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 «Введение в программирование и алгоритмы» для студентов направления подготовки 09.03.04 Программная инженерия
Направленность (профиль)
«Разработка и сопровождение программного обеспечения»

Ставрополь 2026

Содержание

Введение 2

Лабораторная работа №1 6

Лабораторная работа №2 31

Лабораторная работа №3 62

Лабораторная работа №4 89

Лабораторная работа №5 129

Лабораторная работа №6 170

Лабораторная работа №7 212

Лабораторная работа №8 255

Лабораторная работа №9 295

Лабораторная работа №10 341

Лабораторная работа №11 387

Лабораторная работа №12 430

Лабораторная работа №13 477

Лабораторная работа №14 526

Лабораторная работа №15 576

Лабораторная работа №16 635

Лабораторная работа №17 692

Лабораторная работа №18 749

Лабораторная работа №19 800

Введение

Настоящий лабораторный практикум предназначен для студентов направления подготовки 09.03.04 «Программная инженерия» и других IT-специальностей, изучающих основы программирования на языке C++.

Практикум охватывает полный цикл обучения программированию - от первых шагов в среде разработки до сложных тем объектно-ориентированного программирования и многофайловой сборки проектов. Всего в состав практикума входит 19 лабораторных работ, каждая из которых посвящена отдельной теме и содержит теоретический материал, методические указания, примеры выполнения и индивидуальные задания трёх уровней сложности.

Цель практикума: Сформировать у студентов системные знания и практические навыки в области алгоритмизации и программирования на языке C++, необходимые для дальнейшего изучения профильных дисциплин и профессиональной деятельности в области программной инженерии.

Задачи практикума:

1. Освоение синтаксиса и базовых конструкций языка C++:
2. переменные, типы данных, арифметические операции;
3. условные операторы и циклы;
4. массивы и строки;
5. функции и рекурсия.
6. Изучение принципов управления памятью:
7. указатели и ссылки;
8. динамическая память (new / delete);
9. предотвращение утечек памяти.

Освоение основ объектно-ориентированного программирования:

1. классы и объекты;
2. инкапсуляция, наследование, полиморфизм;
3. конструкторы, деструктор, перегрузка операторов.
4. Формирование навыков структурного проектирования:
5. декомпозиция задач на подпрограммы;
6. разделение программы на заголовочные файлы и файлы реализации;
7. сборка многофайловых проектов.

Развитие алгоритмического мышления:

1. разработка и анализ алгоритмов;
2. выбор оптимальных структур данных;
3. оценка сложности алгоритмов (начальный уровень).

3. Структура лабораторного практикума

3.1. Перечень лабораторных работ

№	Тема	Уровень сложности
1	Настройка среды разработки. Первая программа	Вводный
2	Переменные и арифметические операции	Базовый
3	Ветвления. Логические операторы	Базовый
4	Множественный выбор (switch)	Базовый
5	Циклы. Табуляция функций	Базовый
6	Одномерные массивы	Базовый
7	Двумерные массивы (матрицы)	Базовый
8	Строки. Базовые операции	Базовый
9	Функции. Декомпозиция задач	Средний
10	Передача параметров. Ссылки	Средний
11	Указатели. Основы работы с памятью	Средний
12	Динамическая память. Массивы переменного размера	Средний
13	Введение в ООП. Классы и объекты	Продвинутый
14	Конструкторы и деструктор	Продвинутый

№	Тема	Уровень сложности
15	Инкапсуляция. Геттеры и сеттеры	Продвинутый
16	Наследование. Базовый и производный классы	Продвинутый
17	Полиморфизм. Виртуальные методы	Продвинутый
18	Перегрузка операторов. Начальное знакомство	Продвинутый
19	Разделение программы на файлы. Сборка проекта	Продвинутый

Логическая структура практикума

Практикум построен по принципу «от простого к сложному» и логически разделён на три модуля:

Модуль 1. Основы алгоритмизации и программирования (работы №1–8)

- Знакомство со средой разработки
- Базовые конструкции языка: переменные, операции, ветвления, циклы
- Работа с одномерными и двумерными массивами
- Строковые данные

Модуль 2. Процедурное программирование и управление памятью (работы №9–12)

- Функции и декомпозиция задач
- Передача параметров по значению, по ссылке, по указателю
- Указатели и арифметика указателей
- Динамическое выделение памяти

Модуль 3. Объектно-ориентированное программирование (работы №13–19)

- Классы и объекты
- Конструкторы и деструктор
- Инкапсуляция, наследование, полиморфизм
- Перегрузка операторов
- Многофайловая структура проектов

Лабораторная работа №1

Настройка среды разработки. Первая программа

1. **Цель работы**: Освоить среду разработки (IDE), научиться создавать, компилировать и запускать простые программы на C++. Получить первый практический опыт работы с потоковым выводом cout и вводом cin.

2. Теоретическое обоснование (подробное)

2.1. Жизненный цикл разработки программы на C++

Программа на C++ проходит несколько этапов от написания исходного кода до исполнения:

Этап	Что происходит	Результат
1. Написание кода	Программист создаёт .cpp и .h файлы	Исходный код
2. Препроцессинг	Обрабатываются директивы #include, #define, #pragma	Расширенный исходный код
3. Компиляция	Каждый .cpp файл превращается в объектный код	.obj или .o файлы
4. Линковка	Объектные файлы связываются с библиотеками	Исполняемый .exe файл
5. Загрузка	ОС загружает программу в память	Процесс в ОЗУ
6.	Процессор выполняет машинные	Результат работы

Этап	Что происходит	Результат
Исполнение	инструкции	

Важно для инженера: Понимание этих этапов помогает диагностировать ошибки:

- ошибки препроцессора (#include не найден)
- ошибки компиляции (синтаксис)
- ошибки линковки (undefined reference)

2.2. Структура программы на C++

2.2.1. Функция main - точка входа

```
int main() {   // int — тип возвращаемого значения
   // тело функции
   return 0;  // 0 означает успех
}
```

Варианты объявления main (стандартные):

```
int main() { }                  // без аргументов
int main(int argc, char* argv[]) { } // с аргументами командной строки
```

Что означает возвращаемое значение:

- 0 или EXIT_SUCCESS → программа завершена успешно
- 1 (или любое ненулевое) → ошибка

2.2.2. Директивы препроцессора #include

Препроцессор текстуально вставляет содержимое указанного файла вместо #include.

```
#include <iostream>  // угловые скобки — для системных заголовков
#include "myfile.h"   // кавычки — для своих файлов
```

2.2.3. Пространство имён std

Все компоненты стандартной библиотеки C++ находятся в пространстве имён std. Это сделано, чтобы избежать конфликтов имён.

```cpp
std::cout << "Hello";  // явное указание
```

Способы работы с пространством имён:

```cpp
// Способ 1 (не рекомендуется на крупных проектах)
using namespace std;  // импортирует всё, риск конфликтов

// Способ 2 (рекомендуется для обучения)
using std::cout;
using std::cin;
using std::endl;

// Способ 3 (наиболее безопасный, используем в этой лабораторной)
std::cout << "Hello";
```

2.3. Потоковый ввод-вывод

В C++ используется концепция потоков - абстракция последовательности байтов.

Поток	Назначение	Объект
Стандартный вывод	Вывод на экран	std::cout
Стандартный ввод	Ввод с клавиатуры	std::cin
Вывод ошибок	Вывод сообщений об ошибках	std::cerr

2.3.1. Оператор вывода <<

Оператор << называется оператором вставки (insertion operator). Он направляет данные в поток.

```
cpp
std::cout << "Текст";    // вывод строки
std::cout << 42;         // вывод числа
std::cout << 3.14;       // вывод double
```

Цепочка операций:

```
std::cout << "x = " << x << std::endl;
// эквивалентно (((std::cout << "x = ") << x) << std::endl);
// Каждая операция возвращает ссылку на std::cout
```

2.3.2. Манипуляторы вывода

Манипулятор	Эффект	Пример
std::endl	Перевод строки + сброс буфера	std::cout << std::endl;
\n	Только перевод строки (быстрее)	std::cout << "\n";
std::flush	Принудительный сброс буфера	std::cout << std::flush;

Инженерная деталь: std::endl сбрасывает буфер, что может снижать производительность при большом количестве выводов. В циклах лучше использовать "\n".

2.3.3. Оператор ввода >>

Оператор >> - оператор извлечения (extraction operator). Он читает данные из потока в переменную.

```
int age;
std::cin >> age;  // ожидает ввод числа с клавиатуры
```

Особенности cin:

- Пропускает пробельные символы (пробел, табуляцию, перевод строки)
- Останавливается при первом неподходящем символе

Пример проблемного ввода:

```
int x;
std::cin >> x;  // пользователь вводит "abc" → x остаётся неинициализированным
```

2.4. Переменные и типы данных

2.4.1. Основные встроенные типы C++

Тип	Размер (обычно)	Диапазон	Пример
int	4 байта	$-2^{31} \dots 2^{31}-1$	int x = 10;
double	8 байт	±1.7e±308	double pi = 3.14159;
float	4 байта	±3.4e±38	float f = 3.14f;
char	1 байт	-128 … 127	char c = 'A';
bool	1 байт	true / false	bool flag = true;

Правило выбора типа:

- целые числа → int (или long long для очень больших)
- дробные → double (дает больше точности, чем float)

2.4.2. Инициализация переменных

В C++ существуют разные способы инициализации:

```
int a = 10;      // копирующая инициализация (C-style)
int b(20);       // прямая инициализация
int c{30};       // унифицированная инициализация (C++11, рекомендуется)
int d = {40};    // копирующая-унифицированная
```

Преимущество {} (списочной инициализации):

```
int x{3.14};     // ОШИБКА компиляции: сужение типа
int x = 3.14;    // OK, но x станет 3 (потеря данных)
```

2.5. Арифметические операции

Оператор	Значение	Примечание
+	Сложение	
-	Вычитание	
*	Умножение	
/	Деление	Для int - целочисленное
%	Остаток от деления	Только для целых типов

Критическое замечание о делении:

```
int a = 5, b = 2;
```

```
double c = a / b    // c = 2.0 (а не 2.5!)
// Почему? Выполняется целочисленное деление, потом преобразование в double

// Правильно:
double d = 5.0 / 2;   // 2.5
double e = static_cast<double>(a) / b;  // явное преобразование
```

2.6. Обзор интегрированных сред разработки (IDE)

2.6.1. Что такое IDE?

IDE (Integrated Development Environment) объединяет:

- Редактор кода (с подсветкой синтаксиса)
- Компилятор/интерпретатор
- Отладчик (debugger)
- Систему сборки
- Менеджер проектов

2.6.2. Рекомендуемые IDE для C++

IDE	Платформы	Особенность
Visual Studio	Windows	Мощный отладчик, отличная поддержка C++
Code::Blocks	Windows, Linux, Mac	Легкая, хорошо подходит для начинающих
CLion	Все	Платная, интеллектуальная, от JetBrains
Qt Creator	Все	Хороша для GUI приложений

IDE	Платформы	Особенность
VS Code	Все	Легковесный редактор + расширения

2.6.3. Основные действия в IDE

- Создание проекта - файловая структура, настройки компилятора
- Добавление файла - .cpp файл с исходным кодом
- Компиляция - Build / Compile (Ctrl+Shift+B)
- Запуск - Run / Start (Ctrl+F5)

2.6.4. Типичные ошибки новичков и их решения

Ошибка компиляции	Причина	Решение
'cout' was not declared	Забыли #include <iostream>	Добавить include
'cout' is not a member of 'std'	Не указали std::	Использовать std::cout
undefined reference to 'main'	Нет функции main	Добавить int main()
expected ';' before 'return'	Пропущена точка с запятой	Добавить ;
stray '\XXX' in	Проблемы с	Сохранить файл в

Ошибка компиляции	Причина	Решение
program	кодировкой	UTF-8 без BOM

2.7. Базовые принципы структурного программирования

Хотя первая программа простая, важно закладывать правильные привычки:

2. Одна задача - один блок кода
3. Информативные имена переменных: radius вместо r
4. Комментарии к сложным моментам:

```
// Используем 9.0 вместо 9 для вещественного деления
fahrenheit = celsius * 9.0 / 5.0 + 32;
```

Проверка корректности ввода (будет позже, но нужно знать):

```
if (std::cin.fail()) {
    std::cout << "Ошибка ввода!" << std::endl;
}
```

2.8. Полная схема работы программы (на примере перевода градусов)

```
Пользователь запускает программу
        ↓
Выполняется функция main()
        ↓
Вывод приглашения: cout << "Введите градусы: "
        ↓
Ожидание ввода: cin >> celsius
        ↓
Вычисление: fahrenheit = celsius * 9.0 / 5.0 + 32
```

↓

Вывод результата: cout << celsius << "°C = " << fahrenheit << "°F"

↓

Программа завершается, возвращая 0 в ОС

3. Методика выполнения работы

3.1. Пошаговая инструкция для Visual Studio

Запуск Visual Studio → Create a new project

Выбор шаблона: Console App (C++)

Настройка проекта: задать имя и расположение

Структура по умолчанию:

```
#include <iostream>
int main() {
    std::cout << "Hello World!\n";
}
```

Замена кода на свой

Запуск без отладки: Ctrl+F5

3.2. Пример с детальным разбором каждой строки

```
#include <iostream>   // 1. Подключаем библиотеку для ввода/вывода

int main() {          // 2. Объявляем главную функцию
    double celsius;   // 3. Создаём переменную для хранения градусов Цельсия
    double fahr;      // 4. Создаём переменную для результата

    // 5. Выводим приглашение для пользователя
    std::cout << "Enter temperature in Celsius: ";

    // 6. Считываем введённое значение в переменную celsius
    std::cin >> celsius;
```

```
    // 7. Вычисляем значение по формуле
    fahr = celsius * 9.0 / 5.0 + 32;

    // 8. Выводим результат
    std::cout << celsius << " C = " << fahr << " F" << std::endl;

    return 0;       // 9. Успешное завершение программы
}
```

4. Индивидуальные задания

Базовый уровень

Задание 1: Вывести на экран сообщение и выполнить простое вычисление.

№ вар.	Сообщение	Вычисление
1	"Калькулятор периметра квадрата"	сторона → периметр (4*a)
2	"Перевод минут в секунды"	минуты → секунды
3	"Площадь круга"	радиус → площадь (πR^2)
4	"Объём куба"	ребро → объём (a^3)
5	"Стоимость покупки"	цена, кол-во → итог
6	"Среднее двух чисел"	два числа → среднее
7	"Остаток от деления"	a, b → a % b

№ вар.	Сообщение	Вычисление
8	"Квадрат числа"	число → его квадрат
9	"Перевод км в м"	км → метры
10	"Площадь прямоугольника"	a, b → площадь
11	"Периметр треугольника"	три стороны → периметр
12	"Длина окружности"	радиус → 2πR
13	"Разность чисел"	a, b → a - b
14	"Часы в минуты"	часы → минуты
15	"Половина числа"	число → число/2

Средний уровень

Задание 2: Помимо вычислений - обработка двух величин, использование cout и cin в одном выражении.

№ вар.	Задание
1	Запросить сторону квадрата, вывести периметр и площадь
2	Запросить скорость (км/ч) и время (ч), вывести расстояние

№ вар.	Задание
3	Запросить массу (кг) и цену за кг, вывести стоимость
4	Запросить два числа, вывести сумму, разность, произведение
5	Запросить радиус, вывести длину окружности и площадь круга
6	Запросить длину ребра куба, вывести объём и площадь поверхности
7	Запросить три оценки, вывести средний балл
8	Запросить число, вывести его квадрат и куб
9	Запросить катеты прямоугольного треугольника, вывести гипотенузу ($\sqrt{(a^2+b^2)}$)
10	Запросить цену товара и скидку %, вывести итоговую цену
11	Запросить два отрезка в см, вывести их сумму в м и см
12	Запросить секунды, вывести часы, минуты, секунды
13	Запросить скорость (м/с) и время (с), вывести путь

№ вар.	Задание
14	Запросить ширину и высоту прямоугольника, вывести периметр и площадь
15	Запросить зарплату и премию %, вывести итог

Повышенный уровень

Задание 3. Ввод/вывод с форматированием, составные выражения, комментарии.

№ вар.	Задание
1	Запросить длину, ширину, высоту параллелепипеда → объём и площадь поверхности
2	Конвертер: км, м, см → всего в мм
3	Запросить 4 числа → среднее арифметическое и среднее геометрическое
4	Запросить радиус → площадь сектора (угол задаётся в градусах)
5	Запросить основание и высоту треугольника → площадь
6	Программа «Калькулятор времени»: ввести часы, минуты → вывести в минутах и секундах

№ вар.	Задание
7	Запросить количество дней → вывести годы, месяцы (30 дн), дни
8	Прямоугольный треугольник: катеты → периметр и площадь
9	Запросить ускорение и время → скорость и путь (v0=0)
10	Запросить 5 чисел → вывести их сумму и произведение
11	Запросить стоимость обеда и чаевые (%) → общая стоимость
12	Конвертер валют (рубли → доллары → евро по заданному курсу)
13	Запросить число → вывести его целую часть, дробную часть (если double)
14	Программа расчёта индекса массы тела (вес, рост → ИМТ)
15	Запросить количество строк и символов → вывести "псевдопрямоугольник" из символов (только вывод)

5. Контрольные вопросы

1. Чем отличается препроцессор от компилятора?
2. Что произойдет, если в #include <iostream> пропустить угловые скобки и написать кавычки?

3. Почему 9/5 даёт 1, а 9.0/5 даёт 1.8?
4. В каких случаях использование std::endl может навредить производительности?
5. Что случится, если не инициализировать переменную перед использованием?
6. Какие три типа ошибок может содержать программа? Приведите примеры.
7. Что делает оператор << при перегрузке? (общий вопрос, без углубления в ООП)
8. Как можно вывести несколько значений в одном cout без повторного написания std::cout?
9. Зачем нужен return 0;? Что будет, если его не писать?
10. Как проверить, что пользователь ввёл число, а не букву?

6. Требования к отчёту

Отчёт должен содержать:

1. Титульный лист (название работы, дисциплина, студент, преподаватель)
2. Цель работы
3. Краткое теоретическое обоснование (основные понятия)
4. Ход выполнения работы:
   - Скриншот среды разработки с открытым проектом
   - Листинг кода программы (с комментариями)
   - Скриншот(ы) результатов выполнения для разных входных данных
5. Ответы на контрольные вопросы (письменно)
6. Вывод (достигнута ли цель)

Формат отчёта: .docx или .pdf.

В листинге кода обязательно должны быть комментарии к каждой логической части

Скриншоты должны показывать как успешный сценарий, так и реакцию программы на некорректный ввод (если реализована проверка)

В выводе обязательно отразить трудности, возникшие при выполнении, и способы их преодоления

7. Критерии оценивания

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) - правильно выполнено **только базовое задание** (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, отсутствие фатальных ошибок, программа запускается и выводит результат).

Оценка «хорошо» (4) - правильно выполнены **базовый и средний уровни** (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего уровня (использование переменных разных типов, форматирование вывода, осмысленные имена переменных, корректная работа с вещественными числами).

Оценка «отлично» (5) – правильно выполнены **все три уровня** (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (обработка ошибочного ввода через std::cin.fail(), проверка граничных значений, демонстрация понимания приведения типов, подробные комментарии в коде, оформление вывода в виде таблицы или форматированного сообщения).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в

виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	**Базовый (3)**	**Средний (4)**	**Повышенный (5)**
	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
	Компиляция и запуск			
.1	Программа компилируется без ошибок	**+**	**+**	**+**
.2	Программа успешно запускается и не аварийно завершается	**+**	**+**	**+**
.3	Отсутствие предупреждений компилятора	—	**+**	**+**
	Работа с вводом/выводом			
.1	Использование #include <iostream>	**+**	**+**	**+**
.2	Корректное использование std::cout для вывода	**+**	**+**	**+**
	Корректное	**+**	**+**	**+**

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.3	использование std::cin для ввода			
.4	Использование std::endl или \n для перевода строки	+	+	+
.5	Вывод приглашений для пользователя перед вводом	—	+	+
	Переменные и типы данных			
.1	Правильное объявление переменных	+	+	+
.2	Правильный выбор типа данных (int, double, char)	+	+	+
.3	Инициализация переменных перед использованием	+	+	+
.4	Использование осмысленных имён переменных (не a, b, c без пояснений)	—	+	+
.5	Демонстрация разницы между int и double (где применимо)	—	—	+
	Вычисления			
.1	Корректное выполнение арифметических операций	+	+	+
	Правильный порядок операций	+	+	+

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	(использование скобок при необходимости)			
.3	Корректная работа с вещественными числами (9.0/5.0, а не 9/5)	—	+	+
.4	Отсутствие потери точности при делении (явное приведение типов)	—	—	+
	Форматирование вывода			
.1	Вывод результата с поясняющим текстом	+	+	+
.2	Использование std::fixed и std::set precision для точности	—	+	+
.3	Вывод результатов в виде таблицы или форматированных строк	—	—	+
	Обработка ошибок			
.1	Проверка деления на ноль (в заданиях с делением)	—	+	+
.2	Проверка на ввод некорректных данных (std::cin.fail())	—	—	+
.3	Очистка буфера ввода при ошибке (std::cin.clear(), std::cin.ignore())	—	—	+

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Качество кода			
.1	Правильные отступы и форматирование кода	+	+	+
.2	Отсутствие «магических чисел» (константы вынесены в const)	—	+	+
.3	Наличие комментариев в коде (пояснение каждой логической части)	—	+	+
.4	Детальные комментарии с объяснением выбора типа данных	—	—	+
	Отчёт			
.1	Наличие титульного листа, цели, хода работы	+	+	+
.2	Скриншоты результатов выполнения (минимум 2 теста)		+	+
.3	Скриншоты для всех трёх уровней	—	+	+
.4	Таблица тестирования (вход → ожидаемый → фактический результат)	—	+	+
.5	Тестирование на граничных значениях (например, 0, отрицательные числа)	—	—	+

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.6	Блок-схема алгоритма (для повышенного уровня)	—	—	+

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) - обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить **один** вариант из базового уровня (№1–15).

1. Написать программу, которая:
   - Корректно запрашивает входные данные (хотя бы одно значение)
   - Выполняет простое арифметическое вычисление
   - Выводит результат на экран
2. Продемонстрировать понимание:
   - Структуры программы (#include, main, return 0)
   - Использования std::cout для вывода
   - Использования std::cin для ввода
3. Предоставить отчёт с:
   - Титульным листом
   - Целью работы
   - Листингом кода
   - Скриншотами выполнения (минимум 2 набора входных данных)

Средний уровень (оценка 4) - дополнительные требования

Для получения оценки «хорошо» студент должен выполнить **базовый И средний** уровни, а также:

1. Использовать переменные разных типов (int и double).

2. Выводить **приглашения** перед каждым вводом данных.
3. Форматировать вывод с помощью std::fixed и std::setprecision (где уместно).
4. Использовать осмысленные имена переменных (radius, area, price, а не r, a, p).
5. В отчёте представить:
   - Скриншоты для обоих уровней
   - Таблицу тестирования (входные данные → ожидаемый результат → фактический результат)
6. Прокомментировать ключевые строки кода.

Повышенный уровень (оценка 5) - требования на отлично

Для получения оценки «отлично» студент должен выполнить **все три уровня** (базовый, средний, повышенный), а также:

1. Реализовать **обработку ошибочного ввода**:
   - Проверку на ввод букв вместо чисел через std::cin.fail()
   - Очистку буфера ввода (std::cin.clear(), std::cin.ignore())
2. Продемонстрировать понимание **приведения типов**:
   - Явное преобразование static_cast<double>(a) / b при делении
   - Объяснение в комментариях, почему это необходимо
3. Выполнить **тестирование на граничных значениях**:
   - Для калькулятора площади: отрицательные стороны, ноль
   - Для конвертера: нулевая температура, отрицательная
   - Для среднего арифметического: очень большие числа
4. Предоставить **детальные комментарии** в коде:
   - Каждая логическая часть пояснена
   - Объяснён выбор типа данных
   - Указаны потенциальные проблемы (потеря точности, переполнение)

5. В отчёте представить **блок-схему алгоритма** (для самого сложного из выполненных заданий).

6. Оформить вывод в виде **форматированной таблицы** или аккуратного структурированного сообщения.

7.4. Примеры проверки соответствия

Выполненные задания	**Оценка**	**Проверка критериев**
Только базовый вариант №3 (площадь круга)	**3**	+ Компиляция, + ввод/вывод, + вычисление, + скриншоты. Но: ✘ нет форматирования, ✘ нет обработки ошибок, ✘ имена переменных r, s
Базовый №3 + средний №5 (площадь круга + квадрат/куб)	**4**	+ Плюс приглашения, + std::fixed, + осмысленные имена (radius, area), + таблица тестирования. Но: ✘ нет проверки cin.fail(), ✘ нет комментариев о типах
Базовый №3 + средний №5 + повышенный №12 (конвертер валют)	**5**	+ Все критерии выполнены: + обработка ошибок ввода, + явное приведение типов, + тестирование граничных значений (отрицательная сумма, ноль), + подробные комментарии, + блок-схема в отчёте

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	Влияние	Как исправить
Программа не компилируется	Оценка **не ставится**	Проверить синтаксис, наличие ;, закрывающих скобок
Использование int для всех переменных (даже для деления)	Снижение на 1 балл	Использовать double для вещественных результатов
Отсутствие std:: перед cout/cin	Ошибка компиляции	Использовать std::cout или using namespace std;
Деление целых чисел без приведения (5/2=2)	Снижение на 1 балл (для среднего/повышенного)	Использовать 5.0/2 или static_cast<double>(5)/2
Нет приглашений перед вводом	Снижение до «3» (даже при правильных вычислениях)	Добавить std::cout << "Введите...";
Код без комментариев	Снижение на 1 балл	Добавить пояснения к каждой логической части

Ошибка	Влияние	Как исправить
Отсутствие скриншотов в отчёте	Отчёт не принимается	Сделать скриншоты окна консоли с результатами

7.6. Минимальный порог для защиты

- Для допуска к **защите** лабораторной работы студент должен:
- Предоставить работающую программу (компилируется и выполняется)
- Предоставить отчёт, оформленный согласно требованиям (раздел 6)
- Устно объяснить:
- Что делает каждая строка кода
- Почему выбран тот или иной тип данных
- Как происходит ввод и вывод данных

(для оценки 4 и 5) — объяснить разницу между целочисленным и вещественным делением, привести пример

Лабораторная работа №2

Переменные и арифметические операции

1. **Цель работы**: Научиться объявлять переменные базовых типов (int, double, float, char, bool), выполнять арифметические операции (+, -, *, /, %), правильно выбирать тип данных в зависимости от решаемой задачи, понимать механизмы явного и неявного приведения типов, избегать потери точности при вычислениях, корректно работать с целочисленным и вещественным делением.

2. Теоретическое обоснование

2.1. Что такое тип данных?

Тип данных - это характеристика переменной, которая определяет:

- размер выделяемой памяти (сколько байт)
- диапазон возможных значений
- набор допустимых операций над этой переменной

```
int x = 42;        // тип int — 4 байта, диапазон ≈ ±2 млрд
double y = 3.14;   // тип double — 8 байт, диапазон ±1.7e±308, дробные значения
char c = 'A';      // тип char — 1 байт, хранит символ (код 65)
bool flag = true;  // тип bool — 1 байт, значения true (1) или false (0)
```

2.2. Понятие приведения типов (type casting)

Приведение типов - это преобразование значения одного типа данных в значение другого типа. В C++ существует два основных вида приведения:

2.2.1. Неявное приведение (implicit conversion)

Определение: Преобразование, которое компилятор выполняет автоматически, без явного указания программиста. Обычно происходит, когда в выражении смешаны разные типы, или при присваивании значения переменной другого типа.

Когда происходит неявное приведение:

1. При присваивании: int x = 3.14;
2. В арифметических выражениях: double d = 5 + 3.2;
3. При передаче аргументов в функции
4. При возврате значения из функции

```
// Пример 1: неявное приведение при присваивании
int a = 3.14;       // double → int, результат: a = 3 (дробная часть отброшена)
double b = 5;       // int → double, результат: b = 5.0 (без потерь)

// Пример 2: неявное приведение в выражении
double result = 5 / 2;   // оба операнда int → целочисленное деление 5/2=2 → 2.0
// Чтобы получить 2.5, нужно хотя бы один операнд сделать double
double correct = 5.0 / 2;   // double / int → double / double → 2.5

// Пример 3: неявное приведение через повышение (promotion)
char c = 'A';      // ASCII код 65
```

```
int code = c;      // char → int, code = 65 (безопасное расширение)
```

Правила неявного приведения (арифметические преобразования):

1. Если есть long double → остальные приводятся к long double
2. Иначе если есть double → остальные к double
3. Иначе если есть float → остальные к float
4. Иначе выполняются целочисленные преобразования (ранги: int < long < long long)

Исходный тип	**Целевой тип**	**Результат**
double → int	Усечение дробной части	3.14 → 3
int → double	Добавление .0	5 → 5.0
char → int	Код символа	'A' → 65
float → double	Повышение точности	безопасно
double → float	Возможна потеря точности	небезопасно

2.2.2. Явное приведение (explicit conversion)

Определение: Преобразование, которое программист специально указывает в коде с помощью операторов приведения типов. Требуется, когда нужно контролировать преобразование или когда неявное приведение невозможно или даёт нежелательный результат.

Способы явного приведения в C++:

```
// 1. C-style приведение (устаревший, но часто встречается)
```

```
double pi = 3.14159;
int intPi = (int)pi;        // Явное преобразование double → int, результат 3
int intPi2 = int(pi);       // Функциональный стиль, результат 3

// 2. Современное C++ приведение (РЕКОМЕНДУЕТСЯ!)
int intPi3 = static_cast<int>(pi);   // static_cast — проверяет на этапе компиляции
```

Зачем нужно явное приведение:

```
// Сценарий 1: получение вещественного результата от целочисленного деления
int a = 5, b = 2;
double result1 = a / b;             // НЕПРАВИЛЬНО: 5/2=2 → 2.0
double result2 = static_cast<double>(a) / b;  // ПРАВИЛЬНО: (double)5/2=2.5

// Сценарий 2: работа с символами и их кодами
char symbol = 'B';
int code = static_cast<int>(symbol);   // code = 66 (код ASCII 'B')
std::cout << "Код символа: " << code << std::endl;

// Сценарий 3: отбрасывание дробной части с предупреждением компилятора
double price = 19.99;
int dollars = static_cast<int>(price);  // dollars = 19 (явно говорим, что делаем это намеренно)

// Сценарий 4: преобразование для работы с оператором % (требует целые числа)
double x = 10.0, y = 3.0;
// int remainder = x % y;   // ОШИБКА! % не работает с double
int remainder = static_cast<int>(x) % static_cast<int>(y);  // 10 % 3 = 1
```

Таблица: когда нужно явное приведение

Ситуация	**Почему нужно явное приведение**	**Пример**
Целочисленное деление → вещественное	Иначе будет целочисленное деление	static_cast<double>(a)/b
Использование % с вещественными	% работает только с целыми	int(intX) % int(intY)
Сужение (double → int)	Компилятор может выдать предупреждение	static_cast<int>(value)

Ситуация	Почему нужно явное приведение	Пример
Управление точностью	Явно показываем намерение	static_cast<float>(value)

2.3. Опасности приведения типов

2.3.1. Потеря точности (precision loss)

```
// Целочисленные типы
int largeNumber = 2'000'000'000;     // 2 миллиарда
short smallNumber = static_cast<short>(largeNumber);   // ПОТЕРЯ! short вмещает только до 32767

// Вещественные типы
double precise = 1.23456789012345;
float lessPrecise = static_cast<float>(precise);  // Потеря точности (float ~6-7 знаков)
```

2.3.2. Переполнение (overflow)

```
int maxInt = 2'147'483'647;   // максимальное значение int
maxInt = maxInt + 1;                    // ПЕРЕПОЛНЕНИЕ! станет -2_147_483_648 (циклическое зацикливание)
```

2.3.3. Усечение (truncation)

```
double value = 3.9999;
int truncated = static_cast<int>(value);   // truncated = 3 (НЕ округление!)
// Округление делается через std::round()
int rounded = static_cast<int>(std::round(value));  // rounded = 4
```

2.4. Полная классификация приведения типов

```
// ========== 1. НЕЯВНОЕ ПРИВЕДЕНИЕ (автоматическое) ==========
int a = 3.14;         // double → int, предупреждение компилятора
double b = 5;          // int → double, безопасно
char c = 65;          // int → char, c = 'A'
int d = c;           // char → int, d = 65

// ========== 2. ЯВНОЕ ПРИВЕДЕНИЕ (программист управляет) ==========

// 2.1 C-style (не рекомендуется в современном C++)
int e = (int)3.14;     // double → int
int f = int(3.14);     // функциональный стиль

// 2.2 static_cast (РЕКОМЕНДУЕТСЯ)
int g = static_cast<int>(3.14);    // явное, контролируемое преобразование
double h = static_cast<double>(5); // int → double
```

```
// 2.3 const_cast (для снятия const, редко, не сейчас)
// 2.4 reinterpret_cast (опасное, для низкоуровневых операций)
// 2.5 dynamic_cast (для полиморфизма, будет позже)
```

2.5. Базовые типы данных (подробно)

2.5.1. Целочисленные типы

Тип	Размер	Диапазон	Примечание
short	2 байта	-32 768 … 32 767	Экономия памяти
int	4 байта	-2 147 483 648 … 2 147 483 647	Основной выбор
long	4 или 8	зависит от ОС	Для кроссплатформенности
long long	8 байт	$-9 \cdot 10^{18}$ … $9 \cdot 10^{18}$	Очень большие числа
unsigned int	4 байта	0 … 4 294 967 295	Только неотрицательные

```
// Примеры объявления и инициализации
int students = 150;                    // обычное целое
long long population = 8'000'000'000LL;    // 8 миллиардов, LL = long long
unsigned int positive = 4'000'000'000U;    // U = unsigned
short small = 10000S;                  // S = short
```

2.5.2. Вещественные типы

Тип	Размер	Точность	Диапазон
float	4 байта	~7 знаков	$\pm 1.2 \cdot 10^{-38}$ … $\pm 3.4 \cdot 10^{38}$
double	8 байт	~16 знаков	$\pm 2.2 \cdot 10^{-308}$ … $\pm 1.8 \cdot 10^{308}$
long double	8/16 байт	зависит	расширенный

```
float f = 3.14159f;        // f в конце для float
double d = 3.1415926535;   // double по умолчанию
long double ld = 3.141592653589793L;  // L для long double
```

2.6. Арифметические операции и приоритет

Оператор	Название	Приоритет	Ассоциативность
()	Скобки	1 (высший)	слева направо
* / %	Умножение, деление, остаток	2	слева направо
+ -	Сложение, вычитание	3	слева направо

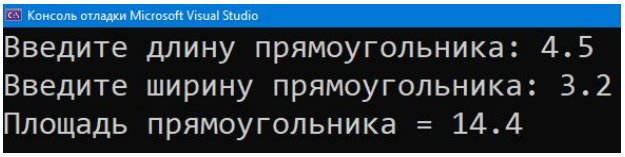
```
// Приоритет определяет порядок вычислений
int result1 = 5 + 3 * 2;     // = 5 + (3*2) = 5 + 6 = 11
int result2 = (5 + 3) * 2;   // = 8 * 2 = 16
int result3 = 10 / 2 * 3;    // = (10/2)*3 = 5*3 = 15 (слева направо)
```

3. Методика выполнения работы с примерами (код с комментариями)

3.1. Пример 1: Площадь прямоугольника (демонстрация выбора типа данных)

```
// ============================================================
// Программа: вычисление площади прямоугольника
// Цель: демонстрация выбора типа double для вещественных чисел
// ============================================================

#include <iostream>  // Подключаем библиотеку для ввода/вывода

int main() {
    // Объявляем переменные типа double.
    // Почему double? Длина и ширина могут быть дробными (например, 2.5 см)
    setlocale(LC_ALL, "ru");
    double length;   // Длина прямоугольника
    double width;    // Ширина прямоугольника
    double area;     // Площадь (результат вычислений)

    // Выводим приглашение для пользователя
    std::cout << "Введите длину прямоугольника: ";
    // Считываем значение с клавиатуры в переменную length
    std::cin >> length;

    std::cout << "Введите ширину прямоугольника: ";
    std::cin >> width;

    // Вычисляем площадь: произведение длины на ширину
    // Оба операнда double -> результат double
    area = length * width;

    // Выводим результат
    std::cout << "Площадь прямоугольника = " << area << std::endl;

    return 0;  // Успешное завершение программы
}

// Пример работы:
// Вход: length = 4.5, width = 3.2
// Выход: Площадь = 14.4
```

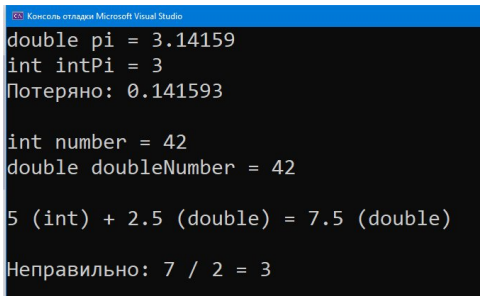
Результат программы:



3.2. Пример 2: Демонстрация НЕЯВНОГО приведения типов

```
// ============================================================
// Программа: демонстрация неявного приведения типов
```

```cpp
// Цель: показать, как компилятор автоматически преобразует типы
// ============================================================

#include <iostream>

int main() {
    setlocale(LC_ALL, "ru");
    // ========== СЦЕНАРИЙ 1: Присваивание double -> int ==========
    double pi = 3.1415926535;  // Исходное значение double
    int intPi = pi;            // НЕЯВНОЕ приведение: double -> int
    // Результат: дробная часть ОТБРАСЫВАЕТСЯ (не округляется!)

    std::cout << "double pi = " << pi << std::endl;        // 3.1415926535
    std::cout << "int intPi = " << intPi << std::endl;     // 3 (потеря данных!)
    std::cout << "Потеряно: " << pi - intPi << std::endl;  // 0.1415926535

    // ========== СЦЕНАРИЙ 2: Присваивание int -> double ==========
    int number = 42;           // Целое значение
    double doubleNumber = number;  // НЕЯВНОЕ приведение: int -> double
    // Результат: 42.0 (без потерь)

    std::cout << "\nint number = " << number << std::endl;           // 42
    std::cout << "double doubleNumber = " << doubleNumber << std::endl; // 42

    // ========== СЦЕНАРИЙ 3: Смешанные операции ==========
    int a = 5;
    double b = 2.5;
    double c = a + b;  // НЕЯВНОЕ приведение: int a преобразуется в double (5 -> 5.0)
    // Затем: 5.0 + 2.5 = 7.5
    std::cout << "\n5 (int) + 2.5 (double) = " << c << " (double)" << std::endl;

    // ========== СЦЕНАРИЙ 4: ОПАСНОСТЬ! Целочисленное деление ==========
    int x = 7;
    int y = 2;
    double wrongResult = x / y;  // НЕПРАВИЛЬНО!
    // Порядок: 7 / 2 = 3 (целочисленное деление!)
    // Затем: 3 НЕЯВНО приводится к double -> 3.0

    std::cout << "\nНеправильно: 7 / 2 = " << wrongResult << std::endl;  // 3.0, а не 3.5!

    return 0;
}

/* ВЫВОД ПРОГРАММЫ:
double pi = 3.14159
int intPi = 3
Потеряно: 0.14159

int number = 42
double doubleNumber = 42

5 (int) + 2.5 (double) = 7.5 (double)

Неправильно: 7 / 2 = 3
*/
```



3.3. Пример 3: Демонстрация ЯВНОГО приведения типов (static_cast)

```
// ============================================================
// Программа: демонстрация явного приведения типов с static_cast
// Цель: показать, как программист может управлять преобразованиями
// ============================================================

#include <iostream>
#include <cmath>   // для std::round (округление)

int main() {
    setlocale(LC_ALL, "ru");
    // ========== СЦЕНАРИЙ 1: Исправление целочисленного деления ==========
    int a = 7;
    int b = 2;

    // Плохой способ (неявное приведение не помогает)
    double bad = a / b;                 // 7/2=3 → 3.0

    // Хороший способ: явное приведение одного операнда к double
    double good1 = static_cast<double>(a) / b;  // (double)7 / 2 = 3.5
    double good2 = a / static_cast<double>(b);  // 7 / (double)2 = 3.5

    std::cout << "=== Исправление деления ===" << std::endl;
    std::cout << "Неправильно: " << bad << std::endl;      // 3
    std::cout << "Правильно: " << good1 << std::endl;      // 3.5

    // ========== СЦЕНАРИЙ 2: Оператор % (остаток) требует целые числа ==========
    double x = 10.7;
    double y = 3.2;

    // double remainder = x % y;   // ОШИБКА! % не работает с double!

    // Решение: явно приводим к int (теряем дробную часть!)
    int ix = static_cast<int>(x);   // 10
    int iy = static_cast<int>(y);   // 3
    int remainder = ix % iy;        // 10 % 3 = 1

    std::cout << "\n=== Оператор % с вещественными ===" << std::endl;
    std::cout << x << " % " << y << " (приведённые) = " << remainder << std::endl;

    // ========== СЦЕНАРИЙ 3: Отбрасывание дробной части (усечение) ==========
    double price = 19.99;
    int dollars = static_cast<int>(price);  // Явно отбрасываем копейки
    int cents = static_cast<int>((price - dollars) * 100);  // 99 копеек

    std::cout << "\n=== Работа с деньгами ===" << std::endl;
    std::cout << "Цена: " << price << " руб" << std::endl;
    std::cout << "Целых рублей: " << dollars << std::endl;
    std::cout << "Копеек: " << cents << std::endl;
```

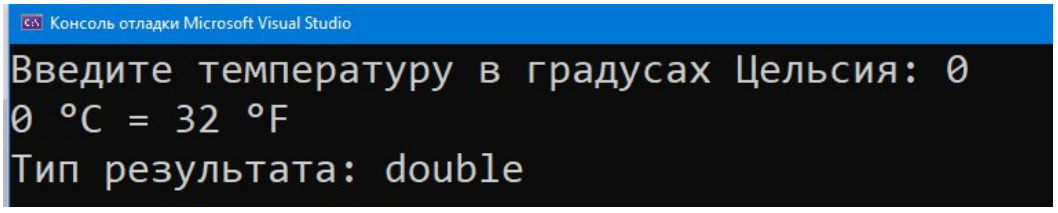
```cpp
    // ========== СЦЕНАРИЙ 4: Округление (не усечение!) ==========
    double values[] = { 3.1, 3.5, 3.9, -3.1, -3.5, -3.9 };

    std::cout << "\n=== Усечение vs Округление ===" << std::endl;
    for (double val : values) {
        int truncated = static_cast<int>(val);          // Просто отбрасываем дробную часть
        int rounded = static_cast<int>(std::round(val)); // Математическое округление

        std::cout << val << " → усечение: " << truncated
            << ", округление: " << rounded << std::endl;
    }

    // ========== СЦЕНАРИЙ 5: Работа с символами ==========
    char letter = 'B';
    int asciiCode = static_cast<int>(letter);   // Явно: char → int

    std::cout << "\n=== Коды символов ===" << std::endl;
    std::cout << "Символ '" << letter << "' имеет ASCII код: " << asciiCode << std::endl;

    // Обратное преобразование: int → char
    int code = 90;
    char symbol = static_cast<char>(code);      // 90 в ASCII — это 'Z'
    std::cout << "Код " << code << " соответствует символу '" << symbol << "'" << std::endl;

    return 0;
}

/* ВЫВОД ПРОГРАММЫ:
=== Исправление деления ===
Неправильно: 3
Правильно: 3.5

=== Оператор % с вещественными ===
10.7 % 3.2 (приведённые) = 1

=== Работа с деньгами ===
Цена: 19.99 руб
Целых рублей: 19
Копеек: 99

=== Усечение vs Округление ===
3.1 → усечение: 3, округление: 3
3.5 → усечение: 3, округление: 4
3.9 → усечение: 3, округление: 4
-3.1 → усечение: -3, округление: -3
-3.5 → усечение: -3, округление: -4
-3.9 → усечение: -3, округление: -4

=== Коды символов ===
Символ 'B' имеет ASCII код: 66
Код 90 соответствует символу 'Z'
*/
```

```
=== Исправление деления ===
Неправильно: 3
Правильно: 3.5

=== Оператор % с вещественными ===
10.7 % 3.2 (приведённые) = 1

=== Работа с деньгами ===
Цена: 19.99 руб
Целых рублей: 19
Копеек: 98

=== Усечение vs Округление ===
3.1 ? усечение: 3, округление: 3
3.5 ? усечение: 3, округление: 4
3.9 ? усечение: 3, округление: 4
-3.1 ? усечение: -3, округление: -3
-3.5 ? усечение: -3, округление: -4
-3.9 ? усечение: -3, округление: -4

=== Коды символов ===
Символ 'B' имеет ASCII код: 66
Код 90 соответствует символу 'Z'
```

3.4. Пример 4: Программа перевода градусов (полный разбор)

```
// ============================================================
// Программа: перевод градусов Цельсия в Фаренгейты
// Формула: F = C * 9/5 + 32
// Цель: демонстрация выбора типа данных и явного приведения
// ============================================================

#include <iostream>   // Для std::cout, std::cin, std::endl

int main() {
    setlocale(LC_ALL, "ru");
    // Объявляем переменные типа double.
    // Почему double? Температура может быть дробной (например, 36.6 °C)
    double celsius;   // Температура в градусах Цельсия (ввод пользователя)
    double fahrenheit;// Температура в градусах Фаренгейта (результат)

    // Выводим приглашение для пользователя
    std::cout << "Введите температуру в градусах Цельсия: ";

    // Считываем значение с клавиатуры в переменную celsius
    std::cin >> celsius;

    // ========== ВАЖНО: вычисление по формуле ==========
    // Проблема: 9/5 в C++ — это ЦЕЛОЧИСЛЕННОЕ деление (9/5=1)
    // Неправильно: fahrenheit = celsius * 9/5 + 32;   // даст celsius * 1 + 32

    // Решение 1: использовать вещественные литералы
    fahrenheit = celsius * 9.0 / 5.0 + 32;
    // 9.0 — это double, 5.0 — это double
    // double / double → вещественное деление (1.8)

    // Решение 2: явное приведение типов (более явное)
    // fahrenheit = celsius * static_cast<double>(9) / 5 + 32;

    // Выводим результат с пояснением
```

```
    std::cout << celsius << " °C = " << fahrenheit << " °F" << std::endl;

    // Дополнительно: выводим тип результата (для демонстрации)
    std::cout << "Тип результата: double" << std::endl;

    return 0;  // Успешное завершение программы
}

// Примеры работы программы:
// Ввод: 0    → Вывод: 0 °C = 32 °F (замерзание воды)
// Ввод: 100  → Вывод: 100 °C = 212 °F (кипение воды)
// Ввод: 36.6 → Вывод: 36.6 °C = 97.88 °F (температура тела)
```



4. Индивидуальные задания (15 вариантов на каждый уровень)

Базовый уровень

Задание 1. Выполнить одно прямое вычисление с правильным выбором типа данных.

№	Задание	Требования к типам
1	Вычислить периметр квадрата со стороной a	int или double в зависимости от ввода
2	Вычислить площадь круга радиуса R (π=3.14)	double для π и результата
3	Вычислить объём куба с ребром a	Целочисленный или вещественный
4	Перевести метры в сантиметры	Целые числа
5	Вычислить стоимость n кг товара по цене p за кг	double для цены

№	Задание	Требования к типам
6	Вычислить среднее арифметическое двух чисел	double (деление на 2.0!)
7	Найти остаток от деления a на b	Только int
8	Вычислить квадрат числа x	Тот же тип, что и x
9	Перевести километры в метры	int
10	Вычислить периметр прямоугольника со сторонами a и b	double (стороны могут быть дробными)
11	Перевести часы в минуты	int
12	Вычислить длину окружности радиуса R	double
13	Найти разность двух чисел a – b	int или double
14	Перевести градусы Цельсия в Фаренгейты	double, обязательно 9.0/5.0
15	Вычислить площадь прямоугольного треугольника	double, S = (a * b) / 2.0

№	Задание	Требования к типам
	(катеты a, b)	

Средний уровень

Задание 2. Выполнить несколько вычислений с использованием явного приведения типов.

№	Задание	Акцент на приведение типов
1	Вычислить периметр и площадь прямоугольника. Объяснить выбор типа.	Показать, почему double лучше int
2	Вычислить длину окружности и площадь круга. Использовать const double PI.	Явное приведение при возведении в квадрат
3	Вычислить объём и площадь поверхности куба.	Сравнить результат с int и double
4	По трём оценкам вычислить средний балл.	Обязательно деление на 3.0 (не на 3)
5	Вычислить квадрат и куб числа. Показать неявное приведение.	Продемонстрировать int → double

№	Задание	Акцент на приведение типов
6	По двум катетам вычислить гипотенузу (c = √(a² + b²)).	static_cast<double> перед sqrt
7	По цене товара и скидке в % вычислить итоговую цену.	Явное приведение для процентов (%/100.0)
8	Перевести секунды в часы, минуты, секунды.	Целочисленное деление и %
9	Вычислить объём параллелепипеда и площадь поверхности.	Все три измерения могут быть дробными
10	Вычислить среднее арифметическое и среднее геометрическое (√(a*b)).	Явное приведение к double
11	По радиусу и высоте вычислить объём цилиндра (V = πR²h).	R*R с явным указанием типа
12	Перевести скорость из км/ч в м/с (умножить на 1000/3600).	Показать проблему 1000/3600 → 0
1	Вычислить расстояние	Если время целое → явное

№	Задание	Акцент на приведение типов
3	по скорости и времени.	приведение
14	По координатам двух точек вычислить расстояние между ними.	Все операции с double
15	Вычислить значение выражения y = (a + b) / (c - d) с проверкой деления на ноль.	Явное приведение для вещественного деления

Повышенный уровень

Задание 3. Сложные вычисления, демонстрация разных способов приведения, анализ потери точности.

№	Задание	Требования
1	Вычислить объём и площадь поверхности сферы. Сравнить результаты с float и double	Показать разницу в точности
2	По трём сторонам вычислить площадь по формуле Герона. Продемонстрировать потерю точности.	Использовать double и static_cast

№	Задание	Требования
3	Перевести число из секунд в формат «дни:часы:минуты:секунды».	Целочисленная арифметика, int
4	Вычислить корни квадратного уравнения. Обработать все случаи.	double для всех переменных
5	Вычислить значение многочлена 4-й степени. Сравнить с float.	Демонстрация накопления погрешности
6	Перевести угол из градусов в радианы и обратно. Показать неявное приведение.	const double PI
7	По координатам трёх точек вычислить площадь треугольника (через определитель).	Все вычисления в double
8	Вычислить sin, cos, tan для угла в градусах. Показать преобразование в радианы.	угол * PI / 180.0
9	Вычислить сумму, разность, произведение и частное. Сравнить int и double.	Отдельно продемонстрировать оба варианта

№	Задание	Требования
10	Преобразовать число: найти сумму и произведение цифр.	Явное приведение при извлечении цифр
11	Вычислить x^y (y — целое положительное). Сравнить std::pow и цикл.	Показать приведение типов в pow
12	Вычислить ИМТ и вывести категорию.	double для веса/роста, static_cast
13	По трём числам вывести максимальное и минимальное. Использовать тернарный оператор.	Сравнение разных типов
14	Вычислить сложные проценты. Показать накопление ошибки float vs double.	Демонстрация через 1000 итераций
15	«Анализатор точности»: ввести число, показать его в int, float, double.	Показать потерю данных при каждом приведении

5. Контрольные вопросы

1. Что такое приведение типов? Дайте определение.
2. Чем отличается явное приведение от неявного? Приведите примеры.

3. Когда происходит неявное приведение? Назовите 3 ситуации.
4. Почему double result = 5 / 2; даёт 2.0, а не 2.5? Объясните механизм.
5. Как исправить целочисленное деление с помощью явного приведения? Покажите код.
6. Что такое static_cast? Чем он лучше C-style (int)?
7. Что произойдёт при int x = 3.14;? Какие данные потеряются?
8. Почему нельзя использовать % с double? Как обойти это ограничение?
9. Что такое усечение (truncation) и округление (rounding)? В чём разница?
10. Что такое потеря точности (precision loss)? Приведите пример с float и double.
11. Что такое переполнение (overflow)? Что произойдёт с int max + 1?
12. Базовые понятия (типы данных, переменные)
13. Что такое переменная в C++? Какие этапы проходит программа от написания кода до исполнения?
14. Какие правила именования переменных существуют? Приведите примеры правильных и неправильных имён.
15. Какие базовые типы данных вы знаете? Заполните таблицу:

Тип	**Размер (байт)**	**Диапазон**	**Когда использовать**
int			
double			

Тип	Размер (байт)	Диапазон	Когда использовать
float			
char			
bool			

16. В чём разница между int и unsigned int? Когда стоит использовать unsigned?
17. Какой тип данных вы выберете для хранения: количества студентов в группе, цены товара в рублях, роста человека в метрах, суммы транзакции в банке? Почему?
18. Дайте определение приведению типов. Чем отличается явное приведение от неявного?
19. Приведите 3 примера неявного приведения типов (с кодом). Объясните, что произойдёт в каждом случае.
20. Что такое static_cast? Чем он лучше C-style приведения (int)? Напишите пример.
21. Почему код double result = 5 / 2; даёт 2.0, а не 2.5? Объясните по шагам, какие преобразования выполняются.
22. Как исправить целочисленное деление? Покажите три разных способа получения вещественного результата.
23. Что произойдёт при выполнении кода:

```
int a = 3.14;
double b = 5;
```

```
char c = 65;
int d = c;
```

24. Объясните результат каждого преобразования.
25. Что такое потеря точности (precision loss)? Приведите пример с float и double.
26. Что такое усечение (truncation) и округление (rounding)? В чём разница? Как в C++ выполнить математическое округление?
27. Что такое переполнение (overflow)? Что произойдёт, если к максимальному int прибавить 1?
28. Почему нельзя использовать оператор % с double? Как получить остаток от деления для вещественных чисел?
29. Какие арифметические операции существуют в C++? Назовите приоритет и ассоциативность.
30. Чем отличается / для целых и вещественных чисел? Приведите пример.
31. Что выведет следующий код и почему?

```
int x = 10, y = 3;
double z = x / y;
std: cout << z;
```

32. Как правильно вычислить среднее арифметическое трёх целых чисел? Напишите код.
33. Что такое магические числа? Почему их следует заменять именованными константами?
34. Раздел Г. Ввод/вывод и форматирование
35. Что такое std::cout и std::cin? Какие операторы используются для вывода и ввода?
36. В чём разница между std::endl и \n? Когда какой использовать?
37. Как вывести число с точностью 3 знака после запятой? Какой заголовочный файл нужен?

38. Что делает std::fixed в связке с std::setprecision?
39. Как вывести несколько значений в одном std::cout? Покажите пример.
40. Найдите ошибки в коде и объясните, как их исправить:

```
#include <iostream>
main() {
    int a, b;
    cout << "Введите числа: ";
    cin >> a, b;
    double c = a / b;
    cout << "Результат: " + c;
    return 0;
}
```

41. Что выведет программа?

```
double a = 0.1 + 0.2;
if (a == 0.3) {
    std: cout << "Равны";
} else {
    std: cout << "Не равны";
}
```

42. Почему так происходит? Как правильно сравнивать double?
43. Какой тип данных имеет литерал 3.14? А 3.14f? А 42L? А 42U?
44. Что такое const? Зачем нужны константы? Напишите пример.
45. Что такое static_assert? Как он может помочь при работе с типами данных?

7. Критерии оценивания

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) - правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, отсутствие фатальных ошибок, программа запускается и выводит результат).

Оценка «хорошо» (4) - правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего уровня (использование static_cast для явного приведения, корректная работа с вещественными типами, осмысленные имена переменных, форматирование вывода).

Оценка «отлично» (5) - правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (демонстрация потери точности между float и double, обработка ошибочного ввода через std::cin.fail(), проверка граничных значений, подробные комментарии с объяснением выбора типа и необходимости приведения, сравнение разных способов приведения).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

№	**Критерий**	**Базовый (3)**	**Средний (4)**	**Повышенный (5)**
1.	Выполненные уровни заданий	Т	Б	Базо

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
		олько базовый	азовый + средний	вый + средний + повышенный
Компиляция и запуск				
2.	Программа компилируется без ошибок	+	+	+
3.	Программа успешно запускается и не аварийно завершается	+	+	+
4.	Отсутствие предупреждений компилятора (уровень /W3 или эквивалент)	—	+	+
Работа с вводом/выводом				
5.	Использование #include <iostream>	+	+	+
6.	Корректное использование std::cout для вывода	+	+	+
7.	Корректное	+	+	+

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	использование std::cin для ввода			
8.	Вывод приглашений для пользователя перед вводом	—	+	+
9.	Использование #include <iomanip> для форматирования	—	+	+
Переменные и типы данных				
10.	Правильное объявление переменных	+	+	+
11.	Правильный выбор типа данных (int, double, char, bool)	+	+	+
12.	Инициализация переменных перед использованием	+	+	+
13.	Использование осмысленных имён переменных (не a, b, c без пояснений)	—	+	+
14.	Использование const для констант (π, коэффициенты)	—	+	+
Арифметические операции и приведение типов				

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
15.	Корректное выполнение арифметических операций	+	+	+
16.	Правильный порядок операций (использование скобок)	+	+	+
17.	Корректная работа с вещественными числами (9.0/5.0, а не 9/5)	+	+	+
18.	Использование static_cast для явного приведения типов	—	+	+
19.	Демонстрация разницы между целочисленным и вещественным делением	—	+	+
20.	Демонстрация потери точности между float и double	—	—	+
21.	Демонстрация усечения (truncation) vs округления (rounding)	—	—	+
Форматирование вывода				
22.	Вывод результата с поясняющим текстом	+	+	+

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
23.	Использование std::fixed и std::setprecision для точности	—	+	+
24.	Использование std::setw для выравнивания (таблицы)	—	—	+
Обработка ошибок				
25.	Проверка деления на ноль (в заданиях с делением)	—	+	+
26.	Проверка на ввод некорректных данных (std::cin.fail())	—	—	+
27.	Очистка буфера ввода при ошибке (std::cin.clear(), std::cin.ignore())	—	—	+
28.	Вывод понятного сообщения об ошибке вместо аварийного завершения	—	—	+
Качество кода				
29.	Правильные отступы и форматирование кода	+	+	+

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
30.	Отсутствие «магических чисел» (константы вынесены в const)	—	+	+
31.	Наличие комментариев в коде (объяснение выбора типа)	—	+	+
32.	Детальные комментарии с объяснением явного приведения	—	—	+
33.	Сравнение разных способов приведения в одном коде	—	—	+
Отчёт				
34.	Наличие титульного листа, цели, хода работы	+	+	+
35.	Скриншоты результатов выполнения (минимум 2 теста)	+	+	+
36.	Скриншоты для всех трёх уровней	—	+	+
37.	Таблица тестирования (вход → ожидаемый → фактический результат)	—	+	+

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
38.	Тестирование на граничных значениях (0, отрицательные, очень большие числа)	—	—	+
39.	Сравнительная таблица float vs double (потеря точности)	—	—	+
40.	Ответы на контрольные вопросы (письменно, развёрнуто)	—	+	

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) — обязательные требования

Для получения оценки «удовлетворительно» студент должен:

- Выполнить один любой вариант из базового уровня (№1–15).
- Написать программу, которая:
- Корректно запрашивает входные данные
- Выполняет простое арифметическое вычисление
- Выводит результат на экран
- Продемонстрировать понимание:
- Разницы между int и double (хотя бы на уровне выбора типа)
- Использования std::cout и std::cin
- Предоставить отчёт с:
- Титульным листом
- Целью работы

- Листингом кода
- Скриншотами выполнения (минимум 2 набора входных данных)

Средний уровень (оценка 4) — дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

- Использовать static_cast для явного приведения типов (демонстрация в коде).
- Показать разницу между целочисленным и вещественным делением (например, через отдельные переменные).
- Выводить приглашения перед каждым вводом данных.
- Форматировать вывод с помощью std::fixed и std::setprecision.
- Использовать const для констант (π, коэффициенты перевода).
- Использовать осмысленные имена переменных (radius, area, pricePerKg, а не r, a, p).
- В отчёте представить:
- Скриншоты для обоих уровней
- Таблицу тестирования
- Ответы на контрольные вопросы (разделы А, Б, В)

Повышенный уровень (оценка 5) — требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

- Демонстрация потери точности:
- Сравнить результат вычислений с float и double для одной и той же формулы
- Показать, как накапливается погрешность (например, при сложении 0.1 много раз)
- Представить сравнительную таблицу в отчёте
- Демонстрация усечения vs округления:

- Показать разницу между static_cast<int>(value) и static_cast<int>(std::round(value))
- Объяснить в комментариях, когда что использовать
- Обработка ошибок ввода:
- Проверка std::cin.fail() после ввода
- Очистка буфера std::cin.clear() и std::cin.ignore()
- Повторный запрос ввода при ошибке (опционально, но приветствуется)
- Тестирование на граничных значениях:
- Ноль
- Отрицательные числа
- Максимальные/минимальные значения (INT_MAX, INT_MIN)
- Дробные числа с большим количеством знаков
- Детальные комментарии в коде:
- Почему выбран тот или иной тип данных
- Зачем нужно явное приведение типов (что будет без него)
- Где может произойти потеря точности
- В отчёте дополнительно представить:
- Сравнительную таблицу float vs double
- Таблицу тестирования на граничных значениях
- Блок-схему алгоритма (для повышенного задания)

Лабораторная работа №3

Ветвления. Логические операторы

1. Цель работы

Научиться использовать условный оператор if-else для реализации ветвлений в программах, применять логические операторы (&&, ||, !) для построения сложных условий, реализовывать защиту от некорректного ввода

(деление на ноль, проверка диапазонов), создавать простое консольное меню с выбором действия.

2. Теоретическое обоснование

2.1. Условный оператор if

2.1.1. Простейшая форма if

```
if (условие) {
    // выполняется, если условие истинно
}
```

Условие — это выражение, которое принимает значение true (истина) или false (ложь). В C++ любое ненулевое значение интерпретируется как true, нулевое — как false.

```
int x = 10;
if (x > 5) {
    std::cout << "x больше 5" << std::endl;
}
```

2.1.2. Форма if-else

```
if (условие) {
    // выполняется, если условие истинно
} else {
    // выполняется, если условие ложно
}
```

```
int age;
std::cin >> age;
if (age >= 18) {
    std::cout << "Доступ разрешён" << std::endl;
} else {
    std::cout << "Доступ запрещён" << std::endl;
}
```

2.1.3. Конструкция if-else if-else (множественное ветвление)

```
if (условие1) {
    // блок 1
} else if (условие2) {
    // блок 2
} else if (условие3) {
    // блок 3
} else {
    // блок по умолчанию (если ни одно условие не подошло)
}

int score;
std::cin >> score;

if (score >= 90) {
    std::cout << "Отлично (5)" << std::endl;
} else if (score >= 75) {
    std::cout << "Хорошо (4)" << std::endl;
} else if (score >= 60) {
    std::cout << "Удовлетворительно (3)" << std::endl;
} else {
    std::cout << "Неудовлетворительно (2)" << std::endl;
}
```

Важно**:** Условия проверяются последовательно. Как только найдено истинное условие, остальные else if и else пропускаются.

2.1.4. Вложенные if

```
if (условие1) {
    if (условие2) {
        // выполняется, если оба условия истинны
    }
```

```
    }
if (x > 0) {
    if (x % 2 == 0) {
        std::cout << "Положительное чётное число" << std::endl;
    }
}
```

2.2. Логические операторы

Логические операторы позволяют объединять простые условия в сложные.

Оператор	Название	Значение	Пример
&&	Логическое И	true только если оба операнда true	(a > 0 && b > 0)
||	Логическое ИЛИ	true если хотя бы один операнд true	(a == 0 || b == 0)
!	Логическое НЕ	Инвертирует значение	!(x < 0) (означает x >= 0)

2.2.1. Таблица истинности

a	b	a && b	a || b	a
false	false	false	false	true

a	b	a && b	a || b	a
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

2.2.2. Примеры сложных условий

```
// Проверка, что число x принадлежит интервалу [10, 20]
if (x >= 10 && x <= 20) {
    std::cout << "x в диапазоне [10,20]" << std::endl;
}

// Проверка, что год високосный (делится на 4, но не на 100, или делится на 400)
if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {
    std::cout << "Високосный год" << std::endl;
}

// Проверка, что символ является гласной буквой
char c;
std::cin >> c;
if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' ||
    c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U') {
    std::cout << "Гласная" << std::endl;
```

```
}
```

2.2.3. Ленивые вычисления (short-circuit evaluation)

C++ вычисляет логические выражения лениво — слева направо, останавливаясь, когда результат уже ясен.

```
// Если a == 0, то выражение (a == 0) истинно, и проверка (b / a) НЕ выполняется
if (a == 0 || b / a > 10) {
    // Безопасно, деления на ноль не произойдёт
}

// Аналогично для &&
if (ptr != nullptr && ptr->value > 0) {
    // Безопасно, разыменование только если ptr не нулевой
}
```

Это свойство критически важно для защиты от ошибок!

2.3. Сравнение чисел с плавающей точкой

Важнейшая проблема: Вещественные числа не следует сравнивать на точное равенство из-за погрешностей представления.

```
double a = 0.1 + 0.2;  // может быть не точно 0.3!
double b = 0.3;

if (a == b) {  // ОПАСНО! Может не выполниться
    std::cout << "Равны" << std::endl;
}
```

Правильный способ — сравнение с допуском (epsilon):

```
#include <cmath>
```

```
const double EPSILON = 1e-9;  // 10^-9

if (std::fabs(a - b) < EPSILON) {
    std::cout << "Практически равны" << std::endl;
}
```

2.4. Защита от некорректного ввода

2.4.1. Проверка деления на ноль

```
double a, b;
std::cin >> a >> b;

if (b == 0) {
    std::cout << "Ошибка: деление на ноль невозможно!" << std::endl;
} else {
    double result = a / b;
    std::cout << "Результат: " << result << std::endl;
}
```

2.4.2. Проверка на ввод чисел (а не букв)

```
int x;
std::cin >> x;

if (std::cin.fail()) {
    std::cout << "Ошибка: введите число!" << std::endl;
    std::cin.clear();  // сбрасываем флаг ошибки
    std::cin.ignore(32767, '\n');  // очищаем буфер ввода
} else {
    std::cout << "Вы ввели: " << x << std::endl;
```

```
}
```

2.4.3. Проверка диапазона

```
int month;
std::cin >> month;

if (month < 1 || month > 12) {
    std::cout << "Ошибка: месяц должен быть от 1 до 12!" << std::endl;
} else {
    std::cout << "Правильный месяц: " << month << std::endl;
}
```

2.5. Тернарный оператор ? :

Тернарный оператор — сокращённая форма if-else, возвращающая значение.

```
// Синтаксис: условие ? значение_если_истина : значение_если_ложь

int absValue = (x >= 0) ? x : -x;

int max = (a > b) ? a : b;

// Эквивалентно:
int max;
if (a > b) {
    max = a;
} else {
    max = b;
}
```

Используйте тернарный оператор только для простых случаев! Злоупотребление снижает читаемость.

2.6. Логические типы и их использование

```
bool isEven = (x % 2 == 0);
bool isValid = (age >= 18 && age <= 65);

// Можно использовать в if напрямую
if (isEven) {
    std::cout << "Чётное" << std::endl;
}

// Упрощённая запись
if (!isValid) {
    std::cout << "Значение вне диапазона" << std::endl;
}
```

2.7. Консольное меню

Реализация простого текстового меню с выбором действия:

```
#include <iostream>

int main() {
    int choice;

    std::cout << "===== КАЛЬКУЛЯТОР =====" << std::endl;
    std::cout << "1. Сложение" << std::endl;
    std::cout << "2. Вычитание" << std::endl;
    std::cout << "3. Умножение" << std::endl;
    std::cout << "4. Деление" << std::endl;
```

```
    std::cout << "0. Выход" << std::endl;
    std::cout << "Выберите операцию: ";
    std::cin >> choice;

    if (choice == 1) {
        // сложение
    } else if (choice == 2) {
        // вычитание
    } else if (choice == 3) {
        // умножение
    } else if (choice == 4) {
        // деление с проверкой
    } else if (choice == 0) {
        std::cout << "До свидания!" << std::endl;
    } else {
        std::cout << "Неверный выбор!" << std::endl;
    }

    return 0;
}
```

2.8. Типичные ошибки при использовании if

Ошибка	Пример	Последствие
Присваивание вместо сравнения	if (x = 5)	Всегда истинно, x становится 5

Ошибка	Пример	Последствие
Пропущены фигурные скобки	if (x > 0) x++; y++;	y++ выполняется всегда
; после условия	if (x > 0); { ... }	Тело выполняется всегда
Сравнение == с double	if (d == 0.3)	Может не сработать
Неправильная логика И/ИЛИ	if (x > 0 && < 10)	Синтаксическая ошибка

```
// Правильно: if (x > 0 && x < 10)

// ОШИБКА: if (x > 0 && < 10)
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Проверка деления на ноль

```
#include <iostream>

int main() {
    double a, b;

    std::cout << "Введите делимое и делитель: ";
    std::cin >> a >> b;
```

```
    if (b == 0) {
        std::cout << "ОШИБКА: На ноль делить нельзя!" << std::endl;
    } else {
        double result = a / b;
        std::cout << a << " / " << b << " = " << result << std::endl;
    }

    return 0;
}
```

Разбор:

Условие b == 0 проверяет ввод пользователя

Блок else выполняется только при корректном вводе

Программа не падает, а выводит понятное сообщение

3.2. Пример 2: Определение времени года по номеру месяца

```
#include <iostream>

int main() {
    int month;

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

    if (month < 1 || month > 12) {
        std::cout << "Ошибка: месяц должен быть от 1 до 12!" << std::endl;
    } else if (month == 12 || month == 1 || month == 2) {
        std::cout << "Зима" << std::endl;
    } else if (month >= 3 && month <= 5) {
        std::cout << "Весна" << std::endl;
```

```
    } else if (month >= 6 && month <= 8) {
        std::cout << "Лето" << std::endl;
    } else {  // month >= 9 && month <= 11
        std::cout << "Осень" << std::endl;
    }

    return 0;
}
```

Разбор

Сначала проверяется корректность ввода

Для зимы — особое условие (декабрь, январь, февраль)

Для остальных сезонов — проверка диапазона

3.3. Пример 3: Простой консольный калькулятор

```
#include <iostream>

int main() {
    double a, b;
    char op;

    std::cout << "Введите выражение (например, 5 + 3): ";
    std::cin >> a >> op >> b;

    if (op == '+') {
        std::cout << a << " + " << b << " = " << a + b << std::endl;
    } else if (op == '-') {
        std::cout << a << " - " << b << " = " << a - b << std::endl;
    } else if (op == '*') {
        std::cout << a << " * " << b << " = " << a * b << std::endl;
    } else if (op == '/') {
```

```
            if (b == 0) {
                std::cout << "Ошибка: деление на ноль!" << std::endl;
            } else {
                std::cout << a << " / " << b << " = " << a / b << std::endl;
            }
        } else {
            std::cout << "Ошибка: неизвестная операция '" << op << "'" <<
std::endl;
        }

        return 0;
    }
```

Разбор:

Вложенный if внутри ветки деления для проверки делителя

Обработка неизвестной операции через else

Читаемый вывод с повторением введённых данных

3.4. Пример 4: Решение квадратного уравнения

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

int main() {
    double a, b, c;

    std::cout << "Решение квадратного уравнения ax^2 + bx + c = 0" <<
std::endl;
    std::cout << "Введите a, b, c: ";
    std::cin >> a >> b >> c;

    if (a == 0) {
```

```
            // Линейное уравнение bx + c = 0
            if (b == 0) {
                if (c == 0) {
                    std::cout << "Бесконечное множество решений" << std::endl;
                } else {
                    std::cout << "Нет решений" << std::endl;
                }
            } else {
                double x = -c / b;
                std::cout << "Линейное уравнение, корень: x = " << x << std::endl;
            }
        } else {
            double d = b * b - 4 * a * c;  // дискриминант

            if (d < 0) {
                std::cout << "Действительных корней нет" << std::endl;
            } else if (d == 0) {
                double x = -b / (2 * a);
                std::cout << "Один корень (кратный): x = " << x << std::endl;
            } else {
                double x1 = (-b + sqrt(d)) / (2 * a);
                double x2 = (-b - sqrt(d)) / (2 * a);
                std::cout << "Два корня: x1 = " << x1 << ", x2 = " << x2 <<
std::endl;
            }
        }

        return 0;
    }
```

Разбор:

Множественное ветвление для различных случаев

Вложенные if для обработки граничных условий

Проверка a == 0 (не квадратное уравнение) перед вычислением дискриминанта

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Простое ветвление с одним условием.

№	Задание
1	Запросить число. Если оно положительное, вывести "Положительное", иначе "Неположительное"
2	Запросить два числа. Вывести большее из них
3	Запросить число. Если оно чётное, вывести "Чётное", иначе "Нечётное"
4	Запросить возраст. Вывести "Совершеннолетний", если возраст ≥ 18, иначе "Несовершеннолетний"
5	Запросить число. Если оно делится на 5, вывести "Кратно 5", иначе вывести его остаток от деления на 5
6	Запросить число. Если оно трёхзначное, вывести "Трёхзначное", иначе "Не трёхзначное"
7	Запросить число. Если оно больше нуля, вывести его квадрат, иначе вывести "Некорректный ввод"

№	Задание
8	Запросить два числа. Если они равны, вывести "Равны", иначе вывести "Не равны"
9	Запросить число. Если оно отрицательное, вывести модуль, иначе вывести само число
10	Запросить символ. Если это цифра ('0'-'9'), вывести "Цифра", иначе "Не цифра"
11	Запросить три числа. Если их сумма больше 100, вывести "Большая сумма", иначе вывести сумму
12	Запросить число. Если оно находится в диапазоне [10, 20], вывести "В диапазоне", иначе вывести "Вне диапазона"
13	Запросить число. Если оно равно 0, вывести "Ноль", иначе вывести "Не ноль"
14	Запросить число. Если оно кратно 3 и 5, вывести "Кратно 15", иначе вывести "Не кратно 15"
15	Запросить температуру. Если температура ниже 0, вывести "Мороз", иначе вывести "Тепло"

Средний уровень (оценка «хорошо»)

Множественное ветвление, сложные условия с логическими операторами.

№	Задание
1	Запросить три числа. Найти и вывести наибольшее и наименьшее
2	Запросить номер месяца (1-12). Вывести название месяца и количество дней в нём (без учёта високосности)
3	Запросить три стороны треугольника. Проверить, существует ли такой треугольник (каждая сторона меньше суммы двух других). Вывести "Существует" или "Не существует"
4	Запросить координаты точки (x, y). Определить, в какой координатной четверти она находится (I, II, III, IV или на оси)
5	Запросить число. Вывести его словесное описание: "отрицательное", "положительное" или "ноль", а также "чётное" или "нечётное" (например: "положительное чётное")
6	Калькулятор: запросить два числа и символ операции (+, -, *, /). Вывести результат с проверкой деления на ноль
7	Запросить год. Определить, является ли он високосным (делится на 4, но не на 100, или делится на 400)
8	Запросить число от 0 до 100 (оценку). Вывести буквенную оценку: A (90+), B (75-89), C (60-74), D (50-59),

№	Задание
	F (ниже 50). Если число вне диапазона — ошибка
9	Запросить три числа. Вывести их в порядке возрастания
10	Запросить символ. Определить, является ли он: гласной буквой, согласной буквой, цифрой, другим символом
11	Запросить число. Если оно положительное, вывести его квадратный корень (если можно), иначе вывести его квадрат
12	Простой конвертер единиц: запросить значение и единицу (cm, m, km). Перевести в метры и вывести результат
13	Запросить число от 1 до 7 (день недели). Вывести название дня и указать, рабочий это день или выходной
14	Запросить возраст и пол (M/F). Вывести: для мужчин "юноша" (до 18), "мужчина" (18+); для женщин "девушка" (до 18), "женщина" (18+)
15	Запросить число. Определить, является ли оно палиндромом (трёхзначное число: abc равно cba)

Повышенный уровень (оценка «отлично»)

Многоуровневые ветвления, обработка ошибок ввода, полная защита от некорректных данных.

№	Задание
1	Расширенный калькулятор: поддержка +, -, *, /, ^ (возведение в степень), % (остаток). Полная обработка ошибок (деление на ноль, отрицательный показатель для целых, неизвестная операция, некорректный ввод)
2	Определитель возраста: запросить год рождения и текущий год. Вывести возраст и категорию: ребёнок (0-12), подросток (13-17), взрослый (18-64), пожилой (65+). Проверка на корректность (год не в будущем, разумный возраст)
3	Игра "Угадай число": программа загадывает целое число от 1 до 10. Пользователь вводит свою догадку. Программа выводит "Угадал!", "Меньше" или "Больше". Реализовать проверку ввода (только числа, только 1-10)
4	Калькулятор ИМТ: запросить рост (см) и вес (кг). Вычислить ИМТ = вес / (рост_в_м)². Вывести категорию: дефицит (<18.5), норма (18.5-25), избыток (25-30), ожирение (30+). Проверка положительности ввода
5	Программа-светофор: запросить цвет (красный, жёлтый, зелёный) на русском или английском. Вывести действие: "Стоять", "Приготовиться", "Идти". При неверном вводе — повторить запрос (или вывести ошибку)
6	Конвертер систем счисления: запросить число от 0 до 255 и выбрать систему (2, 8, 10, 16). Вывести число в

№	Задание
	выбранной системе (использовать встроенные средства или написать свою логику). Проверка диапазона
7	Определение знака зодиака: запросить день и месяц рождения. Вывести знак зодиака и его описание. Полная проверка корректности даты (учёт дней в месяцах, февраль до 29)
8	Квадратное уравнение с комплексными корнями: решить квадратное уравнение. Если дискриминант отрицательный, вывести комплексные корни в формате "x1 = a + bi, x2 = a - bi". Обработать случай a=0, b=0
9	Система скидок: запросить сумму покупки и статус клиента (обычный, VIP, сотрудник). Скидка: сотрудник — 20%, VIP — 10%, обычный — 0% + дополнительная скидка 5% при сумме > 5000. Вывести итоговую сумму
10	Программа-экзаменатор: запросить 3 числа — стороны треугольника. Определить тип: равносторонний, равнобедренный, прямоугольный (проверить теорему Пифагора), разносторонний. Если треугольник не существует — сообщить
11	Логическая задача: запросить три числа (a, b, c). Проверить, могут ли они быть длинами сторон прямоугольного треугольника (упорядочить, затем проверить $a^2 + b^2 = c^2$ с точностью EPSILON)

№	Задание
2 1	Банкомат: запросить сумму (кратную 100). Вывести минимальное количество купюр (5000, 2000, 1000, 500, 200, 100). Проверить, что сумма не превышает лимит (100000) и положительная
3 1	Калькулятор доставки: запросить расстояние (км), вес (кг), срочность (обычная, быстрая). Базовая стоимость: 50 руб + 10 руб/км + 20 руб/кг. Быстрая доставка: +50%. Вывести итоговую стоимость. Проверка положительности
4 1	Билетная система: запросить возраст и день недели (1-7). Цена билета: дети <7 — бесплатно, 7-18 — 200 руб, 18-65 — 500 руб, >65 — 250 руб. В выходной (6-7) +20%. Вывести итоговую цену
5 1	Лотерея: программа загадывает 3 случайных числа (1-9). Пользователь вводит 3 числа. Вычислить: 3 совпадения на своих местах — джекпот, 3 совпадения (в любом порядке) — суперприз, 2 совпадения — приз, 1 — утешительный, 0 — ничего. Вывести загаданную комбинацию и результат

5. Контрольные вопросы

1. Чем отличается = от == в C++?
2. Что произойдёт, если в условии if (x = 5) написать присваивание?
3. Как проверить, что число находится в интервале [10, 20]?
4. В чём разница между && и ||? Приведите примеры.

5. Что такое "ленивые вычисления" (short-circuit evaluation) и как их использовать для защиты?
6. Почему нельзя сравнивать double через ==? Как правильно их сравнивать?
7. Что делает std::cin.fail() и std::cin.clear()?
8. Зачем нужны фигурные скобки {} в if даже для одной строки?
9. Какой результат даст выражение: int x = 5; if (x) std::cout << "True";?
10. Что такое тернарный оператор? Приведите пример.
11. Как организовать меню с несколькими вариантами выбора?
12. В чём преимущество else if перед несколькими отдельными if?
13. Что выведет код: if (0) std::cout << "A"; std::cout << "B";?
14. Как проверить, что введённое значение является числом, а не строкой?
15. Что такое вложенные if? Какие проблемы могут возникнуть при их использовании (dangling else)?

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (условный оператор, логические операторы, защита от некорректного ввода)

Ход выполнения работы:

Код программы с **подробными комментариями**

Блок-схема алгоритма (обязательно для этой работы)

Скриншоты выполнения для **различных сценариев** (минимум: корректный ввод, некорректный ввод, граничные значения)

Для повышенного уровня — таблица тестирования всех веток

Ответы на контрольные вопросы (развёрнутые, с примерами кода)

Вывод (что нового узнали о ветвлении и логических операторах)

Формат: .docx или .pdf.

7. Критерии оценивания

Критерий	Базовый	Средний	Повышенный
Программа компилируется без ошибок	✔	✔	✔
Корректная работа условных операторов	✔	✔	✔
Правильное использование логических операторов	✔	✔	✔
Проверка деления на ноль (где применимо)	—	✔	✔
Проверка диапазона ввода	—	✔	✔
Обработка неверного ввода (ошибочные символы)	—	—	✔
Наличие блока else для всех веток	—	✔	✔
Читаемость кода (отступы, имена)	✔	✔	✔
Блок-схема алгоритма в	—	✔	✔

Критерий	Базовый	Средний	Повышенный
отчёте			
Тестирование на граничных значениях	—	—	✔
Отсутствие магических чисел (использование констант)	—	✔	✔
Наличие комментариев в коде	—	✔	✔

8. Пример эталонного выполнения (для задания повышенного уровня №1 — расширенный калькулятор)

```cpp
#include <iostream>
#include <cmath>
#include <limits>  // для numeric_limits

int main() {
    double a, b, result;
    char op;
    bool validInput = true;

    std::cout << "===== РАСШИРЕННЫЙ КАЛЬКУЛЯТОР =====" << std::endl;
    std::cout << "Поддерживаемые операции: +, -, *, /, ^, %" << std::endl;
    std::cout << "Введите выражение (например, 5.5 ^ 2): ";
```

```
        std::cin >> a >> op >> b;

        // Проверка на ошибку ввода
        if (std::cin.fail()) {
            std::cout << "ОШИБКА: Некорректный ввод! Используйте формат: число операция число" << std::endl;
            return 1;
        }

        // Выполнение операции
        if (op == '+') {
            result = a + b;
            std::cout << a << " + " << b << " = " << result << std::endl;
        } else if (op == '-') {
            result = a - b;
            std::cout << a << " - " << b << " = " << result << std::endl;
        } else if (op == '*') {
            result = a * b;
            std::cout << a << " * " << b << " = " << result << std::endl;
        } else if (op == '/') {
            if (std::fabs(b) < 1e-9) {  // проверка на близость к нулю
                std::cout << "ОШИБКА: Деление на ноль невозможно!" << std::endl;
                validInput = false;
            } else {
                result = a / b;
                std::cout << a << " / " << b << " = " << result << std::endl;
            }
        } else if (op == '^') {
```

```
            result = std::pow(a, b);
            std::cout << a << " ^ " << b << " = " << result << std::endl;
        } else if (op == '%') {
            // Проверка, что оба операнда целые (с допуском)
            if (std::fabs(a - std::round(a)) > 1e-9 || std::fabs(b - std::round(b)) > 1e-
9) {
                std::cout << "ОШИБКА: Оператор % требует целые числа!" <<
std::endl;
                validInput = false;
            } else if (std::fabs(b) < 1e-9) {
                std::cout << "ОШИБКА: Деление на ноль в операции %!" <<
std::endl;
                validInput = false;
            } else {
                int intA = static_cast<int>(std::round(a));
                int intB = static_cast<int>(std::round(b));
                result = intA % intB;
                std::cout << intA << " % " << intB << " = " << result << std::endl;
            }
        } else {
            std::cout << "ОШИБКА: Неизвестная операция '" << op << "'" <<
std::endl;
            validInput = false;
        }

        // Если была ошибка, завершаем с ненулевым кодом
        if (!validInput) {
            return 1;
        }
```

```
    return 0;
}
```

Тестирование:

Вход	Ожидаемый вывод
`10 / 0`	ОШИБКА: Деление на ноль невозможно!
`5 + 3`	5 + 3

Лабораторная работа №4

Множественный выбор (switch)

1. Цель работы

Освоить оператор множественного выбора switch-case, научиться применять его для реализации меню и выбора действий на основе дискретных значений, обрабатывать значение default, использовать группировку case для объединения нескольких вариантов, понимать разницу между switch и if-else if-else, предотвращать ошибки fall-through с помощью break.

