, выбором	Подсчитать количество сравнений и обменов
7	Пузырьковая, вставками, выбором	Определить лучшую сортировку для разных типов данных

№ вар.	Сортировки для реализации	Дополнительное задание
8	Пузырьковая, вставками, выбором	Реализовать устойчивую версию сортировки выбором
9	Пузырьковая, вставками, выбором	Реализовать сортировку вставками для связного списка
10	Пузырьковая, вставками, выбором	Реализовать сортировку выбором для связного списка
11	Пузырьковая, вставками, выбором	Реализовать рекурсивную версию пузырьковой сортировки
12	Пузырьковая, вставками, выбором	Реализовать рекурсивную версию сортировки вставками
13	Пузырьковая, вставками, выбором	Реализовать рекурсивную версию сортировки выбором
14	Пузырьковая, вставками, выбором	Сравнить производительность с std::sort
15	Пузырьковая, вставками, выбором	Визуализировать процесс сортировки (пошаговый вывод)

Контрольные вопросы

1. Что такое рекурсия? Из каких компонентов состоит рекурсивная функция?
2. Что такое базовый случай рекурсии? Почему он важен?
3. Чем отличается прямая рекурсия от косвенной?

4. Что такое хвостовая рекурсия? В чём её преимущество?
5. Какова временная сложность рекурсивного вычисления чисел Фибоначчи? Почему?
6. Как можно оптимизировать рекурсивное вычисление Фибоначчи?
7. В чём отличие рекурсивной реализации от итеративной?
8. Какова сложность пузырьковой сортировки? Когда она становится O(n)?
9. Чем сортировка вставками отличается от сортировки выбором?
10. Какая из простых сортировок является устойчивой? Что это значит?
11. Какая из простых сортировок работает лучше на почти отсортированных данных?
12. Как реализовать пузырьковую сортировку с оптимизацией?
13. Что такое рекурсивный поиск с возвратом (backtracking)?
14. Как рекурсивно обойти лабиринт?
15. Какова сложность Ханойских башен? Сколько ходов нужно для перемещения n дисков?

Требования к отчёту

Титульный лист

Цель работы

Теоретическая часть (рекурсия, простые сортировки)

Постановка задачи (согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода)

Анализ сложности рекурсивных алгоритмов

Сравнительная таблица сортировок

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Критерии оценивания

Максимальный балл: 100

№	Критерий	Баллы
1	Рекурсивные алгоритмы (4 задачи)	50
1.1	Задача 1 решена корректно	12
1.2	Задача 2 решена корректно	12
1.3	Задача 3 решена корректно	13
1.4	Задача 4 решена корректно	13
2	Простые сортировки	40
2.1	Реализация пузырьковой сортировки	10
2.2	Реализация сортировки вставками	10
2.3	Реализация сортировки выбором	10
2.4	Анализ производительности и сравнение	10
3	Оформление и отчёт	10
3.1	Код отформатирован, есть комментарии	5
3.2	Есть ответы на контрольные вопросы, ссылка на Git	5
Итог		100

№	Критерий	Баллы
о		

Штрафные баллы

Нарушение	Штраф
Отсутствие базового случая в рекурсии (бесконечная рекурсия)	-20
Неверная сложность сортировки	-10
Нет анализа производительности	-10
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №14

Быстрая сортировка, сортировка слиянием, пирамидальная сортировка

Цель работы:

1. Реализовать три эффективных алгоритма сортировки: быструю сортировку (Quick Sort), сортировку слиянием (Merge Sort) и пирамидальную сортировку (Heap Sort)
2. Освоить принципы «разделяй и властвуй» на примере быстрой сортировки и сортировки слиянием
3. Изучить структуру данных «куча» (heap) для реализации пирамидальной сортировки
4. Проанализировать и сравнить производительность различных алгоритмов сортировки
5. Понять преимущества и недостатки каждого алгоритма в различных сценариях использования

Теоретическое обоснование

1. Сравнение эффективных сортировок

Алгоритм	Лучший случай	Средний случай	Худший случай	Память	Устойчивость
Быстрая сортировка	O (n log n)	O (n log n)	O (n^2)	O (log n)	Нет
Сортировка слиянием	O (n log n)	O (n log n)	O (n log n)	O (n)	Да
Пирамидальная	O (n log	O (n log n)	O (n log	O (1)	Нет

Алгоритм	Лучший случай	Средний случай	Худший случай	Память	Устойчивость
	n)		n)		

2. Быстрая сортировка (Quick Sort)

Принцип работы (разделяй и властвуй):

Выбрать опорный элемент (pivot)

Разделить массив: элементы меньше pivot - слева, больше - справа

Рекурсивно применить к левой и правой частям

```
int partition(int arr[], int low, int high) {
    int pivot = arr[high];  // опорный элемент (последний)
    int i = low - 1;        // индекс для меньших элементов

    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;  // позиция опорного элемента
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
```

```
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }
```

Выбор опорного элемента:

Стратегия	**Преимущества**	**Недостатки**
Первый элемент	Просто	Плохо на почти отсортированных
Последний элемент	Просто	Плохо на почти отсортированных
Средний элемент	Лучше	Не гарантирует
Случайный элемент	Хорошо против худшего случая	Требует генерации
Медиана из трёх	Хорошо	Чуть сложнее

3. Сортировка слиянием (Merge Sort)

Принцип работы:

Разделить массив пополам

Рекурсивно отсортировать каждую половину

Слить две отсортированные половины

```
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int* L = new int[n1];
    int* R = new int[n2];

    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];

    delete[] L;
    delete[] R;
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
```

```
    }
}
```

4. Пирамидальная сортировка (Heap Sort)

Принцип работы:

Построить двоичную кучу (max-heap) из массива

Извлекать корень (максимальный элемент) и помещать в конец массива

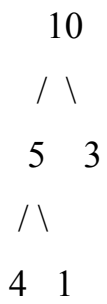
Восстанавливать свойство кучи

```
void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest])
        largest = left;
    if (right < n && arr[right] > arr[largest])
        largest = right;

    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
    // Построение кучи
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
```

```
    // Извлечение элементов
    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}
```

Исходный массив: [4, 10, 3, 5, 1]

Куча (max-heap): [10, 5, 3, 4, 1]



Примеры выполнения практической части

Пример 1. Быстрая сортировка с разными вариантами выбора опорного элемента

```
#include <iostream>
using namespace std;

// Выбор опорного элемента: последний
int partitionLast(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
```

```
            if (arr[j] <= pivot) {
                i++;
                swap(arr[i], arr[j]);
            }
        }
        swap(arr[i + 1], arr[high]);
        return i + 1;
    }

    // Выбор опорного элемента: медиана из трёх
    int medianOfThree(int arr[], int low, int high) {
        int mid = low + (high - low) / 2;
        if (arr[low] > arr[mid]) swap(arr[low], arr[mid]);
        if (arr[low] > arr[high]) swap(arr[low], arr[high]);
        if (arr[mid] > arr[high]) swap(arr[mid], arr[high]);
        swap(arr[mid], arr[high]);
        return arr[high];
    }

    int partitionMedian(int arr[], int low, int high) {
        medianOfThree(arr, low, high);
        return partitionLast(arr, low, high);
    }

    void quickSort(int arr[], int low, int high, int (*partitionFunc)(int[], int, int))
{
        if (low < high) {
            int pi = partitionFunc(arr, low, high);
            quickSort(arr, low, pi - 1, partitionFunc);
```

```
        quickSort(arr, pi + 1, high, partitionFunc);
    }
}

int main() {
    int arr1[] = {64, 34, 25, 12, 22, 11, 90, 5};
    int n = sizeof(arr1) / sizeof(arr1[0]);

    quickSort(arr1, 0, n - 1, partitionLast);

    cout << "Быстрая сортировка (опорный - последний): ";
    for (int x : arr1) cout << x << " ";
    cout << endl;

    return 0;
}
```

Пример 2. Сравнение всех трёх сортировок

```
#include <iostream>
#include <chrono>
#include <algorithm>

using namespace std;
using namespace chrono;

// Быстрая сортировка
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
```

```
    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) swap(arr[++i], arr[j]);
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

// Сортировка слиянием
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1, n2 = right - mid;
    int* L = new int[n1];
    int* R = new int[n2];

    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left
    while (i < n1 && j < n2) arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
```

```
    delete[] L;
    delete[] R;
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

// Пирамидальная сортировка
void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest]) largest = left;
    if (right < n && arr[right] > arr[largest]) largest = right;

    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
```

```
    for (int i = n / 2 - 1; i >= 0; i--) heapify(arr, n, i);
    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}

long long measureTime(void (*sortFunc)(int[], int), int arr[], int n) {
    int* copy = new int[n];
    for (int i = 0; i < n; i++) copy[i] = arr[i];

    auto start = high_resolution_clock::now();
    sortFunc(copy, n);
    auto end = high_resolution_clock::now();

    delete[] copy;
    return duration_cast<microseconds>(end - start).count();
}

int main() {
    const int N = 50000;
    int* arr = new int[N];

    for (int i = 0; i < N; i++) arr[i] = rand() % N;

    cout << "Сравнение производительности (N = " << N << "):" << endl;

    // Для быстрой сортировки нужна обёртка
    int* copy = new int[N];
```

```
        for (int i = 0; i < N; i++) copy[i] = arr[i];
        auto start = high_resolution_clock::now();
        quickSort(copy, 0, N - 1);
        auto end = high_resolution_clock::now();
        cout << "Быстрая сортировка: " << duration_cast<microseconds>(end - start).count() << " мкс" << endl;
        delete[] copy;

        cout << "Сортировка слиянием: " << measureTime([](int a[], int n) { mergeSort(a, 0, n - 1); }, arr, N) << " мкс" << endl;
        cout << "Пирамидальная:       " << measureTime(heapSort, arr, N) << " мкс" << endl;

        delete[] arr;
        return 0;
    }
```

Пример 3. Сортировка слиянием для больших файлов

```
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>

using namespace std;

void merge(vector<int>& arr, int left, int mid, int right) {
    vector<int> temp(right - left + 1);
    int i = left, j = mid + 1, k = 0;
```

```
    while (i <= mid && j <= right) {
        temp[k++] = (arr[i] <= arr[j]) ? arr[i++] : arr[j++];
    }
    while (i <= mid) temp[k++] = arr[i++];
    while (j <= right) temp[k++] = arr[j++];

    for (int idx = 0; idx < k; idx++) arr[left + idx] = temp[idx];
}

void mergeSort(vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

int main() {
    vector<int> numbers;
    ifstream in("numbers.txt");
    int x;
    while (in >> x) numbers.push_back(x);
    in.close();

    mergeSort(numbers, 0, numbers.size() - 1);

    ofstream out("sorted.txt");
```

```
    for (int num : numbers) out << num << " ";
    out.close();

    cout << "Сортировка завершена. Результат в sorted.txt" << endl;
    return 0;
}
```

Индивидуальные задания

Базовый уровень (15 вариантов)

Общее задание: Реализовать указанные алгоритмы сортировки и протестировать их на массивах разного размера.

№ вар.	Алгоритмы	Тестовые данные
1	Быстрая сортировка	Случайный массив, размеры 100, 1000, 10000
2	Сортировка слиянием	Случайный массив, размеры 100, 1000, 10000
3	Пирамидальная сортировка	Случайный массив, размеры 100, 1000, 10000
4	Быстрая + слиянием	Сравнить на случайных массивах
5	Быстрая + пирамидальная	Сравнить на случайных массивах
6	Слиянием +	Сравнить на случайных массивах

№ вар.	Алгоритмы	Тестовые данные
	пирамидальная	
7	Быстрая сортировка	Почти отсортированный массив
8	Сортировка слиянием	Почти отсортированный массив
9	Пирамидальная сортировка	Почти отсортированный массив
10	Все три сортировки	Сравнить на случайных массивах (N = 10000)
11	Все три сортировки	Сравнить на почти отсортированных массивах
12	Все три сортировки	Сравнить на обратно отсортированных массивах
13	Быстрая сортировка	Оптимизировать выбор опорного элемента (медиана трёх)
14	Сортировка слиянием	Реализовать без использования дополнительной памяти
15	Пирамидальная сортировка	Реализовать итеративную версию heapify

Средний уровень (15 вариантов)

Общее задание: Реализовать указанные алгоритмы сортировки с дополнительными требованиями и провести детальный анализ.

№ вар.	Алгоритмы	Дополнительные требования
1	Быстрая сортировка	Реализовать 3 варианта выбора опорного элемента и сравнить
2	Сортировка слиянием	Реализовать сортировку для связного списка
3	Пирамидальная сортировка	Реализовать min-heap (сортировка по убыванию)
4	Быстрая + слиянием	Подсчитать количество сравнений и обменов для каждого
5	Быстрая + пирамидальная	Подсчитать количество сравнений и обменов для каждого
6	Слиянием + пирамидальная	Подсчитать количество сравнений и обменов для каждого
7	Быстрая сортировка	Реализовать итеративную версию (без рекурсии)

№ вар.	Алгоритмы	Дополнительные требования
8	Сортировка слиянием	Реализовать сортировку для строк (лексикографически)
9	Пирамидальная сортировка	Реализовать сортировку для структуры Student (по среднему баллу)
10	Все три сортировки	Построить график зависимости времени от N (N = 1000, 2000, 5000, 10000)
11	Все три сортировки	Протестировать на различных типах данных (int, double, string)
12	Быстрая сортировка	Реализовать 3-х частное разделение (для повторяющихся элементов)
13	Сортировка слиянием	Реализовать естественную сортировку слиянием (используя уже упорядоченные подмассивы)
14	Пирамидальная сортировка	Реализовать сортировку на месте без рекурсии
1	Все три сортировки	Сравнить с std::sort и

№ вар.	Алгоритмы	Дополнительные требования
5		проанализировать разницу

Продвинутый уровень (15 вариантов)

Общее задание: Реализовать все три сортировки с полной функциональностью и дополнительными возможностями. Провести углублённый анализ.

№ вар.	Дополнительные требования
1	Реализовать шаблонные версии всех сортировок для любых типов
2	Реализовать сортировку слиянием для внешней сортировки (файлы)
3	Реализовать параллельную версию быстрой сортировки (OpenMP)
4	Реализовать сортировку слиянием для потоков (Stream API)
5	Реализовать стабильную версию быстрой сортировки
6	Реализовать интроспективную сортировку (гибрид быстрой + пирамидальной)
7	Реализовать блочную сортировку слиянием (TimSort)

№ вар.	Дополнительные требования
8	Реализовать сортировку слиянием для больших файлов (>RAM)
9	Реализовать итеративную версию быстрой сортировки с ручным стеком
10	Реализовать визуализацию процесса сортировки (графическая)
11	Реализовать профилирование и оптимизацию (кэш-промахи, ветвления)
12	Реализовать сортировку слиянием для распределённых систем
13	Реализовать гибридный алгоритм (быстрая + слиянием + вставками для малых массивов)
14	Реализовать сортировку слиянием для анализа больших данных (MapReduce)
15	Реализовать интерактивную демонстрацию работы каждого алгоритма

Контрольные вопросы

1. Какова сложность быстрой сортировки в лучшем, среднем и худшем случае?
2. Когда возникает худший случай для быстрой сортировки?

3. Как можно улучшить быструю сортировку для защиты от худшего случая?
4. Какова сложность сортировки слиянием? Почему она всегда O(n log n)?
5. Какие преимущества и недостатки имеет сортировка слиянием перед быстрой?
6. Как работает пирамидальная сортировка? Какова её сложность?
7. Какая из трёх сортировок устойчивая? Почему?
8. Какая из трёх сортировок требует дополнительной памяти? Сколько?
9. Какая из трёх сортировок лучше работает на почти отсортированных данных?
10. Зачем в пирамидальной сортировке сначала строится куча?
11. Что такое «разделяй и властвуй»? Как он применяется в быстрой сортировке и сортировке слиянием?
12. Можно ли реализовать быструю сортировку итеративно (без рекурсии)?
13. Почему быстрая сортировка обычно быстрее сортировки слиянием на практике?
14. Когда выгодно использовать пирамидальную сортировку?
15. Какой алгоритм сортировки вы бы выбрали для встроенных систем с ограниченной памятью?

Требования к отчёту

Титульный лист

Цель работы

Теоретическая часть (алгоритмы сортировки, их сложность)

Постановка задачи (согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода, таблицы времени выполнения)

Анализ сложности для каждого алгоритма

Сравнительная таблица производительности

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Критерии оценивания

Максимальный балл: 100

№	Критерий	Баллы
1	Быстрая сортировка	25
1.1	Реализация алгоритма	10
1.2	Корректность работы на разных данных	10
1.3	Выбор опорного элемента (оптимизация)	5
2	Сортировка слиянием	25
2.1	Реализация алгоритма	10
2.2	Корректность слияния	10
2.3	Управление памятью (нет утечек)	5
3	Пирамидальная сортировка	25
3.1	Реализация heapify	10
3.2	Построение кучи и сортировка	10
3.3	Корректность работы	5
4	Анализ и сравнение	15

№	Критерий	Баллы
4.1	Измерение времени выполнения	5
4.2	Сравнительная таблица/график	5
4.3	Выводы о производительности	5
5	Оформление и отчёт	10
5.1	Код отформатирован, есть комментарии	5
5.2	Есть ответы на вопросы, ссылка на Git	5
Итого		100

Штрафные баллы

Нарушение	Штраф
Неверная сложность алгоритма	-10
Утечка памяти (в сортировке слиянием)	-10
Нет анализа производительности	-10
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №15

Двоичный поиск. Хеш-таблицы

Цель работы

1. Реализовать алгоритм двоичного (бинарного) поиска в отсортированном массиве
2. Освоить итеративную и рекурсивную реализации двоичного поиска
3. Изучить понятие хеш-таблицы, хеш-функции и методы разрешения коллизий
4. Реализовать хеш-таблицу с методом цепочек и открытой адресацией
5. Сравнить производительность двоичного поиска и хеш-таблицы

Теоретическое обоснование

1. Двоичный (бинарный) поиск

Двоичный поиск - это алгоритм поиска элемента в отсортированном массиве, который на каждом шаге делит интервал поиска пополам.

Принцип работы:

Находим средний элемент интервала поиска

Сравниваем с искомым значением

Если равно - возвращаем индекс

Если больше - продолжаем поиск в левой половине

Если меньше - продолжаем поиск в правой половине

Визуализация поиска числа 7 в массиве [1, 3, 5, 7, 9, 11, 13]:

[1, 3, 5, 7, 9, 11, 13] → средний = 7? Да! Нашли.

[1, 3, 5, 7, 9] → средний = 5? Меньше, ищем справа

[5, 7, 9] → средний = 7? Нашли.

Сложность:

Характеристика	Значение
Лучший случай	O(1)
Средний случай	O(log n)
Худший случай	O(log n)
Память (итеративно)	O(1)
Память (рекурсивно)	O(log n)

Итеративная реализация

```
int binarySearch(const int arr[], int n, int target) {
    int left = 0, right = n - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;  // (left + right) / 2
        if (arr[mid] == target) return mid;
        if (arr[mid] < target) left = mid + 1;
```

```
        else right = mid - 1;
    }
    return -1;
}
```

Рекурсивная реализация

```
int binarySearchRecursive(const int arr[], int left, int right, int target) {
    if (left > right) return -1;
    int mid = left + (right - left) / 2;
    if (arr[mid] == target) return mid;
    if (arr[mid] < target)
        return binarySearchRecursive(arr, mid + 1, right, target);
    return binarySearchRecursive(arr, left, mid - 1, target);
}
```

2. Хеш-таблицы

Хеш-таблица - это структура данных, реализующая ассоциативный массив (словарь) и поддерживающая операции вставки, удаления и поиска в среднем за O(1).

Основные компоненты:

Хеш-функция - преобразует ключ в индекс массива

Бакет (bucket) - элемент массива, хранящий данные

Коллизия - ситуация, когда разные ключи дают одинаковый индекс

Сложность:

Операция	Средний случай	Худший случай
Вставка	O(1)	O(n)
Поиск	O(1)	O(n)
Удаление	O(1)	O(n)

3. Хеш-функции

Требования к хеш-функции:

Детерминированность (один ключ → один хеш)

Равномерное распределение

Быстрое вычисление

Примеры хеш-функций для целых чисел:

```
// Простейшая (не рекомендуется)
int hash1(int key, int size) {
    return key % size;
}

// Для строк (простая)
int hashString(const string& key, int size) {
    int hash = 0;
    for (char c : key) hash = (hash * 31 + c) % size;
    return hash;
}
```

4. Метод цепочек (Chaining)

При коллизии элементы с одинаковым хешем хранятся в связном списке.

```
// Структура узла
template <typename K, typename V>
struct Node {
    K key;
    V value;
    Node* next;
    Node(const K& k, const V& v) : key(k), value(v), next(nullptr) {}
};

// Хеш-таблица с цепочками
template <typename K, typename V>
class HashTableChaining {
private:
    vector<Node<K, V>*> table;
    int size;

    int hash(const K& key) const {
        return key % size;
    }

public:
    HashTableChaining(int s) : size(s), table(s, nullptr) {}

    void insert(const K& key, const V& value) {
        int index = hash(key);
        Node<K, V>* newNode = new Node<K, V>(key, value);
```

```
        newNode->next = table[index];
        table[index] = newNode;
    }

    V* find(const K& key) const {
        int index = hash(key);
        Node<K, V>* current = table[index];
        while (current != nullptr) {
            if (current->key == key) return &current->value;
            current = current->next;
        }
        return nullptr;
    }
};
```

5. Открытая адресация (Open Addressing)

При коллизии ищется следующий свободный бакет.

Методы разрешения коллизий:

Метод	Формула	Описание
Линейное пробирование	(hash + i) % size	Поиск следующей свободной ячейки
Квадратичное пробирование	(hash + i^2) % size	Поиск с квадратичным шагом
Двойное хеширование	(hash1 + i * hash2) % size	Вторая хеш-функция

```
template <typename K, typename V>
class HashTableOpenAddressing {
private:
    struct Entry {
        K key;
        V value;
        bool occupied;
        Entry() : occupied(false) {}
    };

    vector<Entry> table;
    int size;

    int hash(const K& key) const {
        return key % size;
    }

public:
    HashTableOpenAddressing(int s) : size(s), table(s) {}

    void insert(const K& key, const V& value) {
        int index = hash(key);
        int i = 0;
        while (table[(index + i) % size].occupied) {
            if (table[(index + i) % size].key == key) break;
            i++;
        }
        int pos = (index + i) % size;
        table[pos].key = key;
```

```
        table[pos].value = value;
        table[pos].occupied = true;
    }

    V* find(const K& key) {
        int index = hash(key);
        int i = 0;
        while (table[(index + i) % size].occupied) {
            int pos = (index + i) % size;
            if (table[pos].key == key) return &table[pos].value;
            i++;
        }
        return nullptr;
    }
};
```

6. Сравнение методов разрешения коллизий

Характеристика	**Метод цепочек**	**Линейное пробирование**	**Квадратичное пробирование**
Использование памяти	Дополнительная на указатели	Только массив	Только массив
Кластеризация	Нет	Да (первичная)	Меньше (вторичная)

Характеристика	Метод цепочек	Линейное пробирование	Квадратичное пробирование
Производительность	Хорошая	Средняя	Хорошая
Простота реализации	Средняя	Простая	Средняя

Примеры выполнения практической части

Пример 1. Двоичный поиск (базовый уровень)

Задача: Реализовать итеративный и рекурсивный двоичный поиск. Сравнить их производительность.

```
#include <iostream>
#include <chrono>
#include <algorithm>

using namespace std;
using namespace chrono;

// Итеративная версия
int binarySearchIter(const int arr[], int n, int target) {
    int left = 0, right = n - 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;
}

// Рекурсивная версия
int binarySearchRecur(const int arr[], int left, int right, int target) {
    if (left > right) return -1;
    int mid = left + (right - left) / 2;
    if (arr[mid] == target) return mid;
    if (arr[mid] < target)
        return binarySearchRecur(arr, mid + 1, right, target);
    return binarySearchRecur(arr, left, mid - 1, target);
}

// Линейный поиск для сравнения
int linearSearch(const int arr[], int n, int target) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}

int main() {
    const int N = 1000000;
    int* arr = new int[N];
    for (int i = 0; i < N; i++) arr[i] = i;
```

```
    int target = N - 1;

    auto start = high_resolution_clock::now();
    int linearResult = linearSearch(arr, N, target);
    auto end = high_resolution_clock::now();
    long long linearTime = duration_cast<microseconds>(end - start).count();

    start = high_resolution_clock::now();
    int binaryResult = binarySearchIter(arr, N, target);
    end = high_resolution_clock::now();
    long long binaryTime = duration_cast<microseconds>(end - start).count();

    cout << "Линейный поиск:   " << linearTime << " мкс" << endl;
    cout << "Бинарный поиск:   " << binaryTime << " мкс" << endl;

    delete[] arr;
    return 0;
}
```

Ожидаемый вывод:

```
Линейный поиск:   1784 мкс
Бинарный поиск:   2 мкс
```

Пример 2. Хеш-таблица с методом цепочек (средний уровень)

Задача: Реализовать хеш-таблицу для хранения пар (ключ, значение) с методом цепочек.

```
#include <iostream>
#include <vector>
```

```
#include <string>

using namespace std;

template <typename K, typename V>
class HashTable {
private:
    struct Node {
        K key;
        V value;
        Node* next;
        Node(const K& k, const V& v) : key(k), value(v), next(nullptr) {}
    };

    vector<Node*> table;
    int size;
    int count;

    int hash(const K& key) const {
        return key % size;
    }

public:
    HashTable(int s = 10) : size(s), count(0) {
        table.resize(size, nullptr);
    }

    ~HashTable() {
        for (int i = 0; i < size; i++) {
```

```
        Node* current = table[i];
        while (current != nullptr) {
            Node* temp = current;
            current = current->next;
            delete temp;
        }
    }
}

void insert(const K& key, const V& value) {
    int index = hash(key);
    Node* current = table[index];
    while (current != nullptr) {
        if (current->key == key) {
            current->value = value;
            return;
        }
        current = current->next;
    }
    Node* newNode = new Node(key, value);
    newNode->next = table[index];
    table[index] = newNode;
    count++;
}

bool find(const K& key, V& result) const {
    int index = hash(key);
    Node* current = table[index];
    while (current != nullptr) {
```

```
        if (current->key == key) {
            result = current->value;
            return true;
        }
        current = current->next;
    }
    return false;
}

bool remove(const K& key) {
    int index = hash(key);
    Node* current = table[index];
    Node* prev = nullptr;
    while (current != nullptr) {
        if (current->key == key) {
            if (prev == nullptr) table[index] = current->next;
            else prev->next = current->next;
            delete current;
            count--;
            return true;
        }
        prev = current;
        current = current->next;
    }
    return false;
}

int getSize() const { return count; }
```

```
    void print() const {
        for (int i = 0; i < size; i++) {
            cout << i << ": ";
            Node* current = table[i];
            while (current != nullptr) {
                cout << "(" << current->key << ", " << current->value << ") ";
                current = current->next;
            }
            cout << endl;
        }
    }
};

int main() {
    HashTable<int, string> ht(5);

    ht.insert(10, "десять");
    ht.insert(20, "двадцать");
    ht.insert(30, "тридцать");
    ht.insert(15, "пятнадцать");
    ht.insert(25, "двадцать пять");

    ht.print();

    string value;
    if (ht.find(20, value)) {
        cout << "Ключ 20: " << value << endl;
    }
```

```
    ht.remove(20);
    cout << "После удаления 20:" << endl;
    ht.print();

    return 0;
}
```

Пример 3. Сравнение поиска в массиве и хеш-таблице (продвинутый уровень)

Задача: Сравнить производительность бинарного поиска в отсортированном массиве и поиска в хеш-таблице.

```
#include <iostream>
#include <chrono>
#include <unordered_map>
#include <vector>
#include <algorithm>

using namespace std;
using namespace chrono;

int main() {
    const int N = 1000000;
    vector<int> arr(N);
    unordered_map<int, int> hashMap;

    for (int i = 0; i < N; i++) {
        arr[i] = i;
        hashMap[i] = i;
```

```
    }

    vector<int> queries;
    for (int i = 0; i < 10000; i++) queries.push_back(rand() % N);

    // Бинарный поиск
    auto start = high_resolution_clock::now();
    for (int q : queries) {
        binary_search(arr.begin(), arr.end(), q);
    }
    auto end = high_resolution_clock::now();
    long long binaryTime = duration_cast<microseconds>(end - start).count();

    // Хеш-таблица
    start = high_resolution_clock::now();
    for (int q : queries) {
        hashMap.find(q);
    }
    end = high_resolution_clock::now();
    long long hashTime = duration_cast<microseconds>(end - start).count();

    cout << "10000 запросов:" << endl;
    cout << "Бинарный поиск (O(log n)):    " << binaryTime << " мкс" <<
endl;
    cout << "Хеш-таблица (O(1) среднее): " << hashTime << " мкс" <<
endl;

    return 0;
}
```

Индивидуальные задания

Часть 1. Двоичный поиск

Каждый вариант содержит задание реализовать двоичный поиск с указанными особенностями:

№ вар.	Тип данных	Особенности
1	int	Итеративная реализация
2	int	Рекурсивная реализация
3	int	Поиск первого вхождения
4	int	Поиск последнего вхождения
5	int	Поиск границ (левая и правая)
6	double	Сравнение с погрешностью
7	string	Лексикографический поиск
8	struct Person (по возрасту)	Поиск по ключу
9	int	Поиск в циклически

№ вар.	Тип данных	Особенности
		сдвинутом массиве
10	int	Поиск элемента, ближайшего к заданному
11	int	Поиск в массиве, отсортированном по убыванию
12	int	Подсчёт количества вхождений
13	int	Поиск с помощью std::lower_bound
14	int	Поиск с помощью std::upper_bound
15	int	Поиск в массиве с дубликатами (любое вхождение)

Часть 2. Хеш-таблицы

Каждый вариант содержит задание реализовать хеш-таблицу с указанными особенностями:

№ вар.	Метод разрешения коллизий	Тип ключа	Тип значения
1	Метод цепочек	int	int
2	Линейное пробирование	int	int
3	Квадратичное пробирование	int	int
4	Двойное хеширование	int	int
5	Метод цепочек	string	int
6	Линейное пробирование	string	int
7	Метод цепочек	int	string
8	Линейное пробирование	int	string
9	Метод цепочек	string	string
10	Линейное пробирование	string	string
11	Метод цепочек	int	struct Student
12	Линейное пробирование	int	struct Student

№ вар.	Метод разрешения коллизий	Тип ключа	Тип значения
13	Метод цепочек (динамическое расширение)	int	int
14	Линейное пробирование (динамическое расширение)	int	int
15	Метод цепочек + рехеширование	int	int

Часть 3. Сравнительный анализ

Задание для всех вариантов:

1. Реализовать класс хеш-таблицы согласно варианту
2. Реализовать двоичный поиск (итеративно)
3. Сравнить производительность операций поиска для N = 1000, 10000, 100000 элементов:
4. Двоичный поиск в отсортированном массиве
5. Поиск в хеш-таблице
6. Построить таблицу/график зависимости времени от N

Контрольные вопросы

1. Какое условие необходимо для применения двоичного поиска?
2. Какова сложность двоичного поиска? Почему она логарифмическая?
3. В чём разница между итеративной и рекурсивной реализацией двоичного поиска?
4. Чем опасен вариант mid = (left + right) / 2? Как правильно вычислять середину?
5. Что такое хеш-таблица? Из каких компонентов она состоит?
6. Какие требования предъявляются к хеш-функции?
7. Что такое коллизия в хеш-таблице? Как с ней бороться?
8. В чём отличие метода цепочек от открытой адресации?

9. Что такое первичная и вторичная кластеризация?
10. Какова средняя и худшая сложность операций в хеш-таблице?
11. Когда эффективнее использовать двоичный поиск, а когда - хеш-таблицу?
12. Что такое коэффициент загрузки хеш-таблицы? Почему он важен?
13. Как работает рехеширование (динамическое расширение хеш-таблицы)?
14. Как выбрать размер хеш-таблицы? Почему часто выбирают простое число?
15. Как реализовать хеш-функцию для строк?

Требования к отчёту

Титульный лист

Цель работы

Теоретическая часть (двоичный поиск, хеш-таблицы, методы разрешения коллизий)

Постановка задачи (согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода)

Сравнительная таблица производительности

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Критерии оценивания

Максимальный балл: 100

№	Критерий	Баллы
1	**Двоичный поиск**	**25**
1.1	Реализация алгоритма	10

№	Критерий	Баллы
1.2	Корректность на граничных случаях	10
1.3	Обработка ошибок (элемент не найден)	5
2	**Хеш-таблица**	**40**
2.1	Реализация хеш-функции	10
2.2	Реализация метода разрешения коллизий	15
2.3	Реализация insert, find, remove	15
3	**Сравнительный анализ**	**20**
3.1	Измерение времени выполнения	10
3.2	Таблица/график сравнения	5
3.3	Выводы о производительности	5
4	**Оформление и отчёт**	**15**
4.1	Код отформатирован, есть комментарии	5
4.2	Есть ответы на контрольные вопросы	5

№	Критерий	Баллы
4.3	Ссылка на Git-репозиторий	5
Итого		**100**

Штрафные баллы

Нарушение	Штраф
Неверная сложность алгоритма	-10
Плохая хеш-функция (много коллизий)	-10
Утечка памяти (в методе цепочек)	-10
Нет сравнительного анализа	-10
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №16

Бинарное дерево. Обходы. BST

Цель работы:

1. Реализовать структуру данных «бинарное дерево» и изучить различные способы его обхода (прямой, симметричный, обратный, уровневый)
2. Реализовать бинарное дерево поиска (BST) с операциями вставки, удаления и поиска
3. Освоить применение рекурсии для работы с деревьями
4. Проанализировать сложность операций в BST
5. Научиться визуализировать структуру дерева

Теоретическое обоснование

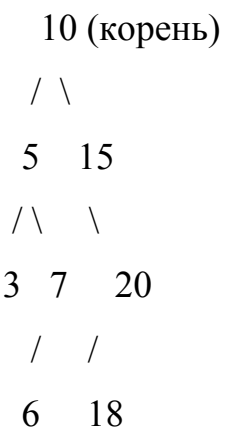
1. Бинарное дерево

Бинарное дерево - это иерархическая структура данных, в которой каждый узел имеет не более двух потомков: левый и правый.

