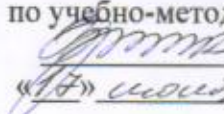


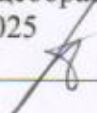
**ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ НОВОСИБИРСКОЙ ОБЛАСТИ
«Новосибирский электромеханический колледж»
(ГБПОУ НСО «НЭК»)**

УТВЕРЖДАЮ
заместитель директора
по учебно-методической работе
 Е.О. Кравченко
«17» июля 2025 г.

**Методические указания
по выполнению практических работ**
ПМ.01. Разработка модулей программного обеспечения для компьютерных
систем

МДК 01.01. Разработка программных модулей
для специальности 09.02.07 Информационные системы и
программирование

Разработчик: Комарова А.В.,
преподаватель

Рассмотрено
на заседании кафедры общеобразовательных дисциплин
Протокол №10 от 16.06.2025
Заведующий кафедрой  Белимова В.Г.

Методические указания по выполнению практических работ по МДК 01.01. Разработка программных модулей разработаны в соответствии с рабочей программой ПМ.01. Разработка модулей программного обеспечения для компьютерных систем по специальности 09.02.07 Информационные системы и программирование.

Организация-разработчик: ГБПОУ НСО «НЭК»

Разработчик:

Комарова А.В., преподаватель высшей квалификационной категории

Согласовано:

Методист



Рахметова Т.Ю.

17.06. 2025 г.

Оглавление

Пояснительная записка.....	4
Порядок выполнения практической и лабораторной работы	5
Рекомендации по оформлению практической и лабораторной работы	5
Критерии оценки практической и лабораторной работы	5
Перечень практических работ.....	6
ПРАКТИЧЕСКАЯ РАБОТА №1. Оценка сложности алгоритмов сортировки.	8
ПРАКТИЧЕСКАЯ РАБОТА №2. Оценка сложности алгоритмов поиска.	10
ПРАКТИЧЕСКАЯ РАБОТА №3. Оценка сложности рекурсивных алгоритмов.	15
ПРАКТИЧЕСКАЯ РАБОТА №4. Оценка сложности рекурсивных алгоритмов.	20
ПРАКТИЧЕСКАЯ РАБОТА №5. Оценка сложности эвристических алгоритмов.....	29
ПРАКТИЧЕСКАЯ РАБОТА №6. Работа с классами. Перегрузка методов.	40
ПРАКТИЧЕСКАЯ РАБОТА №7. Определение операций в классе. Создание наследованных классов. Работа с объектами через интерфейсы.	55
ПРАКТИЧЕСКАЯ РАБОТА №8. Использование стандартных интерфейсов. Работа с типом данных структура.	72
ПРАКТИЧЕСКАЯ РАБОТА №9. Коллекции. Параметризованные классы.	89
ПРАКТИЧЕСКАЯ РАБОТА №10. Использование регулярных выражений. Операции со списками.	112
ПРАКТИЧЕСКАЯ РАБОТА №11. Использование основных шаблонов.	118
ПРАКТИЧЕСКАЯ РАБОТА №12. Использование порождающих шаблонов.....	120
ПРАКТИЧЕСКАЯ РАБОТА №13. Использование структурных шаблонов.	125
ПРАКТИЧЕСКАЯ РАБОТА №14 -15. Использование поведенческих шаблонов.	135
ПРАКТИЧЕСКАЯ РАБОТА №16. Разработка приложения с использованием текстовых компонентов ..	150
ПРАКТИЧЕСКАЯ РАБОТА №17. Разработка приложения с несколькими формами.	161
ПРАКТИЧЕСКАЯ РАБОТА №18. Разработка приложения с не визуальными компонентами.	161
ПРАКТИЧЕСКАЯ РАБОТА №19-20. Разработка игрового приложения.	170
ПРАКТИЧЕСКАЯ РАБОТА №21. Разработка приложения с анимацией.	175
ПРАКТИЧЕСКАЯ РАБОТА №22-24. Оптимизация.	189
ПРАКТИЧЕСКАЯ РАБОТА №25-29. Рефакторинг кода.	206
ПРАКТИЧЕСКАЯ РАБОТА №30. Разработка интерфейса пользователя.	231
ПРАКТИЧЕСКАЯ РАБОТА №31. Разработка интерфейса пользователя.	233
ПРАКТИЧЕСКАЯ РАБОТА №32. Разработка интерфейса пользователя.	236
ПРАКТИЧЕСКАЯ РАБОТА №33. Разработка интерфейса пользователя.	241
ПРАКТИЧЕСКАЯ РАБОТА №34. Разработка интерфейса пользователя.	246
Практическая работа 30-34. Разработка интерфейсов (Альтернатива)	251
ПРАКТИЧЕСКАЯ РАБОТА №35-36. Создание приложения с БД.....	261
ПРАКТИЧЕСКАЯ РАБОТА №37-40. Создание запросов к БД. Создание хранимых процедур	287
Приложение 1. Пример выполнения отчета.....	291

Пояснительная записка

Методические указания по выполнению лабораторных и практических работ по МДК Разработка программных модулей разработаны в соответствии с рабочей программой профессионального модуля и предназначены для приобретения необходимых практических навыков и закрепления теоретических знаний, полученных обучающимися при изучении профессионального модуля, обобщения и систематизации знаний перед экзаменом.

Методические указания предназначены для обучающихся специальности 09.02.07 Информационные системы и программирование.

Освоение содержания МДК 01.01. Разработка программных модулей во время выполнения практических работ обеспечивает достижение обучающимися следующих **результатов:**

Код	Наименование общих компетенций
ОК 1.	Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам
ОК 2.	Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности.
ОК 3	Планировать и реализовывать собственное профессиональное и личностное развитие.
ОК 4	Планировать и реализовывать собственное профессиональное и личностное развитие.
ОК 5	Планировать и реализовывать собственное профессиональное и личностное развитие.
ОК 6	Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей, применять стандарты антикоррупционного поведения.
ОК 7	Содействовать сохранению окружающей среды, ресурсосбережению, эффективно действовать в чрезвычайных ситуациях.
ОК 8	Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности
ОК 9	Использовать информационные технологии в профессиональной деятельности.
ОК 10	Пользоваться профессиональной документацией на государственном и иностранном языках
ОК 11	Использовать знания по финансовой грамотности, планировать предпринимательскую деятельность в профессиональной сфере
ПК 1.1.	Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием
ПК 1.2.	Разрабатывать программные модули в соответствии с техническим заданием
ПК 1.3.	Выполнять отладку программных модулей с использованием специализированных программных средств
ПК 1.4.	Выполнять тестирование программных модулей
ПК 1.5.	Осуществлять рефакторинг и оптимизацию программного кода

В результате освоения профессионального модуля обучающийся должен:

Иметь практический опыт	В разработке кода программного продукта на основе готовой спецификации на уровне модуля; использовании инструментальных средств на этапе отладки программного продукта; проведении тестирования программного модуля по определенному сценарию; использовании инструментальных средств на этапе отладки программного продукта; разработке мобильных приложений
Уметь	осуществлять разработку кода программного модуля на языках низкого и высокого уровней; создавать программу по разработанному алгоритму как отдельный модуль; выполнять отладку и тестирование программы на уровне модуля; осуществлять разработку кода программного модуля на современных языках программирования; уметь выполнять оптимизацию и рефакторинг программного кода; оформлять документацию на программные средства
Знать	основные этапы разработки программного обеспечения; основные принципы технологии структурного и объектно-ориентированного программирования; способы оптимизации и приемы рефакторинга; основные принципы отладки и тестирования программных продуктов

В соответствии с рабочей программой по ПМ 01. Разработка модулей программного обеспечения для компьютерных систем, практические работы по МДК 01.01. Разработка программных модулей проводятся в четверном, пятом, шестом семестре. Целесообразность данной группировки обусловлена необходимостью обобщения и систематизации знаний перед экзаменом.