2. Теоретическое обоснование

2.1. Общая концепция оператора switch

Оператор switch (переключатель) предназначен для выбора одного из нескольких вариантов выполнения в зависимости от значения целочисленного выражения. Он является альтернативой длинной цепочке if-else if-else, когда проверяется равенство одной переменной множеству возможных значений.

Синтаксис:

```
switch (выражение) {
    case константа1:
        // блок кода для константы1
        break;
    case константа2:
        // блок кода для константы2
        break;
    // ... другие case
    default:
        // блок кода, если ни одно значение не подошло
        break;
}
```

Ключевые требования:

1. выражение в switch должно быть целочисленного типа (int, char, short, long, bool, enum)
2. константа после case должна быть константным выражением, известным на этапе компиляции
3. Нельзя использовать float, double, string в switch (для них нужен if-else)

```
// ПРАВИЛЬНО: целочисленные типы
int choice = 2;
switch (choice) { ... }

char grade = 'A';
switch (grade) { ... }

// НЕПРАВИЛЬНО: вещественные типы
double d = 1.5;
```

```
switch (d) { ... }  // ОШИБКА!

// НЕПРАВИЛЬНО: строки
std: string name = "start";
switch (name) { ... }  // ОШИБКА!
```

2.2. Механизм работы switch

Вычисляется значение выражение в скобках.

Это значение последовательно сравнивается с константами после case.

При первом совпадении управление передаётся в соответствующий блок кода.

Если совпадений не найдено, управление передаётся в блок default (если он есть).

После выполнения блока case выполнение продолжается дальше, если не встречен break.

Схема работы:

text

```
 switch (x)
     |
     v
x == 1?  ---Да---> [блок case 1] ---> break ---> выход
     |
     Нет
     v
x == 2?  ---Да---> [блок case 2] ---> break ---> выход
     |
     Нет
     v
x == 3?  ---Да---> [блок case 3] ---> break ---> выход
```

|

Нет

v

[блок default] ---> выход

2.3. Ключевое слово break

break вызывает немедленный выход из блока switch. Без break происходит проваливание (fall-through) — выполнение продолжается со следующего case, даже если его константа не совпадает.

```
// Пример fall-through (без break)
int day = 2;
switch (day) {
    case 1:
        std::cout << "Понедельник";
    case 2:
        std::cout << "Вторник";  // Выполнится!
    case 3:
        std::cout << "Среда";    // Выполнится!
    default:
        std::cout << "Другой день";
}
// Вывод: "ВторникСредаДругой день"
```

Правильное использование (с break):

cpp

```
int day = 2;
switch (day) {
    case 1:
        std::cout << "Понедельник";
```

```
        break;
    case 2:
        std::cout << "Вторник";
        break;  // Выход из switch
    case 3:
        std::cout << "Среда";
        break;
    default:
        std::cout << "Другой день";
        break;
}
// Вывод: "Вторник"
```

Важно: Забытый break — одна из самых распространённых ошибок у начинающих!

2.4. Группировка case (несколько значений для одного действия)

Если несколько значений должны выполнять одинаковый код, их можно записать подряд без break между ними:

```
char grade;
std::cin >> grade;

switch (grade) {
    case 'A':
    case 'B':
    case 'C':
        std::cout << "Удовлетворительно или выше" << std::endl;
        break;
    case 'D':
```

```
    case 'E':
        std::cout << "Нужно подтянуть" << std::endl;
        break;
    case 'F':
        std::cout << "Неудовлетворительно" << std::endl;
        break;
    default:
        std::cout << "Неверная оценка" << std::endl;
        break;
}
```

Группировка для проверки гласных букв:

```
char c;
std::cin >> c;

switch (c) {
    case 'a': case 'e': case 'i': case 'o': case 'u':
    case 'A': case 'E': case 'I': case 'O': case 'U':
        std::cout << "Гласная буква" << std::endl;
        break;
    default:
        std::cout << "Согласная или другой символ" << std::endl;
        break;
}
```

2.5. Блок default

default выполняется, если ни одно значение case не совпало. Он не обязателен, но настоятельно рекомендуется для обработки непредвиденных значений.

```
int month;
std::cin >> month;

switch (month) {
    case 12:
    case 1:
    case 2:
        std::cout << "Зима";
        break;
    case 3: case 4: case 5:
        std::cout << "Весна";
        break;
    case 6: case 7: case 8:
        std::cout << "Лето";
        break;
    case 9: case 10: case 11:
        std::cout << "Осень";
        break;
    default:
        std::cout << "Ошибка: месяц должен быть от 1 до 12";
        break;
}
```

Рекомендация: Всегда добавляйте default даже если кажется, что все варианты учтены. Это делает программу более надёжной.

2.6. switch с char

char — это целочисленный тип, поэтому его можно использовать в switch:

cpp

```cpp
char operation;
std::cin >> operation;

switch (operation) {
    case '+':
        std::cout << "Сложение";
        break;
    case '-':
        std::cout << "Вычитание";
        break;
    case '*':
        std::cout << "Умножение";
        break;
    case '/':
        std::cout << "Деление";
        break;
    default:
        std::cout << "Неизвестная операция";
        break;
}
```

2.7. Вложенные switch

Внутри блока case может находиться другой switch:

```cpp
int mainCategory, subCategory;
std::cin >> mainCategory >> subCategory;
```

```
switch (mainCategory) {
    case 1:  // Фрукты
        switch (subCategory) {
            case 1: std::cout << "Яблоко"; break;
            case 2: std::cout << "Банан"; break;
            default: std::cout << "Другой фрукт"; break;
        }
        break;
    case 2:  // Овощи
        switch (subCategory) {
            case 1: std::cout << "Морковь"; break;
            case 2: std::cout << "Картофель"; break;
            default: std::cout << "Другой овощ"; break;
        }
        break;
    default:
        std::cout << "Неизвестная категория";
        break;
}
```

2.8. switch vs if-else if-else

Критерий	switch	if-else if-else
Условие	Только равенство целочисленной переменной	Любое логическое выражение

Критерий	switch	if-else if-else
Типы	int, char, enum, bool	Любые типы, включая double, string
Скорость	Обычно быстрее (таблица переходов)	Может быть медленнее при многих проверках
Читаемость	Хороша для 3+ вариантов одного значения	Лучше для сложных условий
Диапазоны	Не поддерживает (x > 10 && x < 20)	Поддерживает

Когда использовать switch:

1. Проверка одной переменной на множество конкретных значений
2. Меню с выбором пункта (1, 2, 3...)
3. Обработка кодов операций (+, -, *, /)
4. Определение дня недели, месяца, времени года
5. Когда использовать if-else:
6. Проверка диапазонов (x >= 10 && x <= 20)
7. Сравнение с double или string
8. Сложные логические условия ((x > 0 && y > 0) || (x < 0 && y < 0))

```
// Хорошо для switch
switch (dayOfWeek) {
    case 1: case 2: case 3: case 4: case 5:
        std::cout << "Рабочий день"; break;
    case 6: case 7:
```

```
        std::cout << "Выходной"; break;
    }

    // Плохо для switch (нужен if-else)
    if (temperature < 0) {
        std::cout << "Мороз";
    } else if (temperature < 15) {
        std::cout << "Прохладно";
    } else if (temperature < 25) {
        std::cout << "Тепло";
    } else {
        std::cout << "Жарко";
    }
```

2.9. Типичные ошибки при работе с switch

Ошибка	Пример	Последствие
Забытый break	case 1: x++; case 2: y++;	Fall-through, выполняется несколько блоков
Отсутствие default	Нет обработки других значений	Непредвиденные значения игнорируются
Нецелочисленный тип	switch (3.14)	Ошибка компиляции

Ошибка	Пример	Последствие
Не константа в case	case variable:	Ошибка компиляции
Дублирование case	case 1: и ещё case 1:	Ошибка компиляции
break после return	case 1: return x; break;	break никогда не выполнится (но не ошибка)

```
// ОШИБКА: неконстантное выражение в case
int value = 10;
switch (x) {
    case value:  // ОШИБКА! value должна быть константой
        break;
}

// ПРАВИЛЬНО: константа
const int VALUE = 10;
switch (x) {
    case VALUE:  // OK
        break;
}
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Определение названия дня недели

```
//
==========================================================================
=
// Программа: определение дня недели по номеру (1-7)
// Цель: демонстрация базового использования switch-case
//
==========================================================================
=

#include <iostream>

int main() {
    int day;  // Переменная для хранения номера дня недели

    // Запрашиваем ввод у пользователя
    std::cout << "Введите номер дня недели (1-7): ";
    std::cin >> day;

    // Используем switch для выбора дня
    switch (day) {
        case 1:
            std::cout << "Понедельник" << std::endl;
            break;  // Выход из switch после выполнения
        case 2:
            std::cout << "Вторник" << std::endl;
            break;
        case 3:
            std::cout << "Среда" << std::endl;
```

```
            break;
        case 4:
            std::cout << "Четверг" << std::endl;
            break;
        case 5:
            std::cout << "Пятница" << std::endl;
            break;
        case 6:
            std::cout << "Суббота" << std::endl;
            break;
        case 7:
            std::cout << "Воскресенье" << std::endl;
            break;
        default:  // Если введено число вне диапазона 1-7
            std::cout << "Ошибка: введите число от 1 до 7!" << std::endl;
            break;
    }

    return 0;
}

// Примеры работы:
// Ввод: 3 → Вывод: "Среда"
// Ввод: 7 → Вывод: "Воскресенье"
// Ввод: 0 → Вывод: "Ошибка: введите число от 1 до 7!"
```

3.2. Пример 2: Группировка case для определения времени года

```
//
==========================================================================
=
// Программа: определение времени года по номеру месяца
// Цель: демонстрация группировки case для объединения вариантов
//
==========================================================================
=

#include <iostream>

int main() {
    int month;  // Номер месяца (1-12)

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

    // Группировка: несколько case выполняют один код
    switch (month) {
        case 12:  // Декабрь
        case 1:   // Январь
        case 2:   // Февраль
            std::cout << "Зима" << std::endl;
            break;

        case 3:   // Март
        case 4:   // Апрель
        case 5:   // Май
            std::cout << "Весна" << std::endl;
```

```
            break;

        case 6:  // Июнь
        case 7:  // Июль
        case 8:  // Август
            std::cout << "Лето" << std::endl;
            break;

        case 9:  // Сентябрь
        case 10: // Октябрь
        case 11: // Ноябрь
            std::cout << "Осень" << std::endl;
            break;

        default: // Некорректный ввод
            std::cout << "Ошибка: месяц должен быть от 1 до 12!" << std::endl;
            break;
    }

    return 0;
}

// Объяснение: case 12, 1, 2 выполняют ОДИНАКОВЫЙ код (вывод "Зима")
// Без break между ними происходит fall-through, но это здесь нужно!
```

3.3. Пример 3: Простой консольный калькулятор с меню

```
//
==========================================================================
=
// Программа: консольный калькулятор с выбором операции
// Цель: демонстрация switch для реализации меню пользователя
//
==========================================================================
=

#include <iostream>
#include <limits>  // для numeric_limits

int main() {
    double a, b;    // Операнды (могут быть дробными)
    int choice;     // Пункт меню

    // Вывод меню
    std::cout << "===== КАЛЬКУЛЯТОР =====" << std::endl;
    std::cout << "1. Сложение (+)" << std::endl;
    std::cout << "2. Вычитание (-)" << std::endl;
    std::cout << "3. Умножение (*)" << std::endl;
    std::cout << "4. Деление (/)" << std::endl;
    std::cout << "0. Выход" << std::endl;
    std::cout << "Выберите операцию: ";
    std::cin >> choice;

    // Если выбран выход, завершаем программу
    if (choice == 0) {
        std::cout << "До свидания!" << std::endl;
```

```
        return 0;
    }

    // Если выбран пункт 1-4, запрашиваем числа
    if (choice >= 1 && choice <= 4) {
        std::cout << "Введите два числа: ";
        std::cin >> a >> b;
    }

    // Основной switch для выбора операции
    switch (choice) {
        case 1:  // Сложение
            std::cout << a << " + " << b << " = " << a + b << std::endl;
            break;

        case 2:  // Вычитание
            std::cout << a << " - " << b << " = " << a - b << std::endl;
            break;

        case 3:  // Умножение
            std::cout << a << " * " << b << " = " << a * b << std::endl;
            break;

        case 4:  // Деление с проверкой делителя
            if (b == 0) {
                std::cout << "Ошибка: деление на ноль невозможно!" <<
std::endl;
            } else {
                std::cout << a << " / " << b << " = " << a / b << std::endl;
```

```
            }
            break;

        default:  // Неверный выбор меню
            std::cout << "Ошибка: неверный пункт меню!" << std::endl;
            break;
    }

    return 0;
}
```

3.4. Пример 4: Конвертер единиц измерения (повышенный уровень)

```
// ============================================================
// Программа: конвертер единиц измерения
// Поддерживает: длина (м, см, км, мм), масса (кг, г, т)
// Цель: демонстрация вложенных switch и сложного меню
// ============================================================

#include <iostream>
#include <iomanip>
#include <string>

int main() {
```

```
int category;   // Категория: 1-длина, 2-масса
int fromUnit;   // Из какой единицы
int toUnit;     // В какую единицу
double value;   // Вводимое значение
double result;  // Результат

std::cout << "===== КОНВЕРТЕР ЕДИНИЦ =====" << std::endl;
std::cout << "1. Длина" << std::endl;
std::cout << "2. Масса" << std::endl;
std::cout << "Выберите категорию: ";
std::cin >> category;

if (category == 1) {
    // ========== КАТЕГОРИЯ: ДЛИНА ==========
    std::cout << "\n--- Длина ---" << std::endl;
    std::cout << "1. Метры (м)" << std::endl;
    std::cout << "2. Сантиметры (см)" << std::endl;
    std::cout << "3. Километры (км)" << std::endl;
    std::cout << "4. Миллиметры (мм)" << std::endl;
    std::cout << "Из какой единицы: ";
    std::cin >> fromUnit;
    std::cout << "В какую единицу: ";
    std::cin >> toUnit;
    std::cout << "Введите значение: ";
    std::cin >> value;

    // Сначала переводим всё в метры (базовая единица)
    double inMeters;
    switch (fromUnit) {
```

```
        case 1: inMeters = value; break;                // метры → метры
        case 2: inMeters = value / 100.0; break;        // см → метры
        case 3: inMeters = value * 1000.0; break;       // км → метры
        case 4: inMeters = value / 1000.0; break;       // мм → метры
        default:
            std::cout << "Ошибка: неверная единица!" << std::endl;
            return 1;
    }

    // Затем переводим из метров в нужную единицу
    switch (toUnit) {
        case 1: result = inMeters; break;               // метры
        case 2: result = inMeters * 100.0; break;       // сантиметры
        case 3: result = inMeters / 1000.0; break;      // километры
        case 4: result = inMeters * 1000.0; break;      // миллиметры
        default:
            std::cout << "Ошибка: неверная единица!" << std::endl;
            return 1;
    }

} else if (category == 2) {
    // ========== КАТЕГОРИЯ: МАССА ==========
    std::cout << "\n--- Масса ---" << std::endl;
    std::cout << "1. Килограммы (кг)" << std::endl;
    std::cout << "2. Граммы (г)" << std::endl;
    std::cout << "3. Тонны (т)" << std::endl;
    std::cout << "Из какой единицы: ";
    std::cin >> fromUnit
    std::cout << "В какую единицу: ";
```

```
        std::cin >> toUnit;
        std::cout << "Введите значение: ";
        std::cin >> value;

        // Переводим в килограммы (базовая единица)
        double inKg;
        switch (fromUnit) {
            case 1: inKg = value; break;                    // кг → кг
            case 2: inKg = value / 1000.0; break;           // г → кг
            case 3: inKg = value * 1000.0; break;           // т → кг
            default:
                std::cout << "Ошибка: неверная единица!" << std::endl;
                return 1;
        }

        // Переводим из килограммов в нужную единицу
        switch (toUnit) {
            case 1: result = inKg; break;                   // кг
            case 2: result = inKg * 1000.0; break;          // г
            case 3: result = inKg / 1000.0; break;          // т
            default:
                std::cout << "Ошибка: неверная единица!" << std::endl;
                return 1;
        }

    } else {
        std::cout << "Ошибка: неверная категория!" << std::endl;
        return 1;
    }
```

```
    // Вывод результата с форматированием
    std::cout << std::fixed << std::setprecision(4);
    std::cout << "Результат: " << result << std::endl;

    return 0;
}
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Простой switch без группировки, 4-7 вариантов.

№	Задание
1	По номеру дня недели (1-7) вывести его название
2	По номеру месяца (1-12) вывести название месяца
3	По коду операции (1-4) выполнить: 1-сложение, 2-вычитание, 3-умножение, 4-деление двух чисел
4	По номеру времени года (1-4) вывести: 1-Зима, 2-Весна, 3-Лето, 4-Осень
5	По числу (1-5) вывести оценку словами: 1-«Плохо», 2-«Неудовлетворительно», 3-«Удовлетворительно», 4-«Хорошо», 5-«Отлично»
6	По числу (1-3) вывести название фигуры: 1-Круг, 2-Квадрат, 3-Треугольник

№	Задание
7	По коду цвета (1-4) вывести: 1-Красный, 2-Зелёный, 3-Синий, 4-Жёлтый
8	По числу (1-6) вывести название игральной кости (One, Two, ..., Six)
9	По коду направления (1-4) вывести: 1-Север, 2-Юг, 3-Запад, 4-Восток
10	По числу (1-5) вывести римскую цифру: I, II, III, IV, V
11	По коду месяца (1-12) вывести количество дней в месяце (без учёта високосности, февраль — 28)
12	По коду действия (1-3) вывести: 1-Старт, 2-Пауза, 3-Стоп
13	По числу (1-4) вывести математическую операцию: 1-+, 2--, 3-*, 4-/
14	По коду времени суток (1-4) вывести: 1-Утро, 2-День, 3-Вечер, 4-Ночь
15	По номеру четверти координатной плоскости (1-4) вывести знаки координат: 1-(+,+), 2-(+,−), 3-(−,−), 4-(−,+)

Средний уровень (оценка «хорошо»)

Группировка case, использование default, 8-12 вариантов.

№	Задание
1	По номеру дня недели (1-7) вывести: рабочий день (1-5) или выходной (6-7)
2	По номеру месяца (1-12) вывести время года (группировка: зима, весна, лето, осень)
3	Калькулятор: ввод двух чисел и символа операции (+, -, *, /). Использовать switch с char
4	По числу (1-12) вывести: месяц и его сезон (например, «Январь — зима»)
5	По коду оценки (A, B, C, D, F) вывести описание: A-Отлично, B-Хорошо, C-Удовлетворительно, D-Неудовлетворительно, F-Провал
6	По коду единицы измерения (1-3) и значению перевести: 1-дюймы в см, 2-футы в м, 3-мили в км
7	По номеру месяца вывести количество дней в месяце (с учётом февраля — 28/29, год запросить отдельно)
8	Конвертер валют: выбрать направление (1-USD→RUB, 2-RUB→USD, 3-EUR→RUB, 4-RUB→EUR) и ввести сумму
9	Определить, является ли символ: гласной, согласной, цифрой, другим символом (использовать switch с char)
1	По номеру месяца вывести: пору года и количество дней в

№	Задание
0	месяце
1	Калькулятор с поддержкой: +, -, *, /, ^ (возведение в степень), % (остаток)
2	По числу от 1 до 10 вывести его название на английском (One, Two, ... Ten)
3	Меню программы: 1-Запуск, 2-Настройки, 3-Справка, 0-Выход. При выборе вывести соответствующее сообщение
4	По коду операции (1-4) выполнить: 1-площадь круга, 2-площадь прямоугольника, 3-площадь треугольника, 4-площадь квадрата
5	Конвертер температуры: 1-°C→°F, 2-°F→°C, 3-°C→K, 4-K→°C

Повышенный уровень (оценка «отлично»)

Вложенные switch, полное меню, защита от ошибок ввода.

№	Задание
1	Расширенное меню с вложенными подменю: главное меню (1-Работа с числами, 2-Работа со строками, 0-Выход) и подменю. Для чисел: 1-Чётное/нечётное, 2-Положительное/отрицательное, 3-Простые арифметические операции
2	Геометрический калькулятор: меню фигур (1-Круг, 2-

№	Задание
	Квадрат, 3-Прямоугольник, 4-Треугольник). Для каждой фигуры — подменю (площадь, периметр, объём если применимо)
3	Конвертер величин с двумя уровнями: категория (длина, масса, время, температура) → единицы измерения. Полная обработка ошибок
4	Тест-тренажёр: 5 вопросов с выбором ответа (1-4). После ответов вывести количество правильных ответов и оценку
5	Калькулятор систем счисления: ввести число (0-255), выбрать систему (2, 8, 10, 16), вывести число во всех системах
6	Меню ресторана: выводить блюда по категориям (1-Салаты, 2-Горячее, 3-Напитки, 4-Десерты). Внутри категории — выбор блюда, подсчёт итоговой стоимости
7	Программа-калькулятор дат: ввести день и месяц, вывести: знак зодиака, время года, количество дней до конца года
8	Морской бой (упрощённый): по введённым координатам (буква A-J, число 1-10) определить, попал ли в корабль (заранее заданная расстановка из 3 кораблей)
9	Расписание звонков: по номеру урока (1-8) вывести время начала и конца урока (урок 45 мин, перемена 10 мин, большая перемена 20 мин)

№	Задание
10	Система оценок ECTS: ввести балл (0-100) и вывести оценку A/B/C/D/E/FX/F с описанием. Использовать switch с диапазонами через if внутри case
11	Магазин: меню товаров (категории → товары), подсчёт корзины, применение скидки (VIP-карта — 10%, студенческий — 5%), вывод чека
12	Программа-экзаменатор: 5 случайных вопросов по темам (1-C++, 2-Python, 3-Java). Пользователь выбирает тему, отвечает на вопросы, получает результат
13	Калькулятор кредита: меню (1-Аннуитетный платёж, 2-Дифференцированный платёж, 3-Досрочное погашение). Ввод суммы, ставки, срока — расчёт платежей
14	Игра «Угадай число» с меню: 1-Новая игра (с выбором уровня сложности: 1-10, 1-50, 1-100), 2-Правила, 3-Рекорды, 0-Выход
15	Комплексное меню пользователя: регистрация/вход (3 попытки), после входа — доступ к функциям (1-Калькулятор, 2-Конвертер, 3-Тест, 4-Выход из аккаунта). Все переходы с помощью switch

5. Контрольные вопросы

Раздел А. Синтаксис и базовое использование

1. Что такое оператор switch? Для решения каких задач он предназначен?
2. Напишите полный синтаксис оператора switch-case с примером.
3. Какие типы данных можно использовать в switch? Почему нельзя использовать double и string?
4. Что означает ключевое слово break в контексте switch? Что произойдёт при его отсутствии?
5. Что такое fall-through (проваливание)? Приведите пример, где оно полезно.
6. Для чего нужен блок default? Обязателен ли он?
7. Чем отличается switch от if-else if-else? Когда что предпочтительнее?
8. Можно ли использовать switch для проверки диапазонов (x > 10 && x < 20)? Почему?

Раздел Б. Группировка и вложенность

1. Как выполнить одинаковые действия для нескольких значений case? Приведите пример.
2. Какова максимальная глубина вложенности switch? Есть ли ограничения?
3. Можно ли использовать switch внутри case другого switch? Приведите пример.
4. Можно ли объявить переменную внутри case? С какими ограничениями?

Раздел В. Сравнение с другими конструкциями

1. В каких случаях switch работает быстрее цепочки if-else?
2. Как переписать длинную цепочку if-else if-else на switch? Когда это возможно?

3. Как проверить с помощью switch, является ли символ гласной буквой?
4. Чем отличается switch на int от switch на char?

Раздел Г. Ошибки и лучшие практики

1. Что произойдёт, если забыть break во всех case? Приведите пример.
2. Какие ошибки компиляции могут возникнуть при использовании switch?
3. Можно ли в case использовать переменную? Почему?
4. Что лучше: писать break после default или нет?
5. Как обработать в switch значения 1, 2, 3 одинаково, а 4, 5 — по-другому?
6. Как организовать меню с помощью switch? Приведите пример.

Раздел Д. Анализ кода

1. Что выведет следующий код?

```
int x = 2;
switch (x) {
    case 1: std::cout << "A";
    case 2: std::cout << "B";
    case 3: std::cout << "C";
    default: std::cout << "D";
}
```

2. Исправьте ошибки в коде:

cpp

```
double d = 2.5;
switch (d) {
    case 1.5: std::cout << "Один с половиной"; break;
    case 2.5: std::cout << "Два с половиной"; break;
```

```
}
```

3. Как переписать следующий код на switch?

```
cpp
if (grade == 'A') {
    std::cout << "Отлично";
} else if (grade == 'B') {
    std::cout << "Хорошо";
} else if (grade == 'C') {
    std::cout << "Удовлетворительно";
} else {
    std::cout << "Неудовлетворительно";
}
```

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист (название работы, дисциплина, студент, преподаватель)

Цель работы

Теоретическое обоснование (синтаксис switch, break, default, группировка case, сравнение с if-else)

Ход выполнения работы:

Код программы с подробными комментариями (каждый case пояснён)

Блок-схема алгоритма (обязательно для этой работы)

Скриншоты выполнения для различных сценариев (корректный ввод, неверный ввод, граничные значения)

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали о множественном выборе, когда использовать switch, а когда if-else)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №4)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) — правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, использование switch, наличие break и default).

Оценка «хорошо» (4) — правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего уровня (группировка case, полная обработка всех вариантов через default, осмысленные имена переменных, читаемость кода).

Оценка «отлично» (5) — правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (вложенные switch, полное меню с несколькими уровнями, обработка ошибочного ввода через std::cin.fail(), проверка граничных значений, подробные комментарии в коде, блок-схема в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	**Базовый (3)**	**Средний (4)**	**Повышенный (5)**
	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
	Компиляция и запуск			
.1	Программа компилируется без ошибок			
.2	Программа успешно запускается и не аварийно завершается			
.3	Отсутствие предупреждений компилятора	—		
	Синтаксис switch			
.1	Правильное использование switch (выражение)			
.2	Правильное использование case константа:			

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.3	Наличие break после каждого блока (кроме группировки)			
.4	Наличие блока default для обработки остальных случаев	—		
.5	Отсутствие fall-through там, где он не нужен			
	Группировка и вложенность			
.1	Использование группировки case для одинаковых действий	—		
.2	Правильное использование вложенных switch	—	—	
.3	Отсутствие дублирования кода благодаря группировке	—		

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Типы данных в switch			
.1	Использование целочисленных типов (int, char)			
.2	Отсутствие попыток использовать double или string			
.3	Использование char для операций (+, -, *, /)	—		
	Обработка ошибок			
.1	Обработка неверного выбора через default	—		
.2	Проверка деления на ноль (в калькуляторах)	—		
.3	Проверка на ввод некорректных данных (std::cin.fail())	—	—	
.4	Очистка буфера ввода при ошибке	—	—	

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Качество кода			
.1	Правильные отступы и форматирование кода			
.2	Использование осмысленных имён переменных	—		
.3	Отсутствие «магических чисел» (использование const или комментариев)	—		
.4	Наличие комментариев в коде (пояснение каждого case)	—		
.5	Детальные комментарии с объяснением логики группировки	—	—	
	Отчёт			
.1	Наличие титульного листа, цели, хода работы			

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	Скриншоты результатов выполнения (минимум 3 теста)			
.3	Скриншоты для всех трёх уровней	—		
.4	Таблица тестирования (вход → ожидаемый → фактический результат)	—		
.5	Блок-схема алгоритма	—		
.6	Тестирование на граничных значениях (неверный ввод, крайние варианты)	—	—	
.7	Ответы на контрольные вопросы (письменно, развёрнуто)	—		

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) — обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня (№1–15).

Написать программу, использующую switch с 4-7 case.

Обязательно использовать break после каждого case.

Продемонстрировать понимание:

Синтаксиса switch-case

Разницы между switch и if-else

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) — дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Использовать группировку case для объединения нескольких вариантов.

Обязательно включить блок default для обработки некорректного ввода.

Использовать char в switch (например, для операций калькулятора).

Продемонстрировать понимание fall-through (намеренного проваливания).

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

Блок-схему алгоритма

Повышенный уровень (оценка 5) — требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Вложенные switch — минимум два уровня меню.

Полное меню с возможностью возврата к предыдущим уровням.

Обработка ошибок ввода:

Проверка std::cin.fail() после ввода чисел

Очистка буфера ввода

Повторный запрос при неверном вводе

Демонстрация понимания ограничений switch:

Объяснение, почему нельзя использовать double и string

Сравнение с if-else в комментариях

Тестирование на граничных значениях: неверный выбор меню, выход за пределы диапазона, ввод букв вместо чисел.

В отчёте дополнительно представить:

Диаграмму переходов между меню

Таблицу тестирования с ошибочными сценариями

7.4. Примеры проверки соответствия

Выполненные задания	Оценка	Проверка критериев
Только базовый вариант №1 (дни недели)	3	switch с 7 case, break, ✘ нет default, ✘ нет группировки
Базовый №1 + средний №2 (время года с группировкой)	4	Плюс группировка case, default, осмысленные имена. Но: ✘ нет вложенных switch, ✘ нет обработки cin.fail()
Базовый №1 + средний №2 + повышенный №1 (меню с подменю)	5	Все критерии: вложенные switch, полная обработка ошибок, тестирование границ, блок-схема, комментарии

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	Влияние	Как исправить
Забыт break (непреднамеренный fall-through)	Снижение на 1 балл	Добавить break после каждого case
Отсутствие default	Снижение до «3» (даже при правильных case)	Всегда добавлять default
Использование double в switch	Ошибка компиляции	Переделать на if-else или использовать int
Неверный выбор типа данных для case	Ошибка компиляции	Использовать константы, а не переменные
Нет проверки деления на ноль в калькуляторе	Аварийное завершение	Добавить if (b == 0) перед делением
Отсутствие комментариев о группировке	Снижение до «3»	Добавить пояснения, почему case сгруппированы
Вложенные switch без обработки default на каждом уровне	Снижение на 1 балл	Добавить default для каждого switch

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Синтаксис switch-case

Разницу между switch и if-else

Что такое fall-through и когда оно полезно

(для оценки 4) — объяснить группировку case и назначение default

(для оценки 5) — объяснить, как работают вложенные switch, как обрабатываются ошибки ввода

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Почему в вашем коде нет break между case 1 и case 2?"	Понимание намеренной группировки
"Что произойдёт, если пользователь введёт 0 в вашей программе?"	Понимание необходимости default
"Можно ли использовать switch для проверки високосности года?"	Понимание ограничений switch
"Как бы вы переписали ваш switch на if-else? Что лучше в этой задаче?"	Умение выбирать инструмент под задачу

Лабораторная работа №5

Циклы. Табуляция функций

1. Цель работы

Научиться использовать циклы for, while, do-while для организации повторяющихся вычислений. Освоить табуляцию функций — вывод таблицы значений функции на заданном интервале с заданным шагом. Получить навыки накопления суммы (аккумулятор), поиска минимального и максимального значения функции, организации вложенных циклов.

2. Теоретическое обоснование

2.1. Понятие цикла

Цикл — это управляющая конструкция, позволяющая многократно выполнять один и тот же блок кода, пока выполняется некоторое условие. Циклы используются для:

Обработки последовательностей данных (массивы, строки)

Табуляции функций (вычисление значений в ряде точек)

Накопления сумм и произведений

Поиска экстремумов

Генерации таблиц и графиков

2.2. Цикл for

Синтаксис:

cpp

```
for (инициализация; условие; инкремент/декремент) {
    // тело цикла
}
```

Порядок выполнения:

Выполняется инициализация (один раз в начале)

Проверяется условие: если true → вход в тело, если false → выход из цикла

Выполняется тело цикла

Выполняется инкремент/декремент

Переход к шагу 2

```
// Пример: вывод чисел от 1 до 10
for (int i = 1; i <= 10; i++) {
    std::cout << i << " ";
}
// Вывод: 1 2 3 4 5 6 7 8 9 10
```

Разбор: int i = 1 — инициализация счётчика; i <= 10 — условие продолжения; i++ — увеличение счётчика на 1 после каждой итерации.

```
// Пример: обратный отсчёт от 10 до 1
for (int i = 10; i >= 1; i--) {
    std::cout << i << " ";
}
// Вывод: 10 9 8 7 6 5 4 3 2 1
```

Важные особенности:

Переменная счётчика обычно объявляется внутри for (область видимости — только цикл)

Любая из трёх частей может быть пустой

Можно использовать несколько переменных через запятую

```
// Несколько переменных в for
for (int i = 0, j = 10; i < j; i++, j--) {
    std::cout << "i=" << i << " j=" << j << std::endl;
}
```

2.3. Цикл while

Синтаксис:

cpp

```
while (условие) {
    // тело цикла
}
```

Порядок выполнения:

Проверяется условие

Если true — выполняется тело, затем возврат к шагу 1

Если false — выход из цикла

cpp

```
// Пример: вывод чисел от 1 до 10 через while
int i = 1;          // инициализация ДО цикла
while (i <= 10) {   // проверка условия
    std::cout << i << " ";
    i++;            // изменение счётчика ВНУТРИ тела
}
```

Когда использовать while:

Количество итераций заранее неизвестно

Цикл зависит от внешних условий (ввод пользователя, состояние датчика)

```
// Пример: чтение чисел до тех пор, пока не введён 0
int number;
std::cout << "Введите числа (0 для выхода): ";
std::cin >> number;

while (number != 0) {
    std::cout << "Вы ввели: " << number << std::endl;
```

```
    std::cin >> number;
}
```

2.4. Цикл do-while

Синтаксис:

cpp

```
do {
    // тело цикла
} while (условие);
```

Порядок выполнения:

Выполняется тело цикла (хотя бы один раз!)

Проверяется условие

Если true — повтор, если false — выход

```
// Пример: меню с гарантированным показом хотя бы один раз
int choice;
do {
    std::cout << "1. Начать" << std::endl;
    std::cout << "2. Настройки" << std::endl;
    std::cout << "0. Выход" << std::endl;
    std::cout << "Ваш выбор: ";
    std::cin >> choice;
} while (choice != 0);
```

Когда использовать do-while:

Тело цикла должно выполниться хотя бы один раз в любом случае

Организация меню с проверкой после ввода

Ввод данных с проверкой корректности

```
// Пример: ввод положительного числа (с защитой от некорректного ввода)
int value;
do {
    std::cout << "Введите положительное число: ";
    std::cin >> value;
    if (value <= 0) {
        std::cout << "Ошибка! Число должно быть > 0" << std::endl;
    }
} while (value <= 0);
```

2.5. Сравнение видов циклов

Критерий	for	while	do-while
Минимальное число итераций	0	0	1
Инициализация счётчика	Внутри	Снаружи	Снаружи
Изменение счётчика	В заголовке	В теле	В теле
Когда использовать	Известно количество итераций	Неизвестно, проверка перед входом	Нужно выполнить хотя бы раз
Типичный пример	Табуляция, обход	Ожидание события, чтение	Меню, ввод с

Критерий	for	while	do-while
	массива	до 0	проверкой

```
// Один и тот же алгоритм тремя способами (сумма чисел от 1 до N)

// for — самый компактный
int sum = 0;
for (int i = 1; i <= N; i++) {
    sum += i;
}

// while — более явный
int i = 1;
int sum = 0;
while (i <= N) {
    sum += i;
    i++;
}

// do-while — не подходит, если N может быть 0 (сумма будет 1)
int i = 1;
int sum = 0;
do {
    sum += i;
    i++;
} while (i <= N);
```

2.6. Бесконечные циклы и выход из цикла

Бесконечный цикл возникает, если условие всегда истинно:

```
// Бесконечный цикл
for (;;) { }          // все части пустые
while (true) { }      // условие всегда true
do { } while (true);  // do-while с true
```

Выход из цикла:

break — немедленный выход из цикла (прерывает выполнение)

continue — переход к следующей итерации (пропуск оставшейся части тела)

```
// Пример: break при нахождении нужного элемента
for (int i = 0; i < 100; i++) {
    if (i == 50) {
        break;  // выход из цикла при i=50
    }
}

// Пример: continue для пропуска чётных чисел
for (int i = 1; i <= 10; i++) {
    if (i % 2 == 0) {
        continue;  // пропускаем вывод чётных чисел
    }
    std::cout << i << " ";  // вывод: 1 3 5 7 9
}
```

2.7. Табуляция функции

Табуляция функции — это вычисление и вывод значений функции в ряде точек (аргументов) на заданном интервале с определённым шагом.

Общий алгоритм:

Запросить у пользователя интервал [a, b] и шаг h

Проверить корректность ввода (a ≤ b, h > 0)

Организовать цикл от a до b с шагом h

На каждой итерации:

Вычислить значение функции f(x)

Вывести пару x и f(x) в виде таблицы

```
// Общая структура табуляции
double a, b, h;
std::cout << "Введите начало интервала a: ";
std::cin >> a;
std::cout << "Введите конец интервала b: ";
std::cin >> b;
std::cout << "Введите шаг h: ";
std::cin >> h;

// Проверка корректности
if (a > b || h <= 0) {
    std::cout << "Ошибка: a должно быть <= b, h > 0" << std::endl;
    return 1;
}

// Шапка таблицы
std::cout << "====================" << std::endl;
std::cout << "   x    |   f(x)  " << std::endl;
std::cout << "====================" << std::endl;
```

```
// Табуляция
for (double x = a; x <= b + 1e-9; x += h) {
    double y = f(x);  // вычисление значения функции
    std::cout << std::fixed << std::setprecision(4);
    std::cout << std::setw(7) << x << " | " << std::setw(7) << y << std::endl;
}
```

Важно: Из-за погрешностей вычислений с double используем <= b + 1e-9 для корректного включения последней точки.

2.8. Накопление суммы (аккумулятор)

Аккумулятор — переменная, которая накапливает сумму значений в цикле.

```
// Вычисление суммы элементов
double sum = 0.0;              // начальное значение
for (int i = 1; i <= N; i++) {
    sum += i;                  // sum = sum + i
}
// sum = 1 + 2 + ... + N
```

Общая схема:

Создать переменную-аккумулятор с начальным значением 0 (для суммы) или 1 (для произведения)

В цикле добавлять (или умножать) каждое значение

После цикла использовать накопленное значение

```
// Сумма квадратов чисел от 1 до N
double sumOfSquares = 0;
for (int i = 1; i <= N; i++) {
```

```
    sumOfSquares += i * i;
}

// Произведение чисел от 1 до N (факториал)
long long factorial = 1;
for (int i = 2; i <= N; i++) {  // начни с 2, т.к. 1 не меняет произведение
    factorial *= i;
}
```

2.9. Поиск минимума и максимума

```
// Поиск минимума и максимума на интервале
double minValue = f(a);   // начальное значение — в точке a
double maxValue = f(a);
double xMin = a, xMax = a;

for (double x = a; x <= b + 1e-9; x += h) {
    double y = f(x);
    if (y < minValue) {
        minValue = y;
        xMin = x;
    }
    if (y > maxValue) {
        maxValue = y;
        xMax = x;
    }
}

std::cout << "Минимум: f(" << xMin << ") = " << minValue << std::endl;
```

```
std::cout << "Максимум: f(" << xMax << ") = " << maxValue << std::endl;
```

2.10. Вычисление суммы ряда

Часто требуется вычислить сумму бесконечного ряда с заданной точностью:

S(x) = a0 + a1 + a2 + ... + an + ...

Алгоритм:

Вычислить первый член ряда

Организовать цикл, пока очередной член больше заданной точности

На каждой итерации прибавлять член к сумме

Вычислять следующий член через рекуррентную формулу (чтобы избежать переполнения)

```
// Пример: вычисление sin(x) через ряд Тейлора
// sin(x) = x - x^3/3! + x^5/5! - x^7/7! + ...
double sinTaylor(double x, double epsilon = 1e-6) {
    double sum = 0;
    double term = x;        // первый член ряда (n=0)
    int n = 0;

    while (fabs(term) > epsilon) {
        sum += term;
        n++;
        // Рекуррентная формула: term_{n+1} = -term_n * x^2 / ((2n+1)*(2n+2))
        term = -term * x * x / ((2 * n - 1) * (2 * n));
    }
    return sum;
```

```
}
```

2.11. Вложенные циклы

Вложенный цикл — цикл внутри другого цикла. Внутренний цикл полностью выполняется для каждой итерации внешнего.

cpp

```
// Пример: таблица умножения
for (int i = 1; i <= 9; i++) {          // внешний цикл (строки)
    for (int j = 1; j <= 9; j++) {      // внутренний цикл (столбцы)
        std::cout << std::setw(4) << i * j;
    }
    std::cout << std::endl;             // перевод строки после каждой строки
}
```

Порядок выполнения:

i = 1, затем внутренний цикл выполняется для j = 1..9

i = 2, внутренний цикл снова выполняется для j = 1..9

И так далее до i = 9

2.12. Типичные ошибки при работе с циклами

Ошибка	Пример	Последствие
Бесконечный цикл	for (int i=0; i>=0; i++)	Программа зависает
Сравнение double в условии	for (double x = 0; x <= 1; x += 0.1)	Может не выполниться нужное число раз
Забыт инкремент	for (int i=0;	Бесконечный

Ошибка	Пример	Последствие
	i<10;)	цикл
Лишняя ; после for	for (int i=0; i<10; i++);	Тело не выполнится
Неправильная инициализация	int i; while (i <= 10) { i++; }	Неопределённое поведение
break вместо continue	Пропуск не той итерации	Неправильная логика
Потеря точности в сумме ряда	Суммирование от меньших к большим	Накопление погрешности

```
// ОШИБКА: лишняя ; после for
for (int i = 0; i < 10; i++); {
    std::cout << i << " ";  // Ошибка! i не видна здесь, тело выполнится 1 раз
}

// ПРАВИЛЬНО
for (int i = 0; i < 10; i++) {
    std::cout << i << " ";
}
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Табуляция функции (базовый уровень)

```
//
==========================================================================
=
// Программа: табуляция функции y = x² на интервале [a, b] с шагом h
// Цель: демонстрация цикла for для табуляции
//
==========================================================================
=

#include <iostream>
#include <iomanip>   // для std::setw, std::fixed, std::setprecision
#include <cmath>     // для математических функций

int main() {
    double a, b, h;   // a - начало интервала, b - конец, h - шаг

    // Ввод параметров с проверкой
    std::cout << "Введите начало интервала a: ";
    std::cin >> a;
    std::cout << "Введите конец интервала b: ";
    std::cin >> b;
    std::cout << "Введите шаг h: ";
    std::cin >> h;

    // Проверка корректности ввода
    if (a > b) {
        std::cout << "Ошибка: начало интервала не может быть больше
конца!" << std::endl;
```

```
        return 1;
    }
    if (h <= 0) {
        std::cout << "Ошибка: шаг должен быть положительным!" << std::endl;
        return 1;
    }

    // Вывод шапки таблицы
    std::cout << "\n============================" << std::endl;
    std::cout << "   x     |    y = x²    " << std::endl;
    std::cout << "============================" << std::endl;

    // Настройка форматирования вывода
    std::cout << std::fixed << std::setprecision(4);

    // Цикл табуляции: от a до b с шагом h
    // Используем b + 1e-9 для учёта погрешности double
    for (double x = a; x <= b + 1e-9; x += h) {
        double y = x * x;                    // вычисляем значение функции
        std::cout << std::setw(8) << x << " | " << std::setw(12) << y << std::endl;
    }

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

    return 0;
}
```

```
// Пример работы:
// Вход: a=0, b=2, h=0.5
// Вывод:
// ============================
//    x      |     y = x²
// ============================
//   0.0000 |      0.0000
//   0.5000 |      0.2500
//   1.0000 |      1.0000
//   1.5000 |      2.2500
//   2.0000 |      4.0000
// ============================
```

3.2. Пример 2: Табуляция с нахождением минимума и максимума

```
//
==========================================================================
=
// Программа: табуляция функции y = sin(x) на [a,b] с шагом h
// Дополнительно: поиск минимального и максимального значения
// Цель: демонстрация накопления и сравнения в цикле
//
==========================================================================
=

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

```cpp
int main() {
    double a, b, h;

    std::cout << "Табуляция функции y = sin(x)" << std::endl;
    std::cout << "Введите a, b, h: ";
    std::cin >> a >> b >> h;

    // Проверка корректности
    if (a > b || h <= 0) {
        std::cout << "Ошибка ввода!" << std::endl;
        return 1;
    }

    // Инициализация минимума и максимума первым значением
    double xFirst = a;
    double yFirst = std::sin(xFirst);
    double minY = yFirst;
    double maxY = yFirst;
    double xMin = xFirst;
    double xMax = xFirst;

    // Шапка таблицы
    std::cout << "\n+----------+----------+" << std::endl;
    std::cout << "|    x     |   sin(x)  |" << std::endl;
    std::cout << "+----------+----------+" << std::endl;

    std::cout << std::fixed << std::setprecision(4);

    // Цикл табуляции
```

```
    for (double x = a; x <= b + 1e-9; x += h) {
        double y = std::sin(x);          // вычисляем sin(x)

        // Вывод строки таблицы
        std::cout << "| " << std::setw(7) << x << " | "
                  << std::setw(9) << y << " |" << std::endl;

        // Обновление минимума
        if (y < minY) {
            minY = y;
            xMin = x;
        }

        // Обновление максимума
        if (y > maxY) {
            maxY = y;
            xMax = x;
        }
    }

    std::cout << "+----------+-----------+" << std::endl;

    // Вывод результатов поиска
    std::cout << "\nРезультаты поиска:" << std::endl;
    std::cout << "Минимум: sin(" << xMin << ") = " << minY << std::endl;
    std::cout << "Максимум: sin(" << xMax << ") = " << maxY << std::endl;

    return 0;
}
```

3.3. Пример 3: Вычисление суммы ряда с заданной точностью

```
//
==========================================================================
=
// Программа: вычисление суммы ряда S = 1 + 1/2 + 1/3 + ... + 1/N
// Цель: демонстрация накопления суммы (аккумулятора)
//
==========================================================================
=

#include <iostream>
#include <iomanip>

int main() {
    int N;
    double sum = 0.0;   // аккумулятор суммы, начальное значение 0

    std::cout << "Введите количество членов ряда N: ";
    std::cin >> N;

    if (N <= 0) {
        std::cout << "Ошибка: N должно быть положительным!" <<
std::endl;
        return 1;
    }

    // Цикл накопления суммы
```

```
    for (int i = 1; i <= N; i++) {
        sum += 1.0 / i;          // добавляем очередной член ряда
        // ВАЖНО: 1.0 / i, а не 1 / i (иначе целочисленное деление)
    }

    std::cout << std::fixed << std::setprecision(6);
    std::cout << "Сумма ряда S = " << sum << std::endl;
    std::cout << "Приближённое значение ln(N) + γ = " << std::log(N) +
0.57721 << std::endl;

    return 0;
}

// Пример работы:
// Вход: N=10
// S = 1 + 1/2 + 1/3 + ... + 1/10 = 2.928968
```

3.4. Пример 4: Вложенные циклы (таблица умножения)

```
//
==========================================================
=
// Программа: вывод таблицы умножения (вложенные циклы)
// Цель: демонстрация работы вложенных циклов
//
==========================================================
=

#include <iostream>
```

```cpp
#include <iomanip>

int main() {
    int size;

    std::cout << "Введите размер таблицы (1-12): ";
    std::cin >> size;

    if (size < 1 || size > 12) {
        std::cout << "Ошибка: размер должен быть от 1 до 12!" << std::endl;
        return 1;
    }

    // Шапка таблицы
    std::cout << "    * |";
    for (int j = 1; j <= size; j++) {
        std::cout << std::setw(4) << j;
    }
    std::cout << "\n-----+";
    for (int j = 1; j <= size; j++) {
        std::cout << "----";
    }
    std::cout << std::endl;

    // Внешний цикл по строкам (множитель i)
    for (int i = 1; i <= size; i++) {
        std::cout << std::setw(4) << i << " |";   // вывод номера строки

        // Внутренний цикл по столбцам (множитель j)
```

```
        for (int j = 1; j <= size; j++) {
            std::cout << std::setw(4) << i * j;
        }
        std::cout << std::endl;  // переход на следующую строку
    }

    return 0;
}

// Пример работы (size=5):
//    * |   1   2   3   4   5
// -----+--------------------
//    1 |   1   2   3   4   5
//    2 |   2   4   6   8  10
//    3 |   3   6   9  12  15
//    4 |   4   8  12  16  20
//    5 |   5  10  15  20  25
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Табуляция простой функции на интервале [a, b] с шагом h.

№	Функция	Формула
1	Линейная	$y = 2x + 3$
2	Квадратичная	$y = x^2 - 2x + 1$
3	Кубическая	$y = x^3 - 4x$

№	Функция	Формула
4	Гипербола	$y = 1/x$ (при $x \neq 0$, пропуск при $x=0$)
5	Квадратный корень	$y = \sqrt{x}$ ($x \geq 0$)
6	Модуль	$y = \lvert x \rvert$
7	Обратная пропорциональность	$y = 5/x$ (при $x \neq 0$)
8	Константа	$y = 7$
9	Квадрат сдвинутый	$y = (x-2)^2$
10	Линейная с отрицательным коэффициентом	$y = -3x + 10$
11	Квадратичная с отрицательным старшим членом	$y = -x^2 + 4x$
12	Степенная	$y = x^4$
13	Корень кубический	$y = \sqrt[3]{x}$
14	Дробно-линейная	$y = (x+1)/(x-1)$ (при $x \neq 1$)

№	Функция	Формула
15	Сумма квадрата и константы	$y = x^2 + 5$

Для всех базовых заданий: Запросить a, b, h. Вывести таблицу в формате:

x	y
a	f(a)
a+h	f(a+h)
...	...
b	f(b)

Средний уровень (оценка «хорошо»)

Табуляция + дополнительный анализ (сумма, min, max, количество).

№	Функция	Дополнительное задание
1	$y = \sin(x)$	Найти сумму всех значений функции
2	$y = \cos(x)$	Найти минимальное и максимальное значение
3	$y = x^2 + 2x + 1$	Найти среднее арифметическое значений
4	y = e^x	Найти количество положительных

№	Функция	Дополнительное задание
	(exp(x))	значений
5	y = ln(x) (x>0)	Найти количество отрицательных значений
6	y = tg(x) (пропустить разрывы)	Найти сумму положительных значений
7	y = |x| - 5	Найти произведение значений (с проверкой на ноль)
8	$y = x^3 - 3x^2 + 2x$	Найти, сколько раз функция меняет знак
9	$y = 1/(x^2+1)$	Найти наибольшее значение
10	$y = \sqrt{(x+1)}$ ($x \geq -1$)	Найти наименьшее значение
11	y = x·sin(x)	Найти сумму квадратов значений
12	y = arctg(x)	Найти значение, ближайшее к нулю
13	$y = x/(x^2+1)$	Найти точку максимума
1	$y = 2^x$	Найти количество целых

№	Функция	Дополнительное задание
4	(pow(2,x))	значений (с погрешностью)
1 5	y = sin(x)/x (при x=0 → y=1)	Найти среднее квадратическое отклонение (упрощённо)

Повышенный уровень (оценка «отлично»)

Табуляция + накопление суммы ряда + обработка особых точек + функции.

	Задание
	Вычислить таблицу значений функции y = sin(x) и сумму ряда Тейлора для sin(x) с точностью ε=1e-6, сравнить результаты
	Вычислить таблицу значений функции y = cos(x) и сумму ряда Тейлора для cos(x), вычислить погрешность в каждой точке
	Вычислить таблицу значений функции y = e^x и сумму ряда Тейлора для e^x (1 + x + x²/2! + ...), сравнить
	Вычислить таблицу значений функции y = ln(1+x) и сумму ряда Тейлора (x - x²/2 + x³/3 - ...) для
	Программа-генератор: табуляция любой из 5 функций (выбор через меню). Для каждой функции — поиск min, max, суммы
	Табуляция функции с автоматическим подбором шага: программа сама определяет шаг так, чтобы выводилось не более 20 строк

	Задание
	Табуляция функции с пропуском точек разрыва (анализ знаменателя). Выводить сообщение «разрыв» при x, где функция не определена
	Вычислить определённый интеграл методом левых прямоугольников: $\int_a^b f(x)dx = h\cdot(y_0 + y_1 + ... + y_{n-1})$
	Вычислить определённый интеграл методом трапеций и сравнить с аналитическим значением (для простых функций)
0	Программа-исследователь: табуляция функции на интервале с поиском корней (смена знака) — выводить интервалы, где есть корень
1	Табуляция функции двух переменных: z = f(x,y) с вложенными циклами (x внешний, y внутренний). Вывод в виде матрицы
2	«Анимация» в консоли: последовательный вывод строк таблицы с задержкой (std::this_thread::sleep_for)
3	Вычисление числа π через ряд Лейбница: π/4 = 1 - 1/3 + 1/5 - 1/7 + ... с заданной точностью. Вывести количество итераций
4	Вычисление числа e через ряд: e = 1 + 1/1! + 1/2! + 1/3! + ... с заданной точностью. Вывести значение и количество членов
5	Вычисление суммы бесконечного ряда с заданной точностью для вводимого x (выбор ряда через меню). Сравнить с библиотечной функцией

5. Контрольные вопросы

Раздел А. Основы циклов

Какие виды циклов существуют в C++? Опишите синтаксис каждого.

В чём разница между циклами for, while и do-while? Когда какой использовать?

Что произойдёт, если в for не написать условие? Например: for (int i=0; ; i++)

Как работает оператор break в циклах? Приведите пример.

Как работает оператор continue в циклах? Приведите пример.

Раздел Б. Табуляция функций

Что такое табуляция функции? Для чего она нужна?

Как правильно организовать цикл для табуляции на интервале [a, b] с шагом h?

Почему в условии цикла используют x <= b + 1e-9, а не просто x <= b?

Как обработать ситуацию, когда функция не определена в некоторых точках?

Как вывести таблицу в форматированном виде (выравнивание столбцов)?

Раздел В. Накопление и поиск

Что такое аккумулятор? Как вычислить сумму элементов в цикле?

Как вычислить произведение элементов в цикле? С какого значения нужно начинать?

Как найти минимальное и максимальное значение функции на интервале?

Как найти среднее арифметическое значений функции?

В чём разница между суммированием от 1 до N и от N до 1 с точки зрения точности?

Раздел Г. Сложные циклы

Что такое вложенные циклы? Приведите пример.

Как работает break во вложенных циклах? Выйдет ли он из всех циклов?

Как организовать бесконечный цикл с выходом по условию внутри?

Что такое рекуррентная формула? Зачем она нужна при вычислении суммы ряда?

Как накопление суммы ряда отличается от простой суммы? Какая точность требуется?

Раздел Д. Анализ кода

Что выведет код?

```
int i = 0;
while (i < 5) {
    std::cout << i;
    i++;
}
```

Сколько раз выполнится тело цикла?

```
for (int i = 0; i < 10; i += 2) { }
```

Найдите ошибку в коде:

cpp

```
double sum = 0;
for (int i = 0; i <= N; i++) {
    sum = sum + 1 / i;
}
```

Как переписать этот while на for?

```
int i = 10;
while (i > 0) {
    std::cout << i;
```

```
    i--;
}
```

Что произойдёт при выполнении?

```
for (double x = 0; x != 1; x += 0.1) {
    std::cout << x << std::endl;
}
```

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (виды циклов, табуляция, накопление суммы, поиск min/max)

Ход выполнения работы:

Код программы с подробными комментариями (каждый цикл пояснён)

Блок-схема алгоритма (обязательно для этой работы)

Скриншоты выполнения для различных сценариев (корректный ввод, граничные значения, шаг больше интервала)

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали о циклах, какой вид цикла лучше для табуляции и почему)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №5)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) — правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, правильная организация цикла для табуляции, вывод таблицы).

Оценка «хорошо» (4) — правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего уровня (поиск минимума/максимума или суммы, форматирование таблицы, обработка особых точек, осмысленные имена переменных).

Оценка «отлично» (5) — правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (вычисление суммы ряда с заданной точностью, рекуррентные формулы, обработка погрешностей double, вложенные циклы, полная защита от некорректного ввода, подробные комментарии в коде, блок-схема в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Выполненные уровни заданий	Только	Базовый +	Базовый + средний +

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
		базовый	средний	повышенный
	Компиляция и запуск			
.1	Программа компилируется без ошибок	✔	✔	✔
.2	Программа успешно запускается	✔	✔	✔
.3	Отсутствие предупреждений компилятора	—	✔	✔
	Организация циклов			
.1	Правильное использование for для табуляции	✔	✔	✔
.2	Правильное использование while или do-while (где нужно)	—	✔	✔
.3	Корректное условие продолжения цикла	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.4	Отсутствие бесконечных циклов	✔	✔	✔
.5	Правильная работа с шагом (учёт погрешности double)	✔	✔	✔
	Табуляция функции			
.1	Вычисление значения функции на каждой итерации	✔	✔	✔
.2	Вывод таблицы в форматированном виде	—	✔	✔
.3	Вывод шапки таблицы	—	✔	✔
.4	Обработка особых точек (деление на ноль, область определения)	—	✔	✔
	Накопление и поиск			
.1	Вычисление суммы значений (аккумулятор)	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	Поиск минимума и максимума	—	✔	✔
.3	Вычисление среднего арифметического	—	✔	✔
.4	Вычисление суммы ряда с заданной точностью	—	—	✔
.5	Использование рекуррентных формул для суммы ряда	—	—	✔
	Вложенные циклы			
.1	Правильное использование вложенных циклов (где нужно)	—	—	✔
.2	Отсутствие лишних вычислений во внутреннем цикле	—	—	✔
	Обработка ошибок			
.1	Проверка корректности ввода (a ≤ b,	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	h > 0)			
.2	Проверка на ввод букв вместо чисел (std::cin.fail())	—	—	✔
.3	Очистка буфера ввода при ошибке	—	—	✔
.4	Вывод понятного сообщения об ошибке	—	✔	✔
	Качество кода			
.1	Правильные отступы и форматирование	✔	✔	✔
.2	Использование осмысленных имён переменных	—	✔	✔
.3	Отсутствие «магических чисел»	—	✔	✔
.4	Наличие комментариев в коде (пояснение логики цикла)	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.5	Детальные комментарии с объяснением формул	—	—	✔
	Отчёт			
.1	Наличие титульного листа, цели, хода работы	✔	✔	✔
.2	Скриншоты результатов выполнения (минимум 3 теста)	✔	✔	✔
.3	Скриншоты для всех трёх уровней	—	✔	✔
.4	Таблица тестирования	—	✔	✔
.5	Блок-схема алгоритма	—	✔	✔
.6	Тестирование на граничных значениях (h > b-a, h = 0, a = b)	—	—	✔
	Ответы на	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.7	контрольные вопросы			

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) — обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня (№1–15).

Реализовать табуляцию функции на интервале [a, b] с шагом h.

Использовать цикл for для табуляции.

Вывести таблицу в виде двух столбцов (x, y).

Продемонстрировать понимание:

Синтаксиса цикла for

Как избежать погрешности при сравнении double

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) — дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Выполнить табуляцию и дополнительный анализ (сумма, min/max, количество и т.д.).

Отформатировать вывод в виде аккуратной таблицы (с разделителями, выравниванием).

Обработать особые точки (область определения, деление на ноль).

Использовать осмысленные имена переменных.

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

Блок-схему алгоритма

Повышенный уровень (оценка 5) — требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Сумма ряда с заданной точностью:

Вычислять сумму бесконечного ряда, пока очередной член больше ε

Использовать рекуррентные формулы для эффективного вычисления

Сравнить результат с библиотечной функцией

Обработка погрешностей double:

Объяснить в комментариях, почему используется x <= b + 1e-9

Показать, что может произойти при прямом сравнении x <= b

Вложенные циклы (где требуется по заданию):

Правильная организация внешнего и внутреннего цикла

Отсутствие лишних вычислений

Полная обработка ошибок ввода:

Проверка std::cin.fail() и очистка буфера

Проверка всех граничных случаев (h > b-a, h = 0, a = b, a > b)

Детальные комментарии:

Объяснение выбора типа цикла

Пояснение рекуррентной формулы

Указание потенциальных проблем точности

В отчёте дополнительно представить:

Сравнительную таблицу (сумма ряда vs библиотечная функция)

Тестирование на граничных значениях

7.4. Примеры проверки соответствия

Выполненные задания	Оценка	Проверка критериев
Только базовый	3	✔ Цикл for, ✔

Выполненные задания	Оценка	Проверка критериев
вариант №1 (y=2x+3)		таблица, ✔ проверка a≤b. Но: ✘ нет форматирования, ✘ нет поиска min/max, ✘ нет обработки особых точек
Базовый №1 + средний №2 (cos(x) + min/max)	4	✔ Плюс поиск min/max, ✔ форматирование, ✔ осмысленные имена. Но: ✘ нет суммы ряда, ✘ нет обработки cin.fail()
Базовый №1 + средний №2 + повышенный №1 (sin + ряд Тейлора)	5	✔ Все критерии: ✔ сумма ряда с точностью, ✔ рекуррентная формула, ✔ сравнение с sin(), ✔ обработка ошибок ввода, ✔ подробные комментарии, ✔ блок-схема

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	Влияние	Как исправить
Бесконечный цикл (например, for (double x=a; x<=b; x+=h) с h=0)	Программа зависает	Проверять h > 0

Ошибка	Влияние	Как исправить
Сравнение double через ==	Неправильное число итераций	Использовать x <= b + 1e-9
Прямое сравнение x == b	Может не сработать из-за погрешностей	Не использовать == для double
Целочисленное деление в формуле (1/n)	Неправильные значения	Использовать 1.0 / n
Отсутствие проверки a ≤ b	Аварийное завершение или ошибка	Добавить if (a > b)
Неправильная инициализация min/max	Неверный результат	Инициализировать первым значением
Забыт break при выходе из бесконечного цикла	Бесконечный цикл	Проверить условие выхода
Отсутствие комментариев о формуле ряда	Снижение на 1 балл	Пояснить рекуррентную формулу

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Разницу между for, while и do-while

Почему в табуляции используется x <= b + 1e-9

Как работает аккумулятор суммы

(для оценки 4) — как найти min/max

(для оценки 5) — как работает рекуррентная формула для суммы ряда

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Как изменить код, чтобы шаг был отрицательным (от b до a)?"	Понимание направления обхода
"Что будет, если h > b-a?"	Понимание граничных условий
"Как вычислить сумму ряда с точностью 1e-6 без использования pow и факториала?"	Понимание рекуррентных формул
"Почему сумма ряда может отличаться от библиотечной функции?"	Понимание погрешностей вычислений

Лабораторная работа №6

Одномерные массивы

1. Цель работы

Научиться работать с одномерными статическими массивами: объявлять, инициализировать, заполнять с клавиатуры и случайными

числами. Освоить базовые алгоритмы обработки массивов: поиск суммы, среднего арифметического, минимального и максимального элементов, подсчёт элементов, удовлетворяющих условию. Получить навыки работы с генератором случайных чисел и форматированного вывода массивов.

2. Теоретическое обоснование

2.1. Понятие массива

Массив — это структура данных, представляющая собой набор элементов одного типа, расположенных в памяти последовательно (непрерывным блоком). Каждый элемент массива имеет индекс — целое число, определяющее его позицию.

Визуализация массива в памяти:

```
int arr[5] = {10, 20, 30, 40, 50};

Адрес:  0x100  0x104  0x108  0x10C  0x110
        +-------+-------+-------+-------+-------+
        |  10   |  20   |  30   |  40   |  50   |
        +-------+-------+-------+-------+-------+
Индекс:   0      1      2      3      4
```

Ключевые свойства массива:

Все элементы имеют один и тот же тип

Размер массива фиксирован после объявления (для статических массивов)

Элементы располагаются в памяти последовательно

Доступ к элементу осуществляется по индексу (начиная с 0)

Время доступа к любому элементу — O(1) (константное)

2.2. Объявление и инициализация статического массива

Синтаксис объявления:

тип имя_массива[размер];

Примеры:

```
int numbers[10];        // массив из 10 целых чисел (не инициализирован)
double values[5];       // массив из 5 вещественных чисел
char symbols[20];       // массив из 20 символов
bool flags[8];          // массив из 8 логических значений
```

Инициализация при объявлении:

```
// Полная инициализация
int arr1[5] = {1, 2, 3, 4, 5};

// Частичная инициализация (остальные элементы обнуляются)
int arr2[5] = {1, 2};        // arr2 = {1, 2, 0, 0, 0}

// Без указания размера (размер определяется автоматически)
int arr3[] = {1, 2, 3, 4, 5}; // размер = 5

// Инициализация всех элементов нулями
int arr4[10] = {0};           // все 10 элементов равны 0

// Инициализация с помощью uniform initialization (C++11)
int arr5[5] {1, 2, 3, 4, 5};
```

Важные правила:

1. Размер массива должен быть константой, известной на этапе компиляции
2. Нельзя использовать переменную для размера статического массива

```
const int SIZE = 10;     // правильно: константа
int arr[SIZE];           // OK
```

```
int n;
std::cin >> n;
int arr2[n];              // НЕПРАВИЛЬНО! (в стандартном C++ — VLA, не поддерживается)
// Для динамического размера используйте std::vector или dynamic memory
```

2.3. Доступ к элементам массива

Доступ к элементу осуществляется через оператор [] с указанием индекса.

```
int arr[5] = {10, 20, 30, 40, 50};

arr[0] = 100;          // изменение первого элемента
int x = arr[2];        // чтение третьего элемента (x = 30)

std::cout << arr[4];   // вывод пятого элемента (50)
```

Важно: Индексы массива начинаются с 0 и заканчиваются на размер-1. Выход за границы массива (index out of bounds) приводит к неопределённому поведению — программа может работать неправильно или аварийно завершиться.

```
int arr[5] = {1, 2, 3, 4, 5};

// ПРАВИЛЬНО: индексы от 0 до 4
for (int i = 0; i < 5; i++) {
    std::cout << arr[i] << " ";
}

// НЕПРАВИЛЬНО: выход за границы
arr[5] = 100;  // неопределённое поведение!
```

```
arr[-1] = 10;   // неопределённое поведение!
```

2.4. Заполнение массива

2.4.1. Заполнение с клавиатуры

```
const int SIZE = 5;
int arr[SIZE];

std::cout << "Введите " << SIZE << " элементов: ";
for (int i = 0; i < SIZE; i++) {
    std::cin >> arr[i];
}
```

2.4.2. Заполнение случайными числами

Для генерации случайных чисел используется библиотека <cstdlib> (srand/rand) или <random> (современный способ).

Способ 1: rand() (простой, для учебных целей)

cpp

```
#include <cstdlib>   // для rand(), srand()
#include <ctime>     // для time()

const int SIZE = 10;
int arr[SIZE];

// Инициализация генератора случайных чисел текущим временем
std::srand(static_cast<unsigned>(std::time(nullptr)));

// Заполнение случайными числами от 0 до 99
for (int i = 0; i < SIZE; i++) {
    arr[i] = std::rand() % 100;   // остаток от деления на 100 → 0..99
}
```

Диапазоны случайных чисел:

```
// Случайное число от 0 до (max-1)
int r1 = rand() % max;

// Случайное число от min до max включительно
int r2 = min + rand() % (max - min + 1);

// Случайное вещественное число от 0.0 до 1.0
double r3 = static_cast<double>(rand()) / RAND_MAX;
```

Способ 2: std::mt19937 (современный C++, рекомендуется)

```
#include <random>

std::random_device rd;
std::mt19937 gen(rd());
std::uniform_int_distribution<> dis(1, 100);  // диапазон 1..100

for (int i = 0; i < SIZE; i++) {
    arr[i] = dis(gen);
}
```

2.5. Вывод массива на экран

```
// Вывод в строку
for (int i = 0; i < SIZE; i++) {
    std::cout << arr[i] << " ";
}
```

```
std::cout << std::endl;

// Вывод с индексами
for (int i = 0; i < SIZE; i++) {
    std::cout << "arr[" << i << "] = " << arr[i] << std::endl;
}

// Форматированный вывод
for (int i = 0; i < SIZE; i++) {
    std::cout << std::setw(4) << arr[i];
}
std::cout << std::endl;
```

2.6. Базовые алгоритмы обработки массивов

2.6.1. Сумма всех элементов

```
int sum = 0;
for (int i = 0; i < SIZE; i++) {
    sum += arr[i];          // sum = sum + arr[i]
}
std::cout << "Сумма: " << sum << std::endl;
```

2.6.2. Среднее арифметическое

```
double average = static_cast<double>(sum) / SIZE;
// или
double average = 0.0;
for (int i = 0; i < SIZE; i++) {
    average += arr[i];
```

```
}
average /= SIZE;
```

Важно: Используйте static_cast<double>(sum) или делите на SIZE как вещественное число (/ static_cast<double>(SIZE) или / (double)SIZE).

2.6.3. Поиск минимального и максимального элемента

```
// Инициализация первым элементом
int minVal = arr[0];
int maxVal = arr[0];
int minIndex = 0;
int maxIndex = 0;

for (int i = 1; i < SIZE; i++) {   // начинаем с 1, т.к. 0 уже учтён
    if (arr[i] < minVal) {
        minVal = arr[i];
        minIndex = i;
    }
    if (arr[i] > maxVal) {
        maxVal = arr[i];
        maxIndex = i;
    }
}

std::cout << "Минимум: arr[" << minIndex << "] = " << minVal << std::endl;
std::cout << "Максимум: arr[" << maxIndex << "] = " << maxVal << std::endl;
```

2.6.4. Подсчёт элементов, удовлетворяющих условию

```
// Подсчёт чётных чисел
int evenCount = 0;
for (int i = 0; i < SIZE; i++) {
    if (arr[i] % 2 == 0) {
        evenCount++;
    }
}
std::cout << "Количество чётных элементов: " << evenCount << std::endl;

// Подсчёт элементов больше заданного числа
int threshold;
std::cout << "Введите порог: ";
std::cin >> threshold;

int countGreater = 0;
for (int i = 0; i < SIZE; i++) {
    if (arr[i] > threshold) {
        countGreater++;
    }
}
```

2.6.5. Поиск первого вхождения элемента

```
int searchValue;
std::cout << "Что ищем? ";
std::cin >> searchValue;

int foundIndex = -1;   // -1 означает "не найдено"
for (int i = 0; i < SIZE; i++) {
    if (arr[i] == searchValue) {
        foundIndex = i;
```

```
            break;        // выходим из цикла при первом нахождении
        }
    }

    if (foundIndex != -1) {
        std::cout << "Элемент найден на позиции " << foundIndex << std::endl;
    } else {
        std::cout << "Элемент не найден" << std::endl;
    }
```

2.7. Изменение порядка элементов

2.7.1. Реверс массива (обратный порядок)

```
// Реверс: меняем местами arr[i] и arr[SIZE-1-i]
for (int i = 0; i < SIZE / 2; i++) {
    int temp = arr[i];
    arr[i] = arr[SIZE - 1 - i];
    arr[SIZE - 1 - i] = temp;
}
```

Визуализация реверса для SIZE=5:

Исходный: [1, 2, 3, 4, 5]

i=0: меняем arr[0] и arr[4] → [5, 2, 3, 4, 1]

i=1: меняем arr[1] и arr[3] → [5, 4, 3, 2, 1]

i=2: останов (SIZE/2=2) → [5, 4, 3, 2, 1]

2.7.2. Циклический сдвиг

```
// Циклический сдвиг вправо на 1 позицию
int last = arr[SIZE - 1];
for (int i = SIZE - 1; i > 0; i--) {
    arr[i] = arr[i - 1];
}
```

```
arr[0] = last;
```

2.8. Поиск суммы элементов с условием

```
// Сумма положительных элементов
int sumPositive = 0;
for (int i = 0; i < SIZE; i++) {
    if (arr[i] > 0) {
        sumPositive += arr[i];
    }
}

// Сумма элементов на чётных индексах
int sumEvenIndices = 0;
for (int i = 0; i < SIZE; i += 2) {   // i = 0, 2, 4, ...
    sumEvenIndices += arr[i];
}
```

2.9. Использование const с массивами

Если массив не должен изменяться в функции, его можно передавать как константный:

cpp

```
void printArray(const int arr[], int size) {
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
        // arr[i] = 10;   // ОШИБКА! изменить нельзя
    }
}
```

2.10. std::array (альтернатива статическим массивам C++11)

```cpp
#include <array>

std::array<int, 5> arr = {1, 2, 3, 4, 5};

// Доступ к элементам
arr[0] = 10;
int x = arr.at(2);    // с проверкой границ (бросает исключение)

// Размер
int size = arr.size();

// Итераторы
for (int val : arr) {    // range-based for (C++11)
    std::cout << val << " ";
}
```

2.11. Типичные ошибки при работе с массивами

Ошибка	Пример	Последствие
Выход за границы	arr[SIZE] = 10	Неопределённое поведение
Неинициализированный массив	int arr[5]; int x = arr[0];	Мусорное значение

Ошибка	Пример	Последствие
Размер не константа	int n; cin>>n; int arr[n];	Не поддерживается (VLA)
Сравнение указателя вместо элемента	if (arr > 10)	Сравнение адреса, а не значения
sizeof с массивом в функции	sizeof(arr) в функции	Получится размер указателя, не массива
Путаница с индексацией (с 0)	for (int i=1; i<=SIZE; i++)	Выход за границы

```
// ОШИБКА: неправильное использование sizeof в функции
void printSize(int arr[]) {
    std::cout << sizeof(arr);  // результат = 8 (размер указателя)!
}

// ПРАВИЛЬНО: передавать размер отдельно
void printSize(int arr[], int size) {
    std::cout << size * sizeof(int);
}

// В main: sizeof работает правильно
```

```
int main() {
    int arr[10];
    std::cout << sizeof(arr);  // 40 байт (10 * 4)
}
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Заполнение и вывод массива (базовый уровень)

```
//
==========================================================================
=
// Программа: заполнение массива с клавиатуры и вывод
// Цель: демонстрация базовых операций с массивом
//
==========================================================================
=

#include <iostream>
#include <iomanip>

int main() {
    const int SIZE = 10;      // Размер массива (константа)
    int arr[SIZE];            // Объявление массива

    // ========== ЗАПОЛНЕНИЕ МАССИВА ==========
    std::cout << "Введите " << SIZE << " целых чисел:" << std::endl;
    for (int i = 0; i < SIZE; i++) {
        std::cout << "arr[" << i << "] = ";
        std::cin >> arr[i];
```

```
    }

    // ========== ВЫВОД МАССИВА ==========
    std::cout << "\nВведённый массив:" << std::endl;
    for (int i = 0; i < SIZE; i++) {
        std::cout << std::setw(4) << arr[i];
    }
    std::cout << std::endl;

    // ========== ВЫВОД В ОБРАТНОМ ПОРЯДКЕ ==========
    std::cout << "\nМассив в обратном порядке:" << std::endl;
    for (int i = SIZE - 1; i >= 0; i--) {
        std::cout << std::setw(4) << arr[i];
    }
    std::cout << std::endl;

    return 0;
}

// Пример работы:
// Ввод: 10 20 30 40 50 60 70 80 90 100
// Вывод:  10  20  30  40  50  60  70  80  90 100
// Обратный: 100  90  80  70  60  50  40  30  20  10
```

3.2. Пример 2: Случайные числа + сумма + среднее (средний уровень)

```
//
==========
=
```

```
// Программа: заполнение массива случайными числами
// Вычисление: сумма, среднее, минимум, максимум
// Цель: демонстрация базовых алгоритмов обработки массива
//
================================================================
=

#include <iostream>
#include <iomanip>
#include <cstdlib>   // для rand(), srand()
#include <ctime>     // для time()

int main() {
    const int SIZE = 15;      // Размер массива
    int arr[SIZE];            // Объявление массива

    // ========== ИНИЦИАЛИЗАЦИЯ ГЕНЕРАТОРА СЛУЧАЙНЫХ
ЧИСЕЛ ==========
    std::srand(static_cast<unsigned>(std::time(nullptr)));
    // static_cast<unsigned> - явное приведение типа time_t к unsigned

    // ========== ЗАПОЛНЕНИЕ СЛУЧАЙНЫМИ ЧИСЛАМИ ОТ 0
ДО 99 ==========
    std::cout << "Генерация " << SIZE << " случайных чисел..." <<
std::endl;
    for (int i = 0; i < SIZE; i++) {
        arr[i] = std::rand() % 100;   // случайное число 0..99
    }
```

```
    // ========== ВЫВОД МАССИВА ==========
    std::cout << "\nСгенерированный массив:" << std::endl;
    for (int i = 0; i < SIZE; i++) {
        std::cout << std::setw(4) << arr[i];
        if ((i + 1) % 10 == 0) {      // каждые 10 элементов — перенос строки
            std::cout << std::endl;
        }
    }
    std::cout << std::endl;

    // ========== ВЫЧИСЛЕНИЕ СУММЫ ==========
    int sum = 0;
    for (int i = 0; i < SIZE; i++) {
        sum += arr[i];
    }

    // ========== ВЫЧИСЛЕНИЕ СРЕДНЕГО АРИФМЕТИЧЕСКОГО ==========
    // ВАЖНО: используем static_cast<double> для вещественного деления
    double average = static_cast<double>(sum) / SIZE;

    // ========== ПОИСК МИНИМУМА И МАКСИМУМА ==========
    int minVal = arr[0];
    int maxVal = arr[0];
    int minIndex = 0;
    int maxIndex = 0;
```

```
        for (int i = 1; i < SIZE; i++) {
            if (arr[i] < minVal) {
                minVal = arr[i];
                minIndex = i;
            }
            if (arr[i] > maxVal) {
                maxVal = arr[i];
                maxIndex = i;
            }
        }

        // ========== ВЫВОД РЕЗУЛЬТАТОВ ==========
        std::cout << std::fixed << std::setprecision(2);
        std::cout << "\n=== РЕЗУЛЬТАТЫ ===" << std::endl;
        std::cout << "Сумма элементов: " << sum << std::endl;
        std::cout << "Среднее арифметическое: " << average << std::endl;
        std::cout << "Минимальный элемент: arr[" << minIndex << "] = " <<
minVal << std::endl;
        std::cout << "Максимальный элемент: arr[" << maxIndex << "] = " <<
maxVal << std::endl;

        return 0;
    }
```

3.3. Пример 3: Подсчёт элементов по условию (средний уровень)

```
//
==========================================================
=
// Программа: подсчёт элементов, удовлетворяющих условию
// Условия: чётные числа, числа больше среднего, числа в диапазоне
```

```
// Цель: демонстрация фильтрации элементов массива
//
==========================================================
=

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <ctime>

int main() {
    const int SIZE = 20;
    int arr[SIZE];

    // Заполнение случайными числами от -50 до 50
    std::srand(static_cast<unsigned>(std::time(nullptr)));
    for (int i = 0; i < SIZE; i++) {
        arr[i] = -50 + std::rand() % 101;   // -50..50
    }

    // Вывод массива
    std::cout << "Массив:" << std::endl;
    for (int i = 0; i < SIZE; i++) {
        std::cout << std::setw(5) << arr[i];
        if ((i + 1) % 10 == 0) std::cout << std::endl;
    }
    std::cout << std::endl;

    // ========== 1. Количество чётных элементов ==========
```

```
        int evenCount = 0;
        for (int i = 0; i < SIZE; i++) {
            if (arr[i] % 2 == 0) {
                evenCount++;
            }
        }
        std::cout << "Количество чётных элементов: " << evenCount << std::endl;

        // ========== 2. Количество положительных элементов ==========
        int positiveCount = 0;
        for (int i = 0; i < SIZE; i++) {
            if (arr[i] > 0) {
                positiveCount++;
            }
        }
        std::cout << "Количество положительных элементов: " << positiveCount << std::endl;

        // ========== 3. Количество элементов, кратных 3 ==========
        int multipleOf3Count = 0;
        for (int i = 0; i < SIZE; i++) {
            if (arr[i] % 3 == 0 && arr[i] != 0) {  // не считаем 0
                multipleOf3Count++;
            }
        }
        std::cout << "Количество элементов, кратных 3: " << multipleOf3Count << std::endl;
```

```
    // ========== 4. Элементы больше заданного порога ==========
    int threshold;
    std::cout << "\nВведите пороговое значение: ";
    std::cin >> threshold;

    int greaterCount = 0;
    std::cout << "Элементы больше " << threshold << ": ";
    for (int i = 0; i < SIZE; i++) {
        if (arr[i] > threshold) {
            std::cout << arr[i] << " ";
            greaterCount++;
        }
    }
    std::cout << "\nКоличество: " << greaterCount << std::endl;

    return 0;
}
```

3.4. Пример 4: Поиск элемента и реверс массива (повышенный уровень)

```
//
==========
=

// Программа: поиск элемента, реверс массива, циклический сдвиг
// Цель: демонстрация продвинутых алгоритмов обработки массивов
//
==========
=

#include <iostream>
```

```cpp
#include <iomanip>
#include <cstdlib>
#include <ctime>

// Функция для вывода массива (для удобства)
void printArray(const int arr[], int size, const std::string& title) {
    std::cout << title << ": ";
    for (int i = 0; i < size; i++) {
        std::cout << std::setw(4) << arr[i];
    }
    std::cout << std::endl;
}

int main() {
    const int SIZE = 10;
    int arr[SIZE];

    // Заполнение случайными числами от 1 до 100
    std::srand(static_cast<unsigned>(std::time(nullptr)));
    for (int i = 0; i < SIZE; i++) {
        arr[i] = 1 + std::rand() % 100;
    }

    printArray(arr, SIZE, "Исходный массив");

    // ========== 1. ПОИСК ЭЛЕМЕНТА ==========
    int searchValue;
    std::cout << "\nВведите число для поиска: ";
    std::cin >> searchValue;
```

```
    int foundIndex = -1;
    for (int i = 0; i < SIZE; i++) {
        if (arr[i] == searchValue) {
            foundIndex = i;
            break;  // выход при первом нахождении
        }
    }

    if (foundIndex != -1) {
        std::cout << "Элемент " << searchValue << " найден на позиции " << foundIndex << std::endl;
    } else {
        std::cout << "Элемент " << searchValue << " не найден" << std::endl;
    }

    // ========== 2. РЕВЕРС МАССИВА ==========
    // Создаём копию для реверса (чтобы не повредить исходный массив)
    int reversed[SIZE];
    for (int i = 0; i < SIZE; i++) {
        reversed[i] = arr[SIZE - 1 - i];
    }
    printArray(reversed, SIZE, "Реверс массива");

    // ========== 3. ЦИКЛИЧЕСКИЙ СДВИГ ВПРАВО ==========
    int shiftArr[SIZE];
    for (int i = 0; i < SIZE; i++) {
        shiftArr[i] = arr[i];
    }
```

```
    int last = shiftArr[SIZE - 1];
    for (int i = SIZE - 1; i > 0; i--) {
        shiftArr[i] = shiftArr[i - 1];
    }
    shiftArr[0] = last;
    printArray(shiftArr, SIZE, "Сдвиг вправо на 1");

    // ========== 4. ЦИКЛИЧЕСКИЙ СДВИГ ВЛЕВО ==========
    int shiftLeft[SIZE];
    for (int i = 0; i < SIZE; i++) {
        shiftLeft[i] = arr[i];
    }

    int first = shiftLeft[0];
    for (int i = 0; i < SIZE - 1; i++) {
        shiftLeft[i] = shiftLeft[i + 1];
    }
    shiftLeft[SIZE - 1] = first;
    printArray(shiftLeft, SIZE, "Сдвиг влево на 1");

    return 0;
}

// Пример работы:
// Исходный массив: [23, 56, 12, 89, 45, 67, 34, 78, 90, 11]
// Реверс: [11, 90, 78, 34, 67, 45, 89, 12, 56, 23]
// Сдвиг вправо: [11, 23, 56, 12, 89, 45, 67, 34, 78, 90]
// Сдвиг влево: [56, 12, 89, 45, 67, 34, 78, 90, 11, 23]
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Заполнить массив (с клавиатуры или случайными) и выполнить одно простое вычисление.

	Способ заполнения	Задание
	С клавиатуры	Найти сумму всех элементов массива
	С клавиатуры	Найти среднее арифметическое элементов
	Случайные числа (0..50)	Найти минимальный элемент
	Случайные числа (0..50)	Найти максимальный элемент
	С клавиатуры	Найти количество чётных элементов
	С клавиатуры	Найти количество нечётных элементов
	Случайные числа (-30..30)	Найти количество положительных элементов
	Случайные числа (-30..30)	Найти количество отрицательных элементов
	С клавиатуры	Найти сумму элементов на чётных индексах

	Способ заполнения	Задание
0	С клавиатуры	Найти сумму элементов на нечётных индексах
1	Случайные числа (10..99)	Найти количество элементов больше 50
2	Случайные числа (10..99)	Найти количество элементов меньше 30
3	С клавиатуры	Найти произведение всех элементов (с проверкой на ноль)
4	С клавиатуры	Найти индекс первого нулевого элемента (или -1, если нет)
5	Случайные числа (0..100)	Найти количество нулевых элементов

Средний уровень (оценка «хорошо»)

Заполнить массив случайными числами и выполнить комплексную обработку.

	Размер	Диапазон	Задание
	15	0..100	Найти сумму, среднее, минимум, максимум
	20	-50..50	Найти сумму положительных и сумму отрицательных элементов

	Размер	Диапазон	Задание
	15	1..100	Найти количество элементов, кратных 3 и 5
	20	10..99	Найти количество двузначных чисел
	15	-100..100	Найти количество элементов, попадающих в интервал [a, b] (a, b вводятся)
	20	1..100	Найти разность между максимальным и минимальным элементами
	15	-50..50	Найти индекс минимального и индекс максимального элемента
	20	0..100	Подсчитать, сколько элементов больше среднего арифметического
	15	1..50	Найти сумму элементов, стоящих между первым и последним нулевыми элементами
0	20	-99..99	Найти количество элементов, являющихся точными квадратами (1,4,9,16,25,36,49,64,81)
1	15	0..100	Найти, сколько раз в массиве встречается максимальный элемент
	2	-	Найти произведение положительных

	Размер	Диапазон	Задание
2	10	-100..100	элементов и сумму отрицательных
3	15	1..100	Найти второй по величине элемент (второй максимум)
4	20	-50..50	Найти количество локальных минимумов (элемент меньше соседей)
5	15	0..100	Вывести массив в обратном порядке (не меняя исходный)

Повышенный уровень (оценка «отлично»)

Сложная обработка, реверс, сдвиги, удаление/вставка элементов.

	Размер	Диапазон	Задание
	20	-100..100	Реверс массива без использования дополнительного массива (меняя элементы местами)
	20	1..100	Циклический сдвиг вправо на K позиций (K вводится пользователем)
	20	1..100	Циклический сдвиг влево на K позиций (K вводится пользователем)
	1	0..1	Удаление элемента по индексу (сдвигом

	Размер	Диапазон	Задание
	5	00	влево, размер уменьшается)
	15	0..100	Вставка элемента по индексу (сдвигом вправо, размер увеличивается)
	20	-50..50	Сортировка пузырьком (по возрастанию) — вывести массив до и после
	20	-50..50	Сортировка выбором (по убыванию) — вывести массив до и после
	25	0..200	Поиск всех пар элементов, сумма которых равна заданному числу
	20	1..100	Проверка, является ли массив палиндромом (читается одинаково слева и справа)
0	30	0..100	Удаление всех нулевых элементов (создать новый массив без нулей)
1	20	1..100	Вычисление медианы (среднего элемента отсортированного массива)
2	15	1..20	Построение гистограммы (частоты встречаемости чисел от 1 до 20)
	2	0..1	Подсчёт количества вхождений каждого

	Размер	Диапазон	Задание
3	0	0	числа (от 0 до 10) с выводом в виде таблицы
4	20	0..100	Найти самую длинную возрастающую последовательность и её длину
5	25	-100..100	Найти все локальные экстремумы (пики и впадины) и их индексы

5. Контрольные вопросы

Раздел А. Основы массивов

Что такое массив? Какие свойства массивов вы знаете?

Как объявить статический массив из 20 целых чисел? Приведите пример.

Как проинициализировать массив при объявлении? Покажите 3 способа.

Почему размер статического массива должен быть константой?

Как получить доступ к элементу массива? Что произойдёт при выходе за границы?

Раздел Б. Заполнение и вывод

Как заполнить массив с клавиатуры? Напишите фрагмент кода.

Как заполнить массив случайными числами? Какие библиотеки нужны?

Как получить случайное число в диапазоне от min до max включительно?

Как вывести массив на экран в строку? А в столбец?

Что такое генератор случайных чисел? Зачем нужен srand(time(nullptr))?

Раздел В. Базовые алгоритмы

Как найти сумму всех элементов массива? Напишите алгоритм.

Как найти среднее арифметическое? Почему важно приведение типов?

Как найти минимальный элемент? С какого значения начинать поиск?

Как найти количество элементов, удовлетворяющих условию?

Как найти индекс первого вхождения элемента?

Раздел Г. Продвинутые алгоритмы

Как выполнить реверс массива без использования второго массива?

Как выполнить циклический сдвиг элементов вправо? А влево?

Как удалить элемент из массива по индексу? Что происходит с размером?

Как вставить элемент в массив по индексу?

Что такое сортировка пузырьком? Как она работает?

Раздел Д. Анализ кода и ошибки

Что выведет код?