Терминология:

Термин	Определение
Корень (root)	Верхний узел дерева
Лист (leaf)	Узел, не имеющий потомков
Внутренний узел	Узел, имеющий хотя бы одного потомка
Высота дерева	Максимальное расстояние от корня до листа
Уровень (level)	Расстояние от корня (корень - уровень 0)

Визуализация бинарного дерева:



2. Обходы бинарного дерева

Вид обхода	Порядок посещения	Результат для дерева выше
Прямой (pre-order)	Корень → Левый → Правый	10, 5, 3, 7, 6, 15, 20, 18
Симметричный (in-order)	Левый → Корень → Правый	3, 5, 6, 7, 10, 15, 18, 20
Обратный (post-order)	Левый → Правый → Корень	3, 6, 7, 5, 18, 20, 15, 10
Уровневый (level-order)	По уровням слева направо	10, 5, 15, 3, 7, 20, 6, 18

```
// Прямой обход (pre-order)
void preOrder(Node* node) {
    if (node == nullptr) return;
    cout << node->data << " ";
    preOrder(node->left);
```

```
    preOrder(node->right);
}

// Симметричный обход (in-order)
void inOrder(Node* node) {
    if (node == nullptr) return;
    inOrder(node->left);
    cout << node->data << " ";
    inOrder(node->right);
}

// Обратный обход (post-order)
void postOrder(Node* node) {
    if (node == nullptr) return;
    postOrder(node->left);
    postOrder(node->right);
    cout << node->data << " ";
}

// Уровневый обход (level-order) с использованием очереди
void levelOrder(Node* root) {
    if (root == nullptr) return;
    queue<Node*> q;
    q.push(root);
    while (!q.empty()) {
        Node* current = q.front();
        q.pop();
        cout << current->data << " ";
        if (current->left) q.push(current->left);
```

```
        if (current->right) q.push(current->right);
    }
}
```

3. Бинарное дерево поиска (BST)

Бинарное дерево поиска (BST) - это бинарное дерево, в котором для каждого узла выполняется условие:

Все ключи в левом поддереве < ключ узла < все ключи в правом поддереве

Свойства BST:

Симметричный обход даёт отсортированную последовательность

Поиск, вставка и удаление в среднем выполняются за O(log n)

Визуализация BST:

```
      50
     /  \
    30   70
   / \   / \
  20 40 60 80
```

4. Операции с BST

Поиск O(log n) в среднем

```
Node* search(Node* root, int key) {
    if (root == nullptr || root->data == key) return root;
    if (key < root->data) return search(root->left, key);
    return search(root->right, key);
}
```

Вставка O(log n) в среднем

cpp

```
Node* insert(Node* root, int key) {
    if (root == nullptr) return new Node(key);
    if (key < root->data) root->left = insert(root->left, key);
    else if (key > root->data) root->right = insert(root->right, key);
    return root;
}
```

Удаление O(log n) в среднем

Удаление узла из BST имеет три случая:

Случай	Описание	Действие
1	Удаляемый узел - лист	Просто удаляем
2	Узел имеет одного потомка	Заменяем удаляемый узел его потомком
3	Узел имеет двух потомков	Находим минимальный элемент в правом поддереве (или максимальный в левом), заменяем значение, удаляем тот узел

```
Node* findMin(Node* root) {
    while (root->left != nullptr) root = root->left;
    return root;
}

Node* remove(Node* root, int key) {
```

```
if (root == nullptr) return nullptr;

if (key < root->data) root->left = remove(root->left, key);
else if (key > root->data) root->right = remove(root->right, key);
else {
    // Случай 1: лист
    if (root->left == nullptr && root->right == nullptr) {
        delete root;
        return nullptr;
    }
    // Случай 2: один потомок
    else if (root->left == nullptr) {
        Node* temp = root->right;
        delete root;
        return temp;
    }
    else if (root->right == nullptr) {
        Node* temp = root->left;
        delete root;
        return temp;
    }
    // Случай 3: два потомка
    else {
        Node* minNode = findMin(root->right);
        root->data = minNode->data;
        root->right = remove(root->right, minNode->data);
    }
}
return root;
```

```
}
```

5. Вычисление характеристик дерева

```
// Высота дерева
int height(Node* root) {
    if (root == nullptr) return 0;
    return 1 + max(height(root->left), height(root->right));
}

// Количество узлов
int countNodes(Node* root) {
    if (root == nullptr) return 0;
    return 1 + countNodes(root->left) + countNodes(root->right);
}

// Количество листьев
int countLeaves(Node* root) {
    if (root == nullptr) return 0;
    if (root->left == nullptr && root->right == nullptr) return 1;
    return countLeaves(root->left) + countLeaves(root->right);
}
```

Примеры выполнения практической части

Пример 1. Бинарное дерево и обходы (базовый уровень)

Задача: Создать бинарное дерево и выполнить все четыре обхода.

```
#include <iostream>
#include <queue>
```

```
using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

class BinaryTree {
private:
    Node* root;

    void preOrder(Node* node) {
        if (node == nullptr) return;
        cout << node->data << " ";
        preOrder(node->left);
        preOrder(node->right);
    }

    void inOrder(Node* node) {
        if (node == nullptr) return;
        inOrder(node->left);
        cout << node->data << " ";
        inOrder(node->right);
    }

    void postOrder(Node* node) {
```

```
        if (node == nullptr) return;
        postOrder(node->left);
        postOrder(node->right);
        cout << node->data << " ";
    }

    void destroy(Node* node) {
        if (node == nullptr) return;
        destroy(node->left);
        destroy(node->right);
        delete node;
    }

public:
    BinaryTree() : root(nullptr) {}

    ~BinaryTree() { destroy(root); }

    void buildSample() {
        root = new Node(10);
        root->left = new Node(5);
        root->right = new Node(15);
        root->left->left = new Node(3);
        root->left->right = new Node(7);
        root->right->right = new Node(20);
        root->left->right->left = new Node(6);
        root->right->right->left = new Node(18);
    }
```

```
    void preOrder() { preOrder(root); cout << endl; }
    void inOrder() { inOrder(root); cout << endl; }
    void postOrder() { postOrder(root); cout << endl; }

    void levelOrder() {
        if (root == nullptr) return;
        queue<Node*> q;
        q.push(root);
        while (!q.empty()) {
            Node* current = q.front();
            q.pop();
            cout << current->data << " ";
            if (current->left) q.push(current->left);
            if (current->right) q.push(current->right);
        }
        cout << endl;
    }
};

int main() {
    BinaryTree tree;
    tree.buildSample();

    cout << "Прямой обход (pre-order):   ";
    tree.preOrder();
    cout << "Симметричный (in-order):    ";
    tree.inOrder();
    cout << "Обратный (post-order):      ";
    tree.postOrder();
```

```
    cout << "Уровневый (level-order):    ";
    tree.levelOrder();

    return 0;
}
```

Ожидаемый вывод:

```
Прямой обход (pre-order):   10 5 3 7 6 15 20 18
Симметричный (in-order):    3 5 6 7 10 15 18 20
Обратный (post-order):      3 6 7 5 18 20 15 10
Уровневый (level-order):    10 5 15 3 7 20 6 18
```

Пример 2. Бинарное дерево поиска (средний уровень)

Задача: Реализовать BST с операциями вставки, поиска, удаления и вывода.

```
#include <iostream>
#include <queue>

using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

class BST {
```

```
private:
    Node* root;

    Node* insert(Node* node, int key) {
        if (node == nullptr) return new Node(key);
        if (key < node->data) node->left = insert(node->left, key);
        else if (key > node->data) node->right = insert(node->right, key);
        return node;
    }

    Node* search(Node* node, int key) {
        if (node == nullptr || node->data == key) return node;
        if (key < node->data) return search(node->left, key);
        return search(node->right, key);
    }

    Node* findMin(Node* node) {
        while (node->left != nullptr) node = node->left;
        return node;
    }

    Node* remove(Node* node, int key) {
        if (node == nullptr) return nullptr;

        if (key < node->data) node->left = remove(node->left, key);
        else if (key > node->data) node->right = remove(node->right, key);
        else {
            if (node->left == nullptr && node->right == nullptr) {
                delete node;
```

```
            return nullptr;
        }
        else if (node->left == nullptr) {
            Node* temp = node->right;
            delete node;
            return temp;
        }
        else if (node->right == nullptr) {
            Node* temp = node->left;
            delete node;
            return temp;
        }
        else {
            Node* minNode = findMin(node->right);
            node->data = minNode->data;
            node->right = remove(node->right, minNode->data);
        }
    }
    return node;
}

void inOrder(Node* node) {
    if (node == nullptr) return;
    inOrder(node->left);
    cout << node->data << " ";
    inOrder(node->right);
}

void destroy(Node* node) {
```

```
        if (node == nullptr) return;
        destroy(node->left);
        destroy(node->right);
        delete node;
    }

    int height(Node* node) {
        if (node == nullptr) return 0;
        return 1 + max(height(node->left), height(node->right));
    }

public:
    BST() : root(nullptr) {}

    ~BST() { destroy(root); }

    void insert(int key) { root = insert(root, key); }

    bool search(int key) { return search(root, key) != nullptr; }

    void remove(int key) { root = remove(root, key); }

    void print() {
        inOrder(root);
        cout << endl;
    }

    int getHeight() { return height(root); }
};
```

```
int main() {
    BST tree;

    tree.insert(50);
    tree.insert(30);
    tree.insert(70);
    tree.insert(20);
    tree.insert(40);
    tree.insert(60);
    tree.insert(80);

    cout << "Симметричный обход BST: ";
    tree.print();
    cout << "Высота дерева: " << tree.getHeight() << endl;

    cout << "Поиск 40: " << (tree.search(40) ? "найден" : "не найден") <<
endl;
    cout << "Поиск 100: " << (tree.search(100) ? "найден" : "не найден") <<
endl;

    tree.remove(30);
    cout << "После удаления 30: ";
    tree.print();

    return 0;
}
```

Ожидаемый вывод:

Симметричный обход BST: 20 30 40 50 60 70 80

Высота дерева: 3

Поиск 40: найден

Поиск 100: не найден

После удаления 30: 20 40 50 60 70 80

Индивидуальные задания

Часть 1. Бинарное дерево и обходы

Каждый вариант содержит задание реализовать бинарное дерево с указанными операциями:

№ вар.	Обязательные операции	Дополнительное задание
1	preOrder, inOrder, postOrder	Вычислить количество узлов
2	preOrder, inOrder, postOrder	Вычислить количество листьев
3	preOrder, inOrder, postOrder	Вычислить высоту дерева
4	preOrder, inOrder, postOrder	Найти максимальный элемент
5	preOrder, inOrder, postOrder	Найти минимальный элемент
6	preOrder, inOrder, postOrder, levelOrder	Найти сумму всех элементов

№ вар.	Обязательные операции	Дополнительное задание
7	preOrder, inOrder, postOrder, levelOrder	Найти среднее арифметическое
8	preOrder, inOrder, postOrder, levelOrder	Проверить, является ли дерево вырожденным
9	preOrder, inOrder, postOrder, levelOrder	Проверить, является ли дерево полным
10	preOrder, inOrder, postOrder, levelOrder	Проверить, является ли дерево совершенным
11	preOrder, inOrder, postOrder, levelOrder	Построить зеркальное отображение (swap левого и правого)
12	preOrder, inOrder, postOrder, levelOrder	Найти путь от корня до заданного узла
13	preOrder, inOrder, postOrder, levelOrder	Найти наибольшую сумму на уровне
14	preOrder, inOrder, postOrder, levelOrder	Найти диаметр дерева (максимальное расстояние между

№ вар.	Обязательные операции	Дополнительное задание
		узлами)
15	preOrder, inOrder, postOrder, levelOrder	Проверить, является ли дерево сбалансированным (разница высот ≤ 1)

Часть 2. Бинарное дерево поиска (BST)

Каждый вариант содержит задание реализовать BST с указанными операциями:

№ вар.	Обязательные операции	Дополнительное задание
1	insert, search, inOrder	Удаление (все случаи)
2	insert, search, inOrder	Найти k-й наименьший элемент
3	insert, search, inOrder	Найти k-й наибольший элемент
4	insert, search, inOrder	Найти наименьший общий предок (LCA)
5	insert, search, inOrder	Найти преемника (следующий по порядку)

№ вар.	Обязательные операции	Дополнительное задание
6	insert, search, inOrder	Найти предшественника (предыдущий по порядку)
7	insert, search, inOrder, remove	Визуализация дерева в консоли
8	insert, search, inOrder, remove	Проверка, является ли дерево BST
9	insert, search, inOrder, remove	Построение BST из отсортированного массива (сбалансированно)
10	insert, search, inOrder, remove	Построение BST из префиксного обхода
11	insert, search, inOrder, remove	Восстановление BST из двух переставленных узлов
12	insert, search, inOrder, remove	Подсчёт узлов в заданном диапазоне
13	insert, search, inOrder, remove	Вычислить сумму узлов в заданном диапазоне
14	insert, search, inOrder, remove	Объединить два BST

№ вар.	Обязательные операции	Дополнительное задание
15	insert, search, inOrder, remove	Проверить, является ли BST сбалансированным

Часть 3. Применение BST

Задание для всех вариантов:

Реализовать программу, которая:

1. Создаёт BST из последовательности случайных чисел (N = 20)
2. Выполняет симметричный обход и выводит отсортированную последовательность
3. Находит минимальный и максимальный элемент
4. Находит высоту дерева
5. Удаляет элемент, введённый пользователем, и выводит дерево после удаления

Контрольные вопросы

1. Что такое бинарное дерево? Из каких элементов оно состоит?
2. В чём отличие бинарного дерева от бинарного дерева поиска (BST)?
3. Какие существуют виды обхода бинарного дерева? В чём их особенности?
4. Какой обход даёт отсортированную последовательность для BST?
5. Какова сложность операций поиска, вставки и удаления в BST? От чего она зависит?
6. Что происходит с BST, если вставлять элементы в отсортированном порядке?
7. Какие три случая нужно рассмотреть при удалении узла из BST?
8. Как найти наименьший элемент в BST? Наибольший?
9. Как вычислить высоту бинарного дерева рекурсивно?
10. Что такое наименьший общий предок (LCA) двух узлов в дереве?
11. Как проверить, является ли бинарное дерево BST?
12. Что такое сбалансированное дерево? Почему это важно для производительности?

13. Как преобразовать несбалансированное BST в сбалансированное?
14. Как восстановить дерево по массиву обхода (in-order + pre-order)?
15. Как реализовать итеративный (не рекурсивный) обход бинарного дерева?

Требования к отчёту

Титульный лист

Цель работы

Теоретическая часть (бинарное дерево, обходы, BST)

Постановка задачи (согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода, визуализация дерева)

Анализ сложности операций

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Критерии оценивания

Максимальный балл: 100

№	Критерий	Баллы
1	Бинарное дерево и обходы	30
1.1	Структура узла и класс дерева	10
1.2	Реализация всех обходов	15
1.3	Вычисление характеристик дерева	5
2	Бинарное дерево поиска (BST)	40

№	Критерий	Баллы
2.1	Реализация вставки	10
2.2	Реализация поиска	10
2.3	Реализация удаления (все три случая)	15
2.4	Корректная работа с памятью (деструктор)	5
3	Применение и анализ	15
3.1	Демонстрационная программа	5
3.2	Анализ сложности операций	5
3.3	Выводы и сравнения	5
4	Оформление и отчёт	15
4.1	Код отформатирован, есть комментарии	5
4.2	Есть ответы на контрольные вопросы	5
4.3	Ссылка на Git-репозиторий	5
Итого		100

Штрафные баллы

Нарушение	Штраф
Некорректное удаление узла с двумя потомками	-15
Утечка памяти (не освобождены узлы)	-15
Некорректные обходы дерева	-10
Нет обработки пустого дерева	-5
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №17

AVL-дерево. Префиксное дерево (Trie)

Цель работы:

1. Реализовать самобалансирующееся AVL-дерево с операциями вставки, удаления и поиска

2. Освоить механизмы балансировки: малые и большие повороты (правый, левый, правый-левый, левый-правый)
3. Изучить структуру данных «префиксное дерево» (Trie) для эффективного хранения и поиска строк
4. Реализовать операции вставки, поиска и удаления в Trie
5. Сравнить производительность AVL-дерева, BST и Trie для различных типов данных

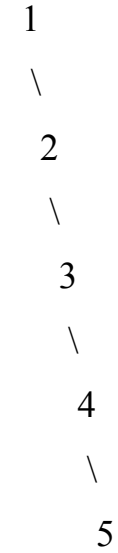
Теоретическое обоснование

1. Проблема вырождения BST

Обычное бинарное дерево поиска (BST) может выродиться в связный список при вставке элементов в отсортированном порядке.

Вырожденное дерево (худший случай):

Вставка: 1, 2, 3, 4, 5



Поиск 5 требует 5 шагов → O(n)

2. AVL-дерево

AVL-дерево - это самобалансирующееся бинарное дерево поиска, в котором для каждого узла выполняется условие:

|высота(левое поддерево) - высота(правое поддерево)| ≤ 1

Фактор баланса:

```
int balanceFactor(Node* node) {
    return height(node->left) - height(node->right);
}
```

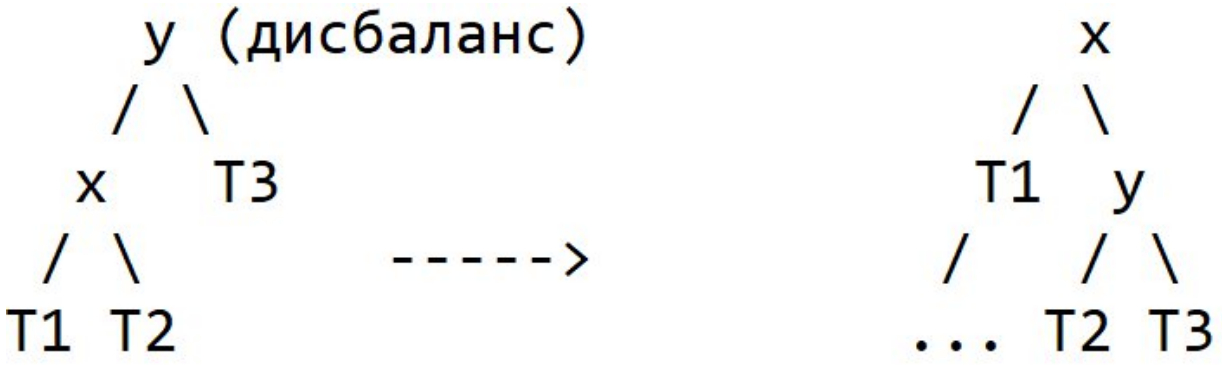
Фактор баланса	Состояние	Действие
-1, 0, 1	Норма	Ничего
2	Левое поддерево выше	Правый поворот
-2	Правое поддерево выше	Левый поворот

3. Повороты

Правый поворот (Right Rotation)

Применяется при: Фактор баланса > 1 и левый потомок имеет фактор баланса ≥ 0

Визуализация:

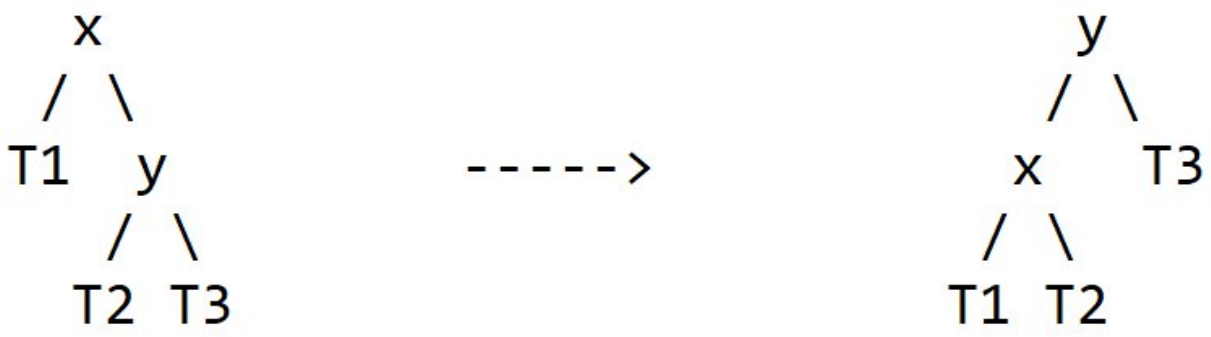



```
Node* rightRotate(Node* y) {
    Node* x = y->left;
    Node* T2 = x->right;

    x->right = y;
    y->left = T2;

    updateHeight(y);
    updateHeight(x);

    return x;
}
```

Левый поворот (Left Rotation)

Применяется при: Фактор баланса < -1 и правый потомок имеет фактор баланса ≤ 0

Визуализация:

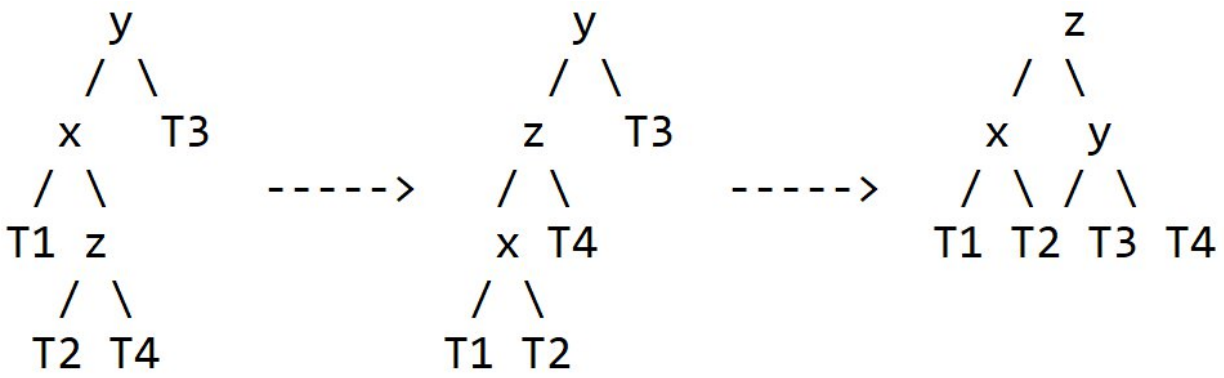



```
Node* leftRotate(Node* x) {
    Node* y = x->right;
    Node* T2 = y->left;

    y->left = x;
    x->right = T2;

    updateHeight(x);
    updateHeight(y);

    return y;
}
```

Левый-правый поворот (Left-Right Rotation)

Применяется при: Фактор баланса > 1 и левый потомок имеет фактор баланса < 0

Визуализация:

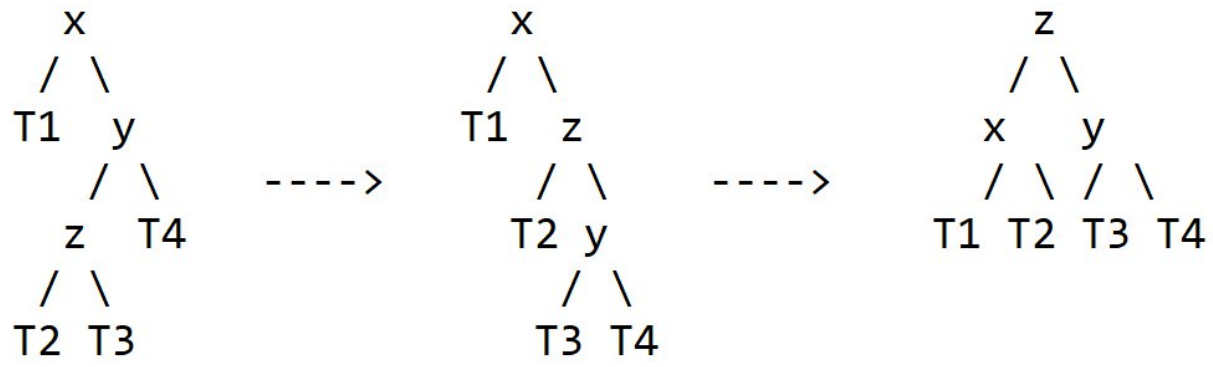


```
Node* leftRightRotate(Node* y) {
    y->left = leftRotate(y->left);
    return rightRotate(y);
}
```

Правый-левый поворот (Right-Left Rotation)

Применяется при: Фактор баланса < -1 и правый потомок имеет фактор баланса > 0

Визуализация:



```
Node* rightLeftRotate(Node* x) {
    x->right = rightRotate(x->right);
    return leftRotate(x);
}
```

4. Префиксное дерево (Trie)

Trie (префиксное дерево) - это структура данных для хранения строк, позволяющая быстро выполнять операции поиска, вставки и удаления.

Визуализация Trie для слов: "cat", "car", "dog"

Структура узла Trie:

```
struct TrieNode {
    TrieNode* children[26];  // для букв a-z
    bool isEndOfWord;        // маркер конца слова

    TrieNode() : isEndOfWord(false) {
        for (int i = 0; i < 26; i++) {
            children[i] = nullptr;
        }
    }
};
```

5. Операции с Trie

Вставка

```
void insert(const string& word) {
    TrieNode* current = root;
    for (char ch : word) {
        int index = ch - 'a';
        if (current->children[index] == nullptr) {
            current->children[index] = new TrieNode();
        }
        current = current->children[index];
    }
    current->isEndOfWord = true;
}
```

Поиск

```
bool search(const string& word) {
    TrieNode* current = root;
    for (char ch : word) {
        int index = ch - 'a';
        if (current->children[index] == nullptr) return false;
        current = current->children[index];
    }
    return current->isEndOfWord;
}
```

Поиск по префиксу

```
bool startsWith(const string& prefix) {
```

```
    TrieNode* current = root;
    for (char ch : prefix) {
        int index = ch - 'a';
        if (current->children[index] == nullptr) return false;
        current = current->children[index];
    }
    return true;  // есть хотя бы одно слово с данным префиксом
}
```

6. Сравнение структур данных

Характеристика	BST	AVL	Trie
Поиск (средний)	O(log n)	O(log n)	O(L) - длина строки
Вставка (средний)	O(log n)	O(log n)	O(L)
Удаление (средний)	O(log n)	O(log n)	O(L)
Худший случай	O(n)	O(log n)	O(L)
Память	O(n)	O(n)	O(алфавит × длина)
Строковые операции	Медленные (сравнение)	Медленные	Быстрые

Характеристика	BST	AVL	Trie
Автодополнение	Нет	Нет	Да

Примеры выполнения практической части

Пример 1. AVL-дерево - вставка и повороты

Задача: Реализовать AVL-дерево с операциями вставки и визуализацией балансировки.

```
#include <iostream>
#include <algorithm>

using namespace std;

struct AVLNode {
    int key;
    AVLNode* left;
    AVLNode* right;
    int height;

    AVLNode(int k) : key(k), left(nullptr), right(nullptr), height(1) {}
};

class AVLTree {
private:
    AVLNode* root;
```

```
int height(AVLNode* node) {
    return node ? node->height : 0;
}

int balanceFactor(AVLNode* node) {
    return height(node->left) - height(node->right);
}

void updateHeight(AVLNode* node) {
    node->height = 1 + max(height(node->left), height(node->right));
}

// Правый поворот
AVLNode* rightRotate(AVLNode* y) {
    AVLNode* x = y->left;
    AVLNode* T2 = x->right;

    x->right = y;
    y->left = T2;

    updateHeight(y);
    updateHeight(x);

    return x;
}

// Левый поворот
AVLNode* leftRotate(AVLNode* x) {
    AVLNode* y = x->right;
```

```
    AVLNode* T2 = y->left;

    y->left = x;
    x->right = T2;

    updateHeight(x);
    updateHeight(y);

    return y;
}

AVLNode* balance(AVLNode* node) {
    if (node == nullptr) return nullptr;

    updateHeight(node);
    int bf = balanceFactor(node);

    // Левое поддерево выше
    if (bf > 1) {
        if (balanceFactor(node->left) < 0) {
            // Левый-правый поворот
            node->left = leftRotate(node->left);
        }
        // Правый поворот
        return rightRotate(node);
    }
    // Правое поддерево выше
    else if (bf < -1) {
        if (balanceFactor(node->right) > 0) {
```

```
            // Правый-левый поворот
            node->right = rightRotate(node->right);
        }
        // Левый поворот
        return leftRotate(node);
    }

    return node;
}

AVLNode* insert(AVLNode* node, int key) {
    if (node == nullptr) return new AVLNode(key);

    if (key < node->key) {
        node->left = insert(node->left, key);
    } else if (key > node->key) {
        node->right = insert(node->right, key);
    } else {
        return node;  // дубликаты не вставляем
    }

    return balance(node);
}

void inOrder(AVLNode* node) {
    if (node == nullptr) return;
    inOrder(node->left);
    cout << node->key << " ";
    inOrder(node->right);
```

```
    }

    void printTree(AVLNode* node, int level = 0) {
        if (node == nullptr) return;
        printTree(node->right, level + 1);
        for (int i = 0; i < level; i++) cout << "    ";
        cout << node->key << " (h=" << node->height << ", bf=" <<
balanceFactor(node) << ")" << endl;
        printTree(node->left, level + 1);
    }

    void destroy(AVLNode* node) {
        if (node == nullptr) return;
        destroy(node->left);
        destroy(node->right);
        delete node;
    }

public:
    AVLTree() : root(nullptr) {}

    ~AVLTree() { destroy(root); }

    void insert(int key) { root = insert(root, key); }

    void print() {
        cout << "AVL-дерево:" << endl;
        printTree(root);
        cout << endl;
```

```
    }

    void printInOrder() {
        inOrder(root);
        cout << endl;
    }
};

int main() {
    AVLTree tree;

    cout << "=== Вставка элементов в AVL-дерево ===" << endl;

    // Элементы, вызывающие балансировку
    vector<int> values = {10, 20, 30, 40, 50, 25};

    for (int val : values) {
        cout << "\nВставка " << val << ":" << endl;
        tree.insert(val);
        tree.print();
    }

    cout << "\nСимметричный обход (отсортированный порядок): ";
    tree.printInOrder();

    return 0;
}
```

Ожидаемый вывод (визуализация):

=== Вставка элементов в AVL-дерево ===

Вставка 10:

AVL-дерево:

10 (h=1, bf=0)

Вставка 20:

AVL-дерево:

20 (h=2, bf=-1)

10 (h=1, bf=0)

Вставка 30:

AVL-дерево:

20 (h=2, bf=0)

10 (h=1, bf=0)

30 (h=1, bf=0)

...

Пример 2. Префиксное дерево (Trie)

Задача: Реализовать Trie с операциями вставки, поиска и поиска по префиксу.