Рабочая программа профессионального модуля предусматривает проведение практических работ МДК 01.01. Разработка программных модулей в объеме 80 часов.

Порядок выполнения практической и лабораторной работы

- записать название работы, ее цель в тетрадь;
- выполнить основные задания в соответствии с ходом работы;
- выполнить индивидуальные задания.

Рекомендации по оформлению практической и лабораторной работы

Задания выполняются обучающимися по шагам. Необходимо строго придерживаться порядка действий, описанного в практической работе

Результаты выполнения практических работ необходимо сохранять в своей папке на компьютере или USB – накопителе.

В случае пропуска занятий обучающийся осваивает материал самостоятельно в свободное от занятий время и сдает практическую работу с пояснениями о выполнении.

Критерии оценки практической и лабораторной работы

- наличие Цель выполняемой работы, выполнение более половины основных заданий (удовлетворительно);
- наличие Цель выполняемой работы, выполнение всех основных и более половины дополнительных заданий (хорошо);
- наличие Цель выполняемой работы, выполнение всех основных и индивидуальных заданий (отлично).

Перечень практических работ

№	Наименование разделов и тем профессионального модуля (ПМ)	Наименование лабораторных работ и практических занятий	Объем часов
	Раздел 1. Разработка программных модулей		
	МДК 01.01. Разработка программных модулей		
1.	Тема 1.1.2 Структурное программирование	ПРАКТИЧЕСКАЯ РАБОТА №1. Оценка сложности алгоритмов сортировки.	2
2.		ПРАКТИЧЕСКАЯ РАБОТА №2. Оценка сложности алгоритмов поиска.	2
3.		ПРАКТИЧЕСКАЯ РАБОТА №3. Оценка сложности рекурсивных алгоритмов.	2
4.		ПРАКТИЧЕСКАЯ РАБОТА №4. Оценка сложности рекурсивных алгоритмов.	2
5.		ПРАКТИЧЕСКАЯ РАБОТА №5. Оценка сложности эвристических алгоритмов.	2
6.	Тема 1.1.3 Объектно-ориентированное программирование	ПРАКТИЧЕСКАЯ РАБОТА №6. Работа с классами. Перегрузка методов.	2
7.		ПРАКТИЧЕСКАЯ РАБОТА №7. Определение операций в классе. Создание наследованных классов. Работа с объектами через интерфейсы.	2
8.		ПРАКТИЧЕСКАЯ РАБОТА №8. Использование стандартных интерфейсов. Работа с типом данных структура.	2
9.		ПРАКТИЧЕСКАЯ РАБОТА №9. Коллекции. Параметризованные классы.	2
10.		ПРАКТИЧЕСКАЯ РАБОТА №10. Использование регулярных выражений. Операции со списками.	2
11.	Тема 1.1.4 Паттерны проектирования	ПРАКТИЧЕСКАЯ РАБОТА №11. Использование основных шаблонов.	2
12.		ПРАКТИЧЕСКАЯ РАБОТА №12. Использование порождающих шаблонов.	2
13.		ПРАКТИЧЕСКАЯ РАБОТА №13. Использование структурных шаблонов.	2
14.		ПРАКТИЧЕСКАЯ РАБОТА №14. Использование поведенческих шаблонов.	2
15.		ПРАКТИЧЕСКАЯ РАБОТА №15. Использование поведенческих шаблонов.	2
16.	Тема 1.1.5 Событийно-управляемое программирование	ПРАКТИЧЕСКАЯ РАБОТА №16. Разработка приложения с использованием текстовых компонентов	2
17.		ПРАКТИЧЕСКАЯ РАБОТА №17. Разработка приложения с несколькими формами.	2
18.		ПРАКТИЧЕСКАЯ РАБОТА №18. Разработка приложения с не визуальными компонентами.	2
19.		ПРАКТИЧЕСКАЯ РАБОТА №19. Разработка игрового приложения.	2
20.		ПРАКТИЧЕСКАЯ РАБОТА №20. Разработка игрового приложения.	2
21.		ПРАКТИЧЕСКАЯ РАБОТА №21. Разработка приложения с анимацией.	2
22.	Тема 1.1.6 Оптимизация и рефакторинг кода	ПРАКТИЧЕСКАЯ РАБОТА №22. Оптимизация и рефакторинг кода.	2
23.		ПРАКТИЧЕСКАЯ РАБОТА №23. Оптимизация и рефакторинг кода.	2
24.		ПРАКТИЧЕСКАЯ РАБОТА №24. Оптимизация и	2

		рефакторинг кода.	
25.		ПРАКТИЧЕСКАЯ РАБОТА №25. Оптимизация и рефакторинг кода.	2
26.		ПРАКТИЧЕСКАЯ РАБОТА №26. Оптимизация и рефакторинг кода.	2
27.		ПРАКТИЧЕСКАЯ РАБОТА №27. Оптимизация и рефакторинг кода.	2
28.		ПРАКТИЧЕСКАЯ РАБОТА №28. Оптимизация и рефакторинг кода.	2
29.		ПРАКТИЧЕСКАЯ РАБОТА №29. Оптимизация и рефакторинг кода.	2
30.	Тема 1.1.7 Разработка пользовательского интерфейса	ПРАКТИЧЕСКАЯ РАБОТА №30. Разработка интерфейса пользователя.	2
31.		ПРАКТИЧЕСКАЯ РАБОТА №31. Разработка интерфейса пользователя.	2
32.		ПРАКТИЧЕСКАЯ РАБОТА №32. Разработка интерфейса пользователя.	2
33.		ПРАКТИЧЕСКАЯ РАБОТА №33. Разработка интерфейса пользователя.	2
34.		ПРАКТИЧЕСКАЯ РАБОТА №34. Разработка интерфейса пользователя.	2
35.	Тема 1.1.8 Основы ADO.Net	ПРАКТИЧЕСКАЯ РАБОТА №35. Создание приложения с БД	2
36.		ПРАКТИЧЕСКАЯ РАБОТА №36. Создание приложения с БД	2
37.		ПРАКТИЧЕСКАЯ РАБОТА №37. Создание запросов к БД	2
38.		ПРАКТИЧЕСКАЯ РАБОТА №38. Создание запросов к БД	2
39.		ПРАКТИЧЕСКАЯ РАБОТА №39. Создание хранимых процедур	2
40.		ПРАКТИЧЕСКАЯ РАБОТА №40. Создание хранимых процедур	2
		Итого:	80

ПРАКТИЧЕСКАЯ РАБОТА №1. Оценка сложности алгоритмов сортировки.

Тема: Оценка сложности алгоритмов сортировки (Пузырьковая сортировка и Сортировка выбором)

Цель работы: Освоить принципы работы и реализации алгоритмов сортировки, научиться оценивать их временную сложность на практике.

Задачи:

1. Изучить алгоритмы пузырьковой сортировки (Bubble Sort) и сортировки выбором (Selection Sort).
2. Реализовать оба алгоритма на языке C++.
3. Экспериментально сравнить время их работы на массивах разного размера.
4. (Для Qt) Создать графический интерфейс для визуализации процесса сортировки или сравнения времени.

Теоретический материал (Сокращенный вариант)

- **Пузырьковая сортировка:** Алгоритм многократно проходит по массиву, сравнивая и меняя местами соседние элементы, если они находятся в неправильном порядке. Проходы повторяются до тех пор, пока массив не будет отсортирован.
 - **Сложность:** $O(n^2)$ в худшем и среднем случае, $O(n)$ в лучшем (уже отсортированный массив).
- **Сортировка выбором:** Алгоритм на каждом проходе находит минимальный (или максимальный) элемент в неотсортированной части массива и меняет его местами с первым элементом неотсортированной части.
 - **Сложность:** Всегда $O(n^2)$.