```
int arr[5] = {1, 2, 3};
for (int i = 0; i < 5; i++) {
    std::cout << arr[i] << " ";
}
```

Найдите ошибку:

```
int n;
std::cin >> n;
int arr[n];
```

Что произойдёт?

```
int arr[5] = {1, 2, 3, 4, 5};
for (int i = 0; i <= 5; i++) {
```

```
    arr[i] = 0;
}
```

Как исправить код для вычисления среднего?

```
int sum = 0;
for (int i = 0; i < SIZE; i++) {
    sum += arr[i];
}
double avg = sum / SIZE;
```

Чем отличается статический массив от std::array?

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (определение массива, объявление, инициализация, доступ к элементам, базовые алгоритмы)

Ход выполнения работы:

Код программы с подробными комментариями (каждый алгоритм пояснён)

Блок-схема алгоритма (для одного из заданий)

Скриншоты выполнения для различных сценариев (разные размеры массива, разные диапазоны)

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали о массивах, какие алгоритмы освоили)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №6)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) — правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, правильное объявление массива, использование цикла для заполнения и обработки).

Оценка «хорошо» (4) — правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего уровня (заполнение случайными числами, комплексная обработка, поиск суммы/среднего/min/max, осмысленные имена переменных).

Оценка «отлично» (5) — правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (реверс на месте, циклический сдвиг на K, удаление/вставка элементов, сортировка, полная защита от некорректного ввода, подробные комментарии в коде, блок-схема в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
	Компиляция и запуск			
.1	Программа компилируется без ошибок	✔	✔	✔
.2	Программа успешно запускается	✔	✔	✔
.3	Отсутствие предупреждений компилятора	—	✔	✔
	Объявление и инициализация массива			
.1	Правильное объявление статического массива	✔	✔	✔
.2	Использование константы для размера	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.3	Корректная инициализация (где требуется)	✔	✔	✔
	Заполнение массива			
.1	Заполнение с клавиатуры (где требуется)	✔	✔	✔
.2	Заполнение случайными числами	—	✔	✔
.3	Правильное использование srand() и rand()	—	✔	✔
	Вывод массива			
.1	Вывод всех элементов массива	✔	✔	✔
.2	Форматированный вывод (столбцы, выравнивание)	—	✔	✔
	Базовые алгоритмы обработки			

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.1	Вычисление суммы элементов	—	✔	✔
.2	Вычисление среднего арифметического	—	✔	✔
.3	Поиск минимума и максимума	—	✔	✔
.4	Подсчёт элементов по условию	✔	✔	✔
	Продвинутые алгоритмы			
.1	Реверс массива (без доп. массива)	—	—	✔
.2	Циклический сдвиг на K позиций	—	—	✔
.3	Удаление/вставка элементов	—	—	✔
.4	Сортировка массива	—	—	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Обработка ошибок			
.1	Проверка корректности ввода пользователя	—	✔	✔
.2	Проверка на ввод букв (std::cin.fail())	—	—	✔
.3	Контроль границ массива (индексы)	✔	✔	✔
	Качество кода			
.1	Правильные отступы и форматирование	✔	✔	✔
.2	Использование осмысленных имён	—	✔	✔
.3	Отсутствие «магических чисел»	—	✔	✔
.4	Наличие комментариев в коде	—	✔	✔
	Отчёт			

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
0				
0.1	Наличие титульного листа, цели, хода работы	✔	✔	✔
0.2	Скриншоты выполнения (минимум 3 теста)	✔	✔	✔
0.3	Скриншоты для всех трёх уровней	—	✔	✔
0.4	Таблица тестирования	—	✔	✔
0.5	Блок-схема алгоритма	—	✔	✔
0.6	Тестирование на граничных значениях	—	—	✔
0.7	Ответы на контрольные вопросы	—	✔	✔

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) — обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня (№1–15).

Правильно объявить статический массив с константным размером.

Заполнить массив (с клавиатуры или случайными числами согласно заданию).

Выполнить требуемое вычисление (сумма, среднее, min, max, подсчёт).

Вывести результат на экран.

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) — дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Заполнить массив случайными числами в заданном диапазоне.

Выполнить комплексную обработку (минимум 3 различных операции: сумма, среднее, min/max, подсчёт по условию).

Вывести массив в форматированном виде (столбцы, выравнивание).

Использовать осмысленные имена переменных.

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

Блок-схему алгоритма

Повышенный уровень (оценка 5) — требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Реализовать один из сложных алгоритмов:

Реверс массива на месте (без дополнительного массива)

Циклический сдвиг на K позиций

Удаление или вставка элемента

Сортировка (пузырьком или выбором)

Обработать все возможные ошибки:

Проверка std::cin.fail() при вводе чисел

Контроль границ массива при вставке/удалении

Обработка случая пустого массива

Продемонстрировать понимание индексации:

Корректная работа с индексами в сложных алгоритмах

Понимание разницы между значением и индексом

Написать подробные комментарии:

Объяснение логики каждого алгоритма

Пояснение выбора типа цикла (for, while)

В отчёте дополнительно представить:

Тестирование на граничных значениях (минимальный/максимальный размер, крайние индексы)

7.4. Примеры проверки соответствия

Выполненные задания	Оценка	Проверка критериев
Только базовый вариант №1 (сумма элементов)	3	✔ Объявление массива, ✔ ввод с клавиатуры, ✔ сумма, ✘ нет случайных чисел, ✘ нет форматирования
Базовый №1 + средний №1 (сумма/среднее/min/max)	4	✔ Плюс случайные числа, ✔ min/max, ✔ форматирование. Но: ✘ нет реверса/сдвига/сортировки
Базовый №1 + средний №1 + повышенный №2 (сдвиг на K)	5	✔ Все критерии: ✔ сдвиг вправо/влево, ✔ обработка K, ✔ защита от

Выполненные задания	Оценка	Проверка критериев
		ошибок, ✔ комментарии, ✔ блок-схема

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	Влияние	Как исправить
Выход за границы массива (index out of bounds)	Неопределённое поведение	Проверять индексы: i < SIZE
Использование переменной для размера статического массива	Ошибка компиляции	Использовать const int SIZE
Неправильная инициализация srand()	Одинаковые числа при каждом запуске	Вызвать srand(time(nullptr)) один раз
Деление int на int при вычислении среднего	Потеря дробной части	Использовать static_cast<double>(sum)/SIZE
Сравнение с rand() % range с отрицательными числами	Неправильный диапазон	Использовать min + rand() % (max - min + 1)

Ошибка	Влияние	Как исправить
Неинициализированный массив при частичной инициализации	Остальные элементы = 0 (это нормально), но нужно знать	Если нужно иначе — заполнить явно
sizeof(arr) в функции	Получается размер указателя, не массива	Передавать размер отдельным параметром

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Как объявляется и заполняется массив

Как выполняется базовый алгоритм обработки (сумма, min, max)

Что такое srand() и зачем он нужен

(для оценки 4) — как найти среднее и почему важно приведение типов

(для оценки 5) — логику реверса, сдвига или сортировки

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Что произойдёт, если ввести размер массива больше объявленного?"	Понимание границ массива
"Как изменить код, чтобы найти второй минимум?"	Понимание алгоритмов поиска
"Почему при сдвиге элементов нужно сохранять последний элемент во временную переменную?"	Понимание перезаписи данных
"Чем отличается реверс с дополнительным массивом от реверса на месте?"	Понимание эффективности по памяти

Лабораторная работа №7

Двумерные массивы (матрицы)

1. Цель работы

Научиться работать с двумерными статическими массивами (матрицами): объявлять, инициализировать, заполнять с клавиатуры и случайными числами, выводить в виде таблицы. Освоить базовые алгоритмы обработки матриц: вычисление суммы элементов по строкам и столбцам, поиск минимума и максимума, работа с главной и побочной диагоналями, транспонирование матрицы, умножение матриц.

2. Теоретическое обоснование

2.1. Понятие двумерного массива

Двумерный массив (матрица) — это структура данных, представляющая собой таблицу (набор строк и столбцов), где каждый элемент имеет два индекса: номер строки и номер столбца. В памяти двумерный массив хранится как непрерывная последовательность элементов (построчно — row-major order).

Визуализация матрицы 3×4:

text

	Столбец 0	Столбец 1	Столбец 2	Столбец 3
Строка 0	[0][0]	[0][1]	[0][2]	[0][3]
Строка 1	[1][0]	[1][1]	[1][2]	[1][3]
Строка 2	[2][0]	[2][1]	[2][2]	[2][3]

Расположение в памяти (row-major order):

[0][0] [0][1] [0][2] [0][3] [1][0] [1][1] [1][2] [1][3] [2][0] [2][1] [2][2] [2][3]
<----- строка 0 -----> <----- строка 1 -----> <----- строка 2 ----->

2.2. Объявление и инициализация статической матрицы

Синтаксис объявления:

тип имя_матрицы[количество_строк][количество_столбцов];

Примеры:

```
int matrix[3][4];       // 3 строки, 4 столбца (не инициализирована)
double table[5][5];     // квадратная матрица 5×5
char grid[10][10];      // символьная сетка 10×10
```

Инициализация при объявлении:

```
// Полная инициализация
int matrix[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};

// Неполная инициализация (остальные элементы обнуляются)
int matrix[2][3] = {
    {1, 2},      // {1, 2, 0}
    {4}          // {4, 0, 0}
};

// Инициализация без указания размеров (размер определяется автоматически)
int matrix[][3] = {
    {1, 2, 3},
    {4, 5, 6}
};  // 2 строки, 3 столбца

// Инициализация всех элементов нулями
int matrix[5][5] = {0};

// C++11 uniform initialization
int matrix[2][3] {
    {1, 2, 3},
    {4, 5, 6}
};
```

Важные правила:

Первый размер (количество строк) может быть опущен при инициализации

Второй размер (количество столбцов) обязателен всегда

Размеры должны быть константами, известными на этапе компиляции

```
const int ROWS = 3;
const int COLS = 4;
int matrix[ROWS][COLS];     // правильно

int n, m;
std::cin >> n >> m;
int matrix2[n][m];          // НЕПРАВИЛЬНО! (VLA не поддерживается)
```

2.3. Доступ к элементам матрицы

Доступ к элементу осуществляется через оператор [][] с указанием номера строки и столбца.

```
int matrix[3][4] = {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
};

matrix[0][0] = 100;         // изменение первого элемента
int x = matrix[1][2];       // чтение элемента (строка 1, столбец 2) → 7

std::cout << matrix[2][3];  // вывод элемента (строка 2, столбец 3) → 12
```

Важно: Индексы начинаются с 0. Выход за границы (например, matrix[3][0] при 3 строках) приводит к неопределённому поведению.

2.4. Обход матрицы с помощью вложенных циклов

```
const int ROWS = 3;
const int COLS = 4;
int matrix[ROWS][COLS] = {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
};

// Обход по строкам (внешний цикл — строки, внутренний — столбцы)
for (int i = 0; i < ROWS; i++) {          // перебор строк
    for (int j = 0; j < COLS; j++) {      // перебор столбцов текущей строки
        std::cout << std::setw(4) << matrix[i][j];
    }
    std::cout << std::endl;               // переход на следующую строку
}

// Вывод:
//    1   2   3   4
//    5   6   7   8
//    9  10  11  12
```

Обход по столбцам:

```
// Внешний цикл — по столбцам, внутренний — по строкам
for (int j = 0; j < COLS; j++) {
    for (int i = 0; i < ROWS; i++) {
        std::cout << std::setw(4) << matrix[i][j];
```

```
    }
    std::cout << std::endl;
}
```

2.5. Заполнение матрицы

2.5.1. Заполнение с клавиатуры

```
const int ROWS = 3;
const int COLS = 3;
int matrix[ROWS][COLS];

std::cout << "Введите элементы матрицы " << ROWS << "x" << COLS << ":" << std::endl;
for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
        std::cout << "matrix[" << i << "][" << j << "] = ";
        std::cin >> matrix[i][j];
    }
}
```

2.5.2. Заполнение случайными числами

```
#include <cstdlib>
#include <ctime>

const int ROWS = 4;
const int COLS = 5;
int matrix[ROWS][COLS];

std::srand(static_cast<unsigned>(std::time(nullptr)));

for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
```

```
        matrix[i][j] = std::rand() % 100;  // число от 0 до 99
    }
}
```

2.5.3. Заполнение с формулой (например, единичная матрица)

```
const int SIZE = 4;
int identity[SIZE][SIZE];

for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
        if (i == j) {
            identity[i][j] = 1;   // единицы на главной диагонали
        } else {
            identity[i][j] = 0;   // нули вне диагонали
        }
    }
}
```

2.6. Базовые алгоритмы обработки матриц

2.6.1. Сумма всех элементов

```
int sum = 0;
for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
        sum += matrix[i][j];
    }
}
std::cout << "Сумма всех элементов: " << sum << std::endl;
```

2.6.2. Сумма элементов каждой строки

```
for (int i = 0; i < ROWS; i++) {
    int rowSum = 0;
    for (int j = 0; j < COLS; j++) {
        rowSum += matrix[i][j];
    }
    std::cout << "Сумма строки " << i << ": " << rowSum << std::endl;
}
```

2.6.3. Сумма элементов каждого столбца

```
for (int j = 0; j < COLS; j++) {
    int colSum = 0;
    for (int i = 0; i < ROWS; i++) {
        colSum += matrix[i][j];
    }
    std::cout << "Сумма столбца " << j << ": " << colSum << std::endl;
}
```

2.6.4. Поиск минимального и максимального элемента

```
int minVal = matrix[0][0];
int maxVal = matrix[0][0];
int minRow = 0, minCol = 0;
int maxRow = 0, maxCol = 0;

for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
        if (matrix[i][j] < minVal) {
            minVal = matrix[i][j];
            minRow = i;
            minCol = j;
        }
```

```
            if (matrix[i][j] > maxVal) {
                maxVal = matrix[i][j];
                maxRow = i;
                maxCol = j;
            }
        }
    }
    std::cout << "Минимум: matrix[" << minRow << "][" << minCol << "] = "
<< minVal << std::endl;
    std::cout << "Максимум: matrix[" << maxRow << "][" << maxCol << "] = "
<< maxVal << std::endl;
```

2.7. Работа с диагоналями (для квадратных матриц)

Для квадратной матрицы размером n × n:

2.7.1. Главная диагональ

Элементы, у которых номер строки равен номеру столбца: matrix[i][i]

```
[1] 0  0  0
 0 [2] 0  0
 0  0 [3] 0
 0  0  0 [4]
```

```
// Сумма элементов главной диагонали
int sumMainDiag = 0;
for (int i = 0; i < n; i++) {
    sumMainDiag += matrix[i][i];
}
```

2.7.2. Побочная диагональ

Элементы, у которых сумма индексов равна n-1: matrix[i][n-1-i]

```
 0  0  0 [1]
 0  0 [2] 0
 0 [3] 0  0
[4] 0  0  0
```

```
// Сумма элементов побочной диагонали
int sumSecondaryDiag = 0;
for (int i = 0; i < n; i++) {
    sumSecondaryDiag += matrix[i][n - 1 - i];
}
```

2.7.3. Все элементы выше/ниже главной диагонали

```
// Выше главной диагонали (j > i)
for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
        // элемент matrix[i][j]
    }
}

// Ниже главной диагонали (j < i)
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        // элемент matrix[i][j]
    }
}
```

2.8. Транспонирование матрицы

Транспонирование — это операция, при которой строки матрицы становятся столбцами, а столбцы — строками. Для матрицы A размером ROWS × COLS транспонированная матрица A^T будет иметь размер COLS × ROWS.

Визуализация:

text

Исходная (3×2): Транспонированная (2×3):

[1, 2] [1, 3, 5]

[3, 4] [2, 4, 6]

[5, 6]

```
// Транспонирование в новую матрицу
const int ROWS = 3;
const int COLS = 4;
int matrix[ROWS][COLS];
int transposed[COLS][ROWS];

for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
        transposed[j][i] = matrix[i][j];
    }
}

// Транспонирование квадратной матрицы на месте (без дополнительной матрицы)
const int N = 4;
int square[N][N];
```

```
for (int i = 0; i < N; i++) {
    for (int j = i + 1; j < N; j++) {
        // Меняем местами matrix[i][j] и matrix[j][i]
        int temp = square[i][j];
        square[i][j] = square[j][i];
        square[j][i] = temp;
    }
}
```

2.9. Умножение матриц

Умножение матрицы A размером m × n на матрицу B размером n × k даёт матрицу C размером m × k.

Формула: C[i][j] = Σ(A[i][t] * B[t][j]) для t = 0..n-1

```
// Умножение матриц: A (m×n) * B (n×k) = C (m×k)
const int M = 3, N = 2, K = 4;
int A[M][N];
int B[N][K];
int C[M][K] = {0};  // обнуляем матрицу результата

// Умножение
for (int i = 0; i < M; i++) {
    for (int j = 0; j < K; j++) {
        for (int t = 0; t < N; t++) {
            C[i][j] += A[i][t] * B[t][j];
        }
    }
}
```

2.10. Типичные ошибки при работе с матрицами

Ошибка	Пример	Последствие
Выход за границы строк/столбцов	matrix[ROWS][0]	Неопределённое поведение
Перепутан порядок индексов	matrix[j][i] вместо matrix[i][j]	Неправильный доступ
Неправильный размер при инициализации	int m[2][3] = {{1,2,3,4}}	Ошибка компиляции
Одинаковые имена переменных для счётчиков	for (int i...){ for (int i...}	Ошибка компиляции
Забыто обнуление матрицы результата	int C[M][K]; C[i][j] += ...	Мусорные значения
Неправильная транспонировка на месте	Цикл по всем i, j (меняет дважды)	Исходная матрица восстановится

```
// ОШИБКА: перепутан порядок индексов при выводе
```

```
for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
        std::cout << matrix[j][i];  // НЕПРАВИЛЬНО!
    }
}

// ПРАВИЛЬНО
for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
        std::cout << matrix[i][j];
    }
}
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Заполнение и вывод матрицы (базовый уровень)

```
//
==========================================================
=
// Программа: заполнение матрицы 3×4 с клавиатуры и вывод
// Цель: демонстрация вложенных циклов для работы с матрицами
//
==========================================================
=

#include <iostream>
#include <iomanip>

int main() {
    const int ROWS = 3;     // Количество строк
    const int COLS = 4;     // Количество столбцов
```

```
    int matrix[ROWS][COLS]; // Объявление матрицы

    // ========== ЗАПОЛНЕНИЕ МАТРИЦЫ С КЛАВИАТУРЫ ==========
    std::cout << "Введите элементы матрицы " << ROWS << "×" << COLS << ":" << std::endl;
    for (int i = 0; i < ROWS; i++) {          // внешний цикл — строки
        for (int j = 0; j < COLS; j++) {      // внутренний цикл — столбцы
            std::cout << "matrix[" << i << "][" << j << "] = ";
            std::cin >> matrix[i][j];
        }
    }

    // ========== ВЫВОД МАТРИЦЫ В ВИДЕ ТАБЛИЦЫ ==========
    std::cout << "\nВведённая матрица:" << std::endl;
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            std::cout << std::setw(6) << matrix[i][j];
        }
        std::cout << std::endl;    // переход на новую строку после каждой строки матрицы
    }

    return 0;
}

// Пример работы:
// Ввод: 1 2 3 4 5 6 7 8 9 10 11 12
```

```
// Вывод:
//    1   2   3   4
//    5   6   7   8
//    9  10  11  12
```

3.2. Пример 2: Сумма по строкам и столбцам + случайные числа

```
//
==========================================================
=
// Программа: заполнение матрицы 4×5 случайными числами
// Вычисление: сумма по строкам, сумма по столбцам, общая сумма
// Цель: демонстрация базовой обработки матриц
//
==========================================================
=

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <ctime>

int main() {
    const int ROWS = 4;
    const int COLS = 5;
    int matrix[ROWS][COLS];

    // ========== ЗАПОЛНЕНИЕ СЛУЧАЙНЫМИ ЧИСЛАМИ
==========
```

```
    std::srand(static_cast<unsigned>(std::time(nullptr)));

    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            matrix[i][j] = std::rand() % 100;   // случайные числа от 0 до 99
        }
    }

    // ========== ВЫВОД МАТРИЦЫ ==========
    std::cout << "Сгенерированная матрица " << ROWS << "×" << COLS
<< ":" << std::endl;
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            std::cout << std::setw(4) << matrix[i][j];
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;

    // ========== СУММА ПО СТРОКАМ ==========
    std::cout << "Суммы по строкам:" << std::endl;
    for (int i = 0; i < ROWS; i++) {
        int rowSum = 0;
        for (int j = 0; j < COLS; j++) {
            rowSum += matrix[i][j];
        }
        std::cout << "  Строка " << i << ": " << rowSum << std::endl;
    }
    std::cout << std::endl;
```

```
    // ========== СУММА ПО СТОЛБЦАМ ==========
    std::cout << "Суммы по столбцам:" << std::endl;
    for (int j = 0; j < COLS; j++) {
        int colSum = 0;
        for (int i = 0; i < ROWS; i++) {
            colSum += matrix[i][j];
        }
        std::cout << "  Столбец " << j << ": " << colSum << std::endl;
    }
    std::cout << std::endl;

    // ========== ОБЩАЯ СУММА ==========
    int totalSum = 0;
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            totalSum += matrix[i][j];
        }
    }
    std::cout << "Общая сумма всех элементов: " << totalSum << std::endl;

    return 0;
}
```

3.3. Пример 3: Работа с диагоналями квадратной матрицы

```
//
==========
=
```

```
// Программа: работа с диагоналями квадратной матрицы
// Вычисление: главная диагональ, побочная диагональ,
//             элементы выше и ниже главной диагонали
// Цель: демонстрация обработки диагональных элементов
//
================================================================
=

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <ctime>

int main() {
    const int N = 5;        // Размер квадратной матрицы (N×N)
    int matrix[N][N];

    // Заполнение случайными числами от 1 до 50
    std::srand(static_cast<unsigned>(std::time(nullptr)));
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            matrix[i][j] = 1 + std::rand() % 50;
        }
    }

    // Вывод матрицы
    std::cout << "Матрица " << N << "×" << N << ":" << std::endl;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
```

```
            std::cout << std::setw(4) << matrix[i][j];
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;

    // ========== 1. ГЛАВНАЯ ДИАГОНАЛЬ ==========
    std::cout << "Главная диагональ: ";
    for (int i = 0; i < N; i++) {
        std::cout << matrix[i][i] << " ";
    }
    std::cout << std::endl;

    // Сумма главной диагонали
    int sumMain = 0;
    for (int i = 0; i < N; i++) {
        sumMain += matrix[i][i];
    }
    std::cout << "Сумма главной диагонали: " << sumMain << std::endl;

    // ========== 2. ПОБОЧНАЯ ДИАГОНАЛЬ ==========
    std::cout << "Побочная диагональ: ";
    for (int i = 0; i < N; i++) {
        std::cout << matrix[i][N - 1 - i] << " ";
    }
    std::cout << std::endl;

    // Сумма побочной диагонали
    int sumSecondary = 0;
```

```
        for (int i = 0; i < N; i++) {
            sumSecondary += matrix[i][N - 1 - i];
        }
        std::cout << "Сумма побочной диагонали: " << sumSecondary << std::endl;

        // ========== 3. ЭЛЕМЕНТЫ ВЫШЕ ГЛАВНОЙ ДИАГОНАЛИ ==========
        std::cout << "\nЭлементы выше главной диагонали:" << std::endl;
        for (int i = 0; i < N; i++) {
            for (int j = i + 1; j < N; j++) {
                std::cout << "matrix[" << i << "][" << j << "] = " << matrix[i][j] << std::endl;
            }
        }

        // ========== 4. ЭЛЕМЕНТЫ НИЖЕ ГЛАВНОЙ ДИАГОНАЛИ ==========
        std::cout << "\nЭлементы ниже главной диагонали:" << std::endl;
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < i; j++) {
                std::cout << "matrix[" << i << "][" << j << "] = " << matrix[i][j] << std::endl;
            }
        }

        return 0;
    }
```

3.4. Пример 4: Транспонирование матрицы (повышенный уровень)

```
//
==========================================================================
=
// Программа: транспонирование матрицы
// Демонстрация: транспонирование в новую матрицу и на месте
// Цель: демонстрация преобразования строк в столбцы
//
==========================================================================
=

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <ctime>

// Функция для вывода матрицы
void printMatrix(const int matrix[][5], int rows, int cols, const std::string&
title) {
    std::cout << title << ":" << std::endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            std::cout << std::setw(4) << matrix[i][j];
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;
}
```

```
int main() {
    const int ROWS = 3;
    const int COLS = 5;
    int matrix[ROWS][COLS];

    // Заполнение случайными числами
    std::srand(static_cast<unsigned>(std::time(nullptr)));
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            matrix[i][j] = 10 + std::rand() % 90;   // числа от 10 до 99
        }
    }

    printMatrix(matrix, ROWS, COLS, "Исходная матрица");

    // ========== 1. ТРАНСПОНИРОВАНИЕ В НОВУЮ МАТРИЦУ
==========
    // Транспонированная матрица будет иметь размер COLS × ROWS
    int transposed[COLS][ROWS];

    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            transposed[j][i] = matrix[i][j];
        }
    }

    printMatrix(transposed, COLS, ROWS, "Транспонированная матрица");
```

```
// ========== 2. ТРАНСПОНИРОВАНИЕ КВАДРАТНОЙ МАТРИЦЫ НА МЕСТЕ ==========
const int N = 4;
int square[N][N];

// Заполнение квадратной матрицы
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        square[i][j] = i * N + j + 1;   // числа от 1 до 16
    }
}

std::cout << "\nКвадратная матрица " << N << "×" << N << ":" << std::endl;
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        std::cout << std::setw(4) << square[i][j];
    }
    std::cout << std::endl;
}
std::cout << std::endl;

// Транспонирование на месте (меняем местами элементы относительно главной диагонали)
// Проходим только по элементам выше главной диагонали, чтобы не менять дважды
for (int i = 0; i < N; i++) {
    for (int j = i + 1; j < N; j++) {
        int temp = square[i][j];
```

```
                square[i][j] = square[j][i];
                square[j][i] = temp;
            }
        }

        std::cout << "Квадратная матрица после транспонирования на месте:" << std::endl;
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                std::cout << std::setw(4) << square[i][j];
            }
            std::cout << std::endl;
        }

        return 0;
    }

    // Пояснение: при транспонировании на месте важно проходить только по i < j
    // Если пройти по всем i и j, то элементы поменяются дважды и матрица восстановится
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Заполнить матрицу (с клавиатуры или случайными числами) и выполнить одно простое вычисление.

	Размер	Способ заполнения	Задание
	3×3	С клавиатуры	Найти сумму всех элементов
	3×3	С клавиатуры	Найти сумму элементов главной диагонали
	4×4	Случайные (0..50)	Найти сумму элементов побочной диагонали
	3×4	С клавиатуры	Найти максимальный элемент
	4×3	С клавиатуры	Найти минимальный элемент
	3×3	Случайные (1..20)	Найти произведение всех элементов (с проверкой на ноль)
	4×4	С клавиатуры	Найти количество нулевых элементов
	3×5	Случайные (-30..30)	Найти количество положительных элементов
	5×3	С клавиатуры	Найти количество отрицательных элементов
	3	Случайны	Найти среднее арифметическое

	Размер	Способ заполнения	Задание
0	3×3	Случайное (10..99)	Вывести все элементы
1	4×4	С клавиатуры	Вывести элементы выше главной диагонали
2	4×4	Случайное (0..100)	Вывести элементы ниже главной диагонали
3	3×5	С клавиатуры	Найти сумму элементов второй строки
4	5×3	Случайное (1..50)	Найти сумму элементов третьего столбца
5	3×3	С клавиатуры	Найти след матрицы (сумму главной диагонали)

Средний уровень (оценка «хорошо»)

Комплексная обработка матрицы (суммы по строкам/столбцам, min/max, работа с диагоналями).

	Размер	Диапазон	Задание
	4×5	-50..50	Найти сумму каждой строки и сумму каждого столбца
	5	1..1	Найти минимальный и максимальный

	Размер	Диапазон	Задание
	×5	00	элемент, их индексы
	4 ×4	0..100	Найти сумму элементов выше главной и ниже главной диагонали
	5 ×5	-100..100	Найти сумму положительных и сумму отрицательных элементов отдельно
	3 ×6	10..99	Найти количество чётных и количество нечётных элементов
	4 ×4	1..50	Найти произведение элементов главной диагонали
	5 ×4	-30..30	Найти строку с максимальной суммой элементов
	4 ×5	0..100	Найти столбец с минимальной суммой элементов
	5 ×5	1..20	Найти сумму элементов, кратных 3
0	4 ×6	-50..50	Найти количество элементов, попадающих в интервал [a, b] (a,b вводятся)
1	3 ×3	1..100	Определить, является ли матрица магическим квадратом (суммы строк, столбцов,

	Размер	Диапазон	Задание
			диагоналей равны)
2	5×5	0..100	Найти среднее арифметическое элементов главной и побочной диагоналей
3	4×4	-100..100	Найти количество локальных минимумов (элемент меньше всех соседей)
4	6×6	1..50	Найти сумму элементов на главной диагонали и выше неё
5	5×5	10..99	Заменить все элементы выше главной диагонали на 0, вывести результат

Повышенный уровень (оценка «отлично»)

Транспонирование, умножение матриц, сложные преобразования.

	Размер	Диапазон	Задание
	4×5	1..100	Транспонировать матрицу (создать новую матрицу размером 5×4)
	5×5	1..50	Транспонировать квадратную матрицу на месте (без дополнительной матрицы)
	3×3	1..10	Умножение двух матриц A(3×3) и B(3×3) — ввести с клавиатуры

	Размер	Диапазон	Задание
	2×3 и 3×4	1..10	Умножение матриц A(2×3) × B(3×4) = C(2×4)
	4×4	0..100	Поворот матрицы на 90° по часовой стрелке
	4×4	0..100	Поворот матрицы на 90° против часовой стрелки
	5×5	1..100	Зеркальное отражение относительно вертикальной оси (обменять столбцы)
	5×5	1..100	Зеркальное отражение относительно горизонтальной оси (обменять строки)
	6×6	-50..50	Удалить строку с заданным номером (сдвигом вверх)
0	6×6	-50..50	Удалить столбец с заданным номером (сдвигом влево)
1	5×5	1..100	Добавить строку в конец матрицы (ввести значения)
2	5×5	1..100	Добавить столбец в конец матрицы (ввести значения)

	Размер	Диапазон	Задание
3	4×4	1..20	Найти определитель матрицы (для 2×2 и 3×3) — для 4×4 разложением
4	5×5	1..100	Сортировка строк матрицы по возрастанию (каждую строку отдельно)
5	5×5	1..100	Сортировка столбцов матрицы по убыванию (каждый столбец отдельно)

5. Контрольные вопросы

Раздел А. Основы двумерных массивов

Что такое двумерный массив (матрица)? Как он хранится в памяти?

Как объявить статическую матрицу 4×5? Приведите пример.

Как проинициализировать матрицу при объявлении? Покажите 3 способа.

Почему второй размер матрицы (количество столбцов) обязательно указывать, а первый можно опустить?

Как получить доступ к элементу матрицы? Что произойдёт при выходе за границы?

Раздел Б. Заполнение и вывод

Как заполнить матрицу с клавиатуры с помощью вложенных циклов?

Как заполнить матрицу случайными числами в заданном диапазоне?

Как вывести матрицу на экран в виде таблицы? Напишите код.

В каком порядке должны быть организованы вложенные циклы для обхода по строкам?

Как изменить порядок циклов для обхода по столбцам?

Раздел В. Базовые алгоритмы

Как найти сумму всех элементов матрицы?

Как найти сумму элементов каждой строки? А каждого столбца?

Как найти минимальный и максимальный элемент матрицы? Как запомнить их индексы?

Что такое главная диагональ квадратной матрицы? Как получить доступ к её элементам?

Что такое побочная диагональ? Как получить доступ к её элементам?

Раздел Г. Продвинутые алгоритмы

Что такое транспонирование матрицы? Как выполнить его для прямоугольной матрицы?

Как выполнить транспонирование квадратной матрицы на месте (без дополнительной памяти)?

Как умножить две матрицы? Какие условия на размеры?

Как выполнить поворот матрицы на 90°? Напишите алгоритм.

Как удалить строку из матрицы? Как удалить столбец?

Раздел Д. Анализ кода и ошибки

Что выведет код?

```
int m[2][3] = {{1,2},{3,4,5}};
std::cout << m[0][2] << " " << m[1][2];
```

Найдите ошибку:

cpp

```
int rows = 3, cols = 4;
int matrix[rows][cols];
```

Что произойдёт?

cpp

```
int m[3][3] = {0};
for (int i = 0; i < 3; i++) {
```

```
    for (int j = 0; j < 3; j++) {
        m[i][j] = i + j;
    }
}
std::cout << m[2][2];
```

Как исправить транспонирование?

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        int temp = m[i][j];
        m[i][j] = m[j][i];
        m[j][i] = temp;
    }
}
```

Чем отличается статическая матрица от std::vector<std::vector<int>>?

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (определение матрицы, объявление, инициализация, доступ к элементам, базовые алгоритмы)

Ход выполнения работы:

Код программы с подробными комментариями (каждый алгоритм пояснён)

Блок-схема алгоритма (для одного из заданий)

Скриншоты выполнения для различных сценариев (разные размеры, разные диапазоны)

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали о матрицах, какие алгоритмы освоили)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №7)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) — правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, правильное объявление матрицы, использование вложенных циклов для заполнения и обработки).

Оценка «хорошо» (4) — правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего уровня (заполнение случайными числами, комплексная обработка, суммы по строкам/столбцам, работа с диагоналями, осмысленные имена переменных).

Оценка «отлично» (5) — правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (транспонирование, умножение матриц, повороты, удаление/вставка строк/столбцов, полная защита от некорректного ввода, подробные комментарии в коде, блок-схема в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
	Компиляция и запуск			
.1	Программа компилируется без ошибок	✔	✔	✔
.2	Программа успешно запускается	✔	✔	✔
.3	Отсутствие предупреждений компилятора	—	✔	✔
	Объявление и инициализация матрицы			
.1	Правильное объявление статической матрицы	✔	✔	✔
.2	Использование констант для размеров	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.3	Корректная инициализация (где требуется)	✔	✔	✔
	Заполнение матрицы			
.1	Заполнение с клавиатуры (где требуется)	✔	✔	✔
.2	Заполнение случайными числами	—	✔	✔
.3	Правильное использование srand() и rand()	—	✔	✔
	Вывод матрицы			
.1	Вывод в виде таблицы (строки и столбцы)	✔	✔	✔
.2	Форматированный вывод (выравнивание)	—	✔	✔
	Базовые алгоритмы обработки			

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.1	Вычисление суммы всех элементов	—	✔	✔
.2	Сумма по строкам/столбцам	—	✔	✔
.3	Поиск минимума/максимума с индексами	—	✔	✔
.4	Работа с главной диагональю	—	✔	✔
.5	Работа с побочной диагональю	—	✔	✔
	Продвинутые алгоритмы			
.1	Транспонирование матрицы	—	—	✔
.2	Умножение матриц	—	—	✔
.3	Поворот матрицы на 90°	—	—	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.4	Удаление/вставка строк/столбцов	—	—	✔
	Обработка ошибок			
.1	Проверка корректности ввода пользователя	—	✔	✔
.2	Проверка на ввод букв (std::cin.fail())	—	—	✔
.3	Контроль границ матрицы (индексы)	✔	✔	✔
	Качество кода			
.1	Правильные отступы и форматирование	✔	✔	✔
.2	Использование осмысленных имён	—	✔	✔
.3	Отсутствие «магических чисел»	—	✔	✔
	Наличие	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.4	комментариев в коде			
0	Отчёт			
0.1	Наличие титульного листа, цели, хода работы	✔	✔	✔
0.2	Скриншоты выполнения (минимум 3 теста)	✔	✔	✔
0.3	Скриншоты для всех трёх уровней	—	✔	✔
0.4	Таблица тестирования	—	✔	✔
0.5	Блок-схема алгоритма	—	✔	✔
0.6	Тестирование на граничных значениях	—	—	✔
0.7	Ответы на контрольные вопросы	—	✔	✔

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) — обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня (№1–15).

Правильно объявить статическую матрицу с константными размерами.

Заполнить матрицу (с клавиатуры или случайными числами согласно заданию).

Выполнить требуемое вычисление (сумма, min, max, количество и т.д.).

Вывести результат на экран и (для задач с выводом элементов) саму матрицу.

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) — дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Заполнить матрицу случайными числами в заданном диапазоне.

Выполнить комплексную обработку (минимум 3 различных операции: суммы по строкам/столбцам, min/max, диагонали).

Вывести матрицу в форматированном виде (выравнивание столбцов).

Использовать осмысленные имена переменных.

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

Блок-схему алгоритма

Повышенный уровень (оценка 5) — требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Реализовать один из сложных алгоритмов:

Транспонирование (прямоугольной или квадратной на месте)

Умножение матриц

Поворот на 90°

Удаление/вставка строк или столбцов

Обработать все возможные ошибки:

Проверка std::cin.fail() при вводе чисел

Контроль границ матрицы при удалении/вставке

Проверка возможности умножения (согласование размеров)

Продемонстрировать понимание индексации:

Корректная работа с индексами при транспонировании

Понимание разницы между i и j в разных алгоритмах

Написать подробные комментарии:

Объяснение логики каждого алгоритма

Пояснение, почему в транспонировании на месте нужен цикл for (int j = i+1; j < N; j++)

В отчёте дополнительно представить:

Тестирование на граничных значениях (минимальные/максимальные размеры)

7.4. Примеры проверки соответствия

Выполненные задания	Оценка	Проверка критериев
Только базовый вариант №1 (сумма всех элементов 3×3)	3	✔ Объявление матрицы, ✔ вложенные циклы, ✔ сумма, ✘ нет сумм по строкам/столбцам, ✘ нет транспонирования
Базовый №1 + средний №1 (суммы по строкам/столбцам)	4	✔ Плюс случайные числа, ✔ суммы по строкам/столбцам, ✔

Выполненные задания	Оценка	Проверка критериев
		форматирование. Но: ✘ нет транспонирования/умножения
Базовый №1 + средний №1 + повышенный №1 (транспонирование)	5	✔ Все критерии: ✔ транспонирование в новую матрицу, ✔ защита от ошибок, ✔ комментарии, ✔ блок-схема

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	Влияние	Как исправить
Выход за границы матрицы	Неопределённое поведение	Проверять индексы: i < ROWS, j < COLS
Перепутан порядок индексов ([j][i] вместо [i][j])	Неправильный доступ	Внимательно следить за порядком: [строка][столбец]
Использование переменной для размера матрицы	Ошибка компиляции	Использовать const int ROWS, const int COLS
Неправильное транспонирование на месте (двойной	Матрица восстанавливается	Использовать for (int j = i+1; j < N; j++)

Ошибка	Влияние	Как исправить
обмен)		
Забыто обнуление матрицы результата при умножении	Мусорные значения	Инициализировать C[i][j] = 0 перед суммированием
Неправильный вывод матрицы (без перевода строки)	Всё в одной строке	Добавить std::cout << std::endl после каждой строки
Одинаковые имена переменных во вложенных циклах	Ошибка компиляции	Использовать разные имена: i, j или row, col

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Как объявляется и заполняется матрица

Как организованы вложенные циклы для обхода

Как выполняется базовый алгоритм обработки (сумма по строкам, работа с диагональю)

(для оценки 4) — как найти сумму по столбцам

(для оценки 5) — логику транспонирования или умножения матриц

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Как изменить код, чтобы найти сумму элементов выше побочной диагонали?"	Понимание диагоналей
"Что произойдёт, если при транспонировании на месте использовать for (int j = 0; j < N; j++)?"	Понимание двойного обмена
"Как проверить, что умножение матриц возможно (размеры согласованы)?"	Понимание условий умножения
"Как удалить строку из статической матрицы?"	Понимание ограничений статических массивов

Лабораторная работа №8

Строки. Базовые операции

1. Цель работы

Научиться работать со строками в C++ с использованием класса std::string. Освоить базовые операции: ввод и вывод строк, определение длины, доступ по индексу, конкатенацию, сравнение, поиск подстроки, извлечение подстроки. Получить навыки посимвольной обработки строк: подсчёт символов (буквы, цифры, гласные, согласные), реверс строки, проверка на палиндром, замена символов.

2. Теоретическое обоснование

2.1. Строки в C++: C-style vs std::string

В C++ существует два основных подхода к работе со строками:

Характеристика	**C-style строки (массивы char)**	**std::string (стандартная библиотека)**
Представление	Массив символов, оканчивающийся '\0'	Объект класса с динамическим управлением памятью
Подключение	Не требуется (<cstring> для функций)	#include <string>
Длина	strlen(s) - O(n)	s.length() или s.size() - O(1)
Присваивание	strcpy(dest, src)	s1 = s2
Конкатенация	strcat(dest, src)	s1 + s2 или s1 += s2
Сравнение	strcmp(s1, s2)	s1 == s2, s1 < s2, s1 > s2
Доступ к символу	s[i]	s[i] или s.at(i)
Безопасность	Низкая (легко выйти за границы)	Высокая (проверка границ через at())

Рекомендация: В современном C++ следует использовать std::string как основной тип для работы со строками.

2.2. Подключение и объявление строк

```
#include <string>      // обязательное подключение
#include <iostream>    // для ввода/вывода

int main() {
    // Пустая строка
    std::string empty;

    // Инициализация строковым литералом
    std::string greeting = "Hello, World!";

    // Инициализация другим объектом string
    std::string copy = greeting;

    // Инициализация с повторением символа
    std::string stars(10, '*');      // "**********"

    // Инициализация частью другой строки (substring)
    std::string hello = greeting.substr(0, 5);   // "Hello"

    return 0;
}
```

2.3. Ввод и вывод строк

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

int main() {
    std::string s;

    // ========== ВВОД ==========

    // Способ 1: cin >> s — читает до первого пробела
    std::cout << "Введите одно слово: ";
    std::cin >> s;                  // ввод: "Hello World" → s = "Hello"

    // Способ 2: std::getline() — читает всю строку до '\n'
    std::cout << "Введите строку: ";
    std::cin.ignore();              // очистка буфера после cin >>
    std::getline(std::cin, s);      // ввод: "Hello World" → s = "Hello World"

    // Способ 3: getline с разделителем
    std::getline(std::cin, s, '|');  // читает до символа '|'

    // ========== ВЫВОД ==========
    std::cout << s << std::endl;

    return 0;
}
```

Важно: После использования std::cin >> перед std::getline() нужно вызвать std::cin.ignore(), чтобы убрать символ новой строки из буфера.

2.4. Основные методы и операции

2.4.1. Длина строки

```
std::string s = "Hello";
int len1 = s.length();    // 5
int len2 = s.size();      // 5 (аналогично)
bool empty = s.empty();   // false (проверка на пустоту)
```

2.4.2. Доступ к символам

```
std::string s = "Hello";

// Оператор [] (без проверки границ)
char ch1 = s[0];          // 'H'
s[0] = 'h';               // "hello"

// Метод at() (с проверкой границ, бросает исключение)
char ch2 = s.at(1);       // 'e'
// s.at(10);              // std::out_of_range
```

2.4.3. Конкатенация (объединение строк)

```
std::string s1 = "Hello";
std::string s2 = "World";

std::string s3 = s1 + " " + s2;      // "Hello World"
s1 += " ";                           // "Hello "
s1 += s2;                            // "Hello World"
```

2.4.4. Сравнение строк

```
std::string s1 = "apple";
std::string s2 = "banana";

if (s1 == s2) { }          // равно
if (s1 != s2) { }          // не равно
if (s1 < s2) { }           // лексикографическое сравнение
if (s1 > s2) { }           // (по кодам символов)
if (s1 <= s2) { }
if (s1 >= s2) { }
```

2.4.5. Поиск подстроки

```
std::string text = "Hello, World! Welcome to C++.";
std::string search = "World";

// Поиск первого вхождения
size_t pos = text.find(search);     // возвращает индекс или std::string::npos
if (pos != std::string::npos) {
    std::cout << "Найдено на позиции: " << pos << std::endl;
}

// Поиск с заданной позиции
pos = text.find("o", 5);            // ищем 'o' начиная с индекса 5

// Поиск последнего вхождения
pos = text.rfind("o");              // последнее 'o'

// Поиск любого символа из набора
```

```
pos = text.find_first_of("aeiou");   // первая гласная
pos = text.find_last_of("aeiou");    // последняя гласная
pos = text.find_first_not_of(" ");   // первый непробельный символ
```

2.4.6. Извлечение подстроки

```
std::string s = "Hello, World!";

// substr(pos, count) — извлекает count символов, начиная с pos
std::string hello = s.substr(0, 5);      // "Hello"
std::string world = s.substr(7, 5);      // "World"
std::string rest = s.substr(7);          // "World!" (до конца)
```

2.4.7. Вставка и удаление

cpp

```
std::string s = "HelloWorld";

// insert(pos, str) — вставка
s.insert(5, ", ");              // "Hello, World"

// erase(pos, count) — удаление
s.erase(5, 2);                  // удаляем ", " → "HelloWorld"

// replace(pos, count, str) — замена
s.replace(5, 5, "C++");         // "HelloC++"
```

2.4.8. Очистка и изменение размера

```
std::string s = "Hello";
s.clear();                      // очистка (s = "")
```

```
s.resize(10, '*');              // увеличиваем до 10, заполняя '*'
                                // "*****" → "*****" (если не хватает)
```

2.5. Посимвольная обработка строк

Строку можно обрабатывать символ за символом, используя цикл for.

```
std::string s = "Hello123";

// Проход по всем символам
for (int i = 0; i < s.length(); i++) {
    char c = s[i];
    // обработка символа
}

// Range-based for (C++11)
for (char c : s) {
    // обработка символа
}
```

2.5.1. Функции для проверки символов (<cctype>)

Функция	Описание	Пример
isalpha(c)	Буква (латиница A-Z, a-z)	isalpha('A') → true
isdigit(c)	Цифра (0-9)	isdigit('5') → true
isalnum(c)	Буква или цифра	isalnum('a') → true

Функция	Описание	Пример
isspace(c)	Пробельный символ (пробел, табуляция, \n)	isspace(' ') → true
islower(c)	Строчная буква	islower('a') → true
isupper(c)	Заглавная буква	isupper('A') → true
tolower(c)	Преобразование в строчную	tolower('A') → 'a'
toupper(c)	Преобразование в заглавную	toupper('a') → 'A'

```
#include <cctype>

std: string s = "Hello, World! 123";
int letterCount = 0;
int digitCount = 0;
int spaceCount = 0;
int otherCount = 0;

for (char c : s) {
    if (std::isalpha(c)) {
        letterCount++;
    } else if (std::isdigit(c)) {
        digitCount++;
```

```
    } else if (std::isspace(c)) {
        spaceCount++;
    } else {
        otherCount++;
    }
}
// Результат: букв: 10, цифр: 3, пробелов: 2, других: 2
```

2.6. Реверс строки

```
#include <algorithm>   // для std::reverse

std::string s = "Hello";

// Способ 1: с использованием std::reverse
std::reverse(s.begin(), s.end());   // "olleH"

// Способ 2: ручной реверс
for (int i = 0; i < s.length() / 2; i++) {
    char temp = s[i];
    s[i] = s[s.length() - 1 - i];
    s[s.length() - 1 - i] = temp;
}

// Способ 3: создание новой строки в обратном порядке
std::string reversed;
for (int i = s.length() - 1; i >= 0; i--) {
    reversed += s[i];
}
```

2.7. Проверка на палиндром

Палиндром - строка, которая читается одинаково слева направо и справа налево (без учёта регистра и пробелов).

```
#include <cctype>

bool isPalindrome(const std::string& s) {
    int left = 0;
    int right = s.length() - 1;

    while (left < right) {
        // Пропускаем неподходящие символы (по желанию)
        // if (!std::isalnum(s[left])) { left++; continue; }
        // if (!std::isalnum(s[right])) { right--; continue; }

        // Приводим к нижнему регистру для сравнения
        if (std::tolower(s[left]) != std::tolower(s[right])) {
            return false;
        }
        left++;
        right--;
    }
    return true;
}
```

Примеры палиндромов:

"шалаш"

"А роза упала на лапу Азора" (с пробелами)

"Madam, I'm Adam" (с учётом регистра и пунктуации)

2.8. Подсчёт слов в строке

```
int countWords(const std::string& s) {
    int wordCount = 0;
    bool inWord = false;

    for (char c : s) {
        if (std::isspace(c)) {
            inWord = false;
        } else {
            if (!inWord) {
                wordCount++;
                inWord = true;
            }
        }
    }
    return wordCount;
}
```

2.9. Замена символов в строке

```
// Замена всех символов 'a' на 'b'
for (char& c : s) {
    if (c == 'a') {
        c = 'b';
    }
}
```

```
// Замена всех гласных на '*'
for (char& c : s) {
    char lower = std::tolower(c);
    if (lower == 'a' || lower == 'e' || lower == 'i' ||
        lower == 'o' || lower == 'u') {
        c = '*';
    }
}
```

2.10. Типичные ошибки при работе со строками

Ошибка	Пример	Последствие
Забыт #include <string>	Использование std::string без подключения	Ошибка компиляции
Смешение cin >> и getline()	cin >> n; getline(cin, s);	getline читает пустую строку
Выход за границы строки	s[s.length()]	Неопределённое поведение
Сравнение с npos без проверки	if (s.find("x"))	npos не 0, поэтому всегда true
Использование std::string с C-функциями	strlen(s.c_str())	Можно, но не нужно

```
// ОШИБКА: неправильное использование find
std::string s = "Hello";
if (s.find("x")) {          // неправильно!
```

```
    // s.find("x") возвращает npos (обычно -1), что != 0 → true
}

// ПРАВИЛЬНО
if (s.find("x") != std::string::npos) {
    // найдено
}
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Ввод и вывод строк, определение длины

```
//
==========================================================
=
// Программа: ввод строки и вывод её характеристик
// Цель: демонстрация базовых операций со строками
//
==========================================================
=

#include <iostream>
#include <string>

int main() {
    std::string line;

    // Ввод строки с пробелами
    std::cout << "Введите строку: ";
    std::getline(std::cin, line);
```

```
    // Вывод информации о строке
    std::cout << "\n=== Информация о строке ===" << std::endl;
    std::cout << "Строка: \"" << line << "\"" << std::endl;
    std::cout << "Длина: " << line.length() << " символов" << std::endl;

    // Проверка на пустоту
    if (line.empty()) {
        std::cout << "Строка пустая!" << std::endl;
    }

    // Вывод первого и последнего символа
    if (!line.empty()) {
        std::cout << "Первый символ: '" << line[0] << "'" << std::endl;
        std::cout << "Последний символ: '" << line[line.length() - 1] << "'" <<
std::endl;
    }

    return 0;
}

// Пример работы:
// Ввод: "Hello, World!"
// Вывод: Длина: 13, первый символ: 'H', последний: '!'
```

3.2. Пример 2: Подсчёт символов (буквы, цифры, гласные)

```
//
==========================================================================
=
// Программа: подсчёт различных типов символов в строке
// Цель: демонстрация посимвольной обработки и <cctype>
//
==========================================================================
=

#include <iostream>
#include <string>
#include <cctype>   // для isalpha, isdigit, tolower и др.

int main() {
    std::string text;

    std::cout << "Введите строку: ";
    std::getline(std::cin, text);

    // Инициализация счётчиков
    int letterCount = 0;      // буквы
    int digitCount = 0;       // цифры
    int vowelCount = 0;       // гласные (a, e, i, o, u)
    int consonantCount = 0;   // согласные
    int spaceCount = 0;       // пробелы
    int otherCount = 0;       // остальные символы

    // Посимвольная обработка
    for (char c : text) {
```

```
    if (std::isalpha(c)) {
        letterCount++;

        // Проверка на гласные (приводим к нижнему регистру)
        char lower = std::tolower(c);
        if (lower == 'a' || lower == 'e' || lower == 'i' ||
            lower == 'o' || lower == 'u') {
            vowelCount++;
        } else {
            consonantCount++;
        }
    } else if (std::isdigit(c)) {
        digitCount++;
    } else if (std::isspace(c)) {
        spaceCount++;
    } else {
        otherCount++;
    }
}

// Вывод результатов
std::cout << "\n=== Результаты подсчёта ===" << std::endl;
std::cout << "Всего букв: " << letterCount << std::endl;
std::cout << "  Гласных: " << vowelCount << std::endl;
std::cout << "  Согласных: " << consonantCount << std::endl;
std::cout << "Цифр: " << digitCount << std::endl;
std::cout << "Пробелов: " << spaceCount << std::endl;
std::cout << "Других символов: " << otherCount << std::endl;
```

```
    return 0;
}

// Пример работы:
// Ввод: "Hello, 123 World!"
// Результат: букв: 10, гласных: 3 (e, o, o), согласных: 7,
//            цифр: 3, пробелов: 2, других: 2 (запятая и восклицательный знак)
```

3.3. Пример 3: Реверс строки и проверка на палиндром

```
//
================================================================
=
// Программа: реверс строки и проверка на палиндром
// Цель: демонстрация алгоритмов обработки строк
//
================================================================
=

#include <iostream>
#include <string>
#include <algorithm>  // для std::reverse
#include <cctype>     // для tolower

// Функция для проверки, является ли строка палиндромом
// (учитывает только буквы и цифры, игнорирует регистр)
bool isPalindrome(const std::string& s) {
    int left = 0;
```

```
    int right = s.length() - 1;

    while (left < right) {
        // Пропускаем символы, которые не являются буквами и цифрами
        while (left < right && !std::isalnum(s[left])) {
            left++;
        }
        while (left < right && !std::isalnum(s[right])) {
            right--;
        }

        // Сравниваем символы (без учёта регистра)
        if (std::tolower(s[left]) != std::tolower(s[right])) {
            return false;
        }
        left++;
        right--;
    }
    return true;
}

int main() {
    std::string original;

    std::cout << "Введите строку: ";
    std::getline(std::cin, original);

    // ========== РЕВЕРС СТРОКИ ==========
    std::string reversed = original;
```

```
    std::reverse(reversed.begin(), reversed.end());

    std::cout << "\nИсходная строка: \"" << original << "\"" << std::endl;
    std::cout << "Реверс: \"" << reversed << "\"" << std::endl;

    // ========== ПРОВЕРКА НА ПАЛИНДРОМ ==========
    if (isPalindrome(original)) {
        std::cout << "\nСтрока является палиндромом!" << std::endl;
    } else {
        std::cout << "\nСтрока не является палиндромом." << std::endl;
    }

    // ========== ДЕМОНСТРАЦИЯ РУЧНОГО РЕВЕРСА ==========
    std::string manualReverse = original;
    int len = manualReverse.length();
    for (int i = 0; i < len / 2; i++) {
        char temp = manualReverse[i];
        manualReverse[i] = manualReverse[len - 1 - i];
        manualReverse[len - 1 - i] = temp;
    }
    std::cout << "Ручной реверс: \"" << manualReverse << "\"" << std::endl;

    return 0;
}

// Пример работы:
// Ввод: "А роза упала на лапу Азора"
// Результат: является палиндромом!
```

```
// Ввод: "Hello"
// Результат: не является палиндромом, реверс: "olleH"
```

3.4. Пример 4: Подсчёт слов и замена символов

```
//
==========================================================================
=
// Программа: подсчёт слов и замена символов в строке
// Цель: демонстрация поиска слов и преобразования строк
//
==========================================================================
=

#include <iostream>
#include <string>
#include <cctype>
#include <sstream>   // для std::stringstream

int main() {
    std::string text;

    std::cout << "Введите строку: ";
    std::getline(std::cin, text);

    // ========== ПОДСЧЁТ СЛОВ (через флаг) ==========
    int wordCount = 0;
    bool inWord = false;
```

```
        for (char c : text) {
            if (std::isspace(c)) {
                inWord = false;
            } else {
                if (!inWord) {
                    wordCount++;
                    inWord = true;
                }
            }
        }
        std::cout << "\nКоличество слов: " << wordCount << std::endl;

        // ========== ПОДСЧЁТ СЛОВ (через stringstream) ==========
        std::stringstream ss(text);
        std::string word;
        int wordCount2 = 0;
        while (ss >> word) {
            wordCount2++;
        }
        std::cout << "Количество слов (stringstream): " << wordCount2 <<
std::endl;

        // ========== ЗАМЕНА ВСЕХ ГЛАСНЫХ НА '*' ==========
        std::string replacedVowels = text;
        for (char& c : replacedVowels) {
            char lower = std::tolower(c);
            if (lower == 'a' || lower == 'e' || lower == 'i' ||
                lower == 'o' || lower == 'u') {
                c = '*';
```

```
            }
        }
        std::cout << "\nЗамена гласных на '*': " << replacedVowels << std::endl;

        // ========== ПРЕОБРАЗОВАНИЕ В ВЕРХНИЙ РЕГИСТР ==========
        std::string upper = text;
        for (char& c : upper) {
            c = std::toupper(c);
        }
        std::cout << "Верхний регистр: " << upper << std::endl;

        // ========== ПРЕОБРАЗОВАНИЕ В НИЖНИЙ РЕГИСТР ==========
        std::string lower = text;
        for (char& c : lower) {
            c = std::tolower(c);
        }
        std::cout << "Нижний регистр: " << lower << std::endl;

        // ========== УДАЛЕНИЕ ПРОБЕЛОВ В НАЧАЛЕ И КОНЦЕ ==========
        size_t start = text.find_first_not_of(" \t\n\r");
        size_t end = text.find_last_not_of(" \t\n\r");

        if (start != std::string::npos && end != std::string::npos) {
            std::string trimmed = text.substr(start, end - start + 1);
            std::cout << "Без пробелов по краям: \"" << trimmed << "\"" << std::endl;
```

```
    }

    return 0;
}

// Пример работы:
// Ввод: "   Hello,   World!   "
// Количество слов: 2
// Замена гласных: "H*ll*,   W*rld!   "
// Верхний регистр: "   HELLO,   WORLD!   "
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Выполнить одну базовую операцию со строкой.

№	Задание
1	Ввести строку и вывести её длину
2	Ввести строку и вывести её первый и последний символ
3	Ввести строку и вывести все символы на чётных позициях (0, 2, 4...)
4	Ввести строку и вывести все символы на нечётных позициях (1, 3, 5...)
5	Ввести строку и вывести её в обратном порядке (используя цикл)
6	Ввести строку и подсчитать количество символов 'a'

№	Задание
7	Ввести строку и подсчитать количество пробелов
8	Ввести строку и подсчитать количество букв (латиница)
9	Ввести строку и подсчитать количество цифр
10	Ввести строку и заменить все пробелы на символ '_'
11	Ввести строку и заменить все символы 'a' на 'b'
12	Ввести две строки и объединить их (конкатенация)
13	Ввести строку и проверить, содержит ли она подстроку "abc"
14	Ввести строку и вывести её без первого и последнего символа
15	Ввести строку и определить, является ли она пустой

Средний уровень (оценка «хорошо»)

Комплексная обработка строки с использованием <cctype>.

№	Задание
1	Подсчитать количество гласных и согласных букв (для латиницы)
2	Подсчитать количество слов в строке (разделители — пробелы)

№	Задание
3	Преобразовать все символы строки в верхний регистр
4	Преобразовать все символы строки в нижний регистр
5	Заменить все цифры в строке на символ '#'
6	Удалить из строки все пробелы
7	Вывести строку, где каждый символ повторяется дважды (Hello → HHeellllloo)
8	Проверить, является ли строка палиндромом (без учёта регистра и пробелов)
9	Подсчитать количество слов, начинающихся с заглавной буквы
10	Подсчитать количество букв 'e' и 'E' (суммарно)
11	Найти позицию первого вхождения символа 'x' (вывести -1, если нет)
12	Извлечь подстроку с позиции 3 длиной 5 символов
13	Заменить все гласные буквы на символ '*'

№	Задание
14	Вывести строку, где порядок слов обратный (Hello World → World Hello)
15	Определить, есть ли в строке одинаковые соседние символы (например, "aa")

Повышенный уровень (оценка «отлично»)

Сложные алгоритмы обработки строк, поиск, замена, форматирование.

№	Задание
1	Проверка на палиндром с игнорированием всех не буквенно-цифровых символов и учётом регистра (пример: "A man, a plan, a canal: Panama" → true)
2	Реверс слов в строке (порядок слов меняется, порядок букв в слове сохраняется)
3	Реверс букв в каждом слове (порядок слов сохраняется, буквы в каждом слове переворачиваются)
4	Поиск и замена всех вхождений подстроки (заменить все "abc" на "def")
5	Подсчёт частоты встречаемости каждого символа (вывести таблицу: символ → количество)
6	Сжатие строки (Run-length encoding): "aaabbbbcc" → "a3b4c2"

№	Задание
7	Распаковка строки (обратная операция): "a3b4c2" → "aaabbbbcc"
8	Проверка, является ли строка анаграммой другой строки (например, "listen" и "silent")
9	Выделение чисел из строки (из строки "abc123def45" получить массив чисел [123, 45])
10	Форматирование строки под заданную ширину (дополнение пробелами слева или справа)
11	Генерация всех подстрок (вывести все возможные подстроки заданной строки)
12	Нахождение самой длинной общей подстроки для двух строк
13	Шифр Цезаря (сдвиг каждого символа на K позиций в алфавите, только латиница)
14	Проверка корректности email-адреса (наличие '@', точка после '@', нет пробелов)
15	Простой текстовый редактор (поддержка команд: ввод строки, вывод, реверс, замена, вставка, удаление, выход)

5. Контрольные вопросы

Раздел А. Основы std::string

Что такое std::string? Чем он отличается от C-style строк (массива char)?

Какой заголовочный файл нужно подключить для работы с std::string?

Как объявить и инициализировать строку? Приведите 4 способа.

Как ввести строку с пробелами? Почему std::cin >> не подходит?

В чём разница между s.length() и s.size()?

Раздел Б. Доступ к символам и операции

Как получить доступ к символу по индексу? Чем отличается оператор [] от метода at()?

Как объединить две строки? Покажите 2 способа.

Как сравнить две строки? Какие операторы сравнения доступны?

Как извлечь подстроку из строки? Какой синтаксис метода substr()?

Как найти подстроку в строке? Что возвращает метод find() при неудаче?

Раздел В. Посимвольная обработка

Как пройти по всем символам строки? Покажите 2 способа.

Какие функции из <cctype> вы знаете? Для чего используются isalpha(), isdigit(), isspace()?

Как преобразовать символ в верхний/нижний регистр? Какие функции нужны?

Как посчитать количество слов в строке? Напишите алгоритм.

Как выполнить реверс строки? Покажите 2 способа (ручной и через std::reverse).

Раздел Г. Продвинутые алгоритмы

Что такое палиндром? Как проверить, является ли строка палиндромом?

Как заменить все вхождения символа в строке? А подстроки?

Как удалить пробелы в начале и конце строки? (trim)

Как разбить строку на слова? Какие способы существуют?

Как проверить, что строка является анаграммой другой строки?

Раздел Д. Анализ кода и ошибки

Что выведет код?

```
std::string s = "Hello";
std::cout << s[5];
```

Найдите ошибку:

cpp

```
std::string s;
std::cout << "Введите строку: ";
std::cin >> s;
std::getline(std::cin, s);
```

Что произойдёт?

```
std::string s = "Hello";
if (s.find("H") == 0) {
    std::cout << "Найдено";
}
```

Как исправить код для подсчёта букв?

```
int count = 0;
for (int i = 1; i <= s.length(); i++) {
    if (isalpha(s[i])) count++;
}
```

Чем отличается s.find(c) от s.find_first_of(c)?

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (определение std::string, основные методы, посимвольная обработка, функции из <cctype>)

Ход выполнения работы:

Код программы с подробными комментариями (каждый алгоритм пояснён)

Блок-схема алгоритма (для одного из заданий)

Скриншоты выполнения для различных сценариев (разные строки, граничные случаи)

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали о строках, какие алгоритмы освоили)

Формат: .docx или .pdf.

7. Критерии оценивания

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) — правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, использование std::string, базовые операции).

Оценка «хорошо» (4) — правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего

уровня (использование <cctype>, посимвольная обработка, подсчёт слов, преобразование регистра, осмысленные имена переменных).

Оценка «отлично» (5) — правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (палиндром с игнорированием символов, реверс слов, RLE-сжатие, шифр Цезаря, полная защита от некорректного ввода, подробные комментарии в коде, блок-схема в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
1	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
2	Компиляция и запуск			
2	Программа компилируется	✔	✔	✔

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
1	без ошибок			
2.2	Программа успешно запускается	✔	✔	✔
2.3	Отсутствие предупреждений компилятора	—	✔	✔
3	Работа с std::string			
3.1	Правильное подключение #include <string>	✔	✔	✔
3.2	Корректный ввод (getline для строк с пробелами)	✔	✔	✔
3.3	Использование .length() / .size()	✔	✔	✔
3.4	Использование оператора [] или at()	✔	✔	✔
3.5	Конкатенация (+ или +=)	✔	✔	✔
4	Посимвольная обработка			

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
4.1	Цикл по символам строки	✔	✔	✔
4.2	Использование <cctype> (isalpha, isdigit, isspace)	—	✔	✔
4.3	Использование tolower / toupper	—	✔	✔
5	Базовые алгоритмы			
5.1	Подсчёт определённых символов	✔	✔	✔
5.2	Подсчёт слов в строке	—	✔	✔
5.3	Реверс строки	—	✔	✔
5.4	Проверка на палиндром	—	✔	✔
6	Продвинутые алгоритмы			
6	Палиндром с	—	—	✔

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.1	игнорированием символов			
6.2	Реверс слов / букв в словах	—	—	✔
6.3	Поиск и замена подстрок	—	—	✔
6.4	RLE-сжатие / распаковка	—	—	✔
6.5	Шифр Цезаря	—	—	✔
7	Обработка ошибок			
7.1	Проверка на пустую строку	—	✔	✔
7.2	Проверка на ввод букв/цифр	—	—	✔
7.3	Контроль выхода за границы	—	✔	✔
8	Качество кода			

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
8.1	Правильные отступы и форматирование	✔	✔	✔
8.2	Использование осмысленных имён	—	✔	✔
8.3	Наличие комментариев в коде	—	✔	✔
8.4	Использование range-based for	—	—	✔
9	Отчёт			
9.1	Титульный лист, цель, ход работы	✔	✔	✔
9.2	Скриншоты выполнения (минимум 3 теста)	✔	✔	✔
9.3	Скриншоты для всех трёх уровней	—	✔	✔
9.4	Таблица тестирования	—	✔	✔

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
9.5	Блок-схема алгоритма	—	✔	✔
9.6	Тестирование на граничных значениях	—	—	✔
9.7	Ответы на контрольные вопросы	—	✔	✔

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) — обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня (№1–15).

Правильно использовать std::string и базовые операции.

Корректно ввести строку (с учётом пробелов через getline).

Выполнить требуемую операцию (вывод длины, символов, подсчёт, замена).

Вывести результат на экран.

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) — дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Использовать функции из <cctype> для классификации символов.

Выполнить комплексную обработку (подсчёт гласных/согласных, преобразование регистра, подсчёт слов).

Корректно обработать пустую строку.

Использовать осмысленные имена переменных.

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

Блок-схему алгоритма

Повышенный уровень (оценка 5) — требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Реализовать один из сложных алгоритмов:

Палиндром с игнорированием всех не буквенно-цифровых символов

Реверс слов (на уровне слов, а не символов)

RLE-сжатие или распаковка

Шифр Цезаря

Поиск и замена подстрок

Обработать все возможные ошибки:

Проверка на пустую строку

Обработка случая, когда искомая подстрока не найдена

Проверка границ при извлечении подстрок

Продемонстрировать понимание алгоритмов:

Двухпроходные алгоритмы (сначала анализ, потом преобразование)

Использование std::string::npos

Написать подробные комментарии:

Объяснение логики каждого алгоритма

Пояснение, почему выбран тот или иной подход

В отчёте дополнительно представить:

Тестирование на граничных значениях (пустая строка, строка из одного символа)

Примеры сложных случаев (палиндромы с пунктуацией)

7.4. Примеры проверки соответствия

Выполненные задания	**Оценка**	**Проверка критериев**
Только базовый вариант №1 (длина строки)	3	✔ std::string, ✔ getline, ✔ .length(), ✘ нет <cctype>, ✘ нет посимвольной обработки
Базовый №1 + средний №1 (гласные/согласные)	4	✔ Плюс <cctype>, ✔ посимвольная обработка, ✔ форматирование. Но: ✘ нет сложного алгоритма (палиндром/RLE)
Базовый №1 + средний №1 + повышенный №1 (палиндром)	5	✔ Все критерии: ✔ палиндром с игнорированием символов, ✔ обработка пустой строки, ✔ комментарии, ✔ блок-схема

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	Влияние	Как исправить
Забыт #include <string>	Ошибка компиляции	Добавить подключение

Ошибка	Влияние	Как исправить
Использование cin >> для строк с пробелами	Читает только до пробела	Использовать std::getline()
Забыт cin.ignore() после cin >>	getline читает пустую строку	Вызвать cin.ignore()
Выход за границы строки	Неопределённое поведение	Проверять i < s.length()
Неправильное сравнение с npos	Всегда true	Проверять != std::string::npos
Изменение строки во время итерации (range-based for)	Неопределённое поведение	Использовать for (int i...) или копию
Не учтён регистр при проверке палиндрома	Неверный результат	Приводить к tolower()

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Как объявить и ввести строку с пробелами

Как получить доступ к отдельному символу

Как выполнить посимвольную обработку

(для оценки 4) — как использовать функции из <cctype>

(для оценки 5) — логику сложного алгоритма (палиндром, RLE, реверс слов)

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Как изменить код, чтобы подсчитать количество предложений в тексте?"	Понимание разбиения по разделителям
"Что произойдёт при вызове s.substr(10, 5) в строке длиной 8?"	Понимание поведения substr
"Как заменить все вхождения подстроки, а не только одно?"	Понимание циклического поиска
"Как проверить, что строка является анаграммой другой строки?"	Понимание сортировки или подсчёта частот

Лабораторная работа №9

Функции. Декомпозиция задач

1. Цель работы

Научиться разбивать программу на функции для декомпозиции сложных задач. Освоить определение и вызов функций, передачу параметров,

возврат значений, использование прототипов функций. Получить навыки создания функций для вычисления суммы, поиска минимума/максимума, проверки простоты числа, организации модульной структуры программы.

2. Теоретическое обоснование

2.1. Понятие функции

Функция - это именованный блок кода, который выполняет определённую задачу и может быть вызван из других частей программы. Функции позволяют:

Избежать дублирования кода (DRY - Don't Repeat Yourself)

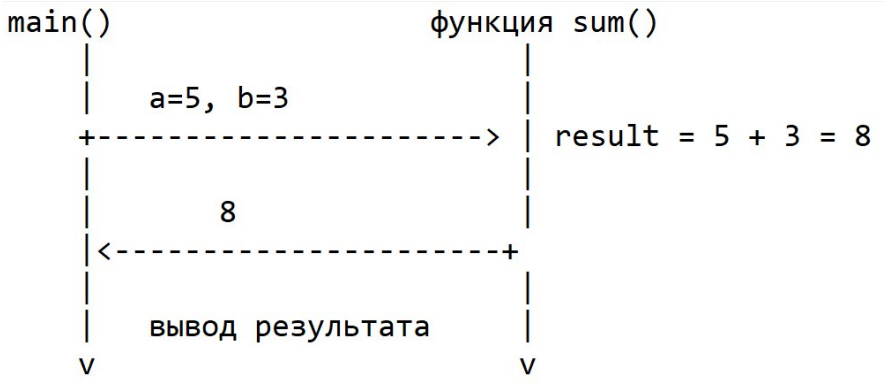
Разбить сложную задачу на подзадачи (декомпозиция)

Улучшить читаемость программы

Облегчить тестирование и отладку

Обеспечить повторное использование кода

Визуализация вызова функции:



2.2. Синтаксис определения функции

```
тип_возвращаемого_значения имя_функции(параметры) {
    // тело функции
    return значение;  // если тип возвращаемого значения не void
}
```

Компоненты функции:

Компонент	Описание	Пример
Тип возвращаемого значения	Тип данных, который возвращает функция (int, double, void и др.)	int
Имя функции	Идентификатор, по которому вызывается функция	sum
Параметры	Переменные, принимающие значения из вызывающего кода	int a, int b
Тело функции	Блок кода в фигурных скобках {}	return a + b;
return	Оператор возврата значения из функции	return result;

Пример простой функции:

```
// Функция вычисления суммы двух целых чисел
int sum(int a, int b) {
    int result = a + b;   // вычисление
    return result;        // возврат результата
}
```

Функция, не возвращающая значение (void):

```
// Функция вывода приветствия
void printGreeting() {
    std::cout << "Hello, World!" << std::endl;
    // return; - можно опустить (неявный return в конце)
```

```
}
```

2.3. Вызов функции

```
#include <iostream>

// Определение функции (до вызова)
int sum(int a, int b) {
    return a + b;
}

int main() {
    int x = 5, y = 3;

    // Вызов функции: передаём аргументы 5 и 3, получаем результат 8
    int result = sum(x, y);

    std::cout << "Сумма: " << result << std::endl;  // Сумма: 8

    // Можно использовать результат непосредственно в выражении
    std::cout << "Сумма 10 и 20: " << sum(10, 20) << std::endl;  // 30

    return 0;
}
```

Важные термины:

Аргументы - значения, передаваемые в функцию при вызове (x, y в примере)

Параметры - переменные, определённые в заголовке функции (a, b в примере)

2.4. Прототипы функций (объявление перед использованием)

Функция должна быть объявлена (прототип) до её первого вызова, либо определена полностью. Прототип - это заголовок функции без тела, заканчивающийся точкой с запятой.

```
#include <iostream>

// Прототипы функций (объявления)
int sum(int a, int b);
void printResult(int value);
double calculateAverage(int arr[], int size);

int main() {
    int result = sum(5, 3);        // можно вызывать, даже если определение ниже
    printResult(result);
    return 0;
}

// Определения функций (реализации)
int sum(int a, int b) {
    return a + b;
}

void printResult(int value) {
    std::cout << "Результат: " << value << std::endl;
}
```

Зачем нужны прототипы:

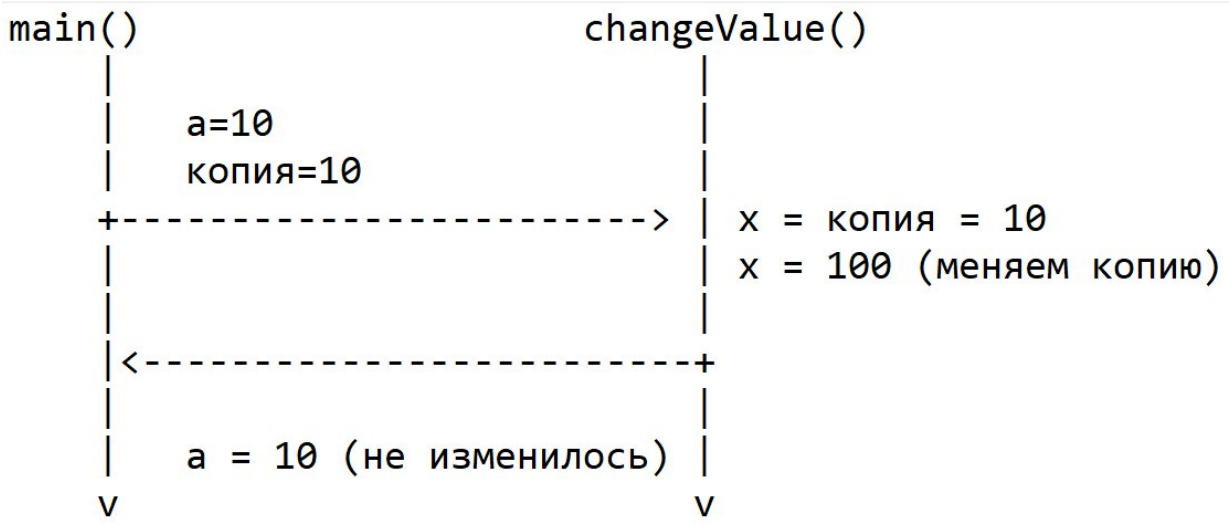
1. Позволяют вызывать функцию до её определения
2. Позволяют разделить код на несколько файлов
3. Обеспечивают проверку типов на этапе компиляции

2.5. Передача параметров: по значению (по умолчанию)

По умолчанию в C++ параметры передаются по значению - функция получает копию аргумента. Изменение параметра внутри функции не влияет на исходную переменную.

```
void changeValue(int x) {
    x = 100;                    // меняем копию
    std::cout << "Внутри функции: " << x << std::endl;
}

int main() {
    int a = 10;
    changeValue(a);
    std::cout << "В main: " << a << std::endl;  // всё ещё 10
    return 0;
}
// Вывод:
// Внутри функции: 100
// В main: 10
```

Визуализация передачи по значению:



2.6. Передача массивов в функции

При передаче массива в функцию передаётся указатель на первый элемент (адрес). Поэтому функция может изменять элементы исходного массива.

```
// Передача массива: нужно также передать его размер
void fillArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        arr[i] = i + 1;   // изменяем исходный массив
    }
}

// Альтернативный синтаксис (эквивалентен)
void fillArray2(int* arr, int size) {
    for (int i = 0; i < size; i++) {
        arr[i] = i + 1;
    }
}
```

```
// Передача массива как константного (только чтение)
void printArray(const int arr[], int size) {
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
        // arr[i] = 0;   // ОШИБКА! нельзя изменять const массив
    }
}

int main() {
    int numbers[5];
    fillArray(numbers, 5);   // numbers теперь {1, 2, 3, 4, 5}
    printArray(numbers, 5);
    return 0;
}
```

Важно: Размер массива при передаче не сохраняется - его нужно передавать отдельным параметром.

```
// НЕПРАВИЛЬНО: sizeof(arr) внутри функции вернёт размер указателя, не массива
void badFunction(int arr[]) {
    int size = sizeof(arr) / sizeof(arr[0]);  // ОШИБКА! size = 1 или 2
}
```

2.7. Передача по ссылке (обзор для следующей лабораторной)

Хотя подробно ссылки будут в ЛР №10, для декомпозиции полезно знать, что ссылки позволяют изменять переменные внутри функции:

```
// Передача по ссылке (функция может изменить исходную переменную)
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
    int x = 5, y = 10;
    swap(x, y);   // x = 10, y = 5
    return 0;
}
```

2.8. Возврат нескольких значений из функции

Функция может вернуть только одно значение. Для возврата нескольких значений используются:

Ссылки (параметры-результаты)

Структуры (будет в ООП)

std::pair / std::tuple

```
// Возврат двух значений через параметры-ссылки
void findMinMax(int arr[], int size, int& minVal, int& maxVal) {
    minVal = arr[0];
    maxVal = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] < minVal) minVal = arr[i];
        if (arr[i] > maxVal) maxVal = arr[i];
    }
```

```
}

int main() {
    int data[] = {3, 7, 1, 9, 4};
    int min, max;
    findMinMax(data, 5, min, max);
    std::cout << "Min: " << min << ", Max: " << max << std::endl;
    return 0;
}
```

2.9. Локальные и глобальные переменные

Локальные переменные - объявлены внутри функции, доступны только в ней, создаются при вызове, уничтожаются при выходе.

Глобальные переменные - объявлены вне всех функций, доступны всем функциям (не рекомендуется использовать без крайней необходимости).