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;
```

```
class Trie {
private:
    struct TrieNode {
        TrieNode* children[26];
        bool isEndOfWord;

        TrieNode() : isEndOfWord(false) {
            for (int i = 0; i < 26; i++) {
                children[i] = nullptr;
            }
        }
    };

    TrieNode* root;

    void destroy(TrieNode* node) {
        if (node == nullptr) return;
        for (int i = 0; i < 26; i++) {
            destroy(node->children[i]);
        }
        delete node;
    }

public:
    Trie() : root(new TrieNode()) {}

    ~Trie() { destroy(root); }
```

```
void insert(const string& word) {
    TrieNode* current = root;
    for (char ch : word) {
        int index = ch - 'a';
        if (current->children[index] == nullptr) {
            current->children[index] = new TrieNode();
        }
        current = current->children[index];
    }
    current->isEndOfWord = true;
    cout << "Слово \"" << word << "\" добавлено" << endl;
}

bool search(const string& word) {
    TrieNode* current = root;
    for (char ch : word) {
        int index = ch - 'a';
        if (current->children[index] == nullptr) return false;
        current = current->children[index];
    }
    return current->isEndOfWord;
}

bool startsWith(const string& prefix) {
    TrieNode* current = root;
    for (char ch : prefix) {
        int index = ch - 'a';
        if (current->children[index] == nullptr) return false;
        current = current->children[index];
```

```
        }
        return true;
    }

    void collectWords(TrieNode* node, string current, vector<string>& result) {
        if (node->isEndOfWord) {
            result.push_back(current);
        }
        for (int i = 0; i < 26; i++) {
            if (node->children[i] != nullptr) {
                collectWords(node->children[i], current + char('a' + i), result);
            }
        }
    }

    vector<string> getAllWords() {
        vector<string> result;
        collectWords(root, "", result);
        return result;
    }

    vector<string> getWordsByPrefix(const string& prefix) {
        TrieNode* current = root;
        for (char ch : prefix) {
            int index = ch - 'a';
            if (current->children[index] == nullptr) return {};
            current = current->children[index];
        }
```

```
        vector<string> result;
        collectWords(current, prefix, result);
        return result;
    }
};

int main() {
    Trie trie;

    cout << "=== Префиксное дерево (Trie) ===" << endl;

    trie.insert("cat");
    trie.insert("car");
    trie.insert("dog");
    trie.insert("cart");
    trie.insert("catfish");

    cout << "\nПоиск слов:" << endl;
    cout << "search(\"cat\"): " << (trie.search("cat") ? "найдено" : "не найдено") << endl;
    cout << "search(\"car\"): " << (trie.search("car") ? "найдено" : "не найдено") << endl;
    cout << "search(\"cattle\"): " << (trie.search("cattle") ? "найдено" : "не найдено") << endl;

    cout << "\nПоиск по префиксу:" << endl;
    cout << "startsWith(\"ca\"): " << (trie.startsWith("ca") ? "true" : "false") << endl;
```

```
    cout << "startsWith(\"do\"): " << (trie.startsWith("do") ? "true" : "false")
<< endl;
    cout << "startsWith(\"ba\"): " << (trie.startsWith("ba") ? "true" : "false")
<< endl;

    cout << "\nВсе слова в Trie:" << endl;
    vector<string> allWords = trie.getAllWords();
    for (const auto& word : allWords) {
        cout << "  " << word << endl;
    }

    cout << "\nСлова с префиксом \"ca\":" << endl;
    vector<string> caWords = trie.getWordsByPrefix("ca");
    for (const auto& word : caWords) {
        cout << "  " << word << endl;
    }

    return 0;
}
```

Ожидаемый вывод:

```
=== Префиксное дерево (Trie) ===
Слово "cat" добавлено
Слово "car" добавлено
Слово "dog" добавлено
Слово "cart" добавлено
Слово "catfish" добавлено

Поиск слов:
```

search("cat"): найдено

search("car"): найдено

search("cattle"): не найдено

Поиск по префиксу:

startsWith("ca"): true

startsWith("do"): true

startsWith("ba"): false

Все слова в Trie:

car

cart

cat

catfish

dog

Слова с префиксом "ca":

car

cart

cat

catfish

Индивидуальные задания

Часть 1. AVL-дерево

Каждый вариант содержит задание реализовать AVL-дерево с указанными операциями:

№ вар.	Обязательные операции	Дополнительное задание
1	Вставка, балансировка	Найти минимальный элемент
2	Вставка, балансировка	Найти максимальный элемент
3	Вставка, балансировка	Поиск элемента
4	Вставка, балансировка, поиск	Удаление (лист)
5	Вставка, балансировка, поиск	Удаление (один потомок)
6	Вставка, балансировка, поиск	Удаление (два потомка)
7	Вставка, балансировка, поиск, удаление	Найти k-й наименьший элемент
8	Вставка, балансировка, поиск, удаление	Найти наименьший общий предок (LCA)
9	Вставка, балансировка, поиск, удаление	Подсчёт узлов в диапазоне
10	Вставка, балансировка, поиск, удаление	Визуализация дерева
1	Вставка, балансировка, поиск,	Проверка

№ вар.	Обязательные операции	Дополнительное задание
1	удаление	сбалансированности
2	Вставка, балансировка, поиск, удаление	Построение AVL из массива
3	Вставка, балансировка, поиск, удаление	Объединение двух AVL-деревьев
4	Вставка, балансировка, поиск, удаление	Восстановление AVL из отсортированного массива
5	Вставка, балансировка, поиск, удаление	Сравнение производительности с BST

Часть 2. Префиксное дерево (Trie)

Каждый вариант содержит задание реализовать Trie с указанными операциями:

№ вар.	Обязательные операции	Дополнительное задание
1	Вставка, поиск	Поиск по префиксу
2	Вставка, поиск	Удаление слова

№ вар.	Обязательные операции	Дополнительное задание
3	Вставка, поиск, поиск по префиксу	Получение всех слов
4	Вставка, поиск, поиск по префиксу	Автодополнение (топ-5)
5	Вставка, поиск, удаление	Подсчёт количества слов
6	Вставка, поиск, поиск по префиксу	Поиск самого длинного префикса
7	Вставка, поиск, поиск по префиксу	Проверка, есть ли слово с заданным префиксом
8	Вставка, поиск, удаление	Импорт слов из файла
9	Вставка, поиск, поиск по префиксу	Экспорт слов в файл
10	Вставка, поиск, удаление	Поддержка букв верхнего регистра
11	Вставка, поиск, поиск по префиксу	Поддержка цифр (0-9)
12	Вставка, поиск, удаление	Поиск слов по маске (с подстановкой '?')

№ вар.	Обязательные операции	Дополнительное задание
13	Вставка, поиск, поиск по префиксу	Сжатое префиксное дерево (Radix Tree)
14	Вставка, поиск, удаление	Частотный словарь (счётчик вхождений)
15	Вставка, поиск, поиск по префиксу	Автодополнение с учётом популярности

Часть 3. Сравнительный анализ

Задание для всех вариантов:

1. Реализовать AVL-дерево и обычное BST
2. Сравнить производительность вставки и поиска для N = 1000, 5000, 10000 элементов:
3. Случайные данные
4. Отсортированные данные (худший случай для BST)
5. Построить таблицу/график зависимости времени от N

Контрольные вопросы

1. AVL-дерево
2. Что такое AVL-дерево? В чём его отличие от обычного BST?
3. Что такое фактор баланса? Как он вычисляется?
4. Какие существуют типы поворотов в AVL-дереве?
5. Когда применяется правый поворот? Левый поворот?
6. Когда применяется левый-правый поворот? Правый-левый?
7. Какова сложность операций в AVL-дереве?
8. Почему AVL-дерево называется самобалансирующимся?
9. Как часто происходит балансировка при вставке?
10. Чем AVL-дерево отличается от красно-чёрного дерева?
11. В каких случаях AVL-дерево эффективнее BST?

12. Префиксное дерево (Trie)
13. Что такое Trie (префиксное дерево)? Для решения каких задач оно используется?
14. Какова сложность поиска в Trie? От чего она зависит?
15. Какая структура данных эффективнее для строк: Trie или хеш-таблица?
16. Как реализовать автодополнение с помощью Trie?
17. Каковы недостатки Trie? Какой объём памяти он требует?

Требования к отчёту

Титульный лист

Цель работы

Теоретическая часть (AVL-дерево, повороты, Trie)

Постановка задачи (согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода, визуализация)

Анализ сложности операций

Сравнительная таблица BST vs AVL vs Trie

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Критерии оценивания

Максимальный балл: 100

№	Критерий	Баллы
1	AVL-дерево	45
1.1	Реализация узла и высот	5
1.2	Реализация поворотов	15

№	Критерий	Баллы
1.3	Реализация балансировки	10
1.4	Реализация вставки	10
1.5	Реализация удаления (по варианту)	5
2	Префиксное дерево (Trie)	35
2.1	Структура узла Trie	5
2.2	Реализация вставки	10
2.3	Реализация поиска	10
2.4	Реализация дополнительных операций	10
3	Сравнительный анализ	10
3.1	Измерение времени выполнения	5
3.2	Таблица/график сравнения	5
4	Оформление и отчёт	10
4.1	Код отформатирован, есть комментарии	5
4.2	Ссылка на Git-репозиторий	5

№	Критерий	Баллы
Итого		100

Штрафные баллы

Нарушение	Штраф
Некорректная балансировка (нет поворотов)	-20
Утечка памяти (не освобождены узлы)	-15
Неверная реализация поиска по префиксу	-10
Нет сравнительного анализа	-10
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №18

Представление графов. DFS и BFS

Цель работы:

1. Изучить способы представления графов в памяти компьютера: матрица смежности и список смежности
2. Освоить алгоритмы обхода графа: поиск в глубину (DFS) и поиск в ширину (BFS)
3. Научиться применять DFS и BFS для решения практических задач (поиск компонент связности, кратчайших путей)
4. Проанализировать сложность алгоритмов для разных способов представления графа
5. Сравнить эффективность матрицы смежности и списка смежности

Теоретическое обоснование

1. Основные понятия теории графов

Граф - это совокупность вершин (узлов) и рёбер (связей) между ними.

Основные термины:

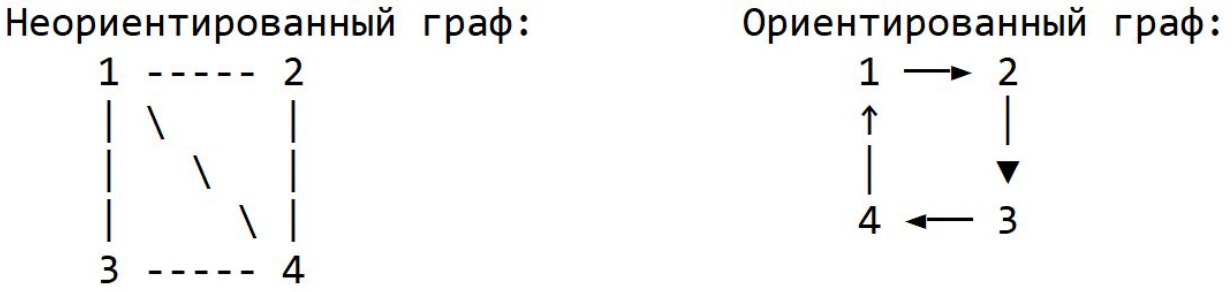
Термин	Определение
Вершина (vertex)	Узел графа
Ребро (edge)	Связь между двумя вершинами
Ориентированный граф	Рёбра имеют направление
Неориентированный граф	Рёбра не имеют направления
Взвешенный граф	Рёбра имеют веса (стоимость прохода)

Термин	Определение
Степень вершины	Количество рёбер, инцидентных вершине
Путь	Последовательность вершин, соединённых рёбрами
Цикл	Путь, начинающийся и заканчивающийся в одной вершине
Компонента связности	Максимальное множество вершин, любые две из которых соединены путём

Визуализация графа:

text

Неориентированный граф: Ориентированный граф:



Вершины: 1, 2, 3, 4
Рёбра: (1,2), (1,3), (1,4), (2,4), (3,4)

Вершины: 1, 2, 3, 4

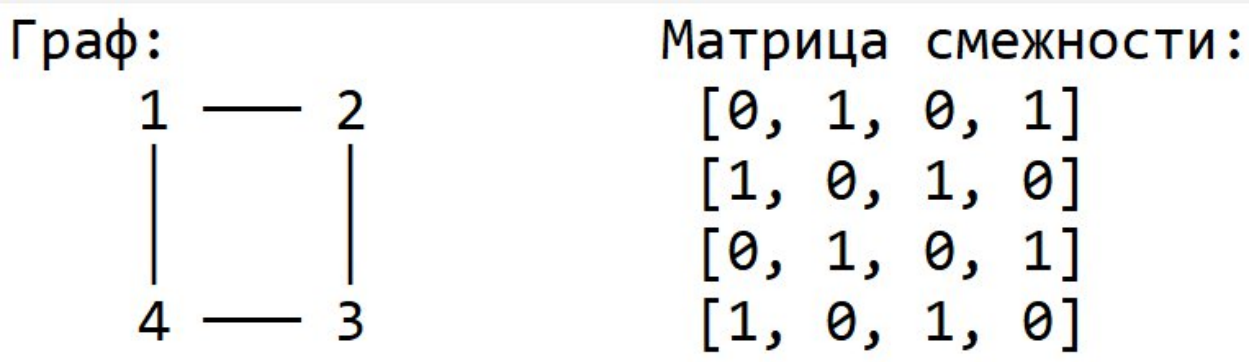
Рёбра: (1,2), (1,3), (1,4), (2,4), (3,4)

2. Способы представления графов

Матрица смежности (Adjacency Matrix)

Матрица смежности - это квадратная матрица размером V × V, где элемент matrix[i][j] = 1, если есть ребро из вершины i в вершину j.

Визуализация:




```
class GraphMatrix {
private:
    int V;  // количество вершин
    vector<vector<int>> adj;

public:
    GraphMatrix(int vertices) : V(vertices) {
        adj.resize(V, vector<int>(V, 0));
    }

    void addEdge(int u, int v) {
        adj[u][v] = 1;
        adj[v][u] = 1;  // для неориентированного графа
    }

    void print() {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                cout << adj[i][j] << " ";
            }
```

```
            cout << endl;
        }
    }
};
```

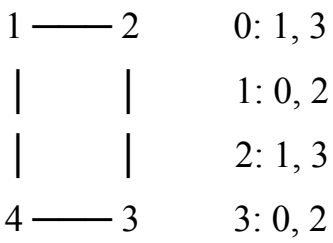
Список смежности (Adjacency List)

Список смежности - это массив списков, где для каждой вершины хранится список смежных с ней вершин.

Визуализация:

text

Граф: Список смежности:



0: 1, 3

1: 0, 2

2: 1, 3

3: 0, 2

```
class GraphList {
private:
    int V;  // количество вершин
    vector<vector<int>> adj;

public:
    GraphList(int vertices) : V(vertices) {
        adj.resize(V);
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
```

```
        adj[v].push_back(u);  // для неориентированного графа
    }

    void print() {
        for (int i = 0; i < V; i++) {
            cout << i << ": ";
            for (int v : adj[i]) {
                cout << v << " ";
            }
            cout << endl;
        }
    }
};
```

3. Сравнение способов представления

Характеристика	Матрица смежности	Список смежности
Память	$O(V^2)$	$O(V + E)$
Проверка наличия ребра (u,v)	O(1)	O(deg(u))
Перебор всех соседей вершины	O(V)	O(deg(u))
Добавление ребра	O(1)	O(1)

Характеристика	Матрица смежности	Список смежности
Удаление ребра	O(1)	O(deg(u))
Когда лучше	Плотные графы ($E \approx V^2$)	Разреженные графы ($E << V^2$)

4. Поиск в глубину (DFS - Depth-First Search)

Принцип работы: Идём «в глубину», пока не достигнем тупика, затем возвращаемся и исследуем другие пути.

Визуализация DFS:

Граф:

Порядок обхода DFS (начиная с 0):

0 → 1 → 2 → 4 → 3 → 5

(если соседи перебираются в порядке возрастания)

Стек вызовов:

push(0) → pop(0) → push(1,3) → pop(3) → push(5) → ...

```
void DFS(int start) {
    vector<bool> visited(V, false);
    DFSUtil(start, visited);
}

void DFSUtil(int v, vector<bool>& visited) {
    visited[v] = true;
    cout << v << " ";

    for (int neighbor : adj[v]) {
        if (!visited[neighbor]) {
            DFSUtil(neighbor, visited);
        }
    }
}
```

Рекурсивная реализация:

```
void DFS_recursive(int v, vector<bool>& visited) {
    visited[v] = true;
    cout << v << " ";

    for (int u : adj[v]) {
        if (!visited[u]) {
            DFS_recursive(u, visited);
        }
    }
}
```

Итеративная реализация (с использованием стека):

```
void DFS_iterative(int start) {
    vector<bool> visited(V, false);
    stack<int> st

    st.push(start);
    visited[start] = true;

    while (!st.empty()) {
        int v = st.top();
        st.pop();
        cout << v << " ";

        for (int u : adj[v]) {
            if (!visited[u]) {
                visited[u] = true;
                st.push(u);
            }
        }
    }
}
```

5. Поиск в ширину (BFS - Breadth-First Search)

Принцип работы: Сначала посещаем все соседние вершины, затем переходим к их соседям (по уровням).

Визуализация BFS:

text

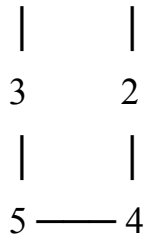
Граф:

```
0 —— 1
|    |
```

Порядок обхода BFS (начиная с 0):

0 → 1 → 3 → 2 → 5 → 4

(по уровням)



Очередь:

queue: [0] → pop(0) → push(1,3) → queue: [1,3]

pop(1) → push(2) → queue: [3,2]

pop(3) → push(5) → queue: [2,5]

pop(2) → push(4) → queue: [5,4]

...

```
void BFS(int start) {
    vector<bool> visited(V, false);
    queue<int> q;

    visited[start] = true;
    q.push(start);

    while (!q.empty()) {
        int v = q.front();
        q.pop();
        cout << v << " ";

        for (int u : adj[v]) {
            if (!visited[u]) {
                visited[u] = true;
                q.push(u);
            }
```

```
        }
    }
}
```

6. Применение DFS и BFS

Задача	DFS	BFS
Поиск компонент связности	+	+
Проверка наличия цикла	+	+
Топологическая сортировка	+	-
Поиск кратчайшего пути (невзвешенный граф)	-	+
Поиск мостов и точек сочленения	+	-
Поиск пути (любого)	+	+

7. Поиск компонент связности

```
int countConnectedComponents() {
    vector<bool> visited(V, false);
    int components = 0;

    for (int i = 0; i < V; i++) {
        if (!visited[i]) {
            components++;
            DFSUtil(i, visited);
```

```
        }
    }
    return components;
}
```

8. Поиск кратчайшего пути в невзвешенном графе (BFS)

```
vector<int> shortestPath(int start, int end) {
    vector<int> dist(V, -1);
    vector<int> prev(V, -1);
    queue<int> q;

    dist[start] = 0;
    q.push(start);

    while (!q.empty()) {
        int v = q.front();
        q.pop();

        for (int u : adj[v]) {
            if (dist[u] == -1) {
                dist[u] = dist[v] + 1;
                prev[u] = v;
                q.push(u);
            }
        }
    }

    // Восстановление пути
```

```
    vector<int> path;
    for (int v = end; v != -1; v = prev[v]) {
        path.push_back(v);
    }
    reverse(path.begin(), path.end());
    return path;
}
```

Примеры выполнения практической части

Пример 1. Создание графа и обходы (базовый уровень)

Задача: Создать граф, реализовать DFS и BFS, вывести порядок обхода.

```
#include <iostream>
#include <vector>
#include <stack>
#include <queue>

using namespace std;

class Graph {
private:
    int V;  // количество вершин
    vector<vector<int>> adj;  // список смежности

    void DFS_util(int v, vector<bool>& visited) {
        visited[v] = true;
        cout << v << " ";
```

```
        for (int u : adj[v]) {
            if (!visited[u]) {
                DFS_util(u, visited);
            }
        }
    }

public:
    Graph(int vertices) : V(vertices) {
        adj.resize(V);
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);  // неориентированный граф
    }

    void DFS(int start) {
        vector<bool> visited(V, false);
        cout << "DFS (рекурсивный): ";
        DFS_util(start, visited);
        cout << endl;
    }

    void DFS_iterative(int start) {
        vector<bool> visited(V, false);
        stack<int> st;
```

```
    st.push(start);
    visited[start] = true;

    cout << "DFS (итеративный): ";
    while (!st.empty()) {
        int v = st.top();
        st.pop();
        cout << v << " ";

        for (int u : adj[v]) {
            if (!visited[u]) {
                visited[u] = true;
                st.push(u);
            }
        }
    }
    cout << endl;
}

void BFS(int start) {
    vector<bool> visited(V, false);
    queue<int> q;

    visited[start] = true;
    q.push(start);

    cout << "BFS: ";
    while (!q.empty()) {
        int v = q.front();
```

```
            q.pop();
            cout << v << " ";

            for (int u : adj[v]) {
                if (!visited[u]) {
                    visited[u] = true;
                    q.push(u);
                }
            }
        }
        cout << endl;
    }

    void print() {
        for (int i = 0; i < V; i++) {
            cout << i << ": ";
            for (int u : adj[i]) {
                cout << u << " ";
            }
            cout << endl;
        }
    }
};

int main() {
    Graph g(6);

    g.addEdge(0, 1);
    g.addEdge(0, 3);
```

```
    g.addEdge(1, 2);
    g.addEdge(2, 4);
    g.addEdge(3, 5);
    g.addEdge(4, 5);

    cout << "=== Список смежности ===" << endl;
    g.print();

    cout << "\n=== Обход графа (начиная с вершины 0) ===" << endl;
    g.DFS(0);
    g.DFS_iterative(0);
    g.BFS(0);

    return 0;
}
```

Ожидаемый вывод:

=== Список смежности ===

0: 1 3

1: 0 2

2: 1 4

3: 0 5

4: 2 5

5: 3 4

=== Обход графа (начиная с вершины 0) ===

DFS (рекурсивный): 0 1 2 4 5 3

DFS (итеративный): 0 3 5 4 2 1

BFS: 0 1 3 2 5 4

Пример 2. Поиск компонент связности (средний уровень)

Задача: Определить количество компонент связности в графе.

```
#include <iostream>
#include <vector>

using namespace std;

class Graph {
private:
    int V;
    vector<vector<int>> adj;

    void DFS(int v, vector<bool>& visited) {
        visited[v] = true;
        for (int u : adj[v]) {
            if (!visited[u]) {
                DFS(u, visited);
            }
        }
    }

public:
    Graph(int vertices) : V(vertices) {
        adj.resize(V);
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
```

```
int countConnectedComponents() {
    vector<bool> visited(V, false);
    int components = 0;

    for (int i = 0; i < V; i++) {
        if (!visited[i]) {
            components++;
            DFS(i, visited);
        }
    }
    return components;
}

void printComponents() {
    vector<bool> visited(V, false);
    int componentNum = 1;

    for (int i = 0; i < V; i++) {
        if (!visited[i]) {
            cout << "Компонента " << componentNum++ << ": ";
            vector<int> comp;
            stack<int> st
            st.push(i);
            visited[i] = true;

            while (!st.empty()) {
                int v = st.top();
                st.pop();
```

```
                comp.push_back(v);

                for (int u : adj[v]) {
                    if (!visited[u]) {
                        visited[u] = true;
                        st.push(u);
                    }
                }
            }

            for (int v : comp) {
                cout << v << " ";
            }
            cout << endl;
        }
    }
};

int main() {
    // Граф с двумя компонентами связности
    Graph g(7);

    // Первая компонента: 0-1-2
    g.addEdge(0, 1);
    g.addEdge(1, 2);

    // Вторая компонента: 3-4-5-6
    g.addEdge(3, 4);
```

```
    g.addEdge(4, 5);
    g.addEdge(5, 6);

    cout << "=== Количество компонент связности: " << g.countConnectedComponents() << endl;
    g.printComponents();

    return 0;
}
```

Ожидаемый вывод:

=== Количество компонент связности: 2

Компонента 1: 0 2 1

Компонента 2: 3 6 5 4

Пример 3. Кратчайший путь в невзвешенном графе (продвинутый уровень)

Задача: Найти кратчайший путь между двумя вершинами с помощью BFS.

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>

using namespace std;

class Graph {
private:
    int V;
    vector<vector<int>> adj;
```

```
public:
    Graph(int vertices) : V(vertices) {
        adj.resize(V);
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    vector<int> shortestPath(int start, int end) {
        vector<int> dist(V, -1);
        vector<int> prev(V, -1);
        queue<int> q;

        dist[start] = 0;
        q.push(start);

        while (!q.empty()) {
            int v = q.front();
            q.pop();

            if (v == end) break;

            for (int u : adj[v]) {
                if (dist[u] == -1) {
                    dist[u] = dist[v] + 1;
                    prev[u] = v;
                    q.push(u);
```

```
            }
        }
    }

    // Восстановление пути
    vector<int> path;
    if (dist[end] == -1) return path;

    for (int v = end; v != -1; v = prev[v]) {
        path.push_back(v);
    }
    reverse(path.begin(), path.end());

    return path;
}

void printPath(int start, int end) {
    vector<int> path = shortestPath(start, end);
    if (path.empty()) {
        cout << "Путь между " << start << " и " << end << " не
существует" << endl;
        return;
    }

    cout << "Кратчайший путь от " << start << " до " << end << ": ";
    for (int i = 0; i < path.size(); i++) {
        cout << path[i];
        if (i < path.size() - 1) cout << " → ";
    }
```

```
        cout << " (длина: " << path.size() - 1 << ")" << endl;
    }
};

int main() {
    Graph g(8);

    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(2, 4);
    g.addEdge(3, 5);
    g.addEdge(4, 5);
    g.addEdge(5, 6);
    g.addEdge(6, 7);

    cout << "=== Поиск кратчайшего пути ===" << endl;
    g.printPath(0, 6);
    g.printPath(0, 7);
    g.printPath(1, 4);
    g.printPath(0, 5);

    return 0;
}
```

Ожидаемый вывод:

=== Поиск кратчайшего пути ===

Кратчайший путь от 0 до 6: 0 → 2 → 4 → 5 → 6 (длина: 4)

Кратчайший путь от 0 до 7: 0 → 2 → 4 → 5 → 6 → 7 (длина: 5)

Кратчайший путь от 1 до 4: 1 → 0 → 2 → 4 (длина: 3)

Кратчайший путь от 0 до 5: 0 → 2 → 4 → 5 (длина: 3)

Индивидуальные задания

Часть 1. Представление графов

Каждый вариант содержит задание реализовать граф с указанным способом представления:

№ вар.	Способ представления	Тип графа	Особенности
1	Список смежности	Неориентированный	Без весов
2	Матрица смежности	Неориентированный	Без весов
3	Список смежности	Ориентированный	Без весов
4	Матрица смежности	Ориентированный	Без весов
5	Список смежности	Неориентированный	Взвешенный
6	Матрица смежности	Неориентированный	Взвешенный
7	Список смежности	Ориентированный	Взвешенный
8	Матрица	Ориентирован	Взвешенный

№ вар.	Способ представления	Тип графа	Особенности
	смежности	ный	
9	Список смежности	Неориентированный	Разреженный
10	Матрица смежности	Неориентированный	Плотный
11	Список смежности	Ориентированный	С петлями
12	Матрица смежности	Ориентированный	С петлями
13	Список смежности	Неориентированный	С удалением рёбер
14	Матрица смежности	Неориентированный	С удалением рёбер
15	Список смежности + матрица	Неориентированный	Сравнение производительности

Часть 2. Обход графов (DFS и BFS)

Каждый вариант содержит задание реализовать обход графа и выполнить указанную задачу:

№ вар.	Алгоритмы	Задача
1	DFS (рекурсивный), BFS	Подсчитать количество вершин
2	DFS (рекурсивный), BFS	Найти все вершины, достижимые из заданной
3	DFS (рекурсивный), BFS	Найти компоненты связности
4	DFS (рекурсивный), BFS	Проверить, является ли граф связным
5	DFS (рекурсивный), BFS	Найти количество компонент связности
6	DFS (рекурсивный), BFS	Проверить наличие цикла
7	DFS (рекурсивный), BFS	Найти кратчайший путь (BFS)
8	DFS (рекурсивный), BFS	Найти расстояние между двумя вершинами
9	DFS (рекурсивный), BFS	Определить, есть ли путь между вершинами
10	DFS (итеративный), BFS	Вывести вершины в порядке обхода

№ вар.	Алгоритмы	Задача
11	DFS (итеративный), BFS	Найти все вершины на заданном расстоянии
12	DFS (рекурсивный), BFS	Найти диаметр графа (BFS дважды)
13	DFS (рекурсивный), BFS	Найти эйлеров путь
14	DFS (рекурсивный), BFS	Найти количество мостов
15	DFS (рекурсивный), BFS	Найти точки сочленения

Часть 3. Применение обходов

Задание для всех вариантов:

Реализовать программу, которая:

1. Создаёт граф по заданию (количество вершин и рёбер)
2. Выполняет обход DFS и BFS, начиная с указанной вершины
3. Находит все компоненты связности и выводит их
4. Для выбранной пары вершин находит кратчайший путь (BFS)
5. Выводит результаты в наглядной форме

Пример графа (для тестирования):

text

0 - 1 - 2 - 3

| |

5 - 4 - - 6

Контрольные вопросы

1. Что такое граф? Из каких элементов он состоит?
2. Чем отличается ориентированный граф от неориентированного?
3. Что такое матрица смежности? Каковы её достоинства и недостатки?
4. Что такое список смежности? Каковы его достоинства и недостатки?
5. В каких случаях лучше использовать матрицу смежности, а в каких - список смежности?
6. Какова сложность проверки наличия ребра в матрице смежности? В списке смежности?
7. Какова сложность перебора всех соседей вершины в матрице смежности? В списке смежности?
8. Что такое поиск в глубину (DFS)? Как он работает?
9. Какие задачи решаются с помощью DFS?
10. Что такое поиск в ширину (BFS)? Как он работает?
11. Какие задачи решаются с помощью BFS?
12. В чём отличие рекурсивной реализации DFS от итеративной?
13. Почему BFS находит кратчайший путь в невзвешенном графе, а DFS - нет?
14. Как найти компоненты связности графа с помощью DFS?
15. Как проверить наличие цикла в графе с помощью DFS?

Требования к отчёту

Титульный лист

Цель работы

Теоретическая часть (графы, матрица и список смежности, DFS, BFS)

Постановка задачи (согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода, визуализация графа)

Анализ сложности операций для выбранного представления

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Критерии оценивания

Максимальный балл: 100

№	Критерий	Баллы
1	Представление графа	25
1.1	Реализация выбранного способа представления	10
1.2	Функции добавления/удаления рёбер	10
1.3	Вывод графа	5
2	Обход графа (DFS и BFS)	35
2.1	Реализация DFS (рекурсивной)	10
2.2	Реализация DFS (итеративной)	5
2.3	Реализация BFS	10
2.4	Корректность обходов на разных графах	10
3	Применение обходов	25
3.1	Поиск компонент связности	10
3.2	Поиск кратчайшего пути (BFS)	10
3.3	Дополнительная задача по варианту	5

№	Критерий	Баллы
4	Оформление и отчёт	15
4.1	Код отформатирован, есть комментарии	5
4.2	Есть ответы на контрольные вопросы	5
4.3	Ссылка на Git-репозиторий	5
Итого		100

Штрафные баллы

Нарушение	Штраф
Некорректная работа DFS (зацикливание)	-15
Некорректная работа BFS (порядок обхода)	-10
Неверная сложность алгоритма	-10
Нет проверки на пустой граф	-5
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №19

Алгоритм Дейкстры. Алгоритм Флойда-Уоршелла

Цель работы:

1. Реализовать алгоритм Дейкстры для нахождения кратчайших путей от одной вершины до всех остальных во взвешенном графе с неотрицательными весами
2. Реализовать алгоритм Флойда-Уоршелла для нахождения кратчайших путей между всеми парами вершин
3. Изучить особенности работы алгоритмов с графами, содержащими отрицательные веса
4. Научиться восстанавливать кратчайшие пути после выполнения алгоритмов
5. Сравнить временную сложность и области применения алгоритмов

Теоретическое обоснование

1. Постановка задачи

Задача поиска кратчайшего пути - одна из классических задач теории графов. Требуется найти путь между двумя вершинами графа, сумма весов рёбер которого минимальна.

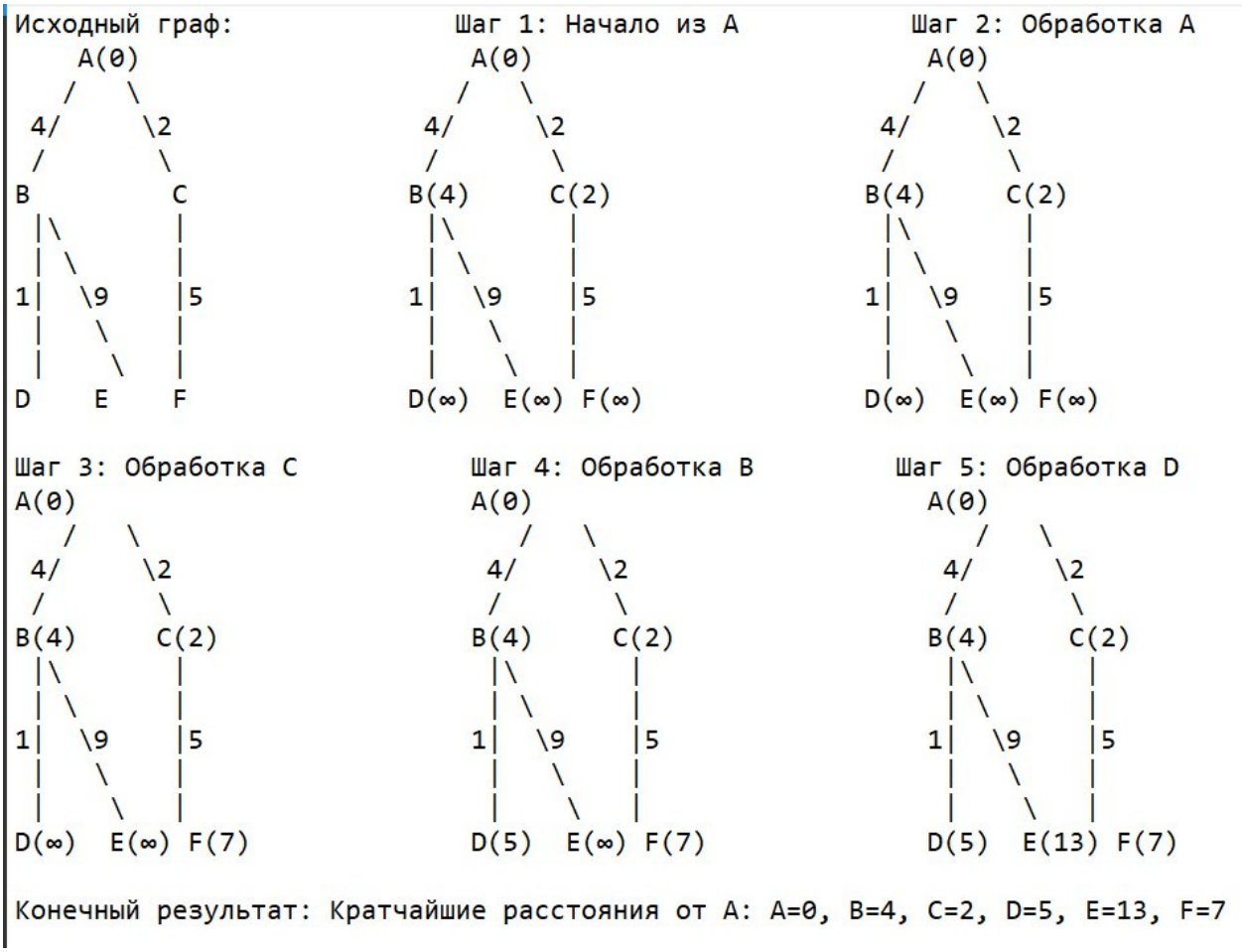
Алгоритм	Назначение	Сложность	Отрицательные веса
Дейкстры	Из одной вершины до всех остальных	O((V + E) log V)	Не допускаются
Флойда-Уоршелла	Между всеми парами вершин	$O(V^3)$	Допускаются (нет отрицательных циклов)

2. Алгоритм Дейкстры (Dijkstra's Algorithm)

Принцип работы: Жадный алгоритм, который на каждом шаге выбирает вершину с наименьшим известным расстоянием и релаксирует её соседей.

Условие применимости: Все веса рёбер должны быть неотрицательными.

Визуализация работы алгоритма Дейкстры:



Алгоритм:

```
vector<int> dijkstra(int start) {
    vector<int> dist(V, INF);
    vector<bool> visited(V, false);

    dist[start] = 0;

    for (int i = 0; i < V - 1; i++) {
        // Находим вершину с минимальным расстоянием среди необработанных
        int u = -1;
        for (int j = 0; j < V; j++) {
            if (!visited[j] && (u == -1 || dist[j] < dist[u])) {
```

```
            u = j;
        }
    }

    visited[u] = true;

    // Релаксация рёбер
    for (int v = 0; v < V; v++) {
        if (adj[u][v] != 0 && !visited[v]) {
            if (dist[u] + adj[u][v] < dist[v]) {
                dist[v] = dist[u] + adj[u][v];
            }
        }
    }
}

return dist;
}
```

Оптимизированная версия с priority_queue:

```
vector<int> dijkstra_pq(int start) {
    vector<int> dist(V, INF);
    using pii = pair<int, int>;  // {расстояние, вершина}
    priority_queue<pii, vector<pii>, greater<pii>> pq;

    dist[start] = 0;
    pq.push({0, start});
```

```
    while (!pq.empty()) {
        int d = pq.top().first;
        int u = pq.top().second;
        pq.pop();

        if (d > dist[u]) continue;  // устаревшая запись

        for (int v = 0; v < V; v++) {
            if (adj[u][v] != 0) {
                if (dist[u] + adj[u][v] < dist[v]) {
                    dist[v] = dist[u] + adj[u][v];
                    pq.push({dist[v], v});
                }
            }
        }
    }

    return dist;
}
```

Восстановление пути:

```
vector<int> dijkstraWithPath(int start, int end) {
    vector<int> dist(V, INF);
    vector<int> prev(V, -1);
    vector<bool> visited(V, false);

    dist[start] = 0;

    for (int i = 0; i < V - 1; i++) {
        int u = -1;
```

```
        for (int j = 0; j < V; j++) {
            if (!visited[j] && (u == -1 || dist[j] < dist[u])) {
                u = j;
            }
        }

        visited[u] = true;

        for (int v = 0; v < V; v++) {
            if (adj[u][v] != 0 && !visited[v]) {
                if (dist[u] + adj[u][v] < dist[v]) {
                    dist[v] = dist[u] + adj[u][v];
                    prev[v] = u;
                }
            }
        }
    }

    // Восстановление пути
    vector<int> path;
    for (int v = end; v != -1; v = prev[v]) {
        path.push_back(v);
    }
    reverse(path.begin(), path.end());
    return path;
}
```

3. Алгоритм Флойда-Уоршелла (Floyd-Warshall Algorithm)

Принцип работы: Динамическое программирование - на каждой итерации k разрешаем использовать вершины 0..k в качестве промежуточных.