Пример решения (C++ консоль)

cpp

```
#include <iostream>
#include <vector>
#include <chrono>
```

```
void bubbleSort(std::vector<int>& arr) {
    // ... (код из шаблона выше)
}
```

```
void selectionSort(std::vector<int>& arr) {
    int n = arr.size();
    for (int i = 0; i < n-1; i++) {
        int min_idx = i;
        for (int j = i+1; j < n; j++) {
            if (arr[j] < arr[min_idx]) min_idx = j;
        }
        std::swap(arr[min_idx], arr[i]);
    }
}
```

```
int main() {
    std::vector<int> data1 = {64, 34, 25, 12, 22, 11, 90};
    std::vector<int> data2 = data1; // Копия для второго алгоритма
```

```
    auto start = std::chrono::high_resolution_clock::now();
    bubbleSort(data1);
    auto end = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(end - start);
    std::cout << "Bubble Sort time: " << duration.count() << " microseconds\n";
```

```

start = std::chrono::high_resolution_clock::now();
selectionSort(data2);
end = std::chrono::high_resolution_clock::now();
duration = std::chrono::duration_cast<std::chrono::microseconds>(end - start);
std::cout << "Selection Sort time: " << duration.count() << " microseconds\n";

return 0;
}

```

Задания для выполнения

Вариант 1 (Консольный):

1. Реализуйте функции bubbleSort и selectionSort.
2. Напишите программу, которая генерирует три массива размером 1000, 10000 и 100000 случайных целых чисел.
3. Для каждого массива замерьте и выведите в консоль время работы каждого алгоритма сортировки. Каждый замер проведите 5 раз, возьмите среднее значение.
4. Составьте таблицу результатов и сделайте вывод о соответствии экспериментальных данных теоретической сложности $O(n^2)$.

Вариант 2 (Qt):

1. Выполните пункты 1-3 из Варианта 1.
2. Создайте приложение Qt Widgets с интерфейсом, содержащим:
 - QSpinBox для выбора размера генерируемого массива (N).
 - Кнопку "Сгенерировать и отсортировать".
 - QTextEdit для вывода исходного и отсортированного массивов (для $N \leq 20$).
 - Два QLabel для вывода времени выполнения каждого алгоритма.
3. Реализуйте логику: по нажатию кнопки генерируется массив, выполняются обе сортировки, выводятся результаты.
4. *Дополнительно:* Добавьте QProgressBar для индикации процесса сортировки (используйте QApplication::processEvents() внутри циклов сортировки для обновления GUI).

Критерии оценивания (Адаптированные)

- «Удовлетворительно»: Реализованы оба алгоритма, консольная программа работает, производит замеры времени.
- «Хорошо»: Консольная программа работает корректно, включает многократные замеры и вывод осмысленной таблицы. Код чистый, с комментариями.
- «Отлично»: Выполнены условия на "хорошо" и создано полностью рабочее Qt-приложение с графическим интерфейсом по заданию Варианта 2.

ПРАКТИЧЕСКАЯ РАБОТА №2. Оценка сложности алгоритмов поиска.

Цель работы: Освоить принципы работы и реализации базовых алгоритмов поиска, научиться оценивать их временную и пространственную сложность, сравнить эффективность линейного и бинарного поиска на практике.

Задачи:

1. Изучить алгоритмы линейного (последовательного) и бинарного (двоичного) поиска.
2. Реализовать оба алгоритма на языке C++.
3. Экспериментально сравнить время их работы на массивах разного размера и при различных сценариях (лучший, средний, худший случай).
4. (Для Qt) Создать графический интерфейс для визуализации шагов поиска или сравнения производительности.

Теоретический материал

1. Линейный поиск (Linear Search)

- **Принцип работы:** Последовательное сравнение каждого элемента массива с искомым значением (key) до тех пор, пока элемент не будет найден или не будет достигнут конец массива.

- **Алгоритм (псевдокод):**

text

Для каждого индекса i от 0 до $n-1$:

 Если $\text{array}[i] == \text{key}$:

 Вернуть i // Элемент найден

Вернуть -1 // Элемент не найден

- **Сложность:**

- **Временная:** $O(n)$ — в худшем и среднем случае (элемент отсутствует или находится в конце). $O(1)$ — в лучшем случае (элемент первый).

- **Пространственная:** $O(1)$ — не требует дополнительной памяти.

2. Бинарный поиск (Binary Search)

- **Принцип работы:** Эффективный алгоритм для поиска в **отсортированном** массиве. На каждом шаге интервал поиска делится пополам. Искомое значение сравнивается со средним элементом. Если значение равно среднему элементу — поиск завершен. Если значение меньше — поиск продолжается в левой половине, если больше — в правой.

- **Алгоритм (псевдокод, итеративный):**

text

Установить $\text{left} = 0$, $\text{right} = n - 1$

Пока $\text{left} \leq \text{right}$:

$\text{mid} = \text{left} + (\text{right} - \text{left}) / 2$ // Избегаем переполнения

 Если $\text{array}[\text{mid}] == \text{key}$:

 Вернуть mid

 Иначе если $\text{array}[\text{mid}] < \text{key}$:

$\text{left} = \text{mid} + 1$

 Иначе:

$\text{right} = \text{mid} - 1$

Вернуть -1

- **Сложность:**

- **Временная:** $O(\log n)$ — в худшем и среднем случае. $O(1)$ — в лучшем случае (элемент в середине).

- **Пространственная (итеративная версия):** $O(1)$. Рекурсивная версия имеет сложность $O(\log n)$ из-за стека вызовов.

- **Важное условие:** Массив должен быть отсортирован.

3. Сравнение алгоритмов

Критерий	Линейный поиск	Бинарный поиск
Условие применения	Любой массив	Только отсортированный массив
Временная сложность	$O(n)$	$O(\log n)$
Эффективность	Низкая для больших массивов	Очень высокая для больших массивов
Реализация	Очень простая	Сложнее, требует контроля границ

Пример решения (Консольное приложение на C++)

```
cpp
#include <iostream>
#include <vector>
#include <algorithm> // для std::sort
#include <chrono>
#include <random>

// 1. Линейный поиск (возвращает индекс или -1)
int linearSearch(const std::vector<int>& arr, int key) {
    for (size_t i = 0; i < arr.size(); ++i) {
        if (arr[i] == key) {
            return static_cast<int>(i);
        }
    }
    return -1;
}

// 2. Бинарный поиск (итеративный, для отсортированного массива)
int binarySearch(const std::vector<int>& arr, int key) {
    int left = 0;
    int right = static_cast<int>(arr.size()) - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2; // Предотвращает переполнение

        if (arr[mid] == key) {
            return mid;
        } else if (arr[mid] < key) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1;
}

int main() {
```

```

// Генерация случайного массива
std::random_device rd;
std::mt19937 gen(rd());
std::uniform_int_distribution<> dis(1, 10000);

const int SIZE = 10000;
std::vector<int> data(SIZE);
for (int& val : data) {
    val = dis(gen);
}

// Ключ для поиска (возьмем элемент из середины для демонстрации)
int searchKey = data[SIZE / 2];

// --- Линейный поиск ---
auto start = std::chrono::high_resolution_clock::now();
int linearResult = linearSearch(data, searchKey);
auto end = std::chrono::high_resolution_clock::now();
auto linearTime = std::chrono::duration_cast<std::chrono::microseconds>(end - start);

std::cout << "=== Линейный поиск ===\n";
std::cout << "Ключ: " << searchKey << "\n";
std::cout << "Индекс: " << linearResult << "\n";
std::cout << "Время: " << linearTime.count() << " мкс\n";

// --- Бинарный поиск ---
// Бинарный поиск требует отсортированный массив!
std::vector<int> sortedData = data; // Копируем исходные данные
std::sort(sortedData.begin(), sortedData.end()); // Сортируем копию

start = std::chrono::high_resolution_clock::now();
int binaryResult = binarySearch(sortedData, searchKey);
end = std::chrono::high_resolution_clock::now();
auto binaryTime = std::chrono::duration_cast<std::chrono::microseconds>(end - start);

std::cout << "\n=== Бинарный поиск ===\n";
std::cout << "Ключ: " << searchKey << "\n";
std::cout << "Индекс (в отсортированной копии): " << binaryResult << "\n";
std::cout << "Время (с учетом сортировки - НЕТ!): " << binaryTime.count() << " мкс\n";
std::cout << "Примечание: Время только на сам поиск. Сортировка массива (" << SIZE
    << " элементов) заняла бы дополнительно ~O(n log n) времени.\n";

// Сравнение
std::cout << "\n=== Сравнение времени поиска (только поиск!) ===\n";
std::cout << "Линейный / Бинарный: " << static_cast<double>(linearTime.count()) /
    binaryTime.count()
    << " раз\n";

return 0;
}

```

Пояснение к примеру:

1. `linearSearch`: Простой цикл `for` с проверкой на равенство.