```
#include <iostream>

int globalVar = 100;  // глобальная переменная (плохая практика!)

void myFunction() {
    int localVar = 10;     // локальная переменная
    globalVar = 200;       // изменение глобальной переменной
    // std::cout << x;     // ОШИБКА! x не видна здесь
}

int main() {
    int x = 5;             // локальная переменная main
    myFunction();
```

```
    std::cout << globalVar;  // 200
    // std::cout << localVar; // ОШИБКА! localVar не видна
    return 0;
}
```

Правило: Избегайте глобальных переменных. Передавайте данные через параметры и возвращайте результат.

2.10. Область видимости (scope)

Область видимости - часть программы, где переменная доступна. Основные области видимости в C++:

Глобальная (вне всех функций)

Локальная (внутри функции)

Блочная (внутри { }, например в цикле)

```
int global = 1;          // глобальная

void func() {
    int local = 2;       // локальная
    for (int i = 0; i < 3; i++) {
        int block = 3;   // блочная (только внутри цикла)
        // visible: global, local, i, block
    }
    // i и block не видны здесь
}
```

2.11. Рекурсия (функция вызывает саму себя)

Рекурсия - вызов функции из самой себя. Используется для решения задач, которые можно разбить на подобные подзадачи (факториал, числа Фибоначчи, обход дерева).

```
// Факториал через рекурсию: n! = n * (n-1)!
int factorial(int n) {
    if (n <= 1) {
        return 1;                    // базовый случай
    } else {
        return n * factorial(n - 1); // рекурсивный вызов
    }
}
// factorial(5) = 5 * 4 * 3 * 2 * 1 = 120
```

Визуализация рекурсии factorial(5):

text

```
factorial(5) = 5 * factorial(4)
             = 5 * (4 * factorial(3))
             = 5 * (4 * (3 * factorial(2)))
             = 5 * (4 * (3 * (2 * factorial(1))))
             = 5 * (4 * (3 * (2 * 1)))
             = 120
```

Важно: Каждая рекурсия должна иметь базовый случай (условие выхода), иначе произойдёт бесконечная рекурсия и переполнение стека.

```
// ОШИБКА: нет базового случая
void infiniteRecursion(int n) {
    std::cout << n << std::endl;
    infiniteRecursion(n + 1);   // бесконечная рекурсия!
}
```

2.12. Перегрузка функций (function overloading)

Перегрузка функций - создание нескольких функций с одинаковым именем, но разными параметрами (по количеству, типу или порядку). Компилятор определяет, какую функцию вызывать, по аргументам.

```
// Перегруженные функции
int sum(int a, int b) {
    return a + b;
}

double sum(double a, double b) {
    return a + b;
}

int sum(int a, int b, int c) {
    return a + b + c;
}

int main() {
    std::cout << sum(5, 3);       // вызывает int sum(int, int) → 8
    std::cout << sum(5.5, 3.2);    // вызывает double sum(double, double) → 8.7
    std::cout << sum(1, 2, 3);    // вызывает int sum(int, int, int) → 6
    return 0;
}
```

2.13. Стандартные математические функции (<cmath>)

```
#include <cmath>
```

```
double x = 2.0;
double y = std::sqrt(x);     // квадратный корень: 1.414
double z = std::pow(x, 3);   // возведение в степень: 8.0
double a = std::abs(-5);     // модуль: 5
double b = std::ceil(3.14);  // округление вверх: 4.0
double c = std::floor(3.14); // округление вниз: 3.0
double d = std::round(3.6);  // математическое округление: 4.0
double e = std::sin(3.14);   // синус (в радианах)
double f = std::cos(3.14);   // косинус
double g = std::exp(1.0);    // e^1 = 2.718
double h = std::log(10.0);   // натуральный логарифм
```

2.14. Типичные ошибки при работе с функциями

Ошибка	**Пример**	**Последствие**
Вызов функции до объявления/определения	Вызов sum(5,3) до прототипа	Ошибка компиляции
Несоответствие типов аргументов	sum(5, 3.14) если ожидается int, int	Неявное преобразование или ошибка
Забыт return в не-void функции	Функция без return	Неопределённое поведение
Несоответствие типа возврата	return 3.14; в int функции	Усечение

Ошибка	Пример	Последствие
Изменение массива без const, когда не нужно	Функция случайно меняет массив	Нежелательные побочные эффекты
Рекурсия без базового случая	Вызов без условия выхода	Переполнение стека

```
// ОШИБКА: функция возвращает int, но return double с усечением
int getValue() {
    return 3.14;   // вернёт 3 (потеря данных)
}

// ОШИБКА: забыт return
int badFunction(int x) {
    x = x * 2;
    // нет return - неопределённое поведение!
}
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Функция для вычисления суммы элементов массива

```
//
==========================================================
=
// Программа: вычисление суммы элементов массива через функцию
// Цель: демонстрация передачи массива в функцию
```

```
//
==========================================================================
=

#include <iostream>

// Функция вычисления суммы элементов массива
// Параметры:
//   arr[] - массив целых чисел
//   size  - количество элементов в массиве
// Возвращает: сумму всех элементов
int arraySum(const int arr[], int size) {
    int sum = 0;                    // аккумулятор суммы
    for (int i = 0; i < size; i++) {
        sum += arr[i];              // добавляем каждый элемент
    }
    return sum;                     // возвращаем результат
}

// Функция для заполнения массива значениями от 1 до N
void fillSequential(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        arr[i] = i + 1;             // 1, 2, 3, ..., size
    }
}

// Функция для вывода массива
void printArray(const int arr[], int size) {
    std::cout << "[";
```

```
    for (int i = 0; i < size; i++) {
        std::cout << arr[i];
        if (i < size - 1) {
            std::cout << ", ";
        }
    }
    std::cout << "]";
}

int main() {
    const int SIZE = 10;
    int numbers[SIZE];

    // Заполняем массив
    fillSequential(numbers, SIZE);

    // Выводим массив
    std::cout << "Массив: ";
    printArray(numbers, SIZE);
    std::cout << std::endl;

    // Вычисляем сумму через функцию
    int total = arraySum(numbers, SIZE);
    std::cout << "Сумма элементов: " << total << std::endl;

    return 0;
}

// Пример работы:
```

```
// Массив: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
// Сумма элементов: 55
```

3.2. Пример 2: Функции для поиска минимума и максимума

```
//
==========================================================================
=
// Программа: поиск минимума и максимума через функции
// Цель: демонстрация возврата значений и параметров-ссылок
//
==========================================================================
=

#include <iostream>
#include <cstdlib>   // для rand(), srand()
#include <ctime>     // для time()

// Функция поиска минимального элемента
// Возвращает: значение минимального элемента
int findMin(const int arr[], int size) {
    int minVal = arr[0];                    // предполагаем, что первый -
минимальный
    for (int i = 1; i < size; i++) {
        if (arr[i] < minVal) {
            minVal = arr[i];
        }
    }
    return minVal;
```

```
}

// Функция поиска максимального элемента
// Возвращает: значение максимального элемента
int findMax(const int arr[], int size) {
    int maxVal = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] > maxVal) {
            maxVal = arr[i];
        }
    }
    return maxVal;
}

// Функция поиска минимума и максимума одновременно (через ссылки)
void findMinMax(const int arr[], int size, int& minVal, int& maxVal) {
    minVal = arr[0];
    maxVal = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] < minVal) minVal = arr[i];
        if (arr[i] > maxVal) maxVal = arr[i];
    }
}

// Функция заполнения массива случайными числами
void fillRandom(int arr[], int size, int min, int max) {
    for (int i = 0; i < size; i++) {
        arr[i] = min + std::rand() % (max - min + 1);
```

```
    }
}

// Функция вывода массива
void printArray(const int arr[], int size) {
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

int main() {
    const int SIZE = 10;
    int numbers[SIZE];

    // Инициализация генератора случайных чисел
    std::srand(static_cast<unsigned>(std::time(nullptr)));

    // Заполняем массив случайными числами от 1 до 100
    fillRandom(numbers, SIZE, 1, 100);

    // Выводим массив
    std::cout << "Массив: ";
    printArray(numbers, SIZE);

    // Поиск минимума и максимума
    int min = findMin(numbers, SIZE);
    int max = findMax(numbers, SIZE);
```

```
    std::cout << "Минимум: " << min << std::endl;
    std::cout << "Максимум: " << max << std::endl;

    // Одновременный поиск через ссылки
    int min2, max2;
    findMinMax(numbers, SIZE, min2, max2);
    std::cout << "Через ссылки: min=" << min2 << ", max=" << max2 <<
std::endl;

    return 0;
}

// Пример работы:
// Массив: 45 67 23 89 12 34 78 56 91 5
// Минимум: 5
// Максимум: 91
```

3.3. Пример 3: Функция проверки простоты числа

```
//
==========================================================
=
// Программа: проверка, является ли число простым
// Цель: демонстрация функции с одним параметром и возвратом bool
//
==========================================================
=

#include <iostream>
```

```cpp
#include <cmath>     // для sqrt()

// Функция проверки, является ли число простым
// Простое число - натуральное число > 1, которое делится только на 1 и на себя
bool isPrime(int n) {
    // 1 и меньше - не простые
    if (n <= 1) {
        return false;
    }

    // 2 - простое
    if (n == 2) {
        return true;
    }

    // Чётные числа > 2 - не простые
    if (n % 2 == 0) {
        return false;
    }

    // Проверяем делители от 3 до sqrt(n) с шагом 2 (только нечётные)
    int limit = static_cast<int>(std::sqrt(n));
    for (int i = 3; i <= limit; i += 2) {
        if (n % i == 0) {
            return false;   // найден делитель - число не простое
        }
    }
    return true;            // делителей не найдено - число простое
```

```
    }

    // Функция вывода всех простых чисел в диапазоне
    void printPrimesInRange(int start, int end) {
        std::cout << "Простые числа в диапазоне [" << start << ", " << end <<
"]: ";
        for (int i = start; i <= end; i++) {
            if (isPrime(i)) {
                std::cout << i << " ";
            }
        }
        std::cout << std::endl;
    }

    int main() {
        int number;

        std::cout << "Введите число: ";
        std::cin >> number;

        if (isPrime(number)) {
            std::cout << number << " - простое число" << std::endl;
        } else {
            std::cout << number << " - составное число" << std::endl;
        }

        // Демонстрация поиска простых чисел в диапазоне
        printPrimesInRange(1, 100);
```

```
    return 0;
}

// Пример работы:
// Ввод: 17
// Вывод: 17 - простое число
// Простые числа в диапазоне [1, 100]: 2 3 5 7 11 13 17 19 23 29 31 37 41
43 47 53 59 61 67 71 73 79 83 89 97
```

3.4. Пример 4: Декомпозиция задачи на функции (калькулятор)

```
//
==========================================================
=
// Программа: консольный калькулятор с использованием функций
// Цель: демонстрация полной декомпозиции задачи на функции
//
==========================================================
=

#include <iostream>
#include <limits>   // для std::numeric_limits

// Очистка буфера ввода после некорректного ввода
void clearInput() {
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}
```

```
// Функция ввода числа с защитой от ошибок
double getNumber(const std::string& prompt) {
    double value;
    std::cout << prompt;
    std::cin >> value;

    while (std::cin.fail()) {
        std::cout << "Ошибка! Введите число: ";
        clearInput();
        std::cin >> value;
    }
    return value;
}

// Арифметические функции
double add(double a, double b) {
    return a + b;
}

double subtract(double a, double b) {
    return a - b;
}

double multiply(double a, double b) {
    return a * b;
}

double divide(double a, double b) {
    if (b == 0) {
```

```
        std::cout << "Ошибка: деление на ноль!" << std::endl;
        return 0;
    }
    return a / b;
}

// Функция вывода меню
void printMenu() {
    std::cout << "\n===== КАЛЬКУЛЯТОР =====" << std::endl;
    std::cout << "1. Сложение (+)" << std::endl;
    std::cout << "2. Вычитание (-)" << std::endl;
    std::cout << "3. Умножение (*)" << std::endl;
    std::cout << "4. Деление (/)" << std::endl;
    std::cout << "0. Выход" << std::endl;
    std::cout << "Выберите операцию: ";
}

// Функция выбора операции и выполнения вычислений
void runCalculator() {
    int choice;
    double a, b, result;

    do {
        printMenu();
        std::cin >> choice

        if (std::cin.fail()) {
            clearInput();
            std::cout << "Неверный ввод. Попробуйте снова." << std::endl;
```

```
            continue;
        }

        if (choice == 0) {
            std::cout << "До свидания!" << std::endl;
            break;
        }

        if (choice >= 1 && choice <= 4) {
            a = getNumber("Введите первое число: ");
            b = getNumber("Введите второе число: ");

            switch (choice) {
                case 1: result = add(a, b); break;
                case 2: result = subtract(a, b); break;
                case 3: result = multiply(a, b); break;
                case 4: result = divide(a, b); break;
            }

            std::cout << "Результат: " << result << std::endl;
        } else {
            std::cout << "Неверный выбор. Попробуйте снова." << std::endl;
        }
    } while (choice != 0);
}

int main() {
    runCalculator();
    return 0;
```

}

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Написать функцию для выполнения одного базового вычисления.

	Задание	**Прототип функции**
	Вычислить сумму двух целых чисел	int sum(int a, int b)
	Вычислить разность двух целых чисел	int diff(int a, int b)
	Вычислить произведение двух целых чисел	int product(int a, int b)
	Вычислить частное двух вещественных чисел	double divide(double a, double b)
	Вычислить квадрат числа	int square(int x)
	Вычислить куб числа	int cube(int x)
	Вычислить площадь прямоугольника	double rectangleArea(double a, double b)
	Вычислить площадь круга	double circleArea(double radius)
	Вычислить периметр прямоугольника	double

	Задание	Прототип функции
		rectanglePerimeter(double a, double b)
0	Вычислить длину окружности	double circumference(double radius)
1	Вычислить среднее арифметическое двух чисел	double average(double a, double b)
2	Вычислить факториал числа ($n \leq 10$)	int factorial(int n)
3	Проверить, является ли число чётным	bool isEven(int n)
4	Найти максимум двух чисел	int max(int a, int b)
5	Найти минимум двух чисел	int min(int a, int b)

Средний уровень (оценка «хорошо»)

Написать функцию для обработки массива или составного вычисления.

№	Задание	Прототип функции
1	Найти сумму элементов массива	int arraySum(const int arr[], int size)
2	Найти среднее арифметическое элементов массива	double arrayAverage(const int arr[], int size)
3	Найти минимальный элемент массива	int arrayMin(const int arr[], int size)
4	Найти максимальный элемент массива	int arrayMax(const int arr[], int size)
5	Найти количество чётных элементов массива	int countEven(const int arr[], int size)
6	Найти количество положительных элементов массива	int countPositive(const int arr[], int size)
7	Проверить, является ли число простым	bool isPrime(int n)
8	Найти сумму цифр числа	int digitSum(int n)
9	Проверить, является ли число палиндромом	bool isPalindrome(int n)
1	Вычислить НОД	int gcd(int a, int b)

№	Задание	Прототип функции
0	(наибольший общий делитель)	
1 1	Вычислить НОК (наименьшее общее кратное)	int lcm(int a, int b)
1 2	Проверить, является ли число степенью двойки	bool isPowerOfTwo(int n)
1 3	Вычислить значение многочлена $a \cdot x^2 + b \cdot x + c$	double quadratic(double a, double b, double c, double x)
1 4	Заполнить массив случайными числами	void fillRandom(int arr[], int size, int min, int max)
1 5	Вывести массив на экран	void printArray(const int arr[], int size)

Повышенный уровень (оценка «отлично»)

Комплексная программа с несколькими функциями и полной декомпозицией.

№	Задание
1	Анализ массива: функции для заполнения, вывода, нахождения суммы, среднего, минимума, максимума, количества положительных/отрицательных. Главная функция демонстрирует работу всех функций.
2	Поиск простых чисел: функция isPrime() и

№	Задание
	функция printPrimesInRange(start, end), выводящая все простые числа в заданном диапазоне.
3	Статистика строки: функции для подсчёта букв, цифр, гласных, согласных, слов в строке. Главная функция вызывает их и выводит отчёт.
4	Калькулятор с историей: арифметические функции + функция сохранения результата в массив истории + функция вывода истории.
5	Сортировка массива: функции для заполнения, вывода, сортировки пузырьком, бинарного поиска элемента.
6	Числа Фибоначчи: функция для вычисления n-го числа Фибоначчи (рекурсивно и итеративно), функция вывода последовательности до n.
7	Обработка матрицы: функции для заполнения, вывода, поиска суммы по строкам, суммы по столбцам, транспонирования.
8	Система управления студентами: функции для ввода данных студента (имя, возраст, группа), вывода списка, поиска по группе.
9	Генератор паролей: функция генерации пароля заданной длины (буквы + цифры), функция проверки надёжности пароля.
10	Конвертер систем счисления: функции для перевода из десятичной в двоичную, восьмеричную, шестнадцатеричную и

№	Задание
	обратно.
1	Решение квадратного уравнения: функция solveQuadratic(a, b, c, &x1, &x2), возвращающая количество корней и сами корни через ссылки.
2	Игра "Угадай число": функции: генерация числа, ввод догадки, проверка, подсчёт попыток, вывод статистики.
3	Калькулятор комплексных чисел: структура Complex (позже) или функции для сложения, вычитания, умножения, деления комплексных чисел.
4	Текстовый редактор: функции: ввод строки, вывод, реверс, замена подстроки, подсчёт слов, выход (меню через switch).
5	Многофункциональный калькулятор: меню с 10+ операциями, каждая операция - отдельная функция, защита от ошибок ввода.

5. Контрольные вопросы

Раздел А. Основы функций

1. Что такое функция? Какие преимущества даёт использование функций?
2. Каков синтаксис определения функции? Из каких частей она состоит?
3. Что такое прототип функции? Для чего он нужен?
4. Чем отличаются параметры от аргументов функции?

5. Как вызвать функцию? Что произойдёт при вызове?

Раздел Б. Передача параметров

1. Что означает передача параметров по значению? Как это работает?
2. Как передать массив в функцию? Что при этом передаётся на самом деле?
3. Почему при передаче массива нужно передавать его размер отдельным параметром?
4. Что такое const параметр? Когда его следует использовать?
5. Как вернуть несколько значений из функции? Какие способы существуют?

Раздел В. Возврат значений

1. Что такое void функция? Нужен ли в ней return?
2. Что произойдёт, если в функции, объявленной как int, забыть return?
3. Может ли функция вернуть массив? Почему?
4. Что такое локальные переменные? Когда они создаются и уничтожаются?
5. Чем отличаются глобальные переменные от локальных? Почему глобальные плохи?

Раздел Г. Рекурсия и перегрузка

1. Что такое рекурсия? Из каких частей состоит рекурсивная функция?
2. Что такое базовый случай в рекурсии? Что произойдёт без него?
3. Что такое перегрузка функций? Приведите пример.
4. По каким критериям компилятор различает перегруженные функции?
5. Можно ли перегрузить функции, отличающиеся только типом возвращаемого значения? Почему?

Раздел Д. Анализ кода и ошибки

1. Что выведет код?

```
void f(int x) { x = 10; }
int main() { int a = 5; f(a); std::cout << a; return 0; }
```

2. Найдите ошибку:

```
int sum(int a, b) {
    return a + b;
}
```

3. Что не так с этой рекурсией?

```
int countdown(int n) {
    std::cout << n << " ";
    return countdown(n - 1);
}
```

4. Почему sizeof(arr) в функции не работает ожидаемым образом?

```
void func(int arr[]) {
    int size = sizeof(arr) / sizeof(arr[0]);
}
```

5. Что такое std::abs() и чем отличается от abs() без std::?

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (определение функций, синтаксис, параметры, возврат значений, прототипы, рекурсия, перегрузка)

Ход выполнения работы:

Код программы с подробными комментариями (каждая функция пояснена)

Блок-схема алгоритма (для одной из функций или для main)

Скриншоты выполнения для различных сценариев

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали о функциях, как они помогают в декомпозиции)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №9)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) - правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, правильное определение и вызов функции, возврат значения).

Оценка «хорошо» (4) - правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего уровня (работа с массивами, передача const массива, правильная обработка размеров, осмысленные имена функций).

Оценка «отлично» (5) - правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (полная декомпозиция задачи на 5+ функций, использование прототипов, обработка ошибок ввода через функции, подробные комментарии в коде, блок-схема в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в

виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
1	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
2	Компиляция и запуск			
2.1	Программа компилируется без ошибок	✔	✔	✔
2.2	Программа успешно запускается	✔	✔	✔
2.3	Отсутствие предупреждений компилятора	—	✔	✔
3	Определение функций			

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
3.1	Правильный синтаксис определения функции	✔	✔	✔
3.2	Правильный выбор типа возвращаемого значения	✔	✔	✔
3.3	Правильное именование функций (глаголы)	—	✔	✔
3.4	Использование прототипов функций	—	✔	✔
4	Параметры функций			
4.1	Правильное описание параметров	✔	✔	✔
4.2	Использование const для параметров-массивов	—	✔	✔

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
4.3	Передача размера массива отдельным параметром	—	✔	✔
5	Вызов функций			
5.1	Корректный вызов функций из main()	✔	✔	✔
5.2	Использование возвращаемого значения	✔	✔	✔
5.3	Передача аргументов правильного типа	✔	✔	✔
6	Организация программы			
6.1	Логическая декомпозиция задачи на функции	—	✔	✔
6	Отсутствие	—	✔	✔

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	дублирования кода (DRY)			
6.3	Каждая функция выполняет одну задачу	—	—	✔
6.4	Размер каждой функции ≤ 20-30 строк	—	—	✔
7	Алгоритмы			
7.1	Корректная реализация требуемого алгоритма	✔	✔	✔
7.2	Работа с массивами в функциях	—	✔	✔
7.3	Рекурсивные функции (где требуется)	—	—	✔
7	Перегрузка	—	—	✔

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.4	функций (где требуется)			
8	Обработка ошибок			
8.1	Проверка корректности ввода в функциях	—	✔	✔
8.2	Проверка граничных условий (пустой массив, n≤0)	—	✔	✔
8.3	Защита от деления на ноль	—	✔	✔
9	Качество кода			
9.1	Правильные отступы и форматирование	✔	✔	✔
9.2	Использование осмысленных имён функций	—	✔	✔

№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
9.3	Наличие комментариев (описание каждой функции)	—	✔	✔
9.4	Документация функций (что делает, параметры, возврат)	—	—	✔
10	Отчёт			
10.1	Титульный лист, цель, ход работы	✔	✔	✔
10.2	Скриншоты выполнения (минимум 3 теста)	✔	✔	✔
10.3	Скриншоты для всех трёх уровней	—	✔	✔
10.4	Таблица тестирования	—	✔	✔
11	Блок-схема	—	✔	✔

	№	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
0.5		алгоритма			
0.6	1	Тестирование на граничных значениях	—	—	✔
0.7	1	Ответы на контрольные вопросы	—	✔	✔

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) - обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня (№1–15).

Написать корректное определение функции с правильным синтаксисом.

Реализовать вызов функции из main().

Использовать возвращаемое значение функции (вывод на экран или присваивание).

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) - дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Написать функцию, работающую с массивом (передача массива и размера).

Использовать const для защиты массива от изменений (где нужно).

Корректно обработать граничные случаи (пустой массив, $n \leq 0$).

Дать функциям осмысленные имена (глаголы, описывающие действие).

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

Блок-схему алгоритма

Повышенный уровень (оценка 5) - требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Полная декомпозиция задачи:

Минимум 5 функций (включая функции ввода, вывода, обработки)

Каждая функция выполняет одну задачу (SRP - Single Responsibility Principle)

Использование прототипов функций:

Все функции объявлены до main() с прототипами

Определения расположены после main() (или в отдельном файле)

Обработка ошибок:

Функции проверяют корректность входных данных

Информативные сообщения об ошибках

Документация функций:

Комментарий перед каждой функцией с описанием:

Что делает функция

Что означают параметры

Что возвращает

В отчёте дополнительно представить:

Тестирование на граничных значениях

Диаграмму вызовов функций (call graph)

7.4. Примеры проверки соответствия

Выполненные задания	Оценка	Проверка критериев
Только базовый вариант №1 (сумма двух чисел)	3	✔ Определение функции, ✔ вызов, ✔ возврат значения. Но: ✘ нет массива, ✘ нет const, ✘ нет декомпозиции
Базовый №1 + средний №1 (сумма массива)	4	✔ Плюс передача массива, ✔ const, ✔ проверка размера. Но: ✘ нет нескольких функций, ✘ нет документации
Базовый №1 + средний №1 + повышенный №1 (анализ массива)	5	✔ Все критерии: ✔ 5+ функций, ✔ прототипы, ✔ документация, ✔ обработка ошибок, ✔ блок-схема

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	**Влияние**	**Как исправить**
Забыт прототип функции (определение	Ошибка компиляции	Добавить прототип перед main()

Ошибка	Влияние	Как исправить
после вызова)		
Несоответствие типа возврата	Неопределённое поведение	Убедиться, что return возвращает правильный тип
Передача массива без передачи размера	Невозможно определить длину	Добавить параметр int size
Изменение массива в функции без необходимости	Побочные эффекты	Использовать const для параметра-массива
Глобальные переменные вместо параметров	Плохой стиль, трудно тестировать	Передавать данные через параметры
Рекурсия без базового случая	Переполнение стека	Всегда проверять условие выхода
Функция слишком большая (>50 строк)	Трудно читать, тестировать	Разбить на более мелкие функции

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Синтаксис определения и вызова функции

Разницу между параметрами и аргументами

Как передаётся массив в функцию и зачем нужен const

(для оценки 4) - как выполнить декомпозицию задачи

(для оценки 5) - как работают прототипы и рекурсия

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	**Проверяет**
"Как изменить функцию, чтобы она возвращала несколько значений?"	Понимание ссылок/структур
"Что произойдёт, если вызвать factorial(10000) рекурсивно?"	Понимание ограничений рекурсии
"Как перегрузить функцию max для двух и трёх аргументов?"	Понимание перегрузки
"Почему использование глобальных переменных считается плохой практикой?"	Понимание принципов модульности

Лабораторная работа №10

Передача параметров. Ссылки

1. Цель работы

Понять разницу между передачей параметров по значению, по указателю и по ссылке. Научиться использовать ссылки для модификации переменных в вызывающей функции, реализовать функцию swap (обмен значений), освоить передачу массивов и структур по ссылке для повышения эффективности, научиться возвращать несколько значений из функции с помощью ссылочных параметров.

2. Теоретическое обоснование

2.1. Три способа передачи параметров в C++

В C++ существует три основных способа передачи параметров в функции:

Способ передачи	**Синтаксис**	**Что передаётся**	**Может ли функция изменить исходную переменную?**
По значению	int x	Копия значения	Нет
По указателю	int* ptr	Адрес переменной	Да (через разыменование *ptr)
По ссылке	int& ref	Ссылка на переменную (псевдоним)	Да

Визуализация:

```
Переменная в main:      int a = 10;  (адрес: 0x7fff, значение: 10)

Передача по значению:   void func(int x)   → x = 10 (копия, отдельный адрес)
Передача по указателю:  void func(int* p)  → p = 0x7fff (адрес a)
Передача по ссылке:     void func(int& r)  → r — это псевдоним для a (тот же адрес)
```

Переменная в main: int a = 10; (адрес: 0x7fff, значение: 10)

Передача по значению: void func(int x) → x = 10 (копия, отдельный адрес)

Передача по указателю: void func(int* p) → p = 0x7fff (адрес a)

Передача по ссылке: void func(int& r) → r - это псевдоним для a (тот же адрес)

2.2. Передача по значению (повторение)

При передаче по значению функция получает копию аргумента. Изменения параметра внутри функции не влияют на исходную переменную.

```
#include <iostream>

void changeByValue(int x) {
    x = 100;                    // меняем копию
    std::cout << "Внутри: " << x << std::endl;  // 100
}

int main() {
    int a = 10;
    changeByValue(a);
    std::cout << "Снаружи: " << a << std::endl;  // 10 (не изменилось!)
    return 0;
}
// Вывод:
// Внутри: 100
// Снаружи: 10
```

Когда использовать: когда функция не должна изменять исходные данные, или когда работа с копией безопасна и эффективна (для простых типов).

2.3. Передача по ссылке (основная тема)

Ссылка - это псевдоним (альтернативное имя) для уже существующей переменной. Ссылка не является отдельным объектом в памяти - она просто даёт другое имя той же переменной.

Синтаксис объявления ссылки: тип& имя = переменная;

```
int a = 10;
int& ref = a;      // ref - ссылка на a (ref и a - одно и то же)

ref = 20;          // теперь a = 20
std::cout << a;    // 20
```

Передача параметра по ссылке:

```
#include <iostream>

void changeByReference(int& x) {      // x - ссылка на переданную переменную
    x = 100;                          // меняем исходную переменную
    std::cout << "Внутри: " << x << std::endl;  // 100
}

int main() {
    int a = 10;
    changeByReference(a);             // передаём a по ссылке
```

```
    std::cout << "Снаружи: " << a << std::endl;  // 100 (изменилось!)
    return 0;
}
// Вывод:
// Внутри: 100
// Снаружи: 100
```

Визуализация:

2.4. Передача по указателю (для сравнения)

Указатель - это переменная, которая хранит адрес другой переменной. В отличие от ссылки, указатель:

Может быть nullptr (не ссылаться ни на что)

Может менять адрес, на который указывает

Требует разыменования (*ptr) для доступа к значению

```
#include <iostream>

void changeByPointer(int* ptr) {   // ptr - указатель на int
    if (ptr != nullptr) {
        *ptr = 100;              // разыменование и изменение
```

```
    }
}

int main() {
    int a = 10;
    changeByPointer(&a);          // передаём адрес a
    std::cout << a                // 100
    return 0;
}
```

Сравнение ссылок и указателей:

Характеристика	Ссылка (&)	Указатель (*)
Может быть nullptr	Нет	Да
Может менять объект	Нет (привязана навсегда)	Да
Синтаксис доступа	Прямой (ref = value)	Разыменование (*ptr = value)
Адресная арифметика	Нет	Да (ptr++, ptr+5)
Обязательность инициализации	Да	Нет (но опасно)
Рекомендация C++	Предпочтительнее	Для низкоуровневых

Характеристика	Ссылка (&)	Указатель (*)
		операций

2.5. Функция swap (обмен значений) - классический пример

```
#include <iostream>

// Обмен значений двух переменных через ссылки
void swap(int& a, int& b) {
    int temp = a;   // сохраняем старое значение a
    a = b;          // a = b
    b = temp;       // b = старое a
}

int main() {
    int x = 5, y = 10;

    std::cout << "До swap: x = " << x << ", y = " << y << std::endl;  // 5, 10

    swap(x, y);     // передаём x и y по ссылке

    std::cout << "После swap: x = " << x << ", y = " << y << std::endl; // 10, 5

    return 0;
}
```

Почему без ссылок не работает:

```
// НЕПРАВИЛЬНО: передача по значению
void badSwap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;     // меняются только копии, исходные переменные не изменятся
}
```

2.6. Передача больших объектов по ссылке (оптимизация)

При передаче по значению создаётся копия объекта. Для больших структур или классов это может быть дорого. Передача по константной ссылке позволяет избежать копирования, сохраняя защиту от изменений.

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

// ПЛОХО: передача по значению - копирование всей строки (O(n))
void printStringByValue(std::string s) {
    std::cout << s << std::endl;
    // s - копия, изменения не видны снаружи
}

// ХОРОШО: передача по константной ссылке - без копирования (O(1))
void printStringByReference(const std::string& s) {
    std::cout << s << std::endl;
    // s - ссылка на исходную строку, но изменить нельзя (const)
}
```

```
// ЕСЛИ НУЖНО ИЗМЕНИТЬ: передача по ссылке
void toUpperCase(std::string& s) {
    for (char& c : s) {
        c = std::toupper(c);
    }
}

int main() {
    std::string text = "Hello, World!";

    printStringByValue(text);      // создаётся копия
    printStringByReference(text);  // копии нет
    toUpperCase(text);             // изменяем исходную строку
    std::cout << text;             // "HELLO, WORLD!"

    return 0;
}
```

Правила использования:

1. Для простых типов (int, double, char) - передача по значению (дешево)
2. Для больших объектов (std::string, std::vector, структуры) - по константной ссылке
3. Если нужно изменить - по ссылке (не const)
4. Если нужно изменить или может быть nullptr - по указателю

2.7. Возврат нескольких значений из функции через ссылки

Функция может вернуть только одно значение. Для возврата нескольких значений используются ссылочные параметры (параметры-результаты).

```cpp
#include <iostream>

// Функция возвращает сумму, а минимум и максимум - через ссылки
int sumAndMinMax(const int arr[], int size, int& minVal, int& maxVal) {
    if (size <= 0) {
        minVal = maxVal = 0;
        return 0;
    }

    int sum = 0;
    minVal = arr[0];
    maxVal = arr[0];

    for (int i = 0; i < size; i++) {
        sum += arr[i];
        if (arr[i] < minVal) minVal = arr[i];
        if (arr[i] > maxVal) maxVal = arr[i];
    }

    return sum;
}

int main() {
    int data[] = {3, 7, 1, 9, 4, 2, 8};
    int size = sizeof(data) / sizeof(data[0]);

    int minimum, maximum;
    int total = sumAndMinMax(data, size, minimum, maximum);
```

```
    std::cout << "Сумма: " << total << std::endl;       // 34
    std::cout << "Минимум: " << minimum << std::endl;   // 1
    std::cout << "Максимум: " << maximum << std::endl;  // 9

    return 0;
}
```

Другие способы возврата нескольких значений:

std::pair<T1, T2> - для двух значений

std::tuple<T1, T2, ...> - для трёх и более

Пользовательская структура (struct)

2.8. Передача массива по ссылке (с фиксированным размером)

Для передачи массива с сохранением размера можно использовать ссылку на массив:

```
// Передача массива фиксированного размера по ссылке
void processArray(int (&arr)[5]) {   // arr - ссылка на массив из 5 int
    for (int i = 0; i < 5; i++) {
        arr[i] *= 2;
    }
}

int main() {
    int numbers[5] = {1, 2, 3, 4, 5};
    processArray(numbers);    // передаём массив по ссылке

    for (int x : numbers) {
```

```
        std::cout << x << " ";   // 2 4 6 8 10
    }
    return 0;
}
```

Шаблонная версия (для любого размера):

```
template<size_t N>
void processArray(int (&arr)[N]) {   // N выводится автоматически
    for (int i = 0; i < N; i++) {
        arr[i] *= 2;
    }
}
```

2.9. Константные ссылки (const T&)

Константная ссылка позволяет:

1. Избежать копирования (эффективность)
2. Запретить изменение объекта (безопасность)
3. Принимать временные объекты (литералы, результаты выражений)

```
// Принимает const ссылку - можно передать и переменную, и временный объект
void print(const std::string& s) {
    std::cout << s << std::endl;
    // s = "new";   // ОШИБКА! нельзя изменить const ссылку
}

int main() {
    std::string text = "Hello";
```

```
    print(text);          // OK: переменная
    print("World");       // OK: временный объект (литерал)
    print(text + "!");    // OK: временный объект (результат выражения)
    return 0;
}
```

Важно: Временные объекты (литералы, результаты выражений) нельзя привязать к неконстантной ссылке.

```
void modify(std::string& s) { s = "new"; }

int main() {
    modify("Hello");      // ОШИБКА! нельзя привязать временный объект к &
    return 0;
}
```

2.10. Ссылка как возвращаемое значение

Функция может возвращать ссылку. Это позволяет использовать результат функции в левой части присваивания (lvalue).

```
// Функция возвращает ссылку на элемент массива
int& getElement(int arr[], int index) {
    return arr[index];
}

int main() {
    int numbers[5] = {1, 2, 3, 4, 5};
```

```
    getElement(numbers, 2) = 100;   // меняем 3-й элемент через ссылку
    std::cout << numbers[2];        // 100

    return 0;
}
```

Осторожно: Не возвращайте ссылку на локальную переменную! Она

```
// ОШИБКА: возврат ссылки на локальную переменную
int& badFunction() {
    int x = 10;
    return x;   // x уничтожится после выхода - висячая ссылка!
}
```

2.11. std::swap из стандартной библиотеки

В C++ уже есть готовая функция обмена - std::swap из <utility> (или <algorithm>).

```
#include <algorithm>
#include <iostream>

int main() {
    int a = 5, b = 10;
    std::swap(a, b);          // встроенная функция обмена
    std::cout << a << " " << b;  // 10 5
    return 0;
}
```

2.12. Типичные ошибки при работе со ссылками

Ошибка	Пример	После дствие
Неинициализированная ссылка	int& ref;	Ошибка компиляции
Привязка временного объекта к неконстантной ссылке	void f(int&); f(5);	Ошибка компиляции
Возврат ссылки на локальную переменную	int& f() { int x; return x; }	Висячая ссылка (UB)
Путаница между & в объявлении и & как адрес	int& ref = *ptr;	Путаница в синтаксисе
Изменение const ссылки	void f(const int& x) { x = 5; }	Ошибка компиляции
Ссылка на указатель vs указатель на ссылку	int&* ptr;	Ссылок на ссылки не бывает

```
// ОШИБКА: ссылка должна быть инициализирована
int& ref;  // ОШИБКА!

// ОШИБКА: нельзя изменить привязку ссылки
int a = 5, b = 10;
int& ref = a;  // ref - ссылка на a
ref = b;           // Это не перенаправление! Это a = b (присваивание значения)
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Функция swap (обмен значений)

```
//
==========================================================
=
// Программа: обмен значений двух переменных через ссылки
// Цель: демонстрация передачи по ссылке для модификации
//
==========================================================
=

#include <iostream>

// Функция обмена значениями двух целых чисел
// Параметры: a и b - ссылки на переменные в вызывающем коде
// Возвращает: void (ничего не возвращает, но изменяет исходные
переменные)
void swap(int& a, int& b) {
    int temp = a;   // сохраняем старое значение a
    a = b;          // записываем в a значение b
    b = temp;       // записываем в b сохранённое значение
}

// Перегрузка для double
void swap(double& a, double& b) {
    double temp = a;
    a = b;
```

```
    b = temp;
}

// Функция для демонстрации неправильного подхода (по значению)
void badSwap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
    std::cout << "Внутри badSwap: a=" << a << ", b=" << b << std::endl;
}

int main() {
    int x = 5, y = 10;

    std::cout << "Исходные значения: x = " << x << ", y = " << y <<
std::endl;

    // Пробуем неправильный обмен (по значению)
    badSwap(x, y);
    std::cout << "После badSwap: x = " << x << ", y = " << y << std::endl;
    // x и y не изменились!

    // Правильный обмен (по ссылке)
    swap(x, y);
    std::cout << "После swap: x = " << x << ", y = " << y << std::endl;

    // Демонстрация с double
    double d1 = 3.14, d2 = 2.71;
    std::cout << "\nДо swap double: " << d1 << ", " << d2 << std::endl;
```

```
    swap(d1, d2);
    std::cout << "После swap double: " << d1 << ", " << d2 << std::endl;

    return 0;
}

// Пример работы:
// Исходные значения: x = 5, y = 10
// Внутри badSwap: a=10, b=5
// После badSwap: x = 5, y = 10 (не изменились!)
// После swap: x = 10, y = 5
```

3.2. Пример 2: Заполнение массива через функцию со ссылкой

```
//
==========================================================
=
// Программа: заполнение массива через функцию
// Цель: демонстрация передачи больших объектов по ссылке
//
==========================================================
=

#include <iostream>
#include <cstdlib>
#include <ctime>

// Функция заполнения массива случайными числами
```

```
// Параметр: arr - ссылка на массив (в данном случае указатель, но для вектора была бы ссылка)
//           size - размер массива
//           min, max - диапазон случайных чисел
void fillArray(int arr[], int size, int min, int max) {
    for (int i = 0; i < size; i++) {
        arr[i] = min + std::rand() % (max - min + 1);
    }
}

// Функция вывода массива (const ссылка для защиты от изменений)
// Используем указатель, но для наглядности показываем const
void printArray(const int arr[], int size) {
    std::cout << "[";
    for (int i = 0; i < size; i++) {
        std::cout << arr[i];
        if (i < size - 1) std::cout << ", ";
    }
    std::cout << "]";
}

// Функция, которая возвращает несколько значений через ссылки
// Возвращает: сумму, а min и max через ссылки
int sumAndMinMax(const int arr[], int size, int& minVal, int& maxVal) {
    if (size <= 0) {
        minVal = maxVal = 0;
        return 0;
    }
```

```
    int sum = 0;
    minVal = arr[0];
    maxVal = arr[0];

    for (int i = 0; i < size; i++) {
        sum += arr[i];
        if (arr[i] < minVal) minVal = arr[i];
        if (arr[i] > maxVal) maxVal = arr[i];
    }

    return sum;
}

int main() {
    const int SIZE = 10;
    int numbers[SIZE];

    // Инициализация генератора случайных чисел
    std::srand(static_cast<unsigned>(std::time(nullptr)));

    // Заполняем массив
    fillArray(numbers, SIZE, 10, 99);

    // Выводим массив
    std::cout << "Массив: ";
    printArray(numbers, SIZE);
    std::cout << std::endl;

    // Получаем статистику
```

```
    int min, max;
    int sum = sumAndMinMax(numbers, SIZE, min, max);

    std::cout << "Сумма: " << sum << std::endl;
    std::cout << "Минимум: " << min << std::endl;
    std::cout << "Максимум: " << max << std::endl;

    return 0;
}

// Пример работы:
// Массив: [45, 67, 23, 89, 12, 34, 78, 56, 91, 5]
// Сумма: 500
// Минимум: 5
// Максимум: 91
```

3.3. Пример 3: Функция с константной ссылкой (оптимизация)

```
//
==========================================================
=
// Программа: демонстрация эффективности константных ссылок
// Цель: показать, как избежать копирования больших объектов
//
==========================================================
=

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

```cpp
#include <chrono>   // для измерения времени (опционально)

// ПЛОХО: передача по значению - создаётся копия всей строки
void processByValue(std::string s) {
    // s - копия исходной строки
    int len = s.length();
    // здесь могла быть сложная обработка
}

// ХОРОШО: передача по константной ссылке - без копирования
void processByReference(const std::string& s) {
    // s - ссылка на исходную строку
    int len = s.length();
    // здесь могла быть сложная обработка
}

// ЕСЛИ НУЖНО ИЗМЕНИТЬ: передача по ссылке
void toUpperCase(std::string& s) {
    for (char& c : s) {
        c = std::toupper(static_cast<unsigned char>(c));
    }
}

// Проверка, является ли строка палиндромом (используя константную ссылку)
bool isPalindrome(const std::string& s) {
    int left = 0;
    int right = s.length() - 1;
```

```
        while (left < right) {
            // Пропускаем не буквенно-цифровые символы
            while (left < right && !std::isalnum(s[left])) left++;
            while (left < right && !std::isalnum(s[right])) right--;

            if (std::tolower(s[left]) != std::tolower(s[right])) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }

    int main() {
        std::string text = "A man, a plan, a canal: Panama";
        std::string longText = "Это очень длинная строка, которая может содержать множество символов...";

        // Передача по константной ссылке - эффективно и безопасно
        if (isPalindrome(text)) {
            std::cout << "\"" << text << "\" - палиндром!" << std::endl;
        } else {
            std::cout << "\"" << text << "\" - не палиндром" << std::endl;
        }

        // Передача по ссылке для изменения
        std::string lowerText = "hello, world!";
        std::cout << "\nДо toUpperCase: " << lowerText << std::endl;
```

```
    toUpperCase(lowerText);
    std::cout << "После toUpperCase: " << lowerText << std::endl;

    // Демонстрация работы с временными объектами (только const ссылка!)
    processByReference("Временная строка");     // OK: const ссылка принимает временный объект
    // processByValue("Временная строка");    // Тоже OK, но создаётся копия

    return 0;
}

// Пример работы:
// "A man, a plan, a canal: Panama" - палиндром!
// До toUpperCase: hello, world!
// После toUpperCase: HELLO, WORLD!
```

3.4. Пример 4: Возврат нескольких значений через ссылки

```
//
==========
// Программа: статистика массива с возвратом нескольких значений
// Цель: демонстрация использования ссылок как выходных параметров
//
==========
```

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

// Структура для хранения статистики (альтернативный подход)
struct ArrayStats {
    int sum;
    int min;
    int max;
    double average;
    int evenCount;
    int oddCount;
};

// Вариант 1: возврат через ссылки (классический C++ подход)
void getStatistics(const int arr[], int size,
                   int& sum, int& min, int& max,
                   double& average, int& evenCount, int& oddCount) {
    if (size <= 0) {
        sum = 0;
        min = max = 0;
        average = 0.0;
        evenCount = oddCount = 0;
        return;
    }

    sum = 0;
    min = arr[0];
    max = arr[0];
```

```
    evenCount = 0;
    oddCount = 0;

    for (int i = 0; i < size; i++) {
        sum += arr[i];
        if (arr[i] < min) min = arr[i];
        if (arr[i] > max) max = arr[i];
        if (arr[i] % 2 == 0) {
            evenCount++;
        } else {
            oddCount++;
        }
    }

    average = static_cast<double>(sum) / size;
}

// Вариант 2: возврат структуры (современный C++ подход)
ArrayStats getStatisticsStruct(const int arr[], int size) {
    ArrayStats stats = {0, 0, 0, 0.0, 0, 0};

    if (size <= 0) return stats;

    stats.sum = 0;
    stats.min = arr[0];
    stats.max = arr[0];

    for (int i = 0; i < size; i++) {
        stats.sum += arr[i];
```

```
        if (arr[i] < stats.min) stats.min = arr[i];
        if (arr[i] > stats.max) stats.max = arr[i];
        if (arr[i] % 2 == 0) {
            stats.evenCount++;
        } else {
            stats.oddCount++;
        }
    }

    stats.average = static_cast<double>(stats.sum) / size;
    return stats;
}

// Функция заполнения массива
void fillRandom(int arr[], int size, int min, int max) {
    for (int i = 0; i < size; i++) {
        arr[i] = min + std::rand() % (max - min + 1);
    }
}

// Функция вывода массива
void printArray(const int arr[], int size) {
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

int main() {
```

```
    const int SIZE = 15;
    int numbers[SIZE];

    std::srand(static_cast<unsigned>(std::time(nullptr)));
    fillRandom(numbers, SIZE, 1, 100);

    std::cout << "Массив: ";
    printArray(numbers, SIZE);

    // Вариант 1: через ссылки
    int sum, min, max, evenCount, oddCount;
    double average;

    getStatistics(numbers, SIZE, sum, min, max, average, evenCount,
oddCount);

    std::cout << "\n=== Статистика (через ссылки) ===" << std::endl;
    std::cout << "Сумма: " << sum << std::endl;
    std::cout << "Среднее: " << average << std::endl;
    std::cout << "Минимум: " << min << std::endl;
    std::cout << "Максимум: " << max << std::endl;
    std::cout << "Чётных: " << evenCount << std::endl;
    std::cout << "Нечётных: " << oddCount << std::endl;

    // Вариант 2: через структуру
    ArrayStats stats = getStatisticsStruct(numbers, SIZE);

    std::cout << "\n=== Статистика (через структуру) ===" << std::endl;
    std::cout << "Сумма: " << stats.sum << std::endl;
```

```
    std::cout << "Среднее: " << stats.average << std::endl;
    std::cout << "Минимум: " << stats.min << std::endl;
    std::cout << "Максимум: " << stats.max << std::endl;
    std::cout << "Чётных: " << stats.evenCount << std::endl;
    std::cout << "Нечётных: " << stats.oddCount << std::endl;

    return 0;
}
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Написать функцию, использующую передачу по ссылке для изменения переменной.

	Задание	Прототип функции
	Увеличить число на заданное значение	void increment(int& value, int delta)
	Уменьшить число на заданное значение	void decrement(int& value, int delta)
	Удвоить значение переменной	void doubleValue(int& value)
	Утроить значение переменной	void tripleValue(int& value)
	Присвоить переменной абсолютное значение	void makeAbsolute(int& value)

	Задание	Прототип функции
	Обнулить переменную, если она отрицательная	void zeroIfNegative(int& value)
	Поменять знак переменной на противоположный	void negate(int& value)
	Установить значение в 0, если оно чётное	void zeroIfEven(int& value)
	Возвести число в квадрат	void square(int& value)
0	Вычислить обратное значение (1/x) для double	void inverse(double& value)
1	Установить минимум из двух чисел в первую переменную	void setMin(int& a, int b)
2	Установить максимум из двух чисел в первую переменную	void setMax(int& a, int b)
3	Обменять значения двух переменных	void swap(int& a, int& b)
4	Обменять значения двух вещественных переменных	void swap(double& a, double& b)

	Задание	Прототип функции
5	Обменять значения двух символов	void swap(char& a, char& b)

Средний уровень (оценка «хорошо»)

Использование ссылок для работы с массивами и возврата нескольких значений.

	Задание	Прототип функции
	Найти минимум и максимум массива	void findMinMax(const int arr[], int size, int& min, int& max)
	Найти сумму и количество элементов массива	void sumAndCount(const int arr[], int size, int& sum, int& count)
	Найти сумму положительных и отрицательных элементов	void sumPosNeg(const int arr[], int size, int& sumPos, int& sumNeg)
	Найти первый и последний положительный элемент	void firstLastPositive(const int arr[], int size, int& first, int& last)
	Заполнить массив и вернуть его сумму	int fillAndSum(int arr[], int size, int min, int max, int& minVal, int& maxVal)
	Преобразовать строку в верхний регистр	void toUpper(std::string& str)

	Задание	Прототип функции
	Преобразовать строку в нижний регистр	void toLower(std::string& str)
	Удалить все пробелы из строки	void removeSpaces(std::string& str)
	Заменить все вхождения символа в строке	void replaceChar(std::string& str, char oldChar, char newChar)
0	Реверс строки (на месте)	void reverseString(std::string& str)
1	Обменять значения двух целых чисел (используя арифметику без временной переменной)	void swapArithmetic(int& a, int& b)
2	Обменять значения двух целых чисел (используя XOR без временной переменной)	void swapXOR(int& a, int& b)
3	Вычислить корни квадратного уравнения	int solveQuadratic(double a, double b, double c, double& x1, double& x2)
4	Разложить число на цифры (сумма и произведение)	void digitStats(int n, int& sum, int& product)

	Задание	Прототип функции
5	Получить часы, минуты, секунды из общего количества секунд	void secondsToHMS(int totalSeconds, int& hours, int& minutes, int& seconds)

Повышенный уровень (оценка «отлично»)

Комплексные задачи с использованием ссылок, указателей и оптимизацией.

№	Задание
1	Статистика массива: функция, возвращающая сумму, среднее, min, max, количество чётных/нечётных через ссылки. Продемонстрировать оба подхода: через ссылки и через структуру.
2	Сортировка с обменом: реализовать сортировку пузырьком, используя функцию swap для обмена элементов. Показать эффективность ссылок.
3	Обработка строки: функции trimLeft, trimRight, trim (удаление пробелов в начале/конце) через ссылку на строку.
4	Разбиение массива: функция, которая разделяет массив на два: чётные и нечётные числа (возврат через ссылки на новые массивы).
5	Полином: функция вычисления значения полинома и его производной в точке через ссылки (возврат двух значений).
6	Статистика матрицы: функция, возвращающая сумму по строкам, сумму по столбцам, след матрицы через ссылки.

№	Задание
7	Калькулятор с историей: функции для вычислений (через ссылки для возврата результата и кода ошибки), история операций - массив структур.
8	Комплексные числа: структура Complex с полями re, im. Функции сложения, вычитания, умножения, деления (результат через ссылку).
9	Бинарный поиск: функция поиска элемента в отсортированном массиве, возвращающая индекс и флаг успеха через ссылку.
10	Связанный список (упрощённый): функции вставки, удаления, поиска с использованием ссылок на указатели (Node*& head).
11	Работа с файлом: функция чтения чисел из файла, возвращающая массив и его размер через ссылки (динамическое выделение).
12	Сжатие массива: функция удаления дубликатов из отсортированного массива, возвращающая новый размер через ссылку.
13	Продвинутый калькулятор: функция calculate, принимающая строку выражения и возвращающая результат и код ошибки через ссылки.
14	Множественная сортировка: функция, сортирующая массив и возвращающая количество операций обмена и сравнений через ссылки.

№	Задание
5	1 Мониторинг производительности: функция, измеряющая время выполнения другой функции, передаваемой через ссылку на функцию.

5. Контрольные вопросы

Раздел А. Основы ссылок

1. Что такое ссылка в C++? Как объявить ссылку?
2. Чем отличается ссылка от указателя? Перечислите основные различия.
3. Какие ограничения есть у ссылок? (Что нельзя делать со ссылками?)
4. Обязательно ли инициализировать ссылку при объявлении? Что произойдёт при попытке использовать неинициализированную ссылку?
5. Можно ли изменить объект, на который ссылается ссылка? Можно ли перенаправить ссылку на другой объект?

Раздел Б. Передача параметров

1. Чем отличается передача по значению от передачи по ссылке? Приведите пример.
2. Как передать массив по ссылке с сохранением размера?
3. Что такое константная ссылка? Когда её следует использовать?
4. Почему передача больших объектов по ссылке эффективнее, чем по значению?
5. Как вернуть несколько значений из функции с помощью ссылок?

Раздел В. Функция swap и обмен

1. Как реализовать функцию swap для двух целых чисел? Почему нужно передавать по ссылке?

2. Что делает std::swap? Из какого заголовочного файла она доступна?
3. Можно ли обменять значения без временной переменной? Приведите примеры.
4. Что произойдёт, если попытаться использовать swap с передачей по значению?
5. Как реализовать swap для пользовательского типа (структуры)?

Раздел Г. Константность и ссылки

1. Что такое const T&? Какие преимущества даёт использование константных ссылок?
2. Почему временные объекты (литералы, результаты выражений) можно передавать только в константную ссылку?
3. Что произойдёт при попытке изменить константную ссылку?
4. В чём разница между const int& и int const&?
5. Что такое ссылка на указатель и указатель на ссылку? Возможно ли последнее?

Раздел Д. Анализ кода и ошибки

1. Что выведет код?

```
void f(int& x) { x = 10; }
int main() { int a = 5; f(a); std::cout << a; return 0; }
```

Найдите ошибку:

cpp

```
int& getValue() {
    int x = 100;
    return x;
}
```

2. Что не так с этим кодом?

```
void print(const std::string& s) {
    s = "new";
```

```
    std::cout << s;
}
```

3. Почему этот код не скомпилируется?

```
void process(int& x) { }
int main() { process(5); return 0; }
```

4. Что такое «висячая ссылка» (dangling reference)? Как её избежать?

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (передача по значению, по ссылке, по указателю; синтаксис ссылок; константные ссылки; возврат ссылки; swap)

Ход выполнения работы:

Код программы с подробными комментариями (каждая функция пояснена)

Блок-схема алгоритма (для функции swap или другой ключевой функции)

Скриншоты выполнения для различных сценариев

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Сравнительная таблица передачи по значению, указателю и ссылке

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали о ссылках, в каких случаях их использовать)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №10)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) - правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, правильное использование ссылок для модификации переменной).

Оценка «хорошо» (4) - правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего уровня (использование ссылок для работы с массивами, возврат нескольких значений через ссылки, правильное применение const для защиты данных).

Оценка «отлично» (5) - правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (полная демонстрация всех трёх способов передачи параметров, обработка ошибок через ссылки, возврат ссылки (с осторожностью), подробные комментарии в коде, блок-схема в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	**Базовый (3)**	**Средний (4)**	**Повышенный (5)**
	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный

	Критерий	**Базовый (3)**	**Средний (4)**	**Повышенный (5)**
	Компиляция и запуск			
.1	Программа компилируется без ошибок	✔	✔	✔
.2	Программа успешно запускается	✔	✔	✔
.3	Отсутствие предупреждений компилятора	—	✔	✔
	Синтаксис ссылок			
.1	Правильное объявление ссылок (int& ref)	✔	✔	✔
.2	Правильная инициализация ссылок	✔	✔	✔
.3	Понимание, что ссылка - это псевдоним	✔	✔	✔
	Передача			

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	параметров по ссылке			
.1	Функция изменяет исходную переменную через ссылку	✔	✔	✔
.2	Функция swap реализована правильно	—	✔	✔
.3	Возврат нескольких значений через ссылки	—	✔	✔
.4	Передача массива по ссылке	—	✔	✔
	Константные ссылки			
.1	Использование const T& для защиты данных	—	✔	✔
.2	Понимание, что временные объекты привязываются только к const&	—	—	✔
.3	Отсутствие попыток изменить const ссылку	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Сравнение способов передачи			
.1	Демонстрация передачи по значению (для контраста)	—	✔	✔
.2	Демонстрация передачи по ссылке	✔	✔	✔
.3	Демонстрация передачи по указателю (для сравнения)	—	—	✔
	Обработка ошибок			
.1	Проверка деления на ноль (где применимо)	—	✔	✔
.2	Проверка границ массива	—	✔	✔
.3	Возврат кода ошибки через ссылку	—	—	✔
	Качество кода			

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.1	Правильные отступы и форматирование	✔	✔	✔
.2	Использование осмысленных имён функций	—	✔	✔
.3	Наличие комментариев (пояснение выбора способа передачи)	—	✔	✔
.4	Сравнение эффективности разных способов в комментариях	—	—	✔
	Отчёт			
.1	Титульный лист, цель, ход работы	✔	✔	✔
.2	Скриншоты выполнения (минимум 3 теста)	✔	✔	✔
.3	Скриншоты для всех трёх уровней	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.4	Таблица тестирования	—	✔	✔
.5	Блок-схема алгоритма	—	✔	✔
.6	Сравнительная таблица способов передачи параметров	—	—	✔
.7	Ответы на контрольные вопросы	—	✔	✔

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) - обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня (№1–15).

Написать функцию, принимающую параметр по ссылке и изменяющую его.

Продемонстрировать вызов функции из main() и изменение исходной переменной.

Понимать разницу между передачей по значению и по ссылке.

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) - дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Использовать ссылки для возврата нескольких значений из функции.

Написать функцию swap или аналогичную для обмена значениями.

Использовать const T& для защиты больших объектов от копирования.

Показать пример передачи массива в функцию с изменением элементов.

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

Блок-схему алгоритма

Повышенный уровень (оценка 5) - требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Демонстрация всех трёх способов передачи:

Функции с передачей по значению, по ссылке, по указателю

Сравнение поведения и эффективности

Возврат ссылки (осторожно):

Функция, возвращающая ссылку на элемент массива

Использование результата в левой части присваивания

Обработка ошибок через ссылки:

Функция возвращает код ошибки через ссылку

Или использование ссылок для возврата дополнительной информации

Сравнительный анализ:

В комментариях объяснить, когда какой способ передачи предпочтительнее

В отчёте дополнительно представить:

Сравнительную таблицу передачи по значению, указателю и ссылке

Примеры опасных ситуаций (висячая ссылка)

7.4. Примеры проверки соответствия

Выполненные задания	Оценка	Проверка критериев
Только базовый вариант №1 (increment)	3	✔ Ссылка в параметре, ✔ изменение переменной, ✘ нет swap, ✘ нет const&
Базовый №1 + средний №1 (min/max массива)	4	✔ Плюс возврат нескольких значений через ссылки, ✔ работа с массивом, ✔ const для входного массива. Но: ✘ нет сравнения с указателями, ✘ нет возврата ссылки
Базовый №1 + средний №1 + повышенный №1 (статистика массива)	5	✔ Все критерии: ✔ демонстрация всех способов передачи, ✔ обработка ошибок, ✔ сравнительный анализ, ✔ блок-схема

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	**Влияние**	**Как исправить**
Неинициализированная ссылка	Ошибка компиляции	Всегда инициализировать ссылку
Попытка изменить const ссылку	Ошибка компиляции	Убрать const или не изменять

Ошибка	Влияние	Как исправить
Возврат ссылки на локальную переменную	Висячая ссылка (UB)	Возвращать ссылку только на статические или переданные объекты
Путаница между & в объявлении и & как адрес	Логическая ошибка	Помнить контекст: int& ref - объявление, &var - взятие адреса
Передача временного объекта в неконстантную ссылку	Ошибка компиляции	Использовать const T& или передавать переменную
swap без ссылок	Переменные не меняются	Использовать int& a, int& b

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Что такое ссылка и как она объявляется

Разницу между передачей по значению и по ссылке

Как реализовать функцию swap

(для оценки 4) - что такое const ссылка и зачем она нужна

(для оценки 5) - в чём отличие ссылки от указателя

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Что произойдёт, если в swap передать swap(a, a)?"	Понимание работы с одинаковыми ссылками
"Почему нельзя изменить привязку ссылки?"	Понимание природы ссылок
"Как передать двумерный массив по ссылке с сохранением размеров?"	Понимание ссылок на массивы
"Что такое std::ref и когда он используется?"	Знание стандартной библиотеки

Лабораторная работа №11

Указатели. Основы работы с памятью

1. Цель работы

Получить базовое понимание адресации памяти и указателей. Научиться объявлять указатели, получать адреса переменных, разыменовывать указатели для доступа к значению, изменять значение переменной через указатель. Освоить арифметику указателей для обхода массивов, понять связь между указателями и массивами, научиться передавать указатели в функции.

2. Теоретическое обоснование

2.1. Понятие памяти и адресации

Компьютерная память представляет собой огромный массив байтов, каждый из которых имеет свой адрес (целое число, идентифицирующее его местоположение). Когда мы объявляем переменную, компилятор выделяет для неё участок памяти и связывает с ней адрес первого байта этого участка.

Визуализация памяти:

```
Адрес:    0x100  0x101  0x102  0x103  0x104  0x105  0x106
        +-------+-------+-------+-------+-------+-------+-------+
        |       |       |       |       |       |       |       |
        +-------+-------+-------+-------+-------+-------+-------+
        ↑
        Переменная int x = 42 занимает 4 байта (0x100-0x103)
        Значение 42 хранится в этих байтах
```

```
int x = 42;                 // объявление переменной
std::cout << x;             // вывод значения: 42
std::cout << &x;            // вывод адреса: например, 0x7fff5fbff58c
```

Оператор & (амперсанд) - взятие адреса:

Возвращает адрес переменной в памяти

Результат имеет тип "указатель на тип переменной"

```
int x = 10;
int* ptr = &x;    // ptr = адрес x (например, 0x7fff)
```

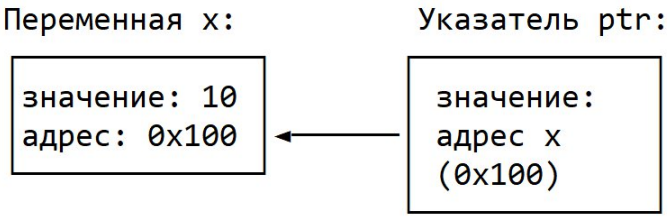
Оператор * (звёздочка) - разыменование указателя:

Позволяет получить доступ к значению, хранящемуся по адресу

Может использоваться для чтения или записи

```
int x = 10;
int* ptr = &x;    // ptr указывает на x
std: cout << *ptr;   // вывод значения: 10
*ptr = 20;          // изменение x через указатель
std: cout << x;      // вывод: 20
```

Визуализация работы указателя:




```
*ptr – разыменование: «возьми значение по адресу, хранящемуся в ptr»
ptr – значение указателя: адрес (0x100)
&x – тоже адрес (0x100)
```

*ptr - разыменование: «возьми значение по адресу, хранящемуся в ptr»

ptr - значение указателя: адрес (0x100)

&x - тоже адрес (0x100)

2.2. Объявление и инициализация указателей

Синтаксис: тип* имя_указателя;

```
int* pInt;         // указатель на int
double* pDouble;   // указатель на double
char* pChar;       // указатель на char
void* pVoid;       // указатель на неизвестный тип (универсальный)
```

Инициализация указателя:

```
int x = 10;
```

```
int* ptr1 = &x;        // ptr1 указывает на x
int*  ptr2  =  nullptr;      // нулевой указатель (C++11) - ни на что не указывает
int* ptr3 = NULL;      // устаревший способ (из C)
int* ptr4 = 0;         // тоже нулевой указатель
```

Важно: Всегда инициализируйте указатели! Неинициализированный указатель содержит мусорный адрес, разыменование которого приводит к неопределённому поведению (обычно к аварийному завершению программы).

```
int* badPtr;       // НЕПРАВИЛЬНО: не инициализирован
*badPtr = 10;       // ОПАСНО! попытка записи по случайному адресу - крах!

int*  goodPtr  =  nullptr;     // ПРАВИЛЬНО: инициализирован нулевым значением
if (goodPtr != nullptr) {  // всегда проверяйте перед разыменованием
    *goodPtr = 10;
}
```

2.3. Операторы & и * - подробно

Оператор	Название	Что делает	Пример	Результат
&	Взятие адреса	Возвращает адрес переменной	&x	Адрес x (например, 0x100)

Оператор	Название	Что делает	Пример	Результат
*	Разыменование	Возвращает значение по адресу	*ptr	Значение, на которое указывает ptr

```cpp
#include <iostream>

int main() {
    int a = 42;          // обычная переменная
    int* ptr = &a;       // указатель, хранит адрес a

    std::cout << "Значение a: " << a << std::endl;          // 42
    std::cout << "Адрес a: " << &a << std::endl;            // 0x7fff...
    std::cout << "Значение ptr: " << ptr << std::endl;      // тот же адрес
    std::cout << "Значение по адресу ptr: " << *ptr << std::endl; // 42

    // Изменение a через ptr
    *ptr = 100;
    std::cout << "Новое значение a: " << a << std::endl;    // 100

    // Адрес самого указателя
    std::cout << "Адрес ptr: " << &ptr << std::endl;        // другой адрес

    return 0;
}
```

2.4. Указатели и константность

```
// 1. Указатель на константу (значение менять нельзя)
const int* ptrToConst = &x;
*ptrToConst = 10;    // ОШИБКА! нельзя изменить значение

// 2. Константный указатель (адрес менять нельзя)
int* const constPtr = &x;
constPtr = &y;       // ОШИБКА! нельзя перенаправить

// 3. Константный указатель на константу (ни адрес, ни значение менять нельзя)
const int* const constPtrToConst = &x;
```

2.5. Арифметика указателей

Указатели поддерживают операции сложения, вычитания и сравнения. При прибавлении к указателю числа n адрес увеличивается на n * sizeof(тип).

```
int arr[5] = {10, 20, 30, 40, 50};
int* ptr = arr;       // ptr указывает на arr[0] (адрес 0x100)

std::cout << "*ptr = " << *ptr << std::endl;       // 10
ptr++;                                  // ptr = ptr + 1
std::cout << "*ptr = " << *ptr << std::endl;       // 20 (адрес 0x104)

ptr += 2;                               // ptr = ptr + 2
std::cout << "*ptr = " << *ptr << std::endl;       // 40 (адрес 0x10C)

ptr--;                                  // ptr = ptr - 1
std::cout << "*ptr = " << *ptr << std::endl;       // 30 (адрес 0x108)
```

```
// Разность указателей (количество элементов между ними)
int* start = arr;
int* end = &arr[4];
int count = end - start;    // 4
```

Визуализация арифметики указателей:

```
Адреса:   0x100   0x104   0x108   0x10C   0x110
        +-------+-------+-------+-------+-------+
arr:    |  10   |  20   |  30   |  40   |  50   |
        +-------+-------+-------+-------+-------+
индекс:    0       1       2       3       4
```

ptr = arr → указывает на 0x100

ptr++ → указывает на 0x104 (сдвиг на 4 байта, т.к. sizeof(int)=4)

ptr + 3 → указывает на 0x110 (на 12 байт вперёд)

ptr2 - ptr1 → количество элементов (разность адресов / sizeof(int))

2.6. Указатели и массивы

В C++ имя массива является константным указателем на первый элемент массива.

```
int arr[5] = {1, 2, 3, 4, 5};

// Эти записи эквивалентны:
arr[0] == *(arr + 0)    // 1
arr[1] == *(arr + 1)    // 2
arr[2] == *(arr + 2)    // 3
```

```
// Обход массива через указатель
int* ptr = arr;
for (int i = 0; i < 5; i++) {
    std::cout << *(ptr + i) << " ";   // 1 2 3 4 5
}

// Альтернативный обход (сдвигая указатель)
for (int* p = arr; p < arr + 5; p++) {
    std::cout << *p << " ";           // 1 2 3 4 5
}
```

Важные соотношения:

cpp

```
arr[i] == *(arr + i)
&arr[i] == arr + i
*(arr + i) == i[arr]   // да, это работает, но НЕ ИСПОЛЬЗУЙТЕ!
```

2.7. Указатели в функциях

2.7.1. Передача указателя в функцию (для изменения переменной)

```
#include <iostream>

// Передача по указателю - функция может изменить исходную переменную
void increment(int* ptr) {
    if (ptr != nullptr) {      // всегда проверяйте!
        (*ptr)++;                  // скобки важны: ++ имеет более высокий приоритет
        // или *ptr += 1;
    }
```

```
}

int main() {
    int a = 5;
    increment(&a);          // передаём адрес a
    std::cout << a;         // 6
    return 0;
}
```

2.7.2. Передача массива в функцию (фактически передаётся указатель)

```
// Три эквивалентных способа передачи массива:
void process1(int arr[], int size) {}     // arr - указатель
void process2(int arr[10], int size) {}   // размер игнорируется
void process3(int* arr, int size) {}      // явно указатель

// Пример: заполнение массива через указатель
void fillArray(int* arr, int size, int value) {
    for (int i = 0; i < size; i++) {
        arr[i] = value;       // или *(arr + i) = value
    }
}
```

Важно: При передаче массива в функцию размер теряется. Его нужно передавать отдельным параметром.

```
void badFunction(int arr[]) {
    int size = sizeof(arr) / sizeof(arr[0]);  // НЕПРАВИЛЬНО!
```

```
    // sizeof(arr) - размер указателя (8 байт на 64-битной системе)
}
```

2.8. Указатель на указатель

Указатель может хранить адрес другого указателя. Это используется для динамических многомерных массивов и изменения указателей в функциях.

```
int x = 10;
int* ptr = &x;      // указатель на int
int** pptr = &ptr;  // указатель на указатель на int

std::cout << **pptr;   // 10 (дважды разыменовываем)
```

Использование для изменения указателя в функции:

```
void allocateMemory(int** ptr, int size) {
    *ptr = new int[size];   // изменяем указатель в вызывающем коде
}

int main() {
    int* arr = nullptr;
    allocateMemory(&arr, 10);   // передаём адрес указателя
    delete[] arr;
    return 0;
}
```

2.9. nullptr (C++11)

nullptr - это специальное значение, обозначающее, что указатель ни на что не указывает.

```
int* ptr = nullptr;        // современный способ
// int* ptr = NULL;        // устаревший способ
// int* ptr = 0;           // тоже работает, но не рекомендуется

if (ptr == nullptr) {
    std::cout << "Указатель нулевой" << std::endl;
}

// Всегда проверяйте указатель перед разыменованием!
if (ptr != nullptr) {
    *ptr = 10;
}
```

2.10. Обход массива с помощью указателя (идиоматический способ)

```
#include <iostream>

int main() {
    int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    // Способ 1: указатель + смещение
    int* ptr = arr;
    for (int i = 0; i < 10; i++) {
        std::cout << *(ptr + i) << " ";
    }
    std::cout << std::endl;
```

```
    // Способ 2: движущийся указатель
    for (int* p = arr; p < arr + 10; p++) {
        std::cout << *p << " ";
    }
    std::cout << std::endl;

    // Способ 3: использование указателей begin/end (аналогично итераторам)
    int* begin = arr;
    int* end = arr + 10;
    for (int* p = begin; p != end; p++) {
        std::cout << *p << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

2.11. Указатели на функции (введение)

Указатель может хранить адрес функции, что позволяет передавать функции как параметры.

```
// Функция, которую будем вызывать через указатель
int add(int a, int b) {
    return a + b;
}

int multiply(int a, int b) {
```

```
    return a * b;
}

// Функция, принимающая указатель на функцию
int compute(int x, int y, int (*operation)(int, int)) {
    return operation(x, y);
}

int main() {
    std::cout << compute(5, 3, add);       // 8
    std::cout << compute(5, 3, multiply);  // 15
    return 0;
}
```

2.12. void* - указатель на неизвестный тип

void* может хранить адрес любого типа данных, но его нельзя разыменовывать без приведения.

```
int x = 10;
double d = 3.14;
void* ptr = &x;      // OK
ptr = &d;            // OK, теперь указывает на double

// std::cout << *ptr;  // ОШИБКА! нельзя разыменовать void*
// Нужно явное приведение:
std::cout << *(static_cast<double*>(ptr));  // 3.14
```

2.13. Типичные ошибки при работе с указателями

Ошибка	Пример	Последствие
Неинициализированный указатель	int* ptr; *ptr = 5;	Неопределённое поведение (крах)
Разыменование nullptr	int* ptr = nullptr; *ptr = 5;	Аварийное завершение
Путаница между * в объявлении и * в выражении	int* ptr = &x; int y = *ptr;	Синтаксис верен, но путаница в понимании
Возврат указателя на локальную переменную	int* f() { int x=5; return &x; }	Висячий указатель (UB)
Забытые скобки при инкременте	*ptr++ (vs (*ptr)++)	Неправильный приоритет
Арифметика указателей с void*	void* ptr; ptr++;	Ошибка компиляции
Сравнение указателей разных типов	int* ptr1; double* ptr2; if (ptr1 == ptr2)	Ошибка компиляции

```
// ОШИБКА: неправильный приоритет операторов
int* ptr = &x;
*ptr++;      // эквивалентно *(ptr++), т.е. сдвиг указателя, а не инкремент значения
```

```
// ПРАВИЛЬНО
(*ptr)++;    // инкремент значения, на которое указывает ptr

// ОШИБКА: возврат указателя на локальную переменную
int* dangerous() {
    int local = 42;
    return &local;   // local уничтожится после выхода!
}
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Вывод адресов переменных

```
//
==========================================================
=
// Программа: вывод адресов переменных разных типов
// Цель: демонстрация взятия адреса через оператор &
//
==========================================================
=

#include <iostream>
#include <iomanip>

int main() {
    // Объявляем переменные разных типов
    int a = 10;
    double b = 3.14159;
    char c = 'A';
```

```
    bool d = true;

    // Вывод значений и адресов
    std::cout << "=== Адреса переменных ===" << std::endl;
    std::cout << "Переменная a (int):    значение = " << a
              << ", адрес = " << &a << std::endl;
    std::cout << "Переменная b (double): значение = " << b
              << ", адрес = " << &b << std::endl;
    std::cout << "Переменная c (char):   значение = " << c
              << ", адрес = " << static_cast<void*>(&c) << std::endl;
    // static_cast<void*> нужен для вывода адреса char (иначе выведет символ)
    std::cout << "Переменная d (bool):   значение = " << d
              << ", адрес = " << &d << std::endl;

    // Демонстрация, что адреса разных переменных отличаются
    std::cout << "\n=== Указатели на переменные ===" << std::endl;

    int* ptrA = &a;          // указатель на int
    double* ptrB = &b;       // указатель на double
    char* ptrC = &c;         // указатель на char
    bool* ptrD = &d;         // указатель на bool

    std::cout << "ptrA указывает на значение: " << *ptrA << std::endl;
    std::cout << "ptrB указывает на значение: " << *ptrB << std::endl;
    std::cout << "ptrC указывает на значение: " << *ptrC << std::endl;
    std::cout << "ptrD указывает на значение: " << *ptrD << std::endl;

    // Изменение переменной через указатель
```

```
    std::cout << "\n=== Изменение через указатель ===" << std::endl;
    std::cout << "Исходное значение a: " << a << std::endl;
    *ptrA = 100;
    std::cout << "После *ptrA = 100: a = " << a << std::endl;

    return 0;
}

// Пример вывода:
// === Адреса переменных ===
// Переменная a (int):    значение = 10, адрес = 0x7fff5fbff58c
// Переменная b (double): значение = 3.14159, адрес = 0x7fff5fbff580
// Переменная c (char):   значение = A, адрес = 0x7fff5fbff57f
// Переменная d (bool):   значение = 1, адрес = 0x7fff5fbff57e
```

3.2. Пример 2: Обход массива с помощью указателя

```
//
==========================================================
=
// Программа: обход массива с использованием указателей
// Цель: демонстрация арифметики указателей и связи с массивами
//
==========================================================
=

#include <iostream>
#include <iomanip>
#include <cstdlib>
```

```cpp
#include <ctime>

int main() {
    const int SIZE = 10;
    int arr[SIZE];

    // Заполнение массива случайными числами
    std::srand(static_cast<unsigned>(std::time(nullptr)));
    for (int i = 0; i < SIZE; i++) {
        arr[i] = std::rand() % 100;
    }

    // ========== СПОСОБ 1: Обычная индексация ==========
    std::cout << "Способ 1 (индексация): ";
    for (int i = 0; i < SIZE; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    // ========== СПОСОБ 2: Указатель + смещение ==========
    std::cout << "Способ 2 (указатель + смещение): ";
    int* ptr = arr;                    // ptr указывает на arr[0]
    for (int i = 0; i < SIZE; i++) {
        std::cout << *(ptr + i) << " ";  // эквивалентно arr[i]
    }
    std::cout << std::endl;

    // ========== СПОСОБ 3: Движущийся указатель ==========
    std::cout << "Способ 3 (движущийся указатель): ";
```

```
    for (int* p = arr; p < arr + SIZE; p++) {
        std::cout << *p << " ";
    }
    std::cout << std::endl;

    // ========== ДЕМОНСТРАЦИЯ АРИФМЕТИКИ УКАЗАТЕЛЕЙ ==========
    std::cout << "\n=== Арифметика указателей ===" << std::endl;

    int* begin = arr;
    int* end = arr + SIZE;

    std::cout << "Адрес arr[0]: " << begin << std::endl;
    std::cout << "Адрес arr[5]: " << begin + 5 << std::endl;
    std::cout << "Разница (end - begin): " << (end - begin) << " элементов" << std::endl;
    std::cout << "Разница в байтах: " << (reinterpret_cast<char*>(end) - reinterpret_cast<char*>(begin)) << " байт" << std::endl;

    // ========== ИЗМЕНЕНИЕ МАССИВА ЧЕРЕЗ УКАЗАТЕЛЬ ==========
    std::cout << "\n=== Изменение массива через указатель ===" << std::endl;

    // Увеличим каждый элемент на 10 через указатель
    for (int* p = arr; p < arr + SIZE; p++) {
        *p += 10;
    }
```

```
    std::cout << "Массив после увеличения на 10: ";
    for (int i = 0; i < SIZE; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

3.3. Пример 3: Функции с указателями

```
//
==========================================================
=
// Программа: передача указателей в функции
// Цель: демонстрация изменения переменных через указатели
//
==========================================================
=

#include <iostream>
#include <cstdlib>
#include <ctime>

// Функция, изменяющая значение переменной через указатель
void setValue(int* ptr, int newValue) {
    if (ptr != nullptr) {       // всегда проверяем!
        *ptr = newValue;
    }
```

```
}

// Функция обмена значений через указатели
void swapPointers(int* a, int* b) {
    if (a == nullptr || b == nullptr) return;

    int temp = *a;
    *a = *b;
    *b = temp;
}

// Функция заполнения массива через указатель
void fillArray(int* arr, int size, int min, int max) {
    if (arr == nullptr) return;

    for (int i = 0; i < size; i++) {
        arr[i] = min + std::rand() % (max - min + 1);
    }
}

// Функция поиска минимума в массиве (возвращает указатель на элемент)
int* findMin(int* arr, int size) {
    if (arr == nullptr || size <= 0) return nullptr;

    int* minPtr = arr;          // указатель на минимальный элемент
    for (int i = 1; i < size; i++) {
        if (arr[i] < *minPtr) {
            minPtr = &arr[i];   // запоминаем адрес нового минимума
```

```
            }
        }
        return minPtr;
    }

    // Функция подсчёта суммы элементов (возвращает сумму, но можно и через указатель)
    int sumArray(int* arr, int size, int* errorCode = nullptr) {
        if (arr == nullptr || size <= 0) {
            if (errorCode != nullptr) *errorCode = -1;
            return 0;
        }

        int sum = 0;
        for (int i = 0; i < size; i++) {
            sum += arr[i];
        }

        if (errorCode != nullptr) *errorCode = 0;
        return sum;
    }

    int main() {
        // Инициализация генератора
        std::srand(static_cast<unsigned>(std::time(nullptr)));

        // ========== 1. Изменение переменной через указатель ==========
        int x = 10;
```

```
    std::cout << "До setValue: x = " << x << std::endl;
    setValue(&x, 42);
    std::cout << "После setValue: x = " << x << std::endl;

    // ========== 2. Обмен значений ==========
    int a = 5, b = 15;
    std::cout << "\nДо swap: a = " << a << ", b = " << b << std::endl;
    swapPointers(&a, &b);
    std::cout << "После swap: a = " << a << ", b = " << b << std::endl;

    // ========== 3. Работа с массивом ==========
    const int SIZE = 10;
    int arr[SIZE];

    fillArray(arr, SIZE, 10, 99);

    std::cout << "\nМассив: ";
    for (int i = 0; i < SIZE; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    // ========== 4. Поиск минимума ==========
    int* minPtr = findMin(arr, SIZE);
    if (minPtr != nullptr) {
        int minIndex = minPtr - arr;  // арифметика указателей
        std::cout << "Минимум: arr[" << minIndex << "] = " << *minPtr <<
std::endl;
    }
```

```
    // ========== 5. Функция с кодом ошибки через указатель ==========
    int error;
    int sum = sumArray(arr, SIZE, &error);
    if (error == 0) {
        std::cout << "Сумма элементов: " << sum << std::endl;
    } else {
        std::cout << "Ошибка при вычислении суммы!" << std::endl;
    }

    return 0;
}
```

3.4. Пример 4: Указатели и константность

```
//
==========
// Программа: демонстрация константных указателей
// Цель: показать различия между разными видами const с указателями
//
==========

#include <iostream>

int main() {
    int x = 10;
```

```
    int y = 20;

    std::cout << "=== 1. Указатель на константу ===" << std::endl;
    const int* ptrToConst = &x;   // указатель на константу
    // *ptrToConst = 30;          // ОШИБКА! нельзя изменить значение
    ptrToConst = &y;              // ОК: можно перенаправить
    std::cout << "*ptrToConst = " << *ptrToConst << std::endl;

    std::cout << "\n=== 2. Константный указатель ===" << std::endl;
    int* const constPtr = &x;     // константный указатель
    *constPtr = 30;               // ОК: значение менять можно
    // constPtr = &y;             // ОШИБКА! нельзя перенаправить
    std::cout << "*constPtr = " << *constPtr << std::endl;

    std::cout << "\n=== 3. Константный указатель на константу ===" <<
std::endl;
    const int* const constPtrToConst = &x;   // константный указатель на
константу
    // *constPtrToConst = 40;     // ОШИБКА! нельзя изменить значение
    // constPtrToConst = &y;      // ОШИБКА! нельзя перенаправить
    std::cout << "*constPtrToConst = " << *constPtrToConst << std::endl;

    std::cout << "\n=== 4. Передача в функцию с const ===" << std::endl;

    // Функция, которая обещает не изменять данные через указатель
    auto printValue = [](const int* ptr) {
        if (ptr != nullptr) {
            std::cout << "Значение: " << *ptr << std::endl;
            // *ptr = 100;  // ОШИБКА! ptr - указатель на константу
```

```
    }
};

printValue(&x);

// Функция, которая может изменить данные
auto modifyValue = [](int* ptr) {
    if (ptr != nullptr) {
        *ptr = 100;
    }
};

modifyValue(&x);
std::cout << "После modifyValue: x = " << x << std::endl;

return 0;
}
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Работа с указателями на отдельные переменные.

№	Задание	Прототип функции
1	Вывести адрес переменной	void printAddress(int x)
2	Изменить значение переменной через указатель	void setToZero(int* ptr)

№	Задание	Прототип функции
3	Увеличить значение на 1 через указатель	void increment(int* ptr)
4	Уменьшить значение на 1 через указатель	void decrement(int* ptr)
5	Присвоить переменной абсолютное значение	void makeAbsolute(int* ptr)
6	Поменять знак переменной на противоположный	void negate(int* ptr)
7	Удвоить значение переменной	void doubleValue(int* ptr)
8	Вывести значение и адрес переменной	void printValueAndAddress(int x)
9	Обменять значения двух переменных через указатели	void swap(int* a, int* b)
10	Найти максимум двух чисел (через указатели)	void max(int* a, int* b, int* result)
11	Найти минимум двух чисел (через указатели)	void min(int* a, int* b, int* result)
1	Проверить, является ли	bool isEven(int* ptr)

№	Задание	Прототип функции
2	число чётным (через указатель)	
13	Присвоить переменной квадрат числа	void square(int* ptr)
14	Присвоить переменной куб числа	void cube(int* ptr)
15	Установить значение в 0, если оно отрицательное	void zeroIfNegative(int* ptr)

Средний уровень (оценка «хорошо»)

Работа с указателями и массивами.

№	Задание	Прототип функции
1	Вывести массив через указатель	void printArray(int* arr, int size)
2	Найти сумму элементов массива через указатель	int sumArray(int* arr, int size)
3	Найти минимальный элемент массива (вернуть указатель)	int* findMin(int* arr, int size)
4	Найти максимальный элемент массива (вернуть указатель)	int* findMax(int* arr, int size)

№	Задание	Прототип функции
5	Заполнить массив значением через указатель	void fillArray(int* arr, int size, int value)
6	Инвертировать массив (реверс) через указатели	void reverseArray(int* arr, int size)
7	Найти сумму положительных и отрицательных элементов	void sumPosNeg(int* arr, int size, int* sumPos, int* sumNeg)
8	Найти количество чётных элементов через указатель	int countEven(int* arr, int size)
9	Умножить каждый элемент массива на число	void multiplyArray(int* arr, int size, int factor)
10	Сдвинуть массив циклически вправо на 1	void shiftRight(int* arr, int size)
11	Сдвинуть массив циклически влево на 1	void shiftLeft(int* arr, int size)
12	Найти разность между максимумом и минимумом	int range(int* arr, int size)
13	Скопировать массив в другой массив	void copyArray(int* src, int* dest, int size)
1	Проверить, отсортирован	bool isSorted(int* arr,

№	Задание	Прототип функции
4	ли массив (по возрастанию)	int size)
5	1 Найти индекс первого вхождения элемента (вернуть указатель)	int* findFirst(int* arr, int size, int value)

Повышенный уровень (оценка «отлично»)

Сложные задачи с указателями, указатели на указатели, указатели на функции.

№	Задание
1	Функция swap через указатели - реализовать обмен значений двух переменных. Продемонстрировать вызов.
2	Обработка массива с использованием указателей - все операции (заполнение, вывод, сумма, min, max) выполняются через указатели (без индексации []).
3	Функция поиска подстроки - реализовать аналог strstr с использованием указателей (без std::string).
4	Функция конкатенации строк - реализовать аналог strcat с использованием указателей (C-style строки).
5	Сортировка массива пузырьком с использованием указателей - передача массива через указатель, сравнение и обмен через указатели.

№	Задание
6	Указатель на указатель для динамического массива - функция, создающая массив и возвращающая его через указатель на указатель.
7	Функция, возвращающая указатель на максимальный элемент - и возврат индекса через указатель.
8	Указатели на функции - создать массив указателей на арифметические функции (+, -, *, /) и вызывать их по выбору пользователя.
9	Обход массива указателем с шагом - вывести элементы массива через один (например, индексы 0,2,4...) с помощью указателя.
10	Функция, изменяющая указатель в вызывающем коде - выделение памяти внутри функции и возврат через int**.
11	Сравнение эффективности - сравнить время доступа к массиву через индексацию и через указатель (для больших массивов).
12	Указатели на элементы структуры - создать структуру Point с полями x, y, написать функцию, изменяющую координаты через указатель.
13	Многоуровневая адресация - создать цепочку: переменная → указатель → указатель на указатель → ... (3 уровня), вывести значения всех уровней.

№		Задание
4	1	Арифметика указателей для void* - демонстрация того, что с void* нельзя выполнять арифметику, и приведение к char* для побайтового доступа.
5	1	Функция, возвращающая указатель на функцию - реализовать функцию, которая в зависимости от переданного параметра возвращает указатель на разные функции.

5. Контрольные вопросы

Раздел А. Основы указателей

1. Что такое указатель? Какую информацию он хранит?
2. Как объявить указатель на int? Как объявить указатель на double?
3. Что делает оператор &? Приведите пример.
4. Что делает оператор *? Приведите пример.
5. Чем отличаются int* ptr, int * ptr и int *ptr?
6. Что такое nullptr? Зачем он нужен?
7. Что произойдёт при разыменовании неинициализированного указателя?

Раздел Б. Арифметика указателей

1. Как работает сложение указателя с целым числом? Что происходит с адресом?
2. Что такое разность указателей? Какой тип имеет результат?
3. Можно ли складывать два указателя? Почему?
4. Как связаны указатели и массивы? Что означает имя массива?
5. Чему равно *(arr + i)? А arr[i]?

Раздел В. Указатели и функции

1. Как передать указатель в функцию? Зачем это нужно?

2. Как изменить значение переменной в функции через указатель?
3. Чем отличается передача указателя от передачи по ссылке?
4. Почему при передаче массива в функцию нужно передавать его размер отдельно?
5. Что такое const указатель? А указатель на const?

Раздел Г. Продвинутые темы

1. Что такое указатель на указатель? Где это может пригодиться?
2. Что такое void*? Какие операции с ним разрешены?
3. Что такое указатель на функцию? Как его объявить и использовать?
4. Что такое «висячий указатель» (dangling pointer)? Как его избежать?

Раздел Д. Анализ кода и ошибки

1. Что выведет код?