Визуализация:

Матрица расстояний после каждой итерации k:

k = -1 (начальная):	k = 0:	k = 1:
[0, ∞, ∞, ∞]	[0, 1, 3, ∞]	[0, 1, 3, 4]
[1, 0, ∞, ∞]	[1, 0, 2, ∞]	[1, 0, 2, 3]
[∞, 2, 0, 5]	[3, 2, 0, 5]	[3, 2, 0, 4]
[∞, ∞, 5, 0]	[∞, ∞, 5, 0]	[4, 3, 4, 0]

k = 2:	k = 3 (конечная):
[0, 1, 3, 4]	[0, 1, 3, 4]
[1, 0, 2, 3]	[1, 0, 2, 3]
[3, 2, 0, 4]	[3, 2, 0, 4]
[4, 3, 4, 0]	[4, 3, 4, 0]

Алгоритм:

```
vector<vector<int>> floydWarshall() {
   // Инициализация матрицы расстояний
   vector<vector<int>> dist(V, vector<int>(V, INF));

   for (int i = 0; i < V; i++) {
      dist[i][i] = 0;
      for (int j = 0; j < V; j++) {
         if (adj[i][j] != 0) {
            dist[i][j] = adj[i][j];
         }
```

```
        }
    }

    // Основной цикл алгоритма
    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dist[i][k] != INF && dist[k][j] != INF) {
                    if (dist[i][k] + dist[k][j] < dist[i][j]) {
                        dist[i][j] = dist[i][k] + dist[k][j];
                    }
                }
            }
        }
    }

    return dist;
}
```

Обнаружение отрицательных циклов:

```
bool hasNegativeCycle() {
    vector<vector<int>> dist = floydWarshall();

    for (int i = 0; i < V; i++) {
        if (dist[i][i] < 0) {
            return true;  // найден отрицательный цикл
        }
    }
    return false;
```

```
}
```

Восстановление путей:

```
vector<vector<int>> floydWarshallWithPath(vector<vector<int>>& next) {
    vector<vector<int>> dist(V, vector<int>(V, INF));

    // Инициализация next матрицы для восстановления путей
    next.assign(V, vector<int>(V, -1));

    for (int i = 0; i < V; i++) {
        dist[i][i] = 0;
        for (int j = 0; j < V; j++) {
            if (adj[i][j] != 0) {
                dist[i][j] = adj[i][j];
                next[i][j] = j;
            }
        }
    }

    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dist[i][k] != INF && dist[k][j] != INF) {
                    if (dist[i][k] + dist[k][j] < dist[i][j]) {
                        dist[i][j] = dist[i][k] + dist[k][j];
                        next[i][j] = next[i][k];
                    }
                }
            }
```

```
        }
    }

    return dist;
}

vector<int> getPath(int u, int v, const vector<vector<int>>& next) {
    if (next[u][v] == -1) return {};

    vector<int> path = {u};
    while (u != v) {
        u = next[u][v];
        path.push_back(u);
    }
    return path;
}
```

4. Сравнение алгоритмов

Характеристика	Алгоритм Дейкстры	Алгоритм Флойда-Уоршелла
Количество вершин	От одной до всех	Все пары
Сложность	O((V+E) log V) или O(V^2)	O(V^3)
Отрицательные	Нет	Да (без

Характеристика	Алгоритм Дейкстры	Алгоритм Флойда-Уоршелла
веса		циклов)
Отрицательные циклы	Нет	Обнаруживает
Память	O(V)	O(V²)
Релаксация	Жадная	Динамическое программирование
Применимость	Большие разреженные графы	Малые графы (V ≤ 500)

Примеры выполнения практической части

Пример 1. Алгоритм Дейкстры (базовый уровень)

Задача: Реализовать алгоритм Дейкстры для нахождения кратчайших расстояний от заданной вершины.

```
#include <iostream>
#include <vector>
#include <climits>

using namespace std;
```

```
class Graph {
private:
    int V;
    vector<vector<int>> adj;

public:
    Graph(int vertices) : V(vertices) {
        adj.resize(V, vector<int>(V, 0));
    }

    void addEdge(int u, int v, int weight) {
        adj[u][v] = weight;
        adj[v][u] = weight;  // для неориентированного графа
    }

    vector<int> dijkstra(int start) {
        vector<int> dist(V, INT_MAX);
        vector<bool> visited(V, false);

        dist[start] = 0;

        for (int i = 0; i < V - 1; i++) {
            // Находим вершину с минимальным расстоянием
            int u = -1;
            for (int j = 0; j < V; j++) {
                if (!visited[j] && (u == -1 || dist[j] < dist[u])) {
                    u = j;
                }
```

```
            }

            if (dist[u] == INT_MAX) break;  // остальные вершины недостижимы

            visited[u] = true;

            // Релаксация
            for (int v = 0; v < V; v++) {
                if (adj[u][v] != 0 && !visited[v]) {
                    if (dist[u] + adj[u][v] < dist[v]) {
                        dist[v] = dist[u] + adj[u][v];
                    }
                }
            }
        }

        return dist;
    }

    void printDistances(int start) {
        vector<int> dist = dijkstra(start);

        cout << "Кратчайшие расстояния от вершины " << start << ":" << endl;
        for (int i = 0; i < V; i++) {
            if (dist[i] == INT_MAX) {
                cout << "  До " << i << ": недостижима" << endl;
            } else {
```

```
                cout << "  До " << i << ": " << dist[i] << endl;
            }
        }
    }
};

int main() {
    Graph g(6);

    g.addEdge(0, 1, 4);
    g.addEdge(0, 2, 2);
    g.addEdge(1, 2, 1);
    g.addEdge(1, 3, 5);
    g.addEdge(2, 3, 8);
    g.addEdge(2, 4, 10);
    g.addEdge(3, 4, 2);
    g.addEdge(3, 5, 6);
    g.addEdge(4, 5, 3);

    g.printDistances(0);

    return 0;
}
```

Ожидаемый вывод:

Кратчайшие расстояния от вершины 0:
  До 0: 0
  До 1: 3
  До 2: 2

До 3: 8

До 4: 10

До 5: 13

Пример 2. Алгоритм Флойда-Уоршелла (средний уровень)

Задача: Реализовать алгоритм Флойда-Уоршелла для нахождения кратчайших путей между всеми парами вершин.

```
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

class Graph {
private:
    int V;
    vector<vector<int>> adj;

public:
    Graph(int vertices) : V(vertices) {
        adj.resize(V, vector<int>(V, 0));
    }

    void addEdge(int u, int v, int weight) {
        adj[u][v] = weight;
    }

    vector<vector<int>> floydWarshall() {
```

```
    // Инициализация матрицы расстояний
    vector<vector<int>> dist(V, vector<int>(V, INT_MAX));

    for (int i = 0; i < V; i++) {
        dist[i][i] = 0;
        for (int j = 0; j < V; j++) {
            if (adj[i][j] != 0) {
                dist[i][j] = adj[i][j];
            }
        }
    }

    // Основной алгоритм
    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX) {
                    if (dist[i][k] + dist[k][j] < dist[i][j]) {
                        dist[i][j] = dist[i][k] + dist[k][j];
                    }
                }
            }
        }
    }

    return dist;
}

void printAllPairsShortestPaths() {
```

```
        vector<vector<int>> dist = floydWarshall();

        cout << "Матрица кратчайших расстояний:" << endl;
        cout << "     ";
        for (int j = 0; j < V; j++) {
            cout << j << "   ";
        }
        cout << endl;
        cout << "   ";
        for (int j = 0; j < V; j++) {
            cout << "----";
        }
        cout << endl;

        for (int i = 0; i < V; i++) {
            cout << i << " | ";
            for (int j = 0; j < V; j++) {
                if (dist[i][j] == INT_MAX) {
                    cout << "∞   ";
                } else {
                    cout << dist[i][j] << "   ";
                }
            }
            cout << endl;
        }
    }
};

int main() {
```

```
    Graph g(4);

    g.addEdge(0, 1, 3);
    g.addEdge(0, 2, 6);
    g.addEdge(1, 2, 2);
    g.addEdge(1, 3, 4);
    g.addEdge(2, 3, 1);
    g.addEdge(3, 2, 1);

    g.printAllPairsShortestPaths();

    return 0;
}
```

Ожидаемый вывод:

Матрица кратчайших расстояний:

```
    0   1   2   3
  ----------------
0 | 0   3   5   7
1 | ∞   0   2   3
2 | ∞   ∞   0   1
3 | ∞   ∞   1   0
```

Пример 3. Восстановление путей (продвинутый уровень)

Задача: Реализовать оба алгоритма с возможностью восстановления кратчайших путей.

```
#include <iostream>
#include <vector>
#include <climits>
```

```cpp
#include <algorithm>

using namespace std;

class Graph {
private:
    int V;
    vector<vector<int>> adj;

public:
    Graph(int vertices) : V(vertices) {
        adj.resize(V, vector<int>(V, 0));
    }

    void addEdge(int u, int v, int weight) {
        adj[u][v] = weight;
        adj[v][u] = weight;
    }

    // Алгоритм Дейкстры с восстановлением пути
    pair<vector<int>, vector<int>> dijkstraWithPath(int start) {
        vector<int> dist(V, INT_MAX);
        vector<int> prev(V, -1);
        vector<bool> visited(V, false);

        dist[start] = 0;

        for (int i = 0; i < V - 1; i++) {
            int u = -1;
```

```
        for (int j = 0; j < V; j++) {
            if (!visited[j] && (u == -1 || dist[j] < dist[u])) {
                u = j;
            }
        }

        if (dist[u] == INT_MAX) break;

        visited[u] = true;

        for (int v = 0; v < V; v++) {
            if (adj[u][v] != 0 && !visited[v]) {
                if (dist[u] + adj[u][v] < dist[v]) {
                    dist[v] = dist[u] + adj[u][v];
                    prev[v] = u;
                }
            }
        }
    }

    return {dist, prev};
}

vector<int> getPath(int start, int end, const vector<int>& prev) {
    vector<int> path;
    if (prev[end] == -1 && start != end) return path;

    for (int v = end; v != -1; v = prev[v]) {
        path.push_back(v);
```

```
        }
        reverse(path.begin(), path.end());
        return path;
    }

    // Алгоритм Флойда-Уоршелла с восстановлением путей
    pair<vector<vector<int>>, vector<vector<int>>> floydWarshallWithPath() {
        vector<vector<int>> dist(V, vector<int>(V, INT_MAX));
        vector<vector<int>> next(V, vector<int>(V, -1));

        for (int i = 0; i < V; i++) {
            dist[i][i] = 0;
            for (int j = 0; j < V; j++) {
                if (adj[i][j] != 0) {
                    dist[i][j] = adj[i][j];
                    next[i][j] = j;
                }
            }
        }

        for (int k = 0; k < V; k++) {
            for (int i = 0; i < V; i++) {
                for (int j = 0; j < V; j++) {
                    if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX) {
                        if (dist[i][k] + dist[k][j] < dist[i][j]) {
                            dist[i][j] = dist[i][k] + dist[k][j];
                            next[i][j] = next[i][k];
                        }
```

```
                }
            }
        }
    }

    return {dist, next};
}

vector<int> getPathFloyd(int u, int v, const vector<vector<int>>& next) {
    if (next[u][v] == -1) return {};

    vector<int> path = {u};
    while (u != v) {
        u = next[u][v];
        path.push_back(u);
    }
    return path;
}
};

int main() {
    Graph g(5);

    g.addEdge(0, 1, 2);
    g.addEdge(0, 2, 5);
    g.addEdge(1, 2, 1);
    g.addEdge(1, 3, 6);
    g.addEdge(2, 3, 2);
    g.addEdge(2, 4, 3);
```

```
    g.addEdge(3, 4, 1);

    cout << "=== Алгоритм Дейкстры ===" << endl;
    auto [dist, prev] = g.dijkstraWithPath(0);

    cout << "Расстояния от вершины 0:" << endl;
    for (int i = 0; i < 5; i++) {
        cout << "  До " << i << ": " << dist[i] << " (путь: ";
        vector<int> path = g.getPath(0, i, prev);
        for (int v : path) cout << v << " ";
        cout << ")" << endl;
    }

    cout << "\n=== Алгоритм Флойда-Уоршелла ===" << endl;
    auto [distFW, next] = g.floydWarshallWithPath();

    cout << "Кратчайшие пути между всеми парами:" << endl;
    for (int i = 0; i < 5; i++) {
        for (int j = i + 1; j < 5; j++) {
            cout << "  " << i << " → " << j << ": расстояние = " << distFW[i][j];
            vector<int> path = g.getPathFloyd(i, j, next);
            cout << ", путь: ";
            for (int v : path) cout << v << " ";
            cout << endl;
        }
    }

    return 0;
}
```

Ожидаемый вывод:

=== Алгоритм Дейкстры ===

Расстояния от вершины 0:

До 0: 0 (путь: 0)

До 1: 2 (путь: 0 1)

До 2: 3 (путь: 0 1 2)

До 3: 5 (путь: 0 1 2 3)

До 4: 6 (путь: 0 1 2 4)

=== Алгоритм Флойда-Уоршелла ===

Кратчайшие пути между всеми парами:

0 → 1: расстояние = 2, путь: 0 1

0 → 2: расстояние = 3, путь: 0 1 2

0 → 3: расстояние = 5, путь: 0 1 2 3

0 → 4: расстояние = 6, путь: 0 1 2 4

1 → 2: расстояние = 1, путь: 1 2

1 → 3: расстояние = 3, путь: 1 2 3

1 → 4: расстояние = 4, путь: 1 2 4

2 → 3: расстояние = 2, путь: 2 3

2 → 4: расстояние = 3, путь: 2 4

3 → 4: расстояние = 1, путь: 3 4

Индивидуальные задания

Часть 1. Алгоритм Дейкстры

Каждый вариант содержит задание реализовать алгоритм Дейкстры с указанными особенностями:

№ вар.	Способ реализации	Особенности
1	Без оптимизации ($O(V^2)$)	Неориентированный граф
2	С priority_queue ($O((V+E) \log V)$)	Неориентированный граф
3	Без оптимизации ($O(V^2)$)	Ориентированный граф
4	С priority_queue ($O((V+E) \log V)$)	Ориентированный граф
5	Без оптимизации ($O(V^2)$)	С восстановлением пути
6	С priority_queue ($O((V+E) \log V)$)	С восстановлением пути
7	Без оптимизации ($O(V^2)$)	Разреженный граф ($E << V^2$)
8	С priority_queue ($O((V+E) \log V)$)	Плотный граф ($E \approx V^2$)
9	Без оптимизации ($O(V^2)$)	Поиск пути между двумя вершинами
10	С priority_queue ($O((V+E) \log V)$)	Поиск пути между двумя вершинами
11	Без оптимизации ($O(V^2)$)	Нахождение всех кратчайших путей

№ вар.	Способ реализации	Особенности
12	С priority_queue (O((V+E) log V))	Нахождение всех кратчайших путей
13	Без оптимизации (O(V²))	Граф с большим количеством вершин (V=1000)
14	С priority_queue (O((V+E) log V))	Граф с большим количеством вершин (V=1000)
15	Оба варианта	Сравнение производительности

Часть 2. Алгоритм Флойда-Уоршелла

Каждый вариант содержит задание реализовать алгоритм Флойда-Уоршелла с указанными особенностями:

№ вар.	Особенности	Дополнительное задание
1	Неориентированный граф	Вывод матрицы расстояний
2	Ориентированный граф	Вывод матрицы расстояний
3	Неориентированный граф	Восстановление путей

№ вар.	Особенности	Дополнительное задание
4	Ориентированный граф	Восстановление путей
5	Неориентированный граф	Обнаружение отрицательных циклов
6	Ориентированный граф	Обнаружение отрицательных циклов
7	Неориентированный граф	Поиск диаметра графа
8	Ориентированный граф	Поиск эксцентриситета вершин
9	Неориентированный граф	Нахождение центра графа
10	Ориентированный граф	Нахождение радиуса графа
11	Неориентированный граф	Сравнение с Дейкстрой (V=50)
12	Ориентированный граф	Сравнение с Дейкстрой (V=50)
13	Неориентированный граф	Вычисление транзитивного замыкания

№ вар.	Особенности	Дополнительное задание
14	Ориентированный граф	Вычисление транзитивного замыкания
15	Оба варианта	Сравнение производительности (V=100)

Часть 3. Сравнительный анализ

Задание для всех вариантов:

1. Реализовать граф на 20 вершин со случайными весами рёбер
2. Выполнить алгоритм Дейкстры для случайной стартовой вершины
3. Выполнить алгоритм Флойда-Уоршелла
4. Сравнить результаты (расстояния между одними и теми же парами)
5. Измерить время выполнения для V = 10, 20, 50, 100 (для Дейкстры - из одной вершины, для Флойда-Уоршелла - между всеми)
6. Построить график зависимости времени от V

Контрольные вопросы

1. Какие условия должны выполняться для корректной работы алгоритма Дейкстры?
2. Почему алгоритм Дейкстры не работает с отрицательными весами?
3. Какова временная сложность алгоритма Дейкстры с приоритетной очередью?
4. В чём заключается идея алгоритма Флойда-Уоршелла?
5. Какова временная сложность алгоритма Флойда-Уоршелла?
6. Как алгоритм Флойда-Уоршелла обнаруживает отрицательные циклы?

7. В каких случаях предпочтительнее использовать алгоритм Дейкстры?
8. В каких случаях предпочтительнее использовать алгоритм Флойда-Уоршелла?
9. Как восстановить кратчайший путь после выполнения алгоритма Дейкстры?
10. Как восстановить кратчайший путь после выполнения алгоритма Флойда-Уоршелла?
11. Что такое релаксация в контексте алгоритмов поиска кратчайших путей?
12. Как можно оптимизировать алгоритм Дейкстры для разреженных графов?
13. Какую память требует алгоритм Флойда-Уоршелла?
14. При каком количестве вершин алгоритм Флойда-Уоршелла становится неприменимым?
15. Как модифицировать алгоритм Дейкстры для нахождения второго кратчайшего пути?

Требования к отчёту

Титульный лист

Цель работы

Теоретическая часть (алгоритмы Дейкстры и Флойда-Уоршелла)

Постановка задачи (согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода)

Анализ сложности алгоритмов

Сравнительная таблица производительности

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Критерии оценивания

Максимальный балл: 100

№	Критерий	Баллы
1	Алгоритм Дейкстры	35
1.1	Реализация алгоритма	15
1.2	Корректность работы	10
1.3	Восстановление пути (по варианту)	10
2	Алгоритм Флойда-Уоршелла	40
2.1	Реализация алгоритма	15
2.2	Корректность работы	10
2.3	Восстановление путей	10
2.4	Обнаружение отрицательных циклов (по варианту)	5
3	Сравнительный анализ	15
3.1	Измерение времени выполнения	5
3.2	Таблица/график сравнения	5
3.3	Выводы о производительности	5
4	Оформление и отчёт	10
4.1	Код отформатирован, есть комментарии	5

№	Критерий	Баллы
4.2	Ссылка на Git-репозиторий	5
Итого		100

Штрафные баллы

Нарушение	Штраф
Неправильная обработка недостижимых вершин	-10
Неверная сложность алгоритма	-10
Отсутствие восстановления путей (по требованию)	-10
Нет сравнительного анализа	-10
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка	Баллы
Отлично	90–100
Хорошо	75–89

Оценка	Баллы
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №20

Алгоритмы Прима и Краскала. Минимальное остовное дерево

Цель работы:

1. Изучить понятие минимального остовного дерева (Minimum Spanning Tree, MST) и его применение
2. Реализовать алгоритм Прима для построения MST в связном взвешенном графе
3. Реализовать алгоритм Краскала для построения MST с использованием системы непересекающихся множеств (DSU)
4. Сравнить эффективность алгоритмов Прима и Краскала на различных типах графов
5. Научиться применять MST для решения практических задач (сети связи, дорожные сети)

Теоретическое обоснование

1. Понятие минимального остовного дерева

Остовное дерево (Spanning Tree) - это подграф связного графа, который:

Содержит все вершины исходного графа

Является деревом (не содержит циклов)

Имеет ровно V-1 рёбер

Минимальное остовное дерево (Minimum Spanning Tree, MST) - остовное дерево с минимальной суммой весов рёбер.

Визуализация:

Вес исходного графа: 4+3+4+2+2+5+... = 20+
Вес MST: 4+3+2+2+5 = 16

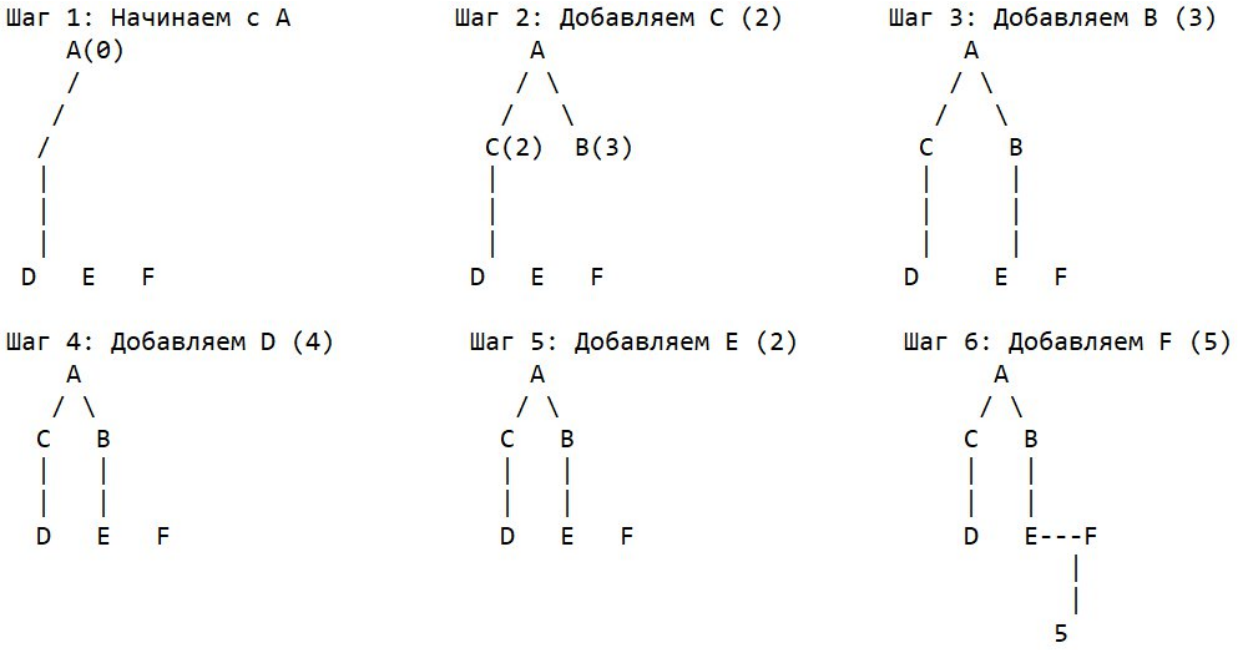
2. Свойства минимального остовного дерева

Свойство	Описание
Единственность	Если все веса рёбер различны, MST единственно
Разрез	Минимальное ребро, пересекающее разрез, принадлежит MST
Цикл	Максимальное ребро в любом цикле не принадлежит MST

3. Алгоритм Прима (Prim's Algorithm)

Принцип работы: Жадный алгоритм, который начинает с одной вершины и на каждом шаге добавляет минимальное ребро, соединяющее текущее дерево с ещё не включённой вершиной.

Визуализация алгоритма Прима:



Алгоритм:

```
int primMST(int start) {
    vector<int> minEdge(V, INF);    // минимальное ребро для каждой вершины
    vector<bool> inMST(V, false); // входит ли вершина в MST
    vector<int> parent(V, -1);    // родитель в MST
    int totalWeight = 0;

    minEdge[start] = 0;

    for (int i = 0; i < V; i++) {
```

```
            // Находим вершину с минимальным minEdge среди не включённых
            int u = -1;
            for (int j = 0; j < V; j++) {
                if (!inMST[j] && (u == -1 || minEdge[j] < minEdge[u])) {
                    u = j;
                }
            }

            inMST[u] = true;
            totalWeight += minEdge[u];

            // Обновляем соседей
            for (int v = 0; v < V; v++) {
                if (adj[u][v] != 0 && !inMST[v] && adj[u][v] < minEdge[v]) {
                    minEdge[v] = adj[u][v];
                    parent[v] = u;
                }
            }
        }

        return totalWeight;
    }
```

Оптимизированная версия с priority_queue:

```
int primMST_pq(int start) {
    vector<int> minEdge(V, INF);
    vector<bool> inMST(V, false);
    vector<int> parent(V, -1);
```

```
    int totalWeight = 0;

    using pii = pair<int, int>;  // {вес, вершина}
    priority_queue<pii, vector<pii>, greater<pii>> pq;

    minEdge[start] = 0;
    pq.push({0, start});

    while (!pq.empty()) {
        int w = pq.top().first;
        int u = pq.top().second;
        pq.pop();

        if (inMST[u]) continue;

        inMST[u] = true;
        totalWeight += w;

        for (int v = 0; v < V; v++) {
            if (adj[u][v] != 0 && !inMST[v] && adj[u][v] < minEdge[v]) {
                minEdge[v] = adj[u][v];
                parent[v] = u;
                pq.push({minEdge[v], v});
            }
        }
    }

    return totalWeight;
}
```

4. Алгоритм Краскала (Kruskal's Algorithm)

Принцип работы: Сортирует все рёбра по весу и добавляет их в MST, если они не создают цикл (проверка через DSU).

Визуализация алгоритма Краскала:

```
Список рёбер, отсортированный по весу:
1. (A,B): 4   6. (E,F): 5
2. (C,D): 2   7. (D,E): 5
3. (B,C): 3   8. (A,F): 6
4. (B,D): 2   9. (C,E): 7
5. (A,C): 3
```

```
Шаг 1: (C,D):2
    C-D
```

```
Шаг 2: (B,D):2
     C-D
     |
     B
```

```
Шаг 3: (B,C):3
      C-D
      |
      B-C
       |
       D
```

```
Шаг 4: (A,C):3
    A-C-D
    |
    B
```

```
Шаг 5: (E,F):5
     A-C-D
     |
     B
     |
     E-F
```

Алгоритм:

```
struct Edge {
    int u, v, weight;
    Edge(int u, int v, int w) : u(u), v(v), weight(w) {}
    bool operator<(const Edge& other) const {
        return weight < other.weight;
    }
```

```
};

int kruskalMST(vector<Edge>& edges) {
    // Сортировка рёбер по весу
    sort(edges.begin(), edges.end());

    DSU dsu(V);
    int totalWeight = 0;
    vector<Edge> mstEdges;

    for (const Edge& e : edges) {
        if (dsu.find(e.u) != dsu.find(e.v)) {
            dsu.unite(e.u, e.v);
            totalWeight += e.weight;
            mstEdges.push_back(e);
            if (mstEdges.size() == V - 1) break;
        }
    }

    return totalWeight;
}
```

5. Система непересекающихся множеств (DSU)

DSU (Disjoint Set Union) - структура данных для объединения множеств и проверки принадлежности элементов одному множеству.

```
class DSU {
private:
    vector<int> parent;
```

```
    vector<int> rank;

public:
    DSU(int n) {
        parent.resize(n);
        rank.resize(n, 0);
        for (int i = 0; i < n; i++) {
            parent[i] = i;
        }
    }

    int find(int x) {
        if (parent[x] != x) {
            parent[x] = find(parent[x]);  // сжатие пути
        }
        return parent[x];
    }

    void unite(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);

        if (rootX == rootY) return;

        // объединение по рангу
        if (rank[rootX] < rank[rootY]) {
            parent[rootX] = rootY;
        } else if (rank[rootX] > rank[rootY]) {
            parent[rootY] = rootX;
```

```
        } else {
            parent[rootY] = rootX;
            rank[rootX]++;
        }
    }

    bool connected(int x, int y) {
        return find(x) == find(y);
    }
};
```

6. Сравнение алгоритмов

Характеристика	**Алгоритм Прима**	**Алгоритм Краскала**
Тип графа	Неориентированный	Неориентированный
Сложность (простая)	$O(V^2)$	O(E log E)
Сложность (оптимизированная)	O((V+E) log V)	O(E log E)
Подход	Наращивание дерева	Объединение компонент
Лучше для	Плотных графов	Разреженных графов

Характеристика	Алгоритм Прима	Алгоритм Краскала
Память	O(V) + матрица	O(E)
Структура данных	Очередь с приоритетами	DSU + сортировка

Примеры выполнения практической части

Пример 1. Алгоритм Прима (базовый уровень)

Задача: Реализовать алгоритм Прима для нахождения минимального остовного дерева.

```
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

class PrimMST {
private:
    int V;
    vector<vector<int>> graph;

public:
    PrimMST(int vertices) : V(vertices) {
        graph.resize(V, vector<int>(V, 0));
    }
```

```
void addEdge(int u, int v, int weight) {
    graph[u][v] = weight;
    graph[v][u] = weight;
}

int primMST() {
    vector<int> minEdge(V, INT_MAX);
    vector<bool> inMST(V, false);
    vector<int> parent(V, -1);
    int totalWeight = 0;

    minEdge[0] = 0;

    for (int i = 0; i < V; i++) {
        // Находим вершину с минимальным minEdge
        int u = -1;
        for (int j = 0; j < V; j++) {
            if (!inMST[j] && (u == -1 || minEdge[j] < minEdge[u])) {
                u = j;
            }
        }

        if (minEdge[u] == INT_MAX) {
            cout << "Граф не связный!" << endl;
            return -1;
        }

        inMST[u] = true;
        totalWeight += minEdge[u];
```

```
            // Обновляем соседей
            for (int v = 0; v < V; v++) {
                if (graph[u][v] != 0 && !inMST[v] && graph[u][v] < minEdge[v])
{
                    minEdge[v] = graph[u][v];
                    parent[v] = u;
                }
            }
        }

        // Вывод рёбер MST
        cout << "Рёбра минимального остовного дерева:" << endl;
        for (int i = 1; i < V; i++) {
            cout << "  " << parent[i] << " - " << i << " (вес: " << minEdge[i] <<
")" << endl;
        }

        return totalWeight;
    }
};

int main() {
    PrimMST g(5);

    g.addEdge(0, 1, 2);
    g.addEdge(0, 3, 6);
    g.addEdge(1, 2, 3);
    g.addEdge(1, 3, 8);
```

```
    g.addEdge(1, 4, 5);
    g.addEdge(2, 4, 7);
    g.addEdge(3, 4, 9);

    int weight = g.primMST();
    cout << "Общий вес MST: " << weight << endl;

    return 0;
}
```

Ожидаемый вывод:

Рёбра минимального остовного дерева:

0 - 1 (вес: 2)

1 - 2 (вес: 3)

1 - 4 (вес: 5)

0 - 3 (вес: 6)

Общий вес MST: 16

Пример 2. Алгоритм Краскала (средний уровень)

Задача: Реализовать алгоритм Краскала с использованием DSU.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

class DSU {
private:
    vector<int> parent;
```

```
    vector<int> rank;

public:
    DSU(int n) {
        parent.resize(n);
        rank.resize(n, 0);
        for (int i = 0; i < n; i++) {
            parent[i] = i;
        }
    }

    int find(int x) {
        if (parent[x] != x) {
            parent[x] = find(parent[x]);
        }
        return parent[x];
    }

    void unite(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);

        if (rootX == rootY) return;

        if (rank[rootX] < rank[rootY]) {
            parent[rootX] = rootY;
        } else if (rank[rootX] > rank[rootY]) {
            parent[rootY] = rootX;
        } else {
```

```
            parent[rootY] = rootX;
            rank[rootX]++;
        }
    }

    bool connected(int x, int y) {
        return find(x) == find(y);
    }
};

struct Edge {
    int u, v, weight;
    Edge(int u, int v, int w) : u(u), v(v), weight(w) {}
};

class KruskalMST {
private:
    int V;
    vector<Edge> edges;

public:
    KruskalMST(int vertices) : V(vertices) {}

    void addEdge(int u, int v, int weight) {
        edges.push_back(Edge(u, v, weight));
    }

    int kruskalMST() {
        // Сортировка рёбер по весу
```

```
    sort(edges.begin(), edges.end(),
        [](const Edge& a, const Edge& b) {
            return a.weight < b.weight;
        });

    DSU dsu(V);
    int totalWeight = 0;
    int edgesUsed = 0;

    cout << "Рёбра минимального остовного дерева:" << endl;

    for (const Edge& e : edges) {
        if (!dsu.connected(e.u, e.v)) {
            dsu.unite(e.u, e.v);
            totalWeight += e.weight;
            edgesUsed++;
            cout << "  " << e.u << " - " << e.v << " (вес: " << e.weight << ")"
<< endl;

            if (edgesUsed == V - 1) break;
        }
    }

    if (edgesUsed != V - 1) {
        cout << "Граф не связный!" << endl;
        return -1;
    }

    return totalWeight;
```

```
    }
};

int main() {
    KruskalMST g(5);

    g.addEdge(0, 1, 2);
    g.addEdge(0, 3, 6);
    g.addEdge(1, 2, 3);
    g.addEdge(1, 3, 8);
    g.addEdge(1, 4, 5);
    g.addEdge(2, 4, 7);
    g.addEdge(3, 4, 9);

    int weight = g.kruskalMST();
    cout << "Общий вес MST: " << weight << endl;

    return 0;
}
```

Ожидаемый вывод:

Рёбра минимального остовного дерева:

0 - 1 (вес: 2)

1 - 2 (вес: 3)

1 - 4 (вес: 5)

0 - 3 (вес: 6)

Общий вес MST: 16

Пример 3. Сравнение алгоритмов (продвинутый уровень)

Задача: Сравнить производительность алгоритмов Прима и Краскала на графах разного размера.