2. **binarySearch:** Используется итеративный подход с переменными `left`, `right`, `mid`. Важно вычислять `mid` как $\text{left} + (\text{right} - \text{left}) / 2$, чтобы избежать потенциального переполнения при больших значениях `left` и `right`.
3. **Важный нюанс:** Время бинарного поиска измеряется только на этапе поиска в **уже отсортированном** массиве. В реальной задаче необходимо учитывать стоимость предварительной сортировки ($O(n \log n)$), которую бинарный поиск требует, а линейный — нет.

Задания для выполнения

Вариант 1 (Консольный - Анализ сложности):

1. Реализуйте функции `linearSearch` и `binarySearch` (итеративную версию), как в примере.
2. Напишите программу, которая:
 - Генерирует **отсортированный** массив целых чисел размером `N` (например, 1000, 10 000, 100 000). Можно генерировать последовательные числа или использовать `std::sort`.
 - Для каждого `N` выполняет **1000** поисков:
 - 333 раза ищет элемент в **начале** массива (лучший случай для линейного, средний для бинарного).
 - 333 раза ищет элемент в **конце** массива (худший для линейного, средний для бинарного).
 - 334 раза ищет **отсутствующий** элемент (худший случай для обоих).
3. Замерьте **суммарное** время выполнения всех 1000 поисков для каждого алгоритма и каждого `N`.
4. Результаты представьте в таблице. Сделайте выводы: подтверждаются ли теоретические оценки $O(n)$ и $O(\log n)$? В каких сценариях линейный поиск может быть предпочтительнее?

Вариант 2 (Qt - Визуализация и приложение):

1. Реализуйте функции линейного и бинарного поиска.
2. Создайте приложение Qt Widgets с интерфейсом, содержащим:
 - `QSpinBox` для задания размера массива (`N`, до 50 для наглядности).
 - Кнопку "Сгенерировать массив", которая заполняет `std::vector<int>` случайными числами и выводит их в `QListWidget` или `QTextEdit`.
 - Кнопку "Отсортировать массив", которая сортирует вектор и обновляет его отображение.
 - `QLineEdit` для ввода искомого ключа.
 - Две кнопки "Линейный поиск" и "Бинарный поиск".
 - `QLabel` для вывода результата (найден/не найден, индекс, количество шагов).
 - `QTextEdit` для пошагового лога (опционально, для визуализации: выводите текущие `left`, `right`, `mid` для бинарного или текущий индекс для линейного).
3. Реализуйте логику:
 - При нажатии на кнопку поиска запускается соответствующий алгоритм.
 - **Визуализация:** Во время поиска подсвечивайте текущий проверяемый элемент в `QListWidget` (используйте `QTimer` с небольшой задержкой `QThread::msleep(100)` или `QElapsedTimer`, чтобы анимация была заметна).
 - Выводите количество сравнений (шагов), сделанных каждым алгоритмом.
4. *Дополнительное задание:* Добавьте возможность выбора "худшего/лучшего/среднего" случая для генерации ключа автоматически.

Критерии оценивания

- «Удовлетворительно» (3):
 - Реализованы работающие функции линейного и бинарного поиска.

- Консольная программа (Вариант 1) или Qt-приложение (Вариант 2) компилируется и выполняет базовый поиск.
 - В отчете приведены теоретические выкладки по сложности алгоритмов.
 - **«Хорошо» (4):**
 - Для **Варианта 1:** Программа корректно проводит серию экспериментов для разных N и сценариев, результаты представлены в четкой таблице. Код хорошо структурирован.
 - Для **Варианта 2:** Qt-приложение полностью функционально: генерация, сортировка, оба поиска, вывод результата. Интерфейс интуитивно понятен.
 - В отчете присутствует анализ результатов, подтверждающий/объясняющий теоретическую сложность.
 - **«Отлично» (5):**
 - Для **Варианта 1:** Выполнен полный анализ, включая построение простого графика зависимости времени от N (можно вывести в консоль данные для построения в Excel/Python). Проведено сравнение с `std::binary_search`. Дан глубокий вывод о практической применимости.
 - Для **Варианта 2:** Реализована **визуализация шагов поиска** с подсветкой и задержкой. Добавлен счетчик шагов и их пошаговый лог. Приложение устойчиво обрабатывает ввод пользователя.
 - **Общее:** Отчет содержит подробное описание алгоритмов, скриншоты работающих программ, обоснованные выводы.
- Советы по выполнению:**
- Для точных замеров времени в C++ используйте `<chrono>` (как в примере). Выполняйте каждый замер несколько раз, усредняйте.
 - В Qt для визуализации не блокируйте основной поток долгими циклами. Используйте `QTimer` или выносите поиск в отдельный поток (`QThread`, `QtConcurrent`).
 - При реализации бинарного поиска будьте внимательны с граничными условиями (`left <= right` или `left < right`), это частый источник ошибок.
 - Не забывайте, что бинарный поиск применяется только к отсортированным данным. В вашей программе это должно быть четко отражено.

ПРАКТИЧЕСКАЯ РАБОТА №3. Оценка сложности рекурсивных алгоритмов.

Цель работы: Освоить принципы разработки и анализа рекурсивных алгоритмов, научиться оценивать их временную и пространственную сложность, понять особенности реализации рекурсии на языке C++.

Задачи:

1. Изучить понятие рекурсии, базовый случай и рекурсивный вызов.
2. Разработать рекурсивные алгоритмы для решения классических задач (числа Фибоначчи, факториал, Ханойские башни).
3. Проанализировать временную сложность рекурсивных алгоритмов с помощью рекуррентных соотношений.
4. Сравнить рекурсивную и итеративную реализации по времени выполнения и использованию памяти.
5. (Для Qt) Создать визуализацию работы рекурсивного алгоритма.

Теоретический материал

1. Основные понятия рекурсии

- **Рекурсия** — способ определения функции, при котором функция вызывает саму себя непосредственно или через другие функции.
- **Базовый случай** — условие, при котором рекурсия прекращается. Без корректного базового случая возникает бесконечная рекурсия и переполнение стека.
- **Рекурсивный случай** — часть функции, где происходит вызов самой себя с измененными параметрами.
- **Глубина рекурсии** — количество вложенных вызовов функции до достижения базового случая.

2. Примеры рекурсивных функций

Факториал:

text

$n! = n * (n-1)!$, где $0! = 1$ (базовый случай)

Сложность: $O(n)$

Числа Фибоначчи:

text

$F(n) = F(n-1) + F(n-2)$, где $F(0) = 0$, $F(1) = 1$ (базовые случаи)

Сложность: $O(2^n)$ для наивной реализации

Ханойские башни:

text

Переместить n дисков с стержня A на стержень C:

1. Переместить $n-1$ дисков с A на B
2. Переместить самый большой диск с A на C
3. Переместить $n-1$ дисков с B на C

Сложность: $O(2^n)$

3. Анализ сложности рекурсивных алгоритмов

Для анализа используется метод **рекуррентных соотношений**:

- **Факториал:** $T(n) = T(n-1) + O(1) \Rightarrow T(n) = O(n)$
- **Наивный Фибоначчи:** $T(n) = T(n-1) + T(n-2) + O(1) \Rightarrow T(n) = O(2^n)$
- **Ханойские башни:** $T(n) = 2T(n-1) + O(1) \Rightarrow T(n) = O(2^n)$

4. Проблемы рекурсии в C++

- **Переполнение стека** — при большой глубине рекурсии.
- **Избыточные вычисления** — как в наивной реализации чисел Фибоначчи.
- **Решение:** использование мемоизации (кэширования), итеративных алгоритмов или хвостовой рекурсии (которую компиляторы C++ не всегда оптимизируют).