```
int x = 5;
int* ptr = &x;
*ptr = 10;
std: cout << x;
```

2. Найдите ошибку:

```
int* ptr;
*ptr = 10;
```

3. Что не так?

```
int arr[5] = {1,2,3,4,5};
int* ptr = arr;
ptr += 10;
*ptr = 100;
```

4. Почему этот код опасен?

cpp

```
int* getPointer() {
```

```
    int x = 42;
    return &x;
}
```

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (понятие указателя, операторы & и *, арифметика указателей, связь с массивами)

Ход выполнения работы:

Код программы с подробными комментариями (каждая операция с указателями пояснена)

Блок-схема алгоритма (для одной из ключевых функций)

Схема памяти (визуализация адресов и значений указателей)

Скриншоты выполнения для различных сценариев

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали об указателях, как они связаны с массивами)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №11)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) - правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, правильное использование операторов & и *, понимание адресации).

Оценка «хорошо» (4) - правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего уровня (арифметика указателей для обхода массива, передача указателей в функции, проверка на nullptr).

Оценка «отлично» (5) - правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (указатели на указатели, указатели на функции, возврат указателя из функции (с осторожностью), арифметика указателей с разными типами, подробные комментарии в коде, схема памяти в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
	Компиляция и запуск			
.1	Программа компилируется без	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	ошибок			
.2	Программа успешно запускается	✔	✔	✔
.3	Отсутствие предупреждений компилятора	—	✔	✔
	Основы указателей			
.1	Правильное объявление указателей (int* ptr)	✔	✔	✔
.2	Использование & для получения адреса	✔	✔	✔
.3	Использование * для разыменования	✔	✔	✔
.4	Инициализация указателей (в т.ч. nullptr)	✔	✔	✔
.5	Проверка на nullptr перед	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	разыменованием			
	Арифметика указателей			
.1	Сложение указателя с целым числом	—	✔	✔
.2	Вычитание указателей	—	✔	✔
.3	Обход массива через движущийся указатель	—	✔	✔
.4	Понимание связи arr[i] и *(arr + i)	—	✔	✔
	Указатели и функции			
.1	Передача указателя в функцию	—	✔	✔
.2	Изменение переменной через	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	указатель в функции			
.3	Возврат указателя из функции	—	—	✔
.4	Указатель на указатель (int**)	—	—	✔
	Указатели и константность			
.1	Понимание const int* vs int* const	—	—	✔
.2	Передача const указателя в функцию	—	—	✔
	Обработка ошибок			
.1	Проверка указателя на nullptr перед использованием	—	✔	✔
.2	Отсутствие выхода за границы при арифметике	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.3	Предотвращение висячих указателей	—	—	✔
	Качество кода			
.1	Правильные отступы и форматирование	✔	✔	✔
.2	Использование осмысленных имён указателей	—	✔	✔
.3	Наличие комментариев (пояснение операций с указателями)	—	✔	✔
.4	Схема памяти в комментариях или отчёте	—	—	✔
	Отчёт			
.1	Титульный лист, цель, ход работы	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	Скриншоты выполнения (минимум 3 теста)	✔	✔	✔
.3	Скриншоты для всех трёх уровней	—	✔	✔
.4	Таблица тестирования	—	✔	✔
.5	Блок-схема алгоритма	—	✔	✔
.6	Схема памяти (адреса, значения указателей)	—	—	✔
.7	Ответы на контрольные вопросы	—	✔	✔

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) - обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня (№1–15).

Объявить указатель и инициализировать его адресом переменной.

Использовать оператор * для разыменования.

Изменить значение переменной через указатель.

Понимать разницу между значением переменной и её адресом.

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) - дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Использовать арифметику указателей для обхода массива.

Передавать указатели в функции для изменения переменных.

Возвращать указатель из функции (например, на найденный элемент).

Всегда проверять указатели на nullptr перед разыменованием.

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

Блок-схему алгоритма

Повышенный уровень (оценка 5) - требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Демонстрация сложных конструкций:

Указатель на указатель (int**)

Указатель на функцию

Константные указатели разных видов

Безопасная работа с указателями:

Предотвращение висячих указателей

Корректная обработка nullptr

Арифметика указателей:

Демонстрация связи arr[i] и *(arr + i)

Вычисление количества элементов через разность указателей

Визуализация в отчёте:

Схема памяти с адресами и значениями указателей

В отчёте дополнительно представить:

Схему памяти (адреса переменных и указателей)

Сравнение указателей и ссылок

7.4. Примеры проверки соответствия

Выполненные задания	Оценка	Проверка критериев
Только базовый вариант №1 (вывод адреса)	3	✔ &, ✔ *, ✔ изменение через указатель. Но: ✘ нет работы с массивом, ✘ нет арифметики указателей
Базовый №1 + средний №2 (сумма массива через указатель)	4	✔ Плюс арифметика указателей, ✔ обход массива, ✔ передача указателя в функцию. Но: ✘ нет указателя на указатель, ✘ нет возврата указателя
Базовый №1 + средний №2 + повышенный №1 (swap)	5	✔ Все критерии: ✔ арифметика, ✔ передача в функции, ✔ проверка nullptr, ✔ схема памяти в отчёте

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	**Влияние**	**Как исправить**
Неинициализированны	Неопредел	Всегда

Ошибка	Влияние	Как исправить
й указатель	ённое поведение	инициализировать (nullptr или адрес)
Разыменование nullptr	Аварийное завершение	Проверять if (ptr != nullptr)
Выход за границы массива при арифметике	Неопределённое поведение	Контролировать диапазон
Возврат указателя на локальную переменную	Висячий указатель	Возвращать указатель на статическую или динамическую память
Путаница между * в объявлении и * в выражении	Синтаксические ошибки	Помнить контекст
Забытые скобки при (*ptr)++	Неправильный приоритет	Использовать скобки
sizeof на указателе в функции	Неверный размер	Передавать размер отдельным параметром

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Что такое указатель и как его объявить

Разницу между & и *

Как обойти массив через указатель

(для оценки 4) - что такое арифметика указателей

(для оценки 5) - разницу между const int* и int* const

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Что произойдёт при int* ptr; *ptr = 5;?"	Понимание опасности неинициализированных указателей
"Чем отличается int* p = &x от int* p; p = &x?"	Понимание инициализации
"Как получить количество элементов между двумя указателями?"	Понимание разности указателей
"Почему нельзя складывать два указателя?"	Понимание смысла арифметики

Лабораторная работа №12

Динамическая память. Массивы переменного размера

1. Цель работы

Научиться управлять динамической памятью в C++ с помощью операторов new и delete. Освоить создание массивов переменного размера (размер определяется во время выполнения), заполнение и обработку динамических массивов. Научиться предотвращать утечки памяти, правильно

освобождать динамически выделенную память, понимать разницу между стеком (автоматическая память) и кучей (динамическая память).

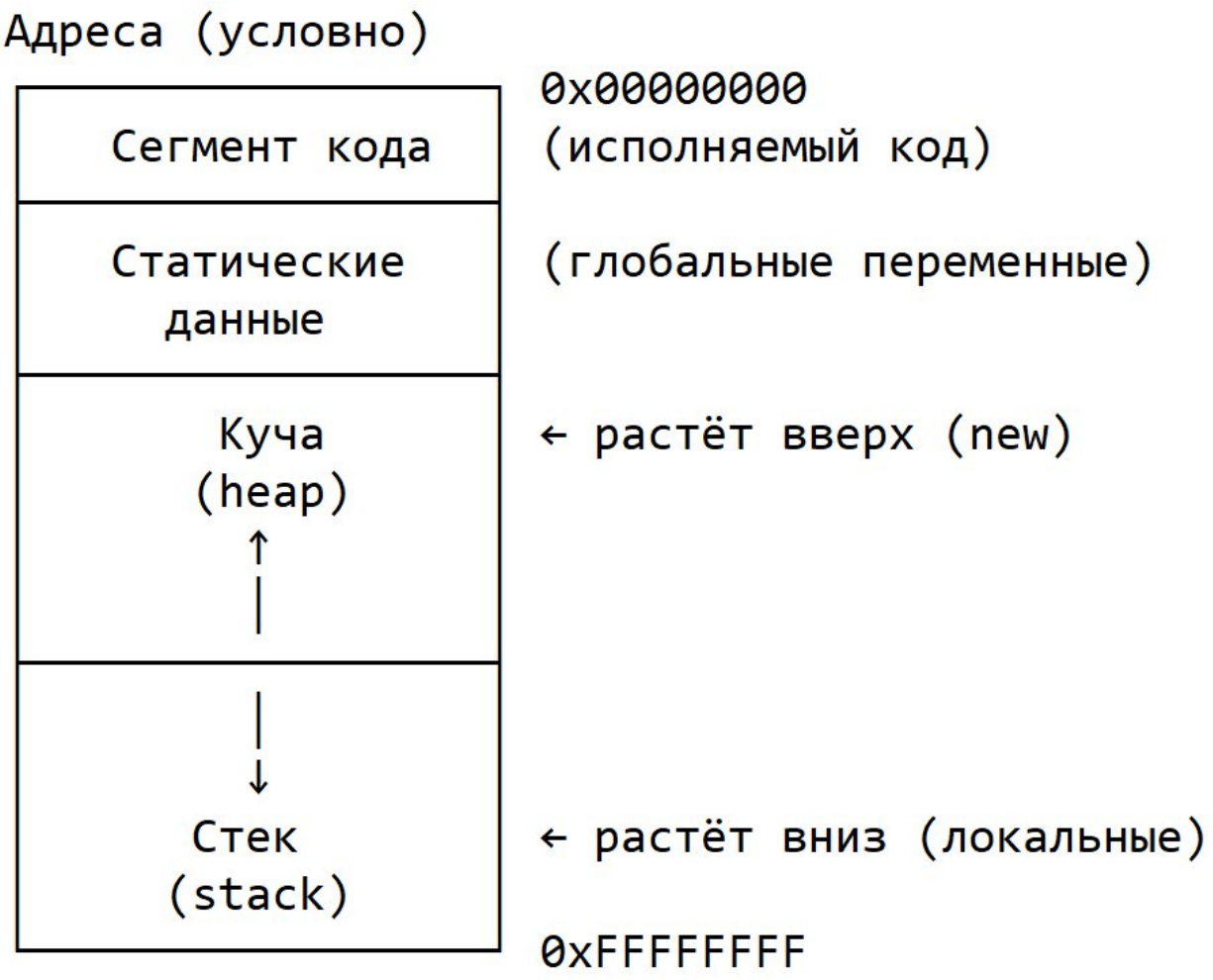
2. Теоретическое обоснование

2.1. Виды памяти в C++

В C++ программы используют несколько видов памяти:

Тип памяти	Выделение	Освобождение	Размер	Скорость	Пример
Стек (stack)	Автоматическое при входе в блок	Автоматическое при выходе	Фиксированный (известен на этапе компиляции)	Очень быстрая	int x = 5;
Куча (heap)	Ручное (new)	Ручное (delete)	Переменный (известен во время выполнения)	Медленнее	int* ptr = new int;
Статическая	При запуске программы	При завершении	Фиксированный	Быстрая	static int x;

Визуализация памяти:



2.2. Выделение памяти: оператор new

Синтаксис для одного объекта:

```
тип* указатель = new тип;          // выделение памяти
тип* указатель = new тип(значение); // выделение + инициализация
```

Примеры:

cpp

```
int* pInt = new int;          // выделяем память для одного int
*pInt = 42;                   // записываем значение

int* pIntInit = new int(100);  // выделяем и инициализируем значением
```

```
double* pDouble = new double;  // выделяем память для double
*pDouble = 3.14159;

int* pArray = new int[10];     // выделяем память для массива из 10 int
```

Что происходит при new:

1. Оператор new запрашивает у операционной системы блок памяти нужного размера
2. Если память успешно выделена, возвращается адрес первого байта блока
3. Если память не может быть выделена, выбрасывается исключение std::bad_alloc

2.3. Освобождение памяти: оператор delete

Синтаксис:

```
delete указатель;       // для одного объекта
delete[] указатель;     // для массива объектов
```

Примеры:

```
int* pInt = new int(42);
delete pInt;            // освобождаем память
pInt = nullptr;         // хорошая практика: обнуляем указатель

int* pArray = new int[100];
delete[] pArray;        // освобождаем массив
pArray = nullptr;
```

Важно:

1. Каждому new должен соответствовать delete

2. Каждому new[] должен соответствовать delete[]
3. Удаление nullptr безопасно (ничего не происходит)
4. После delete указатель становится "висячим" - лучше присвоить nullptr

```
// ПРАВИЛЬНО
int* p = new int;
delete p;          // new → delete

int* arr = new int[10];
delete[] arr;      // new[] → delete[]

// НЕПРАВИЛЬНО (неопределённое поведение!)
int* p = new int;
delete[] p;        // ОШИБКА: new с delete[]

int* arr = new int[10];
delete arr;        // ОШИБКА: new[] с delete
```

2.4. Создание массива переменного размера

Главное преимущество динамической памяти: размер массива может быть определён во время выполнения программы.

```
#include <iostream>

int main() {
    int size;
    std::cout << "Введите размер массива: ";
    std::cin >> size;
```

```
    // Создаём массив, размер которого вводит пользователь
    int* arr = new int[size];   // размер известен только сейчас!

    // Заполняем массив
    for (int i = 0; i < size; i++) {
        arr[i] = i * 10;
    }

    // Выводим массив
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    // ОБЯЗАТЕЛЬНО освобождаем память!
    delete[] arr;
    arr = nullptr;

    return 0;
}
```

Статический массив (размер фиксирован):

```
int arr[10];          // размер известен на этапе компиляции
int n; std::cin >> n;
int arr2[n];          // НЕПРАВИЛЬНО! VLA не в стандартном C++
```

Динамический массив (размер переменный):

```
int n; std::cin >> n;
```

```
int* arr = new int[n]; // ПРАВИЛЬНО! размер может быть любым
delete[] arr;
```

2.5. Обработка динамического массива

С динамическим массивом можно работать так же, как со статическим - через индексы или через указатели.

```
#include <iostream>

int main() {
    int size;
    std::cout << "Введите размер: ";
    std::cin >> size;

    // Создание
    int* arr = new int[size];

    // Заполнение (через индексы)
    for (int i = 0; i < size; i++) {
        arr[i] = i + 1;
    }

    // Сумма (через указатель)
    int sum = 0;
    for (int* p = arr; p < arr + size; p++) {
        sum += *p;
    }
    std::cout << "Сумма: " << sum << std::endl;
```

```
    // Поиск максимума
    int max = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] > max) max = arr[i];
    }
    std::cout << "Максимум: " << max << std::endl;

    // Освобождение
    delete[] arr;

    return 0;
}
```

2.6. Динамическая память для двумерных массивов

Существует несколько способов создания динамических двумерных массивов. Рассмотрим самый распространённый: массив указателей на строки.

Способ 1: Массив указателей на строки (непрерывная память не гарантирована)

```
int rows = 3, cols = 4;

// 1. Создаём массив указателей на строки
int** matrix = new int*[rows];

// 2. Для каждой строки создаём массив столбцов
for (int i = 0; i < rows; i++) {
    matrix[i] = new int[cols];
}
```

```cpp
// 3. Заполнение
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        matrix[i][j] = i * cols + j;
    }
}

// 4. Вывод
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        std::cout << matrix[i][j] << " ";
    }
    std::cout << std::endl;
}

// 5. Освобождение (в обратном порядке!)
for (int i = 0; i < rows; i++) {
    delete[] matrix[i];
}
delete[] matrix;
```

Способ 2: Одномерный массив с пересчётом индекса (непрерывная память)

cpp

```cpp
int rows = 3, cols = 4;

// Выделяем один непрерывный блок
int* matrix = new int[rows * cols];
```

```
// Доступ: matrix[i * cols + j]
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        matrix[i * cols + j] = i * cols + j;
    }
}

// Освобождение
delete[] matrix;
```

2.7. Утечки памяти (memory leak)

Утечка памяти - это ситуация, когда динамически выделенная память не освобождается. Программа продолжает занимать память, но уже не может её использовать.

```
void leakExample() {
    int* ptr = new int(42);
    // забыли delete ptr - утечка памяти!
    // память будет занята до завершения программы
}

void noLeak() {
    int* ptr = new int(42);
    delete ptr;      // правильно освобождено
}
```

Как обнаружить утечки:

1. Использовать специальные инструменты (Valgrind, Dr. Memory, Visual Studio Debug Heap)

2. Визуальный осмотр кода: каждому new должен соответствовать delete

Типичные ситуации с утечками:

```
// 1. Потеря указателя
int* ptr = new int(10);
ptr = new int(20);   // первый адрес потерян - утечка!
delete ptr;          // освобождается только последний

// 2. Исключение между new и delete
int* ptr = new int[1000];
// если здесь произойдёт исключение, delete не выполнится
delete[] ptr;

// 3. Выход из функции без delete
void func() {
    int* ptr = new int(10);
    return;          // забыли delete
}
```

2.8. Динамическая память в функциях

2.8.1. Функция, создающая массив (возвращает указатель)

```
int* createArray(int size) {
    int* arr = new int[size];
    for (int i = 0; i < size; i++) {
        arr[i] = 0;
    }
```

```
    return arr;   // возвращаем указатель на динамическую память
}

int main() {
    int size = 10;
    int* arr = createArray(size);
    // используем arr
    delete[] arr;   // ответственность на вызывающем!
    return 0;
}
```

Важно: Документируйте, кто отвечает за освобождение памяти!

2.8.2. Функция, изменяющая указатель (указатель на указатель)

```
void createArray(int** ptr, int size) {
    *ptr = new int[size];
    for (int i = 0; i < size; i++) {
        (*ptr)[i] = 0;
    }
}

int main() {
    int* arr = nullptr;
    createArray(&arr, 10);   // передаём адрес указателя
    delete[] arr;
    return 0;
}
```

2.8.3. Функция, изменяющая размер массива (перевыделение)

```
void resizeArray(int*& arr, int oldSize, int newSize) {
    int* newArr = new int[newSize];

    // Копируем старые данные
    int copySize = (oldSize < newSize) ? oldSize : newSize;
    for (int i = 0; i < copySize; i++) {
        newArr[i] = arr[i];
    }

    delete[] arr;      // освобождаем старый массив
    arr = newArr;      // перенаправляем указатель
}
```

2.9. new и delete для объектов (введение в ООП)

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

int main() {
    // Одиночный объект
    Point* p = new Point(10, 20);
    std::cout << p->x << ", " << p->y << std::endl;
    delete p;

    // Массив объектов
    Point* arr = new Point[5]{ {1,1}, {2,2}, {3,3}, {4,4}, {5,5} };
```

```
    delete[] arr;

    return 0;
}
```

2.10. std::vector - современная альтернатива динамическим массивам

C++ предоставляет контейнер std::vector, который автоматически управляет динамической памятью.

```
#include <vector>

int main() {
    int size;
    std::cin >> size;

    std::vector<int> arr(size);   // автоматическое выделение памяти
    arr[0] = 10;
    arr.push_back(20);            // автоматическое расширение

    // Не нужно вызывать delete!
    // Память освободится автоматически при выходе из области видимости

    return 0;
}
```

Сравнение new[] / delete[] и std::vector:

Характеристика	new[] / delete[]	std::vector
Управление памятью	Ручное	Автоматическое
Риск утечек	Высокий	Низкий
Изменение размера	Требует перевыделения вручную	.push_back(), .resize()
Проверка границ	Нет (неопределённое поведение)	.at() с исключением
Совместимость с алгоритмами	Указатели	Итераторы
Рекомендация	Только в учебных целях/низкоуровневый код	В реальных проектах

2.11. Типичные ошибки при работе с динамической памятью

Ошибка	Пример	Последствие
Забыт delete	int* p = new int; без delete	Утечка памяти
Несоответствие new и delete	new int[] + delete	Неопределённое поведение

Ошибка	Пример	Последствие
Двойное удаление	delete p; delete p;	Неопределённое поведение
Использование после удаления	delete p; *p = 10;	Неопределённое поведение
Утечка при исключении	new в конструкторе до delete	Утечка памяти
sizeof на динамическом массиве	sizeof(arr) где arr = new int[10]	8 (размер указателя), не 40

```
// ОШИБКА: двойное удаление
int* p = new int(10);
delete p;
delete p;   // НЕПРАВИЛЬНО! неопределённое поведение

// ОШИБКА: использование после удаления
int* p = new int(10);
delete p;
*p = 20;    // НЕПРАВИЛЬНО! доступ к освобождённой памяти

// ПРАВИЛЬНО: после delete присваиваем nullptr
delete p;
p = nullptr;  // теперь случайное использование вызовет ошибку (легче найти)
// if (p) *p = 20;  // безопасно, не выполнится
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Создание и обработка одномерного динамического массива

```
//
==========================================================
=
// Программа: создание динамического массива, заполнение, обработка
// Цель: демонстрация операторов new[] и delete[]
//
==========================================================
=

#include <iostream>
#include <cstdlib>
#include <ctime>

int main() {
    int size;

    // Ввод размера массива (определяется во время выполнения)
    std::cout << "Введите размер массива: ";
    std::cin >> size;

    // Проверка корректности размера
    if (size <= 0) {
        std::cout << "Ошибка: размер должен быть положительным!" <<
std::endl;
        return 1;
```

```
    }

    // ========== ВЫДЕЛЕНИЕ ДИНАМИЧЕСКОЙ ПАМЯТИ ==========
    int* arr = new int[size];   // размер может быть переменной!

    // ========== ЗАПОЛНЕНИЕ СЛУЧАЙНЫМИ ЧИСЛАМИ ==========
    std::srand(static_cast<unsigned>(std::time(nullptr)));
    for (int i = 0; i < size; i++) {
        arr[i] = std::rand() % 100;   // числа от 0 до 99
    }

    // ========== ВЫВОД МАССИВА ==========
    std::cout << "\nМассив: ";
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    // ========== ВЫЧИСЛЕНИЕ СУММЫ ==========
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += arr[i];
    }
    std::cout << "Сумма элементов: " << sum << std::endl;

    // ========== ПОИСК МИНИМУМА И МАКСИМУМА ==========
```

```
    int minVal = arr[0];
    int maxVal = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] < minVal) minVal = arr[i];
        if (arr[i] > maxVal) maxVal = arr[i];
    }
    std::cout << "Минимум: " << minVal << std::endl;
    std::cout << "Максимум: " << maxVal << std::endl;

    // ========== ОСВОБОЖДЕНИЕ ПАМЯТИ ==========
    delete[] arr;      // ОБЯЗАТЕЛЬНО! и не забыть []
    arr = nullptr;     // хорошая практика: обнуляем указатель

    return 0;
}

// Пример работы:
// Введите размер массива: 10
// Массив: 45 67 23 89 12 34 78 56 91 5
// Сумма элементов: 500
// Минимум: 5
// Максимум: 91
```

3.2. Пример 2: Функции для работы с динамическим массивом

```
//
==========================================================
=
// Программа: функции для создания, заполнения, вывода и удаления
```

```
// Цель: демонстрация передачи динамических массивов в функции
//
============================================================
=

#include <iostream>
#include <cstdlib>
#include <ctime>

// Создание динамического массива
int* createArray(int size) {
    if (size <= 0) return nullptr;

    int* arr = new int[size];
    return arr;
}

// Заполнение массива случайными числами
void fillRandom(int* arr, int size, int min, int max) {
    if (arr == nullptr) return;

    for (int i = 0; i < size; i++) {
        arr[i] = min + std::rand() % (max - min + 1);
    }
}

// Вывод массива на экран
void printArray(const int* arr, int size) {
    if (arr == nullptr) {
```

```
        std::cout << "Массив не существует!" << std::endl;
        return;
    }

    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

// Поиск минимума и максимума (через ссылки для возврата)
void findMinMax(const int* arr, int size, int& minVal, int& maxVal) {
    if (arr == nullptr || size <= 0) {
        minVal = maxVal = 0;
        return;
    }

    minVal = arr[0];
    maxVal = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] < minVal) minVal = arr[i];
        if (arr[i] > maxVal) maxVal = arr[i];
    }
}

// Удаление массива и обнуление указателя
void deleteArray(int*& arr) {
    delete[] arr;
    arr = nullptr;   // обнуляем, чтобы избежать висячего указателя
```

```
}

int main() {
    std::srand(static_cast<unsigned>(std::time(nullptr)));

    int size;
    std::cout << "Введите размер массива: ";
    std::cin >> size;

    // Создание
    int* numbers = createArray(size);
    if (numbers == nullptr) {
        std::cout << "Ошибка создания массива!" << std::endl;
        return 1;
    }

    // Заполнение
    fillRandom(numbers, size, 10, 99);

    // Вывод
    std::cout << "Массив: ";
    printArray(numbers, size);

    // Поиск min/max
    int minVal, maxVal;
    findMinMax(numbers, size, minVal, maxVal);
    std::cout << "Минимум: " << minVal << ", Максимум: " << maxVal <<
std::endl;
```

```
    // Освобождение памяти
    deleteArray(numbers);

    // Проверка, что указатель обнулён
    if (numbers == nullptr) {
        std::cout << "Массив успешно удалён, указатель обнулён." <<
std::endl;
    }

    return 0;
}
```

3.3. Пример 3: Двумерный динамический массив

```
//
==========================================================
=
// Программа: создание и обработка двумерного динамического массива
// Цель: демонстрация new[] для строк и столбцов
//
==========================================================
=

#include <iostream>
#include <cstdlib>
#include <ctime>
#include <iomanip>

// Создание матрицы (массив указателей на строки)
```

```
int** createMatrix(int rows, int cols) {
    if (rows <= 0 || cols <= 0) return nullptr;

    // 1. Выделяем память под массив указателей на строки
    int** matrix = new int*[rows];

    // 2. Для каждой строки выделяем память под столбцы
    for (int i = 0; i < rows; i++) {
        matrix[i] = new int[cols];
    }

    return matrix;
}

// Заполнение матрицы случайными числами
void fillRandom(int** matrix, int rows, int cols, int min, int max) {
    if (matrix == nullptr) return;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            matrix[i][j] = min + std::rand() % (max - min + 1);
        }
    }
}

// Вывод матрицы
void printMatrix(int** matrix, int rows, int cols) {
    if (matrix == nullptr) return;
```

```
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            std::cout << std::setw(4) << matrix[i][j];
        }
        std::cout << std::endl;
    }
}

// Сумма элементов на главной диагонали (для квадратной матрицы)
int sumMainDiagonal(int** matrix, int size) {
    if (matrix == nullptr) return 0;

    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += matrix[i][i];
    }
    return sum;
}

// Удаление матрицы
void deleteMatrix(int**& matrix, int rows) {
    if (matrix == nullptr) return;

    // Сначала удаляем каждую строку
    for (int i = 0; i < rows; i++) {
        delete[] matrix[i];
    }
    // Затем удаляем массив указателей
    delete[] matrix;
```

```
    matrix = nullptr;
}

int main() {
    std::srand(static_cast<unsigned>(std::time(nullptr)));

    int rows, cols;
    std::cout << "Введите количество строк и столбцов: ";
    std::cin >> rows >> cols;

    if (rows <= 0 || cols <= 0) {
        std::cout << "Неверный размер!" << std::endl;
        return 1;
    }

    // Создание
    int** matrix = createMatrix(rows, cols);

    // Заполнение
    fillRandom(matrix, rows, cols, 1, 50);

    // Вывод
    std::cout << "\nСгенерированная матрица:" << std::endl;
    printMatrix(matrix, rows, cols);

    // Сумма по строкам
    std::cout << "\nСуммы по строкам:" << std::endl;
    for (int i = 0; i < rows; i++) {
        int rowSum = 0;
```

```
        for (int j = 0; j < cols; j++) {
            rowSum += matrix[i][j];
        }
        std::cout << "Строка " << i << ": " << rowSum << std::endl;
    }

    // Если матрица квадратная - сумма диагонали
    if (rows == cols) {
        int diagSum = sumMainDiagonal(matrix, rows);
        std::cout << "\nСумма главной диагонали: " << diagSum << std::endl;
    }

    // Освобождение памяти
    deleteMatrix(matrix, rows);

    return 0;
}
```

3.4. Пример 4: Перевыделение памяти (изменение размера массива)

```
//
==========================================================
=
// Программа: динамическое изменение размера массива
// Цель: демонстрация выделения нового блока и копирования
//
==========================================================
=
```

```cpp
#include <iostream>

// Функция расширения массива
void resizeArray(int*& arr, int oldSize, int newSize) {
    if (newSize <= 0) {
        delete[] arr;
        arr = nullptr;
        return;
    }

    // Выделяем новый блок памяти
    int* newArr = new int[newSize];

    // Копируем старые данные (не больше, чем вмещается)
    int copySize = (oldSize < newSize) ? oldSize : newSize;
    for (int i = 0; i < copySize; i++) {
        newArr[i] = arr[i];
    }

    // Заполняем новые элементы нулями (если расширение)
    for (int i = copySize; i < newSize; i++) {
        newArr[i] = 0;
    }

    // Освобождаем старый массив
    delete[] arr;

    // Перенаправляем указатель на новый массив
    arr = newArr;
```

```
}

// Функция добавления элемента в конец массива
void push_back(int*& arr, int& size, int value) {
    resizeArray(arr, size, size + 1);
    arr[size] = value;   // после resize размер уже увеличен
    size++;
}

// Функция удаления последнего элемента
void pop_back(int*& arr, int& size) {
    if (size <= 0) return;
    resizeArray(arr, size, size - 1);
    size--;
}

// Вывод массива
void printArray(const int* arr, int size) {
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

int main() {
    int size;
    std::cout << "Введите начальный размер: ";
    std::cin >> size;
```

```
int* arr = nullptr;

// Создаём начальный массив
resizeArray(arr, 0, size);
for (int i = 0; i < size; i++) {
    arr[i] = i + 1;
}

std::cout << "Начальный массив: ";
printArray(arr, size);

// Добавляем элементы
push_back(arr, size, 100);
push_back(arr, size, 200);
push_back(arr, size, 300);
std::cout << "После добавления элементов: ";
printArray(arr, size);

// Удаляем элементы
pop_back(arr, size);
pop_back(arr, size);
std::cout << "После удаления элементов: ";
printArray(arr, size);

// Увеличиваем массив
int newSize;
std::cout << "Введите новый размер: ";
std::cin >> newSize;
resizeArray(arr, size, newSize);
```

```
    std::cout << "После изменения размера на " << newSize << ": ";
    printArray(arr, newSize);

    // Освобождение памяти
    delete[] arr;
    arr = nullptr;

    return 0;
}
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Создать одномерный динамический массив, выполнить простую обработку.

№	Задание	Дополнительно
1	Найти сумму элементов	Размер вводится с клавиатуры
2	Найти среднее арифметическое	Заполнение случайными числами
3	Найти минимальный элемент	Заполнение с клавиатуры
4	Найти максимальный элемент	Заполнение случайными числами
5	Найти количество чётных элементов	Размер и элементы вводятся

№	Задание	Дополнительно
6	Найти количество нечётных элементов	Заполнение случайными числами
7	Найти количество положительных элементов	Диапазон -50..50
8	Найти количество отрицательных элементов	Диапазон -100..100
9	Вывести массив в обратном порядке	Не меняя исходный
10	Найти произведение элементов (с проверкой на ноль)	Заполнение случайными числами
11	Найти сумму элементов на чётных позициях	Индексы 0,2,4,...
12	Найти сумму элементов на нечётных позициях	Индексы 1,3,5,...
13	Найти количество элементов больше заданного числа	Порог вводится
14	Найти количество элементов меньше заданного числа	Порог вводится

№	Задание	Дополнительно
15	1 Найти индекс первого нулевого элемента (или -1)	Заполнение случайными числами

Средний уровень (оценка «хорошо»)

Создать динамический массив и выполнить комплексную обработку.

	Задание	Требования
	Вычислить сумму, min, max, среднее	Все результаты вывести
	Отсортировать массив (пузырьком)	Вывести до и после сортировки
	Найти второй максимум и второй минимум	Без полной сортировки
	Удалить все чётные элементы (создать новый массив)	Динамическое изменение размера
	Удалить все нулевые элементы	Динамическое изменение размера
	Добавить элемент в начало массива	Сдвиг элементов вправо
	Добавить элемент в конец массива	Простое расширение

	Задание	Требования
	Вставить элемент по индексу	Сдвиг вправо, расширение
	Удалить элемент по индексу	Сдвиг влево, уменьшение
0	Найти и удалить все вхождения заданного числа	Динамическое изменение размера
1	Развернуть массив (реверс)	На месте, без второго массива
2	Циклический сдвиг вправо на K позиций	K вводится
3	Циклический сдвиг влево на K позиций	K вводится
4	Найти количество различных элементов	Без использования дополнительных структур
5	Проверить, является ли массив палиндромом	Сравнение с реверсированным

Повышенный уровень (оценка «отлично»)

Динамические двумерные массивы, перевыделение памяти.

№	Задание
1	Двумерный динамический массив: создать матрицу, найти сумму по строкам и столбцам, вывести результат.
2	Транспонирование матрицы: создать динамическую матрицу, транспонировать, вывести результат.
3	Умножение матриц: создать две динамические матрицы (размеры вводятся), вычислить произведение.
4	Динамическая матрица (непрерывная память): выделить один блок new int[rows*cols], реализовать доступ через arr[i*cols+j].
5	Динамический массив с изменяемым размером: реализовать аналог std::vector с методами push_back, pop_back, insert, erase.
6	Массив строк: создать динамический массив строк (C-style: char**), заполнить, отсортировать по алфавиту.
7	Динамический массив функций: массив указателей на функции, выполнение по выбору пользователя.
8	Сравнение эффективности: сравнить время доступа к статическому и динамическому массиву (большой размер, много итераций).
9	Многоуровневая динамическая структура: массив указателей на массивы указателей (трёхмерный массив).

№	Задание
10	Работа с файлом: чтение количества элементов из файла, создание динамического массива, запись обратно.
11	Динамическая структура данных: реализовать простой односвязный список (Node - динамический объект).
12	Реализация std::vector (упрощённая): шаблонный класс с capacity, size, динамическим перевыделением при расширении.
13	Массив с автоматическим расширением: при добавлении элемента, если size == capacity, увеличить capacity вдвое.
14	Динамические структуры: создать массив структур Student (динамически), выполнить поиск, сортировку.
15	Игра «Жизнь» (Conway's Game of Life): динамическое поле, реализация правил, пошаговый вывод.

5. Контрольные вопросы

Раздел А. Основы динамической памяти

1. Чем отличается стек от кучи (heap)? Каковы преимущества и недостатки каждого?
2. Что делает оператор new? Что возвращает new?
3. Что делает оператор delete? Что произойдёт, если забыть вызвать delete?
4. В чём разница между delete и delete[]? Когда какой использовать?
5. Что такое «утечка памяти» (memory leak)? Как её обнаружить?

Раздел Б. Динамические массивы

1. Как создать динамический массив, размер которого вводит пользователь?
2. Как правильно освободить динамический массив?
3. Почему статический массив нельзя создать с переменным размером?
4. Как динамический массив передать в функцию?
5. Как вернуть динамический массив из функции? Кто отвечает за его удаление?

Раздел В. Двумерные динамические массивы

1. Как создать динамическую матрицу (двумерный массив)?
2. Как правильно освободить память динамической матрицы? Почему важен порядок?
3. В чём разница между двумя способами выделения матрицы: int** с циклами и int* с пересчётом индекса?
4. Что такое «висячий указатель» (dangling pointer)? Как его избежать?
5. Почему после delete указатель следует обнулять (ptr = nullptr)?

Раздел Г. Ошибки и лучшие практики

1. Что произойдёт при двойном удалении (delete ptr; delete ptr;)?
2. Что произойдёт при использовании delete вместо delete[] для массива?
3. Что такое «утечка памяти при исключении»? Как её предотвратить?
4. Почему std::vector предпочтительнее ручного управления памятью?
5. Как проверить, что память выделена успешно?

Раздел Д. Анализ кода

1. Что не так?

```
int* arr = new int[10];
arr[10] = 5;
delete arr;
```

2. Найдите утечку памяти:

cpp

```
void func() {
    int* p = new int(5);
    if (p == nullptr) return;
    // ... код
}
```

3. Что выведет код?

```
int* p = new int(10);
int* q = p;
delete p;
*q = 20;
std::cout << *q;
```

4. Почему этот код опасен?

cpp

```
int* getArray() {
    int arr[10] = {1,2,3,4,5,6,7,8,9,10};
    return arr;
}
```

5. Что такое RAII? Как он связан с динамической памятью?

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (стек и куча, операторы new/delete, new[]/delete[], утечки памяти, двумерные динамические массивы)

Ход выполнения работы:

Код программы с подробными комментариями (каждый new и delete пояснён)

Блок-схема алгоритма (для одной из ключевых функций)

Схема памяти (визуализация выделения и освобождения)

Скриншоты выполнения для различных сценариев (разные размеры массива)

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Сравнительная таблица статического и динамического массива

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали о динамической памяти, как предотвращать утечки)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №12)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) - правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, правильное использование new[] и delete[], отсутствие утечек памяти).

Оценка «хорошо» (4) - правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего

уровня (изменение размера массива, вставка/удаление элементов, сортировка, корректное освобождение памяти).

Оценка «отлично» (5) - правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (двумерные динамические массивы, перевыделение памяти, реализация аналога std::vector, полная защита от утечек памяти, подробные комментарии в коде, схема памяти в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
	Компиляция и запуск			
.1	Программа компилируется без ошибок	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	Программа успешно запускается	✔	✔	✔
.3	Отсутствие предупреждений компилятора	—	✔	✔
	Выделение памяти			
.1	Правильное использование new[] для массива	✔	✔	✔
.2	Размер массива определяется во время выполнения	✔	✔	✔
.3	Проверка на успешность выделения (или корректная обработка)	—	✔	✔
	Освобождение памяти			
.1	Правильное использование delete[]	✔	✔	✔
.2	Отсутствие утечек памяти	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.3	Отсутствие двойного удаления	✔	✔	✔
.4	Обнуление указателя после delete	—	✔	✔
	Обработка массива			
.1	Корректное заполнение массива	✔	✔	✔
.2	Корректный вывод массива	✔	✔	✔
.3	Выполнение требуемой операции (сумма, min, max и т.д.)	✔	✔	✔
	Продвинутые операции			
.1	Изменение размера массива (перевыделение)	—	✔	✔
.2	Вставка/удаление элементов	—	✔	✔
	Сортировка массива	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.3				
.4	Двумерный динамический массив	—	—	✔
.5	Реализация аналога std::vector	—	—	✔
	Обработка ошибок			
.1	Проверка корректности размера (size > 0)	—	✔	✔
.2	Проверка индексов при вставке/удалении	—	✔	✔
.3	Обработка nullptr после new (при failure)	—	—	✔
	Качество кода			
.1	Правильные отступы и форматирование	✔	✔	✔
.2	Использование осмысленных имён	—	✔	✔
	Наличие комментариев	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.3	(пояснение new/delete)			
.4	Разделение на функции (декомпозиция)	—	✔	✔
	Отчёт			
.1	Титульный лист, цель, ход работы	✔	✔	✔
.2	Скриншоты выполнения (минимум 3 теста)	✔	✔	✔
.3	Скриншоты для всех трёх уровней	—	✔	✔
.4	Таблица тестирования	—	✔	✔
.5	Блок-схема алгоритма	—	✔	✔
.6	Схема памяти (выделение/освобождение)	—	—	✔
.7	Ответы на контрольные вопросы	—	✔	✔

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) - обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня (№1–15).

Выделить память под динамический массив с помощью new[].

Освободить память с помощью delete[] (без утечек!).

Выполнить требуемую операцию (сумма, среднее, min, max и т.д.).

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) - дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Реализовать изменение размера массива (удаление/вставку элементов).

Выполнить сортировку массива.

Продемонстрировать отсутствие утечек памяти (например, с помощью инструментов или анализа кода).

Обнулять указатель после delete.

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

Блок-схему алгоритма

Повышенный уровень (оценка 5) - требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Двумерный динамический массив:

Корректное выделение памяти (массив указателей на строки или непрерывный блок)

Корректное освобождение в обратном порядке

Перевыделение памяти:

Реализация функции resize, сохраняющей данные

Реализация аналога push_back, pop_back

Защита от утечек:

Все new имеют соответствующий delete

Отсутствие висячих указателей

В отчёте дополнительно представить:

Схему памяти с визуализацией выделения двумерного массива

Сравнительный анализ с std::vector

7.4. Примеры проверки соответствия

Выполненные задания	Оценка	Проверка критериев
Только базовый вариант №1 (сумма)	3	✔ new[], ✔ delete[], ✔ сумма, ✘ нет изменения размера, ✘ нет сортировки
Базовый №1 + средний №2 (сортировка пузырьком)	4	✔ Плюс сортировка, ✔ изменение размера, ✔ проверка nullptr. Но: ✘ нет двумерного массива
Базовый №1 + средний №2 + повышенный №1 (матрица)	5	✔ Все критерии: ✔ двумерный массив, ✔ правильное освобождение, ✔ схема памяти, ✔ нет утечек

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	Влияние	Как исправить
Забыт delete[]	Утечка памяти	Добавить delete[]
Несоответствие new и delete (new[] + delete)	Неопределённое поведение	Использовать delete[] для массивов
Двойное удаление	Неопределённое поведение	Обнулять указатель после delete
Использование после delete	Неопределённое поведение	Не использовать после освобождения
sizeof(arr) для динамического массива	Неверный размер	Хранить размер в отдельной переменной
Утечка при исключении	Утечка памяти	Использовать умные указатели или RAII
Неправильный порядок освобождения матрицы	Утечка	Сначала строки, потом массив указателей

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Разницу между стеком и кучей

Как использовать new[] и delete[]

Как предотвратить утечку памяти

(для оценки 4) - как изменить размер массива

(для оценки 5) - как создать и удалить двумерный динамический массив

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Что произойдёт, если вызвать delete вместо delete[] для массива?"	Понимание разницы
"Как проверить, что память выделена успешно?"	Понимание исключений/nullptr
"Почему после delete нужно обнулять указатель?"	Понимание висячих указателей
"В каком порядке нужно удалять двумерный динамический массив?"	Понимание вложенных выделений

Лабораторная работа №13

Введение в ООП. Классы и объекты

1. Цель работы

Познакомиться с основами объектно-ориентированного программирования (ООП) в C++. Научиться определять классы, создавать объекты, использовать поля (данные) и методы (функции-члены). Освоить спецификаторы доступа public и private, понять разницу между классом и объектом, научиться создавать простые классы с методами для работы с данными (например, класс Point с координатами x, y и методами set(), print()).

2. Теоретическое обоснование

2.1. Парадигма объектно-ориентированного программирования

Объектно-ориентированное программирование (ООП) - это парадигма программирования, в которой программа строится как совокупность объектов, каждый из которых является экземпляром определённого класса. Классы описывают структуру и поведение объектов.

Основные принципы ООП:

1. Инкапсуляция - сокрытие внутреннего состояния объекта и предоставление доступа только через методы.
2. Наследование - создание новых классов на основе существующих.
3. Полиморфизм - возможность объектов разных классов реагировать на одни и те же вызовы методов по-разному.

В этой лабораторной работе мы начнём с изучения классов и объектов - фундаментальных понятий ООП.

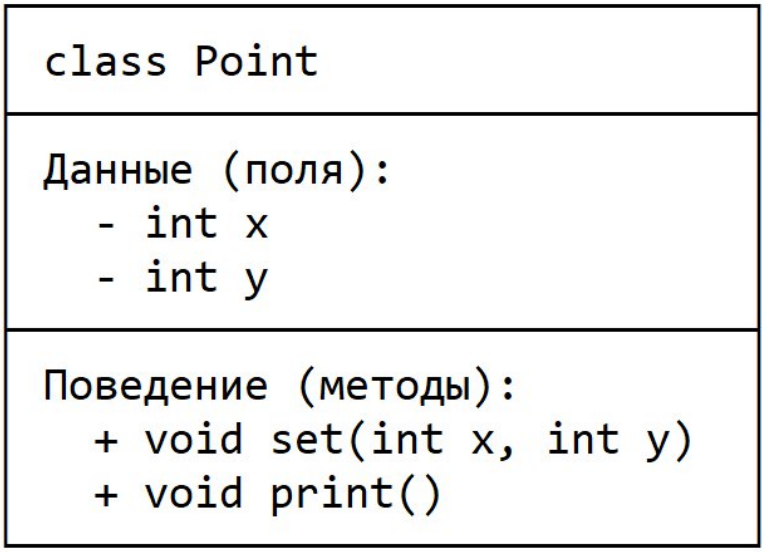
2.2. Класс и объект: определения

Понятие	Определение	Пример
Класс	Шаблон (чертёж),	class Point { ... };

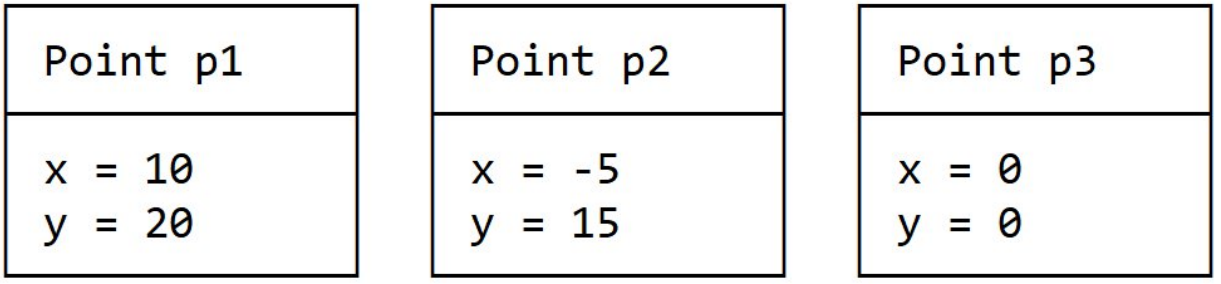
Понятие	Определение	Пример
Класс	описывающий структуру и поведение объектов	
Объект	Конкретный экземпляр класса, имеющий собственное состояние	Point p1;

Аналогия: Класс - это форма для выпечки печенья. Объект - это конкретное печенье, испечённое по этой форме.

Класс Point (чертёж):



Объекты (экземпляры):



2.3. Синтаксис определения класса

```
class ИмяКласса {
private:
    // закрытые поля и методы (доступны только внутри класса)

public:
    // открытые поля и методы (доступны всем)
};
```

Пример простейшего класса:

```
class Point {
public:
    int x;   // координата x
    int y;   // координата y
};
```

Создание и использование объектов:

```
int main() {
    Point p1;        // создание объекта
    p1.x = 10;       // доступ к полю (через точку)
    p1.y = 20;

    std::cout << "(" << p1.x << ", " << p1.y << ")" << std::endl;

    Point p2 = {30, 40};  // агрегатная инициализация (C++11)

    return 0;
}
```

2.4. Спецификаторы доступа

Спецификатор	Доступ	Когда использовать
public	Доступен везде (изнутри и снаружи класса)	Для методов интерфейса
private	Доступен только внутри класса	Для данных (инкапсуляция)
protected	Доступен внутри класса и в классах-наследниках	Для наследования (будет позже)

Принцип инкапсуляции: Данные класса должны быть скрыты (private), а доступ к ним должен осуществляться через открытые методы (public).

```
class GoodPoint {
private:
    int x;   // данные - скрыты
    int y;

public:
    void set(int newX, int newY) {   // открытый метод для изменения
        x = newX;
        y = newY;
    }

    int getX() { return x; }        // открытый метод для чтения
    int getY() { return y; }
```

```
    void print() {
        std::cout << "(" << x << ", " << y << ")" << std::endl;
    }
};

int main() {
    GoodPoint p;
    // p.x = 10;   // ОШИБКА! x - private
    p.set(10, 20);   // правильно: через метод
    p.print();       // (10, 20)
    return 0;
}
```

2.5. Методы класса (функции-члены)

Методы - это функции, определённые внутри класса и работающие с его данными.

Определение метода внутри класса (inline):

```
class Point {
private:
    int x, y;

public:
    void set(int x, int y) {
        this->x = x;   // this - указатель на текущий объект
        this->y = y;
    }
```

```
    void print() {
        std::cout << "(" << x << ", " << y << ")" << std::endl;
    }

    void move(int dx, int dy) {
        x += dx;
        y += dy;
    }
};
```

Определение метода вне класса (с помощью оператора ::):

```
class Point {
private:
    int x, y;

public:
    void set(int x, int y);
    void print();
};

// Определение вне класса
void Point::set(int x, int y) {
    this->x = x;
    this->y = y;
}

void Point::print() {
    std::cout << "(" << x << ", " << y << ")" << std::endl;
}
```

2.6. Указатель this

Внутри каждого нестатического метода класса доступен указатель this, который указывает на текущий объект (тот, для которого был вызван метод).

```
class Point {
private:
    int x, y;

public:
    void set(int x, int y) {
        this->x = x;   // this->x - поле класса, x - параметр
        this->y = y;
    }

    Point& getThis() {
        return *this;  // возвращаем ссылку на текущий объект
    }
};
```

Зачем нужен this:

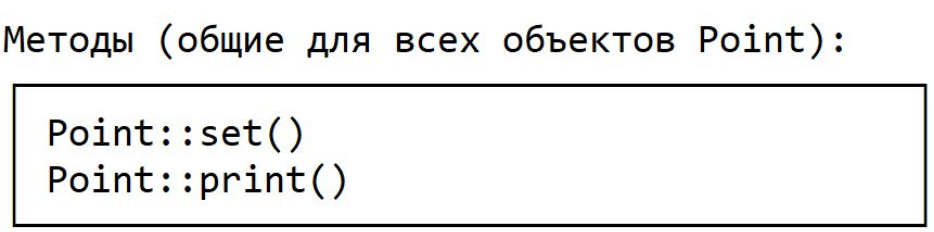
1. Разрешить конфликт имён (параметр и поле имеют одинаковые имена)
2. Возвращать ссылку на текущий объект (для цепочек вызовов)
3. Передавать текущий объект в другие функции

2.7. Создание нескольких объектов

Каждый объект имеет собственную копию полей (данных), но методы общие для всех объектов класса.

```cpp
class Point {
private:
    int x, y;

public:
    void set(int x, int y) {
        this->x = x;
        this->y = y;
    }

    void print() {
        std::cout << "(" << x << ", " << y << ")" << std::endl;
    }
};

int main() {
    Point p1, p2, p3;   // три разных объекта

    p1.set(10, 20);
    p2.set(-5, 15);
    p3.set(0, 0);

    p1.print();   // (10, 20)
    p2.print();   // (-5, 15)
    p3.print();   // (0, 0)

    // Каждый объект хранит свои координаты
    return 0;
}
```

Визуализация в памяти:



2.8. Геттеры и сеттеры (доступ к закрытым полям)

Это стандартный паттерн для доступа к private полям:

```
class Point {
private:
    int x, y;

public:
    // Сеттеры (установка значений)
    void setX(int x) { this->x = x; }
    void setY(int y) { this->y = y; }
    void set(int x, int y) {
        this->x = x;
        this->y = y;
    }

    // Геттеры (получение значений)
    int getX() { return x; }
    int getY() { return y; }
```

```
    void print() {
        std::cout << "(" << x << ", " << y << ")" << std::endl;
    }
};
```

Преимущества геттеров/сеттеров:

1. Контроль доступа (можно добавить проверку корректности)
2. Инкапсуляция (внутреннее представление можно изменить, не меняя интерфейс)
3. Возможность добавить логирование или другую обработку

```
void setX(int x) {
    if (x < 0) {
        this->x = 0;  // защита от отрицательных координат
    } else {
        this->x = x;
    }
}
```

2.9. Объявление объектов: статические и динамические

```
// Статический объект (на стеке)
Point p1;
p1.set(10, 20);

// Динамический объект (в куче)
Point* p2 = new Point;
p2->set(30, 40);    // доступ через -> (стрелка)
p2->print();
```

```
delete p2;          // не забыть освободить!

// Массив объектов
Point points[3];
points[0].set(1, 1);
points[1].set(2, 2);
points[2].set(3, 3);

// Динамический массив объектов
Point* arr = new Point[5];
for (int i = 0; i < 5; i++) {
    arr[i].set(i, i * i);
}
delete[] arr;
```

Различия в синтаксисе доступа:

Тип объекта	Доступ к полю/методу
Статический объект (Point p)	p.x, p.set()
Указатель на объект (Point* p)	p->x, p->set() (или (*p).x)

2.10. Класс Point - полный пример

```
#include <iostream>
#include <cmath>   // для std::sqrt

class Point {
private:
    double x;   // координата X
```

```
    double y;   // координата Y

public:
    // Установка координат
    void set(double x, double y) {
        this->x = x;
        this->y = y;
    }

    // Получение координат
    double getX() { return x; }
    double getY() { return y; }

    // Вывод точки
    void print() {
        std::cout << "(" << x << ", " << y << ")" << std::endl;
    }

    // Расстояние от начала координат
    double distanceFromOrigin() {
        return std::sqrt(x * x + y * y);
    }

    // Расстояние до другой точки
    double distanceTo(const Point& other) {
        double dx = x - other.x;
        double dy = y - other.y;
        return std::sqrt(dx * dx + dy * dy);
    }
```

```
    // Сдвиг точки
    void translate(double dx, double dy) {
        x += dx;
        y += dy;
    }
};

int main() {
    Point a, b;

    a.set(3, 4);
    b.set(6, 8);

    std::cout << "Точка A: ";
    a.print();
    std::cout << "Расстояние от (0,0): " << a.distanceFromOrigin() << std::endl;

    std::cout << "Точка B: ";
    b.print();

    std::cout << "Расстояние между A и B: " << a.distanceTo(b) << std::endl;

    a.translate(1, 1);
    std::cout << "A после сдвига: ";
    a.print();
```

```
    return 0;
}
```

2.11. Типичные ошибки при работе с классами

Ошибка	Пример	Последствие
Забыта точка с запятой после } класса	class Point {} (без ;)	Ошибка компиляции
Попытка доступа к private полю	p.x = 10; если x - private	Ошибка компиляции
Путаница между . и ->	p->x для статического объекта	Ошибка компиляции
Забыт this-> при конфликте имён	x = x; (параметр и поле)	Поле не изменится
Вызов метода на nullptr	Point* p = nullptr; p->set(1,2);	Аварийное завершение
Возврат ссылки/указателя на локальный объект	Point& f() { Point p; return p; }	Висячая ссылка

```
// ОШИБКА: забыта ;
class Point {
    int x, y;
}   // ← забыта ;
```

```
// ОШИБКА: конфликт имён без this
class Point {
    int x;
    void set(int x) {
        x = x;   // присваивание параметра самому себе! поле не меняется
    }
};

// ПРАВИЛЬНО
class Point {
    int x;
    void set(int x) {
        this->x = x;   // явно указываем, что поле = параметр
    }
};
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Простейший класс Point с открытыми полями

```
//
==========================================================
=
// Программа: простейший класс Point с public-полями
// Цель: демонстрация базового синтаксиса класса и создания объектов
//
==========================================================
=

#include <iostream>
```

```
// Определение класса Point
class Point {
public:
    int x;  // координата X (открыта)
    int y;  // координата Y (открыта)
};

int main() {
    // Создание объекта (экземпляра класса Point)
    Point p1;

    // Доступ к полям (через оператор .)
    p1.x = 10;
    p1.y = 20;

    std::cout << "Точка p1: (" << p1.x << ", " << p1.y << ")" << std::endl;

    // Создание второго объекта
    Point p2;
    p2.x = -5;
    p2.y = 15;

    std::cout << "Точка p2: (" << p2.x << ", " << p2.y << ")" << std::endl;

    // Создание и инициализация с помощью списка (агрегатная инициализация)
    Point p3 = {30, 40};
    std::cout << "Точка p3: (" << p3.x << ", " << p3.y << ")" << std::endl;
```

```
    // Массив объектов
    Point points[3] = {{1, 2}, {3, 4}, {5, 6}};

    std::cout << "\nМассив точек:" << std::endl;
    for (int i = 0; i < 3; i++) {
        std::cout << "points[" << i << "]: (" << points[i].x << ", "
                  << points[i].y << ")" << std::endl;
    }

    return 0;
}

// Пример работы:
// Точка p1: (10, 20)
// Точка p2: (-5, 15)
// Точка p3: (30, 40)
// Массив точек:
// points[0]: (1, 2)
// points[1]: (3, 4)
// points[2]: (5, 6)
```

3.2. Пример 2: Класс Point с методами (инкапсуляция)

```
//
==========================================================
=
// Программа: класс Point с закрытыми полями и открытыми методами
// Цель: демонстрация инкапсуляции и методов класса
```

```
//
==========================================================================
=

#include <iostream>
#include <cmath>

class Point {
private:
    int x;    // координата X (скрыта)
    int y;    // координата Y (скрыта)

public:
    // Сеттеры (методы для установки значений)
    void setX(int x) {
        this->x = x;   // this->x - поле, x - параметр
    }

    void setY(int y) {
        this->y = y;
    }

    void set(int x, int y) {
        this->x = x;
        this->y = y;
    }

    // Геттеры (методы для получения значений)
    int getX() {
```

```
    return x;
}

int getY() {
    return y;
}

// Метод для вывода точки
void print() {
    std::cout << "(" << x << ", " << y << ")" << std::endl;
}

// Метод для сдвига точки
void translate(int dx, int dy) {
    x += dx;
    y += dy;
}

// Расстояние от начала координат
double distanceFromOrigin() {
    return std::sqrt(x * x + y * y);
}

// Расстояние между двумя точками
double distanceTo(const Point& other) {
    int dx = x - other.x;
    int dy = y - other.y;
    return std::sqrt(dx * dx + dy * dy);
}
```

```
};

int main() {
    // Создание объектов
    Point p1, p2;

    // Использование сеттеров
    p1.set(3, 4);
    p2.set(6, 8);

    // Использование геттеров
    std::cout << "p1: x = " << p1.getX() << ", y = " << p1.getY() << std::endl;

    // Использование метода print
    std::cout << "p1 = ";
    p1.print();
    std::cout << "p2 = ";
    p2.print();

    // Расстояния
    std::cout << "Расстояние p1 от (0,0): " << p1.distanceFromOrigin() <<
std::endl;
    std::cout << "Расстояние между p1 и p2: " << p1.distanceTo(p2) <<
std::endl;

    // Сдвиг
    p1.translate(1, -1);
    std::cout << "p1 после сдвига: ";
    p1.print();
```

```
    // Демонстрация, что поля нельзя изменить напрямую
    // p1.x = 100;   // ОШИБКА! x - private

    return 0;
}

// Пример работы:
// p1: x = 3, y = 4
// p1 = (3, 4)
// p2 = (6, 8)
// Расстояние p1 от (0,0): 5
// Расстояние между p1 и p2: 5
// p1 после сдвига: (4, 3)
```

3.3. Пример 3: Класс Point с проверкой корректности

```
//
==========================================================
=
// Программа: класс Point с защитой от некорректных значений
// Цель: демонстрация валидации в сеттерах
//
==========================================================
=

#include <iostream>

class Point {
```

```
    private:
        double x;
        double y;

        // Приватный метод для проверки корректности
        bool isValid(double value) {
            // Предположим, координаты должны быть в диапазоне [-100, 100]
            return value >= -100 && value <= 100;
        }

    public:
        // Сеттер с проверкой
        void setX(double x) {
            if (isValid(x)) {
                this->x = x;
            } else {
                std::cout << "Ошибка: значение x = " << x << " вне допустимого
диапазона!"
                          << std::endl;
                this->x = 0;   // значение по умолчанию
            }
        }

        void setY(double y) {
            if (isValid(y)) {
                this->y = y;
            } else {
                std::cout << "Ошибка: значение y = " << y << " вне допустимого
диапазона!"
```

```
                << std::endl;
            this->y = 0;
        }
    }

    void set(double x, double y) {
        setX(x);
        setY(y);
    }

    double getX() { return x; }
    double getY() { return y; }

    void print() {
        std::cout << "(" << x << ", " << y << ")" << std::endl;
    }

    // Сравнение точек
    bool isEqual(const Point& other) {
        return (x == other.x) && (y == other.y);
    }

    // Масштабирование координат
    void scale(double factor) {
        x *= factor;
        y *= factor;
    }
};
```

```cpp
int main() {
    Point p1, p2;

    // Корректные значения
    p1.set(10, 20);
    std::cout << "p1: ";
    p1.print();

    // Некорректные значения (сработает проверка)
    p2.setX(200);    // значение вне диапазона
    p2.setY(50);
    std::cout << "p2: ";
    p2.print();

    // Сравнение точек
    if (p1.isEqual(p2)) {
        std::cout << "Точки равны" << std::endl;
    } else {
        std::cout << "Точки не равны" << std::endl;
    }

    // Масштабирование
    p1.scale(2);
    std::cout << "p1 после масштабирования x2: ";
    p1.print();

    return 0;
}
```

3.4. Пример 4: Динамические объекты и массив объектов

```
//
==========================================================================
=
// Программа: динамические объекты класса Point
// Цель: демонстрация создания объектов в куче
//
==========================================================================
=

#include <iostream>
#include <cstdlib>
#include <ctime>

class Point {
private:
    int x, y;

public:
    void set(int x, int y) {
        this->x = x;
        this->y = y;
    }

    int getX() { return x; }
    int getY() { return y; }

    void print() {
```

```
        std::cout << "(" << x << ", " << y << ")" << std::endl;
    }

    // Расстояние между точками
    double distanceTo(const Point& other) {
        int dx = x - other.x;
        int dy = y - other.y;
        return std::sqrt(dx * dx + dy * dy);
    }
};

int main() {
    std::srand(static_cast<unsigned>(std::time(nullptr)));

    // ========== ДИНАМИЧЕСКИЙ ОБЪЕКТ ==========
    Point* p1 = new Point;   // выделение памяти
    p1->set(10, 20);         // доступ через ->
    std::cout << "Динамический объект p1: ";
    p1->print();
    delete p1;               // освобождение памяти

    // ========== ДИНАМИЧЕСКИЙ МАССИВ ОБЪЕКТОВ ==========
    int size;
    std::cout << "\nВведите количество точек: ";
    std::cin >> size;

    Point* points = new Point[size];   // массив объектов
```

```
// Заполнение случайными координатами
for (int i = 0; i < size; i++) {
    points[i].set(std::rand() % 100, std::rand() % 100);
}

// Вывод массива
std::cout << "\nМассив точек:" << std::endl;
for (int i = 0; i < size; i++) {
    std::cout << "points[" << i << "] = ";
    points[i].print();
}

// Поиск пары точек с минимальным расстоянием
if (size >= 2) {
    double minDist = points[0].distanceTo(points[1]);
    int minI = 0, minJ = 1;

    for (int i = 0; i < size; i++) {
        for (int j = i + 1; j < size; j++) {
            double dist = points[i].distanceTo(points[j]);
            if (dist < minDist) {
                minDist = dist;
                minI = i;
                minJ = j;
            }
        }
    }
```

```
        std::cout << "\nМинимальное расстояние между точками:" <<
std::endl;
        std::cout << "points[" << minI << "] и points[" << minJ << "] = "
             << minDist << std::endl;
    }

    // Освобождение памяти
    delete[] points;

    return 0;
}
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Создать простой класс с открытыми полями и базовыми методами.

№	Класс	Поля	Методы
1	Point	int x, y	set(), print()
2	Rectangle	int width, height	set(), print(), area()
3	Circle	double radius	set(), print(), area(), circumference()

№	Класс	Поля	Методы
4	Book	string title, author, int year	set(), print()
5	Student	string name, int age, double grade	set(), print()
6	Car	string brand, int speed	set(), print(), accelerate()
7	Counter	int value	increment(), decrement(), reset(), getValue()
8	BankAccount	string owner, double balance	deposit(), withdraw(), print()
9	Time	int hours,	set(), print(), addMinutes()

№	Класс	Поля	Методы
		minutes	
10	Fraction	int numerator, denominator	set(), print(), decimal()
11	Triangle	double a, b, c	set(), print(), perimeter()
12	Clock	int hours, minutes, seconds	set(), print(), tick()
13	Date	int day, month, year	set(), print(), nextDay()
14	Product	string name, double price, int	set(), print(), totalCost()

№	Класс	Поля	Методы
		quantity	
15	Vector2D	double x, y	set(), print(), length()

Средний уровень (оценка «хорошо»)

Класс с закрытыми полями, геттерами, сеттерами и дополнительной логикой.

№	Класс	Поля	Методы
1	Point	int x, y	set(), getX(), getY(), print(), distanceTo()
2	Rectangle	int width, height	set(), getWidth(), getHeight(), area(), perimeter(), isSquare()
3	Circle	double radius	set(), getRadius(), area(), circumference(), containsPoint()
4	BankAccount	string owner, double balance (private)	deposit(), withdraw() (с проверкой), getBalance(), print()
5	Student	string name, int age, vector<double> grades	addGrade(), getAverage(), print()

№	Класс	Поля	Методы
6	Counter	int value, int min, int max	increment(), decrement() (с проверкой), reset(), getValue()
7	Time	int hours, minutes, seconds	set(), print(), addSeconds(), toSeconds(), difference()
8	Fraction	int numerator, denominator	set(), simplify(), add(), multiply(), print()
9	String	char* str	set(), print(), length(), concat() (простая версия)
10	Temperature	double celsius	setCelsius(), setFahrenheit(), getCelsius(), getFahrenheit(), print()
11	Vector3D	double x, y, z	set(), print(), length(), dot(), cross()
12	Queue (на массиве)	int* arr, int front, rear, capacity	push(), pop(), isEmpty(), isFull(), print()
13	Stack (на массиве)	int* arr, int top, int capacity	push(), pop(), peek(), isEmpty(), isFull(), print()
14	Complex	double re, im	set(), print(), add(), subtract(), multiply(), abs()
15	Line	Point start,	set(), print(), length(), slope

№	Класс	Поля	Методы
5		end	(), isHorizontal(), isVertical()

Повышенный уровень (оценка «отлично»)

Класс с полной инкапсуляцией, валидацией и дополнительными возможностями.

№	Класс	Задание
1	Point	+ методы для сравнения (isEqual, isGreater), расстояние до произвольной точки, поворот относительно начала координат
2	Polygon (многоугольник)	Массив точек (динамический), методы: добавление вершины, периметр, площадь, проверка выпуклости
3	Matrix	Динамическая матрица, методы: сложение, умножение, транспонирование, определитель (для 3×3)
4	Rational (рациональное число)	Поля: числитель, знаменатель. Операции: сложение, вычитание, умножение, деление, сокращение, сравнение
5	BigInteger (дл	Хранение числа в массиве цифр,

№	Класс	Задание
	инная арифметика)	сложение, вычитание, умножение, сравнение
6	Polynomial (многочлен)	Коэффициенты (динамический массив), методы: вычисление значения, сложение, умножение, производная
7	Set (множество целых чисел)	Динамический массив уникальных элементов, методы: добавление, удаление, объединение, пересечение, разность
8	SparseMatrix (разреженная матрица)	Хранение в виде списка ненулевых элементов, методы: сложение, умножение на вектор
9	Graph (граф)	Матрица смежности или список смежности, методы: добавление ребра, удаление, поиск в глубину/ширину
10	Statistics	Динамический массив чисел, методы: добавление, среднее, медиана, мода, дисперсия, стандартное отклонение
11	Date	Проверка корректности даты (учёт високосных лет, дней в месяцах),

№	Класс	Задание
		методы: добавление дней, разница в днях, день недели
12	MatrixSolver	Решение систем линейных уравнений (метод Гаусса), вычисление обратной матрицы
13	ExpressionParser	Парсинг и вычисление арифметических выражений (+, -, *, /, ^, скобки)
14	Cache (кэш LRU)	Хранение ключ-значение, ограниченный размер, вытеснение наименее используемых
15	SmartPointer	Упрощённая реализация умного указателя (подсчёт ссылок) для одного типа

5. Контрольные вопросы

Раздел А. Основы классов

1. Что такое класс в C++? Чем класс отличается от структуры (struct)?
2. Что такое объект? Как создать объект класса?
3. Какие спецификаторы доступа вы знаете? Что означают public, private, protected?
4. Как обратиться к полю или методу объекта? В чём разница между . и ->?

5. Что такое инкапсуляция? Зачем она нужна?

Раздел Б. Методы класса

1. Что такое метод класса? Как определить метод внутри и вне класса?
2. Что такое указатель this? Для чего он используется?
3. Как разрешить конфликт имён, когда параметр метода совпадает с полем класса?
4. Может ли метод быть private? Зачем?
5. Что такое геттеры и сеттеры? Какие преимущества они дают?

Раздел В. Создание и использование объектов

1. Как создать статический объект на стеке? Как создать динамический объект в куче?
2. Как создать массив объектов? Как создать динамический массив объектов?
3. В чём разница между p.x и p->x? Когда что использовать?
4. Что произойдёт, если вызвать метод на nullptr?
5. Как передать объект в функцию? Что эффективнее: передача по значению, по ссылке или по указателю?

Раздел Г. Принципы ООП

1. Назовите три основных принципа ООП. Кратко опишите каждый.
2. Чем отличается класс от объекта? Приведите аналогию.
3. Почему поля класса обычно делают private, а методы - public?
4. Что такое «интерфейс класса»?
5. Может ли класс не иметь методов? А не иметь полей?

Раздел Д. Анализ кода и ошибки

1. Что не так?

```
class Point {
   int x, y;
}
```

```
int main() {
    Point p;
    p.x = 10;
}
```

2. Найдите ошибку:

```
class Counter {
    int value;
public:
    void increment(int value) {
        value++;
    }
};
```

3. Что выведет код?

```
class A {
    int x = 5;
public:
    void set(int x) { this->x = x; }
    int get() { return x; }
};
int main() {
    A a;
    a.set(10);
    std::cout << a.get();
}
```

4. Чем отличается этот код от предыдущего?

```
class A {
    int x = 5;
public:
    void set(int x) { x = x; }
```

```
    int get() { return x; }
};
```

5. Почему это опасно?

```
class Point {
    int x, y;
public:
    Point* getThis() { return this; }
};
Point* f() {
    Point p;
    return p.getThis();
}
```

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (определение класса и объекта, спецификаторы доступа, методы, указатель this, геттеры/сеттеры)

Ход выполнения работы:

Код программы с подробными комментариями (каждый метод пояснён)

Блок-схема алгоритма (для одного из методов)

Диаграмма классов (UML-подобная схема)

Скриншоты выполнения для различных сценариев

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали о классах и объектах, чем ООП отличается от процедурного программирования)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №13)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) - правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, правильное определение класса, создание объектов, вызов методов).

Оценка «хорошо» (4) - правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего уровня (инкапсуляция с private полями, геттеры/сеттеры, методы с бизнес-логикой, корректное использование this).

Оценка «отлично» (5) - правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (полная инкапсуляция, валидация данных в сеттерах, динамические поля и их корректное освобождение, сложные методы, UML-диаграмма в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
	Компиляция и запуск			
.1	Программа компилируется без ошибок	✔	✔	✔
.2	Программа успешно запускается	✔	✔	✔
.3	Отсутствие предупреждений компилятора	—	✔	✔
	Определение класса			
.1	Правильный синтаксис класса (ключевое слово class, {}, ;)	✔	✔	✔
	Правильное	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	именование класса (с заглавной буквы)			
.3	Наличие спецификаторов доступа	—	✔	✔
.4	Поля объявлены как private (принцип инкапсуляции)	—	✔	✔
	Методы класса			
.1	Правильное определение методов внутри класса	✔	✔	✔
.2	Использование this при конфликте имён	—	✔	✔
.3	Наличие геттеров и сеттеров для полей	—	✔	✔
.4	Валидация входных данных в сеттерах	—	—	✔
	Создание объектов			

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.1	Создание объектов на стеке	✔	✔	✔
.2	Создание динамических объектов (через new)	—	—	✔
.3	Корректное освобождение динамических объектов (delete)	—	—	✔
	Работа с объектами			
.1	Доступ к полям/методам через .	✔	✔	✔
.2	Доступ к полям/методам через -> для указателей	—	—	✔
.3	Создание и использование массивов объектов	—	✔	✔
	Логика класса			

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.1	Корректная реализация требуемой функциональности	✔	✔	✔
.2	Бизнес-методы работают корректно (area, perimeter, deposit и т.д.)	—	✔	✔
.3	Сложные методы (сравнение, расстояние, операции над объектами)	—	—	✔
	Обработка ошибок			
.1	Проверка корректности входных данных в сеттерах	—	—	✔
.2	Проверка отрицательных значений (для размеров, возраста)	—	—	✔
.3	Защита от деления на ноль (для Fraction и т.д.)	—	—	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Качество кода			
.1	Правильные отступы и форматирование	✔	✔	✔
.2	Использование осмысленных имён классов, полей, методов	—	✔	✔
.3	Наличие комментариев (описание класса и методов)	—	✔	✔
.4	Документация метода (что делает, параметры, возврат)	—	—	✔
0	Отчёт			
0.1	Титульный лист, цель, ход работы	✔	✔	✔
0.2	Скриншоты выполнения (минимум 3 теста)	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
0.3	Скриншоты для всех трёх уровней	—	✔	✔
0.4	Таблица тестирования	—	✔	✔
0.5	Блок-схема алгоритма	—	✔	✔
0.6	UML-диаграмма класса	—	—	✔
0.7	Ответы на контрольные вопросы	—	✔	✔

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) - обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня (№1–15).

Правильно определить класс (синтаксис, поля, методы).

Создать объект(ы) класса.

Вызвать методы для демонстрации работы.

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) - дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Использовать инкапсуляцию: поля private, методы public.

Реализовать геттеры и сеттеры для доступа к полям.

Добавить бизнес-методы (area, perimeter, deposit, withdraw и т.д.).

Корректно использовать указатель this при конфликте имён.

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

Блок-схему алгоритма

Повышенный уровень (оценка 5) - требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Полная инкапсуляция и валидация:

Все поля private

Сеттеры проверяют корректность входных данных

Обработка ошибок (например, отрицательный радиус, деление на ноль)

Динамическая память (где применимо):

Динамические поля (массивы, строки)

Правильное выделение и освобождение памяти в конструкторах/деструкторах (хотя деструкторы формально в ЛР14, но можно опередить)

Сложные методы:

Операции над объектами (сложение, вычитание, сравнение)

Возврат новых объектов

В отчёте дополнительно представить:

UML-диаграмму класса

Документацию всех методов

7.4. Примеры проверки соответствия

Выполненные задания	**Оценка**	**Проверка критериев**
Только базовый вариант №1 (Point с public полями)	3	✔ Класс определён, ✔ объекты созданы, ✔ поля открыты. Но: ✘ нет инкапсуляции, ✘ нет геттеров/сеттеров
Базовый №1 + средний №1 (Point с private полями)	4	✔ Плюс инкапсуляция, ✔ геттеры/сеттеры, ✔ методы distanceTo. Но: ✘ нет валидации, ✘ нет динамических полей
Базовый №1 + средний №1 + повышенный №1 (Point с валидацией)	5	✔ Все критерии: ✔ полная инкапсуляция, ✔ валидация в сеттерах, ✔ сравнение точек, ✔ UML-диаграмма

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	Влияние	Как исправить
Забыта ; после } класса	Ошибка компиляции	Добавить ;
Попытка доступа к private полю	Ошибка компиляции	Использовать геттеры/сеттеры
Конфликт имён без this	Поле не	Использовать

Ошибка	Влияние	Как исправить
	изменяется	this->поле = параметр
Путаница между . и ->	Ошибка компиляции	Для статических объектов - ., для указателей - ->
Утечка памяти для динамических объектов	Утечка памяти	Вызывать dele te
Метод изменяет объект, но назван как геттер	Путаница	Геттеры не должны изменять состояние

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Что такое класс и объект

Разницу между public и private

Как создать объект и вызвать его метод

(для оценки 4) - зачем нужны геттеры и сеттеры

(для оценки 5) - как работает инкапсуляция и валидация

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Что произойдёт, если в методе написать x = x; без this?"	Понимание this
"Зачем делать поля private, если есть геттеры и сеттеры?"	Понимание инкапсуляции
"Может ли класс не иметь методов?"	Понимание минимальной структуры класса
"Как передать объект в функцию, чтобы избежать копирования?"	Понимание передачи по ссылке

Лабораторная работа №14

Конструкторы и деструктор

1. Цель работы

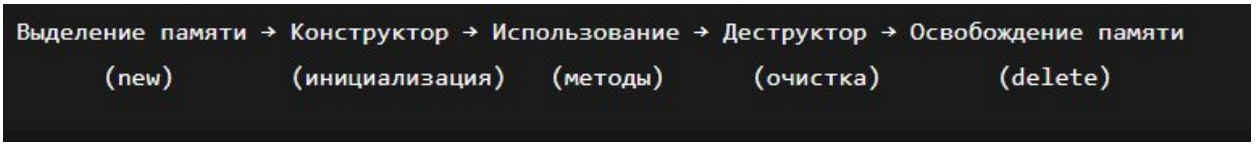
Освоить механизмы автоматической инициализации и очистки объектов в C++. Научиться создавать и использовать конструкторы (конструктор по умолчанию, параметризованный конструктор, конструктор копирования) и деструктор. Понимать порядок вызова конструкторов и деструктора, научиться управлять динамическими ресурсами в классах, предотвращать утечки памяти.

2. Теоретическое обоснование

2.1. Жизненный цикл объекта

Каждый объект в C++ проходит следующие этапы:

1. Выделение памяти - под объект выделяется место (на стеке или в куче)
2. Инициализация - вызывается конструктор для установки начального состояния
3. Использование - вызов методов объекта
4. Уничтожение - вызывается деструктор для освобождения ресурсов
5. Освобождение памяти - память возвращается системе



Важно: Если вы не определите конструктор и деструктор, компилятор создаст их автоматически (но они могут не делать то, что вам нужно).