```
#include <iostream>
#include <vector>
#include <chrono>
#include <random>
#include <algorithm>

using namespace std;
using namespace chrono;

class GraphComparison {
private:
    int V;
    vector<vector<int>> adjMatrix;
    vector<tuple<int, int, int>> edgeList;

public:
    GraphComparison(int vertices) : V(vertices) {
        adjMatrix.resize(V, vector<int>(V, 0));
    }

    void generateRandomGraph(int edgeDensity) {
        mt19937 rng(42);
        uniform_int_distribution<int> weightDist(1, 100);
        uniform_int_distribution<int> edgeDist(0, 100);

        for (int i = 0; i < V; i++) {
            for (int j = i + 1; j < V; j++) {
```

```
            if (edgeDist(rng) < edgeDensity) {
                int w = weightDist(rng);
                adjMatrix[i][j] = w;
                adjMatrix[j][i] = w;
                edgeList.push_back({i, j, w});
            }
        }
    }
}

int primMST() {
    vector<int> minEdge(V, INT_MAX);
    vector<bool> inMST(V, false);
    minEdge[0] = 0;
    int totalWeight = 0;

    for (int i = 0; i < V; i++) {
        int u = -1;
        for (int j = 0; j < V; j++) {
            if (!inMST[j] && (u == -1 || minEdge[j] < minEdge[u])) {
                u = j;
            }
        }
        if (minEdge[u] == INT_MAX) return -1;
        inMST[u] = true;
        totalWeight += minEdge[u];
        for (int v = 0; v < V; v++) {
            if (adjMatrix[u][v] != 0 && !inMST[v] && adjMatrix[u][v] <
minEdge[v]) {
```

```
                minEdge[v] = adjMatrix[u][v];
            }
        }
    }
    return totalWeight;
}

int kruskalMST() {
    vector<tuple<int, int, int>> edges = edgeList;
    sort(edges.begin(), edges.end(), [](const auto& a, const auto& b) {
        return get<2>(a) < get<2>(b);
    });

    vector<int> parent(V);
    for (int i = 0; i < V; i++) parent[i] = i;

    function<int(int)> find = [&](int x) {
        if (parent[x] != x) parent[x] = find(parent[x]);
        return parent[x];
    };

    int totalWeight = 0;
    int edgesUsed = 0;

    for (const auto& e : edges) {
        int u = get<0>(e), v = get<1>(e), w = get<2>(e);
        if (find(u) != find(v)) {
            parent[find(u)] = find(v);
            totalWeight += w;
```

```
                edgesUsed++;
                if (edgesUsed == V - 1) break;
            }
        }

        return edgesUsed == V - 1 ? totalWeight : -1;
    }

    void compare(int edgeDensity) {
        generateRandomGraph(edgeDensity);

        auto start = high_resolution_clock::now();
        int primWeight = primMST();
        auto end = high_resolution_clock::now();
        auto primTime = duration_cast<microseconds>(end - start).count();

        start = high_resolution_clock::now();
        int kruskalWeight = kruskalMST();
        end = high_resolution_clock::now();
        auto kruskalTime = duration_cast<microseconds>(end - start).count();

        cout << "V = " << V << ", плотность = " << edgeDensity << "%" <<
endl;
        cout << "  Прим:    время = " << primTime << " мкс, вес = " <<
primWeight << endl;
        cout << "  Краскал: время = " << kruskalTime << " мкс, вес = " <<
kruskalWeight << endl;
        cout << "  Отношение (Краскал/Прим): " << (double)kruskalTime /
primTime << endl;
```

```
            cout << endl;
        }
    };

    int main() {
        cout << "=== Сравнение алгоритмов Прима и Краскала ===" << endl
<< endl;

        // Разреженный граф (20% рёбер)
        GraphComparison sparse(100);
        sparse.compare(20);

        // Плотный граф (60% рёбер)
        GraphComparison dense(100);
        dense.compare(60);

        // Очень плотный граф (90% рёбер)
        GraphComparison veryDense(100);
        veryDense.compare(90);

        return 0;
    }
```

Индивидуальные задания

Часть 1. Алгоритм Прима

Каждый вариант содержит задание реализовать алгоритм Прима с указанными особенностями:

№ вар.	Способ реализации	Особенности
1	Без оптимизации (O(V²))	Неориентированный граф
2	С priority_queue (O((V+E) log V))	Неориентированный граф
3	Без оптимизации (O(V²))	С восстановлением дерева
4	С priority_queue (O((V+E) log V))	С восстановлением дерева
5	Без оптимизации (O(V²))	Разреженный граф
6	С priority_queue (O((V+E) log V))	Плотный граф
7	Без оптимизации (O(V²))	Поиск MST с указанной начальной вершиной
8	С priority_queue (O((V+E) log V))	Поиск MST с указанной начальной вершиной
9	Без оптимизации (O(V²))	Вывод рёбер в порядке добавления
10	С priority_queue (O((V+E) log V))	Вывод рёбер в порядке добавления

№ вар.	Способ реализации	Особенности
1 1	Без оптимизации (O(V²))	Граф с целыми весами
1 2	С priority_queue (O((V+E) log V))	Граф с вещественными весами
1 3	Без оптимизации (O(V²))	Граф с большим количеством вершин (V=500)
1 4	С priority_queue (O((V+E) log V))	Граф с большим количеством вершин (V=500)
1 5	Оба варианта	Сравнение производительности

Часть 2. Алгоритм Краскала

Каждый вариант содержит задание реализовать алгоритм Краскала с указанными особенностями:

№ вар.	Особенности	Дополнительное задание
1	Простая сортировка	Вывод рёбер в порядке добавления
2	С использованием DSU	Вывод рёбер в порядке добавления

№ вар.	Особенности	Дополнительное задание
3	Простая сортировка	Проверка связности графа
4	С использованием DSU	Проверка связности графа
5	Простая сортировка	Разреженный граф
6	С использованием DSU	Плотный граф
7	Простая сортировка	С вещественными весами
8	С использованием DSU	С вещественными весами
9	Простая сортировка	Граф с большим количеством вершин (V=1000)
10	С использованием DSU	Граф с большим количеством вершин (V=1000)
11	Простая сортировка	Поиск второго минимального MST
12	С использованием DSU	Поиск второго минимального MST
13	Простая сортировка	Обработка нескольких компонент связности

№ вар.	Особенности	Дополнительное задание
14	С использованием DSU	Обработка нескольких компонент связности
15	Оба варианта	Сравнение эффективности DSU с рангами и без

Часть 3. Применение MST

Задание для всех вариантов:

Реализовать программу для решения задачи о прокладке сетей. Дано N городов и M возможных дорог между ними с указанием стоимости строительства.

Найти минимальную стоимость соединения всех городов (MST)

Вывести список дорог, которые нужно построить

Если граф не связный, вывести компоненты связности

Пример входных данных:

Города: A, B, C, D, E, F

Дороги:

A-B: 4

A-C: 3

B-C: 2

B-D: 2

C-E: 3

D-E: 5

E-F: 5

Ожидаемый результат:

Минимальная стоимость: 16

Дороги для строительства:

B-C (2)

B-D (2)

A-C (3)

C-E (3)

E-F (5)

Контрольные вопросы

1. Что такое минимальное остовное дерево (MST)?
2. Какие свойства имеет минимальное остовное дерево?
3. В чём заключается идея алгоритма Прима?
4. Какова временная сложность алгоритма Прима на матрице смежности?
5. Какова временная сложность алгоритма Прима с приоритетной очередью?
6. В чём заключается идея алгоритма Краскала?
7. Что такое DSU и как она используется в алгоритме Краскала?
8. Какова временная сложность алгоритма Краскала?
9. В каких случаях алгоритм Прима эффективнее алгоритма Краскала?
10. В каких случаях алгоритм Краскала эффективнее алгоритма Прима?
11. Как проверить, что построенное дерево является минимальным?
12. Что произойдёт, если граф не связный?
13. Как модифицировать алгоритмы для поиска максимального остовного дерева?
14. Где применяются MST в реальной жизни?
15. Как найти второй по минимальности вес MST?

Требования к отчёту

Титульный лист

Цель работы

Теоретическая часть (MST, Прим, Краскал, DSU)

Постановка задачи (согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода)

Анализ сложности алгоритмов

Сравнительная таблица производительности

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Критерии оценивания

Максимальный балл: 100

№	Критерий	Баллы
1	Алгоритм Прима	30
1.1	Реализация алгоритма	15
1.2	Корректность работы	10
1.3	Восстановление дерева (по варианту)	5
2	Алгоритм Краскала	35
2.1	Реализация алгоритма	15
2.2	Реализация DSU	10
2.3	Корректность работы	10

№	Критерий	Баллы
3	Применение и анализ	20
3.1	Решение прикладной задачи	10
3.2	Сравнение производительности	5
3.3	Выводы о применимости	5
4	Оформление и отчёт	15
4.1	Код отформатирован, есть комментарии	5
4.2	Есть ответы на контрольные вопросы	5
4.3	Ссылка на Git-репозиторий	5
Итого		100

Штрафные баллы

Нарушение	Штраф
Неправильная работа DSU (циклы)	-15
Неверная сложность алгоритма	-10
Отсутствие обработки несвязного графа	-10
Нет сравнения производительности	-10

Нарушение	Штраф
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №21

Динамическое программирование. Задача о рюкзаке. Наибольшая общая подпоследовательность (LCS)

Цель работы:

1. Изучить основные принципы динамического программирования: оптимальная подструктура, перекрывающиеся подзадачи, мемоизация
2. Реализовать классическую задачу о рюкзаке (0/1) с использованием табличного метода ДП
3. Реализовать алгоритм поиска наибольшей общей подпоследовательности (LCS) для двух строк

4. Научиться восстанавливать решение (какие предметы выбраны, какая подпоследовательность)
5. Сравнить рекурсивный подход с мемоизацией и итеративный подход

Теоретическое обоснование

1. Динамическое программирование

Динамическое программирование (ДП) - это метод решения задач, который разбивает сложную задачу на более простые подзадачи, решает каждую подзадачу один раз и сохраняет результат для последующего использования.

Основные принципы ДП:

Принцип	Описание
Оптимальная подструктура	Оптимальное решение задачи содержит оптимальные решения подзадач
Перекрывающиеся подзадачи	Одна и та же подзадача возникает многократно
Мемоизация	Сохранение результатов подзадач для повторного использования

Способы реализации ДП:

Способ	Описание	Сложность	Память
Рекурсия + мемоизация	Сверху вниз (top-down)	Зависит от задачи	O(таблица)

Способ	Описание	Сложность	Память
Табличный метод	Снизу вверх (bottom-up)	Зависит от задачи	O(таблица)

2. Задача о рюкзаке (0/1 Knapsack)

Постановка задачи: Дано N предметов, каждый предмет имеет вес weight[i] и стоимость value[i]. Дан рюкзак вместимостью W. Нужно выбрать предметы так, чтобы их суммарный вес не превышал W, а суммарная стоимость была максимальной. Каждый предмет можно взять только один раз (0/1).

Визуализация:

```
Предметы:
0: вес=2, ценность=3
1: вес=3, ценность=4
2: вес=4, ценность=5
3: вес=5, ценность=8

Оптимальный набор:
взять предметы 0 и 3:
суммарный вес = 2+5=7
суммарная ценность = 3+8=11
```

Таблица ДП (W=10, N=4):

i\w	0	1	2	3	4	5	6	7	8	9	10
0	0	0	3	3	3	3	3	3	3	3	3
1	0	0	3	4	4	7	7	7	7	7	7
2	0	0	3	4	5	7	8	9	9	9	9
3	0	0	3	4	5	8	8	11	12	13	13

Рекурсивное решение (без оптимизации):

```
int knapsackRecursive(int i, int W) {
    if (i < 0 || W == 0) return 0;

    if (weight[i] > W) {
        return knapsackRecursive(i - 1, W);
    } else {
        return max(knapsackRecursive(i - 1, W),
                   knapsackRecursive(i - 1, W - weight[i]) + value[i]);
```

```
    }
}
```

Рекурсивное решение с мемоизацией (top-down):

```
vector<vector<int>> memo;

int knapsackMemo(int i, int W) {
    if (i < 0 || W == 0) return 0;
    if (memo[i][W] != -1) return memo[i][W];

    if (weight[i] > W) {
        memo[i][W] = knapsackMemo(i - 1, W);
    } else {
        memo[i][W] = max(knapsackMemo(i - 1, W),
                         knapsackMemo(i - 1, W - weight[i]) + value[i]);
    }
    return memo[i][W];
}
```

Табличное решение (bottom-up):

```
int knapsackTable(int N, int W) {
    vector<vector<int>> dp(N + 1, vector<int>(W + 1, 0));

    for (int i = 1; i <= N; i++) {
        for (int w = 1; w <= W; w++) {
            if (weight[i-1] > w) {
                dp[i][w] = dp[i-1][w];
            } else {
                dp[i][w] = max(dp[i-1][w],
                               dp[i-1][w - weight[i-1]] + value[i-1]);
```

```
            }
        }
    }

    return dp[N][W];
}
```

Оптимизация по памяти (одномерный массив):

```
int knapsackOptimized(int N, int W) {
    vector<int> dp(W + 1, 0);

    for (int i = 0; i < N; i++) {
        for (int w = W; w >= weight[i]; w--) {
            dp[w] = max(dp[w], dp[w - weight[i]] + value[i]);
        }
    }

    return dp[W];
}
```

Восстановление выбранных предметов:

```
vector<int> getSelectedItems(int N, int W, const vector<vector<int>>& dp)
{
    vector<int> selected
    int w = W;

    for (int i = N; i > 0; i--) {
        if (dp[i][w] != dp[i-1][w]) {
            selected.push_back(i-1);  // предмет i-1 выбран
            w -= weight[i-1];
        }
```

```
    }

    return selected;
}
```

3. Наибольшая общая подпоследовательность (LCS)

Постановка задачи: Даны две строки (последовательности). Найти их наибольшую общую подпоследовательность (не обязательно непрерывную).

Визуализация LCS:

```
Строка X: "ABCBDAB"
Строка Y: "BDCABA"

LCS: "BCBA" или "BCAB"


Символы совпадают:
B → B, C → C, B → B

Длина LCS = 4
```

Таблица LCS (7×7):

	0	1	2	3	4	5	6	7 (Y)
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	1	1	1	1	2	2	2
3	0	1	2	2	2	2	3	3
4	0	1	2	2	2	3	3	4
5	0	1	2	3	3	3	4	4
6	0	1	2	3	4	4	4	4
7	0	1	2	3	4	4	5	5

Рекурсивное решение:

```
int lcsRecursive(const string& X, const string& Y, int i, int j) {
    if (i == 0 || j == 0) return 0;

    if (X[i-1] == Y[j-1]) {
        return 1 + lcsRecursive(X, Y, i-1, j-1);
    } else {
        return max(lcsRecursive(X, Y, i-1, j),
                   lcsRecursive(X, Y, i, j-1));
    }
}
```

Табличное решение (bottom-up):

```
int lcsTable(const string& X, const string& Y) {
    int m = X.length();
    int n = Y.length();

    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (X[i-1] == Y[j-1]) {
                dp[i][j] = dp[i-1][j-1] + 1;
            } else {
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
            }
        }
    }

    return dp[m][n];
}
```

Восстановление LCS:

```
string getLCS(const string& X, const string& Y, const vector<vector<int>>& dp) {
    int i = X.length();
    int j = Y.length();
    string lcs = "";

    while (i > 0 && j > 0) {
        if (X[i-1] == Y[j-1]) {
            lcs = X[i-1] + lcs;
            i--; j--;
```

```
        } else if (dp[i-1][j] > dp[i][j-1]) {
            i--;
        } else {
            j--;
        }
    }

    return lcs;
}
```

Примеры выполнения практической части

Пример 1. Задача о рюкзаке (базовый уровень)

Задача: Реализовать решение задачи о рюкзаке табличным методом.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int knapsack(int W, const vector<int>& weights, const vector<int>& values)
{
    int n = weights.size();
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= W; w++) {
            if (weights[i-1] > w) {
                dp[i][w] = dp[i-1][w];
            } else {
```

```
                dp[i][w] = max(dp[i-1][w],
                               dp[i-1][w - weights[i-1]] + values[i-1]);
            }
        }
    }

    return dp[n][W];
}

int main() {
    vector<int> weights = {2, 3, 4, 5};
    vector<int> values = {3, 4, 5, 8};
    int W = 10;

    int maxValue = knapsack(W, weights, values);
    cout << "Максимальная стоимость: " << maxValue << endl;

    return 0;
}
```

Ожидаемый вывод:

Максимальная стоимость: 13

Пример 2. Задача о рюкзаке с восстановлением (средний уровень)

Задача: Реализовать задачу о рюкзаке с восстановлением выбранных предметов.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;
```

```
struct KnapsackResult {
    int maxValue;
    vector<int> selectedItems;
};

KnapsackResult knapsackWithRecovery(int W, const vector<int>& weights,
const vector<int>& values) {
    int n = weights.size();
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= W; w++) {
            if (weights[i-1] > w) {
                dp[i][w] = dp[i-1][w];
            } else {
                dp[i][w] = max(dp[i-1][w],
                               dp[i-1][w - weights[i-1]] + values[i-1]);
            }
        }
    }

    // Восстановление выбранных предметов
    vector<int> selected;
    int w = W;
    for (int i = n; i > 0; i--) {
        if (dp[i][w] != dp[i-1][w]) {
            selected.push_back(i-1);
            w -= weights[i-1];
```

```
        }
    }
    reverse(selected.begin(), selected.end());

    return {dp[n][W], selected};
}

int main() {
    vector<int> weights = {2, 3, 4, 5};
    vector<int> values = {3, 4, 5, 8};
    int W = 10;

    auto result = knapsackWithRecovery(W, weights, values);

    cout << "Максимальная стоимость: " << result.maxValue << endl;
    cout << "Выбранные предметы (индексы): ";
    for (int idx : result.selectedItems) {
        cout << idx << " ";
    }
    cout << endl;
    cout << "Вес выбранных предметов: ";
    int totalWeight = 0;
    for (int idx : result.selectedItems) {
        totalWeight += weights[idx];
    }
    cout << totalWeight << " (из " << W << ")" << endl;

    return 0;
}
```

Ожидаемый вывод:

Максимальная стоимость: 13

Выбранные предметы (индексы): 0 3

Вес выбранных предметов: 7 (из 10)

Пример 3. Наибольшая общая подпоследовательность (продвинутый уровень)

Задача: Реализовать LCS с восстановлением строки.

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

struct LCSResult {
    int length;
    string subsequence;
};

LCSResult lcs(const string& X, const string& Y) {
    int m = X.length();
    int n = Y.length();

    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (X[i-1] == Y[j-1]) {
                dp[i][j] = dp[i-1][j-1] + 1;
```

```
            } else {
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
            }
        }
    }

    // Восстановление подпоследовательности
    string lcsStr = "";
    int i = m, j = n;
    while (i > 0 && j > 0) {
        if (X[i-1] == Y[j-1]) {
            lcsStr = X[i-1] + lcsStr;
            i--; j--;
        } else if (dp[i-1][j] > dp[i][j-1]) {
            i--;
        } else {
            j--;
        }
    }

    return {dp[m][n], lcsStr};
}

int main() {
    string X = "ABCBDAB";
    string Y = "BDCABA";

    auto result = lcs(X, Y);
```

```
    cout << "Строка X: " << X << endl;
    cout << "Строка Y: " << Y << endl;
    cout << "Длина LCS: " << result.length << endl;
    cout << "LCS: " << result.subsequence << endl;

    return 0;
}
```

Ожидаемый вывод:

Строка X: ABCBDAB

Строка Y: BDCABA

Длина LCS: 4

LCS: BCBA

Пример 4. Сравнение подходов (продвинутый уровень)

Задача: Сравнить производительность рекурсивного и табличного подходов для LCS.

```
#include <iostream>
#include <vector>
#include <string>
#include <chrono>
#include <functional>
#include <map>

using namespace std;
using namespace chrono;

// Рекурсивный LCS (экспоненциальный)
int lcsRecursive(const string& X, const string& Y, int i, int j) {
    if (i == 0 || j == 0) return 0;
    if (X[i-1] == Y[j-1]) {
```

```
        return 1 + lcsRecursive(X, Y, i-1, j-1);
    }
    return max(lcsRecursive(X, Y, i-1, j),
               lcsRecursive(X, Y, i, j-1));
}

// LCS с мемоизацией (top-down)
int lcsMemo(const string& X, const string& Y, int i, int j,
            vector<vector<int>>& memo) {
    if (i == 0 || j == 0) return 0;
    if (memo[i][j] != -1) return memo[i][j];

    if (X[i-1] == Y[j-1]) {
        memo[i][j] = 1 + lcsMemo(X, Y, i-1, j-1, memo);
    } else {
        memo[i][j] = max(lcsMemo(X, Y, i-1, j, memo),
                         lcsMemo(X, Y, i, j-1, memo));
    }
    return memo[i][j];
}

// LCS табличный (bottom-up)
int lcsTable(const string& X, const string& Y) {
    int m = X.length();
    int n = Y.length();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
```

```
            if (X[i-1] == Y[j-1]) {
                dp[i][j] = dp[i-1][j-1] + 1;
            } else {
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
            }
        }
    }
    return dp[m][n];
}

int main() {
    string X = "ABCBDAB";
    string Y = "BDCABA";

    cout << "=== Сравнение подходов LCS ===" << endl;
    cout << "X: " << X << " (длина " << X.length() << ")" << endl;
    cout << "Y: " << Y << " (длина " << Y.length() << ")" << endl;

    // Рекурсивный (только для маленьких строк)
    auto start = high_resolution_clock::now();
    int lenRec = lcsRecursive(X, Y, X.length(), Y.length());
    auto end = high_resolution_clock::now();
    auto timeRec = duration_cast<microseconds>(end - start).count();
    cout << "\nРекурсивный: длина = " << lenRec << ", время = " <<
timeRec << " мкс" << endl;

    // С мемоизацией
    vector<vector<int>> memo(X.length() + 1, vector<int>(Y.length() + 1, -
1));
```

```
    start = high_resolution_clock::now();
    int lenMemo = lcsMemo(X, Y, X.length(), Y.length(), memo);
    end = high_resolution_clock::now();
    auto timeMemo = duration_cast<microseconds>(end - start).count();
    cout << "С мемоизацией: длина = " << lenMemo << ", время = " << timeMemo << " мкс" << endl;

    // Табличный
    start = high_resolution_clock::now();
    int lenTable = lcsTable(X, Y);
    end = high_resolution_clock::now();
    auto timeTable = duration_cast<microseconds>(end - start).count();
    cout << "Табличный: длина = " << lenTable << ", время = " << timeTable << " мкс" << endl;

    return 0;
}
```

Ожидаемый вывод:

=== Сравнение подходов LCS ===

X: ABCBDAB (длина 7)

Y: BDCABA (длина 6)

Рекурсивный: длина = 4, время = 45 мкс

С мемоизацией: длина = 4, время = 15 мкс

Табличный: длина = 4, время = 8 мкс

Индивидуальные задания

Часть 1. Задача о рюкзаке

Каждый вариант содержит задание реализовать решение задачи о рюкзаке с указанными особенностями:

№ вар.	**Размерность**	**Особенности**
1	2D таблица	Только максимальная стоимость
2	2D таблица	С восстановлением предметов
3	2D таблица	Веса и ценности - целые числа
4	2D таблица	Веса и ценности - вещественные
5	1D оптимизация	Только максимальная стоимость
6	1D оптимизация	С восстановлением предметов
7	Рекурсия + мемоизация	Только максимальная стоимость
8	Рекурсия + мемоизация	С восстановлением предметов
9	Все три подхода	Сравнение производительности

№ вар.	Размерность	Особенности
10	2D таблица	Ограниченный рюкзак (не более K предметов)
11	2D таблица	Рюкзак с несколькими ограничениями (вес + объём)
12	1D оптимизация	Вывод всех оптимальных наборов
13	Рекурсия + мемоизация	Непрерывный рюкзак (дробный)
14	2D таблица	Зависимые предметы (взять A → можно взять B)
15	2D таблица	Максимизация веса при фиксированной стоимости

Часть 2. Наибольшая общая подпоследовательность (LCS)

Каждый вариант содержит задание реализовать LCS с указанными особенностями:

№ вар.	Особенности	Дополнительное задание
1	Табличный метод	Только длина LCS
2	Табличный метод	Восстановление

№ вар.	Особенности	Дополнительное задание
		строки
3	Табличный метод	Все LCS (все варианты)
4	Табличный метод	LCS для трёх строк
5	Рекурсивный (без оптимизации)	Сравнение производительности
6	Рекурсия + мемоизация	Сравнение производительности
7	Табличный метод	LCS для строк с кириллицей
8	Табличный метод	LCS для массивов чисел
9	Табличный метод	LCS с весами символов
10	Табличный метод	LCS с ограничением на длину
11	Табличный метод	LCS с пропусками (gap penalty)
1	Табличный метод	Наибольшая общая

№ вар.	Особенности	Дополнительное задание
2		подстрока (непрерывная)
13	Табличный метод	LCS для векторов строк
14	Табличный метод	LCS для списков (шаблонный)
15	Табличный метод	LCS с восстановлением и визуализацией таблицы

Часть 3. Применение ДП

Задание для всех вариантов:

1. Реализовать программу, которая:
2. Запрашивает у пользователя:
3. Для задачи о рюкзаке: количество предметов, вместимость, вес и ценность каждого предмета
4. Для LCS: две строки
5. Вычисляет оптимальное решение
6. Выводит результат (максимальную стоимость и выбранные предметы / длину LCS и саму подпоследовательность)
7. Выводит таблицу ДП для наглядности

Пример входных данных для рюкзака:

Количество предметов: 4

Вместимость рюкзака: 10

Предмет 0: вес=2, ценность=3

Предмет 1: вес=3, ценность=4

Предмет 2: вес=4, ценность=5

Предмет 3: вес=5, ценность=8

Пример входных данных для LCS:

Строка X: ABCBDAB

Строка Y: BDCABA

Контрольные вопросы

1. Что такое динамическое программирование? Какие принципы лежат в его основе?
2. В чём разница между рекурсивным подходом с мемоизацией (top-down) и табличным (bottom-up)?
3. Как формулируется задача о рюкзаке (0/1)?
4. Какова временная сложность решения задачи о рюкзаке методом ДП?
5. Почему для оптимизации памяти нужно проходить по весам в обратном порядке?
6. Как восстановить выбранные предметы после заполнения таблицы ДП?
7. Как формулируется задача LCS?
8. Какова временная сложность алгоритма LCS?
9. В чём разница между подпоследовательностью и подстрокой?
10. Как восстановить LCS после заполнения таблицы?
11. Какие ещё задачи решаются с помощью ДП?
12. В чём отличие задачи о рюкзаке от задачи о рюкзаке с повторениями?
13. Можно ли решить LCS за меньшее время, чем O(m×n)?
14. Как адаптировать ДП для LCS трёх строк?

15. Как модифицировать задачу о рюкзаке для случая, когда предмет можно взять несколько раз?

Требования к отчёту

Титульный лист

Цель работы

Теоретическая часть (ДП, задача о рюкзаке, LCS)

Постановка задачи (согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода)

Анализ сложности алгоритмов

Сравнительная таблица подходов (при наличии)

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Критерии оценивания

Максимальный балл: 100

№	Критерий	Баллы
1	Задача о рюкзаке	40
1.1	Реализация алгоритма	15
1.2	Корректность работы	10
1.3	Восстановление предметов (по варианту)	10
1.4	Оптимизация памяти (по	5

№	Критерий	Баллы
	варианту)	
2	Наибольшая общая подпоследовательность (LCS)	40
2.1	Реализация алгоритма	15
2.2	Корректность работы	10
2.3	Восстановление подпоследовательности	10
2.4	Дополнительные возможности (по варианту)	5
3	Применение и анализ	10
3.1	Демонстрационная программа	5
3.2	Анализ сложности	5
4	Оформление и отчёт	10
4.1	Код отформатирован, есть комментарии	5
4.2	Ссылка на Git-репозиторий	5
Итог		100

№	Критерий	Баллы
о		

Штрафные баллы

Нарушение	Штраф
Неправильное заполнение таблицы ДП	-15
Неверное восстановление выбранных предметов/LCS	-15
Неверная сложность алгоритма	-10
Нет демонстрационной программы	-10
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №22

Алгоритм Кнута-Морриса-Пратта. STL (обзор)

Цель работы:

1. Изучить алгоритм Кнута-Морриса-Пратта (КМП) для эффективного поиска подстроки в строке
2. Реализовать построение префикс-функции для заданной строки
3. Реализовать поиск всех вхождений образца в текст с использованием КМП
4. Освоить основные контейнеры и алгоритмы стандартной библиотеки шаблонов (STL)
5. Научиться применять STL для решения практических задач

Теоретическое обоснование

1. Поиск подстроки: постановка задачи

Задача поиска подстроки (pattern matching): Даны текст T длины n и образец (pattern) P длины m. Найти все позиции в тексте, где начинается вхождение образца.

Алгоритм	Временная сложность	Память
Наивный	O(n × m)	O(1)
КМП	O(n + m)	O(m)

2. Наивный алгоритм поиска

```
vector<int> naiveSearch(const string& text, const string& pattern) {
    vector<int> positions;
    int n = text.length();
    int m = pattern.length();

    for (int i = 0; i <= n - m; i++) {
```

```
        bool found = true;
        for (int j = 0; j < m  j++) {
            if (text[i + j] != pattern[j]) {
                found = false;
                break;
            }
        }
        if (found) positions.push_back(i);
    }
    return positions;
}
```

Проблема: При несовпадении сдвигаемся на 1 позицию, теряя уже проверенную информацию.

3. Префикс-функция (Prefix Function)

Префикс-функция π[i] - это длина наибольшего собственного префикса строки s[0..i], который также является её суффиксом.

Визуализация префикс-функции для строки "ababaca":

Индекс: 0 1 2 3 4 5 6
Строка: a b a b a c a
π: 0 0 1 2 3 0 1

π[0] = 0 (префикс "a" - нет собственного префикса)
π[1] = 0 ("ab" - префиксы: "a", суффиксы: "b" → нет совпадений)
π[2] = 1 ("aba" - префикс "a" = суффикс "a")
π[3] = 2 ("abab" - префикс "ab" = суффикс "ab")

π[4] = 3 ("ababa" - префикс "aba" = суффикс "aba")

π[5] = 0 ("ababac" - нет совпадений)

π[6] = 1 ("ababaca" - префикс "a" = суффикс "a")

Алгоритм вычисления префикс-функции O(n):

```
vector<int> computePrefixFunction(const string& pattern) {
    int m = pattern.length();
    vector<int> pi(m, 0);

    for (int i = 1; i < m; i++) {
        int j = pi[i - 1];
        while (j > 0 && pattern[i] != pattern[j]) {
            j = pi[j - 1];
        }
        if (pattern[i] == pattern[j]) {
            j++;
        }
        pi[i] = j;
    }

    return pi;
}
```

4. Алгоритм Кнута-Морриса-Пратта (КМП)

Принцип работы: Использует префикс-функцию для эффективного сдвига при несовпадении, не возвращаясь назад по тексту.

Визуализация работы КМП:

Текст: a b a b a b c a

Образец: a b a b c

Префикс-функция образца: [0,0,1,2,0]

Сравнение:

a b a b a b c a

a b a b c

↑ несовпадение (b ≠ c)

По префикс-функции: для j=4, π[3]=2 → сдвигаем образец на (4-2)=2 позиции

a b a b a b c a

a b a b c

↑ продолжаем сравнение

Алгоритм поиска O(n + m):

```
vector<int> kmpSearch(const string& text, const string& pattern) {
    vector<int> positions;
    int n = text.length();
    int m = pattern.length();

    if (m == 0) return positions;

    vector<int> pi = computePrefixFunction(pattern);

    int j = 0;  // количество совпавших символов образца
    for (int i = 0; i < n; i++) {
        while (j > 0 && text[i] != pattern[j]) {
            j = pi[j - 1];
        }
```

```
        if (text[i] == pattern[j]) {
            j++;
        }
        if (j == m) {
            positions.push_back(i - m + 1);
            j = pi[j - 1];
        }
    }

    return positions;
}
```

5. Обзор стандартной библиотеки шаблонов (STL)

STL - это мощная библиотека C++, состоящая из контейнеров, итераторов, алгоритмов и функциональных объектов.

Контейнеры STL

Категория	Контейнеры	Описание
Последовательные	vector, list, deque, array	Линейное хранение элементов
Ассоциативные	set, map, multiset, multimap	Хранение по ключу (сбалансированное дерево)
Неупорядоченные	unordered_set, unordered_map	Хранение по ключу (хеш-таблица)

Категория	Контейнеры	Описание
Адаптеры	stack, queue, priority_ queue	Ограниченные интерфейсы

Итераторы

```
// Пример использования итераторов
vector<int> v = {1, 2, 3, 4, 5};

// Обычный итератор
for (auto it = v.begin(); it != v.end(); ++it) {
    cout << *it << " ";
}

// Константный итератор
for (auto it = v.cbegin(); it != v.cend(); ++it) {
    // *it = 10; // Ошибка!
}

// Обратный итератор
for (auto it = v.rbegin(); it != v.rend(); ++it) {
    cout << *it << " ";
}

// Range-based for (C++11)
for (int x : v) {
    cout << x << " ";
}
```

Алгоритмы STL

cpp

```cpp
#include <algorithm>
#include <numeric>

vector<int> v = {5, 2, 8, 1, 9, 3};

// Сортировка
sort(v.begin(), v.end());                    // {1, 2, 3, 5, 8, 9}
sort(v.begin(), v.end(), greater<int>());    // {9, 8, 5, 3, 2, 1}

// Поиск
auto it = find(v.begin(), v.end(), 5);       // ищет значение 5
auto it2 = find_if(v.begin(), v.end(), [](int x) { return x > 5; });

// Копирование
vector<int> dest(v.size());
copy(v.begin(), v.end(), dest.begin());

// Преобразование
transform(v.begin(), v.end(), v.begin(), [](int x) { return x * 2; });

// Подсчёт
int cnt = count(v.begin(), v.end(), 3);
int cntIf = count_if(v.begin(), v.end(), [](int x) { return x % 2 == 0; });

// Удаление (erase-remove idiom)
v.erase(remove(v.begin(), v.end(), 5), v.end());

// Минимум/максимум
```

```
int mn = *min_element(v.begin(), v.end());
int mx = *max_element(v.begin(), v.end());

// Сумма (numeric)
int sum = accumulate(v.begin(), v.end(), 0);
```

Примеры выполнения практической части

Пример 1. Префикс-функция (базовый уровень)

Задача: Реализовать вычисление префикс-функции для заданной строки.