Пример решения (Консольное приложение на C++)

```

cpp
#include <iostream>
#include <chrono>
#include <map>
#include <vector>

// 1. Рекурсивный факториал
unsigned long long factorialRecursive(int n) {
    if (n <= 1) return 1; // Базовый случай
    return n * factorialRecursive(n - 1); // Рекурсивный случай
}

// 2. Итеративный факториал (для сравнения)
unsigned long long factorialIterative(int n) {
    unsigned long long result = 1;
    for (int i = 2; i <= n; ++i) {
        result *= i;
    }
    return result;
}

// 3. Наивный рекурсивный Фибоначчи (очень неэффективный!)
int fibonacciNaive(int n) {
    if (n <= 1) return n; // Базовый случай: F(0)=0, F(1)=1
    return fibonacciNaive(n - 1) + fibonacciNaive(n - 2);
}

// 4. Фибоначчи с мемоизацией (кэшированием результатов)
int fibonacciMemo(int n, std::map<int, int>& cache) {
    if (n <= 1) return n;

    // Проверяем, есть ли результат в кэше
    if (cache.find(n) != cache.end()) {
        return cache[n];
    }

    // Вычисляем и сохраняем в кэш
    cache[n] = fibonacciMemo(n - 1, cache) + fibonacciMemo(n - 2, cache);
    return cache[n];
}

// 5. Ханойские башни (рекурсивная реализация)
void hanoiTowers(int n, char from, char to, char aux, int& moveCount) {
    if (n == 1) {
        // Базовый случай: переместить один диск
        // std::cout << "Move disk 1 from " << from << " to " << to << std::endl;
        moveCount++;
        return;
    }

    // Рекурсивные шаги
    hanoiTowers(n - 1, from, aux, to, moveCount);

```

```

// std::cout << "Move disk " << n << " from " << from << " to " << to << std::endl;
moveCount++;
hanoiTowers(n - 1, aux, to, from, moveCount);
}

int main() {
    std::cout << "=== Рекурсивные алгоритмы: анализ сложности ===\n\n";

    // Тест факториала
    int n = 15;

    auto start = std::chrono::high_resolution_clock::now();
    unsigned long long factRec = factorialRecursive(n);
    auto end = std::chrono::high_resolution_clock::now();
    auto timeRec = std::chrono::duration_cast<std::chrono::nanoseconds>(end - start);

    start = std::chrono::high_resolution_clock::now();
    unsigned long long factIter = factorialIterative(n);
    end = std::chrono::high_resolution_clock::now();
    auto timeIter = std::chrono::duration_cast<std::chrono::nanoseconds>(end - start);

    std::cout << "Факториал " << n << ":\n";
    std::cout << " Рекурсивно: " << factRec << " (" << timeRec.count() << " нс)\n";
    std::cout << " Итеративно: " << factIter << " (" << timeIter.count() << " нс)\n";

    // Тест Фибоначчи
    std::cout << "\nЧисла Фибоначчи:\n";

    // Наивная реализация (очень медленно для n > 40)
    int fibN = 15; // Меньшее значение для демонстрации
    start = std::chrono::high_resolution_clock::now();
    int fibNaive = fibonacciNaive(fibN);
    end = std::chrono::high_resolution_clock::now();
    auto timeNaive = std::chrono::duration_cast<std::chrono::microseconds>(end - start);
    std::cout << " F(" << fibN << ") наивно: " << fibNaive << " (" << timeNaive.count() << "
мс)\n";

    // С мемоизацией
    std::map<int, int> cache;
    start = std::chrono::high_resolution_clock::now();
    int fibMemo = fibonacciMemo(fibN, cache);
    end = std::chrono::high_resolution_clock::now();
    auto timeMemo = std::chrono::duration_cast<std::chrono::microseconds>(end - start);
    std::cout << " F(" << fibN << ") с мемоизацией: " << fibMemo << " (" <<
timeMemo.count() << " мкс)\n";

    // Тест Ханойских башен
    std::cout << "\nХанойские башни:\n";
    for (int disks = 1; disks <= 5; disks++) {
        int moves = 0;
        start = std::chrono::high_resolution_clock::now();
        hanoiTowers(disks, 'A', 'C', 'B', moves);
    }
}

```

```

end = std::chrono::high_resolution_clock::now();
auto timeHanoi = std::chrono::duration_cast<std::chrono::microseconds>(end - start);

std::cout << " " << disks << " дисков: " << moves << " ходов";
std::cout << " (" << timeHanoi.count() << " мкс)" << std::endl;
}

// Анализ роста количества вызовов для Фибоначчи
std::cout << "\nАнализ роста вызовов для FibonacciNaive:\n";
std::cout << "\tВызовы функции (примерно 2^(n+1)-1)\n";
for (int i = 1; i <= 10; i++) {
    // Подсчет вызовов можно сделать с помощью глобальной переменной
    // или статической переменной внутри функции
    std::cout << i << "\t~" << (1 << (i+1)) - 1 << std::endl;
}

return 0;
}

```

Задания для выполнения

Вариант 1 (Консольный - Анализ сложности):

1. Реализуйте функции:
 - Рекурсивный и итеративный алгоритмы вычисления **суммы цифр** числа.
 - Рекурсивный алгоритм **быстрого возведения в степень** (a^n).
 - Рекурсивный алгоритм поиска **наибольшего общего делителя (НОД)** по алгоритму Евклида.
2. Для каждой функции определите:
 - Базовый и рекурсивный случаи.
 - Рекуррентное соотношение для временной сложности.
 - Теоретическую оценку сложности (O-нотация).
3. Напишите программу, которая:
 - Замеряет время выполнения каждой рекурсивной функции и её итеративного аналога (где применимо).
 - Для суммы цифр: протестируйте на числах разной длины (10^3 , 10^6 , 10^9 цифр).
 - Для быстрого возведения: протестируйте на разных основаниях и показателях (2^{10} , 2^{20} , 2^{30}).
 - Для НОД: протестируйте на парах чисел Фибоначчи $F(n)$ и $F(n+1)$.
4. Постройте таблицу результатов и графики зависимости времени от размера входа. Сделайте выводы о практической применимости рекурсии в каждом случае.

Вариант 2 (Qt - Визуализация рекурсии):

1. Реализуйте рекурсивный алгоритм для построения **фрактала "Снежинка Коха"** или **дерева Пифагора**.
2. Создайте приложение Qt Widgets с интерфейсом, содержащим:
 - Виджет для рисования (наследник `QWidget` с переопределённым `paintEvent`).
 - `QSpinBox` для задания глубины рекурсии (уровня детализации фрактала).
 - `QPushButton` для запуска построения.
 - `QLabel` для вывода информации (количество рекурсивных вызовов, время выполнения).
 - `QCheckBox` для выбора анимации построения (пошаговое отображение).
3. Реализуйте логику:
 - При изменении глубины рекурсии или нажатии кнопки фрактал перерисовывается.
 - Реализуйте анимацию: используйте `QTimer` для последовательной отрисовки уровней рекурсии.

- Подсчитывайте количество рекурсивных вызовов функции рисования.
4. *Дополнительно:*
- Добавьте возможность изменения цвета, угла поворота, коэффициента масштабирования.
 - Реализуйте рекурсивный алгоритм решения задачи о 8 ферзях с визуализацией расстановки.