2.2. Конструктор: определение и синтаксис

Конструктор - это специальный метод класса, который:

1. Имеет то же имя, что и класс
2. Не возвращает значение (даже void не пишется)
3. Вызывается автоматически при создании объекта
4. Может быть перегружен (несколько конструкторов с разными параметрами)

Синтаксис:

```
class ClassName {
public:
    ClassName();                // конструктор по умолчанию
```

```
    ClassName(параметры);          // параметризованный конструктор
    ClassName(const ClassName& other); // конструктор копирования
};
```

Пример:

```
class Point {
private:
    int x, y;

public:
    // Конструктор по умолчанию (без параметров)
    Point() {
        x = 0;
        y = 0;
        std::cout << "Point() вызван" << std::endl;
    }

    // Параметризованный конструктор
    Point(int x, int y) {
        this->x = x;
        this->y = y;
        std::cout << "Point(int, int) вызван" << std::endl;
    }
};

int main() {
    Point p1;          // вызов конструктора по умолчанию
    Point p2(10, 20);  // вызов параметризованного конструктора
    Point p3 = {30, 40}; // агрегатная инициализация (C++11)
```

```
    return 0;
}
```

2.3. Конструктор по умолчанию (default constructor)

Конструктор по умолчанию - это конструктор, который можно вызвать без аргументов.

```
class Point {
private:
    int x, y;

public:
    // Явно определённый конструктор по умолчанию
    Point() {
        x = 0;
        y = 0;
    }

    // Или с параметрами по умолчанию
    Point(int x = 0, int y = 0) {
        this->x = x;
        this->y = y;
    }
};
```

Если вы не определили ни одного конструктора, компилятор создаст конструктор по умолчанию автоматически (он не инициализирует поля - в них будет мусор!).

```
class Point {
```

```
public:
    int x, y;   // public, чтобы можно было вывести
};

int main() {
    Point p;    // автоматический конструктор по умолчанию
    std::cout << p.x << ", " << p.y;  // мусорные значения!
    return 0;
}
```

Как попросить компилятор создать конструктор по умолчанию (C++11):

```
class Point {
private:
    int x = 0, y = 0;   // in-class инициализация

public:
    Point() = default;   // компилятор сгенерирует конструктор
};
```

2.4. Параметризованный конструктор

Конструктор с параметрами позволяет инициализировать объект нужными значениями при создании.

```
class Point {
private:
    int x, y;
```

```
public:
    Point(int x, int y) {
        this->x = x;
        this->y = y;
    }

    void print() {
        std::cout << "(" << x << ", " << y << ")" << std::endl;
    }
};

int main() {
    Point p1(10, 20);        // прямая инициализация
    Point p2 = Point(30, 40); // копирующая инициализация
    Point p3{50, 60};        // uniform initialization (C++11)
    Point p4 = {70, 80};     // агрегатная инициализация

    return 0;
}
```

Список инициализации (member initializer list) - предпочтительный способ инициализации полей в конструкторе:

```
class Point {
private:
    int x, y;
    const int id;      // константное поле
    int& ref;          // ссылка

public:
```

```
    // Список инициализации: инициализирует поля до входа в тело конструктора
    Point(int x, int y, int id, int& ref)
        : x(x), y(y), id(id), ref(ref)   // обязательно для const и ссылок
    {
        // тело конструктора (можно добавить логику)
        std::cout << "Point создан: (" << x << ", " << y << ")" << std::endl;
    }
};
```

Преимущества списка инициализации:

1. Обязателен для const полей и ссылок
2. Более эффективен (прямая инициализация вместо присваивания)
3. Единственный способ инициализации объектов-членов перед их конструкторами

2.5. Конструктор копирования (copy constructor)

Конструктор копирования создаёт новый объект как копию существующего.

Синтаксис: ClassName(const ClassName& other)

```
class Point {
private:
    int x, y;

public:
    Point(int x, int y) : x(x), y(y) {}

    // Конструктор копирования
    Point(const Point& other) : x(other.x), y(other.y) {
```

```
        std::cout << "Copy constructor called" << std::endl;
    }
};

int main() {
    Point p1(10, 20);
    Point p2 = p1;      // вызов конструктора копирования
    Point p3(p1);       // вызов конструктора копирования
    Point p4{p1};       // вызов конструктора копирования

    return 0;
}
```

Когда вызывается конструктор копирования:

1. При инициализации нового объекта существующим: `Point p2 = p1;`
2. При передаче объекта в функцию по значению
3. При возврате объекта из функции по значению

```
void func(Point p) { }    // передача по значению - копирование

Point createPoint() {
    Point p(10, 20);
    return p;           // возврат по значению - копирование
}
```

Если не определить конструктор копирования, компилятор создаст поверхностную копию (копирует все поля побайтово). Для классов с динамической памятью это опасно!

2.6. Деструктор

Деструктор - это специальный метод класса, который:

1. Имеет имя ~ClassName (тильда + имя класса)
2. Не возвращает значение и не принимает параметров
3. Вызывается автоматически при уничтожении объекта
4. Служит для освобождения ресурсов (память, файлы, сетевые соединения)

Синтаксис:

```
class ClassName {
public:
    ~ClassName();   // деструктор
};
```

Пример:

```
#include <iostream>

class Point {
private:
    int x, y;

public:
    Point(int x, int y) : x(x), y(y) {
        std::cout << "Point(" << x << ", " << y << ") создан" << std::endl;
    }

    ~Point() {
        std::cout << "Point(" << x << ", " << y << ") уничтожен" << std::endl;
    }
};
```

```
int main() {
    Point p1(10, 20);
    Point p2(30, 40);

    std::cout << "Конец main" << std::endl;
    return 0;
}
// Вывод:
// Point(10, 20) создан
// Point(30, 40) создан
// Конец main
// Point(30, 40) уничтожен
// Point(10, 20) уничтожен   (обратный порядок!)
```

Порядок вызова деструкторов: объекты уничтожаются в порядке, обратном созданию (LIFO - Last In, First Out).

2.7. Правило трёх (Rule of Three)

Если класс управляет динамическими ресурсами (память, файлы, и т.д.), следует определить три специальные функции:

1. Деструктор - освободить ресурсы
2. Конструктор копирования - скопировать ресурсы (глубокая копия)
3. Оператор присваивания копированием (operator=) - скопировать ресурсы (будет в ЛР18)

```
class String {
private:
    char* data;
```

```
    int length;

public:
    // Конструктор
    String(const char* str = "") {
        length = strlen(str);
        data = new char[length + 1];
        strcpy(data, str);
    }

    // Конструктор копирования (глубокая копия)
    String(const String& other) {
        length = other.length;
        data = new char[length + 1];
        strcpy(data, other.data);
    }

    // Деструктор
    ~String() {
        delete[] data;
    }
};
```

Почему это важно:

```
String s1("Hello");
String s2 = s1;  // без конструктора копирования - поверхностная копия!
// s1.data и s2.data указывают на одну и ту же память
// при уничтожении - double free!
```

2.8. Порядок вызова конструкторов и деструктора

```
#include <iostream>

class A {
public:
    A() { std::cout << "A constructor" << std::endl; }
    ~A() { std::cout << "A destructor" << std::endl; }
};

class B {
public:
    B() { std::cout << "B constructor" << std::endl; }
    ~B() { std::cout << "B destructor" << std::endl; }
};

class C {
private:
    A a;  // объект A как поле
    B b;  // объект B как поле

public:
    C() { std::cout << "C constructor" << std::endl; }
    ~C() { std::cout << "C destructor" << std::endl; }
};

int main() {
    C c;
    return 0;
```

```
}
// Вывод:
// A constructor  (сначала поля в порядке объявления)
// B constructor
// C constructor
// C destructor    (деструктор вызывается для полей в обратном порядке)
// B destructor
// A destructor
```

Порядок:

Конструкторы: сначала базовый класс (если есть), потом поля в порядке объявления, потом тело конструктора

Деструкторы: сначала тело деструктора, потом поля в обратном порядке объявления, потом базовый класс

2.9. explicit конструктор

Ключевое слово explicit запрещает неявные преобразования.

```
class Point {
private:
    int x, y;

public:
    explicit Point(int x) : x(x), y(0) {
        std::cout << "Point(int) called" << std::endl;
    }
};

void printPoint(Point p) {
    // ...
```

```
}

int main() {
    Point p1 = 5;    // ОШИБКА! explicit запрещает
    Point p2(5);     // OK: явный вызов

    printPoint(10);  // ОШИБКА! нельзя преобразовать int → Point
    printPoint(Point(10)); // OK

    return 0;
}
```

2.10. = default и = delete (C++11)

```
class Point {
private:
    int x, y;

public:
    Point() = default;                  // стандартный конструктор
    Point(int x, int y) : x(x), y(y) {}

    Point(const Point&) = delete;       // запретить копирование
    Point& operator=(const Point&) = delete;
};
```

2.11. Типичные ошибки при работе с конструкторами/деструктором

Ошибка	Пример	Последствие
Конструктор возвращает значение	int Point() {}	Ошибка компиляции
Деструктор с параметрами	~Point(int x) {}	Ошибка компиляции
Не инициализированы поля	Конструктор без списка инициализации	Мусорные значения
Забыт деструктор для класса с динамической памятью	Нет delete[]	Утечка памяти
Поверхностное копирование	Нет конструктора копирования	Double free
Вызов delete[] в деструкторе для new (не new[])	Несоответствие	Неопределённое поведение

```
// ОШИБКА: конструктор возвращает тип
int Point(int x, int y) {  // не может быть int перед именем!
    this->x = x;
    this->y = y;
}

// ПРАВИЛЬНО
```

```
    Point(int x, int y) {
        this->x = x;
        this->y = y;
    }
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Класс Point с конструкторами и деструктором

```
    //
==========================================================================
=
    // Программа: класс Point с конструкторами и деструктором
    // Цель: демонстрация вызова различных конструкторов и деструктора
    //
==========================================================================
=

    #include <iostream>

    class Point {
    private:
        int x;
        int y;

    public:
        // Конструктор по умолчанию
        Point() {
            x = 0;
            y = 0;
```

```
        std::cout << "Point() вызван: (" << x << ", " << y << ")" << std::endl;
    }

    // Параметризованный конструктор
    Point(int x, int y) {
        this->x = x;
        this->y = y;
        std::cout << "Point(int, int) вызван: (" << x << ", " << y << ")" <<
std::endl;
    }

    // Конструктор копирования
    Point(const Point& other) {
        x = other.x;
        y = other.y;
        std::cout << "Point(const Point&) вызван: (" << x << ", " << y << ")"
<< std::endl;
    }

    // Деструктор
    ~Point() {
        std::cout << "~Point() вызван: (" << x << ", " << y << ")" << std::endl;
    }

    void set(int x, int y) {
        this->x = x;
        this->y = y;
    }
```

```
    void print() const {
        std::cout << "Point(" << x << ", " << y << ")" << std::endl;
    }
};

// Функция для демонстрации вызова конструктора копирования при передаче по значению
void printByValue(Point p) {
    std::cout << "printByValue: ";
    p.print();
}

// Функция для демонстрации возврата по значению
Point createPoint(int x, int y) {
    Point p(x, y);
    std::cout << "createPoint: создан локальный объект" << std::endl;
    return p;  // вызов конструктора копирования (может быть оптимизирован)
}

int main() {
    std::cout << "=== Создание объектов ===" << std::endl;

    Point p1;                  // конструктор по умолчанию
    Point p2(10, 20);          // параметризованный конструктор
    Point p3 = p2;             // конструктор копирования
    Point p4(30, 40);

    std::cout << "\n=== Объекты после создания ===" << std::endl;
```

```
    p1.print();
    p2.print();
    p3.print();
    p4.print();

    std::cout << "\n=== Передача по значению ===" << std::endl;
    printByValue(p4);           // копирование при передаче

    std::cout << "\n=== Возврат из функции ===" << std::endl;
    Point p5 = createPoint(50, 60);
    p5.print();

    std::cout << "\n=== Выход из main ===" << std::endl;
    return 0;
}
// Вывод будет показывать порядок вызовов конструкторов и деструкторов
```

3.2. Пример 2: Класс с динамической памятью (String)

```
//
==========================================================
=
// Программа: класс String с динамической памятью
// Цель: демонстрация необходимости конструктора копирования и деструктора
//
==========================================================
=
```

```
#include <iostream>
#include <cstring>

class String {
private:
    char* data;      // указатель на динамический массив символов
    int length;      // длина строки

public:
    // Конструктор из C-строки
    String(const char* str = "") {
        length = std::strlen(str);
        data = new char[length + 1];    // +1 для '\0'
        std::strcpy(data, str);
        std::cout << "String(\"" << str << "\") создан" << std::endl;
    }

    // Конструктор копирования (глубокая копия!)
    String(const String& other) {
        length = other.length;
        data = new char[length + 1];
        std::strcpy(data, other.data);
        std::cout << "String(const String&) скопирован: \"" << data << "\"" <<
std::endl;
    }

    // Деструктор
    ~String() {
```

```
        std::cout << "~String(): \"" << data << "\" удалён" << std::endl;
        delete[] data;
    }

    // Вывод строки
    void print() const {
        std::cout << "String: \"" << data << "\"" << std::endl;
    }

    // Получение длины
    int getLength() const {
        return length;
    }

    // Конкатенация (создаёт новую строку)
    String concat(const String& other) const {
        int newLength = length + other.length;
        char* newData = new char[newLength + 1];
        std::strcpy(newData, data);
        std::strcat(newData, other.data);

        String result(newData);
        delete[] newData;
        return result;
    }
};

int main() {
    std::cout << "=== Создание строк ===" << std::endl;
```

```
    String s1("Hello");
    String s2("World");

    std::cout << "\n=== Копирование ===" << std::endl;
    String s3 = s1;  // конструктор копирования

    std::cout << "\n=== Конкатенация ===" << std::endl;
    String s4 = s1.concat(s2);

    std::cout << "\n=== Вывод строк ===" << std::endl;
    s1.print();
    s2.print();
    s3.print();
    s4.print();

    std::cout << "\n=== Выход из main ===" << std::endl;
    return 0;
}
// Важно: без конструктора копирования была бы double free!
```

3.3. Пример 3: Класс Array с динамическим массивом

```
//
==========================================================
=
// Программа: класс Array для работы с динамическим массивом
// Цель: демонстрация управления динамической памятью
```

```
//
==========================================================================
=

#include <iostream>

class Array {
private:
    int* data;      // указатель на массив
    int size;       // размер массива

public:
    // Конструктор с размером
    Array(int size) : size(size) {
        data = new int[size];
        for (int i = 0; i < size; i++) {
            data[i] = 0;
        }
        std::cout << "Array(" << size << ") создан" << std::endl;
    }

    // Конструктор копирования
    Array(const Array& other) : size(other.size) {
        data = new int[size];
        for (int i = 0; i < size; i++) {
            data[i] = other.data[i];
        }
        std::cout << "Array(const Array&) скопирован" << std::endl;
    }
```

```
    // Деструктор
    ~Array() {
        std::cout << "~Array() удалён (size=" << size << ")" << std::endl;
        delete[] data;
    }

    // Доступ к элементам
    int& at(int index) {
        if (index < 0 || index >= size) {
            std::cout << "Ошибка: индекс " << index << " вне диапазона!" <<
std::endl;
            static int dummy = 0;
            return dummy;
        }
        return data[index];
    }

    const int& at(int index) const {
        if (index < 0 || index >= size) {
            std::cout << "Ошибка: индекс " << index << " вне диапазона!" <<
std::endl;
            static int dummy = 0;
            return dummy;
        }
        return data[index];
    }

    // Получение размера
```

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

    // Заполнение массива значением
    void fill(int value) {
        for (int i = 0; i < size; i++) {
            data[i] = value;
        }
    }

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

// Функция, демонстрирующая копирование при передаче
void printArray(Array arr) {
    arr.print();   // переданная копия
    std::cout << " (копия)" << std::endl;
}

int main() {
```

```
    std::cout << "=== Создание массива ===" << std::endl;
    Array a1(5);
    a1.fill(10);

    std::cout << "a1 = ";
    a1.print();
    std::cout << std::endl;

    std::cout << "\n=== Копирование ===" << std::endl;
    Array a2 = a1;  // конструктор копирования
    std::cout << "a2 = ";
    a2.print();
    std::cout << std::endl;

    std::cout << "\n=== Изменение a2 (не влияет на a1) ===" << std::endl;
    a2.fill(99);
    std::cout << "a1 = ";
    a1.print();
    std::cout << std::endl;
    std::cout << "a2 = ";
    a2.print();
    std::cout << std::endl;

    std::cout << "\n=== Передача по значению ===" << std::endl;
    printArray(a1);

    std::cout << "\n=== Доступ по индексу ===" << std::endl;
    a1.at(2) = 42;
    std::cout << "a1[2] = " << a1.at(2) << std::endl;
```

```
    a1.at(10);   // проверка границ

    std::cout << "\n=== Выход из main ===" << std::endl;
    return 0;
}
```

3.4. Пример 4: Класс Counter с разными конструкторами

```
//
==========================================================================
=
// Программа: класс Counter с различными конструкторами
// Цель: демонстрация перегрузки конструкторов и явного/explicit
//
==========================================================================
=

#include <iostream>

class Counter {
private:
    int count;
    int minValue;
    int maxValue;

public:
    // Конструктор по умолчанию: значение = 0, диапазон [0, 100]
    Counter() : count(0), minValue(0), maxValue(100) {
        std::cout << "Counter() создан: count=" << count
```

```
                << ", диапазон [" << minValue << ", " << maxValue << "]" <<
std::endl
        }

        // Конструктор с заданным начальным значением (диапазон по
умолчанию)
        explicit Counter(int initial) : count(initial), minValue(0), maxValue(100) {
            if (count < minValue) count = minValue;
            if (count > maxValue) count = maxValue;
            std::cout << "Counter(int) создан: count=" << count
                << ", диапазон [" << minValue << ", " << maxValue << "]" <<
std::endl
        }

        // Конструктор с диапазоном и начальным значением
        Counter(int min, int max, int initial = 0)
            : minValue(min), maxValue(max), count(initial) {
            if (count < minValue) count = minValue;
            if (count > maxValue) count = maxValue;
            std::cout << "Counter(int, int, int) создан: count=" << count
                << ", диапазон [" << minValue << ", " << maxValue << "]" <<
std::endl
        }

        // Конструктор копирования
        Counter(const Counter& other)
            :           count(other.count),           minValue(other.minValue),
maxValue(other.maxValue) {
```

```
        std::cout << "Counter(const Counter&) скопирован: count=" << count
<< std::endl;
    }

    // Деструктор
    ~Counter() {
        std::cout << "~Counter() уничтожен: count=" << count << std::endl;
    }

    // Увеличение счётчика
    void increment() {
        if (count < maxValue) {
            count++;
            std::cout << "increment: count=" << count << std::endl;
        } else {
            std::cout << "Ошибка: достигнут максимум (" << maxValue << ")"
<< std::endl;
        }
    }

    // Уменьшение счётчика
    void decrement() {
        if (count > minValue) {
            count--;
            std::cout << "decrement: count=" << count << std::endl;
        } else {
            std::cout << "Ошибка: достигнут минимум (" << minValue << ")"
<< std::endl;
        }
```

```
        }

        int getValue() const {
            return count;
        }

        void print() const {
            std::cout << "Counter: count=" << count
                      << ", range=[" << minValue << ", " << maxValue << "]" <<
std::endl;
        }
    };

    // Демонстрация explicit конструктора
    void useCounter(Counter c) {
        c.print();
    }

    int main() {
        std::cout << "=== Создание счётчиков ===" << std::endl;

        Counter c1;                 // конструктор по умолчанию
        Counter c2(42);             // explicit конструктор
        Counter c3(10, 20, 15);     // параметризованный конструктор
        Counter c4 = c2;            // конструктор копирования

        std::cout << "\n=== Использование счётчиков ===" << std::endl;
        c1.print();
        c2.print();
```

```
    c3.print();
    c4.print();

    std::cout << "\n=== Операции с c3 ===" << std::endl;
    c3.increment();
    c3.increment();
    c3.decrement();

    std::cout << "\n=== explicit демонстрация ===" << std::endl;
    useCounter(c1);              // OK
    // useCounter(100);          // ОШИБКА! explicit запрещает
    useCounter(Counter(100));    // OK: явное создание

    std::cout << "\n=== Выход из main ===" << std::endl;
    return 0;
}
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Добавить конструктор и деструктор к классам из ЛР13.

	Класс из ЛР13	Добавить
	Point	конструктор по умолчанию, конструктор с параметрами, деструктор с выводом
	Rectangle	конструкторы, деструктор
	Circle	конструкторы, деструктор

	Класс из ЛР13	Добавить
	Book	конструкторы, деструктор
	Student	конструкторы, деструктор
	Car	конструкторы, деструктор
	Counter	конструкторы (по умолчанию, с параметром), деструктор
	BankAccount	конструкторы (по умолчанию, с параметрами), деструктор
	Time	конструкторы, деструктор
0	Fraction	конструкторы (по умолчанию, с параметрами), деструктор
1	Triangle	конструкторы, деструктор
2	Clock	конструкторы, деструктор
3	Date	конструкторы, деструктор
4	Product	конструкторы, деструктор
	Vector2D	конструкторы, деструктор

	Класс из ЛР13	Добавить
5		

Средний уровень (оценка «хорошо»)

Добавить конструктор копирования, продемонстрировать его вызов.

	Класс	Дополнительные требования
	Point	+ конструктор копирования, демонстрация вызова при передаче/возврате
	Rectangle	+ конструктор копирования, демонстрация
	Circle	+ конструктор копирования, проверка, что радиус не отрицательный
	BankAccount	+ конструктор копирования, демонстрация отдельного копирования баланса
	Student	+ конструктор копирования, копирование массива оценок (динамический)
	Counter	+ конструктор копирования, копирование диапазона и значения

		Класс	Дополнительные требования
		Time	+ конструктор копирования, проверка корректности времени
		Fraction	+ конструктор копирования, сокращение дроби в конструкторе
		String (упрощённый)	конструкторы, деструктор, конструктор копирования (глубокая копия)
0		Array (упрощённый)	конструкторы, деструктор, конструктор копирования
1		Vector3D	+ конструктор копирования, демонстрация
2		Complex	+ конструктор копирования, демонстрация
3		Date	+ конструктор копирования, проверка корректности даты
4		Temperature	+ конструктор копирования, демонстрация
5		Line	+ конструктор копирования, демонстрация

Повышенный уровень (оценка «отлично»)

Класс с динамической памятью, конструктор копирования, деструктор, демонстрация глубокого копирования.

№	Класс	Задание
1	String	Полная реализация: конструкторы (по умолчанию, из C-строки, копирования), деструктор, методы: length(), c_str(), print(), concat(), substring()
2	Array	Полная реализация: конструкторы (размер, копирования), деструктор, методы: at(), size(), fill(), resize(), sort()
3	Matrix	Динамическая матрица: конструкторы, деструктор, копирования, методы: get(), set(), print(), transpose(), add()
4	Polynomial	Многочлен: динамический массив коэффициентов, конструкторы, деструктор, копирования, методы: evaluate(), add(), multiply(), derivative()
5	Vector	Шаблонный вектор (упрощённый аналог std::vector): конструкторы, деструктор, копирования, методы: push_back(), pop_back(), at(), size(), capacity()
6	StringBui	Класс для эффективной конкатенации

№	Класс	Задание
	lder	строк: динамический буфер, конструкторы, деструктор, копирования, методы: append(), toString(), clear()
7	IntArray	Массив целых чисел с проверкой границ: конструкторы, деструктор, копирования, перегрузка operator[] (будет в ЛР18)
8	Stack (шаблонный)	Стек на динамическом массиве: конструкторы, деструктор, копирования, методы: push(), pop(), top(), isEmpty(), size()
9	Queue (шаблонный)	Очередь на динамическом массиве (кольцевая): конструкторы, деструктор, копирования, методы: enqueue(), dequeue(), front(), isEmpty(), size()
10	SparseVector	Разреженный вектор (хранение ненулевых элементов в динамических массивах индексов и значений): конструкторы, деструктор, копирования, методы: dot(), norm()
11	DynamicStringArray	Массив строк с автоматическим управлением памятью, конструкторы, деструктор, копирования, методы: add(), remove(), get(), size()

№	Класс	Задание
12	Bitset	Динамический битовый массив: конструкторы, деструктор, копирования, методы: set(), reset(), test(), count(), to_string()
13	PoolAllocator	Простой пул памяти для объектов фиксированного размера: конструктор, деструктор (запретить копирование), методы: allocate(), deallocate()
14	SharedPointer	Упрощённый умный указатель со счётчиком ссылок: конструкторы, деструктор, копирования (инкремент счётчика), методы: get(), use_count()
15	Image	Класс для работы с изображением (динамическая матрица пикселей): конструкторы, деструктор, копирования, методы: getPixel(), setPixel(), grayscale(), invert(), printAscii()

5. Контрольные вопросы

Раздел А. Конструкторы

1. Что такое конструктор? Каковы его особенности (имя, возвращаемое значение, вызов)?
2. Что такое конструктор по умолчанию? Когда он вызывается?
3. Что такое параметризованный конструктор? Как его вызвать?
4. Что такое список инициализации (member initializer list)? В каких случаях он обязателен?

5. Может ли класс иметь несколько конструкторов? Как это называется?
6. Что произойдёт, если не определить конструктор?
7. Что такое explicit конструктор? Зачем он нужен?

Раздел Б. Конструктор копирования

1. Что такое конструктор копирования? Какой у него прототип?
2. В каких случаях вызывается конструктор копирования? Приведите 3 примера.
3. Что такое поверхностная (shallow) копия? Чем она опасна?
4. Что такое глубокая (deep) копия? Когда она нужна?
5. Что произойдёт, если для класса с динамической памятью не определить конструктор копирования?

Раздел В. Деструктор

1. Что такое деструктор? Каков его синтаксис и особенности?
2. Когда вызывается деструктор?
3. В каком порядке вызываются деструкторы для полей класса?
4. Что такое «утечка памяти»? Как деструктор помогает её предотвратить?
5. Может ли деструктор быть перегружен? Почему?

Раздел Г. Правило трёх

1. Что такое «Правило трёх» (Rule of Three)? Какие функции нужно определить?
2. Почему для класса с динамической памятью нужно определять все три функции?
3. Что произойдёт при использовании = default и = delete для специальных функций?

Раздел Д. Анализ кода и ошибки

1. Что выведет код?

```
class A {
```

```
public:
    A() { std::cout << "A() "; }
    ~A() { std::cout << "~A() "; }
};
int main() { A a; A b; return 0; }
```

2. Найдите ошибку:

```
class String {
    char* data;
public:
    String(const char* s) { data = new char[strlen(s)+1]; strcpy(data, s); }
};
```

3. Почему этот код вызывает проблему?

```
class Array {
    int* data;
public:
    Array(int size) { data = new int[size]; }
};
int main() { Array a1(10); Array a2 = a1; return 0; }
```

4. Что не так с конструктором копирования?

```
class Point {
    int x, y;
public:
    Point(const Point& other) { other.x = x; }
};
```

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (конструкторы: по умолчанию, параметризованный, копирования; список инициализации; деструктор; правило трёх)

Ход выполнения работы:

Код программы с подробными комментариями (каждый конструктор и деструктор пояснён)

Блок-схема алгоритма (для одного из методов)

Диаграмма классов (UML-подобная схема с конструкторами/деструктором)

Схема вызовов (порядок вызова конструкторов и деструкторов)

Скриншоты выполнения с демонстрацией порядка вызовов

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали о конструкторах и деструкторе, почему они важны для управления ресурсами)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №14)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) - правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, наличие конструктора по умолчанию, параметризованного конструктора и деструктора с выводом сообщений).

Оценка «хорошо» (4) - правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего уровня (наличие конструктора копирования, демонстрация его вызова при передаче/возврате объектов, корректное управление динамической памятью).

Оценка «отлично» (5) - правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (класс с динамической памятью, глубокая копия, корректные конструкторы копирования и деструктор, полное отсутствие утечек памяти, демонстрация правила трёх, UML-диаграмма в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
	Компиляция и запуск			
	Программа	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.1	компилируется без ошибок			
.2	Программа успешно запускается	✔	✔	✔
.3	Отсутствие предупреждений компилятора	—	✔	✔
	Конструктор по умолчанию			
.1	Конструктор по умолчанию определён	✔	✔	✔
.2	Корректная инициализация полей	✔	✔	✔
.3	Вывод сообщения о вызове	✔	✔	✔
	Параметризованный конструктор			
.1	Конструктор с параметрами определён	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	Корректная инициализация (список инициализации или тело)	✔	✔	✔
.3	Вывод сообщения о вызове	✔	✔	✔
	Конструктор копирования			
.1	Конструктор копирования определён	—	✔	✔
.2	Глубокая копия (для динамических ресурсов)	—	✔	✔
.3	Демонстрация вызова при передаче/возврате	—	✔	✔
.4	Отсутствие поверхностного копирования	—	—	✔
	Деструктор			
.1	Деструктор определён	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	Освобождение ресурсов (память, файлы)	—	✔	✔
.3	Вывод сообщения о вызове	✔	✔	✔
.4	Отсутствие утечек памяти	—	✔	✔
	Правило трёх			
.1	Все три функции определены (для динамических классов)	—	—	✔
.2	Корректная семантика копирования	—	—	✔
.3	Отсутствие double-free	—	—	✔
	Демонстрация порядка вызовов			
.1	Вывод сообщений в конструкторах/деструкторе	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	Создание нескольких объектов для демонстрации	✔	✔	✔
.3	Демонстрация обратного порядка уничтожения	—	✔	✔
	Обработка ошибок			
.1	Проверка корректности входных данных в конструкторах	—	✔	✔
.2	Проверка границ (для массивов)	—	✔	✔
.3	Защита от отрицательных значений	—	—	✔
0	Качество кода			
0.1	Правильные отступы и форматирование	✔	✔	✔
0.2	Использование списка инициализации	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
0.3	Наличие комментариев (пояснение работы конструкторов)	—	✔	✔
0.4	Использование explicit для однопараметрических конструкторов	—	—	✔
1	Отчёт			
1.1	Титульный лист, цель, ход работы	✔	✔	✔
1.2	Скриншоты выполнения (с демонстрацией вызовов)	✔	✔	✔
1.3	Скриншоты для всех трёх уровней	—	✔	✔
1.4	Таблица тестирования	—	✔	✔
1.5	Блок-схема алгоритма	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
1.6	UML-диаграмма класса	—	—	✔
1.7	Схема порядка вызовов конструкторов/деструкторов	—	—	✔
1.8	Ответы на контрольные вопросы	—	✔	✔

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) - обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня.

Определить конструктор по умолчанию, параметризованный конструктор и деструктор.

В конструкторах и деструкторе выводить сообщения о вызове.

Создать объекты разными способами для демонстрации вызовов.

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) - дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Определить конструктор копирования.

Продемонстрировать вызов конструктора копирования при передаче/возврате объектов.

Для классов с динамической памятью - реализовать глубокое копирование.

Показать отсутствие утечек памяти.

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

Блок-схему алгоритма

Повышенный уровень (оценка 5) - требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Полная реализация правила трёх:

Конструктор копирования (глубокая копия)

Деструктор (освобождение ресурсов)

(Оператор присваивания будет в ЛР18, но для отлично можно опередить)

Корректное управление динамической памятью:

Отсутствие утечек памяти (проверка через Valgrind или визуально)

Отсутствие double-free

Правильное использование new[]/delete[]

Демонстрация порядка вызовов:

Схема вызовов в отчёте

Объяснение, почему деструкторы вызываются в обратном порядке

В отчёте дополнительно представить:

UML-диаграмму класса

Схему порядка вызовов конструкторов и деструкторов

7.4. Примеры проверки соответствия

Выполненные задания	**Оценка**	**Проверка критериев**
Только базовый вариант №1 (Point с конструкторами и деструктором)	3	✔ Конструкторы, ✔ деструктор, ✔ вывод сообщений. Но: ✘ нет конструктора копирования, ✘ нет динамической памяти
Базовый №1 + средний №1 (Point с конструктором копирования)	4	✔ Плюс конструктор копирования, ✔ демонстрация вызова. Но: ✘ нет динамической памяти, ✘ нет глубокого копирования
Базовый №1 + средний №1 + повышенный №1 (String)	5	✔ Все критерии: ✔ правило трёх, ✔ динамическая память, ✔ глубокая копия, ✔ отсутствие утечек, ✔ UML-диаграмма

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	Влияние	Как исправить
Конструктор возвращает	Ошибка	Убрать void

Ошибка	Влияние	Как исправить
значение (написан void)	компиляции	
Забыт список инициализации для const полей	Ошибка компиляции	Использовать список инициализации
Нет конструктора копирования для класса с динамической памятью	Двойное освобождение (double free)	Определить конструктор копирования
Поверхностная копия вместо глубокой	Double free или утечка	Выделять новую память и копировать данные
Забыт delete[] в деструкторе	Утечка памяти	Добавить delete[] data
Несоответствие new и delete (new[] + delete)	Неопределённое поведение	Использовать delete[]
Деструктор не виртуальный (для классов-наследников)	Утечка памяти	Добавить virtual (будет в ЛР17)

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Что такое конструктор и как он вызывается

Что такое деструктор и когда он вызывается

Разницу между конструктором по умолчанию и параметризованным

(для оценки 4) - что такое конструктор копирования, когда он вызывается

(для оценки 5) - что такое правило трёх и почему оно важно

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Что произойдёт, если в конструкторе копирования написать this = &other;?"	Понимание сути конструктора копирования
"Почему деструктор не может принимать параметры?"	Понимание автоматического вызова
"Как запретить копирование объектов?"	Понимание = delete
"Почему конструктор копирования принимает const&, а не просто &?"	Понимание необходимости копирования константных объектов

Лабораторная работа №15

Инкапсуляция. Геттеры и сеттеры

1. Цель работы

Реализовать принцип инкапсуляции в C++: скрыть внутренние данные класса с помощью спецификатора private и предоставить контролируемый доступ к ним через открытые методы - геттеры (getters) и сеттеры (setters). Научиться добавлять валидацию в сеттеры для защиты состояния объекта, реализовать класс с полной инкапсуляцией (например, BankAccount с закрытым полем balance и методами deposit(), withdraw(), getBalance()).

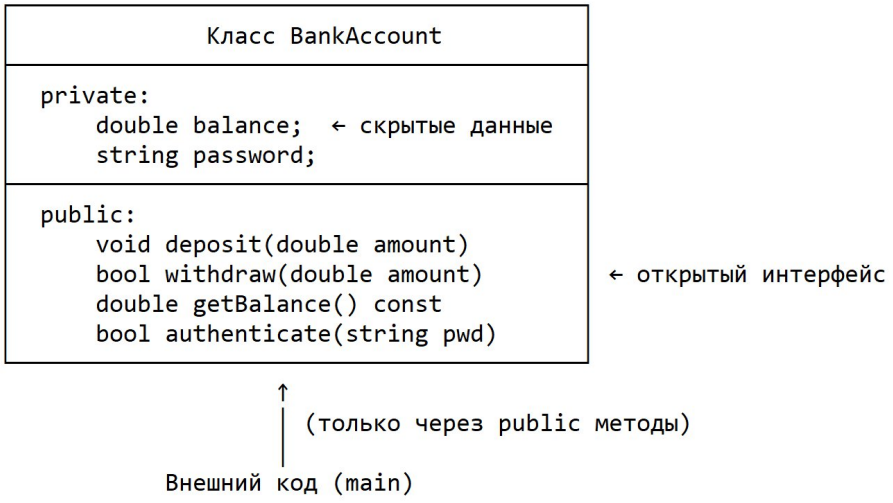
2. Теоретическое обоснование

2.1. Принцип инкапсуляции

Инкапсуляция - это один из фундаментальных принципов объектно-ориентированного программирования, который объединяет данные (поля) и методы, работающие с этими данными, внутри класса, а также скрывает внутреннюю реализацию от внешнего мира.

Основная идея: объект должен быть «чёрным ящиком» - внешний код взаимодействует с ним только через определённый интерфейс, не имея прямого доступа к внутренним данным.

Визуализация инкапсуляции:



Внешний код (main)

Преимущества инкапсуляции:

1. Контроль доступа - можно запретить или ограничить изменение полей
2. Валидация данных - сеттеры могут проверять корректность перед сохранением
3. Сокрытие реализации - внутреннее устройство можно изменить, не меняя интерфейс
4. Поддержка инвариантов - можно гарантировать, что объект всегда находится в корректном состоянии
5. Упрощение отладки - все изменения данных проходят через известные точки входа

2.2. Спецификаторы доступа для инкапсуляции

Спецификатор	Доступ	Применение в инкапсуляции
private	Только внутри класса	Все поля данных должны быть здесь
public	Откуда угодно	Методы интерфейса (геттеры, сеттеры, бизнес-методы)
protected	Внутри класса и наследников	Для полей, которые могут понадобиться наследникам (с осторожностью)

Правильный подход к инкапсуляции:

```
class GoodClass {
private:          // ← все данные скрыты
```

```
    int data;
    string name;

public:          // ← открытый интерфейс
    void setData(int value) { /* валидация */ }
    int getData() const { return data; }
};
```

Неправильный подход (нарушение инкапсуляции):

```
class BadClass {
public:          // ← данные открыты всем!
    int data;
    string name;
};
```

2.3. Геттеры (getters) - методы для чтения данных

Геттер - это открытый метод, который возвращает значение закрытого поля. Обычно имеет имя getПоле() или просто поле().

Синтаксис:

cpp

```
Тип getИмя() const {
    return поле;
}
```

Зачем нужен const? Геттер не должен изменять объект, поэтому его объявляют const. Это позволяет вызывать геттеры для константных объектов.

```
class BankAccount {
private:
    double balance;
    string owner;
```

```
public:
    // Геттеры (только чтение)
    double getBalance() const {    // const - не изменяет объект
        return balance;
    }

    string getOwner() const {
        return owner;
    }
};
```

Преимущества геттеров:

1. Позволяют читать значения закрытых полей
2. Могут возвращать копию (безопасно) или константную ссылку (эффективно)
3. Можно добавить логирование или другую обработку

```
// Возврат по константной ссылке (для больших объектов, без копирования)
const string& getOwner() const {
    return owner;
}

// Возврат по значению (для простых типов)
int getBalance() const {
    return balance;
}
```

2.4. Сеттеры (setters) - методы для изменения данных

Сеттер - это открытый метод, который устанавливает новое значение закрытого поля. Обычно имеет имя setПоле().

Синтаксис:

cpp

```
void setИмя(Тип значение) {
    // валидация (опционально)
    поле = значение;
}
```

cpp

```
class BankAccount {
private:
    double balance;

public:
    // Сеттер с валидацией
    void setBalance(double newBalance) {
        if (newBalance < 0) {
            std::cout << "Ошибка: баланс не может быть отрицательным!" << std::endl;
            return;  // или balance = 0;
        }
        balance = newBalance;
    }
};
```

Преимущества сеттеров:

1. Валидация входных данных
2. Контроль над изменениями (логирование, уведомления)
3. Поддержка инвариантов класса (например, withdraw не может сделать баланс отрицательным)

4. Возможность изменить внутреннее представление, не меняя интерфейс

2.5. Полная инкапсуляция: класс BankAccount

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

class BankAccount {
private:
    std::string accountNumber;   // номер счёта
    std::string owner;           // владелец
    double balance;              // баланс (должен быть >= 0)
    double interestRate;         // процентная ставка (0-100)
    bool isActive;               // активен ли счёт

public:
    // ========== КОНСТРУКТОРЫ ==========

    BankAccount() : balance(0.0), interestRate(0.0), isActive(true) {}

    BankAccount(const std::string& accountNumber, const std::string& owner, double initialBalance)
        : accountNumber(accountNumber), owner(owner), balance(0.0), interestRate(0.0), isActive(true) {
        deposit(initialBalance);   // используем сеттер для валидации
    }

    // ========== ГЕТТЕРЫ ==========
```

```
    std::string getAccountNumber() const {
        return accountNumber;
    }

    std::string getOwner() const {
        return owner;
    }

    double getBalance() const {
        return balance;
    }

    double getInterestRate() const {
        return interestRate;
    }

    bool isAccountActive() const {
        return isActive;
    }

    // ========== СЕТТЕРЫ С ВАЛИДАЦИЕЙ ==========

    void setInterestRate(double rate) {
        if (rate < 0.0 || rate > 100.0) {
            std::cout << "Ошибка: процентная ставка должна быть от 0 до
100" << std::endl;
            return;
        }
```

```
        interestRate = rate;
        std::cout << "Процентная ставка установлена: " << interestRate << "%" << std::endl;
    }

    void setActive(bool active) {
        isActive = active;
        std::cout << "Счёт " << (active ? "активирован" : "деактивирован") << std::endl;
    }

    // ========== БИЗНЕС-МЕТОДЫ ==========

    // Пополнение счёта
    void deposit(double amount) {
        if (amount <= 0) {
            std::cout << "Ошибка: сумма пополнения должна быть положительной!" << std::endl;
            return;
        }
        if (!isActive) {
            std::cout << "Ошибка: счёт деактивирован!" << std::endl;
            return;
        }
        balance += amount;
        std::cout << "Пополнение на " << amount << ". Новый баланс: " << balance << std::endl;
    }
```

```
    // Снятие со счёта
    bool withdraw(double amount) {
        if (amount <= 0) {
            std::cout << "Ошибка: сумма снятия должна быть положительной!" << std::endl;
            return false;
        }
        if (!isActive) {
            std::cout << "Ошибка: счёт деактивирован!" << std::endl;
            return false;
        }
        if (amount > balance) {
            std::cout << "Ошибка: недостаточно средств! Баланс: " << balance << std::endl;
            return false;
        }
        balance -= amount;
        std::cout << "Снятие " << amount << ". Новый баланс: " << balance << std::endl;
        return true;
    }

    // Начисление процентов
    void applyInterest() {
        if (!isActive) {
            std::cout << "Ошибка: счёт деактивирован!" << std::endl;
            return;
        }
        double interest = balance * interestRate / 100.0;
```

```
            balance += interest;
            std::cout << "Начислены проценты: " << interest << ". Новый баланс: " << balance << std::endl;
        }

        // Вывод информации о счёте
        void printInfo() const {
            std::cout << "\n=== Информация о счёте ===" << std::endl;
            std::cout << "Владелец: " << owner << std::endl;
            std::cout << "Номер счёта: " << accountNumber << std::endl;
            std::cout << "Баланс: " << balance << std::endl;
            std::cout << "Ставка: " << interestRate << "%" << std::endl;
            std::cout << "Статус: " << (isActive ? "Активен" : "Деактивирован") << std::endl;
        }
    };

    int main() {
        BankAccount account("123456789", "Иван Петров", 1000.0);

        account.printInfo();

        account.setInterestRate(5.0);
        account.deposit(500);
        account.withdraw(200);
        account.applyInterest();

        account.printInfo();
```

```
    return 0;
}
```

2.6. Валидация данных в сеттерах

Валидация - это проверка корректности входных данных перед сохранением.

Примеры валидации:

```
class Person {
private:
    string name;
    int age;
    double salary;
    string email;

public:
    void setName(const string& newName) {
        if (newName.empty()) {
            cout << "Ошибка: имя не может быть пустым!" << endl;
            return;
        }
        name = newName;
    }

    void setAge(int newAge) {
        if (newAge < 0 || newAge > 150) {
            cout << "Ошибка: возраст должен быть от 0 до 150!" << endl;
            return;
        }
```

```
        age = newAge;
    }

    void setSalary(double newSalary) {
        if (newSalary < 0) {
            cout << "Ошибка: зарплата не может быть отрицательной!" <<
endl;
            // Можно установить значение по умолчанию
            salary = 0;
            return;
        }
        salary = newSalary;
    }

    void setEmail(const string& newEmail) {
        // Простая проверка наличия '@' и '.'
        if (newEmail.find('@') == string::npos || newEmail.find('.') ==
string::npos) {
            cout << "Ошибка: некорректный email!" << endl;
            return;
        }
        email = newEmail;
    }
};
```

2.7. Константные методы и геттеры

Методы, которые не изменяют состояние объекта, следует объявлять с ключевым словом const.

```
class Point {
private:
    int x, y;

public:
    // Константные методы - не изменяют объект
    int getX() const { return x; }      // можно вызвать для const объекта
    int getY() const { return y; }

    // Неконстантные методы - могут изменять объект
    void setX(int newX) { x = newX; }   // нельзя вызвать для const объекта
    void setY(int newY) { y = newY; }
};

void printPoint(const Point& p) {   // p - константная ссылка
    // p.setX(10);   // ОШИБКА! нельзя вызвать неконстантный метод
    cout << p.getX() << ", " << p.getY();  // OK: getX и getY - const
}
```

2.8. Возврат ссылок из геттеров (осторожно!)

Геттер может возвращать константную ссылку для больших объектов (эффективность) или обычную ссылку (если нужно разрешить изменение).

```
class Person {
private:
    string name;
    vector<int> scores;

public:
```

```
    // Возврат по константной ссылке - безопасно, без копирования
    const string& getName() const {
        return name;
    }

    // Возврат по ссылке - позволяет изменять элемент (опасно, но иногда нужно)
    int& getScore(int index) {
        return scores[index];
    }

    // Константная версия для доступа без изменения
    const int& getScore(int index) const {
        return scores[index];
    }
};
```

Опасность возврата ссылки на поле класса:

```
class BadClass {
private:
    int secret;

public:
    int& getSecret() {      // возвращает ссылку
        return secret;      // ОПАСНО! можно изменить secret напрямую
    }
};

int main() {
```

```
    BadClass obj;
    obj.getSecret() = 100;    // изменение secret через ссылку - нарушение инкапсуляции!
    return 0;
}
```

Как избежать: возвращать по значению или по константной ссылке.

2.9. Инварианты класса

Инвариант - это условие, которое всегда истинно для корректного объекта. Инкапсуляция помогает поддерживать инварианты.

Пример инварианта для BankAccount: баланс никогда не может быть отрицательным.

```
class BankAccount {
private:
    double balance;   // инвариант: balance >= 0

public:
    void deposit(double amount) {
        if (amount > 0) {
            balance += amount;   // инвариант сохраняется
        }
    }

    bool withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;   // инвариант сохраняется
            return true;
        }
```

```
        return false;           // операция отменена, инвариант не нарушен
    }

    // Нет прямого доступа к balance - инвариант защищён!
};
```

2.10. Пример класса с полной инкапсуляцией: Temperature

```
#include <iostream>

class Temperature {
private:
    double celsius;   // внутреннее представление: градусы Цельсия
                      // инвариант: celsius >= -273.15 (абсолютный ноль)

public:
    // Конструктор
    Temperature(double celsius = 0) {
        setCelsius(celsius);
    }

    // Геттеры для разных единиц
    double getCelsius() const {
        return celsius;
    }

    double getFahrenheit() const {
        return celsius * 9.0 / 5.0 + 32;
    }
```

```
    double getKelvin() const {
        return celsius + 273.15;
    }

    // Сеттеры с валидацией
    void setCelsius(double value) {
        if (value < -273.15) {
            std::cout << "Ошибка: температура не может быть ниже абсолютного нуля!" << std::endl;
            celsius = -273.15;  // устанавливаем минимальное допустимое значение
        } else {
            celsius = value;
        }
    }

    void setFahrenheit(double value) {
        double newCelsius = (value - 32) * 5.0 / 9.0;
        setCelsius(newCelsius);  // переиспользуем валидацию
    }

    void setKelvin(double value) {
        if (value < 0) {
            std::cout << "Ошибка: температура в Кельвинах не может быть отрицательной!" << std::endl;
            setCelsius(-273.15);
        } else {
            setCelsius(value - 273.15);
```

```
    }
};

// Бизнес-методы
void print() const {
    std::cout << getCelsius() << "°C = "
              << getFahrenheit() << "°F = "
              << getKelvin() << "K" << std::endl;
}
};
```

2.11. Типичные ошибки при реализации инкапсуляции

Ошибка	Пример	Последствие
Поля объявлены public	class A { public: int x; };	Нарушение инкапсуляции
Геттер возвращает ссылку на поле	int& getX() { return x; }	Можно изменить поле напрямую
Нет валидации в сеттере	void setAge(int a) { age = a; }	Можно установить некорректное значение
Геттер не объявлен const	int getX() { return x; }	Нельзя вызвать для константного объекта

Ошибка	Пример	Последствие
Сеттер не проверяет границы	void setBalance(double b) { balance = b; }	Можно сделать баланс отрицательным
Возврат const char* из геттера	const char* getName() { return name.c_str(); }	Висячий указатель при временном объекте

```
// ОШИБКА: геттер возвращает ссылку
class Point {
    int x;
public:
    int& getX() { return x; }    // нарушает инкапсуляцию!
};

// ПРАВИЛЬНО: возврат по значению
class Point {
    int x;
public:
    int getX() const { return x; }
};
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Класс BankAccount (базовая инкапсуляция)

```
//
==========================================================================
=
// Программа: класс BankAccount с инкапсуляцией
// Цель: демонстрация скрытия поля balance и доступа через методы
//
==========================================================================
=

#include <iostream>
#include <string>

class BankAccount {
private:
    std::string owner;      // владелец счёта
    double balance;         // баланс (инвариант: balance >= 0)

public:
    // Конструктор
    BankAccount(const std::string& owner, double initialBalance = 0) {
        this->owner = owner;
        balance = 0;
        deposit(initialBalance);   // используем deposit для валидации
    }

    // ========== ГЕТТЕРЫ ==========
    std::string getOwner() const {
        return owner;
    }
```

```
    double getBalance() const {
        return balance;
    }

    // ========== БИЗНЕС-МЕТОДЫ ==========

    // Пополнение счёта
    void deposit(double amount) {
        if (amount <= 0) {
            std::cout << "Ошибка: сумма пополнения должна быть положительной!" << std::endl;
            return;
        }
        balance += amount;
        std::cout << "Пополнение на " << amount << ". Текущий баланс: " << balance << std::endl;
    }

    // Снятие со счёта
    bool withdraw(double amount) {
        if (amount <= 0) {
            std::cout << "Ошибка: сумма снятия должна быть положительной!" << std::endl;
            return false;
        }
        if (amount > balance) {
            std::cout << "Ошибка: недостаточно средств! Доступно: " << balance << std::endl;
```

```
            return false;
        }
        balance -= amount;
        std::cout << "Снятие " << amount << ". Текущий баланс: " << balance << std::endl;
        return true;
    }

    // Вывод информации
    void printInfo() const {
        std::cout << "Владелец: " << owner << ", Баланс: " << balance << std::endl;
    }
};

int main() {
    // Создание объекта
    BankAccount account("Иван Петров", 1000);

    // Демонстрация работы
    account.printInfo();

    account.deposit(500);
    account.withdraw(200);
    account.withdraw(2000);  // недостаточно средств

    // Нет прямого доступа к balance!
    // account.balance = 999999;  // ОШИБКА! balance - private
```

```
    std::cout << "\nИтоговый баланс: " << account.getBalance() <<
std::endl;

    return 0;
}
```

3.2. Пример 2: Класс Person с валидацией в сеттерах

```
// ============================================================
// Программа: класс Person с полной инкапсуляцией и валидацией
// Цель: демонстрация сеттеров с проверкой корректности
// ============================================================

#include <iostream>
#include <string>
#include <cctype>  // для isdigit, isalpha

class Person {
private:
    std::string name;
    int age;
    std::string phone;
    std::string email;

    // Приватный метод для проверки имени
```

```
    bool isValidName(const std::string& name) const {
        if (name.empty()) return false;
        for (char c : name) {
            if (!std::isalpha(c) && c != ' ' && c != '-') {
                return false;
            }
        }
        return true;
    }

    // Приватный метод для проверки телефона (простая версия)
    bool isValidPhone(const std::string& phone) const {
        if (phone.length() < 10) return false;
        // Упрощённая проверка: должны быть только цифры, пробелы, скобки, дефисы
        for (char c : phone) {
            if (!std::isdigit(c) && c != ' ' && c != '-' && c != '(' && c != ')') {
                return false;
            }
        }
        return true;
    }

    // Приватный метод для проверки email
    bool isValidEmail(const std::string& email) const {
        size_t atPos = email.find('@');
        size_t dotPos = email.rfind('.');
        return (atPos != std::string::npos &&
                dotPos != std::string::npos &&
```

```
                atPos < dotPos &&
                atPos > 0 &&
                dotPos < email.length() - 1);
        }

    public:
        // Конструктор
        Person(const std::string& name, int age, const std::string& phone, const std::string& email) {
            setName(name);
            setAge(age);
            setPhone(phone);
            setEmail(email);
        }

        // ========== ГЕТТЕРЫ ==========
        std::string getName() const { return name; }
        int getAge() const { return age; }
        std::string getPhone() const { return phone; }
        std::string getEmail() const { return email; }

        // ========== СЕТТЕРЫ С ВАЛИДАЦИЕЙ ==========
        void setName(const std::string& newName) {
            if (!isValidName(newName)) {
                std::cout << "Ошибка: недопустимое имя \"" << newName << "\"!" << std::endl;
                return;
            }
            name = newName;
```

```
        std::cout << "Имя установлено: " << name << std::endl;
    }

    void setAge(int newAge) {
        if (newAge < 0 || newAge > 150) {
            std::cout << "Ошибка: возраст " << newAge << " вне допустимого диапазона (0-150)!" << std::endl;
            return;
        }
        age = newAge;
        std::cout << "Возраст установлен: " << age << std::endl;
    }

    void setPhone(const std::string& newPhone) {
        if (!isValidPhone(newPhone)) {
            std::cout << "Ошибка: недопустимый номер телефона \"" << newPhone << "\"!" << std::endl;
            return;
        }
        phone = newPhone;
        std::cout << "Телефон установлен: " << phone << std::endl;
    }

    void setEmail(const std::string& newEmail) {
        if (!isValidEmail(newEmail)) {
            std::cout << "Ошибка: недопустимый email \"" << newEmail << "\"!" << std::endl;
            return;
        }
```

```
            email = newEmail;
            std::cout << "Email установлен: " << email << std::endl;
        }

        // Вывод информации
        void printInfo() const {
            std::cout << "\n=== Информация о человеке ===" << std::endl;
            std::cout << "Имя: " << name << std::endl;
            std::cout << "Возраст: " << age << std::endl;
            std::cout << "Телефон: " << phone << std::endl;
            std::cout << "Email: " << email << std::endl;
        }
    };

    int main() {
        // Корректные данные
        Person p1("Иван Петров", 25, "+7 (123) 456-78-90", "ivan@example.com");
        p1.printInfo();

        // Некорректные данные - сработает валидация
        std::cout << "\n=== Попытка установить некорректные данные ===" << std::endl;
        p1.setName("Иван123");      // содержит цифры
        p1.setAge(200);             // слишком большой
        p1.setPhone("abc");         // слишком короткий
        p1.setEmail("invalid");     // нет @ и .

        p1.printInfo();             // данные не изменились
```

```
    return 0;
}
```

3.3. Пример 3: Класс Product с защитой инвариантов

```
//
==========================================================================
=
// Программа: класс Product (товар) с инкапсуляцией
// Цель: демонстрация поддержки инвариантов класса
//
==========================================================================
=

#include <iostream>
#include <string>

class Product {
private:
    std::string name;
    double price;      // цена (инвариант: price >= 0)
    int quantity;      // количество (инвариант: quantity >= 0)
    double discount;   // скидка в % (инвариант: 0 <= discount <= 100)

public:
    // Конструктор
```

```
    Product(const std::string& name, double price, int quantity, double
discount = 0) {
        setName(name);
        setPrice(price);
        setQuantity(quantity);
        setDiscount(discount);
    }

    // ========== ГЕТТЕРЫ ==========
    std::string getName() const { return name; }
    double getPrice() const { return price; }
    int getQuantity() const { return quantity; }
    double getDiscount() const { return discount; }

    // Цена со скидкой
    double getDiscountedPrice() const {
        return price * (100 - discount) / 100.0;
    }

    // Общая стоимость (цена * количество)
    double getTotalPrice() const {
        return getDiscountedPrice() * quantity;
    }

    // ========== СЕТТЕРЫ С ВАЛИДАЦИЕЙ ==========
    void setName(const std::string& newName) {
        if (newName.empty()) {
            std::cout << "Ошибка: название товара не может быть пустым!"
<< std::endl;
```

```
            return;
        }
        name = newName;
    }

    void setPrice(double newPrice) {
        if (newPrice < 0) {
            std::cout << "Ошибка: цена не может быть отрицательной!" << std::endl;
            return;
        }
        price = newPrice;
    }

    void setQuantity(int newQuantity) {
        if (newQuantity < 0) {
            std::cout << "Ошибка: количество не может быть отрицательным!" << std::endl;
            return;
        }
        quantity = newQuantity;
    }

    void setDiscount(double newDiscount) {
        if (newDiscount < 0 || newDiscount > 100) {
            std::cout << "Ошибка: скидка должна быть от 0 до 100%" << std::endl;
            return;
        }
```

```
        discount = newDiscount;
    }

    // ========== БИЗНЕС-МЕТОДЫ ==========

    // Добавление товара на склад
    void addStock(int amount) {
        if (amount <= 0) {
            std::cout << "Ошибка: количество для добавления должно быть положительным!" << std::endl;
            return;
        }
        quantity += amount;
        std::cout << "Добавлено " << amount << " шт. Теперь на складе: " << quantity << std::endl;
    }

    // Продажа товара
    bool sell(int amount) {
        if (amount <= 0) {
            std::cout << "Ошибка: количество для продажи должно быть положительным!" << std::endl;
            return false;
        }
        if (amount > quantity) {
            std::cout << "Ошибка: недостаточно товара на складе! Доступно: " << quantity << std::endl;
            return false;
        }
```

```
            quantity -= amount;
            std::cout << "Продано " << amount << " шт. Осталось: " << quantity
<< std::endl;
            return true;
        }

        // Вывод информации
        void printInfo() const {
            std::cout << "\n=== Товар ===" << std::endl;
            std::cout << "Название: " << name << std::endl;
            std::cout << "Цена: " << price << " руб." << std::endl;
            std::cout << "Скидка: " << discount << "%" << std::endl;
            std::cout << "Цена со скидкой: " << getDiscountedPrice() << " руб."
<< std::endl;
            std::cout << "Количество: " << quantity << " шт." << std::endl;
            std::cout << "Общая стоимость: " << getTotalPrice() << " руб." <<
std::endl;
        }
    };

    int main() {
        Product product("Ноутбук", 50000, 10, 10);

        product.printInfo();

        product.addStock(5);
        product.sell(3);
        product.setDiscount(15);
```

```
    product printInfo();

    return 0;
}
```

3.4. Пример 4: Класс Temperature с конвертацией единиц

```
//
==========================================================
=
// Программа: класс Temperature с инкапсуляцией внутреннего
представления
// Цель: демонстрация сокрытия формата хранения и конвертации
//
==========================================================
=

#include <iostream>
#include <iomanip>

class Temperature {
private:
    double celsius;   // внутреннее представление: градусы Цельсия
                      // инвариант: celsius >= -273.15

public:
    // ========== КОНСТРУКТОРЫ ==========
    Temperature() : celsius(0) {}
```

```
    explicit Temperature(double value, char unit = 'C') {
        set(value, unit);
    }

    // ========== СЕТТЕРЫ (с валидацией) ==========
    void setCelsius(double value) {
        if (value < -273.15) {
            std::cout << "Ошибка: температура ниже абсолютного нуля! Установлено -273.15°C" << std::endl;
            celsius = -273.15;
        } else {
            celsius = value;
        }
    }

    void setFahrenheit(double value) {
        double newCelsius = (value - 32) * 5.0 / 9.0;
        setCelsius(newCelsius);
    }

    void setKelvin(double value) {
        if (value < 0) {
            std::cout << "Ошибка: температура в Кельвинах не может быть отрицательной!" << std::endl;
            setCelsius(-273.15);
        } else {
            setCelsius(value - 273.15);
        }
    }
```

```
    void set(double value, char unit) {
        switch (unit) {
            case 'C': case 'c': setCelsius(value); break;
            case 'F': case 'f': setFahrenheit(value); break;
            case 'K': case 'k': setKelvin(value); break;
            default:
                std::cout << "Ошибка: неизвестная единица измерения '" <<
unit << "'" << std::endl;
        }
    }

    // ========== ГЕТТЕРЫ ==========
    double getCelsius() const {
        return celsius;
    }

    double getFahrenheit() const {
        return celsius * 9.0 / 5.0 + 32;
    }

    double getKelvin() const {
        return celsius + 273.15;
    }

    // ========== БИЗНЕС-МЕТОДЫ ==========

    // Сравнение температур
    bool isEqualTo(const Temperature& other) const {
```

```
        const double EPS = 1e-9;
        return std::abs(celsius - other.celsius) < EPS;
    }

    bool isGreaterThan(const Temperature& other) const {
        return celsius > other.celsius;
    }

    bool isLessThan(const Temperature& other) const {
        return celsius < other.celsius;
    }

    // Вывод
    void print() const {
        std::cout << std::fixed << std::setprecision(2);
        std::cout << getCelsius() << "°C = "
                  << getFahrenheit() << "°F = "
                  << getKelvin() << "K" << std::endl;
    }
};

int main() {
    std::cout << "=== Создание температур ===" << std::endl;

    Temperature t1;                 // 0°C
    Temperature t2(100, 'C');       // 100°C
    Temperature t3(212, 'F');       // 212°F (100°C)
    Temperature t4(0, 'K');         // -273.15°C (абсолютный ноль)
```

```
    t1.print();
    t2.print();
    t3.print();
    t4.print();

    std::cout << "\n=== Сравнение ===" << std::endl;
    std::cout << "t2 == t3: " << (t2.isEqualTo(t3) ? "да" : "нет") << std::endl;
    std::cout << "t2 > t1: " << (t2.isGreaterThan(t1) ? "да" : "нет") <<
std::endl;

    std::cout << "\n=== Изменение температуры ===" << std::endl;
    t1.setFahrenheit(32);   // 32°F = 0°C
    t1.print();

    t1.setKelvin(300);      // 300K = 26.85°C
    t1.print();

    return 0;
}
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Создать класс с инкапсуляцией: поля private, геттеры и сеттеры для всех полей, простые бизнес-методы.

	Класс	**Поля**	**Дополнительные методы**
	Bank	balance	deposit(), withdraw(), getBa

	Класс	Поля	Дополнительные методы
	Account		lance()
	Student	name, age, grade	геттеры/сеттеры, printInfo()
	Rectangle	width, height	area(), perimeter(), геттеры/сеттеры
	Circle	radius	area(), circumference(), геттеры/сеттеры
	Book	title, author, year	геттеры/сеттеры, isOld() (год < 1900)
	Car	brand, speed, fuel	accelerate(), brake(), refuel()
	Counter	count, min, max	increment(), decrement(), reset()
	Time	hours, minutes, seconds	addSeconds(), toSeconds(), print()
	Fraction	numerator, denominator	decimal(), reciprocal(), геттеры/сеттеры
0	Triangle	sideA, sideB, sideC	isValid(), perimeter(), area() (формула Герона)

	Класс	**Поля**	**Дополнительные методы**
1	Product	name, price, quantity	totalCost(), sell(), addStock()
2	Point	x, y	distanceTo(), translate(), геттеры/сеттеры
3	Employee	name, position, salary	giveRaise(), printInfo()
4	Laptop	brand, ram, storage	upgradeRam(), printSpecs()
5	Quiz	question, answer, score	checkAnswer(), геттеры/сеттеры

Средний уровень (оценка «хорошо»)

Класс с валидацией в сеттерах и защитой инвариантов.

	Класс	Поля	Валидация	Бизнес-методы
	BankAccount	balance, interestRate	balance >=0, interestRate 0-100	applyInterest(), transferTo()
	Person	name, age, email	имя без	isAdult(), printInfo()

	Класс	Поля	Валидация	Бизнес-методы
			цифр, возраст 0 - 1 5 0 , email с @ и .	
	Temperature	celsius	≥ -273.15	геттеры для °C, °F, K, сравнение
	Product	name, price, discount, quantity	price≥0, discount 0 - 1 0 0 , quantity≥ 0	getDiscountedPrice(), sell()
	Student	name, grades[] (vector)	оценки 0-100	addGrade(), getAverage(), getLetterGrade()
	Clock	hours, minutes, seconds	0-23, 0-59, 0-59	tick(), addTime(), difference()
	Date	day, month, year	корректная дата	nextDay(), daysInMonth(), isLeapYear()

	Класс	Поля	Валидация	Бизнес-методы
			(учёт високосных)	
	BankCard	cardNumber, pin, balance, isBlocked	pin 4 цифры, balance≥0	validatePin(), withdraw(), block()
	PasswordManager	password, attempts	password длина≥8, попытки ≤3	validatePassword(), isLocked()
0	Thermostat	currentTemp, targetTemp, mode	temp ≥ -50, ≤ 50, mode: heat/cool/off	adjust(), getStatus()
1	TrafficLight	color, timer	color: red/yellow/green, timer≥0	change(), getRemainingTime()

	Класс	Поля	Валидация	Бизнес-методы
2	Elevator	currentFloor, targetFloor, maxFloor	floors ≥1, ≤max	move(), call()
3	Smartphone	battery, isOn, apps[]	battery 0-100	charge(), installApp(), use()
4	LibraryBook	title, isBorrowed, dueDate	dueDate не может быть в прошлом	borrow(), return(), isOverdue()
5	GameCharacter	name, health, mana, level	health 0-100, mana 0-100, level 1-99	takeDamage(), heal(), levelUp()

Повышенный уровень (оценка «отлично»)

Класс с полной инкапсуляцией, сложной валидацией, поддержкой инвариантов и дополнительными возможностями.

№	Класс	Задание
1	BankAccount	+ история транзакций (массив), метод getTransactionHistory(),

№	Класс	Задание
		ограничение на количество операций в день
2	VendingMachine	товары (название, цена, количество), деньги, сдача, методы: addMoney(), buy(), refund()
3	ParkingLot	машин (номера), время въезда, тариф, методы: enter(), exit(), calculateFee()
4	ShoppingCart	товары (Product), скидочные купоны, методы: addItem(), removeItem(), getTotal(), applyCoupon()
5	Airplane	места (матрица), классы (бизнес/эконом), методы: bookSeat(), cancelBooking(), getAvailableSeats()
6	HotelReservation	комнаты (тип, цена, доступность), методы: bookRoom(), checkIn(), checkOut(), calculateStayCost()
7	OnlineStore	пользователи (корзина, история заказов), методы: registerUser(), login(), placeOrder(

№	Класс	Задание
		), trackOrder()
8	BankSystem	счета (массив), переводы между счетами, ежедневный лимит, методы: createAccount(), transfer(), getTotalBalance()
9	Library	книги (наличие, читатели), методы: addBook(), borrowBook(), returnBook(), getOverdueBooks()
10	ChatBot	вопросы-ответы (словарь), методы: learn(), respond(), getStats()
11	TaskManager	задачи (название, приоритет, статус, дедлайн), методы: addTask(), completeTask(), getTasksByPriority()
12	InventorySystem	товары (артикул, название, цена, количество), методы: addProduct(), removeProduct(), search(), updateStock()
13	Exam	вопросы (текст, варианты, правильный ответ), методы: addQuestion(), takeExam(), calculateScore(), getFeedback()

№	Класс	Задание
14	Calendar	события (дата, время, название, описание), методы: addEvent(), removeEvent(), getEventsByDate(), getReminders()
15	MusicPlayer	плейлисты (название, треки), методы: createPlaylist(), addTrack(), shuffle(), play(), getQueue()

5. Контрольные вопросы

Раздел А. Основы инкапсуляции

1. Что такое инкапсуляция? Какие преимущества она даёт?
2. Какие спецификаторы доступа используются в C++ для реализации инкапсуляции?
3. Почему поля класса обычно объявляются private, а не public?
4. Что произойдёт, если все поля класса сделать public? Какие проблемы это создаёт?
5. Что такое «интерфейс класса»? Как он связан с инкапсуляцией?

Раздел Б. Геттеры и сеттеры

1. Что такое геттер? Какой у него типичный прототип? Зачем ключевое слово const?
2. Что такое сеттер? Какой у него типичный прототип?
3. Зачем нужны геттеры, если можно просто сделать поле public?
4. Зачем нужны сеттеры, если можно просто сделать поле public?
5. Когда геттер должен возвращать ссылку, а когда значение? Каковы риски возврата ссылки?

Раздел В. Валидация и инварианты

1. Что такое валидация данных? Приведите пример валидации в сеттере.
2. Что такое инвариант класса? Приведите пример инварианта для BankAccount.
3. Как инкапсуляция помогает поддерживать инварианты?
4. Что делать, если сеттер получает некорректное значение? (варианты: игнорировать, установить значение по умолчанию, выбросить исключение)
5. Как обработать некорректные данные в конструкторе?

Раздел Г. Константность

1. Что означает const после объявления метода? Например: int getX() const;
2. Почему геттеры обычно объявляются const?
3. Какой метод нельзя вызвать для константного объекта?
4. Что такое const-корректность (const-correctness)?
5. Как объявить константный объект? Какой синтаксис?

Раздел Д. Анализ кода и ошибки

1. Что нарушает инкапсуляцию в этом коде?

```
class BankAccount {
public:
    double balance;
    void deposit(double amount) { balance += amount; }
};
```

2. Почему этот геттер опасен?

```
class Person {
    string name;
public:
    string& getName() { return name; }
};
```

```
void setAge(int age) { age = age; }
```

3. Почему этот код не скомпилируется?

```
class Point {
    int x;
public:
    int getX() { return x; }
};
int main() {
    const Point p;
    std::cout << p.getX();
}
```

4. Как исправить этот класс для полной инкапсуляции?

```
class Temperature {
public:
    double celsius;
    double getFahrenheit() { return celsius * 9/5 + 32; }
};
```

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (инкапсуляция, геттеры, сеттеры, валидация, инварианты, const-методы)

Ход выполнения работы:

Код программы с подробными комментариями (объяснение инкапсуляции в каждом методе)

Блок-схема алгоритма (для одного из методов)

Диаграмма классов (UML-подобная схема с указанием public/private)

Скриншоты выполнения с демонстрацией валидации

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали об инкапсуляции, как геттеры и сеттеры помогают защитить данные)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №15)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) - правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, все поля private, наличие геттеров и сеттеров для всех полей, простые бизнес-методы).

Оценка «хорошо» (4) - правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего уровня (валидация в сеттерах, проверка инвариантов, защита от некорректных данных, const-методы для геттеров).

Оценка «отлично» (5) - правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (полная инкапсуляция, сложная валидация, поддержка инвариантов, дополнительные бизнес-методы, история операций/транзакций, UML-диаграмма в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с

соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
	Компиляция и запуск			
.1	Программа компилируется без ошибок	✔	✔	✔
.2	Программа успешно запускается	✔	✔	✔
.3	Отсутствие предупреждений компилятора	—	✔	✔
	Инкапсуляция			
.1	Все поля объявлены private	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	Нет прямого доступа к полям извне	✔	✔	✔
.3	Доступ только через public методы	✔	✔	✔
	Геттеры			
.1	Для каждого поля есть геттер	✔	✔	✔
.2	Геттеры объявлены const	—	✔	✔
.3	Геттеры возвращают правильный тип	✔	✔	✔
.4	Геттеры не изменяют объект	—	✔	✔
	Сеттеры			
.1	Для каждого изменяемого поля есть сеттер	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	Сеттеры имеют валидацию (где нужно)	—	✔	✔
.3	Корректная обработка некорректных значений	—	✔	✔
.4	Сеттеры используют this при конфликте имён	—	✔	✔
	Бизнес-методы			
.1	Методы реализуют требуемую логику	✔	✔	✔
.2	Методы корректно изменяют состояние объекта	✔	✔	✔
.3	Методы защищают инварианты	—	✔	✔
.4	Дополнительные методы (история,	—	—	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	статистика)			
	Валидация и инварианты			
.1	Проверка границ значений (диапазон)	—	✔	✔
.2	Проверка типов (email, телефон)	—	✔	✔
.3	Защита от отрицательных значений	—	✔	✔
.4	Сложная валидация (пересекающиеся условия)	—	—	✔
	Обработка ошибок			
.1	Вывод информативных сообщений об ошибках	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	Возврат кода ошибки из методов (bool)	—	✔	✔
.3	Предотвращение некорректных операций	—	✔	✔
	Качество кода			
.1	Правильные отступы и форматирование	✔	✔	✔
.2	Осмысленные имена полей и методов	—	✔	✔
.3	Наличие комментариев (объяснение инкапсуляции)	—	✔	✔
.4	Константная корректность (const везде, где нужно)	—	✔	✔
	Отчёт			

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
0				
0.1	Титульный лист, цель, ход работы	✔	✔	✔
0.2	Скриншоты выполнения (минимум 3 теста)	✔	✔	✔
0.3	Скриншоты для всех трёх уровней	—	✔	✔
0.4	Таблица тестирования	—	✔	✔
0.5	Блок-схема алгоритма	—	✔	✔
0.6	UML-диаграмма класса (public/private)	—	—	✔
0.7	Ответы на контрольные вопросы	—	✔	✔

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) - обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня.

Объявить все поля класса как private.

Реализовать геттеры и сеттеры для всех полей.

Реализовать простые бизнес-методы (deposit, withdraw, area, perimeter и т.д.).

Продемонстрировать, что прямой доступ к полям извне невозможен.

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) - дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Добавить валидацию в сеттеры (проверка диапазонов, форматов).

Обеспечить защиту инвариантов класса.

Объявить геттеры с ключевым словом const.

Корректно обрабатывать некорректный ввод (сообщения об ошибках, возврат bool).

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

Блок-схему алгоритма

Повышенный уровень (оценка 5) - требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Сложная валидация:

Проверка перекрёстных условий (например, дата не может быть в прошлом)

Проверка форматов (email, телефон, пин-код)

Дополнительные бизнес-методы:

История транзакций/операций

Расширенная статистика

Поиск и фильтрация

Поддержка инвариантов:

Все методы гарантируют, что объект остаётся в корректном состоянии

Конструктор также выполняет валидацию

В отчёте дополнительно представить:

UML-диаграмму класса

Описание инвариантов класса

7.4. Примеры проверки соответствия

Выполненные задания	Оценка	Проверка критериев
Только базовый вариант №1 (BankAccount с deposit/withdraw)	3	✔ Поля private, ✔ геттеры/сеттеры, ✔ бизнес-методы. Но: ✘ нет валидации, ✘ геттеры не const
Базовый №1 + средний №1 (BankAccount с валидацией)	4	✔ Плюс валидация (balance≥0), ✔ const-геттеры, ✔ защита инвариантов. Но: ✘ нет истории транзакций
Базовый №1 + средний №1 + повышенный №1 (BankAccount с историей)	5	✔ Все критерии: ✔ история операций, ✔ сложная валидация, ✔ инварианты, ✔

Выполненные задания	Оценка	Проверка критериев
		UML-диаграмма

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	Влияние	Как исправить
Поля объявлены public	Нарушение инкапсуляции	Изменить на private
Геттер возвращает ссылку на поле	Можно изменить поле напрямую	Возвращать по значению или const&
Геттер не объявлен const	Нельзя вызвать для const-объекта	Добавить const
Сеттер не проверяет входные данные	Можно установить некорректное значение	Добавить валидацию
Нет проверки инвариантов в бизнес-методах	Объект может стать некорректным	Добавить проверки
Игнорирование некорректных значений	Программа продолжает с ошибкой	Выводить сообщение или выбрасывать исключение

Ошибка	Влияние	Как исправить
Конфликт имён без this	Поле не изменяется	Использовать this->поле = параметр

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Что такое инкапсуляция и зачем она нужна

Как геттеры и сеттеры реализуют инкапсуляцию

Почему поля должны быть private

(для оценки 4) - как работает валидация в сеттерах

(для оценки 5) - что такое инварианты и как они защищаются

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Как изменить класс, чтобы геттер возвращал константную ссылку?"	Понимание возврата const T&
"Что произойдёт, если сеттер не проверяет отрицательные значения для balance?"	Понимание важности валидации
"Как сделать класс, чтобы его	Понимание = delete

Вопрос	Проверяет
нельзя было копировать?"	
"Почему геттеры должны быть const, а сеттеры - нет?"	Понимание константности

Лабораторная работа №16

Наследование. Базовый и производный классы

1. Цель работы

Изучить механизм наследования в C++ - один из ключевых принципов объектно-ориентированного программирования. Научиться создавать иерархии классов: определять базовый класс (родительский, суперкласс) и производные классы (наследники, подклассы). Освоить переопределение методов, использование конструкторов базового класса в производных, понимать спецификаторы доступа protected и порядок вызова конструкторов/деструкторов при наследовании.

2. Теоретическое обоснование

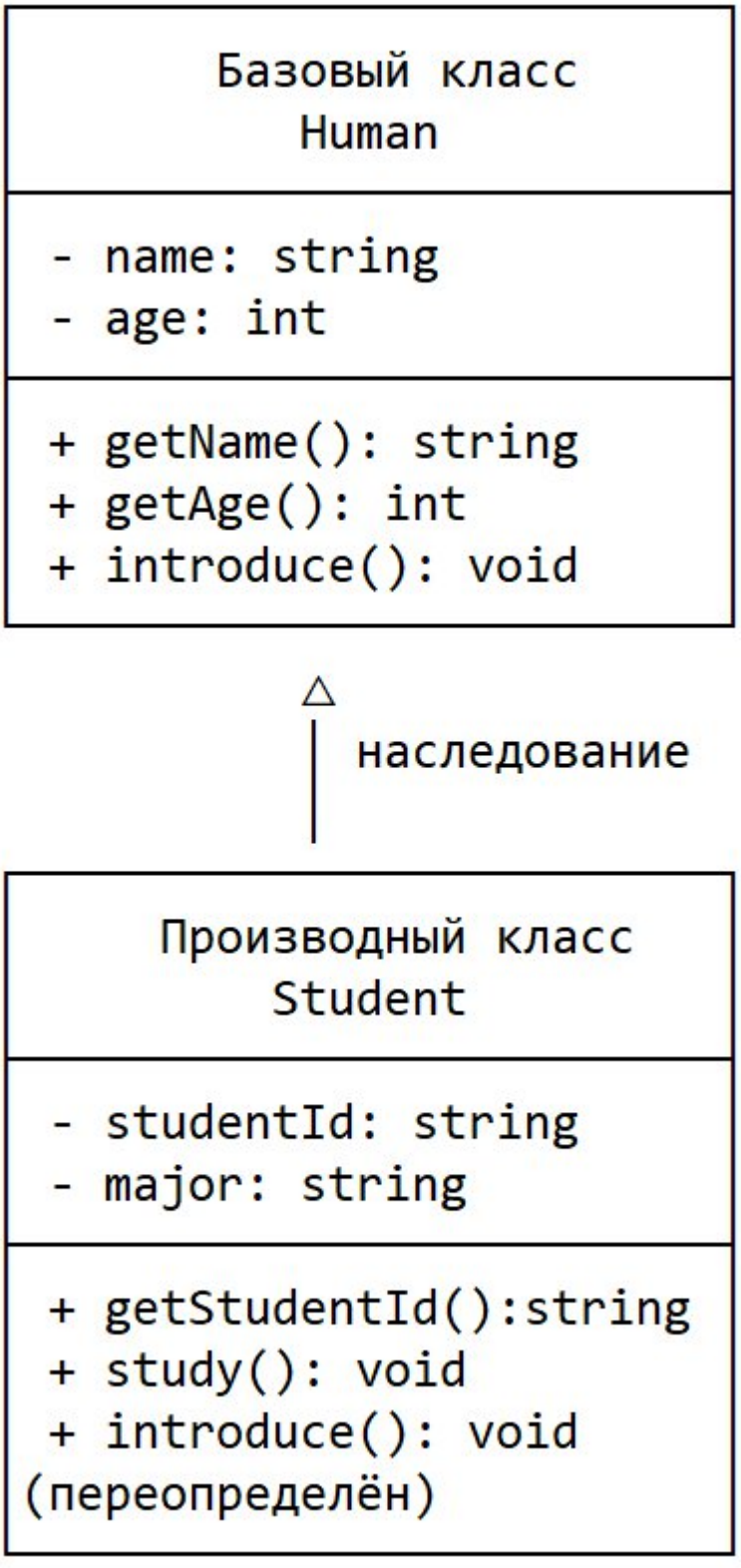
2.1. Понятие наследования

Наследование - это механизм ООП, позволяющий создавать новый класс (производный класс) на основе существующего (базового класса). Производный класс автоматически получает (наследует) все поля и методы базового класса (кроме конструкторов, деструктора и оператора присваивания), а также может:

Добавлять новые поля и методы

Переопределять (изменять поведение) унаследованные методы

Визуализация наследования:



2.2. Синтаксис наследования

```
class БазовыйКласс {
```

```
    // поля и методы
};

class ПроизводныйКласс : спецификатор_доступа БазовыйКласс {
    // дополнительные поля и методы
};
```

Спецификаторы доступа при наследовании:

Спецификатор	Доступ к public членам базового	Доступ к protected членам базового
public	public в производном	protected в производном
protected	protected в производном	protected в производном
private	private в производном	private в производном

Наиболее часто используется public наследование (отношение IS-A).

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

// Базовый класс
class Human {
private:
    string passportNumber;   // доступен только внутри Human
```

```
protected:
    string name;            // доступен в Human и в наследниках
    int age;                // доступен в Human и в наследниках

public:
    Human(const string& name, int age) : name(name), age(age) {
        cout << "Human constructor called" << endl;
    }

    void introduce() {
        cout << "I am " << name << ", " << age << " years old" << endl;
    }

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

// Производный класс (public наследование)
class Student : public Human {
private:
    string studentId;
    string university;

public:
    Student(const string& name, int age, const string& studentId, const
string& university)
        : Human(name, age), studentId(studentId), university(university) {
        cout << "Student constructor called" << endl;
    }
```

```
    void study() {
        cout << name << " is studying at " << university << endl;
    }

    string getStudentId() const { return studentId; }

    // Переопределение метода
    void introduce() {
        cout << "I am student " << name << ", " << age << " years old, ID: " <<
studentId << endl;
    }
};

int main() {
    Human h("Alice", 30);
    h.introduce();

    Student s("Bob", 20, "S12345", "MSU");
    s.introduce();   // переопределённый метод
    s.study();

    // Доступ к protected членам через методы базового класса
    cout << "Age: " << s.getAge() << endl;

    return 0;
}
```

2.3. Спецификатор protected

protected - это третий уровень доступа (после public и private). Члены, объявленные как protected, доступны:

1. Внутри самого класса
2. Внутри производных классов (наследников)
3. Недоступны вне классов (в main и других функциях)

```
class Base {
private:
    int privateVar;     // доступен только внутри Base

protected:
    int protectedVar;   // доступен внутри Base и в наследниках

public:
    int publicVar;      // доступен везде
};

class Derived : public Base {
public:
    void test() {
        // privateVar = 10;   // ОШИБКА! private
        protectedVar = 20;    // OK: protected
        publicVar = 30;       // OK: public
    }
};

int main() {
    Derived d;
    // d.protectedVar = 10;   // ОШИБКА! protected вне классов
```

```
    d publicVar = 30;        // OK
    return 0;
}
```

Рекомендация: Поля делайте private или protected (если предполагаете, что наследникам может понадобиться прямой доступ). Методы делайте public для интерфейса, protected для внутренних вспомогательных методов, доступных наследникам.

2.4. Конструкторы при наследовании

Порядок вызова конструкторов:

1. Сначала вызывается конструктор базового класса
2. Затем вызываются конструкторы полей-объектов (в порядке объявления)
3. Затем вызывается конструктор производного класса

```
#include <iostream>

class Base {
public:
    Base() { cout << "Base constructor" << endl; }
    Base(int x) { cout << "Base constructor with " << x << endl; }
    ~Base() { cout << "Base destructor" << endl; }
};

class Derived : public Base {
public:
    Derived() : Base() { cout << "Derived constructor" << endl; }
    Derived(int x) : Base(x) { cout << "Derived constructor with " << x <<
endl; }
```

```
    ~Derived() { cout << "Derived destructor" << endl; }
};

int main() {
    cout << "Creating Derived d1:" << endl;
    Derived d1;

    cout << "\nCreating Derived d2(42):" << endl;
    Derived d2(42);

    return 0;
}
// Вывод:
// Creating Derived d1:
// Base constructor
// Derived constructor
//
// Creating Derived d2(42):
// Base constructor with 42
// Derived constructor with 42
// Derived destructor
// Base destructor
// Derived destructor
// Base destructor
```

Важно: Если конструктор базового класса требует параметры, их нужно явно передать через список инициализации производного конструктора.

```
class Base {
    int value;
```

```
public:
    Base(int v) : value(v) {}
};

// ОШИБКА: нет конструктора по умолчанию в Base
class Derived1 : public Base {
public:
    Derived1() {}    // ОШИБКА! Base требует параметр
};

// ПРАВИЛЬНО: явный вызов конструктора Base
class Derived2 : public Base {
public:
    Derived2() : Base(0) {}
    Derived2(int x) : Base(x) {}
};
```

2.5. Деструкторы при наследовании

Порядок вызова деструкторов:

1. Сначала вызывается деструктор производного класса
2. Затем деструкторы полей-объектов (в обратном порядке объявления)
3. Затем деструктор базового класса

Важно: Деструктор базового класса должен быть виртуальным (подробнее в ЛР17), чтобы при удалении объекта через указатель на базовый класс корректно вызывался деструктор производного.