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

vector<int> computePrefixFunction(const string& s) {
    int n = s.length();
    vector<int> pi(n, 0);

    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j]) {
            j = pi[j - 1];
        }
        if (s[i] == s[j]) {
            j++;
        }
```

```
            pi[i] = j;
        }

        return pi;
    }

    int main() {
        string s = "ababaca";
        vector<int> pi = computePrefixFunction(s);

        cout << "Строка: " << s << endl;
        cout << "Префикс-функция: ";
        for (int x : pi) {
            cout << x << " ";
        }
        cout << endl;

        return 0;
    }
```

Ожидаемый вывод:

Строка: ababaca

Префикс-функция: 0 0 1 2 3 0 1

Пример 2. Алгоритм КМП (средний уровень)

Задача: Реализовать поиск всех вхождений образца в текст с помощью КМП.

```
#include <iostream>
#include <vector>
```

```cpp
#include <string>

using namespace std;

vector<int> computePrefixFunction(const string& pattern) {
    int m = pattern.length();
    vector<int> pi(m, 0);

    for (int i = 1; i < m; i++) {
        int j = pi[i - 1];
        while (j > 0 && pattern[i] != pattern[j]) {
            j = pi[j - 1];
        }
        if (pattern[i] == pattern[j]) {
            j++;
        }
        pi[i] = j;
    }

    return pi;
}

vector<int> kmpSearch(const string& text, const string& pattern) {
    vector<int> positions;
    int n = text.length();
    int m = pattern.length();

    if (m == 0) return positions;
    if (m > n) return positions;
```

```
    vector<int> pi = computePrefixFunction(pattern);

    int j = 0;
    for (int i = 0; i < n; i++) {
        while (j > 0 && text[i] != pattern[j]) {
            j = pi[j - 1];
        }
        if (text[i] == pattern[j]) {
            j++;
        }
        if (j == m) {
            positions.push_back(i - m + 1);
            j = pi[j - 1];
        }
    }

    return positions;
}

int main() {
    string text = "ababaababcabab";
    string pattern = "abab";

    vector<int> positions = kmpSearch(text, pattern);

    cout << "Текст: " << text << endl;
    cout << "Образец: " << pattern << endl;
    cout << "Вхождения на позициях: ";
```

```
    for (int pos : positions) {
        cout << pos << " ";
    }
    cout << endl;

    // Визуализация
    cout << "\nВизуализация:" << endl;
    cout << text << endl;
    for (int pos : positions) {
        for (int i = 0; i < pos; i++) cout << " ";
        cout << pattern << endl;
    }

    return 0;
}
```

Ожидаемый вывод:

Текст: ababaababcabab
Образец: abab
Вхождения на позициях: 0 2 6 10

Визуализация:
ababaababcabab
abab
  abab
    abab
      abab

Пример 3. Сравнение наивного и КМП алгоритмов (продвинутый уровень)

Задача: Сравнить производительность наивного и КМП алгоритмов.

```
#include <iostream>
#include <vector>
#include <string>
#include <chrono>
#include <random>

using namespace std;
using namespace chrono;

// Наивный алгоритм
vector<int> naiveSearch(const string& text, const string& pattern) {
    vector<int> positions;
    int n = text.length();
    int m = pattern.length();

    for (int i = 0; i <= n - m; i++) {
        bool found = true;
        for (int j = 0; j < m; j++) {
            if (text[i + j] != pattern[j]) {
                found = false;
                break;
            }
        }
        if (found) positions.push_back(i);
    }
    return positions;
```

```
}

// Префикс-функция
vector<int> computePrefixFunction(const string& pattern) {
    int m = pattern.length();
    vector<int> pi(m, 0);

    for (int i = 1; i < m; i++) {
        int j = pi[i - 1];
        while (j > 0 && pattern[i] != pattern[j]) {
            j = pi[j - 1];
        }
        if (pattern[i] == pattern[j]) {
            j++;
        }
        pi[i] = j;
    }
    return pi;
}

// Алгоритм КМП
vector<int> kmpSearch(const string& text, const string& pattern) {
    vector<int> positions;
    int n = text.length();
    int m = pattern.length();

    if (m == 0 || m > n) return positions;

    vector<int> pi = computePrefixFunction(pattern);
```

```
    int j = 0;

    for (int i = 0; i < n; i++) {
        while (j > 0 && text[i] != pattern[j]) {
            j = pi[j - 1];
        }
        if (text[i] == pattern[j]) {
            j++;
        }
        if (j == m) {
            positions.push_back(i - m + 1);
            j = pi[j - 1];
        }
    }
    return positions;
}

// Генерация случайной строки
string generateRandomString(int length, int alphabetSize = 2) {
    string result;
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(0, alphabetSize - 1);

    for (int i = 0; i < length; i++) {
        result += 'a' + dis(gen);
    }
    return result;
}
```

```
int main() {
    cout << "=== Сравнение алгоритмов поиска подстроки ===" << endl;

    vector<int> lengths = {1000, 2000, 5000, 10000, 20000};

    cout << "\nТекст: случайный алфавит (a,b)" << endl;
    cout << "Образец: \"ababbaba\"" << endl;
    cout << "\nДлина текста\tНаивный (мкс)\tКМП (мкс)" << endl;

    string pattern = "ababbaba";

    for (int n : lengths) {
        string text = generateRandomString(n);

        // Наивный алгоритм
        auto start = high_resolution_clock::now();
        auto naiveResult = naiveSearch(text, pattern);
        auto end = high_resolution_clock::now();
        auto naiveTime = duration_cast<microseconds>(end - start).count();

        // Алгоритм КМП
        start = high_resolution_clock::now();
        auto kmpResult = kmpSearch(text, pattern);
        end = high_resolution_clock::now();
        auto kmpTime = duration_cast<microseconds>(end - start).count();

        cout << n << "\t\t" << naiveTime << "\t\t" << kmpTime << endl;
    }
```

```
    return 0;
}
```

Пример 4. Применение STL (обзор)

Задача: Продемонстрировать основные возможности STL.

```
#include <iostream>
#include <vector>
#include <list>
#include <deque>
#include <set>
#include <map>
#include <unordered_set>
#include <unordered_map>
#include <algorithm>
#include <numeric>
#include <string>

using namespace std;

int main() {
    cout << "=== STL: контейнеры и алгоритмы ===" << endl;

    // 1. vector - динамический массив
    cout << "\n1. vector:" << endl;
    vector<int> v = {5, 2, 8, 1, 9, 3};
    cout << "Исходный: ";
    for (int x : v) cout << x << " ";
    cout << endl;
```

```
sort(v.begin(), v.end());
cout << "Отсортированный: ";
for (int x : v) cout << x << " ";
cout << endl;

v.push_back(10);
cout << "После push_back(10): ";
for (int x : v) cout << x << " ";
cout << endl;

// 2. list - двусвязный список
cout << "\n2. list:" << endl;
list<int> lst = {1, 2, 3, 4, 5};
lst.push_front(0);
lst.push_back(6);
cout << "Элементы: ";
for (int x : lst) cout << x << " ";
cout << endl;

// 3. set - упорядоченное множество
cout << "\n3. set:" << endl;
set<int> s = {3, 1, 4, 1, 5, 9, 2, 6};
cout << "Уникальные элементы: ";
for (int x : s) cout << x << " ";
cout << endl;
cout << "Есть ли 5? " << (s.count(5) ? "да" : "нет") << endl;

// 4. map - словарь
```

```
    cout << "\n4. map:" << endl;
    map<string, int> ages = {{"Alice", 25}, {"Bob", 30}, {"Charlie", 35}};
    ages["David"] = 28;
    for (const auto& [name, age] : ages) {
        cout << name << ": " << age << endl;
    }

    // 5. unordered_set - хеш-множество
    cout << "\n5. unordered_set:" << endl;
    unordered_set<string> words = {"hello", "world", "c++", "stl"};
    words.insert("algorithm");
    for (const string& w : words) {
        cout << w << " ";
    }
    cout << endl;

    // 6. Алгоритмы STL
    cout << "\n6. Алгоритмы STL:" << endl;
    vector<int> nums = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    // find
    auto it = find(nums.begin(), nums.end(), 7);
    cout << "find(7): позиция " << (it - nums.begin()) << endl;

    // count_if
    int evenCount = count_if(nums.begin(), nums.end(), [](int x) { return x %
2 == 0; });
    cout << "Количество чётных: " << evenCount << endl;
```

```
    // transform
    vector<int> squares(nums.size());
    transform(nums.begin(), nums.end(), squares.begin(), [](int x) { return x * x; });
    cout << "Квадраты: ";
    for (int x : squares) cout << x << " ";
    cout << endl;

    // accumulate
    int sum = accumulate(nums.begin(), nums.end(), 0);
    cout << "Сумма: " << sum << endl;

    // reverse
    reverse(nums.begin(), nums.end());
    cout << "После reverse: ";
    for (int x : nums) cout << x << " ";
    cout << endl;

    return 0;
}
```

Индивидуальные задания

Часть 1. Алгоритм КМП

Каждый вариант содержит задание реализовать алгоритм КМП с указанными особенностями:

№ вар.	Особенности	Дополнительное задание
1	Префикс-функция	Вывести все вхождения
2	Префикс-функция	Вывести первое вхождение
3	КМП поиск	Вывести все вхождения
4	КМП поиск	Вывести первое вхождение
5	КМП поиск	Подсчитать количество вхождений
6	КМП поиск	Поиск с перекрытием
7	КМП поиск	Поиск без перекрытия
8	КМП поиск	Регистронезависимый поиск
9	КМП поиск	Поиск для кириллицы
10	КМП поиск	Поиск всех образцов из списка
11	КМП поиск	Замена всех вхождений
12	КМП поиск	Наибольшее совпадение с префиксом
13	КМП поиск	Наибольшее совпадение с суффиксом

№ вар.	Особенности	Дополнительное задание
14	КМП поиск	Поиск в файле
15	КМП поиск	Сравнение с наивным алгоритмом

Часть 2. STL (обзор)

Каждый вариант содержит задание с использованием STL для решения задачи:

№ вар.	Задача	Требуемые контейнеры/алгоритмы
1	Подсчёт частоты слов в тексте	map<string, int>, istringstream
2	Удаление дубликатов из массива	vector, sort, unique
3	Проверка на анаграмму	sort, string
4	Нахождение пересечения двух множеств	set, set_intersection
5	Нахождение объединения двух множеств	set, set_union

№ вар.	Задача	Требуемые контейнеры/алгоритмы
6	Сортировка студентов по баллам	vector, sort, pair
7	Поиск ближайшего числа в отсортированном массиве	vector, lower_bound
8	Подсчёт уникальных элементов	unordered_set
9	Частотный словарь символов	unordered_map<char, int>
10	Топ K самых частых слов	map, vector, sort
11	Проверка палиндрома	string, reverse, equal
12	Циклический сдвиг	deque, rotate
13	Объединение двух отсортированных списков	list, merge
14	Фильтрация по условию	vector, copy_if

№ вар.	Задача	Требуемые контейнеры/алгоритмы
15	Группировка элементов по признаку	map, partition

Часть 3. Применение

Задание для всех вариантов:

1. Реализовать программу, которая:
2. Считывает текст из файла или с клавиатуры
3. Считывает образец для поиска
4. Используя КМП, находит все вхождения образца
5. Выводит позиции вхождений
6. Используя STL, выполняет дополнительную обработку (по варианту)

Контрольные вопросы

1. В чём заключается идея алгоритма Кнута-Морриса-Пратта?
2. Что такое префикс-функция и как она вычисляется?
3. Какова временная сложность алгоритма КМП? Почему?
4. Как работает восстановление по префикс-функции при несовпадении?
5. В чём преимущество КМП перед наивным алгоритмом?
6. Какие контейнеры STL вы знаете? Какой контейнер когда лучше использовать?
7. В чём разница между vector и list?
8. В чём разница между set и unordered_set?
9. В чём разница между map и unordered_map?
10. Как работают итераторы в STL?
11. Какие алгоритмы STL вы знаете? Приведите примеры.
12. Что такое лямбда-выражения в C++? Как они применяются с алгоритмами STL?
13. Как удалить элементы из контейнера во время итерации?
14. Что такое идиома erase-remove?

15.Как можно изменить порядок обхода контейнера?

Требования к отчёту

Титульный лист

Цель работы

Теоретическая часть (КМП, префикс-функция, STL)

Постановка задачи (согласно варианту)

Листинг программы с комментариями

Результаты работы (скриншоты вывода)

Анализ сложности алгоритма КМП

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Критерии оценивания

Максимальный балл: 100

№	Критерий	Баллы
1	Префикс-функция	20
1.1	Реализация алгоритма	10
1.2	Корректность работы	10
2	Алгоритм КМП	35
2.1	Реализация поиска	15
2.2	Корректность работы	15
2.3	Дополнительные	5

№	Критерий	Баллы
	возможности (по варианту)	
3	STL (обзор)	30
3.1	Корректное использование контейнеров	15
3.2	Корректное использование алгоритмов	15
4	Оформление и отчёт	15
4.1	Код отформатирован, есть комментарии	5
4.2	Есть ответы на контрольные вопросы	5
4.3	Ссылка на Git-репозиторий	5
Итого		100

Штрафные баллы

Нарушение	Штраф
Неправильное вычисление префикс-функции	-15
Неправильная работа КМП	-15
Неэффективное использование STL	-10

Нарушение	Штраф
Нет ответов на контрольные вопросы	-10
Нет ссылки на Git-репозиторий	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

Лабораторная работа №23

Проектирование иерархии классов. Наследование и полиморфизм

Цель работы:

1. Освоить принципы объектно-ориентированного проектирования иерархий классов
2. Научиться использовать наследование для создания родственных типов
3. Освоить виртуальные методы и позднее связывание (полиморфизм)
4. Изучить абстрактные классы и чисто виртуальные методы
5. Освоить построение диаграмм классов UML для визуализации архитектуры

6. Разработать проектное решение для выбранной предметной области

Теоретическое обоснование

1. Наследование (Inheritance)

Наследование - механизм ООП, позволяющий создавать новый класс на основе существующего, заимствуя его поля и методы и добавляя новые или изменяя существующие.

Отношение «is a»: Производный класс является частным случаем базового.

```
// Базовый класс (родитель)
class Animal {
protected:
    string name;
    int age;
public:
    Animal(const string& n, int a) : name(n), age(a) {}
    virtual void speak() const { cout << "Some sound" << endl; }
    virtual ~Animal() {}
};

// Производный класс (наследник)
class Dog : public Animal {
private:
    string breed;
public:
    Dog(const string& n, int a, const string& b) : Animal(n, a), breed(b) {}
    void speak() const override { cout << "Woof! Woof!" << endl; }
```

```
    void wagTail() const { cout << "Tail wagging" << endl; }
};
```

2. Виды наследования в C++

Тип наследования	public	protected	private
public	public → public	protected → protected	private → недоступен
protected	public → protected	protected → protected	private → недоступен
private	public → private	protected → private	private → недоступен

Правило: Отношение «is a» реализуется только через public наследование.

3. Виртуальные методы (Virtual Functions)

Виртуальный метод - метод, который может быть переопределён в производном классе, а вызов через указатель на базовый класс будет направлен к реализации производного класса (позднее связывание).

```
class Base {
public:
    virtual void show() { cout << "Base" << endl; }
    virtual ~Base() {}  // Виртуальный деструктор ОБЯЗАТЕЛЕН!
};
```

```
class Derived : public Base {
public:
    void show() override { cout << "Derived" << endl; }
};

int main() {
    Base* ptr = new Derived();
    ptr->show();  // Вывод: Derived (полиморфизм!)
    delete ptr;
}
```

4. Абстрактные классы (Abstract Classes)

Абстрактный класс - класс, содержащий хотя бы один чисто виртуальный метод. Нельзя создавать объекты абстрактного класса.

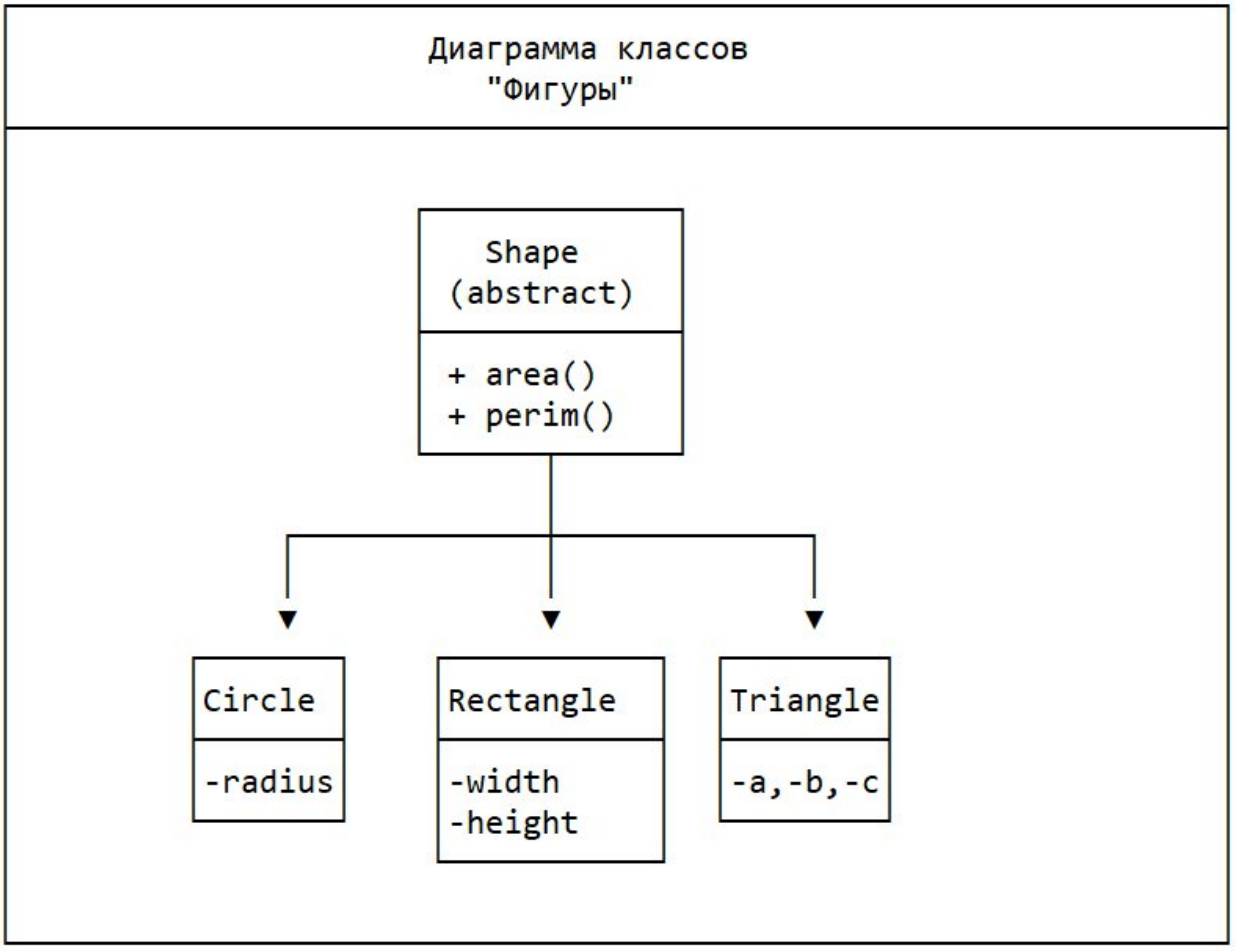
```
class Shape {  // Абстрактный класс
public:
    virtual double area() const = 0;      // Чисто виртуальный метод
    virtual double perimeter() const = 0; // Чисто виртуальный метод
    virtual ~Shape() {}
};

class Circle : public Shape {
private:
    double radius;
public:
    Circle(double r) : radius(r) {}
    double area() const override { return 3.14159 * radius * radius; }
    double perimeter() const override { return 2 * 3.14159 * radius; }
```

};

5. Диаграммы классов UML

Символ	Значение	Пример
▷ (пустая стрелка)	Наследование (is a)	Circle ▷ Shape
◇ (пустой ромб)	Агрегация (has a)	Library ◇ Book
◆ (закрашенный ромб)	Композиция (has a)	House ◆ Room
→ (пунктир)	Зависимость (uses)	Main → Shape

Пример диаграммы классов:



6. Паттерны проектирования (краткое введение)

Паттерн «Стратегия» (Strategy) - инкапсулирует семейство алгоритмов, делая их взаимозаменяемыми.

```
// Интерфейс стратегии
class SortStrategy {
public:
    virtual void sort(vector<int>& data) = 0;
    virtual ~SortStrategy() {}
};

// Конкретные стратегии
```

```
class BubbleSort : public SortStrategy {
public:
    void sort(vector<int>& data) override { /* пузырьковая сортировка */ }
};

class QuickSort : public SortStrategy {
public:
    void sort(vector<int>& data) override { /* быстрая сортировка */ }
};

// Контекст
class Sorter {
private:
    unique_ptr<SortStrategy> strategy;
public:
    void setStrategy(unique_ptr<SortStrategy> s) { strategy = move(s); }
    void sort(vector<int>& data) { strategy->sort(data); }
};
```

Пример выполнения практической части

Сквозной пример: Иерархия транспортных средств

Задача: Разработать иерархию классов для представления транспортных средств.

```
#include <iostream>
#include <string>
#include <vector>
#include <memory>
```

```
using namespace std;

// === Базовый абстрактный класс ===
class Vehicle {
protected:
    string brand;
    string model;
    int year;
    double speed;  // текущая скорость

public:
    Vehicle(const string& b, const string& m, int y)
        : brand(b), model(m), year(y), speed(0) {}

    virtual ~Vehicle() {}

    // Чисто виртуальные методы
    virtual double getMaxSpeed() const = 0;
    virtual double getFuelConsumption() const = 0;
    virtual string getType() const = 0;

    // Виртуальные методы с реализацией по умолчанию
    virtual void accelerate(double delta) {
        speed += delta;
        if (speed > getMaxSpeed()) speed = getMaxSpeed();
        cout << brand << " " << model << " разогнался до " << speed << "
км/ч" << endl;
    }
```

```
    virtual void brake(double delta) {
        speed -= delta;
        if (speed < 0) speed = 0;
        cout << brand << " " << model << " замедлился до " << speed << "
км/ч" << endl;
    }

    // Обычный метод
    void printInfo() const {
        cout << "=== " << brand << " " << model << " ===" << endl;
        cout << "Год выпуска: " << year << endl;
        cout << "Тип: " << getType() << endl;
        cout << "Макс. скорость: " << getMaxSpeed() << " км/ч" << endl;
        cout << "Расход топлива: " << getFuelConsumption() << " л/100км"
<< endl;
        cout << "Текущая скорость: " << speed << " км/ч" << endl;
    }
};

// === Производные классы ===
class Car : public Vehicle {
private:
    int passengerCount;
    double trunkVolume;

public:
    Car(const string& b, const string& m, int y, int passengers, double trunk)
        : Vehicle(b, m, y), passengerCount(passengers), trunkVolume(trunk) {}
```

```
    double getMaxSpeed() const override {
        if (brand == "Ferrari") return 350;
        if (brand == "Toyota") return 220;
        return 200;
    }

    double getFuelConsumption() const override {
        if (brand == "Ferrari") return 15.0;
        if (brand == "Toyota") return 8.0;
        return 10.0;
    }

    string getType() const override { return "Легковой автомобиль"; }

    void printPassengerInfo() const {
        cout << "Пассажировместимость: " << passengerCount << " чел." <<
endl;
        cout << "Объём багажника: " << trunkVolume << " л" << endl;
    }
};

class Truck : public Vehicle {
private:
    double loadCapacity;  // грузоподъёмность (тонн)
    int axles;            // количество осей

public:
```

```
    Truck(const string& b, const string& m, int y, double capacity, int axlesCount)
        : Vehicle(b, m, y), loadCapacity(capacity), axles(axlesCount) {}

    double getMaxSpeed() const override {
        if (loadCapacity > 20) return 90;
        return 110;
    }

    double getFuelConsumption() const override {
        return 25.0 + loadCapacity * 0.5;
    }

    string getType() const override { return "Грузовой автомобиль"; }

    void printLoadInfo() const {
        cout << "Грузоподъёмность: " << loadCapacity << " т" << endl;
        cout << "Количество осей: " << axles << endl;
    }
};

class Motorcycle : public Vehicle {
private:
    bool hasSidecar;

public:
    Motorcycle(const string& b, const string& m, int y, bool sidecar)
        : Vehicle(b, m, y), hasSidecar(sidecar) {}
```

```
    double getMaxSpeed() const override { return 220; }

    double getFuelConsumption() const override { return hasSidecar ? 6.0 : 4.5; }

    string getType() const override {
        return hasSidecar ? "Мотоцикл с коляской" : "Мотоцикл";
    }

    void wheelie() const {
        cout << brand << " " << model << " встаёт на заднее колесо!" << endl;
    }
};

// === Функция, демонстрирующая полиморфизм ===
void testVehicle(Vehicle* v) {
    cout << "\n--- Тестирование транспортного средства ---" << endl;
    v->printInfo();
    v->accelerate(50);
    v->brake(20);
}

// === Главная функция ===
int main() {
    setlocale(LC_ALL, "ru");

    // Создание объектов разных типов
    Car ferrari("Ferrari", "F40", 1987, 2, 120);
```

```
    Truck volvo("Volvo", "FH16", 2020, 25.0, 3);
    Motorcycle harley("Harley-Davidson", "Street Glide", 2019, false);

    // Полиморфное использование через указатель на базовый класс
    vector<Vehicle*> vehicles = {&ferrari, &volvo, &harley};

    for (auto v : vehicles) {
        testVehicle(v);
    }

    // Вызов специфичных методов (требуется приведение типа)
    cout << "\n--- Специфичные методы ---" << endl;
    ferrari.printPassengerInfo();
    volvo.printLoadInfo();
    harley.wheelie();

    return 0;
}
```

Ожидаемый вывод:

=== Ferrari F40 ===

Год выпуска: 1987

Тип: Легковой автомобиль

Макс. скорость: 350 км/ч

Расход топлива: 15 л/100км

Текущая скорость: 0 км/ч

Ferrari F40 разогнался до 50 км/ч

Ferrari F40 замедлился до 30 км/ч

--- Аналогично для Volvo и Harley-Davidson ---

Индивидуальные задания

Базовый уровень (15 вариантов)

Общее задание: Разработать иерархию классов, состоящую из базового абстрактного класса и 2-3 производных классов, согласно варианту. Построить диаграмму классов UML. Продемонстрировать полиморфизм.

№ вар.	Тема	Базовый класс	Поля базового	Производные классы	Специфичные методы
1	Животные	Animal	имя, возраст, вес	Dog, Cat, Bird	bark(), meow(), fly()
2	Сотрудники	Employee	имя, ID, стаж	Manager, Developer, Intern	manage(), code(), learn()
3	Фигуры	Shape	цвет, позиция	Circle, Rectangle, Triangle	area(), perimeter()
4	Транспорт	Vehicle	марка, модель, год	Car, Truck, Motorcycle	openTrunk(), loadCargo(), wheelie()
5	Банковские счета	Account	номер, баланс, владелец	SavingsAccount, CheckingAccount, CreditAccount	applyInterest(), withdrawFee(), calculateCredit()

№ вар.	Тема	Базовый класс	Поля базового	Производные классы	Специфичные методы
6	Студенты	Student	имя, курс, группа	UndergradStudent, GradStudent, PhdStudent	attendClass(), doResearch(), defendThesis()
7	Игровые персонажи	Character	имя, здоровье, уровень	Warrior, Mage, Archer	slash(), castSpell(), shoot()
8	Электроника	Device	бренд, модель, цена	Laptop, Smartphone, Tablet	installOS(), makeCall(), stylusWrite()
9	Домашняя библиотека	MediaItem	название, год, жанр	Book, DVD, Magazine	read(), watch(), browse()
10	Спортивные снаряды	Equipment	название, вес, цена	Ball, Racket, Skates	bounce(), swing(), glide()
11	Музыкал	Instrument	название, тип, цена	Guitar, Piano, Drum	strum(), playKey(),

№ вар.	Тема	Базовый класс	Поля базового	Производные классы	Специфичные методы
	ьные инструменты				hit()
12	Мебель	Furniture	название, материал, цена	Chair, Table, Wardrobe	sit(), place(), store()
13	Одежда	Clothing	название, размер, материал	Shirt, Pants, Jacket	wear(), zip(), pocket()
14	Программное обеспечение	Software	название, версия, цена	OS, App, Game	boot(), run(), save()
15	Курсы обучения	Course	название, часы, сложность	Lecture, Lab, Exam	present(), practice(), test()

Средний уровень (15 вариантов)

Общее задание: Разработать иерархию классов с использованием виртуальных методов и абстрактного базового класса. Добавить статическое поле для подсчёта объектов. Реализовать метод клонирования (виртуальный метод clone()). Построить диаграмму UML. Создать фабрику объектов.

№ вар.	Тема	Особенности
1	Животные	Подсчёт объектов каждого вида; метод makeSound()
2	Фигуры	Вычисление площади и периметра; метод scale(factor)
3	Транспорт	Расчёт стоимости топлива; метод fuelCost(distance, price)
4	Банковские счета	Расчёт процентов; метод transfer(Account&, amount)
5	Работники	Расчёт зарплаты (разные системы оплаты); метод calculateBonus()
6	Игровые персонажи	Расчёт урона; метод attack(Character&)
7	Электроника	Расчёт энергопотребления; метод batteryLife()
8	Медиа	Расчёт времени просмотра/чтения; метод getDuration()
9	Спорт	Расчёт калорий; метод burnCalories(time)

№ вар.	Тема	Особенности
10	Еда	Расчёт калорийности; метод cook()
11	Здания	Расчёт площади и объёма; метод buildCost(pricePerSqM)
12	Часы	Расчёт времени до события; метод timeToEvent()
13	Алгоритмы	Реализация разных алгоритмов сортировки (паттерн Стратегия)
14	Шифрование	Реализация разных методов шифрования (паттерн Стратегия)
15	Сжатие	Реализация разных методов сжатия (паттерн Стратегия)

Продвинутый уровень (15 вариантов)

Общее задание: Разработать полноценную мини-систему с использованием наследования, полиморфизма и паттернов проектирования. Реализовать сериализацию (сохранение/загрузку из файла). Создать меню для демонстрации всех возможностей.

№ вар.	Тема	Требования
1	Система управления зоопарком	Учёт животных, кормление, ветеринарные карты
2	Система управления банковскими счетами	Разные типы счетов, транзакции, проценты
3	Система управления библиотекой	Книги, журналы, DVD, выдача/возврат
4	Система управления персоналом	Сотрудники, отделы, зарплаты, отчёты
5	Система управления отелем	Номера, бронирование, услуги, расчёт
6	Система управления курсами	Студенты, курсы, оценки, расписание
7	Система управления рестораном	Блюда, заказы, скидки, доставка
8	Система управления спортзалом	Абонементы, тренировки, клиенты, тренеры

№ вар.	Тема	Требования
9	Система управления кинотеатром	Фильмы, сеансы, залы, билеты
10	Система управления интернет-магазином	Товары, корзина, заказы, оплата
11	Система управления задачами	Задачи, проекты, исполнители, сроки
12	Система управления расписанием	События, напоминания, участники
13	Система управления складом	Товары, поставки, остатки, отгрузка
14	Система управления поликлиникой	Пациенты, врачи, записи, рецепты
15	Система управления автосервисом	Автомобили, услуги, заказ-наряды

Контрольные вопросы

1. Что такое наследование? Какие виды наследования есть в C++?
2. Что такое открытое (public) наследование? Какое отношение оно реализует?
3. Что такое виртуальный метод? Для чего он нужен?
4. Что такое чисто виртуальный метод и абстрактный класс?
5. Почему деструктор базового класса должен быть виртуальным?

6. Что такое позднее связывание (динамический полиморфизм)?
7. Как переопределить метод в производном классе? Что такое override?
8. Что такое RTTI? Как работает dynamic_cast?
9. Как вызвать метод базового класса из производного?
10. Что такое диаграмма классов UML? Какие основные элементы?
11. Что такое паттерн «Стратегия»? Где он применяется?
12. Что такое виртуальный деструктор? Почему он важен?

Требования к отчёту (ЛР23)

Титульный лист с названием работы, ФИО студента, номером группы, вариантом

Цель работы

Теоретическая часть (наследование, полиморфизм, UML)

Постановка задачи (согласно варианту)

Диаграмма классов UML (можно от руки или в инструменте)

Листинг программы с комментариями

Результаты работы (скриншоты вывода)

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Лабораторная работа №24

Реализация мини-проекта с использованием алгоритмов и структур данных

Цель работы:

1. Завершить разработку проекта, начатого в ЛР23
2. Реализовать методы, использующие изученные алгоритмы и структуры данных
3. Освоить сериализацию (сохранение и загрузку данных в файл)

4. Освоить механизм обработки исключительных ситуаций (try-catch-throw)
5. Создать пользовательский интерфейс (меню) для взаимодействия с системой
6. Защитить разработанный проект

Теоретическое обоснование

1. Сериализация объектов

Сериализация - это процесс преобразования состояния объекта в форму, пригодную для сохранения (в файл) или передачи (по сети). Обратный процесс называется десериализацией.

Зачем нужна сериализация?

Причина	Описание
Сохранение данных	Данные программы не теряются после завершения работы
Передача по сети	Объекты можно отправлять между программами
Межпрограммный обмен	Разные программы могут обмениваться данными через файлы

Способы сериализации в C++

Способ	**Описание**	**Сложность реализации**	**Читаемость файла**
Текстовый	Данные записываются в текстовом виде (CSV,	Средняя	Высокая

Способ	Описание	Сложность реализации	Читаемость файла
	JSON, XML)		
Бинарный	Данные записываются в бинарном виде (как в памяти)	Низкая	Нет
С использованием библиотек	Boost.Serialization, Protobuf, nlohmann/json	Высокая	Зависит от библиотеки

Пример текстовой сериализации