Критерии оценивания

«Удовлетворительно» (3):

- Реализованы все требуемые рекурсивные функции.
- Программа компилируется и демонстрирует базовую работоспособность.
- В отчете приведены рекуррентные соотношения для каждой функции.

«Хорошо» (4):

- Для **Варианта 1**: Проведены корректные замеры времени, результаты представлены в таблицах. Реализованы итеративные аналоги для сравнения. Сделаны верные выводы о сложности.
- Для **Варианта 2**: Qt-приложение полностью функционально, фрактал отрисовывается корректно для разных уровней рекурсии. Интерфейс интуитивен.
- В отчете присутствует анализ результатов, объяснение выбора базовых случаев.

«Отлично» (5):

- Для **Варианта 1**: Выполнен полный анализ с построением графиков. Исследованы граничные случаи (например, очень большая глубина рекурсии). Проанализированы затраты памяти (размер стека). Предложены и реализованы оптимизации (мемоизация, хвостовая рекурсия).
- Для **Варианта 2**: Реализована плавная анимация построения фрактала. Добавлены дополнительные параметры настройки. Реализована визуализация стека вызовов (например, в отдельном виджете показывается текущая цепочка рекурсивных вызовов). Или дополнительно реализована задача о 8 ферзях.
- **Общее:** Отчет содержит детальный анализ сложности, сравнение рекурсивного и итеративного подходов, рекомендации по применению. Представлены скриншоты, листинги кода, таблицы измерений.

Советы по выполнению:

1. Для факториала и Фибоначчи: Учитывайте переполнение целочисленных типов. Используйте `unsigned long long` или большие числа.
2. Для анализа глубины рекурсии: Можно добавить статическую переменную-счетчик или передавать параметр глубины.
3. В Qt для анимации: Не выполняйте длительные рекурсивные вычисления в основном потоке GUI. Используйте:

```
сpp
// Для пошаговой отрисовки
QTimer::singleShot(100, this, [this]() {
    // один шаг рекурсии
    update(); // перерисовать
});
```
4. Для мемоизации: Используйте `std::unordered_map` для более эффективного поиска.
5. При анализе сложности: Учитывайте, что вызов функции в C++ имеет накладные расходы (сохранение контекста в стеке), что делает рекурсию менее эффективной, чем итерацию для простых алгоритмов.
6. Опасайтесь переполнения стека: Для глубокой рекурсии (более ~10000 вызовов) рассматривайте итеративные алгоритмы или явное управление стеком.

ПРАКТИЧЕСКАЯ РАБОТА №4. Оценка сложности рекурсивных алгоритмов.

Цель работы: Углубленное изучение методов анализа сложности рекурсивных алгоритмов, освоение методов решения рекуррентных соотношений, сравнение эффективности различных стратегий рекурсии на практических задачах.

Задачи:

1. Освоить методы решения рекуррентных соотношений: метод подстановки, итерационный метод, основная теорема.
2. Проанализировать сложность рекурсивных алгоритмов деления и властвования.
3. Реализовать и сравнить рекурсивные алгоритмы для задач сортировки и поиска.
4. Исследовать влияние мемоизации и динамического программирования на сложность рекурсивных алгоритмов.
5. (Для Qt) Создать интерактивную визуализацию дерева рекурсивных вызовов.

Теоретический материал

1. Методы решения рекуррентных соотношений

а) Метод подстановки (Substitution Method):

1. Угадать вид решения
2. Доказать корректность по индукции
3. Найти константы

Пример: $T(n) = 2T(n/2) + n \Rightarrow$ Предполагаем $O(n \log n)$

б) Итерационный метод (Iteration Method):

Раскрытие рекурсии до получения ряда:

text

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2[2T(n/4) + n/2] + n = 4T(n/4) + 2n \\ &= 8T(n/8) + 3n \\ &= \dots \\ &= 2^k T(n/2^k) + kn \end{aligned}$$

При $n/2^k = 1 \Rightarrow k = \log_2 n$

$$T(n) = nT(1) + n \log_2 n = O(n \log n)$$

в) Основная теорема (Master Theorem):

Для $T(n) = aT(n/b) + f(n)$, где $a \geq 1$, $b > 1$:

1. Если $f(n) = O(n^{(\log_b a - \epsilon)})$, $\epsilon > 0$, то $T(n) = \Theta(n^{\log_b a})$
2. Если $f(n) = \Theta(n^{\log_b a} \log^k n)$, то $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
3. Если $f(n) = \Omega(n^{(\log_b a + \epsilon)})$, $\epsilon > 0$, и $af(n/b) \leq cf(n)$, $c < 1$, то $T(n) = \Theta(f(n))$

2. Рекурсивные алгоритмы "разделяй и властвуй"

Быстрая сортировка (QuickSort):

text

$$T(n) = T(k) + T(n-k-1) + \Theta(n)$$

Лучший случай (равные части): $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$

Худший случай (несбалансированное разбиение): $T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$

Сортировка слиянием (MergeSort):

text

$$T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n) \text{ (всегда)}$$

Бинарный поиск:

text

$$T(n) = T(n/2) + \Theta(1) = \Theta(\log n)$$

3. Динамическое программирование и мемоизация

Пример с числами Фибоначчи:

- Наивная рекурсия: $T(n) = T(n-1) + T(n-2) + \Theta(1) = O(2^n)$
- С мемоизацией: $T(n) = \Theta(n)$ (каждое значение вычисляется один раз)
- Динамическое программирование (восходящее): $\Theta(n)$

Пример решения (Консольное приложение на C++)

cpp

```
#include <iostream>
#include <vector>
#include <chrono>
#include <algorithm>
#include <map>
#include <cmath>
```

```
using namespace std;
using namespace chrono;
```

// 1. Рекурсивная быстрая сортировка с подсчетом операций

```
class QuickSortAnalyzer {
private:
    long long comparisons;
    long long swaps;

    int partition(vector<int>& arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;

        for (int j = low; j < high; j++) {
            comparisons++;
            if (arr[j] < pivot) {
                i++;
                swaps++;
                swap(arr[i], arr[j]);
            }
        }
        swaps++;
        swap(arr[i + 1], arr[high]);
        return i + 1;
    }

    void quickSortRec(vector<int>& arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSortRec(arr, low, pi - 1);
            quickSortRec(arr, pi + 1, high);
        }
    }

public:
    QuickSortAnalyzer() : comparisons(0), swaps(0) {}

    void sort(vector<int>& arr) {
        comparisons = 0;
        swaps = 0;
        quickSortRec(arr, 0, arr.size() - 1);
    }
}
```

```

    long long getComparisons() const { return comparisons; }
    long long getSwaps() const { return swaps; }
};

```

// 2. Рекурсивная сортировка слиянием с анализом

```

class MergeSortAnalyzer {
private:
    long long comparisons;
    long long assignments;

    void merge(vector<int>& arr, int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;

        vector<int> L(n1), R(n2);

        // Копирование данных
        for (int i = 0; i < n1; i++) {
            L[i] = arr[left + i];
            assignments++;
        }
        for (int j = 0; j < n2; j++) {
            R[j] = arr[mid + 1 + j];
            assignments++;
        }

        // Слияние
        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2) {
            comparisons++;
            if (L[i] <= R[j]) {
                arr[k] = L[i];
                assignments++;
                i++;
            } else {
                arr[k] = R[j];
                assignments++;
                j++;
            }
            k++;
        }

        // Копирование оставшихся элементов
        while (i < n1) {
            arr[k] = L[i];
            assignments++;
            i++;
            k++;
        }
        while (j < n2) {
            arr[k] = R[j];