```
class Base {
public:
```

```
    ~Base() { cout << "Base destructor" << endl; }
};

class Derived : public Base {
public:
    ~Derived() { cout << "Derived destructor" << endl; }
};

int main() {
    Derived d;
    return 0;
}
// Вывод:
// Derived destructor
// Base destructor
```

2.6. Переопределение методов

Производный класс может переопределить метод базового класса, предоставив свою реализацию.

```
class Animal {
public:
    void speak() {
        cout << "Animal makes a sound" << endl;
    }

    virtual void move() {   // virtual - для полиморфизма (ЛР17)
        cout << "Animal moves" << endl;
    }
```

```
};

class Dog : public Animal {
public:
    // Переопределение (скрытие метода базового класса)
    void speak() {
        cout << "Dog barks: Woof!" << endl;
    }

    void move() override {   // override - явное указание (C++11)
        cout << "Dog runs" << endl;
    }
};

int main() {
    Dog dog;
    dog.speak();    // Dog barks: Woof!
    dog.move();     // Dog runs

    Animal* ptr = &dog;
    ptr->speak();   // Animal makes a sound (если метод не virtual)
    ptr->move();    // Dog runs (если метод virtual)

    return 0;
}
```

Ключевое слово override (C++11): явно указывает, что метод переопределяет виртуальный метод базового класса. Если в базовом классе нет такого виртуального метода - ошибка компиляции.

2.7. Пример: иерархия Human → Student

cpp

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

using namespace std;

// ========== БАЗОВЫЙ КЛАСС Human ==========
class Human {
protected:
    string name;
    int age;
    string gender;

public:
    Human(const string& name, int age, const string& gender)
        : name(name), age(age), gender(gender) {
        cout << "Human created: " << name << endl;
    }

    virtual ~Human() {
        cout << "Human destroyed: " << name << endl;
    }

    void introduce() const {
        cout << "I am " << name << ", " << age << " years old, " << gender <<
endl;
    }
```

```
    void celebrateBirthday() {
        age++;
        cout << name << " is now " << age << " years old!" << endl;
    }

    string getName() const { return name; }
    int getAge() const { return age; }
    string getGender() const { return gender; }

    virtual string getRole() const {
        return "Human";
    }
};

// ========== ПРОИЗВОДНЫЙ КЛАСС Student ==========
class Student : public Human {
private:
    string studentId;
    string university;
    double gpa;
    vector<string> courses;

public:
    Student(const string& name, int age, const string& gender,
            const string& studentId, const string& university)
        : Human(name, age, gender), studentId(studentId),
university(university), gpa(0.0) {
```

```
        cout << "Student created: " << name << " (ID: " << studentId << ")" <<
endl;
    }

    ~Student() {
        cout << "Student destroyed: " << name << endl;
    }

    void study() const {
        cout << name << " is studying at " << university << endl;
    }

    void addCourse(const string& course) {
        courses.push_back(course);
        cout << course << " added to " << name << "'s schedule" << endl;
    }

    void setGPA(double gpa) {
        if (gpa >= 0 && gpa <= 4.0) {
            this->gpa = gpa;
        } else {
            cout << "Invalid GPA!" << endl;
        }
    }

    double getGPA() const { return gpa; }
    string getStudentId() const { return studentId; }

    void printCourses() const {
```

```
        cout << name << "'s courses: ";
        for (const auto& course : courses) {
            cout << course << " ";
        }
        cout << endl;
    }

    // Переопределение метода
    string getRole() const override {
        return "Student";
    }

    void introduce() const {
        Human::introduce();   // вызов метода базового класса
        cout << "I am a student with ID: " << studentId << ", GPA: " << gpa <<
endl;
    }
};

// ========== ПРОИЗВОДНЫЙ КЛАСС Teacher ==========
class Teacher : public Human {
private:
    string teacherId;
    string department;
    vector<string> subjects;

public:
    Teacher(const string& name, int age, const string& gender,
            const string& teacherId, const string& department)
```

```
    : Human(name, age, gender), teacherId(teacherId),
department(department) {
        cout << "Teacher created: " << name << " (ID: " << teacherId << ")" <<
endl;
    }

    ~Teacher() {
        cout << "Teacher destroyed: " << name << endl;
    }

    void teach() const {
        cout << name << " teaches at " << department << " department" <<
endl;
    }

    void addSubject(const string& subject) {
        subjects.push_back(subject);
    }

    string getRole() const override {
        return "Teacher";
    }

    void introduce() const {
        Human::introduce();
        cout << "I am a teacher in " << department << " department" << endl;
    }
};
```

```
int main() {
    cout << "=== Creating objects ===" << endl;
    Student s("Alice", 20, "Female", "S12345", "MSU");
    Teacher t("Dr. Smith", 45, "Male", "T789", "Computer Science");

    cout << "\n=== Student actions ===" << endl;
    s.introduce();
    s.study();
    s.addCourse("Mathematics");
    s.addCourse("Programming");
    s.setGPA(3.8);
    s.printCourses();
    cout << "Role: " << s.getRole() << endl;

    cout << "\n=== Teacher actions ===" << endl;
    t.introduce();
    t.teach();
    cout << "Role: " << t.getRole() << endl;

    cout << "\n=== Polymorphism demo (will be covered in Lab 17) ===" <<
endl;
    Human* humans[] = { &s, &t };
    for (Human* h : humans) {
        cout << h->getName() << " is a " << h->getRole() << endl;
    }

    cout << "\n=== Destroying objects ===" << endl;
    return 0;
}
```

2.8. Множественное наследование

C++ поддерживает множественное наследование - класс может наследовать от нескольких базовых классов.

```
class Printable {
public:
    virtual void print() const = 0;   // чисто виртуальный метод
};

class Serializable {
public:
    virtual void serialize() const = 0;
};

class Student : public Human, public Printable, public Serializable {
    // Student наследует от Human, Printable, Serializable
public:
    void print() const override {
        cout << "Student: " << name << endl;
    }

    void serialize() const override {
        cout << "Serializing student data..." << endl;
    }
};
```

2.9. Типичные ошибки при наследовании

Ошибка	Пример	Последствие
Забыт вызов конструктора базового класса	Derived() {} если Base требует параметры	Ошибка компиляции
Неправильный спецификатор доступа	class Derived : Base (по умолчанию private)	Неожиданные проблемы с доступом
Переопределение без virtual	Метод не виртуальный	Полиморфизм не работает
Деструктор базового класса не виртуальный	~Base() {}	Утечка памяти при полиморфизме
Доступ к private членам базового класса	name если он private	Ошибка компиляции
Конфликт имён при множественном наследовании	Два базовых класса имеют метод print()	Неоднозначность (алмаз смерти)

```
// ОШИБКА: private наследование по умолчанию
class Derived : Base { };  // эквивалентно class Derived : private Base
```

```
// ПРАВИЛЬНО: явно указываем public
class Derived : public Base { };
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Базовый класс Shape и производный Rectangle

```
//
==========================================================
=
// Программа: иерархия Shape → Rectangle
// Цель: демонстрация базового наследования и переопределения
методов
//
==========================================================
=

#include <iostream>
#include <string>

using namespace std;

// Базовый класс Shape
class Shape {
protected:
    string color;

public:
    Shape(const string& color = "black") : color(color) {
```

```
        cout << "Shape constructor called" << endl;
    }

    virtual ~Shape() {
        cout << "Shape destructor called" << endl;
    }

    virtual double getArea() const {
        return 0;
    }

    virtual double getPerimeter() const {
        return 0;
    }

    virtual void print() const {
        cout << "Shape [color: " << color << "]" << endl;
    }

    string getColor() const { return color; }
    void setColor(const string& newColor) { color = newColor; }
};

// Производный класс Rectangle
class Rectangle : public Shape {
private:
    double width;
    double height;
```

```
public:
    Rectangle(double width, double height, const string& color = "black")
        : Shape(color), width(width), height(height) {
        cout << "Rectangle constructor called: " << width << "x" << height <<
endl;
    }

    ~Rectangle() {
        cout << "Rectangle destructor called" << endl;
    }

    double getArea() const override {
        return width * height;
    }

    double getPerimeter() const override {
        return 2 * (width + height);
    }

    void print() const override {
        cout << "Rectangle [color: " << color
             << ", width: " << width
             << ", height: " << height
             << ", area: " << getArea()
             << ", perimeter: " << getPerimeter() << "]" << endl;
    }

    double getWidth() const { return width; }
    double getHeight() const { return height; }
```

```
};

int main() {
    cout << "=== Creating Rectangle ===" << endl;
    Rectangle rect(10, 5, "red");

    cout << "\n=== Rectangle info ===" << endl;
    rect.print();

    cout << "\n=== Using Shape pointer ===" << endl;
    Shape* shapePtr = &rect;
    shapePtr->print();   // вызовется метод Rectangle (благодаря virtual)

    cout << "\n=== Destroying objects ===" << endl;
    return 0;
}
```

3.2. Пример 2: Иерархия Employee → Manager → Developer

```
//
==========================================================
=
// Программа: иерархия сотрудников
// Цель: демонстрация многоуровневого наследования
//
==========================================================
=

#include <iostream>
```

```cpp
#include <string>
#include <vector>

using namespace std;

class Employee {
protected:
    string name;
    int id;
    double salary;

public:
    Employee(const string& name, int id, double salary)
        : name(name), id(id), salary(salary) {
        cout << "Employee constructor: " << name << endl;
    }

    virtual ~Employee() {
        cout << "Employee destructor: " << name << endl;
    }

    virtual void work() const {
        cout << name << " is working" << endl;
    }

    virtual double calculateBonus() const {
        return salary * 0.05;   // 5% бонус
    }
```

```
    virtual void printInfo() const {
        cout << "Employee: " << name << ", ID: " << id << ", Salary: $" <<
salary << endl;
    }

    string getName() const { return name; }
    int getId() const { return id; }
    double getSalary() const { return salary; }
};

class Manager : public Employee {
protected:
    vector<Employee*> team;
    int teamSize;

public:
    Manager(const string& name, int id, double salary)
        : Employee(name, id, salary), teamSize(0) {
        cout << "Manager constructor: " << name << endl;
    }

    ~Manager() {
        cout << "Manager destructor: " << name << endl;
    }

    void addToTeam(Employee* emp) {
        team.push_back(emp);
        teamSize++;
```

```
            cout << emp->getName() << " added to " << name << "'s team" << endl;
        }

        void work() const override {
            cout << name << " is managing a team of " << teamSize << " people" << endl;
        }

        double calculateBonus() const override {
            return Employee::calculateBonus() + teamSize * 500;
        }

        void printInfo() const override {
            Employee::printInfo();
            cout << "  Role: Manager, Team size: " << teamSize << endl;
        }
    };

    class Developer : public Employee {
    private:
        string programmingLanguage;

    public:
        Developer(const string& name, int id, double salary, const string& language)
            : Employee(name, id, salary), programmingLanguage(language) {
            cout << "Developer constructor: " << name << endl;
        }
```

```
    ~Developer() {
        cout << "Developer destructor: " << name << endl;
    }

    void work() const override {
        cout << name << " is writing code in " << programmingLanguage <<
endl;
    }

    double calculateBonus() const override {
        return Employee::calculateBonus() + 1000;
    }

    void printInfo() const override {
        Employee::printInfo();
        cout << "  Role: Developer, Language: " << programmingLanguage <<
endl;
    }

    void codeReview() const {
        cout << name << " is conducting code review" << endl;
    }
};

int main() {
    cout << "=== Creating employees ===" << endl;
    Developer dev1("Alice", 101, 50000, "C++");
    Developer dev2("Bob", 102, 48000, "Python");
```

```
    Manager mgr("Charlie", 201, 70000);

    cout << "\n=== Building team ===" << endl;
    mgr.addToTeam(&dev1);
    mgr.addToTeam(&dev2);

    cout << "\n=== Work demonstration ===" << endl;
    dev1.work();
    dev2.work();
    mgr.work();

    cout << "\n=== Bonus calculation ===" << endl;
    cout << dev1.getName() << " bonus: $" << dev1.calculateBonus() <<
endl;
    cout << mgr.getName() << " bonus: $" << mgr.calculateBonus() << endl;

    cout << "\n=== Print info ===" << endl;
    dev1.printInfo();
    mgr.printInfo();

    cout << "\n=== Destroying objects ===" << endl;
    return 0;
}
```

3.3. Пример 3: Иерархия Vehicle → Car → ElectricCar

```
//
==========================================================
=
```

```
// Программа: иерархия транспортных средств
// Цель: демонстрация многоуровневого наследования и переопределения
//
================================================================

#include <iostream>
#include <string>

using namespace std;

class Vehicle {
protected:
    string brand;
    int year;
    double speed;

public:
    Vehicle(const string& brand, int year) : brand(brand), year(year), speed(0) {
        cout << "Vehicle created: " << brand << " (" << year << ")" << endl;
    }

    virtual ~Vehicle() {
        cout << "Vehicle destroyed: " << brand << endl;
    }

    virtual void accelerate(double amount) {
```

```
        speed += amount;
        cout << brand << " accelerates to " << speed << " km/h" << endl;
    }

    virtual void brake(double amount) {
        speed -= amount;
        if (speed < 0) speed = 0;
        cout << brand << " slows down to " << speed << " km/h" << endl;
    }

    virtual void displayInfo() const {
        cout << "Vehicle: " << brand << ", Year: " << year << ", Speed: " <<
speed << " km/h" << endl;
    }

    double getSpeed() const { return speed; }
    string getBrand() const { return brand; }
};

class Car : public Vehicle {
protected:
    int doors;
    string fuelType;

public:
    Car(const string& brand, int year, int doors, const string& fuelType)
        : Vehicle(brand, year), doors(doors), fuelType(fuelType) {
        cout << "Car created: " << doors << " doors, " << fuelType << endl;
    }
```

```
    ~Car() {
        cout << "Car destroyed: " << brand << endl;
    }

    void accelerate(double amount) override {
        // Автомобили разгоняются быстрее
        Vehicle::accelerate(amount * 1.5);
    }

    void displayInfo() const override {
        Vehicle::displayInfo();
        cout << "  Type: Car, Doors: " << doors << ", Fuel: " << fuelType << endl;
    }

    void honk() const {
        cout << brand << " honks: Beep-beep!" << endl;
    }
};

class ElectricCar : public Car {
private:
    double batteryCapacity;
    double batteryLevel;

public:
    ElectricCar(const string& brand, int year, int doors, double batteryCapacity)
```

```
    : Car(brand, year, doors, "Electric"), batteryCapacity(batteryCapacity),
batteryLevel(100) {
    cout << "ElectricCar created: " << batteryCapacity << " kWh battery"
<< endl;
}

~ElectricCar() {
    cout << "ElectricCar destroyed: " << brand << endl;
}

void accelerate(double amount) override {
    // Электромобили разгоняются мгновенно
    speed += amount * 2;
    batteryLevel -= amount * 0.5;
    if (batteryLevel < 0) batteryLevel = 0;
    cout << brand << " (electric) accelerates to " << speed << " km/h,
battery: " << batteryLevel << "%" << endl;
}

void charge() {
    batteryLevel = 100;
    cout << brand << " is fully charged!" << endl;
}

void displayInfo() const override {
    Car::displayInfo();
    cout << "  Type: Electric, Battery: " << batteryCapacity << " kWh,
Level: " << batteryLevel << "%" << endl;
}
```

```
};

int main() {
    cout << "=== Creating vehicles ===" << endl;
    Car toyota("Toyota", 2020, 4, "Petrol");
    ElectricCar tesla("Tesla", 2022, 4, 75);

    cout << "\n=== Driving Toyota ===" << endl;
    toyota.accelerate(20);
    toyota.accelerate(10);
    toyota.brake(15);
    toyota.honk();

    cout << "\n=== Driving Tesla ===" << endl;
    tesla.accelerate(20);
    tesla.accelerate(15);
    tesla.brake(10);
    tesla.charge();

    cout << "\n=== Vehicle info ===" << endl;
    toyota.displayInfo();
    tesla.displayInfo();

    cout << "\n=== Destroying vehicles ===" << endl;
    return 0;
}
```

3.4. Пример 4: Иерархия с множественным наследованием

```
//
==========================================================================
=
// Программа: множественное наследование
// Цель: демонстрация наследования от нескольких базовых классов
//
==========================================================================
=

#include <iostream>
#include <string>

using namespace std;

// Базовый класс 1
class Person {
protected:
    string name;
    int age;

public:
    Person(const string& name, int age) : name(name), age(age) {
        cout << "Person constructor: " << name << endl;
    }

    virtual ~Person() {
        cout << "Person destructor: " << name << endl
    }
```

```
    virtual void introduce() const {
        cout << "My name is " << name << ", I am " << age << " years old." << endl;
    }
};

// Базовый класс 2
class Worker {
protected:
    string jobTitle;
    double salary;

public:
    Worker(const string& jobTitle, double salary) : jobTitle(jobTitle), salary(salary) {
        cout << "Worker constructor: " << jobTitle << endl;
    }

    virtual ~Worker() {
        cout << "Worker destructor: " << jobTitle << endl;
    }

    virtual void work() const {
        cout << "Working as " << jobTitle << " earning $" << salary << endl;
    }
};

// Базовый класс 3
class Learner {
```

```
protected:
    string institution;
    double gpa;

public:
    Learner(const string& institution, double gpa) : institution(institution),
gpa(gpa) {
        cout << "Learner constructor: " << institution << endl;
    }

    virtual ~Learner() {
        cout << "Learner destructor: " << institution << endl;
    }

    virtual void learn() const {
        cout << "Studying at " << institution << " with GPA " << gpa << endl;
    }
};

// Производный класс (множественное наследование)
class WorkingStudent : public Person, public Worker, public Learner {
private:
    int hoursPerWeek;

public:
    WorkingStudent(const string& name, int age,
                   const string& jobTitle, double salary,
                   const string& institution, double gpa,
                   int hoursPerWeek)
```

```
    : Person(name, age), Worker(jobTitle, salary), Learner(institution, gpa),
hoursPerWeek(hoursPerWeek) {
        cout << "WorkingStudent constructor: " << name << " works " <<
hoursPerWeek << "h/week" << endl;
    }

    ~WorkingStudent() {
        cout << "WorkingStudent destructor: " << name << endl;
    }

    void introduce() const override {
        Person::introduce();
        cout << "I work as " << jobTitle << " and study at " << institution <<
endl;
    }

    void work() const override {
        cout << name << " works " << hoursPerWeek << " hours per week as "
<< jobTitle << endl;
    }

    void learn() const override {
        cout << name << " studies at " << institution << " (GPA: " << gpa <<
")" << endl;
    }

    void dailyRoutine() const {
        cout << "\n--- " << name << "'s daily routine ---" << endl;
        work();
```

```
        learn();
        cout << "Balancing work and study!" << endl;
    }
};

int main() {
    cout << "=== Creating WorkingStudent ===" << endl;
    WorkingStudent ws("Alice Johnson", 22, "Software Developer", 60000,
"MIT", 3.8, 20);

    cout << "\n=== Demonstrating capabilities ===" << endl;
    ws.introduce();
    ws.work();
    ws.learn();
    ws.dailyRoutine();

    cout << "\n=== Using base class pointers ===" << endl;
    Person* p = &ws;
    Worker* w = &ws;
    Learner* l = &ws;

    p->introduce();
    w->work();
    l->learn();

    cout << "\n=== Destroying objects ===" << endl;
    return 0;
}
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Создать простую иерархию из двух классов (базовый + один производный).

	Базовый класс	**Производный класс**	**Дополнительные поля/методы**
	Animal (name, age)	Dog	breed, bark(), fetch()
	Animal (name, age)	Cat	color, meow(), scratch()
	Vehicle (brand, year)	Car	doors, honk(), openTrunk()
	Vehicle (brand, year)	Motorcycle	engineCC, wheelie(), rev()
	Shape (color)	Circle	radius, getArea(), getCircumference()
	Shape (color)	Square	side, getArea(), getPerimeter()
	Employee (name, salary)	Manager	teamSize, hire(), fire()
	Employee (name, salary)	Developer	language, code(), debug()

	Базовый класс	**Производный класс**	**Дополнительные поля/методы**
	Person (name, age)	Student	studentId, study(), takeExam()
0	Person (name, age)	Teacher	subject, teach(), grade()
1	BankAccount (balance)	SavingsAccount	interestRate, applyInterest()
2	BankAccount (balance)	CheckingAccount	overdraftLimit, writeCheck()
3	Product (name, price)	Electronics	warrantyMonths, getWarrantyStatus()
4	Product (name, price)	Clothing	size, material, getSize()
5	Media (title, year)	Book	author, pages, getReadingTime()

Средний уровень (оценка «хорошо»)

Иерархия из трёх классов (базовый + два производных), переопределение методов.

	Базовый класс	Производные	Особенности
	Employe	Manager, Develop	calculateBonus() пер

	Базовый класс	Производные	Особенности
	e	er	еопределён, work() виртуальный
	Vehicle	Car, Truck	calculateFuelEfficiency(), loadCapacity для грузовика
	Shape	Rectangle, Triangle	getArea(), getPerimeter() переопределены
	Person	Student, Professor	introduce() переопределён, getRole()
	BankAccount	SavingsAccount, CreditAccount	withdraw() с проверками, applyMonthlyFee()
	Animal	Mammal, Bird	move(), makeSound() переопределены
	Appliance	WashingMachine, Refrigerator	turnOn(), getEnergyConsumption()
	Document	TextDocument, Spreadsheet	open(), save(), getSize()
	User	Admin, RegularUser	getPermissions(), login()

	Базовый класс	Производные	Особенности
0	Payment	CreditCard, PayPal	processPayment(), getFee()
1	GameCharacter	Warrior, Mage	attack(), takeDamage(), levelUp()
2	Furniture	Chair, Table	getMaterial(), assemble()
3	Subscription	MonthlySubscription, YearlySubscription	getCost(), renew()
4	File	ImageFile, TextFile	open(), getExtension(), getSize()
5	Course	OnlineCourse, InPersonCourse	getDuration(), enroll()

Повышенный уровень (оценка «отлично»)

Многоуровневое наследование (3+ уровня) или множественное наследование, виртуальные методы, полиморфное поведение.

№	Иерархия	Задание
1	Vehicle → Car → ElectricCar	+ hybrid, методы: charge(), fuelUp(), расчёт эффективности

№	Иерархия	Задание
2	Employee → Manager → Director	+ calculateBonus() учитывает 业绩 компании, метод approveBudget()
3	Shape → Polygon → Rectangle/Triangle	+ вычисление площади, периметра, проверка на выпуклость
4	Person → Student → GraduateStudent	+ тема диссертации, научный руководитель, метод defendThesis()
5	Animal → Pet → Dog/Cat	+ методы train(), feed(), play() с полиморфным поведением
6	Media → Book → EBook	+ getFileSize(), download(), getReadingProgress()
7	User → Employee → Manager/Developer	множественное наследование: User + Employee → WorkingUser
8	Component → CPU/GPU/RAM → Computer	композиция + наследование, метод calculatePerformance()
9	PaymentMethod (множественное)	CardPayment + MobilePayment → ContactlessPayment
10	Account → BankAcc	+ история транзакций,

№	Иерархия	Задание
	ount → Saving/Checking	метод transferTo(), generateStatement()
11	Transport → PublicTransport → Bus/Train	+ маршрут, расписание, метод calculateFare()
12	Course → OnlineCourse → MOOCCourse	+ платформа, сертификация, метод getCertificate()
13	GameObject → Character → Player/Enemy	+ инвентарь, опыт, методы attack(), defend(), levelUp()
14	Device → SmartDevice → SmartPhone/SmartTV	+ подключение к интернету, метод updateFirmware()
15	Document → EditableDocument → WordDocument/PDFDocument	+ методы edit(), export(), convert()

5. Контрольные вопросы

Раздел А. Основы наследования

1. Что такое наследование в ООП? Какие преимущества оно даёт?
2. Какой синтаксис наследования в C++? Приведите пример.
3. Что такое базовый класс? Что такое производный класс?
4. Какие спецификаторы доступа можно использовать при наследовании? Что они означают?
5. Что наследуется от базового класса? Что не наследуется?

Раздел Б. Доступ и спецификаторы

1. В чём разница между public, protected и private наследованием?
2. Что такое protected-члены? Когда их следует использовать?
3. Может ли производный класс получить доступ к private членам базового класса? Как обойти это ограничение?
4. Что происходит с областью видимости унаследованных методов при переопределении?
5. Как вызвать метод базового класса из производного (если он переопределён)?

Раздел В. Конструкторы и деструкторы

1. В каком порядке вызываются конструкторы при создании объекта производного класса?
2. Как вызвать конструктор базового класса из конструктора производного?
3. Что произойдёт, если конструктор базового класса требует параметры, а в производном конструкторе они не переданы?
4. В каком порядке вызываются деструкторы при уничтожении объекта производного класса?
5. Почему деструктор базового класса должен быть виртуальным? (будет в ЛР17, но важно)

Раздел Г. Переопределение методов

1. Что такое переопределение метода (override)? Чем оно отличается от перегрузки?
2. Что делает ключевое слово override (C++11)? Зачем оно нужно?
3. Что такое «скрытие имени» (name hiding) при наследовании?
4. Как сохранить доступ к переопределённому методу базового класса?
5. Может ли производный класс переопределить статический метод?

Раздел Д. Анализ кода и ошибки

1. Что выведет код?

```
class A { public: A() { cout << "A"; } };
class B : public A { public: B() { cout << "B"; } };
int main() { B b; return 0; }
```

2. Найдите ошибку:

```
class Base { int x; };
class Derived : public Base { };
int main() { Derived d; d.x = 10; return 0; }
```

3. Что не так?

```
class Animal {
public:
    void speak() { cout << "Animal sound"; }
};
class Dog : public Animal {
public:
    void speak() { cout << "Woof"; }
};
Animal* a = new Dog();
a->speak();   // какой вывод?
```

4. Почему этот код не скомпилируется?

```
class Base { int data; public: Base(int d) : data(d) {} };
class Derived : public Base { };
```

5. Что такое «алмаз смерти» (diamond problem) при множественном наследовании? Как его решить?

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (наследование, базовый и производный классы, спецификаторы доступа, конструкторы/деструкторы, переопределение методов)

Ход выполнения работы:

Код программы с подробными комментариями (объяснение иерархии, конструкторов, переопределённых методов)

Диаграмма классов (UML-подобная схема иерархии)

Схема вызовов конструкторов/деструкторов

Скриншоты выполнения с демонстрацией работы всех методов

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали о наследовании, как оно помогает переиспользовать код)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №16)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) - правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, правильное наследование, создание объектов производного класса, вызов унаследованных методов).

Оценка «хорошо» (4) - правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего

уровня (иерархия из трёх классов, переопределение методов, вызов конструкторов базового класса, использование protected).

Оценка «отлично» (5) - правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (многоуровневое или множественное наследование, виртуальные методы, полиморфное поведение, сложная иерархия, UML-диаграмма в отчёте, демонстрация порядка вызовов конструкторов/деструкторов).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
	Компиляция и запуск			
.1	Программа компилируется без ошибок	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	Программа успешно запускается	✔	✔	✔
.3	Отсутствие предупреждений компилятора	—	✔	✔
	Определение классов			
.1	Базовый класс определён корректно	✔	✔	✔
.2	Производный класс определён с правильным синтаксисом наследования	✔	✔	✔
.3	Использование public наследования	✔	✔	✔
.4	Иерархия из 3+ классов	—	—	✔
	Конструкторы и деструкторы			
.1	Конструкторы базового класса вызываются корректно	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.2	Конструкторы производного класса передают параметры базовому	✔	✔	✔
.3	Деструкторы определены (хотя бы с выводом)	✔	✔	✔
.4	Демонстрация порядка вызовов	—	✔	✔
	Наследование членов			
.1	Производный класс имеет доступ к protected членам базового	✔	✔	✔
.2	Производный класс не имеет доступа к private членам базового	✔	✔	✔
.3	Добавлены новые поля/методы в производный класс	✔	✔	✔
	Переопределение методов			

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.1	Методы переопределены в производном классе	—	✔	✔
.2	Использование override (C++11)	—	✔	✔
.3	Вызов переопределённого метода демонстрирует полиморфизм	—	—	✔
	Бизнес-логика			
.1	Методы базового класса работают корректно	✔	✔	✔
.2	Дополнительные методы производного класса работают	✔	✔	✔
.3	Специфическая логика для разных типов (бонусы, эффективность)	—	✔	✔
	Множественное наследование (для повышенного)			
	Корректное	—	—	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
.1	множественное наследование			
.2	Разрешение конфликтов имён	—	—	✔
.3	Вызов конструкторов всех базовых классов	—	—	✔
	Качество кода			
.1	Правильные отступы и форматирование	✔	✔	✔
.2	Осмысленные имена классов и методов	—	✔	✔
.3	Наличие комментариев (объяснение иерархии)	—	✔	✔
.4	Использование virtual деструктора	—	—	✔
0	Отчёт			
0.1	Титульный лист, цель, ход работы	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
0.2	Скриншоты выполнения (минимум 3 теста)	✔	✔	✔
0.3	Скриншоты для всех трёх уровней	—	✔	✔
0.4	Таблица тестирования	—	✔	✔
0.5	Диаграмма классов (UML)	—	✔	✔
0.6	Схема вызовов конструкторов/деструкторов	—	—	✔
0.7	Ответы на контрольные вопросы	—	✔	✔

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) - обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня.

Создать базовый класс и один производный класс с наследованием public.

Добавить в производный класс новые поля и методы.

Продемонстрировать создание объектов производного класса и вызов унаследованных методов.

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) - дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Создать иерархию из трёх классов (базовый + два производных или многоуровневую).

Переопределить методы базового класса в производных.

Корректно вызывать конструкторы базовых классов из производных.

Использовать protected-члены для доступа в наследниках.

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

UML-диаграмму иерархии

Повышенный уровень (оценка 5) - требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Многоуровневое наследование (3+ уровня) или множественное наследование.

Виртуальные методы и демонстрация полиморфного поведения.

Переопределение методов с использованием override.

Демонстрация порядка вызовов конструкторов/деструкторов с выводом сообщений.

В отчёте дополнительно представить:

Схему вызовов конструкторов/деструкторов

Объяснение выбора иерархии

7.4. Примеры проверки соответствия

Выполненные задания	Оценка	Проверка критериев
Только базовый вариант №1 (Animal → Dog)	3	✔ Наследование, ✔ новые поля/методы, ✔ вызов унаследованных методов. Но: ✘ нет переопределения, ✘ нет иерархии из 3 классов
Базовый №1 + средний №1 (Employee → Manager/Developer)	4	✔ Плюс иерархия из 3 классов, ✔ переопределение методов, ✔ вызов конструкторов. Но: ✘ нет множественного наследования, ✘ нет виртуального деструктора
Базовый №1 + средний №1 + повышенный №1 (Vehicle → Car → ElectricCar)	5	✔ Все критерии: ✔ многоуровневое наследование, ✔ переопределение, ✔ полиморфизм, ✔ схема вызовов, ✔ UML-диаграмма

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	Влияние	Как исправить
Забыт public пр и наследовании	Приватное наследование (не IS-A)	Использовать public
Не передан	Ошибка	Явно вызывать

Ошибка	Влияние	Как исправить
параметр конструктору базового класса	компиляции	конструктор базового в списке инициализации
Доступ к private члену базового класса	Ошибка компиляции	Использовать protected или геттеры/сеттеры
Метод переопределён без virtual (в базовом)	Полиморфизм не работает	Добавить virtual в базовый класс
Нет override	Ошибка при изменении сигнатуры	Использовать override
Не виртуальный деструктор в базовом классе	Утечка памяти при полиморфизме	Добавить virtual ~Base()
Алмаз смерти без виртуального наследования	Неоднозначность	Использовать virtual наследование

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Что такое наследование и зачем оно нужно

Как объявить производный класс

Как вызывается конструктор базового класса

(для оценки 4) - разницу между public, protected, private наследованием

(для оценки 5) - что такое множественное наследование и ромбовидная проблема

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Что произойдёт, если в производном классе не вызвать конструктор базового?"	Понимание необходимости вызова
"Почему private члены не наследуются?"	Понимание инкапсуляции
"Как запретить наследование класса?"	Знание final
"Что такое using Base::method в производном классе?"	Понимание восстановления доступа

Лабораторная работа №17

Полиморфизм. Виртуальные методы

1. Цель работы

Изучить принцип полиморфизма в объектно-ориентированном программировании на C++. Научиться использовать виртуальные функции (virtual) для реализации единого интерфейса для объектов разных

классов, переопределять методы в производных классах (override), использовать виртуальные деструкторы для безопасного удаления объектов через указатель на базовый класс, создавать иерархии классов с полиморфным поведением (например, Figure → Circle, Rectangle).

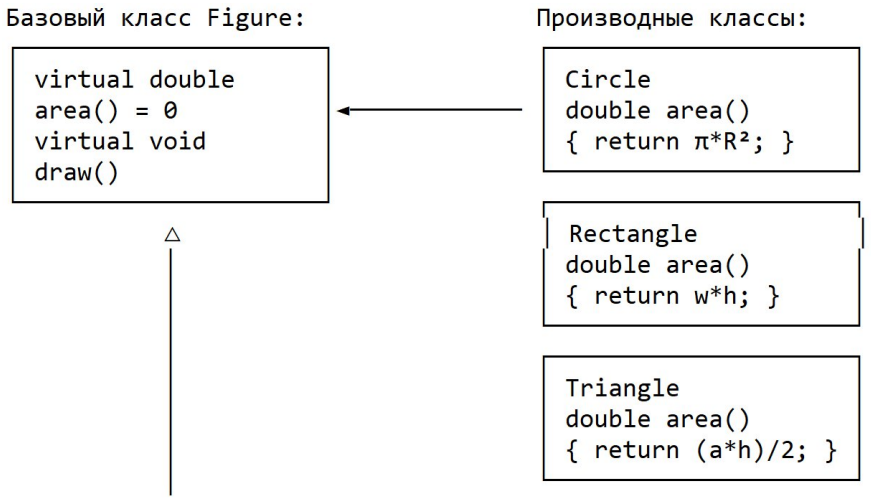
2. Теоретическое обоснование

2.1. Понятие полиморфизма

Полиморфизм (от греч. «много форм») - это один из ключевых принципов ООП, позволяющий объектам разных классов реагировать на одни и те же вызовы методов по-разному, в зависимости от их реального типа.

В C++ полиморфизм реализуется через виртуальные функции (virtual). Когда вызывается виртуальная функция через указатель или ссылку на базовый класс, во время выполнения программы определяется фактический тип объекта и вызывается соответствующая функция производного класса.

Визуализация полиморфизма:



Пример полиморфизма:

```
class Figure {
public:
    virtual double area() const { return 0; }
    virtual void draw() const { cout << "Drawing figure" << endl; }
```

```
    virtual ~Figure() {}
};

class Circle : public Figure {
    double radius;
public:
    Circle(double r) : radius(r) {}
    double area() const override { return 3.14159 * radius * radius; }
    void draw() const override { cout << "Drawing circle" << endl; }
};

int main() {
    // Указатель на базовый класс может указывать на объект производного
    Figure* figures[2];
    figures[0] = new Circle(5);
    figures[1] = new Circle(10);

    for (int i = 0; i < 2; i++) {
        cout << figures[i]->area() << endl;  // вызовется Circle::area()
        figures[i]->draw();                  // вызовется Circle::draw()
    }

    // Очистка
    for (int i = 0; i < 2; i++) delete figures[i];
    return 0;
}
```

2.2. Виртуальные функции (virtual)

Виртуальная функция - это функция-член класса, объявленная с ключевым словом virtual. При вызове виртуальной функции через указатель или ссылку на базовый класс определяется фактический тип объекта и вызывается соответствующая функция.

Синтаксис:

```
class Base {
public:
    virtual void func() { /* реализация по умолчанию */ }
    virtual void pureFunc() = 0;  // чисто виртуальная функция
};
```

Механизм работы:

1. Компилятор создаёт виртуальную таблицу (vtable) для каждого класса, содержащего виртуальные функции.
2. Каждый объект такого класса содержит скрытый указатель на vtable.
3. При вызове виртуальной функции через указатель на базовый класс, используется vtable для определения реальной функции.

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

class Animal {
public:
    virtual void speak() {
        cout << "Animal makes a sound" << endl;
    }
    virtual ~Animal() {}
};
```

```
class Dog : public Animal {
public:
    void speak() override {
        cout << "Dog barks: Woof!" << endl;
    }
};

class Cat : public Animal {
public:
    void speak() override {
        cout << "Cat meows: Meow!" << endl;
    }
};

void makeSound(Animal* animal) {
    animal->speak();   // полиморфный вызов
}

int main() {
    Dog dog;
    Cat cat;

    makeSound(&dog);   // "Dog barks: Woof!"
    makeSound(&cat);   // "Cat meows: Meow!"

    return 0;
}
```

2.3. Чисто виртуальные функции и абстрактные классы

Чисто виртуальная функция объявляется с = 0 в конце. Класс, содержащий хотя бы одну чисто виртуальную функцию, становится абстрактным классом. Создавать объекты абстрактного класса нельзя - он служит только как интерфейс для производных классов.

```
class Figure {
public:
    // Чисто виртуальная функция - делает класс абстрактным
    virtual double area() const = 0;
    virtual double perimeter() const = 0;
    virtual void draw() const = 0;
    virtual ~Figure() {}
};

// Figure f;   // ОШИБКА! нельзя создать объект абстрактного класса

class Circle : public Figure {
    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;
    }
```

```
    void draw() const override {
        cout << "Circle radius=" << radius << endl;
    }
};

int main() {
    Circle c(5);
    Figure* ptr = &c;
    cout << ptr->area() << endl;   // полиморфный вызов
    return 0;
}
```

2.4. Ключевое слово override (C++11)

override явно указывает, что метод переопределяет виртуальный метод базового класса. Если в базовом классе нет виртуального метода с такой сигнатурой - ошибка компиляции.

```
class Base {
public:
    virtual void foo(int x) {}
    virtual void bar() const {}
};

class Derived : public Base {
public:
    // Правильно
    void foo(int x) override {}
```

```
    // ОШИБКА! сигнатура не совпадает (нет const)
    // void bar() override {}

    // ОШИБКА! нет такого метода в Base
    // void baz() override {}
};
```

2.5. Ключевое слово final (C++11)

final запрещает переопределение метода или наследование класса.

```
class Base {
public:
    virtual void foo() {}
    virtual void bar() final {}    // нельзя переопределить в наследниках
};

class Derived : public Base {
public:
    void foo() override {}    // OK
    // void bar() override {}    // ОШИБКА! bar объявлен final
};

class FinalClass final {      // нельзя наследовать от FinalClass
    // ...
};

// class Impossible : public FinalClass {};   // ОШИБКА!
```

2.6. Виртуальный деструктор

Критически важно: Если класс предназначен для наследования, его деструктор должен быть виртуальным. Иначе при удалении объекта производного класса через указатель на базовый класс, деструктор базового класса не вызовет деструктор производного - произойдёт утечка памяти.

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

// БЕЗ виртуального деструктора
class BaseNoVirtual {
public:
    ~BaseNoVirtual() { cout << "BaseNoVirtual destructor" << endl; }
};

class DerivedNoVirtual : public BaseNoVirtual {
    int* data;
public:
    DerivedNoVirtual() { data = new int[100]; }
    ~DerivedNoVirtual() {
        delete[] data;
        cout << "DerivedNoVirtual destructor" << endl;
    }
};

// С виртуальным деструктором
class BaseVirtual {
public:
    virtual ~BaseVirtual() { cout << "BaseVirtual destructor" << endl; }
};
```

```
class DerivedVirtual : public BaseVirtual {
    int* data;
public:
    DerivedVirtual() { data = new int[100]; }
    ~DerivedVirtual() override {
        delete[] data;
        cout << "DerivedVirtual destructor" << endl;
    }
};

int main() {
    cout << "=== Без виртуального деструктора ===" << endl;
    BaseNoVirtual* ptr1 = new DerivedNoVirtual();
    delete ptr1;   // вызовется только BaseNoVirtual destructor - утечка!

    cout << "\n=== С виртуальным деструктором ===" << endl;
    BaseVirtual* ptr2 = new DerivedVirtual();
    delete ptr2;        // вызовется DerivedVirtual, затем BaseVirtual - корректно!

    return 0;
}
```

2.7. Иерархия Figure - полиморфный пример

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

```cpp
using namespace std;

// Абстрактный базовый класс
class Figure {
protected:
    string color;

public:
    Figure(const string& color = "black") : color(color) {}

    virtual ~Figure() {}

    // Чисто виртуальные методы (интерфейс)
    virtual double area() const = 0;
    virtual double perimeter() const = 0;
    virtual void draw() const = 0;

    // Виртуальный метод с реализацией по умолчанию
    virtual void printInfo() const {
        draw();
        cout << "Color: " << color << endl;
        cout << "Area: " << area() << endl;
        cout << "Perimeter: " << perimeter() << endl;
    }

    string getColor() const { return color; }
    void setColor(const string& newColor) { color = newColor; }
};
```

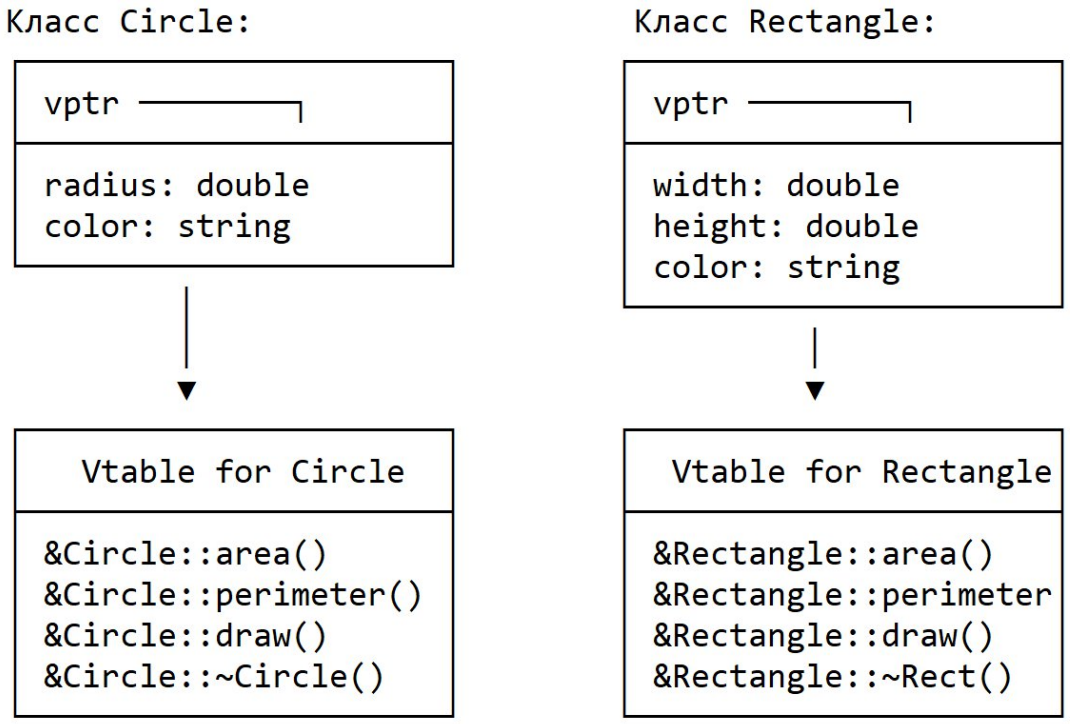
```
class Circle : public Figure {
private:
    double radius;

public:
    Circle(double radius, const string& color = "black")
        : Figure(color), radius(radius) {}

    double area() const override {
        return M_PI * radius * radius;
    }

    double perimeter() const override {
        return 2 * M_PI * radius;
    }

    void draw() const override {
        cout << "Circle with radius " << radius << endl;
    }

    double getRadius() const { return radius; }
    void setRadius(double r) { radius = r; }
};

class Rectangle : public Figure {
private:
    double width;
    double height;
```

```
public:
    Rectangle(double width, double height, const string& color = "black")
        : Figure(color), width(width), height(height) {}

    double area() const override {
        return width * height;
    }

    double perimeter() const override {
        return 2 * (width + height);
    }

    void draw() const override {
        cout << "Rectangle " << width << "x" << height << endl;
    }

    double getWidth() const { return width; }
    double getHeight() const { return height; }
};

class Triangle : public Figure {
private:
    double sideA, sideB, sideC;

public:
    Triangle(double a, double b, double c, const string& color = "black")
        : Figure(color), sideA(a), sideB(b), sideC(c) {}
```

```
    double area() const override {
        double s = (sideA + sideB + sideC) / 2;
        return sqrt(s * (s - sideA) * (s - sideB) * (s - sideC));
    }

    double perimeter() const override {
        return sideA + sideB + sideC;
    }

    void draw() const override {
        cout << "Triangle with sides " << sideA << ", " << sideB << ", " <<
sideC << endl;
    }
};

int main() {
    // Создаём полиморфный массив фигур
    vector<Figure*> figures;

    figures.push_back(new Circle(5, "red"));
    figures.push_back(new Rectangle(4, 6, "blue"));
    figures.push_back(new Triangle(3, 4, 5, "green"));

    // Полиморфный вызов - каждая фигура ведёт себя по-своему
    cout << "=== Figures collection ===" << endl;
    for (Figure* f : figures) {
        f->printInfo();
        cout << "------------------" << endl;
    }
```

```
    // Подсчёт общей площади
    double totalArea = 0;
    for (Figure* f : figures) {
        totalArea += f->area();
    }
    cout << "Total area: " << totalArea << endl;

    // Очистка
    for (Figure* f : figures) {
        delete f;
    }

    return 0;
}
```

2.8. Таблица виртуальных функций (vtable) - как это работает



2.9. Типичные ошибки при работе с полиморфизмом

Ошибка	Пример	Последствие
Не виртуальный деструктор	~Base() {}	Утечка памяти при удалении через указатель на Base
Забыт virtual в базовом классе	void foo() вместо virtual void foo()	Раннее связывание, нет полиморфизма
Несовпадение сигнатуры	void foo(int) в Base, void foo(double) в Derived	Создаётся новая функция (не переопределение)
Вызов виртуальной функции в конструкторе	speak() в конструкторе Animal	Вызывается версия базового класса
Нет override при ошибке сигнатуры	Опечатка в имени	Функция не переопределена, полиморфизм не работает
Хранение объектов по значению	vector<Figure> figures; figures.push_back(Circle(5))	Срезка (slicing) - потеря полиморфизма

```
// ОШИБКА: срезка (slicing) при хранении по значению
vector<Figure> figures;  // вектор базового класса
figures.push_back(Circle(5));  // Circle "срезается" до Figure - потеря данных!
cout << figures[0].area();  // вызывает Figure::area() → 0

// ПРАВИЛЬНО: хранение указателей
vector<Figure*> figures;
figures.push_back(new Circle(5));
cout << figures[0]->area();  // полиморфизм работает!

// ОШИБКА: вызов виртуальной функции в конструкторе
class Animal {
public:
    Animal() { speak(); }  // ОПАСНО! вызовется Animal::speak()
    virtual void speak() { cout << "Animal" << endl; }
};

class Dog : public Animal {
public:
    void speak() override { cout << "Dog" << endl; }
};
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Базовый класс Animal и производные Dog, Cat

```
//
==========================================================
=
// Программа: полиморфная иерархия Animal
```

```
// Цель: демонстрация виртуальных методов и полиморфизма
//
============================================================
=

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

using namespace std;

class Animal {
protected:
    string name;
    int age;

public:
    Animal(const string& name, int age) : name(name), age(age) {}

    virtual ~Animal() {
        cout << "Animal destructor: " << name << endl;
    }

    // Виртуальные методы
    virtual void makeSound() const {
        cout << name << " makes a sound" << endl;
    }

    virtual void move() const {
```

```
        cout << name << " moves" << endl;
    }

    virtual string getType() const {
        return "Animal";
    }

    // Не виртуальный метод
    void introduce() const {
        cout << "I am " << name << ", " << age << " years old" << endl;
    }

    string getName() const { return name; }
    int getAge() const { return age; }
};

class Dog : public Animal {
private:
    string breed;

public:
    Dog(const string& name, int age, const string& breed)
        : Animal(name, age), breed(breed) {}

    ~Dog() {
        cout << "Dog destructor: " << name << endl;
    }

    void makeSound() const override {
```

```
        cout << name << " barks: Woof! Woof!" << endl;
    }

    void move() const override {
        cout << name << " runs happily" << endl;
    }

    string getType() const override {
        return "Dog";
    }

    void fetch() const {
        cout << name << " is fetching the ball!" << endl;
    }

    string getBreed() const { return breed; }
};

class Cat : public Animal {
private:
    string color;

public:
    Cat(const string& name, int age, const string& color)
        : Animal(name, age), color(color) {}

    ~Cat() {
        cout << "Cat destructor: " << name << endl;
    }
```

```
    void makeSound() const override {
        cout << name << " meows: Meow! Meow!" << endl;
    }

    void move() const override {
        cout << name << " jumps gracefully" << endl;
    }

    string getType() const override {
        return "Cat";
    }

    void climb() const {
        cout << name << " is climbing a tree!" << endl
    }

    string getColor() const { return color; }
};

// Функция полиморфного поведения
void interactWithAnimal(Animal* animal) {
    cout << "\n=== Interacting with " << animal->getName() << " ===" <<
endl;
    animal->introduce();
    animal->makeSound();
    animal->move();
    cout << "Type: " << animal->getType() << endl;
}
```

```cpp
int main() {
    cout << "=== Creating animals ===" << endl;
    Dog dog("Rex", 3, "German Shepherd");
    Cat cat("Whiskers", 2, "Orange");

    // Прямой вызов
    cout << "\n=== Direct calls ===" << endl;
    dog.makeSound();
    cat.makeSound();

    // Полиморфные вызовы через указатель на базовый класс
    cout << "\n=== Polymorphic calls via Animal* ===" << endl;
    Animal* ptr = &dog;
    ptr->makeSound();   // вызовется Dog::makeSound()
    ptr = &cat;
    ptr->makeSound();   // вызовется Cat::makeSound()

    // Полиморфная функция
    interactWithAnimal(&dog);
    interactWithAnimal(&cat);

    // Доступ к специфичным методам (нужно привести тип)
    cout << "\n=== Specific methods ===" << endl;
    Dog* dogPtr = dynamic_cast<Dog*>(&dog);
    if (dogPtr) dogPtr->fetch();

    Cat* catPtr = dynamic_cast<Cat*>(&cat);
    if (catPtr) catPtr->climb();
```

```
    cout << "\n=== Destroying objects ===" << endl;
    return 0;
}
```

3.2. Пример 2: Абстрактный класс Figure (полный интерфейс)

```
//
==========================================================
=
// Программа: полиморфная иерархия геометрических фигур
// Цель: демонстрация абстрактного класса и полиморфного поведения
//
==========================================================
=

#include <iostream>
#include <vector>
#include <cmath>
#include <memory>   // для умных указателей

using namespace std;

// Абстрактный базовый класс
class Figure {
public:
    virtual ~Figure() = default;

    // Чисто виртуальные методы - интерфейс
    virtual double area() const = 0;
```

```
    virtual double perimeter() const = 0;
    virtual void draw() const = 0;
    virtual void scale(double factor) = 0;
    virtual Figure* clone() const = 0;

    // Виртуальный метод с реализацией
    virtual void printInfo() const {
        draw();
        cout << "Area: " << area() << endl;
        cout << "Perimeter: " << perimeter() << endl;
    }
};

// Класс Circle
class Circle : public Figure {
private:
    double radius;

public:
    Circle(double radius) : radius(radius) {}

    double area() const override {
        return M_PI * radius * radius;
    }

    double perimeter() const override {
        return 2 * M_PI * radius;
    }
```

```
    void draw() const override {
        cout << "Circle with radius " << radius << endl;
    }

    void scale(double factor) override {
        radius *= factor;
    }

    Figure* clone() const override {
        return new Circle(*this);
    }

    double getRadius() const { return radius; }
    void setRadius(double r) { radius = r; }
};

// Класс Rectangle
class Rectangle : public Figure {
private:
    double width;
    double height;

public:
    Rectangle(double width, double height) : width(width), height(height) {}

    double area() const override {
        return width * height;
    }
```

```
    double perimeter() const override {
        return 2 * (width + height);
    }

    void draw() const override {
        cout << "Rectangle " << width << " x " << height << endl;
    }

    void scale(double factor) override {
        width *= factor;
        height *= factor;
    }

    Figure* clone() const override {
        return new Rectangle(*this);
    }

    double getWidth() const { return width; }
    double getHeight() const { return height; }
};

// Класс Triangle
class Triangle : public Figure {
private:
    double sideA, sideB, sideC;

public:
    Triangle(double a, double b, double c) : sideA(a), sideB(b), sideC(c) {
        // Проверка на существование треугольника
```

```
        if (sideA + sideB <= sideC || sideA + sideC <= sideB || sideB + sideC
<= sideA) {
            throw invalid_argument("Invalid triangle sides");
        }
    }

    double area() const override {
        double s = (sideA + sideB + sideC) / 2;
        return sqrt(s * (s - sideA) * (s - sideB) * (s - sideC));
    }

    double perimeter() const override {
        return sideA + sideB + sideC;
    }

    void draw() const override {
        cout << "Triangle with sides " << sideA << ", " << sideB << ", " <<
sideC << endl;
    }

    void scale(double factor) override {
        sideA *= factor;
        sideB *= factor;
        sideC *= factor;
    }

    Figure* clone() const override {
        return new Triangle(*this);
    }
```

```
};

int main() {
    // Использование умных указателей для автоматического управления памятью
    vector<unique_ptr<Figure>> figures;

    figures.push_back(make_unique<Circle>(5));
    figures.push_back(make_unique<Rectangle>(4, 6));
    figures.push_back(make_unique<Triangle>(3, 4, 5));

    cout << "=== Original figures ===" << endl;
    for (const auto& f : figures) {
        f->printInfo();
        cout << "--------------------" << endl;
    }

    // Масштабирование
    cout << "\n=== After scaling ===" << endl;
    for (auto& f : figures) {
        f->scale(2);
        f->printInfo();
        cout << "--------------------" << endl;
    }

    // Клонирование
    cout << "\n=== Cloned figure ===" << endl;
    unique_ptr<Figure> clone(figures[0]->clone());
    clone->printInfo();
```

```
    // Вычисление общей площади
    double totalArea = 0;
    for (const auto& f : figures) {
        totalArea += f->area();
    }
    cout << "\nTotal area: " << totalArea << endl;

    return 0;
}
```

3.3. Пример 3: Иерархия Shape с разным поведением

```
//
==========================================================
=
// Программа: иерархия Shape с полиморфным поведением
// Цель: демонстрация виртуальных методов и работы с коллекцией
//
==========================================================
=

#include <iostream>
#include <vector>
#include <string>
#include <memory>
#include <algorithm>

using namespace std;

class Shape {
```

```
protected:
    string color;

public:
    Shape(const string& color = "white") : color(color) {}
    virtual ~Shape() = default;

    virtual double area() const = 0;
    virtual double perimeter() const = 0;
    virtual void draw() const = 0;
    virtual void rotate(double angle) = 0;

    string getColor() const { return color; }
    void setColor(const string& c) { color = c; }

    void printInfo() const {
        draw();
        cout << "Color: " << color << endl;
        cout << "Area: " << area() << endl;
        cout << "Perimeter: " << perimeter() << endl;
    }
};

class Square : public Shape {
private:
    double side;

public:
    Square(double side, const string& color = "white")
```

```
    : Shape(color), side(side) {}

    double area() const override {
        return side * side;
    }

    double perimeter() const override {
        return 4 * side;
    }

    void draw() const override {
        cout << "Square side=" << side << endl;
    }

    void rotate(double angle) override {
        cout << "Square rotated by " << angle << " degrees" << endl;
    }
};

class EquilateralTriangle : public Shape {
private:
    double side;

public:
    EquilateralTriangle(double side, const string& color = "white")
        : Shape(color), side(side) {}

    double area() const override {
        return (sqrt(3) / 4) * side * side;
```

```
    }

    double perimeter() const override {
        return 3 * side;
    }

    void draw() const override {
        cout << "Equilateral triangle side=" << side << endl;
    }

    void rotate(double angle) override {
        cout << "Triangle rotated by " << angle << " degrees" << endl;
    }
};

class Ellipse : public Shape {
private:
    double majorAxis;
    double minorAxis;

public:
    Ellipse(double major, double minor, const string& color = "white")
        : Shape(color), majorAxis(major), minorAxis(minor) {}

    double area() const override {
        return M_PI * majorAxis * minorAxis;
    }

    double perimeter() const override {
```

```
        // Приблизительная формула
        return M_PI * (3 * (majorAxis + minorAxis) - sqrt((3 * majorAxis +
minorAxis) * (majorAxis + 3 * minorAxis)));
    }

    void draw() const override {
        cout << "Ellipse axes=" << majorAxis << "x" << minorAxis << endl;
    }

    void rotate(double angle) override {
        cout << "Ellipse rotated by " << angle << " degrees" << endl;
    }
};

int main() {
    // Коллекция фигур с полиморфным поведением
    vector<unique_ptr<Shape>> shapes;

    shapes.push_back(make_unique<Square>(5, "red"));
    shapes.push_back(make_unique<EquilateralTriangle>(4, "blue"));
    shapes.push_back(make_unique<Ellipse>(6, 4, "green"));

    cout << "=== All shapes ===" << endl;
    for (const auto& s : shapes) {
        s->printInfo();
        cout << "--------------------" << endl;
    }

    // Поиск фигуры с максимальной площадью
```

```
    auto maxArea = max_element(shapes.begin(), shapes.end(),
        [](const unique_ptr<Shape>& a, const unique_ptr<Shape>& b) {
            return a->area() < b->area();
        });

    cout << "\n=== Shape with maximum area ===" << endl;
    (*maxArea)->printInfo();

    // Изменение цвета всех фигур
    cout << "\n=== After repainting to yellow ===" << endl;
    for (auto& s : shapes) {
        s->setColor("yellow");
        s->draw();
        cout << "New color: " << s->getColor() << endl;
    }

    return 0;
}
```

3.4. Пример 4: Полиморфизм с умными указателями (C++11)

```
//
==========================================================================
=
// Программа: полиморфизм с использованием smart pointers
// Цель: демонстрация безопасного управления динамической памятью
//
==========================================================================
=
```

```cpp
#include <iostream>
#include <memory>
#include <vector>
#include <typeinfo>

using namespace std;

// Базовый класс
class Product {
protected:
    string name;
    double price;

public:
    Product(const string& name, double price) : name(name), price(price) {}
    virtual ~Product() = default;

    virtual double getPrice() const { return price; }
    virtual string getDescription() const {
        return name + ": $" + to_string(price);
    }
    virtual void printInfo() const {
        cout << getDescription() << endl;
    }

    string getName() const { return name; }
};

// Производный класс 1
```

```
class Electronics : public Product {
private:
    int warrantyMonths;
    string brand;

public:
    Electronics(const string& name, double price, const string& brand, int warranty)
        : Product(name, price), brand(brand), warrantyMonths(warranty) {}

    double getPrice() const override {
        // Электроника может иметь дополнительную наценку
        return price * 1.1;
    }

    string getDescription() const override {
        return Product::getDescription() + ", Brand: " + brand + ", Warranty: " + to_string(warrantyMonths) + " months";
    }

    int getWarranty() const { return warrantyMonths; }
};

// Производный класс 2
class Clothing : public Product {
private:
    string size;
    string material;
```

```
public:
    Clothing(const string& name, double price, const string& size, const string& material)
        : Product(name, price), size(size), material(material) {}

    string getDescription() const override {
        return Product::getDescription() + ", Size: " + size + ", Material: " + material;
    }

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

// Производный класс 3
class Book : public Product {
private:
    string author;
    int pages;

public:
    Book(const string& name, double price, const string& author, int pages)
        : Product(name, price), author(author), pages(pages) {}

    string getDescription() const override {
        return Product::getDescription() + ", Author: " + author + ", Pages: " + to_string(pages);
    }

    string getAuthor() const { return author; }
```

```
};

// Функция для обработки любого продукта (полиморфизм)
void processProduct(Product* product) {
    cout << "Processing: " << product->getDescription() << endl;

    // RTTI (Run-Time Type Information) - определение типа
    if (auto electronics = dynamic_cast<Electronics*>(product)) {
        cout << "  This is electronics with " << electronics->getWarranty() << "
months warranty" << endl;
    } else if (auto clothing = dynamic_cast<Clothing*>(product)) {
        cout << "  This is clothing, size " << clothing->getSize() << endl;
    } else if (auto book = dynamic_cast<Book*>(product)) {
        cout << "  This is a book by " << book->getAuthor() << endl;
    }
}

// Функция расчета общей стоимости корзины
double calculateTotal(const vector<unique_ptr<Product>>& cart) {
    double total = 0;
    for (const auto& item : cart) {
        total += item->getPrice();
    }
    return total;
}

int main() {
    // Создание корзины с использованием умных указателей
    vector<unique_ptr<Product>> cart;
```

```
    cart.push_back(make_unique<Electronics>("Laptop", 1000, "Dell", 24));
    cart.push_back(make_unique<Clothing>("T-Shirt", 25, "M", "Cotton"));
    cart.push_back(make_unique<Book>("C++ Programming", 50, "Bjarne Stroustrup", 1200));
    cart.push_back(make_unique<Electronics>("Smartphone", 800, "Apple", 12));

    cout << "=== Shopping Cart ===" << endl;
    for (const auto& item : cart) {
        processProduct(item.get());
    }

    cout << "\n=== Detailed Info ===" << endl;
    for (const auto& item : cart) {
        item->printInfo();
    }

    cout << "\n=== Total cost: $" << calculateTotal(cart) << endl;

    // Демонстрация RTTI
    cout << "\n=== Type information (RTTI) ===" << endl;
    for (const auto& item : cart) {
        cout << "Type: " << typeid(*item).name() << endl;
    }

    return 0;
}
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Создать иерархию классов с виртуальными методами, продемонстрировать полиморфизм.

№	Базовый класс	Производные	Виртуальные методы
1	Animal	Dog, Cat	speak(), move()
2	Vehicle	Car, Motorcycle	start(), stop(), move()
3	Shape	Circle, Square	area(), draw()
4	Employee	Manager, Developer	work(), getBonus()
5	Instrument	Guitar, Piano	play(), tune()
6	GameCharacter	Warrior, Mage	attack(), defend()
7	Appliance	Washer, Fridge	turnOn(), turnOff()
8	Media	Audio, Video	play(), stop()
9	Document	TextDocument, Spreadsheet	open(), save(), print()
10	Payment	Cash, CreditCard	pay(), refund()
11	Bird	Eagle, Penguin	fly(), eat() (Penguin не умеет летать)

№	Базовый класс	Производные	Виртуальные методы
12	Furniture	Chair, Table	assemble(), getMaterial()
13	Robot	CleaningRobot, CombatRobot	activate(), performTask()
14	Course	OnlineCourse, OfflineCourse	start(), getDuration()
15	Account	SavingAccount, CheckingAccount	deposit(), withdraw(), calculateInterest()

Средний уровень (оценка «хорошо»)

Иерархия с абстрактным базовым классом (чисто виртуальные методы).

№	Абстрактный класс	Производные	Чисто виртуальные методы	Дополнительно
1	Figure	Circle, Rectangle, Triangle	area(), perimeter(), draw()	виртуальный деструктор
2	Transport	Car, Plane, Ship	move(), stop(), refuel()	подсчёт скорости
3	Appliance	Fridge, Oven, Microwave	turnOn(), setTimer(), getPower()	энергопотребление

№	Абстрактный класс	Производные	Чисто виртуальные методы	Дополнительно
4	GameUnit	Infantry, Archer, Cavalry	attack(), defend(), move()	подсчёт урона
5	FileSystem	File, Directory	getSize(), getName(), delete()	рекурсивный подсчёт
6	Expression	Number, AddExpr, MulExpr	evaluate(), toString()	вычисление выражений
7	Sensor	TemperatureSensor, PressureSensor, HumiditySensor	readValue(), getUnit(), calibrate()	мониторинг
8	Drawable	Line, Circle, Rectangle	draw(), move(), getArea()	графический редактор
9	Compression	ZIP, RAR, GZIP	compress(), decompress(), getRatio()	коэффициент сжатия
10	Filter	LowPassFilter, HighPassFilter, BandPassFilter	apply(), getFrequency()	обработка сигнала
11	Sorting	BubbleSort, QuickSort, MergeSo	sort(), getName(), getComplexity(	анализ алгоритмов

№	Абстрактный класс	Производные	Чисто виртуальные методы	Дополнительно
		rt	)	
12	Database	MySQL, PostgreSQL, SQLite	connect(), query(), disconnect()	работа с БД
13	UIElement	Button, TextField, Checkbox	render(), handleClick(), focus()	GUI
14	Parser	JSONParser, XMLParser, CSVParser	parse(), serialize(), getFormat()	обработка данных
15	Authentication	PasswordAuth, BiometricAuth, TwoFactorAuth	authenticate(), getMethod()	безопасность

Повышенный уровень (оценка «отлично»)

Сложная иерархия с полиморфизмом, коллекция объектов, виртуальные деструкторы, умные указатели, RTTI.

№	Задание
	Геометрический пакет: абстрактный класс Shape с методами area(), perimeter(), draw(), scale(). Создать Circle, Rectangle, Triangle, Ellipse. Реализовать функцию вычисления общей площади коллекции фигур. Использовать умные

№	Задание
	указатели.
	Зоопарк: иерархия Animal → Mammal, Bird, Fish. У каждого животного свои методы move(), eat(), makeSound(). Создать вольер (массив животных) и демонстрировать полиморфное поведение.
	Транспортная система: абстрактный Vehicle с методами move(), fuelConsumption(). Производные: Car, Truck, Motorcycle, Bus. Расчёт расхода топлива для каждого, суммарный расход для автопарка.
	Графический редактор: иерархия Drawable с методами draw(), move(), resize(), getBounds(). Реализовать Line, Circle, Rectangle, Text. Список фигур, отрисовка всех.
	Компьютерные игры: иерархия Character с attack(), defend(), useAbility(). Warrior, Mage, Archer, Healer. Боевая система с подсчётом урона, healing, маной.
	Банковская система: Account → SavingsAccount, CheckingAccount, CreditAccount. Полиморфные методы deposit(), withdraw(), calculateInterest(), getMonthlyFee(). История транзакций.
	Система плагинов: абстрактный Plugin с execute(), getName(), getVersion(). Загрузка плагинов из коллекции, динамическое выполнение.

№	Задание
	Компилятор: иерархия ASTNode (узлы абстрактного синтаксического дерева): Number, Variable, BinaryOp, FunctionCall. Метод evaluate(), toString().
	Сеть: NetworkDevice → Router, Switch, Hub. Методы sendPacket(), receivePacket(), getThroughput(). Симуляция передачи данных.
10	Робототехника: Robot → WheeledRobot, LeggedRobot, FlyingRobot. Методы move(), charge(), performTask(). Разное поведение в зависимости от типа.
11	Композиция музыки: MusicalInstrument → StringInstrument, WindInstrument, Percussion Instrument. Метод playNote(), getSound(). Создание оркестра.
12	Криптография: Cipher → CaesarCipher, XORCipher, AESCipher. Методы encrypt(), decrypt(). Полиморфное шифрование.
13	Искусственный интеллект: NeuralLayer → DenseLayer, ConvLayer, PoolingLayer. Метод forward(), backward(). Построение нейросети.
14	Интернет- магазин: Product → Electronics, Clothing, Book, Food. calculatePrice() с учётом скидок и налогов. Корзина с полиморфными товарами.
15	Многопользовательская игра: Player → HumanPlayer, AIPlayer. Методы makeMove(), getScore(), update(). Разное поведение для человека

№	Задание
	и ИИ.

5. Контрольные вопросы

Раздел А. Основы полиморфизма

1. Что такое полиморфизм в ООП? Какие виды полиморфизма существуют в C++?
2. Что такое виртуальная функция (virtual)? Как она работает?
3. Что такое чисто виртуальная функция (= 0)? Что такое абстрактный класс?
4. Можно ли создать объект абстрактного класса? Почему?
5. Что такое виртуальная таблица (vtable) и виртуальный указатель (vptr)?

Раздел Б. Переопределение методов

1. Что делает ключевое слово override? Зачем оно нужно?
2. Что делает ключевое слово final? Где его можно использовать?
3. Чем отличается переопределение от перегрузки?
4. Что произойдёт, если метод объявлен virtual в базовом классе, но не override в производном?
5. Как вызвать версию метода базового класса из переопределённого метода производного?

Раздел В. Виртуальный деструктор

1. Почему деструктор должен быть виртуальным в классах, предназначенных для наследования?
2. Что произойдёт, если удалить объект производного класса через указатель на базовый, а деструктор базового не виртуальный?
3. Можно ли объявить деструктор чисто виртуальным? Зачем?

Раздел Г. Полиморфное поведение

1. Как полиморфизм работает с указателями и ссылками?
2. Что произойдёт при попытке хранить объекты производного класса в контейнере базового класса по значению? (проблема срезки)
3. Что такое RTTI (Run-Time Type Information)? Как получить тип объекта во время выполнения?
4. Что делает оператор dynamic_cast? Когда он возвращает nullptr?

Раздел Д. Анализ кода и ошибки

1. Что выведет код?

```
class A { public: virtual void f() { cout << "A"; } };
class B : public A { public: void f() override { cout << "B"; } };
int main() { A* p = new B(); p->f(); delete p; }
```

2. Почему этот код опасен?

```
class Base { public: ~Base() {} };
class Derived : public Base { int* data; };
Base* p = new Derived();
delete p;
```

3. Найдите ошибку:

```
class Base { virtual void foo() = 0; };
class Derived : public Base { };
int main() { Derived d; }
```

4. Что такое «срезка» (slicing) в контексте полиморфизма? Приведите пример.
5. Почему виртуальные функции не работают в конструкторах и деструкторах?

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (полиморфизм, виртуальные функции, чисто виртуальные функции, виртуальный деструктор, override, final, vtable)

Ход выполнения работы:

Код программы с подробными комментариями (объяснение виртуальных методов, переопределения)

Диаграмма классов (UML-подобная схема иерархии с указанием виртуальных методов)

Схема виртуальных таблиц (vtable) для понимания механизма

Скриншоты выполнения с демонстрацией полиморфного поведения

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали о полиморфизме, как он помогает создавать гибкие программы)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №17)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) - правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, наличие виртуальных методов, демонстрация полиморфизма через указатель на базовый класс).

Оценка «хорошо» (4) - правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего

уровня (абстрактный базовый класс с чисто виртуальными методами, виртуальный деструктор, переопределение методов с override).

Оценка «отлично» (5) - правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (сложная иерархия с полиморфным поведением, коллекция объектов через указатели/умные указатели, предотвращение срезки, виртуальные деструкторы, RTTI с dynamic_cast, UML-диаграмма и схема vtable в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
1	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
2	Компиляция и запуск			
2.1	Программа компилируется без ошибок	✔	✔	✔
2.2	Программа успешно	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	запускается			
2.3	Отсутствие предупреждений компилятора	—	✔	✔
3	Виртуальные функции			
3.1	Методы базового класса объявлены virtual	✔	✔	✔
3.2	Методы производных переопределены	✔	✔	✔
3.3	Использование override (C++ 11)	—	✔	✔
3.4	Чисто виртуальные методы (= 0)	—	✔	✔
4	Абстрактный класс			
4.1	Базовый класс абстрактный (чисто виртуальные методы)	—	✔	✔
4.2	Невозможно создать объект базового класса	—	✔	✔
4.3	Все чисто виртуальные методы переопределены в	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	производных			
5	Виртуальный деструктор			
5.1	Деструктор базового класса виртуальный	—	✔	✔
5.2	Демонстрация корректного удаления через указатель на базовый	—	✔	✔
5.3	Отсутствие утечек памяти	—	✔	✔
6	Полиморфное поведение			
6.1	Вызов виртуальных методов через указатель на базовый класс	✔	✔	✔
6.2	Разное поведение для разных типов объектов	✔	✔	✔
6.3	Использование контейнера (массива/вектора) указателей	—	—	✔
6.4	Предотвращение срезки (slicing)	—	—	✔
7	Управление памятью			

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
7.1	Корректное освобождение динамической памяти	—	✔	✔
7.2	Использование умных указателей (unique_ptr, shared_ptr)	—	—	✔
7.3	Отсутствие delete для умных указателей	—	—	✔
8	RTTI (повышенный уровень)			
8.1	Использование dynamic_cast	—	—	✔
8.2	Использование typeid	—	—	✔
8.3	Проверка успешности приведения	—	—	✔
9	Качество кода			
9.1	Правильные отступы и форматирование	✔	✔	✔
9.2	Осмысленные имена классов и методов	—	✔	✔
9.3	Наличие комментариев (объяснение полиморфизма)	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
9.4	Использование final где уместно	—	—	✔
10	Отчёт			
10.1	Титульный лист, цель, ход работы	✔	✔	✔
10.2	Скриншоты выполнения (минимум 3 теста)	✔	✔	✔
10.3	Скриншоты для всех трёх уровней	—	✔	✔
10.4	Таблица тестирования	—	✔	✔
10.5	UML-диаграмма иерархии	—	✔	✔
10.6	Схема виртуальных таблиц (vtable)	—	—	✔
10.7	Ответы на контрольные вопросы	—	✔	✔

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) - обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня.

Создать базовый класс с виртуальными методами и производные классы.

Переопределить виртуальные методы в производных классах.

Продемонстрировать полиморфизм через указатель на базовый класс.

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) - дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Сделать базовый класс абстрактным (чисто виртуальные методы).

Объявить деструктор базового класса виртуальным.

Использовать ключевое слово override для переопределённых методов.

Продемонстрировать корректное удаление объектов через указатель на базовый класс.

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

UML-диаграмму иерархии

Повышенный уровень (оценка 5) - требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Сложная иерархия с полиморфным поведением:

Минимум 3 производных класса

Разное поведение виртуальных методов

Коллекция объектов:

Массив или вектор указателей на базовый класс

Обход коллекции с вызовом виртуальных методов

Предотвращение срезки:

Хранение указателей, а не объектов

Умные указатели:

Использование std::unique_ptr или std::shared_ptr

Автоматическое управление памятью

RTTI (Run-Time Type Information):

Использование dynamic_cast для проверки типа

Использование typeid для получения информации о типе

В отчёте дополнительно представить:

Схему виртуальных таблиц (vtable)

Объяснение RTTI

7.4. Примеры проверки соответствия

Выполненные задания	**Оценка**	**Проверка критериев**
Только базовый вариант №1 (Animal→Dog/Cat)	3	✔ Виртуальные методы, ✔ переопределение, ✔ полиморфизм. Но: ✘ нет абстрактного класса, ✘ нет виртуального деструктора
Базовый №1 + средний №1 (Figure с чисто виртуальными методами)	4	✔ Плюс абстрактный класс, ✔ чисто виртуальные методы, ✔ виртуальный деструктор. Но: ✘ нет коллекции указателей, ✘ нет умных указателей
Базовый №1 + средний №1 + повышенный №1 (геометрический пакет)	5	✔ Все критерии: ✔ коллекция указателей, ✔ умные указатели, ✔ предотвращение срезки, ✔ RTTI, ✔ vtable-схема

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	Влияние	Как исправить
Забыт virtual в базовом классе	Нет полиморфизма (раннее связывание)	Добавить virtual
Нет override при опечатке	Создаётся новая функция	Использовать override
Не виртуальный деструктор	Утечка памяти	Сделать деструктор виртуальным
Срезка (хранение объектов по значению)	Потеря полиморфизма	Хранить указатели или ссылки
Вызов виртуального метода в конструкторе	Вызывается версия базового класса	Избегать вызовов в конструкторах
dynamic_cast без проверки	nullptr приведёт к краху	Проверять результат if (ptr)
Абстрактный класс без переопределения всех чисто виртуальных методов	Ошибка компиляции	Переопределить все методы

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Что такое полиморфизм и виртуальные функции

Как работает вызов виртуальной функции (vtable)

Зачем нужен виртуальный деструктор

(для оценки 4) - что такое абстрактный класс

(для оценки 5) - как работает RTTI и dynamic_cast

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Что произойдёт при удалении объекта через указатель на базовый класс без виртуального деструктора?"	Понимание виртуального деструктора
"Почему виртуальные функции не могут быть шаблонными?"	Понимание механизма vtable
"Как dynamic_cast определяет тип объекта?"	Понимание RTTI
"Что такое чисто виртуальный деструктор?"	Понимание абстрактных классов

Лабораторная работа №18

Перегрузка операторов. Начальное знакомство

1. Цель работы

Научиться делать классы более интуитивными и удобными в использовании за счёт перегрузки операторов. Освоить синтаксис перегрузки

операторов как методов класса и как дружественных функций. Реализовать перегрузку арифметических операторов (+, -, *, /), операторов сравнения (==, !=, <, >), оператора вывода (<<) и оператора присваивания (=). Создать класс с перегруженными операторами (например, Vector3D с операторами + и <<).

2. Теоретическое обоснование

2.1. Что такое перегрузка операторов?

Перегрузка операторов - это механизм, позволяющий переопределить поведение стандартных операторов C++ (таких как +, -, *, /, ==, <<, [] и др.) для пользовательских типов (классов). Это делает код более естественным и читаемым.

Пример без перегрузки:

```
Vector3D a(1, 2, 3);
Vector3D b(4, 5, 6);
Vector3D c = a.add(b);      // неинтуитивно
```

С перегрузкой:

```
Vector3D a(1, 2, 3);
Vector3D b(4, 5, 6);
Vector3D c = a + b;         // интуитивно понятно
```

Какие операторы можно перегружать?

Арифметические: +, -, *, /, %

Сравнения: ==, !=, <, >, <=, >=

Логические: &&, ||, !