```
#include <fstream>
#include <string>
#include <vector>

class Book {
private:
    std::string title;
    std::string author;
    int year;

public:
    // Конструкторы
    Book() : title(""), author(""), year(0) {}
```

```
    Book(const std::string& t, const std::string& a, int y)
        : title(t), author(a), year(y) {}

    // Геттеры
    std::string getTitle() const { return title; }
    std::string getAuthor() const { return author; }
    int getYear() const { return year; }

    // Сериализация (сохранение)
    void saveToFile(std::ofstream& out) const {
        out << title << std::endl;
        out << author << std::endl;
        out << year << std::endl;
    }

    // Десериализация (загрузка)
    void loadFromFile(std::ifstream& in) {
        std::getline(in, title);
        std::getline(in, author);
        in >> year;
        in.ignore();  // пропускаем символ новой строки
    }
};
```

Пример бинарной сериализации

```
class BookBinary {
private:
    char title[100];
    char author[50];
```

```
    int year;

public:
    void saveToFile(std::ofstream& out) const {
        out.write((const char*)this, sizeof(BookBinary));
    }

    void loadFromFile(std::ifstream& in) {
        in.read((char*)this, sizeof(BookBinary));
    }
};
```

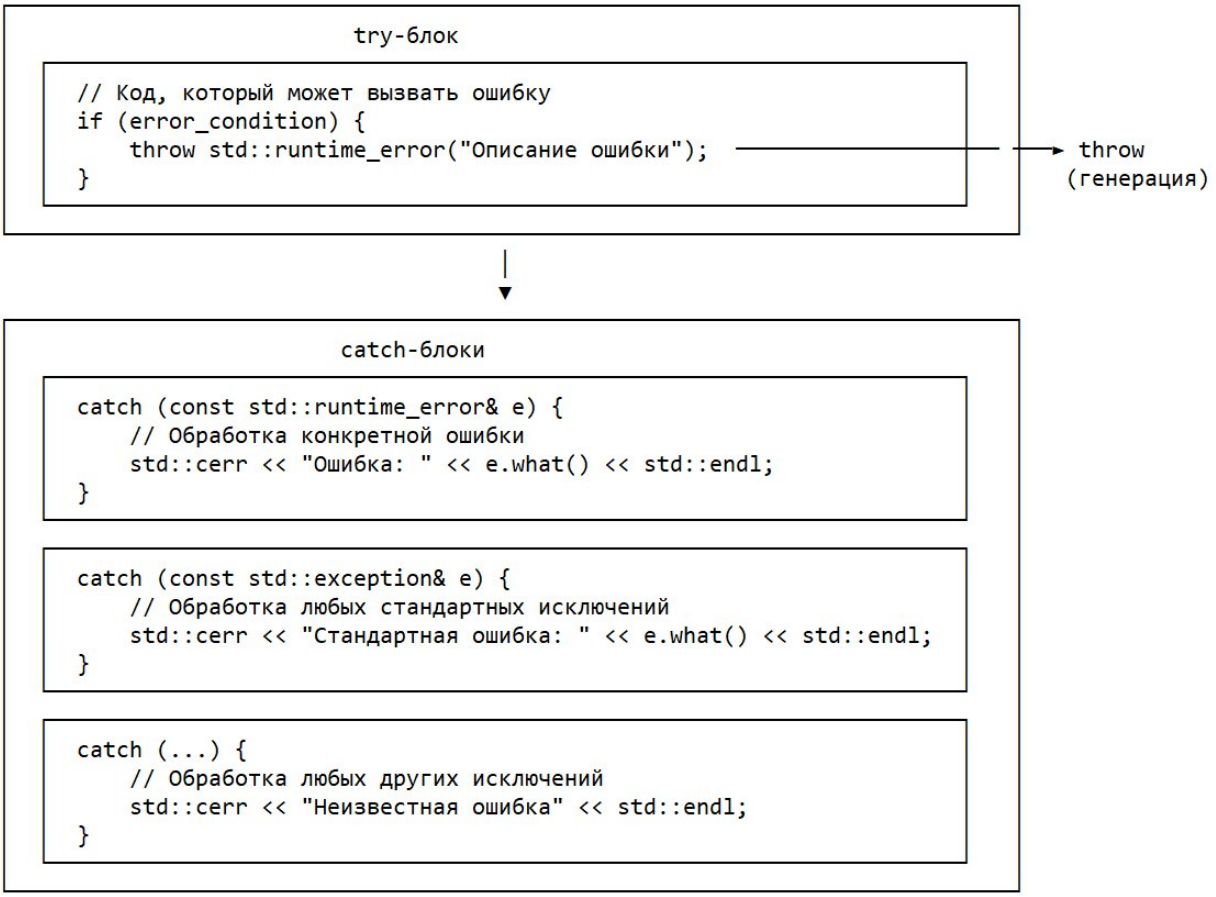
2. Обработка исключительных ситуаций

Исключение (exception) - это событие, которое нарушает нормальный ход выполнения программы. Механизм исключений позволяет отделить код обработки ошибок от основного кода программы.

Основные конструкции

Конструкция	Назначение
try	Блок кода, в котором могут возникнуть исключения
throw	Генерация исключения (передача управления обработчику)
catch	Блок обработки исключения определённого типа

Схема работы механизма исключений



Стандартные классы исключений

Класс исключения	Назначение	Заголовочный файл
std::exception	Базовый класс для всех исключений	<exception>
std::runtime_error	Ошибки времени выполнения	<stdexcept>
std::logic_error	Логические ошибки в программе	<stdexcept>
std::out_of_range	Выход за границы диапазона	<stdexcept>

Класс исключения	Назначение	Заголовочный файл
std::invalid_argument	Некорректный аргумент функции	<stdexcept>
std::bad_alloc	Ошибка выделения памяти	<new>

Примеры использования исключений

Пример 1: Проверка индекса

```
int getElement(const std::vector<int>& vec, int index) {
    if (index < 0 || index >= static_cast<int>(vec.size())) {
        throw std::out_of_range("Индекс " + std::to_string(index) +
                                " вне диапазона (0-" +
                                std::to_string(vec.size() - 1) + ")");
    }
    return vec[index];
}

// Использование
try {
    int value = getElement(myVector, 10);
    std::cout << "Значение: " << value << std::endl;
}
catch (const std::out_of_range& e) {
    std::cerr << "Ошибка: " << e.what() << std::endl;
}
```

Пример 2: Работа с файлами

```
void loadFromFile(const std::string& filename) {
    std::ifstream file(filename);
    if (!file.is_open()) {
        throw std::runtime_error("Не удалось открыть файл " + filename);
    }

    // Чтение данных...

    if (bad_format) {
        throw std::runtime_error("Некорректный формат данных в файле");
    }
}

// Использование
try {
    loadFromFile("data.txt");
    std::cout << "Данные загружены" << std::endl;
}
catch (const std::runtime_error& e) {
    std::cerr << "Ошибка загрузки: " << e.what() << std::endl;
    // Программа продолжает работу с пустой коллекцией
}
```

Пример 3: Пользовательский ввод

```
int getPositiveNumber() {
    int value;
    std::cin >> value;
```

```
        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(32767, '\n');
            throw std::invalid_argument("Введено не число");
        }

        if (value <= 0) {
            throw std::invalid_argument("Число должно быть положительным");
        }

        return value;
    }

    // Использование
    bool valid = false;
    while (!valid) {
        try {
            std::cout << "Введите положительное число: ";
            int num = getPositiveNumber();
            std::cout << "Вы ввели: " << num << std::endl;
            valid = true;
        }
        catch (const std::invalid_argument& e) {
            std::cerr << "Ошибка: " << e.what() << ". Попробуйте снова." <<
std::endl;
        }
    }
```

Создание собственных классов исключений

```cpp
#include <stdexcept>
#include <string>

// Собственный класс исключения для ошибок в библиотеке
class LibraryException : public std::runtime_error {
public:
    LibraryException(const std::string& msg) : std::runtime_error(msg) {}
};

// Специализированные исключения
class FileNotFoundException : public LibraryException {
public:
    FileNotFoundException(const std::string& filename)
        : LibraryException("Файл не найден: " + filename) {}
};

class InvalidDataFormatException : public LibraryException {
public:
    InvalidDataFormatException(const std::string& details)
        : LibraryException("Неверный формат данных: " + details) {}
};

// Использование
try {
    throw FileNotFoundException("data.txt");
}
catch (const LibraryException& e) {
    std::cerr << "Ошибка библиотеки: " << e.what() << std::endl;
}
```

```
catch (const std::exception& e) {
    std::cerr << "Общая ошибка: " << e.what() << std::endl;
}
```

Важные правила работы с исключениями

Правило	Объяснение
Деструкторы не должны выбрасывать исключения	Исключение в деструкторе может вызвать std::terminate()
Перехватывайте исключения по ссылке	catch (const std::exception& e) - избегает срезки объекта
Не используйте исключения для нормального потока управления	Исключения - для исключительных ситуаций
Всегда обрабатывайте возможные исключения	Неперехваченное исключение вызывает аварийное завершение

Требования к проекту

Функциональные требования

№	Требование	Описание
1	Иерархия классов	Минимум базовый абстрактный класс + 2 производных
2	Виртуальные	Полиморфное поведение

№	Требование	Описание
	методы	через указатель на базовый класс
3	Контейнер STL	Использование std::vector (или std::list, std::map)
4	Сортировка	Сортировка коллекции по заданному критерию
5	Поиск	Поиск элементов по различным критериям
6	Сериализация	Сохранение и загрузка данных в/из файла (текстового или бинарного)
7	Обработка исключений	Обработка ошибок с помощью try-catch-throw
8	Пользовательский интерфейс	Консольное меню с выбором действий

Обязательные ситуации для использования исключений

	Ситуация	Тип исключения	Обработка
	Ошибка открытия файла при загрузке	std::runtime_error	Сообщение, продолжение работы

	Ситуация	Тип исключения	Обработка
	Ошибка записи в файл	std::runtime_error	Сообщение, предложение другого имени
	Некорректный ввод данных (не число)	std::invalid_argument	Повтор запроса
	Некорректный ввод данных (диапазон)	std::out_of_range	Повтор запроса
	Попытка удалить несуществующий элемент	std::out_of_range	Сообщение пользователю
	Попытка доступа к пустой коллекции	std::logic_error	Сообщение пользователю

Пример выполнения практической части

Сквозной пример: Система управления библиотекой

Структура проекта

```
library_project/
├── include/
│   ├── MediaItem.h      # Базовый абстрактный класс
│   ├── Book.h           # Класс книга
│   ├── DVD.h            # Класс DVD
│   └── Library.h        # Класс библиотека (контейнер)
```

```
├── src/
│   ├── MediaItem.cpp
│   ├── Book.cpp
│   ├── DVD.cpp
│   ├── Library.cpp
│   └── main.cpp
├── data/
│   └── library.txt      # Файл для сохранения данных
└── README.md
```

Файл: MediaItem.h

```cpp
#ifndef MEDIA_ITEM_H
#define MEDIA_ITEM_H

#include <string>
#include <iostream>
#include <fstream>

// Базовый абстрактный класс
class MediaItem {
protected:
    std::string title;
    int year;
    std::string genre;

public:
    // Конструкторы
    MediaItem();
    MediaItem(const std::string& title, int year, const std::string& genre);
```

```
    virtual ~MediaItem() {}

    // Геттеры (не виртуальные)
    std::string getTitle() const;
    int getYear() const;
    std::string getGenre() const;

    // Сеттеры (с проверкой)
    void setTitle(const std::string& title);
    void setYear(int year);
    void setGenre(const std::string& genre);

    // Чисто виртуальные методы
    virtual double getDuration() const = 0;      // длительность (мин)
    virtual std::string getType() const = 0;     // тип объекта
    virtual void printInfo() const = 0;          // вывод информации

    // Виртуальный метод с реализацией по умолчанию
    virtual bool isLong() const {
        return getDuration() > 120;
    }

    // Сериализация
    virtual void saveToFile(std::ofstream& out) const = 0;
    virtual void loadFromFile(std::ifstream& in) = 0;
};

#endif
```

Файл: MediaItem.cpp

```cpp
#include "MediaItem.h"
#include <stdexcept>

MediaItem::MediaItem() : title(""), year(0), genre("") {}

MediaItem::MediaItem(const std::string& t, int y, const std::string& g)
    : title(t), year(y), genre(g) {}

std::string MediaItem::getTitle() const { return title; }
int MediaItem::getYear() const { return year; }
std::string MediaItem::getGenre() const { return genre; }

void MediaItem::setTitle(const std::string& t) {
    if (t.empty()) {
        throw std::invalid_argument("Название не может быть пустым");
    }
    title = t
}

void MediaItem::setYear(int y) {
    if (y < 0 || y > 2025) {
        throw std::out_of_range("Год должен быть от 0 до 2025");
    }
    year = y;
}

void MediaItem::setGenre(const std::string& g) {
    genre = g;
```

```
}
```

Файл: Book.h

```
#ifndef BOOK_H
#define BOOK_H

#include "MediaItem.h"

class Book : public MediaItem {
private:
    int pages;
    std::string author;

public:
    Book();
    Book(const std::string& title, int year, const std::string& genre,
         int pages, const std::string& author);

    // Геттеры
    int getPages() const;
    std::string getAuthor() const;

    // Переопределение виртуальных методов
    double getDuration() const override;
    std::string getType() const override;
    void printInfo() const override;

    // Сериализация
    void saveToFile(std::ofstream& out) const override;
```

```
    void loadFromFile(std::ifstream& in) override;

    // Специфичный метод
    void read() const;
};

#endif
```

Файл: Book.cpp

```
#include "Book.h"
#include <stdexcept>

Book::Book() : MediaItem(), pages(0), author("") {}

Book::Book(const std::string& title, int year, const std::string& genre,
           int pages, const std::string& author)
    : MediaItem(title, year, genre), pages(pages), author(author) {}

int Book::getPages() const { return pages; }
std::string Book::getAuthor() const { return author; }

double Book::getDuration() const {
    // Для книг "длительность" - время чтения (страницы / 2)
    return pages / 2.0;
}

std::string Book::getType() const {
    return "Книга";
}
```

```cpp
void Book::printInfo() const {
    std::cout << "=== Книга ===" << std::endl;
    std::cout << "Название: " << title << std::endl;
    std::cout << "Автор: " << author << std::endl;
    std::cout << "Год: " << year << std::endl;
    std::cout << "Жанр: " << genre << std::endl;
    std::cout << "Страниц: " << pages << std::endl;
    std::cout << "Время чтения: " << getDuration() << " мин" << std::endl;
}

void Book::saveToFile(std::ofstream& out) const {
    out << "Book" << std::endl;
    out << title << std::endl;
    out << year << std::endl;
    out << genre << std::endl;
    out << pages << std::endl;
    out << author << std::endl;
}

void Book::loadFromFile(std::ifstream& in) {
    std::getline(in, title);
    in >> year;
    in.ignore();
    std::getline(in, genre);
    in >> pages;
    in.ignore();
    std::getline(in, author);
}
```

```
void Book::read() const {
    std::cout << "Читаем книгу \"" << title << "\" (" << pages << " страниц)"
<< std::endl;
}
```

Файл: DVD.h

cpp

```
#ifndef DVD_H
#define DVD_H

#include "MediaItem.h"

class DVD : public MediaItem {
private:
    int duration;      // длительность в минутах
    std::string director;

public:
    DVD();
    DVD(const std::string& title, int year, const std::string& genre,
        int duration, const std::string& director);

    // Геттеры
    int getDurationMinutes() const;
    std::string getDirector() const;

    // Переопределение виртуальных методов
    double getDuration() const override;
    std::string getType() const override;
```

```
    void printInfo() const override;

    // Сериализация
    void saveToFile(std::ofstream& out) const override;
    void loadFromFile(std::ifstream& in) override;

    // Специфичный метод
    void watch() const;
};

#endif
```

Файл: DVD.cpp

```
#include "DVD.h"
#include <stdexcept>

DVD::DVD() : MediaItem(), duration(0), director("") {}

DVD::DVD(const std::string& title, int year, const std::string& genre,
         int duration, const std::string& director)
    : MediaItem(title, year, genre), duration(duration), director(director) {}

int DVD::getDurationMinutes() const { return duration; }
std::string DVD::getDirector() const { return director; }

double DVD::getDuration() const {
    return duration;
}
```

```
std::string DVD::getType() const {
    return "DVD";
}

void DVD::printInfo() const {
    std::cout << "=== DVD ===" << std::endl;
    std::cout << "Название: " << title << std::endl;
    std::cout << "Режиссёр: " << director << std::endl;
    std::cout << "Год: " << year << std::endl;
    std::cout << "Жанр: " << genre << std::endl;
    std::cout << "Длительность: " << duration << " мин" << std::endl;
}

void DVD::saveToFile(std::ofstream& out) const {
    out << "DVD" << std::endl;
    out << title << std::endl;
    out << year << std::endl;
    out << genre << std::endl;
    out << duration << std::endl;
    out << director << std::endl;
}

void DVD::loadFromFile(std::ifstream& in) {
    std::getline(in, title);
    in >> year;
    in.ignore();
    std::getline(in, genre);
    in >> duration;
    in.ignore();
```

```
    std::getline(in, director);
}

void DVD::watch() const {
    std::cout << "Смотрим DVD \"" << title << "\" (" << duration << "
минут)" << std::endl;
}
```

Файл: Library.h

```
#ifndef LIBRARY_H
#define LIBRARY_H

#include <vector>
#include <memory>
#include <string>
#include "MediaItem.h"

class Library {
private:
    std::vector<std::unique_ptr<MediaItem>> items;

public:
    // Добавление элементов
    void addBook();
    void addDVD();

    // Удаление элементов
```

```
    void removeItem(int index);
    void removeByTitle(const std::string& title);

    // Просмотр
    void showAll() const;
    void showLongItems() const;  // элементы длительностью > 120 мин

    // Сортировка
    void sortByYear();
    void sortByTitle();
    void sortByDuration();

    // Поиск
    void findByTitle(const std::string& title) const;
    void findByYearRange(int minYear, int maxYear) const;
    void findByGenre(const std::string& genre) const;

    // Сериализация
    void saveToFile(const std::string& filename) const;
    void loadFromFile(const std::string& filename);

    // Служебные методы
    int getSize() const;
    bool isEmpty() const;
    void clear();
};

#endif
```

Файл: Library.cpp

```cpp
#include "Library.h"
#include "Book.h"
#include "DVD.h"
#include <iostream>
#include <algorithm>
#include <fstream>
#include <stdexcept>
#include <memory>

// ---------------------------------------------------------------
// Добавление элементов
// ---------------------------------------------------------------

void Library::addBook() {
    std::string title, genre, author;
    int year, pages;

    try {
        std::cout << "Введите название книги: ";
        std::cin.ignore();
        std::getline(std::cin, title);

        std::cout << "Введите год издания: ";
        std::cin >> year;
        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(32767, '\n');
            throw std::invalid_argument("Год должен быть числом");
```

```
        }

        std::cout << "Введите жанр: ";
        std::cin.ignore();
        std::getline(std::cin, genre);

        std::cout << "Введите количество страниц: ";
        std::cin >> pages;
        if (std::cin.fail() || pages <= 0) {
            std::cin.clear();
            std::cin.ignore(32767, '\n');
            throw std::invalid_argument("Количество страниц должно быть положительным числом");
        }

        std::cout << "Введите автора: ";
        std::cin.ignore();
        std::getline(std::cin, author);

        items.push_back(std::make_unique<Book>(title, year, genre, pages, author));
        std::cout << "Книга добавлена!" << std::endl;
    }
    catch (const std::exception& e) {
        std::cerr << "Ошибка при добавлении книги: " << e.what() << std::endl;
    }
}
```

```
void Library::addDVD() {
    std::string title, genre, director;
    int year, duration;

    try {
        std::cout << "Введите название DVD: ";
        std::cin.ignore();
        std::getline(std::cin, title);

        std::cout << "Введите год выпуска: ";
        std::cin >> year;
        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(32767, '\n');
            throw std::invalid_argument("Год должен быть числом");
        }

        std::cout << "Введите жанр: ";
        std::cin.ignore();
        std::getline(std::cin, genre);

        std::cout << "Введите длительность (минуты): ";
        std::cin >> duration;
        if (std::cin.fail() || duration <= 0) {
            std::cin.clear();
            std::cin.ignore(32767, '\n');
            throw std::invalid_argument("Длительность должна быть положительным числом");
        }
```

```
            std::cout << "Введите режиссёра: ";
            std::cin.ignore();
            std::getline(std::cin, director);

            items.push_back(std::make_unique<DVD>(title, year, genre, duration,
director));
            std::cout << "DVD добавлен!" << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Ошибка при добавлении DVD: " << e.what() <<
std::endl;
        }
    }

    // ------------------------------------------------------------------
    // Удаление элементов
    // ------------------------------------------------------------------

    void Library::removeItem(int index) {
        try {
            if (index < 0 || index >= static_cast<int>(items.size())) {
                throw std::out_of_range("Индекс " + std::to_string(index) +
                                        " вне диапазона (0-" +
                                        std::to_string(items.size() - 1) + ")");
            }
            items.erase(items.begin() + index);
            std::cout << "Элемент удалён" << std::endl;
        }
```

```
    catch (const std::out_of_range& e) {
        std::cerr << "Ошибка удаления: " << e.what() << std::endl;
    }
}

void Library::removeByTitle(const std::string& title) {
    auto it = std::find_if(items.begin(), items.end(),
        [&title](const std::unique_ptr<MediaItem>& item) {
            return item->getTitle() == title;
        });

    if (it != items.end()) {
        items.erase(it);
        std::cout << "Элемент \"" << title << "\" удалён" << std::endl;
    } else {
        std::cout << "Элемент \"" << title << "\" не найден" << std::endl;
    }
}

// ------------------------------------------------------------
// Просмотр
// ------------------------------------------------------------

void Library::showAll() const {
    if (items.empty()) {
        std::cout << "Библиотека пуста" << std::endl;
        return;
    }
```

```
        std::cout << "\n=== Содержимое библиотеки ===" << std::endl;
        for (size_t i = 0; i < items.size(); ++i) {
            std::cout << i + 1 << ". ";
            items[i]->printInfo();
            std::cout << std::endl;
        }
    }

    void Library::showLongItems() const {
        if (items.empty()) {
            std::cout << "Библиотека пуста" << std::endl;
            return;
        }

        std::cout << "\n=== Длинные произведения (> 120 мин) ===" <<
std::endl;
        bool found = false;
        for (const auto& item : items) {
            if (item->isLong()) {
                item->printInfo();
                std::cout << std::endl;
                found = true;
            }
        }
        if (!found) {
            std::cout << "Нет произведений длительностью более 120 минут"
<< std::endl;
        }
    }
```

```
// ---------------------------------------------------------------
// Сортировка
// ---------------------------------------------------------------

void Library::sortByYear() {
    std::sort(items.begin(), items.end(),
        [](const std::unique_ptr<MediaItem>& a, const
std::unique_ptr<MediaItem>& b) {
            return a->getYear() < b->getYear();
        });
    std::cout << "Коллекция отсортирована по году" << std::endl;
    showAll();
}

void Library::sortByTitle() {
    std::sort(items.begin(), items.end(),
        [](const std::unique_ptr<MediaItem>& a, const
std::unique_ptr<MediaItem>& b) {
            return a->getTitle() < b->getTitle();
        });
    std::cout << "Коллекция отсортирована по названию" << std::endl;
    showAll();
}

void Library::sortByDuration() {
    std::sort(items.begin(), items.end(),
        [](const std::unique_ptr<MediaItem>& a, const
std::unique_ptr<MediaItem>& b) {
```

```
            return a->getDuration() < b->getDuration();
        });
    std::cout << "Коллекция отсортирована по длительности" << std::endl;
    showAll();
}

// ----------------------------------------------------------------
// Поиск
// ----------------------------------------------------------------

void Library::findByTitle(const std::string& title) const {
    auto it = std::find_if(items.begin(), items.end(),
        [&title](const std::unique_ptr<MediaItem>& item) {
            return item->getTitle() == title;
        });

    if (it != items.end()) {
        std::cout << "Найдено:" << std::endl;
        (*it)->printInfo();
    } else {
        std::cout << "Элемент \"" << title << "\" не найден" << std::endl;
    }
}

void Library::findByYearRange(int minYear, int maxYear) const {
    try {
        if (minYear > maxYear) {
            throw std::invalid_argument("Минимальный год не может быть
больше максимального");
```

```
            }

            bool found = false;
            for (const auto& item : items) {
                int year = item->getYear();
                if (year >= minYear && year <= maxYear) {
                    item->printInfo();
                    std::cout << std::endl;
                    found = true;
                }
            }

            if (!found) {
                std::cout << "Нет элементов в диапазоне годов " << minYear << "-
" << maxYear << std::endl;
            }
        }
        catch (const std::invalid_argument& e) {
            std::cerr << "Ошибка поиска: " << e.what() << std::endl;
        }
    }

    void Library::findByGenre(const std::string& genre) const {
        bool found = false;
        for (const auto& item : items) {
            if (item->getGenre() == genre) {
                item->printInfo();
                std::cout << std::endl;
                found = true;
```

```
            }
        }

        if (!found) {
            std::cout << "Нет элементов в жанре \"" << genre << "\"" <<
std::endl;
        }
    }

    // ---------------------------------------------------------------
    // Сериализация
    // ---------------------------------------------------------------

    void Library::saveToFile(const std::string& filename) const {
        try {
            std::ofstream out(filename);
            if (!out.is_open()) {
                throw std::runtime_error("Не удалось открыть файл " + filename +
" для записи");
            }

            out << items.size() << std::endl;

            for (const auto& item : items) {
                item->saveToFile(out);
            }

            std::cout << "Данные сохранены в файл " << filename << std::endl;
        }
```

```
    catch (const std::runtime_error& e) {
        std::cerr << "Ошибка сохранения: " << e.what() << std::endl;
    }
}

void Library::loadFromFile(const std::string& filename) {
    try {
        std::ifstream in(filename);
        if (!in.is_open()) {
            throw std::runtime_error("Не удалось открыть файл " + filename +
" для чтения");
        }

        clear();

        int size;
        in >> size;
        in.ignore();

        for (int i = 0; i < size; ++i) {
            std::string type;
            std::getline(in, type);

            if (type == "Book") {
                auto book = std::make_unique<Book>();
                book->loadFromFile(in);
                items.push_back(std::move(book));
            } else if (type == "DVD") {
                auto dvd = std::make_unique<DVD>();
```

```
                dvd->loadFromFile(in);
                items.push_back(std::move(dvd));
            } else {
                throw std::runtime_error("Неизвестный тип объекта: " + type);
            }
        }

        std::cout << "Данные загружены из файла " << filename << std::endl;
        std::cout << "Загружено элементов: " << items.size() << std::endl;
    }
    catch (const std::runtime_error& e) {
        std::cerr << "Ошибка загрузки: " << e.what() << std::endl;
        std::cerr << "Программа продолжит работу с пустой коллекцией" << std::endl;
    }
}

// ------------------------------------------------------------
// Служебные методы
// ------------------------------------------------------------

int Library::getSize() const {
    return items.size();
}

bool Library::isEmpty() const {
    return items.empty();
}
```

```
void Library::clear() {
    items.clear();
}
```

Файл: main.cpp

```
#include <iostream>
#include <limits>
#include "Library.h"

using namespace std;

void showMenu() {
    cout << "\n========================================" <<
endl;
    cout << "   СИСТЕМА УПРАВЛЕНИЯ БИБЛИОТЕКОЙ" << endl;
    cout << "========================================" <<
endl;
    cout << "1. Добавить книгу" << endl;
    cout << "2. Добавить DVD" << endl;
    cout << "3. Показать все" << endl;
    cout << "4. Показать длинные произведения (>120 мин)" << endl;
    cout << "5. Удалить элемент по индексу" << endl;
    cout << "6. Удалить элемент по названию" << endl;
    cout << "7. Сортировать по году" << endl;
    cout << "8. Сортировать по названию" << endl;
    cout << "9. Сортировать по длительности" << endl;
    cout << "10. Найти по названию" << endl;
    cout << "11. Найти по диапазону годов" << endl;
    cout << "12. Найти по жанру" << endl;
```

```
    cout << "13. Сохранить в файл" << endl;
    cout << "14. Загрузить из файла" << endl;
    cout << "15. Очистить библиотеку" << endl;
    cout << "0. Выход" << endl;
    cout << "-------------------------------------" << endl;
    cout << "Выберите действие: ";
}

int getChoice() {
    int choice;
    cin >> choice;

    if (cin.fail()) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        return -1;
    }

    return choice;
}

int getIndex(int maxSize) {
    int index;
    cout << "Введите индекс (0-" << maxSize - 1 << "): ";
    cin >> index;

    if (cin.fail()) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
```

```
            throw invalid_argument("Индекс должен быть числом");
        }

        return index;
    }

    int main() {
        setlocale(LC_ALL, "ru");

        Library lib;
        int choice;

        cout << "Добро пожаловать в систему управления библиотекой!" <<
endl;

        do {
            showMenu();
            choice = getChoice();

            switch (choice) {
                case 1:
                    lib.addBook();
                    break;

                case 2:
                    lib.addDVD();
                    break;

                case 3:
```

```
            lib.showAll();
            break;

        case 4:
            lib.showLongItems();
            break;

        case 5:
            if (lib.isEmpty()) {
                cout << "Библиотека пуста, нечего удалять" << endl;
            } else {
                try {
                    int index = getIndex(lib.getSize());
                    lib.removeItem(index);
                }
                catch (const exception& e) {
                    cerr << "Ошибка: " << e.what() << endl;
                }
            }
            break;

        case 6: {
            if (lib.isEmpty()) {
                cout << "Библиотека пуста, нечего удалять" << endl;
            } else {
                string title;
                cout << "Введите название: ";
                cin.ignore();
                getline(cin, title);
```

```
            lib.removeByTitle(title);
        }
        break;
    }

    case 7:
        if (lib.isEmpty()) {
            cout << "Библиотека пуста, нечего сортировать" << endl;
        } else {
            lib.sortByYear();
        }
        break;

    case 8:
        if (lib.isEmpty()) {
            cout << "Библиотека пуста, нечего сортировать" << endl;
        } else {
            lib.sortByTitle();
        }
        break;

    case 9:
        if (lib.isEmpty()) {
            cout << "Библиотека пуста, нечего сортировать" << endl;
        } else {
            lib.sortByDuration();
        }
        break;
```