```

```

        assignments++;
        j++;
        k++;
    }
}

void mergeSortRec(vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSortRec(arr, left, mid);
        mergeSortRec(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

public:
    MergeSortAnalyzer() : comparisons(0), assignments(0) {}

    void sort(vector<int>& arr) {
        comparisons = 0;
        assignments = 0;
        mergeSortRec(arr, 0, arr.size() - 1);
    }

    long long getComparisons() const { return comparisons; }
    long long getAssignments() const { return assignments; }
};

// 3. Решение задачи о рюкзаке (0/1 Knapsack) рекурсивно
int knapsackRecursive(const vector<int>& weights, const vector<int>& values,
    int capacity, int n, map<pair<int, int>, int>& memo) {
    // Базовый случай
    if (n == 0 || capacity == 0) return 0;

    // Проверка мемоизации
    auto key = make_pair(n, capacity);
    if (memo.find(key) != memo.end()) {
        return memo[key];
    }

    // Если вес текущего предмета больше вместимости, пропускаем его
    if (weights[n-1] > capacity) {
        memo[key] = knapsackRecursive(weights, values, capacity, n-1, memo);
        return memo[key];
    }

    // Максимум из двух случаев: берем или не берем предмет
    int include = values[n-1] + knapsackRecursive(weights, values,
        capacity - weights[n-1], n-1, memo);
    int exclude = knapsackRecursive(weights, values, capacity, n-1, memo);

    memo[key] = max(include, exclude);
}

```

```

    return memo[key];
}

// 4. Быстрое возведение в степень (рекурсивное)
double powerRecursive(double x, int n) {
    if (n == 0) return 1;
    if (n < 0) return 1 / powerRecursive(x, -n);

    double half = powerRecursive(x, n / 2);
    if (n % 2 == 0) {
        return half * half;
    } else {
        return x * half * half;
    }
}

// 5. Анализ глубины рекурсии
void analyzeRecursionDepth(int n, int& maxDepth, int currentDepth = 1) {
    maxDepth = max(maxDepth, currentDepth);

    if (n <= 1) return;

    // Имитация разделения на подзадачи
    analyzeRecursionDepth(n / 2, maxDepth, currentDepth + 1);
    analyzeRecursionDepth(n / 2, maxDepth, currentDepth + 1);
}

int main() {
    cout << "=== Продвинутый анализ сложности рекурсивных алгоритмов ===\n\n";

    // Генерация тестовых данных
    vector<int> sizes = {100, 1000, 5000, 10000};

    // Анализ алгоритмов сортировки
    cout << "Анализ рекурсивных алгоритмов сортировки:\n";
    cout << "Размер | QuickSort (сравнения/обмены) | MergeSort\n";
    cout << "      (сравнения/присваивания)\n";
    cout << string(80, '-') << endl;

    for (int size : sizes) {
        // Генерация случайного массива
        vector<int> arr1(size);
        for (int i = 0; i < size; i++) {
            arr1[i] = rand() % 10000;
        }
        vector<int> arr2 = arr1; // Копия для MergeSort

        // QuickSort
        QuickSortAnalyzer qs;
        auto start = high_resolution_clock::now();
        qs.sort(arr1);
        auto end = high_resolution_clock::now();
    }
}

```

```

    auto qsTime = duration_cast<microseconds>(end - start);

    // MergeSort
    MergeSortAnalyzer ms;
    start = high_resolution_clock::now();
    ms.sort(arr2);
    end = high_resolution_clock::now();
    auto msTime = duration_cast<microseconds>(end - start);

    printf("%6d | %10lld/%8lld (%5lld мкс) | %10lld/%10lld (%5lld мкс)\n",
           size,
           qs.getComparisons(), qs.getSwaps(), qsTime.count(),
           ms.getComparisons(), ms.getAssignments(), msTime.count());
}

// Анализ задачи о рюкзаке с мемоизацией
cout << "\n\nЗадача о рюкзаке (0/1 Knapsack):\n";

vector<int> values = {60, 100, 120, 40, 80, 30, 90, 70};
vector<int> weights = {10, 20, 30, 15, 25, 5, 35, 12};
int capacity = 50;

map<pair<int, int>, int> memo;

auto start = high_resolution_clock::now();
int maxValue = knapsackRecursive(weights, values, capacity, values.size(), memo);
auto end = high_resolution_clock::now();
auto knapsackTime = duration_cast<microseconds>(end - start);

cout << "Максимальная ценность: " << maxValue << endl;
cout << "Время выполнения: " << knapsackTime.count() << " мкс" << endl;
cout << "Размер кэша (уникальных подзадач): " << memo.size() << endl;

// Быстрое возведение в степень
cout << "\n\nБыстрое возведение в степень:\n";
double x = 2.0;
vector<int> exponents = {10, 20, 30, 40};

cout << "x = " << x << endl;
cout << "\n\tРезультат\tРекурсивных вызовов\n";

for (int n : exponents) {
    start = high_resolution_clock::now();
    double result = powerRecursive(x, n);
    end = high_resolution_clock::now();
    auto powerTime = duration_cast<nanoseconds>(end - start);

    // Оценка количества вызовов: примерно log2(n)
    int estimatedCalls = 2 * static_cast<int>(log2(n)) + 1;

    cout << n << "\t" << result << "\t~" << estimatedCalls
         << " ВЫЗОВОВ (" << powerTime.count() << " нс)" << endl;
}

```

```

}

// Анализ глубины рекурсии
cout << "\n\nАнализ глубины рекурсии для задачи размера n:\n";
cout << "n\tМаксимальная глубина (теория: ~log2(n))\n";

for (int n : {10, 100, 1000, 10000}) {
    int maxDepth = 0;
    analyzeRecursionDepth(n, maxDepth);
    cout << n << "\t" << maxDepth << " (теория: "
        << static_cast<int>(log2(n)) + 1 << ")" << endl;
}

// Применение основной теоремы
cout << "\n\nПример применения основной теоремы:\n";
cout << "1. MergeSort:  $T(n) = 2T(n/2) + \Theta(n)$ \n";
cout << "   a=2, b=2, f(n)=n,  $n^{(\log_b a)}=n^1=n$ \n";
cout << "   Случай 2:  $f(n) = \Theta(n^{(\log_b a)} \log^0 n) \Rightarrow T(n) = \Theta(n \log n)$ \n\n";

cout << "2. Быстрое возведение:  $T(n) = T(n/2) + \Theta(1)$ \n";
cout << "   a=1, b=2, f(n)=1,  $n^{(\log_b a)}=n^0=1$ \n";
cout << "   Случай 2:  $f(n) = \Theta(n^{(\log_b a)} \log^0 n) \Rightarrow T(n) = \Theta(\log n)$ \n\n";

cout << "3. Наивный Фибоначчи:  $T(n) = T(n-1) + T(n-2) + \Theta(1)$ \n";
cout << "   Не подходит под основную теорему, решаем итерационно\n";
cout << "   Приблизительно  $T(n) = O(2^n)$ \n";

return 0;
}

```

Задания для выполнения

Вариант 1 (Консольный - Глубокий анализ):

1. **Реализуйте и проанализируйте следующие рекурсивные алгоритмы:**
 - Алгоритм Карацубы для умножения больших чисел
 - Рекурсивный алгоритм умножения матриц Штрассена
 - Рекурсивный алгоритм вычисления определителя матрицы (метод разложения по строке)
2. **Для каждого алгоритма:**
 - Выведите рекуррентное соотношение
 - Решите его с использованием основной теоремы или итерационного метода
 - Определите теоретическую сложность
 - Реализуйте алгоритм на C++ с подсчетом операций
3. **Проведите экспериментальный анализ:**
 - Замерьте время выполнения для разных размеров входных данных
 - Постройте графики зависимости времени от размера
 - Сравните с теоретическими предсказаниями
 - Определите практическую область применения каждого алгоритма
4. **Исследуйте влияние мемоизации:**
 - Реализуйте алгоритм вычисления чисел Каталана рекурсивно
 - Добавьте мемоизацию
 - Сравните производительность
 - Проанализируйте размер кэша