Битовые: &, |, ^, ~, <<, >>

Присваивания: =, +=, -=, *=, /=

Инкремент/декремент: ++, -- (префиксный и постфиксный)

Индексирования: []

Вызова функции: ()

Доступа по указателю: ->

Приведения типов

Нельзя перегружать:

:: (разрешение области видимости)

. (доступ к члену)

.* (доступ к члену через указатель)

?: (тернарный оператор)

sizeof, typeid, alignof, noexcept

2.2. Синтаксис перегрузки операторов

Способ 1: как метод класса (член класса)

```
class Vector3D {
public:
    Vector3D operator+(const Vector3D& other) const {
        return Vector3D(x + other.x, y + other.y, z + other.z);
    }
};
// Использование: v1 + v2 → v1.operator+(v2)
```

Способ 2: как дружественная функция (не член класса)

```
class Vector3D {
    friend Vector3D operator+(const Vector3D& a, const Vector3D& b);
};

Vector3D operator+(const Vector3D& a, const Vector3D& b) {
```

```
    return Vector3D(a.x + b.x, a.y + b.y, a.z + b.z);
}
// Использование: v1 + v2 → operator+(v1, v2)
```

Когда какой способ использовать:

Метод класса	Дружественная функция
Левый операнд - обязательно объект класса	Не требует, чтобы левый операнд был объектом класса
Не работает, если левый операнд - не наш тип	Позволяет перегружать операторы с разными типами
Оператор = должен быть методом класса	Оператор << должен быть дружественной функцией

2.3. Арифметические операторы (+, -, *, /)

```
class Vector3D {
private:
    double x, y, z;

public:
    Vector3D(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}

    // Оператор сложения (метод класса)
    Vector3D operator+(const Vector3D& other) const {
        return Vector3D(x + other.x, y + other.y, z + other.z);
    }
```

```
    // Оператор вычитания
    Vector3D operator-(const Vector3D& other) const {
        return Vector3D(x - other.x, y - other.y, z - other.z);
    }

    // Умножение на скаляр (метод класса)
    Vector3D operator*(double scalar) const {
        return Vector3D(x * scalar, y * scalar, z * scalar);
    }

    // Умножение на скаляр (дружественная функция, чтобы можно было писать scalar * v)
    friend Vector3D operator*(double scalar, const Vector3D& v) {
        return Vector3D(v.x * scalar, v.y * scalar, v.z * scalar);
    }
};
```

2.4. Операторы сравнения (==, !=, <, >)

```
class Vector3D {
    // ...

    bool operator==(const Vector3D& other) const {
        return x == other.x && y == other.y && z == other.z;
    }

    bool operator!=(const Vector3D& other) const {
        return !(*this == other);
    }
```

```
    bool operator<(const Vector3D& other) const {
        double len1 = x*x + y*y + z*z;
        double len2 = other.x*other.x + other.y*other.y + other.z*other.z;
        return len1 < len2;
    }
};
```

2.5. Оператор вывода (<<)

Оператор << не может быть методом класса, потому что левый операнд - std::ostream&, а не наш класс. Поэтому он реализуется как дружественная функция.

```
#include <iostream>

class Vector3D {
private:
    double x, y, z;

public:
    Vector3D(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}

    // Дружественная функция для вывода
    friend std::ostream& operator<<(std::ostream& os, const Vector3D& v);
};

// Реализация оператора вывода
std::ostream& operator<<(std::ostream& os, const Vector3D& v) {
    os << "(" << v.x << ", " << v.y << ", " << v.z << ")";
```

```
    return os;  // возвращаем поток для цепочек вывода
}

int main() {
    Vector3D v(1, 2, 3);
    std::cout << v << std::endl;  // (1, 2, 3)
    return 0;
}
```

Почему возвращаем std::ostream&? Чтобы можно было писать цепочки: cout << v1 << " " << v2 << endl;

2.6. Оператор присваивания (=)

Оператор присваивания копированием уже создаётся компилятором автоматически (поверхностное копирование). Для классов с динамической памятью его нужно переопределять (глубокое копирование) - это часть Правила трёх.

```
class String {
private:
    char* data;
    int length;

public:
    // Конструктор
    String(const char* str = "") {
        length = strlen(str);
        data = new char[length + 1];
        strcpy(data, str);
    }
```

```
    // Конструктор копирования
    String(const String& other) {
        length = other.length;
        data = new char[length + 1];
        strcpy(data, other.data);
    }

    // Оператор присваивания копированием
    String& operator=(const String& other) {
        if (this != &other) {          // проверка на самоприсваивание
            delete[] data;             // освобождаем старую память
            length = other.length;
            data = new char[length + 1];
            strcpy(data, other.data);
        }
        return *this;                  // возвращаем ссылку на текущий объект
    }

    // Деструктор
    ~String() {
        delete[] data;
    }
};
```

Важные моменты:

1. Проверка на самоприсваивание (if (this != &other))
2. Освобождение старой памяти перед выделением новой
3. Возврат *this (ссылки на текущий объект) для цепочек: a = b = c;

2.7. Инкремент и декремент (++, --)

Префиксный и постфиксный инкремент имеют разный синтаксис:

```
class Counter {
private:
    int value;

public:
    Counter(int v = 0) : value(v) {}

    // Префиксный инкремент: ++c
    Counter& operator++() {
        ++value;
        return *this;
    }

    // Постфиксный инкремент: c++
    Counter operator++(int) {   // фиктивный параметр int
        Counter temp = *this;
        ++value;
        return temp;
    }

    // Префиксный декремент: --c
    Counter& operator--() {
        --value;
        return *this;
    }
```

```
    // Постфиксный декремент: c--
    Counter operator--(int) {
        Counter temp = *this;
        --value;
        return temp;
    }

    friend std::ostream& operator<<(std::ostream& os, const Counter& c);
};

std::ostream& operator<<(std::ostream& os, const Counter& c) {
    os << c.value;
    return os;
}

int main() {
    Counter c(5);
    std::cout << ++c << std::endl;  // 6 (префиксный)
    std::cout << c++ << std::endl;  // 6 (постфиксный), затем c = 7
    std::cout << c << std::endl;    // 7
    return 0;
}
```

2.8. Оператор индексирования ([])

Позволяет обращаться к элементам контейнера как к массиву.

```
class IntArray {
private:
    int* data;
```

```
    int size;

public:
    IntArray(int size) : size(size) {
        data = new int[size];
    }

    ~IntArray() {
        delete[] data;
    }

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

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

int main() {
```

```
    IntArray arr(5);
    arr[0] = 10;          // int& operator[]
    std::cout << arr[0];  // 10

    const IntArray& carr = arr;
    std::cout << carr[0]; // const int& operator[] (только чтение)
    // carr[0] = 20;      // ОШИБКА! const-версия возвращает const int&

    return 0;
}
```

2.9. Пример: класс Vector3D с перегрузкой операторов

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

class Vector3D {
private:
    double x, y, z;

public:
    // Конструкторы
    Vector3D(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}

    // Геттеры
    double getX() const { return x; }
    double getY() const { return y; }
    double getZ() const { return z; }
```

```
// Арифметические операторы
Vector3D operator+(const Vector3D& other) const {
    return Vector3D(x + other.x, y + other.y, z + other.z);
}

Vector3D operator-(const Vector3D& other) const {
    return Vector3D(x - other.x, y - other.y, z - other.z);
}

Vector3D operator*(double scalar) const {
    return Vector3D(x * scalar, y * scalar, z * scalar);
}

Vector3D operator/(double scalar) const {
    if (scalar == 0) throw std::invalid_argument("Division by zero");
    return Vector3D(x / scalar, y / scalar, z / scalar);
}

// Дружественные функции для умножения скаляра слева
friend Vector3D operator*(double scalar, const Vector3D& v) {
    return Vector3D(v.x * scalar, v.y * scalar, v.z * scalar);
}

// Составные операторы
Vector3D& operator+=(const Vector3D& other) {
    x += other.x;
    y += other.y;
    z += other.z;
    return *this;
```

```
}

Vector3D& operator-=(const Vector3D& other) {
    x -= other.x;
    y -= other.y;
    z -= other.z;
    return *this;
}

Vector3D& operator*=(double scalar) {
    x *= scalar;
    y *= scalar;
    z *= scalar;
    return *this;
}

// Операторы сравнения
bool operator==(const Vector3D& other) const {
    const double EPS = 1e-9;
    return std::abs(x - other.x) < EPS &&
           std::abs(y - other.y) < EPS &&
           std::abs(z - other.z) < EPS;
}

bool operator!=(const Vector3D& other) const {
    return !(*this == other);
}

// Скалярное произведение (через метод)
```

```
    double dot(const Vector3D& other) const {
        return x * other.x + y * other.y + z * other.z;
    }

    // Длина вектора
    double length() const {
        return std::sqrt(x*x + y*y + z*z);
    }

    // Оператор вывода (дружественная функция)
    friend std::ostream& operator<<(std::ostream& os, const Vector3D& v);
};

// Реализация оператора вывода
std::ostream& operator<<(std::ostream& os, const Vector3D& v) {
    os << "(" << v.x << ", " << v.y << ", " << v.z << ")";
    return os;
}

int main() {
    Vector3D v1(1, 2, 3);
    Vector3D v2(4, 5, 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 << "2 * v1 = " << 2 * v1 << std::endl;
```

```
    std::cout << "Scalar product: " << v1.dot(v2) << std::endl;

    Vector3D v3 = v1;
    v3 += v2;
    std::cout << "v3 = v1; v3 += v2 -> " << v3 << std::endl;

    return 0;
}
```

2.10. Типичные ошибки при перегрузке операторов

Ошибка	Пример	Последствие
Не возвращать ссылку из operator=	void operator=(const T& other)	Нельзя делать a = b = c
Не проверять самоприсваивание	operator= без if (this != &other)	Утечка/повреждение данных
Перегрузка без возврата потока	void operator<<	Нельзя писать cout << a << b
Оператор + не const	Vector3D operator+(Vector3D& other)	Нельзя складывать константные объекты
Не обрабатывать деление на ноль	operator/ без проверки	Аварийное завершение
Постфиксный	Возврат ссылки на	Висячая ссылка

Ошибка	Пример	Последствие
инкремент возвращает не по значению	локальный объект	

```
// ОШИБКА: operator+ должен быть const
class Vector {
public:
    Vector operator+(Vector& other) { ... }   // нельзя с const-объектами
};

// ПРАВИЛЬНО
class Vector {
public:
    Vector operator+(const Vector& other) const { ... }
};

// ОШИБКА: оператор = возвращает void
class Counter {
public:
    void operator=(const Counter& other) { value = other.value; }
};

// ПРАВИЛЬНО
class Counter {
public:
    Counter& operator=(const Counter& other) {
        if (this != &other) value = other.value;
        return *this;
```

```
    }
};
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Класс Vector3D с арифметическими операторами

```
//
==========================================================
=
// Программа: класс Vector3D с перегрузкой операторов +, -, *, /
// Цель: демонстрация арифметических операторов
//
==========================================================
=

#include <iostream>
#include <cmath>

class Vector3D {
private:
    double x, y, z;

public:
    // Конструктор
    Vector3D(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}

    // Геттеры для доступа к координатам
    double getX() const { return x; }
    double getY() const { return y; }
```

```
double getZ() const { return z; }

// Сложение векторов
Vector3D operator+(const Vector3D& other) const {
    return Vector3D(x + other.x, y + other.y, z + other.z);
}

// Вычитание векторов
Vector3D operator-(const Vector3D& other) const {
    return Vector3D(x - other.x, y - other.y, z - other.z);
}

// Умножение на скаляр (вектор * скаляр)
Vector3D operator*(double scalar) const {
    return Vector3D(x * scalar, y * scalar, z * scalar);
}

// Деление на скаляр
Vector3D operator/(double scalar) const {
    if (std::abs(scalar) < 1e-9) {
        throw std::invalid_argument("Division by zero");
    }
    return Vector3D(x / scalar, y / scalar, z / scalar);
}

// Унарный минус
Vector3D operator-() const {
    return Vector3D(-x, -y, -z);
}
```

```
// Дружественная функция: скаляр * вектор
friend Vector3D operator*(double scalar, const Vector3D& v) {
    return Vector3D(v.x * scalar, v.y * scalar, v.z * scalar);
}

// Скалярное произведение
double dot(const Vector3D& other) const {
    return x * other.x + y * other.y + z * other.z;
}

// Векторное произведение
Vector3D cross(const Vector3D& other) const {
    return Vector3D(
        y * other.z - z * other.y,
        z * other.x - x * other.z,
        x * other.y - y * other.x
    );
}

// Длина вектора
double length() const {
    return std::sqrt(x*x + y*y + z*z);
}

// Вывод
friend std::ostream& operator<<(std::ostream& os, const Vector3D& v) {
    os << "(" << v.x << ", " << v.y << ", " << v.z << ")";
    return os;
```

```
    }
};

int main() {
    Vector3D v1(1, 2, 3);
    Vector3D v2(4, 5, 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 = " << -v1 << std::endl;
    std::cout << "v1 * 3 = " << v1 * 3 << std::endl;
    std::cout << "2 * v1 = " << 2 * v1 << std::endl;
    std::cout << "v1 / 2 = " << v1 / 2 << std::endl;
    std::cout << "v1 . v2 = " << v1.dot(v2) << std::endl;
    std::cout << "v1 x v2 = " << v1.cross(v2) << std::endl;
    std::cout << "|v1| = " << v1.length() << std::endl;

    return 0;
}
```

3.2. Пример 2: Класс Complex с перегрузкой операторов

```
//
==========================================================
=
// Программа: класс Complex (комплексное число)
// Цель: демонстрация перегрузки арифметических операторов
```

```
//
==========================================================================
=

#include <iostream>
#include <cmath>

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

public:
    Complex(double re = 0, double im = 0) : re(re), im(im) {}

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

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

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

```
// Умножение: (a+bi)(c+di) = (ac - bd) + (ad + bc)i
Complex operator*(const Complex& other) const {
    return Complex(
        re * other.re - im * other.im,
        re * other.im + im * other.re
    );
}

// Деление: (a+bi)/(c+di) = ((ac+bd)/(c²+d²)) + ((bc-ad)/(c²+d²))i
Complex operator/(const Complex& other) const {
    double denominator = other.re * other.re + other.im * other.im;
    if (std::abs(denominator) < 1e-9) {
        throw std::invalid_argument("Division by zero");
    }
    return Complex(
        (re * other.re + im * other.im) / denominator,
        (im * other.re - re * other.im) / denominator
    );
}

// Составные операторы
Complex& operator+=(const Complex& other) {
    re += other.re;
    im += other.im;
    return *this;
}

Complex& operator-=(const Complex& other) {
    re -= other.re;
```

```
    im -= other.im;
    return *this;
}

Complex& operator*=(const Complex& other) {
    double newRe = re * other.re - im * other.im;
    double newIm = re * other.im + im * other.re;
    re = newRe;
    im = newIm;
    return *this;
}

// Операторы сравнения
bool operator==(const Complex& other) const {
    const double EPS = 1e-9;
    return std::abs(re - other.re) < EPS && std::abs(im - other.im) < EPS;
}

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

// Модуль комплексного числа
double abs() const {
    return std::sqrt(re * re + im * im);
}

// Вывод
friend std::ostream& operator<<(std::ostream& os, const Complex& c) {
```

```
        os << c.re;
        if (c.im >= 0) {
            os << "+" << c.im << "i";
        } else {
            os << c.im << "i";
        }
        return os;
    }

    // Ввод
    friend std::istream& operator>>(std::istream& is, Complex& c) {
        std::cout << "Enter real and imaginary parts: ";
        is >> c.re >> c.im;
        return is;
    }
};

int main() {
    Complex c1(3, 4);
    Complex c2(1, 2);

    std::cout << "c1 = " << c1 << std::endl;
    std::cout << "c2 = " << c2 << std::endl;
    std::cout << "c1 + c2 = " << c1 + c2 << std::endl;
    std::cout << "c1 - c2 = " << c1 - c2 << std::endl;
    std::cout << "c1 * c2 = " << c1 * c2 << std::endl;
    std::cout << "c1 / c2 = " << c1 / c2 << std::endl;
    std::cout << "|c1| = " << c1.abs() << std::endl;
```

```
    Complex c3;
    std::cin >> c3;
    std::cout << "You entered: " << c3 << std::endl;

    return 0;
}
```

3.3. Пример 3: Класс Matrix с операторами +, *, []

```
//
==========================================================
=
// Программа: класс Matrix (матрица) с перегрузкой операторов
// Цель: демонстрация операторов индексирования и сложения матриц
//
==========================================================
=

#include <iostream>
#include <vector>
#include <iomanip>

class Matrix {
private:
    int rows;
    int cols;
    std::vector<std::vector<double>> data;

public:
```

```
    Matrix(int rows, int cols, double initial = 0)
        : rows(rows), cols(cols), data(rows, std::vector<double>(cols, initial))
{}

    // Оператор доступа к элементам (строка)
    std::vector<double>& operator[](int row) {
        if (row < 0 || row >= rows) {
            throw std::out_of_range("Row index out of range");
        }
        return data[row];
    }

    // Константная версия для доступа только для чтения
    const std::vector<double>& operator[](int row) const {
        if (row < 0 || row >= rows) {
            throw std::out_of_range("Row index out of range");
        }
        return data[row];
    }

    // Сложение матриц
    Matrix operator+(const Matrix& other) const {
        if (rows != other.rows || cols != other.cols) {
            throw std::invalid_argument("Matrix dimensions must match for
addition");
        }

        Matrix result(rows, cols);
        for (int i = 0; i < rows; i++) {
```

```
            for (int j = 0; j < cols; j++) {
                result[i][j] = data[i][j] + other.data[i][j];
            }
        }
        return result;
    }

    // Умножение матрицы на скаляр
    Matrix operator*(double scalar) const {
        Matrix result(rows, cols);
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = data[i][j] * scalar;
            }
        }
        return result;
    }

    // Умножение матриц
    Matrix operator*(const Matrix& other) const {
        if (cols != other.rows) {
            throw std::invalid_argument("Matrix dimensions incompatible for multiplication");
        }

        Matrix result(rows, other.cols);
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < other.cols; j++) {
                for (int k = 0; k < cols; k++) {
```

```
                result[i][j] += data[i][k] * other.data[k][j];
            }
        }
    }
    return result;
}

// Транспонирование
Matrix transpose() const {
    Matrix result(cols, rows);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result.data[j][i] = data[i][j];
        }
    }
    return result;
}

// Геттеры
int getRows() const { return rows; }
int getCols() const { return cols; }

// Вывод
friend std::ostream& operator<<(std::ostream& os, const Matrix& m) {
    for (int i = 0; i < m.rows; i++) {
        for (int j = 0; j < m.cols; j++) {
            os << std::setw(8) << std::setprecision(4) << m.data[i][j];
        }
        os << std::endl;
```

```
        }
        return os;
    }
};

int main() {
    Matrix A(2, 3, 1);
    Matrix B(2, 3, 2);

    std::cout << "Matrix A (2x3):" << std::endl;
    std::cout << A;
    std::cout << "\nMatrix B (2x3):" << std::endl;
    std::cout << B;
    std::cout << "\nA + B:" << std::endl;
    std::cout << A + B;
    std::cout << "\nA * 5:" << std::endl;
    std::cout << A * 5;

    Matrix C(3, 2, 1);
    std::cout << "\nMatrix C (3x2):" << std::endl;
    std::cout << C;
    std::cout << "\nA * C (2x3 * 3x2 = 2x2):" << std::endl;
    std::cout << A * C;

    std::cout << "\nTranspose of A:" << std::endl;
    std::cout << A.transpose();

    return 0;
}
```

3.4. Пример 4: Класс String с операторами +, =, [], <<

```
//
==========================================================================
=
// Программа: класс String с перегрузкой операторов
// Цель: демонстрация управления динамической памятью и перегрузки
//
==========================================================================
=

#include <iostream>
#include <cstring>

class String {
private:
    char* data;
    int length;

public:
    // Конструктор
    String(const char* str = "") {
        length = strlen(str);
        data = new char[length + 1];
        strcpy(data, str);
    }

    // Конструктор копирования
    String(const String& other) {
        length = other.length;
```

```
    data = new char[length + 1];
    strcpy(data, other.data);
}

// Деструктор
~String() {
    delete[] data;
}

// Оператор присваивания копированием
String& operator=(const String& other) {
    if (this != &other) {
        delete[] data;
        length = other.length;
        data = new char[length + 1];
        strcpy(data, other.data);
    }
    return *this;
}

// Оператор сложения (конкатенация)
String operator+(const String& other) const {
    int newLength = length + other.length;
    char* newData = new char[newLength + 1];
    strcpy(newData, data);
    strcat(newData, other.data);

    String result(newData);
    delete[] newData;
```

```
    return result;
}

// Оператор доступа по индексу
char& operator[](int index) {
    if (index < 0 || index >= length) {
        throw std::out_of_range("Index out of range");
    }
    return data[index];
}

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

// Операторы сравнения
bool operator==(const String& other) const {
    return strcmp(data, other.data) == 0;
}

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

bool operator<(const String& other) const {
    return strcmp(data, other.data) < 0;
```

```
    }

    // Геттеры
    int getLength() const { return length; }
    const char* c_str() const { return data; }

    // Вывод
    friend std::ostream& operator<<(std::ostream& os, const String& s) {
        os << s.data;
        return os;
    }

    // Ввод
    friend std::istream& operator>>(std::istream& is, String& s) {
        char buffer[1024];
        is >> buffer;
        s = String(buffer);
        return is;
    }
};

int main() {
    String s1("Hello");
    String s2("World");

    std::cout << "s1 = " << s1 << std::endl;
    std::cout << "s2 = " << s2 << std::endl;
    std::cout << "s1 + s2 = " << s1 + s2 << std::endl;
    std::cout << "s1 + \" \" + s2 = " << s1 + " " + s2 << std::endl;
```

```
    String s3 = s1;
    std::cout << "s3 = s1 -> " << s3 << std::endl;

    std::cout << "s1[1] = " << s1[1] << std::endl;
    s1[1] = 'a';
    std::cout << "After s1[1] = 'a': " << s1 << std::endl;

    std::cout << "s1 == s2? " << (s1 == s2 ? "true" : "false") << std::endl;

    return 0;
}
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Перегрузить арифметические операторы для простого класса.

№	Класс	Поля	Перегрузить
	Vector2D	x, y	+, -, * (на скаляр), <<
	Complex	re, im	+, -, *, <<
	Fraction	num, den	+, -, *, /, <<
	Point	x, y	+, -, ==, <<
	Time	hours, minutes	+, -, <<
	Money	rubles, kopecks	+, -, * (на число), <<
	Counter	value	++ (префиксный и постфиксный), <<

№	Класс	Поля	Перегрузить
	Date	day, month, year	==, !=, <<
	Rectangle	width, height	+ (объединение), * (масштаб), <<
10	Circle	radius	+ (сложение радиусов), * (масштаб), <<
11	Book	title, author, year	==, <<
12	Student	name, grade	==, <, <<
13	Temperature	celsius	+, -, <<
14	Vector3D	x, y, z	+, -, <<
15	Box	length, width, height	+ (объединение объёмов), * (масштаб), <<

Средний уровень (оценка «хорошо»)

Перегрузить операторы сравнения, составные операторы, оператор индексирования.

№	Класс	Поля	Перегрузить
1	Vector3D	x, y, z	+, -, * (скаляр), * (векторное), ==, !=, +=, -=, <<
2	Complex	re, im	+ , - , * , / , + = , -

№	Класс	Поля	Перегрузить
			=, *=, /=, ==, !=, <<, >>
3	Fraction	num, den	+, -, *, /, ==, !=, <, >, +=, <<
4	Matrix (2x2)	a,b,c,d	+, -, * (матрица × матрица), * (скаляр), ==, <<
5	IntArray	data, size	[] (с проверкой границ), =, ==, !=, <<
6	Polynomial (2 степени)	a,b,c	+, -, * (полином × полином), (), <<
7	String	data, length	+, =, [], ==, !=, <, >, <<
8	Set	vector<int>	+ (объединение), * (пересечение), - (разность), <<
9	Rational	num, den	+, -, *, /, ==, !=, <, >, <<
10	BigInteger (упрощённый)	vector<int>	+, -, *, ==, <<
11	Stack (массив)	data, top, capacity	[], ==, !=, <<
12	Queue (массив)	data, front, rear	[], ==, !=, <<
1	Vector3D с методами	x, y, z	dot (скалярное

№	Класс	Поля	Перегрузить
3			через *), cross (векторное через %), <<
14	ColorRGB	r, g, b	+ (смешение), * (яркость), ==, <<
15	Range	start, end, step	[] (доступ по индексу), size(), <<

Повышенный уровень (оценка «отлично»)

Полная перегрузка для сложных классов, включая операторы приведения типов и пользовательские литералы.

№	Класс	Задание
1	BigInteger	Полная реализация длинной арифметики: +, -, *, /, %, ==, !=, <, >, <<, >>, +=, -=, *=, /=, ++, --
2	Matrix (динамическая)	Операторы: +, -, * (матрица × матрица), * (вектор), [] (доступ к строке), () (доступ к элементу), =, +=, ==, <<, >>
3	Polynomial	Операторы: +, -, *, / (деление), % (остаток), (), [] (доступ к коэффициенту), =, <<, >>, + (с числом)
4	String	Полная

№	Класс	Задание
		реализация: +, =, +=, [], ==, !=, <, >, <<, >>, преобразование в const char*, оператор () для извлечения подстроки
5	SparseVector	Разреженный вектор: +, -, * (скалярное), * (число), [], <<, итераторы
6	MatrixSolver	Решение СЛАУ: операторы для матриц и векторов, перегрузка ~ для транспонирования, ! для обратной матрицы
7	Rational с упрощением	Все арифметические операторы, сравнения, ++, --, +=, -=, *=, /=, <<, >>, double() для приведения к double
8	BitSet	Битовый массив: & (AND), | (OR), ^ (XOR), ~ (NOT), << (сдвиг), [], size(), count(), преобразование в string
9	Fraction с шаблонным числителем	+, -, *, /, ==, !=, <, >, <<, >>, ~ для обратной дроби, double()

№	Класс	Задание
10	Complex с тригонометрической формой	Все арифметические операторы, ! для модуля, ^ для возведения в степень, <<, >>, пользовательские литералы _i
11	Quaternion	Кватернионы: +, -, *, /, ~ (сопряжение), ! (норма), ==, !=, <<, >>
12	Tensor (3×3×3)	Многомерный тензор: [] для доступа, +, -, * (свёртка), <<, итераторы
13	BigDecimal	Длинная десятичная арифметика: все арифметические операторы, сравнения, округление, <<, >>
14	Polynomial с комплексными коэффициентами	Все операторы, вычисление значений для комплексных чисел, нахождение корней
15	Matrix с разреженным представлением	Операторы для разреженных матриц, оптимизированные умножение и сложение, [] для доступа к элементам

5. Контрольные вопросы

Раздел А. Основы перегрузки операторов

1. Что такое перегрузка операторов? Какие преимущества она даёт?

2. Какие операторы нельзя перегружать в C++? Почему?
3. Какими двумя способами можно перегрузить оператор? В чём разница?
4. Какой синтаксис перегрузки оператора как метода класса?
5. Какой синтаксис перегрузки оператора как дружественной функции?

Раздел Б. Арифметические операторы

1. Почему операторы +, -, *, / должны возвращать новый объект (по значению), а не ссылку?
2. Почему эти операторы должны быть объявлены const?
3. Как перегрузить оператор * для умножения вектора на скаляр с обеих сторон (v * s и s * v)?
4. Как перегрузить унарный оператор - (унарный минус)?

Раздел В. Операторы присваивания и составные операторы

1. Какой тип должен возвращать оператор присваивания =? Почему?
2. Что такое самоприсваивание? Как его проверить?
3. Чем отличается префиксный инкремент от постфиксного? Как они перегружаются?
4. Что такое «глубокое копирование»? Когда оно необходимо?

Раздел Г. Операторы сравнения и вывода

1. Как перегрузить оператор ==? Как перегрузить != через него?
2. Почему оператор << не может быть методом класса? Как его правильно перегрузить?
3. Что должен возвращать оператор <<? Почему?

Раздел Д. Анализ кода и ошибки

1. Что не так с этим кодом?

```
class Vector {
    int x, y;
```

```
public:
    Vector operator+(const Vector& other) { return Vector(x+other.x, y+other.y); }
};
```

2. Почему этот оператор = опасен?

```
class String {
    char* data;
public:
    String& operator=(const String& other) {
        delete[] data;
        data = new char[strlen(other.data)+1];
        strcpy(data, other.data);
        return *this;
    }
};
```

3. Что выведет код?

```
Counter c(5);
std::cout << ++c << " " << c++ << " " << c << std::endl;
```

4. Почему этот оператор + не позволяет писать v1 + 5? Как исправить?

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (понятие перегрузки операторов, синтаксис, способы реализации, правила)

Ход выполнения работы:

Код программы с подробными комментариями (объяснение каждого перегруженного оператора)

Блок-схема алгоритма (для одного из операторов)

Диаграмма классов с указанием перегруженных операторов

Скриншоты выполнения с демонстрацией работы перегруженных операторов

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали о перегрузке операторов, как она делает классы удобнее)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №18)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) - правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех обязательных критериев (компиляция, корректность, перегрузка арифметических операторов, перегрузка оператора вывода).

Оценка «хорошо» (4) - правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего уровня (перегрузка операторов сравнения, составных операторов, оператора индексирования, корректная работа с динамической памятью).

Оценка «отлично» (5) - правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (полная перегрузка всех операторов для сложного

класса, перегрузка операторов приведения типов, пользовательские литералы, глубокая копия, правило трёх/пяти, UML-диаграмма в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
1	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
2	Компиляция и запуск			
2.1	Программа компилируется без ошибок	✔	✔	✔
2.2	Программа успешно запускается	✔	✔	✔
2.3	Отсутствие предупреждений компилятора	—	✔	✔
3	Перегрузка арифметических операторов			

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
3.1	+, -, *, / перегружены корректно	✔	✔	✔
3.2	Операторы возвращают новый объект по значению	✔	✔	✔
3.3	Операторы объявлены const	✔	✔	✔
3.4	Умножение скаляра работает с обеих сторон	—	✔	✔
4	Оператор вывода <<			
4.1	Оператор << перегружен как дружественная функция	✔	✔	✔
4.2	Возвращает std::ostream& для цепочек	✔	✔	✔
4.3	Форматирование вывода корректно	✔	✔	✔
5	Операторы сравнения			
5.1	== и != перегружены	—	✔	✔
5.2	<, >, <=, >= перегружены	—	✔	✔
5.3	Корректное сравнение	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	вещественных чисел (epsilon)			
6	Составные операторы и присваивание			
6.1	+=, -=, *=, /= перегружены	—	✔	✔
6.2	Оператор = перегружен (если нужен)	—	✔	✔
6.3	Проверка на самоприсваивание	—	✔	✔
6.4	Возврат *this по ссылке	—	✔	✔
7	Оператор индексирования []			
7.1	Перегружен [] для доступа по индексу	—	✔	✔
7.2	Проверка границ индекса	—	✔	✔
7.3	Константная версия для const объектов	—	—	✔
8	Управление памятью			
8.1	Корректное	—	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	выделение/освобождение памяти			
8.2	Правило трёх/пяти соблюдено	—	—	✔
8.3	Отсутствие утечек памяти	—	✔	✔
9	Качество кода			
9.1	Правильные отступы и форматирование	✔	✔	✔
9.2	Осмысленные имена классов и методов	—	✔	✔
9.3	Наличие комментариев (объяснение перегрузки)	—	✔	✔
9.4	Использование const везде, где нужно	—	✔	✔
10	Отчёт			
10.1	Титульный лист, цель, ход работы	✔	✔	✔
10.2	Скриншоты выполнения (минимум 3 теста)	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
10.3	Скриншоты для всех трёх уровней	—	✔	✔
10.4	Таблица тестирования	—	✔	✔
10.5	UML-диаграмма класса с операторами	—	—	✔
10.6	Ответы на контрольные вопросы	—	✔	✔

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) - обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня.

Перегрузить арифметические операторы (+, -, *, /).

Перегрузить оператор вывода <<.

Продемонстрировать работу перегруженных операторов в main().

Предоставить отчёт с листингом кода и скриншотами.

Средний уровень (оценка 4) - дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Перегрузить операторы сравнения (==, !=, <, >).

Перегрузить составные операторы (+=, -=, *=, /=).

Перегрузить оператор присваивания = (с проверкой на самоприсваивание).

Перегрузить оператор индексирования [] (с проверкой границ).

В отчёте представить:

Скриншоты для обоих уровней

Таблицу тестирования

Повышенный уровень (оценка 5) - требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Полная перегрузка операторов для сложного класса:

Все арифметические операторы

Все операторы сравнения

Все составные операторы

Операторы ++, -- (префиксные и постфиксные)

Оператор [] (константная и неконстантная версии)

Оператор () (функциональный вызов)

Управление динамической памятью:

Корректная реализация конструктора копирования

Корректная реализация оператора присваивания

Корректная реализация деструктора

Правило трёх/пяти

Дополнительные возможности:

Перегрузка операторов приведения типов (operator double())

Пользовательские литералы (например, _i для комплексных чисел)

В отчёте дополнительно представить:

UML-диаграмму класса со всеми операторами

Объяснение выбора способа перегрузки (метод vs дружественная функция)

7.4. Примеры проверки соответствия

Выполненные задания	Оценка	Проверка критериев
Только базовый вариант №1 (Vector2D с +, -, *, <<)	3	✔ Арифметические операторы, ✔ <<. Но: ✘ нет сравнений, ✘ нет составных операторов, ✘ нет []
Базовый №1 + средний №1 (Vector3D с +=, ==, [])	4	✔ Плюс сравнения, ✔ составные операторы, ✔ []. Но: ✘ нет операторов приведения, ✘ нет пользовательских литералов
Базовый №1 + средний №1 + повышенный №1 (BigInteger)	5	✔ Все критерии: ✔ полный набор операторов, ✔ управление памятью, ✔ правило трёх, ✔ UML-диаграмма

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	Влияние	Как исправить
Оператор + возвращает ссылку на локальный объект	Висячая ссылка	Возвращать по значению
Оператор = не проверяет	Утечка/повреждение	Добавить if (this !=

Ошибка	Влияние	Как исправить
самоприсваивание	данных	&other)
Оператор << возвращает void	Нельзя писать цепочки	Возвращать std::ostream&
Операторы сравнения не используют const	Нельзя сравнивать const-объекты	Добавить const после параметров
Отсутствует проверка границ в []	Выход за границы	Добавить проверку и исключение
Забыт виртуальный деструктор при наследовании	Утечка памяти	Добавить virtual ~Class() = default;

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающую программу (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Что такое перегрузка операторов и зачем она нужна

Разницу между перегрузкой как методом и как дружественной функцией

Как перегрузить оператор вывода <<

(для оценки 4) - как перегрузить оператор присваивания =

(для оценки 5) - что такое правило трёх и как оно связано с перегрузкой

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Почему оператор = должен возвращать ссылку на объект?"	Понимание цепочек присваивания
"Что произойдёт, если в operator+ не использовать const?"	Понимание константности
"Почему постфиксный инкремент возвращает по значению, а не по ссылке?"	Понимание семантики
"Как перегрузить оператор * для умножения вектора на матрицу?"	Понимание симметрии типов

Лабораторная работа №19

Разделение программы на файлы. Сборка проекта

1. Цель работы

Освоить разделение программ на модули - заголовочные файлы (.h) и файлы реализации (.cpp). Научиться правильно использовать директивы препроцессора #include, #pragma once или #ifndef/#define/#endif для защиты от повторного включения. Освоить компиляцию и сборку проекта из нескольких файлов, разнести определение класса и его методов по разным файлам, создать многофайловый проект.

2. Теоретическое обоснование

2.1. Модульное программирование в C++

Любую программу на C++ можно (и нужно) разделять на несколько файлов. Это даёт следующие преимущества:

1. Улучшение читаемости - код разбит на логические части
2. Ускорение компиляции - при изменении одного файла перекомпилируется только он
3. Повторное использование - один заголовочный файл можно включить в несколько проектов
4. Совместная разработка - разные разработчики могут работать над разными файлами
5. Сокрытие реализации - пользователи видят только интерфейс (.h), но не реализацию (.cpp)

Структура многофайлового проекта:

```
project/
├── main.cpp          // точка входа, использование классов
├── Point.h           // объявление класса Point (интерфейс)
├── Point.cpp         // реализация методов класса Point
├── Vector3D.h
├── Vector3D.cpp
├── utils.h
├── utils.cpp
└── Makefile / CMakeLists.txt (опционально)
```

2.2. Заголовочные файлы (.h)

Заголовочный файл содержит объявления (declarations):

1. Объявление класса (поля, прототипы методов)
2. Прототипы функций
3. Определения констант (const, constexpr)
4. Определения шаблонов
5. Inline-функции

6. Правила хорошего тона для заголовочных файлов:
7. Не определять обычные (не inline) функции в .h (это приведёт к множественным определениям)
8. Не определять статические данные в .h
9. Использовать защиту от повторного включения
10. Включать минимум необходимых заголовков
11. Использовать #pragma once или #ifndef/#define/#endif

Пример заголовочного файла Point.h:

```
//
==========================================================================
=
// Файл: Point.h
// Описание: объявление класса Point
//
==========================================================================
=

#ifndef POINT_H
#define POINT_H

#include <iostream>   // для std::ostream

class Point {
private:
    double x;
    double y;

public:
```

```
    // Конструкторы
    Point(double x = 0, double y = 0);

    // Конструктор копирования
    Point(const Point& other);

    // Деструктор
    ~Point();

    // Геттеры и сеттеры
    double getX() const;
    double getY() const;
    void setX(double x);
    void setY(double y);

    // Операторы
    Point operator+(const Point& other) const;
    Point operator-(const Point& other) const;
    Point& operator=(const Point& other);
    bool operator==(const Point& other) const;

    // Дружественная функция для вывода
    friend std::ostream& operator<<(std::ostream& os, const Point& p);
};

#endif // POINT_H
```

2.3. Файлы реализации (.cpp)

Файл реализации содержит определения (definitions):

1. Реализацию методов класса
2. Определения функций, объявленных в .h
3. Статические переменные (если нужно)
4. Файл реализации должен подключать свой заголовочный файл.

Пример файла реализации Point.cpp:

```
//
==========================================================================
=
// Файл: Point.cpp
// Описание: реализация методов класса Point
//
==========================================================================
=

#include "Point.h"
#include <cmath>

// Конструкторы
Point::Point(double x, double y) : x(x), y(y) {}

Point::Point(const Point& other) : x(other.x), y(other.y) {}

// Деструктор
Point::~Point() {}

// Геттеры и сеттеры
double Point::getX() const { return x; }
double Point::getY() const { return y; }
void Point::setX(double x) { this->x = x; }
```

```cpp
void Point::setY(double y) { this->y = y; }

// Операторы
Point Point::operator+(const Point& other) const {
    return Point(x + other.x, y + other.y);
}

Point Point::operator-(const Point& other) const {
    return Point(x - other.x, y - other.y);
}

Point& Point::operator=(const Point& other) {
    if (this != &other) {
        x = other.x;
        y = other.y;
    }
    return *this;
}

bool Point::operator==(const Point& other) const {
    const double EPS = 1e-9;
    return std::abs(x - other.x) < EPS && std::abs(y - other.y) < EPS;
}

// Дружественная функция вывода
std::ostream& operator<<(std::ostream& os, const Point& p) {
    os << "(" << p.x << ", " << p.y << ")";
    return os;
}
```

2.4. Защита от повторного включения (include guards)

Если один заголовочный файл включён несколько раз (прямо или косвенно), это может вызвать ошибки множественного определения.

Способ 1: #pragma once (нестандартный, но поддерживается всеми основными компиляторами)

```
#pragma once
// содержимое заголовочного файла
```

Способ 2: макросы #ifndef/#define/#endif (стандартный, переносимый)

```
#ifndef ИМЯ_ФАЙЛА_H
#define ИМЯ_ФАЙЛА_H

// содержимое заголовочного файла

#endif // ИМЯ_ФАЙЛА_H
```

Пример с проверкой:

```
#ifndef POINT_H
#define POINT_H

class Point {
    // ...
};

#endif // POINT_H
```

Почему это важно:

```
// main.cpp
#include "Point.h"
#include "Point.h"   // второй раз - НЕ включится из-за защиты!
```

2.5. #include - правила использования

Синтаксис	Поиск файла	Когда использовать
#include <имя>	В системных директориях компилятора	Стандартные библиотеки: <iostream>, <vector>, <cmath>
#include "имя"	Сначала в текущей директории, затем в системных	Свои заголовочные файлы: "Point.h", "utils.h"

Что происходит при #include:

Препроцессор текстуально вставляет содержимое указанного файла вместо директивы #include.

```
// До препроцессинга
#include <iostream>
#include "Point.h"

int main() {
    std::cout << "Hello!" << std::endl;
}

// После препроцессинга (в развёрнутом виде)
// содержимое iostream
// содержимое Point.h
int main() {
```

```
    std::cout << "Hello!" << std::endl;
}
```

2.6. Основной файл (main.cpp)

main.cpp содержит функцию main и использует классы/функции из других модулей.

```
//
==========================================================
=
    // Файл: main.cpp
    // Описание: главный файл программы, использующий класс Point
    //
==========================================================
=

    #include <iostream>
    #include "Point.h"

    int main() {
        Point p1(3, 4);
        Point p2(1, 2);

        Point p3 = p1 + p2;

        std::cout << "p1 = " << p1 << std::endl;
        std::cout << "p2 = " << p2 << std::endl;
        std::cout << "p1 + p2 = " << p3 << std::endl;

        return 0;
```

```
}
```

2.7. Разделение шаблонных классов

Важное ограничение: Шаблонные классы и функции нельзя разделить на .h и .cpp стандартным способом, потому что компилятор должен видеть полное определение шаблона в момент инстанцирования.

Решение 1: Весь код шаблона в .h файле.

```
// Vector.h
template<typename T>
class Vector {
    // полная реализация здесь
};
```

Решение 2: Включить .cpp в конце .h (не рекомендуется).

```
// Vector.h
template<typename T>
class Vector {
    // объявления
};

#include "Vector.cpp"   // включение реализации
```

Решение 3: Явное инстанцирование для нужных типов (для библиотек).

```
// Vector.h
template<typename T>
class Vector {
    // объявления
};

// Vector.cpp
#include "Vector.h"
template class Vector<int>;    // явное инстанцирование для int
```

```
template class Vector<double>; // для double
```

2.8. Компиляция и сборка проекта

Компиляция из командной строки (g++/clang++):

bash

```
# Компиляция всех .cpp файлов в объектные .o файлы
g++ -c main.cpp -o main.o
g++ -c Point.cpp -o Point.o

# Линковка объектных файлов в исполняемый файл
g++ main.o Point.o -o program

# Или одной командой
g++ main.cpp Point.cpp -o program
```

Компиляция в Visual Studio:

Добавить все .cpp и .h файлы в проект

Нажать Build Solution (Ctrl+Shift+B)

Компиляция в Code::Blocks:

Добавить файлы в проект

Нажать Build (Ctrl+F9)

Makefile пример:

makefile

```
CXX = g++
CXXFLAGS = -Wall -std=c++17

TARGET = program
SOURCES = main.cpp Point.cpp Vector3D.cpp
OBJECTS = $(SOURCES:.cpp=.o)
```

```
$(TARGET): $(OBJECTS)
	$(CXX) $(OBJECTS) -o $(TARGET)

%.o: %.cpp
	$(CXX) $(CXXFLAGS) -c $< -o $@

clean:
	rm -f $(OBJECTS) $(TARGET)
```

2.9. Пространства имён (namespace) для модулей

Пространства имён помогают избежать конфликтов имён между разными модулями.

```
// MathUtils.h
#pragma once
namespace MathUtils {
    double square(double x);
    double cube(double x);
}

// MathUtils.cpp
#include "MathUtils.h"
namespace MathUtils {
    double square(double x) { return x * x; }
    double cube(double x) { return x * x * x; }
}

// main.cpp
```

```cpp
#include "MathUtils.h"
#include <iostream>

int main() {
    std::cout << MathUtils::square(5) << std::endl;
    return 0;
}
```

2.10. Организация файлов в проекте

Рекомендуемая структура:

```
project/
├── include/          # публичные заголовочные файлы
│   ├── Point.h
│   └── Vector3D.h
├── src/              # файлы реализации
│   ├── main.cpp
│   ├── Point.cpp
│   └── Vector3D.cpp
├── lib/              # внешние библиотеки
├── build/            # объектные файлы и результат сборки
├── Makefile
└── README.md
```

Настройка путей для компилятора:

bash

```
g++ -I./include -c src/main.cpp -o build/main.o
g++ -I./include -c src/Point.cpp -o build/Point.o
g++ build/main.o build/Point.o -o build/program
```

2.11. Типичные ошибки при разделении на файлы

Ошибка	Пример	Последствие
Отсутствие include guards	Двойное включение файла	Ошибка переопределения
Определение функции в .h (не inline)	void f() {} в .h	Множественное определение
Забыт #include в .cpp	.cpp не включает свой .h	Ошибка: неизвестные типы
Циклические зависимости	A.h включает B.h, B.h включает A.h	Ошибка компиляции
Неправильные кавычки/скобки	#include <Point.h> для своего файла	Файл не найден
Шаблонный класс разделён на .h и .cpp	Реализация шаблона в .cpp	Ошибка линковки

```
// ОШИБКА: определение не-inline функции в .h
// utils.h
void helper() {          // НЕПРАВИЛЬНО: если utils.h включён в несколько .cpp
    std::cout << "help" << std::endl;
}
```

```
// ПРАВИЛЬНО: объявление в .h
// utils.h
void helper();

// ПРАВИЛЬНО: определение в .cpp
// utils.cpp
#include "utils.h"
void helper() {
    std::cout << "help" << std::endl
}
```

3. Методика выполнения работы с примерами

3.1. Пример 1: Разделение класса Point на .h и .cpp

Файл Point.h (заголовочный файл):

```
//
==========================================================
=
// Файл: Point.h
// Описание: объявление класса Point
//
==========================================================
=

#ifndef POINT_H
#define POINT_H

#include <iostream>
```

```
class Point {
private:
    double x;
    double y;

public:
    // Конструкторы
    Point(double x = 0, double y = 0);
    Point(const Point& other);

    // Деструктор
    ~Point();

    // Геттеры
    double getX() const;
    double getY() const;

    // Сеттеры
    void setX(double x);
    void setY(double y);

    // Арифметические операции
    Point add(const Point& other) const;
    Point subtract(const Point& other) const;

    // Расстояние до другой точки
    double distanceTo(const Point& other) const;
```

```
    // Оператор вывода
    friend std::ostream& operator<<(std::ostream& os, const Point& p);
};

#endif // POINT_H
```

Файл Point.cpp (реализация):

```
//
==========================================================
=
// Файл: Point.cpp
// Описание: реализация методов класса Point
//
==========================================================
=

#include "Point.h"
#include <cmath>

// Конструкторы
Point::Point(double x, double y) : x(x), y(y) {
    std::cout << "Point created: (" << x << ", " << y << ")" << std::endl;
}

Point::Point(const Point& other) : x(other.x), y(other.y) {
    std::cout << "Point copied: (" << x << ", " << y << ")" << std::endl;
}

// Деструктор
Point::~Point() {
```

```
    std::cout << "Point destroyed: (" << x << ", " << y << ")" << std::endl;
}

// Геттеры
double Point::getX() const { return x; }
double Point::getY() const { return y; }

// Сеттеры
void Point::setX(double x) { this->x = x; }
void Point::setY(double y) { this->y = y; }

// Арифметические операции
Point Point::add(const Point& other) const {
    return Point(x + other.x, y + other.y);
}

Point Point::subtract(const Point& other) const {
    return Point(x - other.x, y - other.y);
}

// Расстояние до другой точки
double Point::distanceTo(const Point& other) const {
    double dx = x - other.x;
    double dy = y - other.y;
    return std::sqrt(dx * dx + dy * dy);
}

// Оператор вывода
std::ostream& operator<<(std::ostream& os, const Point& p) {
```

```
    os << "(" << p.x << ", " << p.y << ")";
    return os;
}

//
==========================================================================
=
// Файл: main.cpp
// Описание: главная программа, использующая класс Point
//
==========================================================================
=

#include <iostream>
#include "Point.h"

int main() {
    std::cout << "=== Creating points ===" << std::endl;
    Point p1(3, 4);
    Point p2(1, 2);

    std::cout << "\n=== Point info ===" << std::endl;
    std::cout << "p1 = " << p1 << std::endl;
    std::cout << "p2 = " << p2 << std::endl;

    std::cout << "\n=== Operations ===" << std::endl;
    Point p3 = p1.add(p2);
    std::cout << "p1 + p2 = " << p3 << std::endl;
```

```
    Point p4 = p1.subtract(p2);
    std::cout << "p1 - p2 = " << p4 << std::endl;

    std::cout << "\n=== Distance ===" << std::endl;
    std::cout << "Distance between p1 and p2: " << p1.distanceTo(p2) <<
std::endl;

    std::cout << "\n=== End of program ===" << std::endl;
    return 0;
}
```

Компиляция:

```
bash
g++ -c Point.cpp -o Point.o
g++ -c main.cpp -o main.o
g++ Point.o main.o -o program
# Или одной командой:
g++ Point.cpp main.cpp -o program
```

3.2. Пример 2: Разделение нескольких классов

Файл Vector3D.h:

```
#ifndef VECTOR3D_H
#define VECTOR3D_H

#include <iostream>
#include <cmath>

class Vector3D {
```

```
private:
    double x, y, z;

public:
    Vector3D(double x = 0, double y = 0, double z = 0);
    Vector3D(const Vector3D& other);
    ~Vector3D();

    double getX() const;
    double getY() const;
    double getZ() const;

    void setX(double x);
    void setY(double y);
    void setZ(double z);

    Vector3D add(const Vector3D& other) const;
    Vector3D subtract(const Vector3D& other) const;
    Vector3D multiply(double scalar) const;
    double dot(const Vector3D& other) const;
    Vector3D cross(const Vector3D& other) const;
    double length() const;
    void normalize();

    friend std::ostream& operator<<(std::ostream& os, const Vector3D& v);
};

#endif
```

Файл Vector3D.cpp:

```
#include "Vector3D.h"

Vector3D::Vector3D(double x, double y, double z) : x(x), y(y), z(z) {}
Vector3D::Vector3D(const Vector3D& other) : x(other.x), y(other.y),
z(other.z) {}
Vector3D::~Vector3D() {}

double Vector3D::getX() const { return x; }
double Vector3D::getY() const { return y; }
double Vector3D::getZ() const { return z; }

void Vector3D::setX(double x) { this->x = x; }
void Vector3D::setY(double y) { this->y = y; }
void Vector3D::setZ(double z) { this->z = z; }

Vector3D Vector3D::add(const Vector3D& other) const {
    return Vector3D(x + other.x, y + other.y, z + other.z);
}

Vector3D Vector3D::subtract(const Vector3D& other) const {
    return Vector3D(x - other.x, y - other.y, z - other.z);
}

Vector3D Vector3D::multiply(double scalar) const {
    return Vector3D(x * scalar, y * scalar, z * scalar);
}

double Vector3D::dot(const Vector3D& other) const {
    return x * other.x + y * other.y + z * other.z;
```

```
}

Vector3D Vector3D::cross(const Vector3D& other) const {
    return Vector3D(
        y * other.z - z * other.y,
        z * other.x - x * other.z,
        x * other.y - y * other.x
    );
}

double Vector3D::length() const {
    return std::sqrt(x*x + y*y + z*z);
}

void Vector3D::normalize() {
    double len = length();
    if (len > 1e-9) {
        x /= len;
        y /= len;
        z /= len;
    }
}

std::ostream& operator<<(std::ostream& os, const Vector3D& v) {
    os << "(" << v.x << ", " << v.y << ", " << v.z << ")";
    return os;
}
```

Файл utils.h (вспомогательные функции):

```
#ifndef UTILS_H
```

```
#define UTILS_H

#include <vector>
#include "Point.h"
#include "Vector3D.h"

namespace Utils {
    // Вычисление центра масс массива точек
    Point centroid(const std::vector<Point>& points);

    // Вычисление суммы векторов
    Vector3D sum(const std::vector<Vector3D>& vectors);

    // Проверка, является ли число простым
    bool isPrime(int n);
}

#endif
```

Файл utils.cpp:

```
#include "utils.h"
#include <cmath>

namespace Utils {

    Point centroid(const std::vector<Point>& points) {
        if (points.empty()) return Point(0, 0);

        double sumX = 0, sumY = 0;
        for (const auto& p : points) {
```

```
            sumX += p.getX();
            sumY += p.getY();
        }
        return Point(sumX / points.size(), sumY / points.size());
    }

    Vector3D sum(const std::vector<Vector3D>& vectors) {
        Vector3D result;
        for (const auto& v : vectors) {
            result = result.add(v);
        }
        return result;
    }

    bool isPrime(int n) {
        if (n <= 1) return false;
        if (n == 2) return true;
        if (n % 2 == 0) return false;

        int limit = static_cast<int>(std::sqrt(n));
        for (int i = 3; i <= limit; i += 2) {
            if (n % i == 0) return false;
        }
        return true;
    }

} // namespace Utils
```

Файл main.cpp:

```cpp
#include <iostream>
#include <vector>
#include "Point.h"
#include "Vector3D.h"
#include "utils.h"

int main() {
    std::cout << "=== Working with Points ===" << std::endl;
    std::vector<Point> points = {Point(0,0), Point(2,0), Point(1,2)};
    Point c = Utils::centroid(points);
    std::cout << "Centroid: " << c << std::endl;

    std::cout << "\n=== Working with Vectors ===" << std::endl;
    std::vector<Vector3D> vectors = {
        Vector3D(1,0,0),
        Vector3D(0,1,0),
        Vector3D(0,0,1)
    };
    Vector3D s = Utils::sum(vectors);
    std::cout << "Sum of vectors: " << s << std::endl;

    std::cout << "\n=== Prime numbers test ===" << std::endl;
    for (int i = 1; i <= 20; i++) {
        if (Utils::isPrime(i)) {
            std::cout << i << " is prime" << std::endl;
        }
    }

    return 0;
```

```
}
```

3.3. Пример 3: Проект с классами, использующими друг друга

Файл Person.h:

```
#ifndef PERSON_H
#define PERSON_H

#include <string>
#include <vector>

// Предварительное объявление класса Address (для избежания циклической зависимости)
class Address;

class Person {
private:
    std::string name;
    int age;
    Address* address;  // указатель на адрес (чтобы не требовать полного определения)

public:
    Person(const std::string& name, int age);
    ~Person();

    std::string getName() const;
    int getAge() const;

    void setAddress(Address* addr);
    Address* getAddress() const;
```

```
    void print() const;
};

#endif
```

Файл Address.h:

```
#ifndef ADDRESS_H
#define ADDRESS_H

#include <string>
#include <vector>
#include "Person.h"

class Address {
private:
    std::string street;
    std::string city;
    std::string zipCode;
    std::vector<Person*> residents;

public:
    Address(const std::string& street, const std::string& city, const std::string& zipCode);
    ~Address();

    std::string getStreet() const;
    std::string getCity() const;
    std::string getZipCode() const;
```

```
    void addResident(Person* person);
    void removeResident(Person* person);
    void printResidents() const;

    void print() const;
};

#endif
```

Файл Person.cpp:

```
#include "Person.h"
#include "Address.h"
#include <iostream>

Person::Person(const std::string& name, int age)
    : name(name), age(age), address(nullptr) {
    std::cout << "Person created: " << name << std::endl;
}

Person::~Person() {
    std::cout << "Person destroyed: " << name << std::endl;
    // Не удаляем address - он может быть общим для нескольких Person
}

std::string Person::getName() const { return name; }
int Person::getAge() const { return age; }

void Person::setAddress(Address* addr) {
    address = addr;
    if (address) {
```

```
        address->addResident(this);
    }
}

Address* Person::getAddress() const { return address; }

void Person::print() const {
    std::cout << "Person: " << name << ", " << age << " years old" <<
std::endl;
    if (address) {
        std::cout << "  Lives at: ";
        address->print();
    } else {
        std::cout << "  No address assigned" << std::endl;
    }
}
```

Файл Address.cpp:

```
#include "Address.h"
#include <iostream>
#include <algorithm>

Address::Address(const std::string& street, const std::string& city, const
std::string& zipCode)
    : street(street), city(city), zipCode(zipCode) {
    std::cout << "Address created: " << street << ", " << city << std::endl;
}

Address::~Address() {
    std::cout << "Address destroyed: " << street << std::endl;
```

```
}

std::string Address::getStreet() const { return street; }
std::string Address::getCity() const { return city; }
std::string Address::getZipCode() const { return zipCode; }

void Address::addResident(Person* person) {
    residents.push_back(person);
}

void Address::removeResident(Person* person) {
    auto it = std::find(residents.begin(), residents.end(), person);
    if (it != residents.end()) {
        residents.erase(it);
    }
}

void Address::printResidents() const {
    std::cout << "Residents at " << street << ":" << std::endl;
    for (const auto& person : residents) {
        std::cout << "  - " << person->getName() << std::endl;
    }
}

void Address::print() const {
    std::cout << street << ", " << city << ", " << zipCode << std::endl;
}
```

Файл main.cpp:

```
#include <iostream>
```

```cpp
#include "Person.h"
#include "Address.h"

int main() {
    std::cout << "=== Creating objects ===" << std::endl;

    Address addr("Lenina str., 10", "Moscow", "101000");
    Person alice("Alice", 25);
    Person bob("Bob", 30);

    std::cout << "\n=== Assigning addresses ===" << std::endl;
    alice.setAddress(&addr);
    bob.setAddress(&addr);

    std::cout << "\n=== Person info ===" << std::endl;
    alice.print();
    bob.print();

    std::cout << "\n=== Address info ===" << std::endl;
    addr.print();
    addr.printResidents();

    std::cout << "\n=== End of program ===" << std::endl;
    return 0;
}
```

3.4. Пример 4: Использование #pragma once и организация проекта

Файл Config.h (глобальные константы и настройки):

```
//
================================================================
=
// Файл: Config.h
// Описание: глобальные константы и настройки проекта
//
================================================================
=

#pragma once   // простая защита от повторного включения

#include <string>

namespace Config {
    // Версия программы
    constexpr int VERSION_MAJOR = 1;
    constexpr int VERSION_MINOR = 0;
    constexpr int VERSION_PATCH = 0;

    // Настройки вывода
    constexpr int PRECISION = 4;
    constexpr char DECIMAL_SEPARATOR = '.';

    // Максимальные значения
    constexpr int MAX_BUFFER_SIZE = 1024;
    constexpr double EPSILON = 1e-9;

    // Пути к файлам
    const std::string DATA_DIR = "./data/";
```

```
    const std::string LOG_FILE = "log.txt";

    // Inline-функция для получения версии
    inline std::string getVersion() {
        return std::to_string(VERSION_MAJOR) + "." +
               std::to_string(VERSION_MINOR) + "." +
               std::to_string(VERSION_PATCH);
    }
}
```

Файл MathUtils.h:

```
#pragma once

#include <vector>
#include <cmath>

namespace MathUtils {
    // Статистические функции
    double mean(const std::vector<double>& data);
    double variance(const std::vector<double>& data);
    double stddev(const std::vector<double>& data);

    // Линейная алгебра
    double dotProduct(const std::vector<double>& a, const std::vector<double>& b);
    std::vector<double> matrixVectorMultiply(const std::vector<std::vector<double>>& A,
                                             const std::vector<double>& x);
}
```

Файл MathUtils.cpp:

```cpp
#include "MathUtils.h"
#include "Config.h"
#include <numeric>
#include <stdexcept>

namespace MathUtils {

    double mean(const std::vector<double>& data) {
        if (data.empty()) return 0.0;
        double sum = std::accumulate(data.begin(), data.end(), 0.0);
        return sum / data.size();
    }

    double variance(const std::vector<double>& data) {
        if (data.size() < 2) return 0.0;
        double m = mean(data);
        double sumSq = 0.0;
        for (double x : data) {
            double diff = x - m;
            sumSq += diff * diff;
        }
        return sumSq / (data.size() - 1);
    }

    double stddev(const std::vector<double>& data) {
        return std::sqrt(variance(data));
    }
```

```
double dotProduct(const std::vector<double>& a, const std::vector<double>& b) {
    if (a.size() != b.size()) {
        throw std::invalid_argument("Vector sizes must match");
    }
    double result = 0.0;
    for (size_t i = 0; i < a.size(); i++) {
        result += a[i] * b[i];
    }
    return result;
}

std::vector<double> matrixVectorMultiply(const std::vector<std::vector<double>>& A,
                                         const std::vector<double>& x) {
    if (A.empty() || x.empty()) return {};
    if (A[0].size() != x.size()) {
        throw std::invalid_argument("Matrix columns must match vector size");
    }

    std::vector<double> result(A.size(), 0.0);
    for (size_t i = 0; i < A.size(); i++) {
        for (size_t j = 0; j < x.size(); j++) {
            result[i] += A[i][j] * x[j];
        }
    }
    return result;
}
```

```
} // namespace MathUtils
```

Файл main.cpp:

```
#include <iostream>
#include <iomanip>
#include <vector>
#include "Config.h"
#include "MathUtils.h"

int main() {
    std::cout << "Program version: " << Config::getVersion() << std::endl;
    std::cout << "Data directory: " << Config::DATA_DIR << std::endl;

    std::vector<double> data = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0};

    std::cout << std::fixed << std::setprecision(Config::PRECISION);
    std::cout << "\nData: ";
    for (double x : data) std::cout << x << " ";
    std::cout << std::endl;

    std::cout << "Mean: " << MathUtils::mean(data) << std::endl;
    std::cout << "Variance: " << MathUtils::variance(data) << std::endl;
    std::cout << "Standard deviation: " << MathUtils::stddev(data) <<
std::endl;

    return 0;
}
```

4. Индивидуальные задания

Базовый уровень (оценка «удовлетворительно»)

Разделить один класс на .h и .cpp файлы, собрать проект.

№	Класс	Файлы
1	Point	Point.h, Point.cpp, main.cpp
2	Rectangle	Rectangle.h, Rectangle.cpp, main.cpp
3	Circle	Circle.h, Circle.cpp, main.cpp
4	Book	Book.h, Book.cpp, main.cpp
5	Student	Student.h, Student.cpp, main.cpp
6	Car	Car.h, Car.cpp, main.cpp
7	Counter	Counter.h, Counter.cpp, main.cpp
8	BankAccount	BankAccount.h, BankAccount.cpp, main.cpp
9	Time	Time.h, Time.cpp, main.cpp
10	Fraction	Fraction.h, Fraction.cpp, main.cpp
11	Triangle	Triangle.h, Triangle.cpp, main.cpp
12	Clock	Clock.h, Clock.cpp, main.cpp
13	Date	Date.h, Date.cpp, main.cpp

№	Класс	Файлы
14	Product	Product.h, Product.cpp, main.cpp
15	Vector2D	Vector2D.h, Vector2D.cpp, main.cpp

Средний уровень (оценка «хорошо»)

Разделить несколько взаимосвязанных классов на файлы.

№	Классы	Файлы
1	Point, Circle	Point.h/cpp, Circle.h/cpp, main.cpp
2	Person, Student	Person.h/cpp, Student.h/cpp, main.cpp
3	Shape, Rectangle, Circle	Shape.h/cpp, Rectangle.h/cpp, Circle.h/cpp, main.cpp
4	Animal, Dog, Cat	Animal.h/cpp, Dog.h/cpp, Cat.h/cpp, main.cpp
5	Vehicle, Car, Motorcycle	Vehicle.h/cpp, Car.h/cpp, Motorcycle.h/cpp, main.cpp
6	Employee, Manager, Developer	Employee.h/cpp, Manager.h/cpp, Developer.h/cpp, main.cpp
7	BankAccount, SavingsAccount, CheckingAccount	BankAccount.h/cpp, SavingsAccount.h/cpp, CheckingAccount.h/cpp, main.cpp

№	Классы	Файлы
8	Vector3D, Matrix3x3	Vector3D.h/cpp, Matrix3x3.h/cpp, main.cpp
9	Complex, Polynomial	Complex.h/cpp, Polynomial.h/cpp, main.cpp
10	Date, DateTime	Date.h/cpp, DateTime.h/cpp, main.cpp
11	User, Admin, Guest	User.h/cpp, Admin.h/cpp, Guest.h/cpp, main.cpp
12	Product, Electronics, Clothing	Product.h/cpp, Electronics.h/cpp, Clothing.h/cpp, main.cpp
13	Media, Book, DVD	Media.h/cpp, Book.h/cpp, DVD.h/cpp, main.cpp
14	GameCharacter, Warrior, Mage	GameCharacter.h/cpp, Warrior.h/cpp, Mage.h/cpp, main.cpp
15	Document, TextDocument, Spreadsheet	Document.h/cpp, TextDocument.h/cpp, Spreadsheet.h/cpp, main.cpp

Повышенный уровень (оценка «отлично»)

Создать полноценный многофайловый проект с пространствами имён, конфигурационным файлом и сборкой.

№	Задание
1	Геометрическая библиотека: классы Point, Line, Circle, Rectangle, Triangle. Пространство

№	Задание
	имён Geometry. Функции: пересечение, расстояние, площадь.
2	Математическая библиотека: классы Vector3D, Matrix3x3, Quaternion. Пространство имён Math. Операции: умножение, нормализация, интерполяция.
3	База данных сотрудников: классы Person, Employee, Department. Хранение в векторах, сохранение/загрузка в файл.
4	Графический редактор: классы Shape, Circle, Rectangle, Line. Пространство имён Graphics. Команды: добавить, удалить, отрисовать, переместить.
5	Интернет-магазин: классы Product, Cart, Order, User. Пространства имён Store. Работа с корзиной, оформление заказа.
6	Система учёта студентов: классы Student, Course, GradeBook. Пространства имён University. Расчёт среднего балла, рейтинг.
7	Библиотека книг: классы Book, Library, Member. Пространства имён Library. Поиск по автору/названию, выдача книг.
8	Калькулятор выражений: парсер выражений, классы Token, Expression, Calculator. Пространства имён MathParser. Поддержка +, -, *, /, ^, скобки.
9	Игровой движок: классы GameObject, Player, Enemy, Bullet. Пространства имён Game. Обработка столкновений, обновление состояния.

№	Задание
10	Генератор лабиринтов: классы Cell, Maze, MazeGenerator. Пространства имён Maze. Алгоритмы генерации (DFS, Prim).
11	Анализатор текста: классы TextAnalyzer, WordCounter, FrequencyTable. Пространства имён TextAnalysis. Анализ частоты слов, предложений.
12	Графический калькулятор функций: построение графиков функций, классы Function, Plotter. Пространства имён Graphing.
13	Сетевое приложение: классы Socket, Client, Server. Пространства имён Network. Простой эхо-сервер.
14	Компилятор (упрощённый): лексический анализатор, парсер. Классы Token, Lexer, Parser. Пространства имён Compiler.
15	База данных (in-memory): классы Table, Row, Database, Query. Пространства имён Database. Поддержка SELECT, INSERT, DELETE.

5. Контрольные вопросы

Раздел А. Заголовочные файлы

Что такое заголовочный файл (.h)? Что в нём должно находиться?

Что такое файл реализации (.cpp)? Что в нём должно находиться?

Зачем разделять программу на несколько файлов? Какие преимущества это даёт?

Что такое include guards? Как они реализуются?

В чём разница между #pragma once и #ifndef/#define/#endif? Какие преимущества и недостатки у каждого?

Раздел Б. Директивы #include

Чем отличается #include <file> от #include "file"?

Что происходит при использовании #include? Как работает препроцессор?

Какие проблемы могут возникнуть при циклических включениях? Как их решить?

Как правильно организовать включение заголовочных файлов в проекте?

Раздел В. Компиляция и сборка

Какие этапы проходит программа от исходного кода до исполняемого файла?

Что такое объектный файл (.o или .obj)?

Что такое линковка (linking)? Какие ошибки возникают на этапе линковки?

Как скомпилировать многофайловый проект из командной строки?

Что такое Makefile? Для чего он нужен?

Раздел Г. Шаблоны и разделение

Почему шаблонные классы нельзя разделить на .h и .cpp стандартным способом?

Как правильно организовать код шаблонного класса?

Что такое явное инстанцирование (explicit instantiation)? Когда оно используется?

Раздел Д. Анализ кода и ошибки

Почему возникает ошибка multiple definition? Как её исправить?

Почему возникает ошибка undefined reference? Как её исправить?

Что будет, если в .h файле определить не-inline функцию?

Как правильно организовать глобальные константы в многофайловом проекте?

6. Требования к отчёту

Отчёт должен содержать:

Титульный лист

Цель работы

Теоретическое обоснование (разделение на файлы, заголовочные файлы, include guards, компиляция и линковка)

Ход выполнения работы:

Листинги всех созданных файлов (.h, .cpp, main.cpp) с подробными комментариями

Схема зависимостей между файлами (что от чего зависит)

Инструкция по сборке (команды компиляции или настройки IDE)

Скриншоты процесса компиляции и выполнения программы

Скриншоты для всех трёх уровней заданий

Таблица тестирования (входные данные → ожидаемый результат → фактический результат)

Ответы на контрольные вопросы (письменно, развёрнуто)

Вывод (что нового узнали о многофайловых проектах, как это помогает в разработке)

Формат: .docx или .pdf.

7. Критерии оценивания (ЛР №19)

7.1. Шкала оценивания

Шкала оценивания лабораторной работы:

Оценка «удовлетворительно» (3) - правильно выполнено только базовое задание (один из 15 вариантов базового уровня) при соблюдении всех

обязательных критериев (корректное разделение одного класса на .h и .cpp, наличие include guards, успешная компиляция и сборка).

Оценка «хорошо» (4) - правильно выполнены базовый и средний уровни (соответственно один вариант базового и один вариант среднего уровня из выданных вариантов) при соблюдении критериев для среднего уровня (разделение нескольких взаимосвязанных классов на файлы, правильная организация зависимостей, отсутствие циклических включений).

Оценка «отлично» (5) - правильно выполнены все три уровня (базовый, средний и повышенный вариант) при соблюдении всех критериев, включая повышенные требования (создание полноценного многофайлового проекта с пространствами имён, конфигурационным файлом, демонстрация сборки через командную строку или Makefile, схема зависимостей в отчёте).

Важно: Выполнение более сложного уровня без выполнения предыдущего не засчитывается. Каждый уровень должен быть реализован в виде отдельной программы или отдельной, чётко выделенной части кода с соответствующими комментариями. В отчёте должны быть представлены результаты выполнения всех трёх уровней.

7.2. Детальная таблица критериев

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
1	Выполненные уровни заданий	Только базовый	Базовый + средний	Базовый + средний + повышенный
2	Структура файлов			
2.1	Заголовочный файл (.h) создан	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
2.2	Файл реализации (.cpp) создан	✔	✔	✔
2.3	Главный файл (main.cpp) создан	✔	✔	✔
2.4	Все файлы корректно связаны	✔	✔	✔
3	Защита от повторного включения			
3.1	Include guards (#ifndef/#define/#endif или #pragma once)	✔	✔	✔
3.2	Отсутствие ошибок множественного включения	✔	✔	✔
3.3	Корректные #include директивы	✔	✔	✔
4	Содержание файлов			
4.1	В .h только объявления (не определения)	✔	✔	✔
4.2	В .cpp определения методов	✔	✔	✔
4.3	Отсутствие определений обычных функций в .h	✔	✔	✔
5	Организация кода			

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
5.1	Класс правильно объявлен в .h	✔	✔	✔
5.2	Методы реализованы в .cpp	✔	✔	✔
5.3	Использование #include своего .h в .cpp	✔	✔	✔
5.4	Использование пространств имён (для повышенного)	—	—	✔
6	Зависимости			
6.1	Минимальные необходимые #include	—	✔	✔
6.2	Отсутствие циклических зависимостей	—	✔	✔
6.3	Предварительные объявления для сложных зависимостей	—	—	✔
7	Сборка проекта			
7.1	Программа успешно компилируется	✔	✔	✔
7.2	Программа успешно линкуется	✔	✔	✔
7.3	Программа корректно	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
	выполняется			
7.4	Демонстрация сборки из командной строки	—	—	✔
8	Логика программы			
8.1	Функциональность класса сохранена	✔	✔	✔
8.2	Все методы работают корректно	✔	✔	✔
8.3	Дополнительные функции (для повышенного) работают	—	—	✔
9	Качество кода			
9.1	Правильные отступы и форматирование	✔	✔	✔
9.2	Осмысленные имена файлов	✔	✔	✔
9.3	Наличие комментариев (заголовки файлов)	—	✔	✔
9.4	Документация интерфейса в .h	—	—	✔
10	Отчёт			
10.	Титульный лист, цель, ход	✔	✔	✔

	Критерий	Базовый (3)	Средний (4)	Повышенный (5)
1	работы			
10.2	Листинги всех файлов	✔	✔	✔
10.3	Скриншоты компиляции и выполнения	✔	✔	✔
10.4	Скриншоты для всех трёх уровней	—	✔	✔
10.5	Схема зависимостей между файлами	—	✔	✔
10.6	Инструкция по сборке проекта	—	—	✔
10.7	Таблица тестирования	—	✔	✔
10.8	Ответы на контрольные вопросы	—	✔	✔

7.3. Расшифровка критериев по уровням

Базовый уровень (оценка 3) - обязательные требования

Для получения оценки «удовлетворительно» студент должен:

Выполнить один любой вариант из базового уровня.

Разделить класс на заголовочный файл (.h) и файл реализации (.cpp).

Написать главный файл (main.cpp), использующий класс.

Использовать include guards для защиты от повторного включения.

Успешно скомпилировать и запустить программу.

Предоставить отчёт с листингами всех файлов и скриншотами.

Средний уровень (оценка 4) - дополнительные требования

Для получения оценки «хорошо» студент должен выполнить базовый И средний уровни, а также:

Разделить несколько взаимосвязанных классов на файлы.

Правильно организовать зависимости между файлами (минимальные #include).

Избежать циклических зависимостей.

Продемонстрировать успешную сборку проекта.

В отчёте представить:

Листинги всех файлов

Схему зависимостей между файлами

Таблицу тестирования

Повышенный уровень (оценка 5) - требования на отлично

Для получения оценки «отлично» студент должен выполнить все три уровня (базовый, средний, повышенный), а также:

Полноценный многофайловый проект:

Минимум 3 класса

Пространства имён для логической группировки

Конфигурационный файл с глобальными константами

Профессиональная организация:

Чёткое разделение на интерфейс (.h) и реализацию (.cpp)

Предварительные объявления для сложных зависимостей

Inline-функции где уместно

Демонстрация сборки:

Сборка из командной строки (с приведением команд)

Или настройка Makefile / CMake

В отчёте дополнительно представить:

Схему зависимостей между файлами

Инструкцию по сборке проекта

Диаграмму классов (UML) проекта

7.4. Примеры проверки соответствия

Выполненные задания	Оценка	Проверка критериев
Только базовый вариант №1 (Point)	3	✔ .h и .cpp, ✔ include gua компиляция. Но: ✘ один класс, нескольких файлов классов
Базовый №1 + средний №2 (Person, Student)	4	✔ Плюс несколько классов, ✔ кор зависимости, ✔ успешная сборка. Н пространств имён, ✘ нет конфигураци
Базовый №1 + средний №2 + повышенный №1 (геометрическая библиотека)	5	✔ Все критерии: ✔ несколько кла пространства имён, ✔ конфигура схема зависимостей, ✔ инструкция с

7.5. Типичные ошибки, влияющие на снижение оценки

Ошибка	Влияние	Как исправить
Отсутствие include guards	Ошибка множественного определения	Добавить #pragma once или #ifndef

Ошибка	Влияние	Как исправить
Определение не-inline функции в .h	Ошибка множественного определения	Перенести в .cpp
Забыт #include своего .h в .cpp	Ошибка: неизвестные типы	Добавить #include "Class.h"
Циклические зависимости	Ошибка компиляции	Использовать предварительные объявления
Неправильные кавычки/скобки в #include	Файл не найден	Использовать "" для своих файлов
Класс шаблонный, разделённый на .h и .cpp	Ошибка линковки	Реализовывать шаблоны в .h

7.6. Минимальный порог для защиты

Для допуска к защите лабораторной работы студент должен:

Предоставить работающий проект (компилируется и выполняется)

Предоставить отчёт, оформленный согласно требованиям (раздел 6)

Устно объяснить:

Зачем нужно разделять программу на файлы

Как организовать заголовочный файл и файл реализации

Что такое include guards и зачем они нужны

(для оценки 4) - как организовать зависимости между несколькими классами

(для оценки 5) - как собрать проект из командной строки

7.7. Дополнительные задания для проверки понимания (устно на защите)

Вопрос	Проверяет
"Что произойдёт, если включить .cpp файл с реализацией в другой .cpp?"	Понимание линковки
"Как разрешить циклическую зависимость между двумя классами?"	Понимание предварительных объявлений
"Почему библиотека <iostream> включается через <>, а Point.h через \"\"?"	Понимание путей поиска
"Что такое inline функция и когда её можно использовать в .h?"	Понимание ODR (One Definition Rule)

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
«Введение в программирование и алгоритмы»
для студентов направления подготовки
09.03.04 Программная инженерия
Направленность (профиль)
«Разработка и сопровождение программного обеспечения»

Ставрополь 2026

ВВЕДЕНИЕ

Методические указания предназначены для организации самостоятельной работы студентов, обучающихся по направлению подготовки 09.03.04 «Программная инженерия», при изучении дисциплины «Введение в программирование и алгоритмы».

Самостоятельная работа является неотъемлемой частью учебного процесса и направлена на:

1. закрепление и углубление теоретических знаний, полученных на аудиторных занятиях;
2. формирование практических навыков разработки программного кода;
3. развитие способности к самообучению и самоорганизации;
4. подготовку к текущему и промежуточному контролю успеваемости.
5. В данных методических указаниях представлены:
6. виды и объёмы самостоятельной работы;
7. методические рекомендации по изучению теоретического материала;
8. указания по выполнению практических (лабораторных) работ;
9. вопросы для собеседования по самостоятельно изученной литературе;
10. критерии оценивания сформированности компетенций.

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1.1. Цели и задачи самостоятельной работы

Цель самостоятельной работы - формирование у студентов знаний, умений и навыков в области алгоритмизации и программирования на языке C++, необходимых для решения профессиональных задач в области программной инженерии.

Задачи самостоятельной работы:

№	Задача	Формируемый результат
1	Изучение теоретического материала по темам дисциплины	Знание синтаксиса и конструкций языка C++
2	Подготовка к выполнению лабораторных работ	Умение применять теоретические знания на практике
3	Разработка, отладка и тестирование программ	Навык написания корректного и читаемого кода
4	Оформление отчётов по лабораторным работам	Умение документировать результаты работы
5	Подготовка к собеседованию и защите работ	Способность аргументированно объяснять решения

1.2. Виды самостоятельной работы

	Вид самостоятельной работы	Трудоёмкость (часов)	Форма контроля
	Изучение теоретического материала по конспектам лекций и учебной литературе	20	Опрос, собеседование
	Подготовка к лабораторным работам (19 работ × 2 часа)	15	Допуск к работе
	Подготовка к защите	10	Защита

	Вид самостоятельной работы	Трудоёмкость (часов)	Форма контроля
	лабораторных работ (19 работ × 1 час)		
	Самостоятельное изучение дополнительной литературы	14	Собеседование
	Итого	54	

1.3. Планируемые результаты (компетенции)

В результате выполнения самостоятельной работы студент должен овладеть следующими компетенциями:

Код компетенции	Содержание компетенции	Код индикатора	Содержание индикатора	Дескриптор (планируемый результат обучения)
ОПК-3	Способен решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и	ИД-1опк-3	Решает стандартные задачи профессиональной деятельности на основе информационной культуры с применением информационно-коммуникационных технологий	**Знает:** справочные системы (cppreference.com) и официальные источники для поиска синтаксиса языка и описания стандартных библиотек C++. **Умеет:** использовать официальную документацию и электронные образовательные ресурсы для

Код компетенции	Содержание компетенции	Код индикатора	Содержание индикатора	Дескриптор (планируемый результат обучения)
	с учетом основных требований информационной безопасности			решения учебных задач по программированию. **Владеет:** навыками поиска, анализа и применения информации из технических источников при разработке программ.
ПК-1	Способен осуществлять управление программными проектами: планировать, контролировать сроки и ресурсы, применять системы контроля версий и инструменты управления задачами	ИД-2пк-1	Использует системы контроля версий (Git, Git Flow) и платформы совместной разработки (GitLab, GitHub) для организации командной работы	**Знает:** основы работы с системой контроля версий Git (инициализация репозитория, создание коммитов, работа с ветками). **Умеет:** использовать Git для сохранения результатов лабораторных работ, создавать коммиты с понятными описаниями изменений. **Владеет:** навыками базовой работы с Git-репозиторием (clone, add, commit,

Код компетенции	Содержание компетенции	Код индикатора	Содержание индикатора	Дескриптор (планируемый результат обучения)
				push) в рамках индивидуальной учебной деятельности.
ПК-3	Владеет навыками разработки программного обеспечения: способен реализовывать алгоритмы, писать код на современных языках программирования, создавать веб- и desktop-приложения, работать с базами данных	ИД-1пк-3	Разрабатывает алгоритмы и реализует их на современных языках программирования (C++, C#, Python, Java, JavaScript/TypeScript, Go) с использованием эффективных структур данных и стандартных библиотек	**Знает:** синтаксис и семантику языка C++ (типы данных, операторы, управляющие конструкции, функции, классы). **Умеет:** разрабатывать алгоритмы решения типовых вычислительных задач (поиск, сортировка, табулирование, обработка массивов) и реализовывать их на языке C++ в виде консольных приложений. **Владеет:** навыками структурного и объектно-ориентированного программирования на C++ с использованием стандартной библиотеки.

Код компетенции	Содержание компетенции	Код индикатора	Содержание индикатора	Дескриптор (планируемый результат обучения)
ПК-3	Владеет навыками разработки программного обеспечения: способен реализовывать алгоритмы, писать код на современных языках программирования, создавать веб- и desktop-приложения, работать с базами данных	ИД-4пк-3	Пишет модульный, документированный и тестируемый код, соблюдая стандарты оформления и принципы SOLID, использует системы сборки (Make, Gradle, npm)	**Знает:** стандарты оформления программного кода (отступы, именование переменных, комментарии). **Умеет:** оформлять программный код с соблюдением стандартов форматирования, делать код читаемым и пригодным для сопровождения. **Владеет:** навыками написания самодокументируемого кода с комментариями на русском или английском языке.
ПК-8	Способен адаптировать и применять методы искусственного интеллекта и машинного обучения для решения	ИД-1пк-8	Выполняет предобработку и анализ данных (EDA, визуализация, кодирование, масштабирование) для подготовки датасета к	**Знает:** математические основы линейной регрессии и простых статистических алгоритмов (среднее, дисперсия, корреляция). **Умеет:** реализовыва

Код компетенции	Содержание компетенции	Код индикатора	Содержание индикатора	Дескриптор (планируемый результат обучения)
	профессиональных задач, включая интеграцию готовых моделей в программные продукты		обучению	ть на C++ простые статистические алгоритмы для анализа данных. **Владеет:** навыками вычисления базовых статистических характеристик числовых рядов.
ПК-10	Готов соблюдать правовые нормы, стандарты и требования информационной безопасности, участвовать в разработке нормативной и технической документации, учитывать законодательство в области ИТ	ИД-1пк-10	Применяет правовые нормы (лицензирование ПО, авторское право, ФЗ-152, ФЗ-149) и стандарты ИБ (ГОСТ, ISO 27001) при разработке ПО	

1.4. Объём самостоятельной работы

Общая трудоёмкость дисциплины - 5 зачётных единиц (180 часов), из которых на самостоятельную работу отводится 100–120 часов (в зависимости от формы обучения). Распределение по семестрам:

Семестр	Часы на СРС (по учебному плану)	Основные виды СРС
1	54	ЛР №1–19, теория

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

2.1. Организация работы с учебной литературой

Рекомендуемая последовательность изучения:

- Ознакомиться с оглавлением и структурой источника.
- Прочитать материал, выделяя ключевые определения и понятия.
- Составить краткий конспект (схемы, таблицы, примеры кода).
- Ответить на контрольные вопросы в конце раздела (если есть).
- Выполнить практические упражнения для закрепления.

Рекомендуемая литература:

	Источник	Разделы для СРС
	Страуструп Б. «Язык программирования C++» (4-е изд.)	Гл. 1–10, 12–18
	Прата С. «Язык программирования C++. Лекции и упражнения» (6-е изд.)	Гл. 2–9, 11–16
	Лафоре Р. «Объектно-ориентированное программирование в C++» (4-е изд.)	Гл. 1–8, 10–13

2.2. Конспектирование и реферирование

Рекомендации по конспектированию:

Используйте структурированный подход: тема → определения → синтаксис → примеры → типичные ошибки.

Выделяйте ключевые моменты цветом или маркерами.

Записывайте примеры кода с комментариями.

Составляйте сводные таблицы для сравнения (например, for/while/do-while).

Рисуйте схемы и диаграммы для визуализации алгоритмов.

Типичные ошибки:

1. Разыменование неинициализированного указателя.
2. Выход за границы массива при арифметике указателей.

2.3. Подготовка к практическим и лабораторным занятиям

Алгоритм подготовки к лабораторной работе:

Заблаговременно (за 3–5 дней) изучить теоретический материал по теме работы.

Ознакомиться с методическими указаниями и примерами.

Получить индивидуальный вариант задания у преподавателя.

Разработать алгоритм решения (блок-схема, псевдокод).

Написать программу, провести отладку и тестирование.

Оформить отчёт в соответствии с требованиями.

Подготовиться к защите (повторить теорию, разобрать контрольные вопросы).

2.4. Работа с электронными образовательными ресурсами

Рекомендуемые онлайн-ресурсы для СРС:

Ресурс	Адрес	Назначение
cppreference.com	https://en.cppreference.com	Полный справочник по C++
learncpp.com	https://www.learncpp.com	Интерактивные уроки C++
Replit	https://replit.com	Онлайн-редактор для быстрого тестирования

Ресурс	Адрес	Назначение
		кода
Godbolt	https://godbolt.org	Просмотр ассемблерного кода (для углублённого изучения)

Правила работы с онлайн-ресурсами:

Не копируйте готовый код без понимания - пишите сами.

Проверяйте актуальность информации (стандарт C++11/14/17/20).

Используйте несколько источников для перекрёстной проверки.

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

3.1. Порядок выполнения лабораторных работ

Этапы выполнения лабораторной работы:

Этап	Действие	Ориентировочное время
1	Изучение теоретического материала	2–3 часа
2	Разработка алгоритма (блок-схема)	0,5–1 час
3	Написание кода программы	2–3 часа
4	Отладка и тестирование	1–2 часа
5	Оформление отчёта	1 час
6	Подготовка к защите	0,5–1 час

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

4.1. Общие вопросы по дисциплине

Опишите жизненный цикл программы на C++ (препроцессинг, компиляция, линковка).

Сравните три вида циклов (for, while, do-while»). Приведите примеры использования.

В чём разница между статическим и динамическим массивом?

Объясните механизм передачи параметров по значению, по ссылке и по указателю.

Что такое указатель? В чём разница между * и &?

4.2. Вопросы по ООП

Назовите три основных принципа ООП. Раскройте каждый на примерах.

Чем отличается класс от объекта? Приведите аналогию.

Что такое конструктор и деструктор? Когда они вызываются?

Что такое наследование? Приведите пример иерархии классов.

Что такое полиморфизм? Как он реализуется в C++?

4.3. Вопросы по продвинутым темам

Что такое виртуальная таблица (vtable) и виртуальный указатель (vptr)?

Зачем нужен виртуальный деструктор?

Что такое перегрузка операторов? Приведите пример.

Что такое include guards и зачем они нужны?

В чём разница между #include <file> и #include "file"?

4.4. Методика собеседования

Собеседование проводится индивидуально или в малых группах (2–3 человека).

Продолжительность: 10–15 минут на студента.

Студент отвечает на 2–3 вопроса из списка (по выбору преподавателя или случайным образом).

Оценивается: полнота ответа, использование терминологии, способность привести примеры.

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

5.1. Критерии оценивания лабораторных работ

Шкала оценивания лабораторной работы:

Оценка	Уровень выполнения	Требования
5 (отлично)	Повышенный	Выполнены все три уровня (базовый, средний, повышенный); соблюдены все критерии качества; код защищён
4 (хорошо)	Средний	Выполнены базовый и средний уровни; соблюдены критерии среднего уровня; код защищён
3 (удовлетворительно)	Базовый	Выполнен только базовый уровень; соблюдены обязательные критерии; код защищён
2 (неудовлетворительно)	—	Базовый уровень не выполнен или грубые ошибки; критерии не соблюдены

Детализированные критерии (обобщённые):

Критерий	Базовый (3)	Средний (4)	Повышенный (5)
Выполненные уровни	Только базовый	Базовый + средний	Базовый + средний + повышенный
Компиляция без ошибок	✔	✔	✔
Корректная работа программы	✔	✔	✔
Правильный выбор типов данных	✔	✔	✔
Обработка ошибок ввода	—	✔	✔
Использование const (где нужно)	—	✔	✔
Комментарии в коде	—	✔	✔
Оформление отчёта по требованиям	✔	✔	✔
Успешная защита	✔	✔	✔

5.2. Критерии оценивания собеседования по литературе

Оценка	Характеристика ответа
5	Студент демонстрирует глубокое понимание

Оценка	Характеристика ответа
	материала, свободно оперирует терминологией, приводит собственные примеры, отвечает на дополнительные вопросы
4	Студент понимает материал, правильно отвечает на вопросы, но ответ недостаточно полный или есть небольшие неточности
3	Студент отвечает на вопросы в минимальном объёме, допускает ошибки в терминологии, путается в базовых понятиях
2	Студент не может ответить на вопросы или демонстрирует полное непонимание материала

5.3. Шкала перевода баллов в академическую оценку

При балльно-рейтинговой системе:

Баллы (накопленные)	Оценка (ECTS)	Оценка (традиционная)
90–100	A (отлично)	5 (отлично)
75–89	B–C (хорошо)	4 (хорошо)
60–74	D–E (удовлетворительно)	3 (удовлетворительно)
< 60	F (неудовлетворительно)	2 (неудовлетворительно)

5.4. Критерии допуска к зачёту/экзамену

Студент допускается к промежуточной аттестации (зачёту/экзамену) при выполнении следующих условий:

Выполнены и защищены не менее 70% лабораторных работ (минимум 14 из 19).

Каждая выполненная работа оценена не ниже «удовлетворительно» (3).

Пройдено собеседование по самостоятельно изученной литературе (оценка не ниже «удовлетворительно»).

Оформленные отчёты по всем защищённым работам сданы преподавателю.