```
case 10: {
    string title;
    cout << "Введите название: ";
    cin.ignore();
    getline(cin, title);
    lib.findByTitle(title);
    break;
}

case 11: {
    int minYear, maxYear;
    cout << "Введите минимальный год: ";
    cin >> minYear;
    cout << "Введите максимальный год: ";
    cin >> maxYear;
    lib.findByYearRange(minYear, maxYear);
    break;
}

case 12: {
    string genre;
    cout << "Введите жанр: ";
    cin.ignore();
    getline(cin, genre);
    lib.findByGenre(genre);
    break;
}

case 13:
```

```
            lib.saveToFile("library.txt");
            break;

        case 14:
            lib.loadFromFile("library.txt");
            break;

        case 15:
            lib.clear();
            cout << "Библиотека очищена" << endl;
            break;

        case 0:
            cout << "До свидания!" << endl;
            break;

        default:
            cout << "Неверный выбор! Попробуйте снова." << endl;
            break;
    }
} while (choice != 0);

return 0;
}
```

Индивидуальные задания

Продвинутый уровень (15 вариантов) - дополнение

Общее задание: Разработать полноценную мини-систему с использованием наследования, полиморфизма, алгоритмов STL, сериализации и обработки исключений.

Обязательные требования:

№	Требование	Описание
1	Иерархия классов	Базовый абстрактный класс + минимум 2 производных
2	Виртуальные методы	Переопределение методов в производных классах
3	Контейнер STL	Хранение объектов в std::vector или std::map
4	Сортировка	Сортировка коллекции по разным критериям
5	Поиск	Поиск по разным критериям
6	Сериализация	Сохранение/загрузка в текстовый файл

№	Требование	Описание
7	Исключения	Обработка ошибок с помощью try-catch-throw
8	Меню	Консольное меню с выбором действий

№ вар.	Тема	Типы объектов	Критерии сортировки	Критерии поиска
1	Библиотека	Книга, DVD, Журнал	год, название, длительность	название, жанр, год
2	Сотрудники	Менеджер, Разработчик, Стажёр	зарплата, стаж, имя	имя, отдел, опыт
3	Транспорт	Автомобиль, Грузовик, Мотоцикл	цена, год, мощность	марка, тип, цена
4	Животные	Собака, Кошка, Птица	возраст, вес, имя	имя, вид, возраст
5	Фигуры	Круг, Прямоугольник, Треугольник	площадь, периметр	тип, площадь

№ вар.	Тема	Типы объектов	Критерии сортировки	Критерии поиска
6	Банковские счета	Сберегательный, Расчётный, Кредитный	баланс, процент, номер	номер, владелец
7	Студенты	Бакалавр, Магистр, Аспирант	средний балл, курс, имя	имя, курс, факультет
8	Игровые персонажи	Воин, Маг, Лучник	уровень, здоровье, сила	имя, класс, уровень
9	Электроника	Ноутбук, Смартфон, Планшет	цена, производительность	бренд, цена, модель
10	Музыка	Песня, Альбом, Исполнитель	длительность, год, рейтинг	название, исполнитель
11	Спорт	Футбол, Баскетбол, Теннис	количество игроков, популярность	название, вид
12	Еда	Фрукт, Овощ, Мясо	калории, цена, вес	название, тип

№ вар.	Тема	Типы объектов	Критерии сортировки	Критерии поиска
13	Фильмы	Боевик, Комедия, Драма	рейтинг, год, длительность	название, режиссёр
14	Квартиры	Однокомнатная, Двухкомнатная, Трёхкомнатная	цена, площадь, этаж	адрес, цена, комнат
15	Задачи	Простая, Срочная, Долгосрочная	приоритет, срок, сложность	название, исполнитель

Контрольные вопросы

1. Что такое сериализация? Для чего она нужна?
2. Какие способы сериализации в C++ вы знаете?
3. В чём преимущества и недостатки текстовой сериализации?
4. В чём преимущества и недостатки бинарной сериализации?
5. Что такое исключение? Для чего нужен механизм исключений?
6. Какие конструкции используются для работы с исключениями (try, catch, throw)?
7. В чём разница между catch (const std::exception& e) и catch (...)?
8. Какие стандартные классы исключений вы знаете?
9. Как создать собственный класс исключения?
10. Почему деструкторы не должны выбрасывать исключения?
11. Как обрабатывать ошибки ввода данных пользователем с помощью исключений?
12. Как обрабатывать ошибки при работе с файлами с помощью исключений?
13. Что такое RAII? Как он связан с исключениями?
14. Как обрабатывать ошибки выделения памяти (std::bad_alloc)?

15.В каких ситуациях в вашем проекте используются исключения?

Требования к отчёту

Титульный лист

Цель работы

Теоретическая часть (сериализация, исключения)

Постановка задачи (согласно варианту)

Диаграмма классов UML

Листинг программы с комментариями

Результаты работы (скриншоты вывода)

Анализ использования исключений (в каких ситуациях и какие исключения используются)

Ответы на контрольные вопросы

Выводы

Ссылка на Git-репозиторий

Критерии оценивания

Максимальный балл: 100

№	Критерий	Баллы
1	Иерархия классов (из ЛР23)	20
1.1	Базовый абстрактный класс с чисто виртуальными методами	10
1.2	Производные классы (минимум 2), переопределение методов	10
2	Реализация функциональности	25

№	Критерий	Баллы
2.1	Использование контейнеров STL (vector, list, map)	10
2.2	Реализация сортировки элементов коллекции	8
2.3	Реализация поиска элементов по критериям	7
3	Сериализация	15
3.1	Сохранение данных в файл	8
3.2	Загрузка данных из файла	7
4	Обработка исключений	15
4.1	Обработка ошибок файлового ввода/вывода	5
4.2	Обработка ошибок ввода данных пользователем	5
4.3	Обработка ошибок доступа к коллекции (выход за границы, удаление из пустого)	5
5	Пользовательский интерфейс	10
5.1	Меню с выбором действий	4

№	Критерий	Баллы
5.2	Обработка ошибок ввода (нечисловые символы и т.д.)	3
5.3	Информативные сообщения об ошибках	3
6	Защита проекта	15
6.1	Презентация проекта (5-7 минут)	4
6.2	Демонстрация работы программы	4
6.3	Ответы на вопросы по проекту	4
6.4	Качество кода и комментарии	3
Итого		100

Штрафные баллы

Нарушение	Штраф
Отсутствие виртуального деструктора в базовом классе	-15
Утечка памяти (не освобождена динамическая память)	-15
Нет обработки ошибок при работе с файлами (исключениями или проверками)	-10

Нарушение	Штраф
Нет меню или оно неработоспособно	-10
Исключения не используются там, где они необходимы (см. список обязательных ситуаций)	-10
Программа не компилируется	работа не принимается

Итоговая шкала

Оценка	Баллы
Отлично	90–100
Хорошо	75–89
Удовлетворительно	60–74
Неудовлетворительно	менее 60

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по организации и проведению самостоятельной работы по дисциплине
«Алгоритмы и структуры данных»
для студентов направления подготовки
09.03.04 Программная инженерия Направленность (профиль)
«Разработка и сопровождение программного обеспечения»

Ставрополь 2026

ВВЕДЕНИЕ

Общие положения

Настоящие методические указания предназначены для организации самостоятельной работы студентов по дисциплине «Алгоритмы и структуры данных». Дисциплина изучается в течение двух семестров и включает как теоретическую подготовку (лекции), так и практическую (лабораторные работы).

Трудоёмкость дисциплины:

Параметр	Значение
Общая трудоёмкость	5 зачётных единиц (180 часов)
Аудиторная работа	96 часов
Самостоятельная работа	57 часов
Промежуточная аттестация	Экзамен (2 семестр)

Цель и задачи самостоятельной работы

Цель самостоятельной работы - формирование у студентов способности к самостоятельному освоению теоретического материала, выполнению практических заданий и разработке программного проекта с использованием алгоритмов и структур данных.

Задачи самостоятельной работы:

№	Задача	Формируемые компетенции
1	Изучение теоретического материала по конспектам лекций и учебной литературе	ОПК-1, ОПК-7
2	Выполнение лабораторных работ согласно индивидуальным вариантам	ОПК-6, ПК-3
3	Разработка итогового проекта (ЛР23-24)	ПК-2, ПК-3
4	Ведение Git-репозитория с кодом всех лабораторных работ	ОПК-6, ПК-3
5	Подготовка к собеседованиям и экзамену	ОПК-1, ОПК-7

Планируемые результаты обучения
Студент должен знать:

1. Основные конструкции языка C++ (ОПК-6)
2. Принципы объектно-ориентированного программирования (ОПК-7)
3. Классические алгоритмы и структуры данных (ОПК-1)
4. Возможности стандартной библиотеки шаблонов (ПК-3)
5. Студент должен уметь:
6. Разрабатывать алгоритмы решения задач (ОПК-6)
7. Реализовывать структуры данных на C++ (ПК-3)
8. Проектировать иерархии классов (ПК-2)
9. Анализировать сложность алгоритмов (ОПК-1)
10. Студент должен владеть:
11. Навыками программирования на C++ (ПК-3)
12. Методами отладки и тестирования программ (ОПК-6)
13. Инструментарием разработки (IDE, Git) (ПК-3)

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1.1 Виды самостоятельной работы студентов

№	Вид самостоятельной работы	Трудоёмкость (часов)
1	Изучение лекционного материала (конспектирование)	36
2	Подготовка к лабораторным работам	72
3	Выполнение лабораторных работ (кодирование, отладка)	54
4	Разработка итогового проекта (ЛР23-24)	36
5	Подготовка к собеседованиям и экзамену	18

№	Вид самостоятельной работы	Трудоёмкость (часов)
Итого		216

1.2 Объём самостоятельной работы

Вид работы	Часов
Изучение лекций 1-19	10
Выполнение ЛР	10
Работа над проектом	27
Подготовка к экзамену	10
Итого	57

1.3 Календарный план выполнения самостоятельной работы

Модуль 1. Основы программирования на C++ (недели 1–4)

Неделя	Тема лекции	Лабораторная работа	Проект	Рекомендации по самостоятельной работе
1	Л1. Потоки. Текстовые файлы	ЛР1. Текстовые файлы	—	Изучить работу с ifstream/ofstream. Разобрать примеры из лекции. Выполнить задания базового уровня. Оформить отчёт, загрузить в Git
2	Л2. Бинарные файлы. Позиционирование	ЛР2. Бинарные файлы	—	Освоить методы read/write, seekg/seekp. Выполнить задания среднего уровня. Подготовиться к собеседованию по темам 1-2

Неделя	Тема лекции	Лабораторная работа	Проект	Рекомендации по самостоятельной работе
3	Л3. Основы ООП. Классы и структуры	ЛР3. Классы. Геттеры/сеттеры	—	Понять инкапсуляцию. Создать простые классы. Реализовать геттеры и сеттеры с проверкой. Выполнить задания по варианту
4	Л4. Конструкторы. Деструктор. Правило трёх	ЛР4. IntVector	—	Разобрать правило трёх. Реализовать динамический массив. Освоить список инициализации. Проверить отсутствие утечек памяти
Итоги модуля 1	Подготовка к собеседованию №1	Сдача ЛР1-4	—	Повторить темы 1-4. Ответить на контрольные вопросы. Сдать отчёты в Git

Модуль 2. Объектно-ориентированное программирование (недели 5–8)

Неделя	Тема лекции	Лабораторная работа	Проект	Рекомендации по самостоятельной работе
5	Л5. Перегрузка операторов. Дружественные функции	ЛР5. Перегрузка операторов	—	Изучить перегрузку +, -, *, /, ==, <<, >>. Выполнить задания повышенного уровня
6	Л6. Композиция и агрегация классов	ЛР6. Отношения между классами	—	Понять композицию vs агрегацию. Реализовать классы House и Room, Library и Book. Добавить статические члены
7	Л7. Анализ сложности. О-нотация	ЛР7. Сравнение производительности	—	Освоить Big O. Написать программу для сравнения линейного и бинарного

Неделя	Тема лекции	Лабораторная работа	Проект	Рекомендации по самостоятельной работе
				поиска. Построить таблицу времени выполнения
8	Л8. Шаблоны классов	ЛР8. Шаблонный контейнер	Выбор темы проекта	Изучить шаблоны. Создать шаблонный класс Pair или Stack. Определиться с темой итогового проекта
Итоги модуля 2	Подготовка к собеседованию №2	Сдача ЛР5-8	Тема утверждена	Повторить темы 5-8. Согласовать тему проекта с преподавателем. Начать проектирование UML

Модуль 3. Структуры данных (недели 9–12)

Неделя	Тема лекции	Лабораторная работа	Проект	Рекомендации по самостоятельной работе
9	Л9. Динамический массив (аналог vector)	ЛР9. Реализация vector	Проектирование	Реализовать свой Vector<T> с методами push_back, pop_back, insert, erase. Освоить амортизированный анализ
10	Л10. Односвязный и двусвязный списки	ЛР10. Реализация списков	—	Реализовать SinglyLinkedList и DoublyLinkedList. Отработать вставку, удаление, поиск
11	Л11. Стек и очередь	ЛР11. Стек и очередь	—	Реализовать стек и очередь на массиве и списке. Решить задачи: проверка скобок, очередь на двух стеках

Неделя	Тема лекции	Лабораторная работа	Проект	Рекомендации по самостоятельной работе
12	Л12. Дек и приоритетная очередь	ЛР12. Дек, priority_queue	Начало реализации	Реализовать дек. Реализовать приоритетную очередь на куче. Разобрать heapify
Итоги модуля 3	Подготовка к собеседованию №3	Сдача ЛР9-12	Проект: базовые классы	Повторить темы 9-12. Создать базовый абстрактный класс проекта. Нарисовать UML-диаграмму

Модуль 4. Алгоритмы и завершение семестра (недели 13–18)

Неделя	Тема лекции	Лабораторная работа	Проект	Рекомендации по самостоятельной работе
13	Л13. Рекурсия	ЛР13. Рекурсивные алгоритмы	—	Реализовать рекурсивные функции: факториал, Фибоначчи, Ханойские башни. Сравнить с итеративными версиями
14	Л14. Простые сортировки	ЛР14. Пузырёк, вставки, выбор	—	Реализовать три простые сортировки. Сравнить их производительность. Проанализировать сложность
15	Л15. Эффективные сортировки	ЛР15. Быстрая, слиянием	—	Реализовать Quick Sort и Merge Sort. Сравнить с простыми сортировками.

Недел я	Тема лекции	Лабораторна я работа	Проект	Рекомендации по самостоятельной работе
				Проанализировать O(n log n)
16	Л16. Пирамидальна я сортировка	ЛР16. Heap Sort	—	Реализовать пирамидальную сортировку. Сравнить с Quick Sort и Merge Sort
17	Л17. Двоичный поиск. Хеш-таблицы	ЛР17. Хеш-таблицы	Проект: продолжение	Реализовать хеш-таблицу с методом цепочек. Сравнить поиск в хеш-таблице и BST
18	Повторение и зачёт	Защита ЛР13-17	Сдача ЛР23 (проектирование)	Завершить проектирование проекта. Сдать UML-диаграмму и описание классов. Подготовиться к зачёту
Итоги модул я 4	Подготовка к зачёту	Сдача ЛР13-17, ЛР23	Проект: сдан	Повторить весь материал семестра. Защитить ЛР23

Сводная таблица по неделям

Неделя	Лекции	Лабораторные работы	Проект	Отчётность
1	Л1	ЛР1	—	Git-репозиторий
2	Л2	ЛР2	—	—
3	Л3	ЛР3	—	—

Неделя	Лекции	Лабораторные работы	Проект	Отчётность
4	Л4	ЛР4	—	Сдача ЛР1-4
5	Л5	ЛР5	—	—
6	Л6	ЛР6	—	Собеседование №1
7	Л7	ЛР7	—	—
8	Л8	ЛР8	Выбор темы	Сдача ЛР5-8
9	Л9	ЛР9	Проектирование	—
10	Л10	ЛР10	—	Собеседование №2
11	Л11	ЛР11	—	—
12	Л12	ЛР12	Начало реализации	Сдача ЛР9-12
13	Л13	ЛР13	—	—
14	Л14	ЛР14	—	—
15	Л15	ЛР15	—	Собеседование №3
16	Л16	ЛР16	—	—
17	Л17	ЛР17	Продолжение	—
18	—	Защита	Сдача ЛР23	Сдача ЛР13-17, ЛР23

Контрольные точки семестра

№	Неделя	Событие	Форма отчётности
1	4	Сдача ЛР1-4	Git-репозиторий, отчёты
2	6	Собеседование №1	Устный опрос по темам 1-6
3	8	Сдача ЛР5-8	Git-репозиторий, отчёты
4	8	Выбор темы проекта	Согласование с преподавателем
5	10	Собеседование №2	Устный опрос по темам 7-10
6	12	Сдача ЛР9-12	Git-репозиторий, отчёты
7	12	UML-диаграмма проекта	Проверка архитектуры
8	15	Собеседование №3	Устный опрос по темам 11-15
9	18	Сдача ЛР13-17	Git-репозиторий, отчёты
10	18	Сдача ЛР23	Отчёт, UML, описание классов
11	18	Зачёт	Комплексная оценка

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

2.1 Работа с учебной литературой

Основная литература:

Павловская Т.А., Щупак Ю.А. «C/C++. Структурное и объектно-ориентированное программирование: Практикум». - СПб.: Питер, 2011.

Лафоре Р. «Объектно-ориентированное программирование в C++». - СПб.: Питер, 2011.

Страуструп Б. «Язык программирования C++». - М.: Бином, 2008.

Рекомендации по работе с книгой Павловской и Щупака:

Раздел книги	Темы лекций	Рекомендации
Семинар 1-2	Л1-2 (файлы)	Изучить примеры 1.1-1.3, выполнить задания
Семинар 3-4	Л8-9 (списки)	Изучить задачу 9.2 (линейный список)
Семинар 10	Л3-4 (классы)	Изучить задачу 10.1 и 10.2 (треугольники)
Семинар 11	Л16 (наследование)	Изучить задачу 11.1-11.3
Семинар 15	Л22 (STL)	Изучить примеры 15.1-15.3

2.2 Конспектирование лекций

Рекомендуемая структура конспекта каждой лекции:

Тема и дата лекции

Основные определения (выделять маркером)

Ключевые примеры кода с комментариями

Анализ сложности рассматриваемых алгоритмов

Связь с лабораторными работами (какие ЛР опираются на этот материал)

Вопросы для самопроверки (3-5 вопросов по теме)

2.3 Подготовка к собеседованиям

Собеседования проводятся 2 раза в семестр (на 9-й и 18-й неделях).

Порядок подготовки:

Повторить конспекты лекций за пройденный период

Выполнить все лабораторные работы (код должен быть в Git)

Подготовить устные ответы на контрольные вопросы к каждой ЛР

Быть готовым продемонстрировать работу любой из выполненных программ

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

3.1 Общие требования к выполнению лабораторных работ

Структура отчёта по лабораторной работе:

Титульный лист

Цель работы

Теоретическая часть (основные понятия)

Постановка задачи (согласно варианту)

Листинг программы с комментариями
Результаты работы (скриншоты)
Ответы на контрольные вопросы
Выводы
Ссылка на Git-репозиторий

Требования к Git-репозиторию:

```
student_name/
├── README.md                # описание репозитория
├── lab01/                   # текстовые файлы
│   ├── task1.cpp
│   ├── task2.cpp
│   └── report.md
├── lab02/                   # бинарные файлы
│   ├── task1.cpp
│   └── report.md
├── ...
├── lab22/                   # КМП и STL
│   ├── main.cpp
│   └── report.md
├── project/                 # итоговый проект (ЛР23-24)
│   ├── include/
│   ├── src/
│   ├── CMakeLists.txt
│   └── README.md
└── .gitignore
```

3.2 Поэтапный план работы над итоговым проектом (ЛР23-24)
Неделя 8. Выбор темы и проектирование
Задачи:

1. Выбрать тему проекта из списка вариантов ЛР23
2. Определить предметную область и основные сущности
3. Сформулировать требования к системе
4. Результат: Тема проекта, согласованная с преподавателем.

Методические рекомендации:
Не выбирайте слишком узкую тему - будет мало материала для реализации
Не выбирайте слишком широкую - не успеете реализовать
Оптимальный объём: 3-5 классов, 10-15 методов

Неделя 9. Анализ и проектирование

Задачи:

1. Выявить все необходимые классы
2. Определить поля и методы каждого класса
3. Определить отношения между классами (наследование, композиция, агрегация)
4. Построить диаграмму классов UML
5. Результат: Диаграмма классов UML (можно от руки в тетради или в инструменте).

Методические рекомендации:

Используйте обозначения UML:

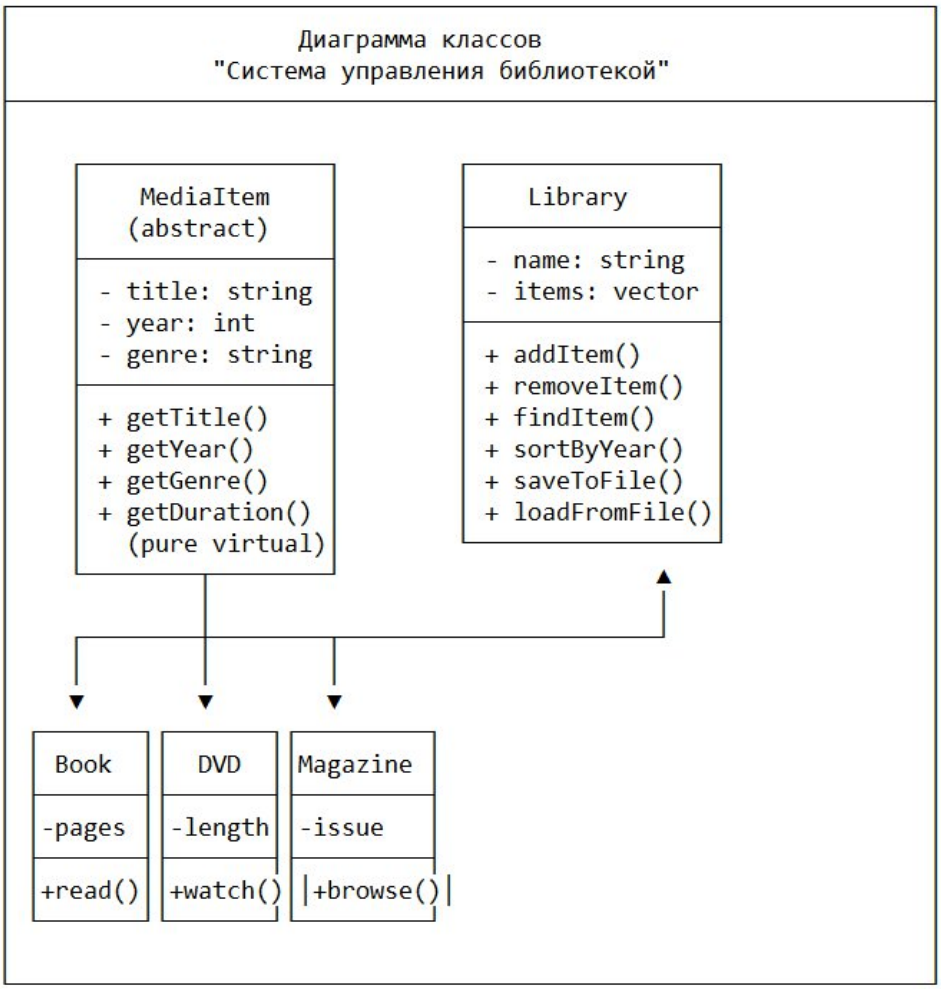
▷ - наследование (стрелка от потомка к базовому)

◇ - агрегация (ромб у целого)

◆ - композиция (закрашенный ромб)

Для каждого метода укажите: имя, тип возвращаемого значения, параметры

Пример UML-диаграммы (от руки):



Неделя 10-11. Реализация базового класса

Задачи:

1. Создать заголовочный файл базового класса (.h)
2. Определить конструкторы, деструктор, виртуальные методы
3. Реализовать методы в .cpp файле
4. Написать простые тесты для проверки работы базового класса
5. Результат: Реализованный базовый абстрактный класс.

Методические рекомендации:

```
// Пример базового класса
#ifndef MEDIA_ITEM_H
#define MEDIA_ITEM_H

#include <string>
#include <iostream>

class MediaItem {
protected:
    std::string title;
    int year;
    std::string genre;

public:
    MediaItem(const std::string& t, int y, const std::string& g);
    virtual ~MediaItem() {}

    // Геттеры (не виртуальные)
    std::string getTitle() const { return title; }
    int getYear() const { return year; }
    std::string getGenre() const { return genre; }

    // Чисто виртуальные методы
    virtual double getDuration() const = 0;  // длительность (мин)
    virtual void printInfo() const = 0;

    // Виртуальный метод с реализацией по умолчанию
    virtual bool isLong() const {
        return getDuration() > 120;
    }
};

#endif
```

Неделя 12-13. Реализация производных классов

Задачи:

1. Создать заголовочные файлы для производных классов
2. Реализовать специфичные поля и методы
3. Переопределить виртуальные методы базового класса
4. Написать тесты для проверки работы производных классов
5. Результат: Реализованные производные классы.

Методические рекомендации:

```
// Пример производного класса
#ifndef BOOK_H
#define BOOK_H

#include "MediaItem.h"

class Book : public MediaItem {
private:
    int pages;          // количество страниц
    std::string author; // автор

public:
    Book(const std::string& t, int y, const std::string& g,
         int p, const std::string& a);

    // Переопределение виртуальных методов
    double getDuration() const override {
        // Для книг "длительность" - время чтения (страницы / 2)
        return pages / 2.0;
    }

    void printInfo() const override;

    // Специфичный метод
    void read() const {
        std::cout << "Читаем книгу \"" << title << "\"" << std::endl;
    }

    // Дополнительные геттеры
    int getPages() const { return pages; }
    std::string getAuthor() const { return author; }
};

#endif
```

Неделя 14. Реализация контейнера и меню

Задачи:

1. Создать класс-контейнер для хранения объектов (или использовать std::vector)
2. Реализовать функции добавления, удаления, просмотра
3. Создать консольное меню
4. Реализовать обработку ошибок ввода
5. Результат: Работающее меню, возможность добавлять/удалять/просматривать объекты.

Методические рекомендации:

```
// Пример меню
class Library {
private:
    std::vector<std::unique_ptr<MediaItem>> items;

public:
    void addBook();
    void addDVD();
    void addMagazine();
    void removeItem(int index);
    void showAll() const;
    void showMenu();
    void run();
};

void Library::showMenu() {
    std::cout << "\n=== Система управления библиотекой ===" << std::endl;
    std::cout << "1. Добавить книгу" << std::endl;
    std::cout << "2. Добавить DVD" << std::endl;
    std::cout << "3. Добавить журнал" << std::endl;
    std::cout << "4. Показать все" << std::endl;
    std::cout << "5. Сортировать по году" << std::endl;
    std::cout << "6. Сохранить в файл" << std::endl;
    std::cout << "7. Загрузить из файла" << std::endl;
    std::cout << "0. Выход" << std::endl;
    std::cout << "Выберите действие: ";
}

void Library::run() {
    int choice;
    do {
```

```
        showMenu();
        std::cin >> choice;

        switch (choice) {
            case 1: addBook(); break;
            case 2: addDVD(); break;
            case 3: addMagazine(); break;
            case 4: showAll(); break;
            case 5: sortByYear(); break;
            case 6: saveToFile(); break;
            case 7: loadFromFile(); break;
            case 0: std::cout << "До свидания!" << std::endl; break;
            default: std::cout << "Неверный выбор!" << std::endl;
        }
    } while (choice != 0);
}
```

Неделя 15. Реализация сортировки и поиска
Задачи:

1. Реализовать сортировку коллекции по одному из критериев
2. Реализовать поиск элементов по критерию
3. Результат: Возможность сортировки и поиска объектов.

Методические рекомендации:

```
// Сортировка по году
void Library::sortByYear() {
    std::sort(items.begin(), items.end(),
        [](const std::unique_ptr<MediaItem>& a,
           const std::unique_ptr<MediaItem>& b) {
            return a->getYear() < b->getYear();
        });
    std::cout << "Коллекция отсортирована по году" << std::endl;
    showAll();
}

// Поиск по названию
void Library::findByTitle(const std::string& title) const {
    auto it = std::find_if(items.begin(), items.end(),
        [&title](const std::unique_ptr<MediaItem>& item) {
            return item->getTitle() == title;
        });
```

```
    if (it != items.end()) {
        (*it)->printInfo();
    } else {
        std::cout << "Элемент не найден" << std::endl;
    }
}
```

Неделя 16. Реализация сериализации

Задачи:

1. Реализовать сохранение коллекции в текстовый или бинарный файл
2. Реализовать загрузку коллекции из файла
3. Обработать ошибки при работе с файлами
4. Результат: Возможность сохранения и восстановления состояния программы.

Методические рекомендации:

```
// Сохранение в текстовый файл
void Library::saveToFile(const std::string& filename) const {
    std::ofstream out(filename);
    if (!out) {
        throw std::runtime_error("Не удалось открыть файл для записи");
    }

    for (const auto& item : items) {
        // Сохраняем тип объекта, затем его данные
        out << item->getType() << std::endl;  // "Book", "DVD", "Magazine"
        out << item->getTitle() << std::endl;
        out << item->getYear() << std::endl;
        out << item->getGenre() << std::endl;

        // Сохраняем специфичные поля
        if (auto* book = dynamic_cast<Book*>(item.get())) {
            out << book->getPages() << std::endl;
            out << book->getAuthor() << std::endl;
        }
        // ... аналогично для DVD и Magazine
    }

    std::cout << "Данные сохранены в файл " << filename << std::endl;
}
```

Неделя 17. Тестирование и отладка

Задачи:

1. Протестировать все функции программы
2. Исправить найденные ошибки
3. Проверить обработку исключительных ситуаций

Результат: Полностью работоспособная программа.

Контрольный список тестирования:

	Проверяемая функция	**Ожидаемый результат**
	Добавление объекта	Объект появляется в коллекции
	Удаление объекта	Объект исчезает из коллекции
	Просмотр всех объектов	Вывод всех объектов на экран
	Сортировка	Элементы упорядочены по критерию
	Поиск	Найден нужный элемент
	Сохранение в файл	Файл создан/обновлён
	Загрузка из файла	Данные восстановлены
	Пустая коллекция	Корректное сообщение
	Неверный ввод	Обработка ошибки, повтор запроса

Неделя 18. Оформление отчёта и защита ЛР23

Задачи:

1. Оформить отчёт по ЛР23 (проектирование)
2. Включить в отчёт диаграмму UML и описание классов
3. Подготовиться к защите ЛР23

Результат: Сданная ЛР23.

Недели 19-24. Доработка и защита ЛР24

Задачи:

1. Доработать проект в соответствии с замечаниями
2. Оформить отчёт по ЛР24
3. Подготовить презентацию проекта (5-7 минут)

Защитить итоговый проект

Результат: Сданная ЛР24.

3.3 Работа с Git-репозиторием

Основные команды Git:

Команда	Назначение
git init	Создать новый репозиторий
git add .	Добавить все файлы в отслеживаемые
git commit -m "message"	Зафиксировать изменения
git push	Отправить изменения на сервер
git pull	Получить изменения с сервера
git status	Показать статус репозитория
git log	Показать историю коммитов

Структура коммитов (рекомендуемая):

feat: добавлен базовый класс MediaItem
feat: добавлен класс Book
feat: добавлен класс DVD
feat: реализовано меню
feat: добавлена сортировка по году
fix: исправлена ошибка в поиске
docs: обновлена документация
test: добавлены тесты для класса Book

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

4.1 Вопросы по структурному программированию

1. Какие типы данных существуют в C++? Чем отличается int от long?
2. Что такое указатель? Как объявить указатель и получить значение по адресу?
3. В чём разница между передачей параметров по значению, по указателю и по ссылке?
4. Что такое массив? Чем статический массив отличается от динамического?

5. Что такое структура? Чем структура отличается от массива?
6. Какие функции работы со строками вы знаете (strcpy, strcat, strlen, strcmp)?
7. Что такое рекурсия? Приведите пример рекурсивной функции.
8. Какие алгоритмы сортировки вы знаете? В чём их сложность?
9. Что такое бинарный поиск? Какая у него сложность?
10. Как организовать работу с текстовыми и бинарными файлами в C++?

4.2 Вопросы по объектно-ориентированному программированию

1. Что такое класс? Чем класс отличается от структуры?
2. Что такое инкапсуляция? Для чего нужны спецификаторы private, protected, public?
3. Что такое конструктор? Какие виды конструкторов бывают?
4. Что такое деструктор? Когда он вызывается?
5. Что такое «правило трёх»? Когда оно применяется?
6. Что такое наследование? Какие виды наследования есть в C++?
7. Что такое виртуальный метод? Для чего он нужен?
8. Что такое абстрактный класс? Чисто виртуальный метод?
9. Что такое полиморфизм? Раннее и позднее связывание?
10. Что такое виртуальный деструктор? Почему он важен?
11. Что такое композиция и агрегация? В чём их различие?
12. Что такое статические поля и методы? Для чего они используются?
13. Что такое дружественные функции и классы?
14. Что такое шаблоны классов? Как их использовать?
15. Что такое перегрузка операторов? Какие операторы можно перегружать?

4.3 Вопросы по алгоритмам и структурам данных

1. Что такое O-нотация (Big O)? Для чего она используется?
2. Какие классы сложности вы знаете? Приведите примеры.
3. Что такое связный список? В чём его преимущества перед массивом?
4. Как реализовать стек? Какие операции для него определены?
5. Как реализовать очередь? Какие операции для него определены?
6. Что такое бинарное дерево? Какие обходы дерева вы знаете?
7. Что такое бинарное дерево поиска (BST)? Какова его сложность?
8. Что такое AVL-дерево? Зачем нужна балансировка?
9. Что такое хеш-таблица? Как разрешаются коллизии?
10. Что такое граф? Какие способы представления графов вы знаете?
11. Что такое DFS и BFS? В чём их различие?
12. Что такое алгоритм Дейкстры? Для чего он используется?

13. Что такое минимальное остовное дерево? Алгоритмы Прима и Краскала?
14. Что такое динамическое программирование? Задача о рюкзаке?
15. Что такое алгоритм Кнута-Морриса-Пратта? В чём его преимущество?

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

5.1 Шкала оценивания лабораторных работ (ЛР1-22)

Максимальный балл за каждую ЛР: 100

Баллы	Традиционная оценка
90-100	Отлично
75-89	Хорошо
60-74	Удовлетворительно
0-59	Неудовлетворительно

Распределение баллов по ЛР1-22:

№	Лабораторная работа	Базовый	Средний	Продвинутый
	Текстовые файлы	60	80	100
	Бинарные файлы	60	80	100
	Классы, геттеры/сеттеры	60	80	100
	IntVector, правило трёх	60	80	100
	Перегрузка операторов	60	80	100
	Композиция, агрегация	60	80	100
	Анализ сложности	60	80	100
	Односвязный список	60	80	100
	Двусвязный список	60	80	100

№	Лабораторная работа	Базовый	Средний	Продвинутый
0	Стек, очередь	60	80	100
1	Дек, priority_queue	60	80	100
2	Шаблонный вектор	60	80	100
3	Рекурсия	60	80	100
4	Простые сортировки	60	80	100
5	Быстрая, слиянием	60	80	100
6	Наследование	60	80	100
7	AVL, Trie	60	80	100
8	Графы, DFS/BFS	60	80	100
9	Дейкстра, Флойд-Уоршелл	60	80	100
0	Прим, Краскал	60	80	100
1	ДП (рюкзак, LCS)	60	80	100
2	КМП, STL	60	80	100

5.2 Критерии защиты итогового проекта (ЛР24)

Максимальный балл: 100

№	Критерий	Баллы
1	Иерархия классов (из ЛР23)	20
1.1	Базовый абстрактный класс с чисто виртуальными методами	10
1.2	Производные классы (минимум 2), переопределение методов	10
2	Реализация функциональности	30
2.1	Использование контейнеров STL (vector, list, map)	10
2.2	Реализация сортировки элементов коллекции	10
2.3	Реализация поиска элементов по критериям	10
3	Сериализация	15
3.1	Сохранение данных в файл	8
3.2	Загрузка данных из файла	7
4	Пользовательский интерфейс	15
4.1	Меню с выбором действий	5
4.2	Обработка ошибок ввода	5
4.3	Обработка исключений	5
5	Защита проекта	20
5.1	Презентация проекта (5-7 минут)	5
5.2	Демонстрация работы программы	5

№	Критерий	Баллы
5.3	Ответы на вопросы по проекту	5
5.4	Качество кода и комментарии	5
Итого		100

Штрафные баллы (ЛР24):

Нарушение	Штраф
Отсутствие виртуального деструктора в базовом классе	-15
Утечка памяти (не освобождена динамическая память)	-15
Отсутствие проверок при работе с файлами	-10
Нет меню или оно неработоспособно	-10
Проект не компилируется	работа не принимается

5.3 Критерии оценивания компетенций

Компетенция	Показатели сформированности
ОПК-1 (естественнонаучные и инженерные знания)	Умеет анализировать сложность алгоритмов, применять математические методы для оценки эффективности
ОПК-6 (разработка алгоритмов и программ)	Разрабатывает корректные алгоритмы, пишет программы на C++, использует методы тестирования
ОПК-7 (концепции информатики)	Демонстрирует понимание структур данных, алгоритмов, принципов ООП
ПК-2 (проектирование архитектуры)	Проектирует иерархии классов, использует паттерны, строит UML-

Компетенция	Показатели сформированности
	диаграммы
ПК-3 (разработка ПО)	Реализует алгоритмы на C++, использует STL, работает с файлами и базами данных

Уровни сформированности:

Уровень	Характеристика	Баллы
Пороговый	Студент демонстрирует базовое понимание, но допускает ошибки при применении	60-74
Базовый	Студент корректно применяет знания в стандартных ситуациях	75-89
Средний	Студент уверенно применяет знания, может решать типовые задачи	90-100

ЗАКЛЮЧЕНИЕ

Данные методические указания призваны помочь студентам в организации самостоятельной работы по дисциплине «Алгоритмы и структуры данных». Следуя рекомендациям, студенты смогут:

Эффективно распределить время на выполнение лабораторных работ и проекта

Систематизировать знания по ключевым темам дисциплины

Разработать полноценный программный проект, демонстрирующий владение ООП, алгоритмами и структурами данных

Сформировать портфолио в виде Git-репозитория

Успешное выполнение всех видов самостоятельной работы позволит студенту получить положительную оценку на экзамене и сформировать необходимые профессиональные компетенции.