Вариант 2 (Qt - Визуализация и интерактивный анализ):

1. **Создайте приложение Qt для визуализации:**
 - Деревя рекурсивных вызовов для произвольной рекурсивной функции
 - Процесса разделения в алгоритмах "разделяй и властвуй"
2. **Функционал приложения должен включать:**
 - Выбор алгоритма из списка (QuickSort, MergeSort, Ханойские башни, др.)
 - Ввод параметров (размер массива, глубина рекурсии)
 - Визуализацию дерева вызовов с анимацией
 - Отображение стека вызовов в реальном времени
 - График зависимости времени от размера задачи
3. **Реализуйте интерактивный анализ:**
 - Возможность пошагового выполнения рекурсии
 - Подсветку текущего вызова в дереве
 - Отображение значений параметров для каждого вызова
 - Лог операций с временными метками
4. **Дополнительные возможности:**
 - Визуализация процесса сортировки с выделением рекурсивных разделов
 - Анализ использования памяти (глубина стека)
 - Экспорт дерева вызовов в графический файл

Критерии оценивания

«Удовлетворительно» (3):

- Реализованы базовые версии всех требуемых алгоритмов
- Приведены рекуррентные соотношения для каждого алгоритма
- Программа компилируется и работает

«Хорошо» (4):

- Для **Варианта 1:** Проведен полный теоретический анализ сложности с использованием разных методов. Реализованы оптимизации (мемоизация). Проведены замеры производительности, результаты корректно представлены.
- Для **Варианта 2:** Создано полнофункциональное Qt-приложение с визуализацией как минимум двух алгоритмов. Реализована анимация и пошаговое выполнение. Интерфейс интуитивен.
- В отчете присутствует сопоставление теоретической и практической сложности.

«Отлично» (5):

- Для **Варианта 1:** Выполнен углубленный анализ, включая асимптотическое разложение. Исследованы граничные случаи и оптимизации. Реализовано сравнение с итеративными аналогами. Построены детальные графики и проведен регрессионный анализ.
- Для **Варианта 2:** Реализована продвинутая визуализация с 3D-представлением дерева вызовов (используя Q3D или кастомную отрисовку). Добавлены инструменты профилирования (замер времени каждого вызова, использование памяти). Реализована возможность определения рекуррентного соотношения по экспериментальным данным.
- **Общее:** Отчет содержит оригинальные исследования, предложения по оптимизации. Представлен анализ преимуществ и недостатков рекурсии для каждого конкретного случая. Даны рекомендации по выбору между рекурсией и итерацией.

Советы по выполнению:

1. Для алгоритма Карацубы:

cpp

// Рекурсивное умножение больших чисел

```
BigInt karatsuba(const BigInt& x, const BigInt& y) {  
    if (x.size() < threshold || y.size() < threshold) {  
        return schoolbookMultiply(x, y);  
    }
```

```
}
```

```
// Разделение чисел
```

```
int m = min(x.size(), y.size()) / 2;
```

```
BigInt high1 = x >> m;
```

```
BigInt low1 = x - (high1 << m);
```

```
// ... аналогично для y
```

```
// Три рекурсивных умножения
```

```
BigInt z0 = karatsuba(low1, low2);
```

```
BigInt z1 = karatsuba((low1 + high1), (low2 + high2));
```

```
BigInt z2 = karatsuba(high1, high2);
```

```
// Комбинирование результатов
```

```
return (z2 << (2*m)) + ((z1 - z2 - z0) << m) + z0;
```

```
}
```

2. **В Qt для визуализации дерева:**

- Используйте QGraphicsScene для отрисовки узлов и связей
- Для анимации применяйте QPropertyAnimation
- Для стека вызовов можно использовать QListWidget с подсветкой

3. **Для точных замеров:**

cpp

```
// Используйте steady_clock для измерений
```

```
auto start = chrono::steady_clock::now();
```

```
// ... выполнение алгоритма
```

```
auto end = chrono::steady_clock::now();
```

```
auto duration = chrono::duration<double, milli>(end - start);
```

4. **При анализе памяти:**

- Оценивайте глубину стека
- Учитывайте затраты на хранение кэша мемоизации
- Для рекурсивных алгоритмов с большими структурами данных учитывайте копирование

5. **Для экспериментальной проверки сложности:**

- Замеряйте время для геометрической прогрессии размеров (10, 100, 1000, ...)
- Стройте log-log графики для определения степени полинома
- Используйте метод наименьших квадратов для аппроксимации

ПРАКТИЧЕСКАЯ РАБОТА №5. Оценка сложности эвристических алгоритмов.

Цель работы: Изучение принципов разработки и анализа эвристических алгоритмов для решения NP-сложных задач, освоение методов оценки их эффективности и качества приближенных решений.

Задачи:

1. Изучить понятие эвристических алгоритмов и их отличие от точных алгоритмов.
2. Освоить методы анализа качества эвристических решений (оценка приближения, эмпирический анализ).
3. Реализовать эвристические алгоритмы для классических NP-сложных задач.
4. Сравнить различные эвристики по времени выполнения и качеству решения.
5. (Для Qt) Создать визуализацию работы эвристических алгоритмов.

Теоретический материал

1. Основные понятия эвристических алгоритмов

Эвристический алгоритм — алгоритм, который не гарантирует нахождение оптимального решения, но находит достаточно хорошее решение за приемлемое время. Используется для NP-сложных задач, где точные алгоритмы требуют экспоненциального времени.

Ключевые характеристики:

- **Точность (качество решения):** Насколько решение близко к оптимальному
- **Время выполнения:** Должно быть полиномиальным
- **Детерминированность:** Детерминированные vs стохастические эвристики
- **Гарантии приближения:** Наличие теоретических границ

2. Классические задачи и эвристики

Задача коммивояжера (TSP):

- **Жадный алгоритм (ближайший сосед):** $O(n^2)$, качество обычно 15-20% от оптимального
- **Алгоритм 2-opt:** Локальный поиск с улучшением, $O(n^2)$ на итерацию
- **Алгоритм Кристофидеса:** Гарантирует решение в 1.5 раза хуже оптимального

Задача о рюкзаке:

- **Жадный по удельной стоимости:** $O(n \log n)$, нет гарантий
- **Динамическое программирование:** Точное, но $O(nW)$ (псевдополиномиальное)

Задача раскраски графа:

- **Жадный алгоритм (последовательная раскраска):** $O(n^2)$
- **Алгоритм DSATUR:** $O(n^3)$, обычно лучше жадного

3. Методы оценки эвристик

а) Экспериментальный анализ:

- Сравнение с известными оптимальными решениями
- Тестирование на стандартных наборах данных
- Статистический анализ результатов

б) Теоретические гарантии:

- **Коэффициент аппроксимации:** $\rho = \text{ALG}/\text{OPT}$ (для минимизации)
- **PTAS (Polynomial Time Approximation Scheme):** $(1+\epsilon)$ -приближение
- **FPTAS (Fully PTAS):** PTAS с полиномом от $1/\epsilon$

4. Методы построения эвристик

math

1. Жадные алгоритмы: локально оптимальный выбор
 2. Локальный поиск: итеративное улучшение
 3. Метаэвристики: генетические алгоритмы, имитация отжига
 4. Гибридные методы: комбинация разных подходов
-

Пример решения (Консольное приложение на C++)

cpp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <chrono>
#include <random>
#include <limits>
#include <fstream>
```

```
using namespace std;
using namespace chrono;
```

// Структура для точки в задаче коммивояжера

```
struct Point {
    double x, y;
    int id;

    double distanceTo(const Point& other) const {
        double dx = x - other.x;
        double dy = y - other.y;
        return sqrt(dx*dx + dy*dy);
    }
};
```

// 1. Жадный алгоритм (ближайший сосед) для TSP

```
vector<int> greedyTSP(const vector<Point>& points) {
    int n = points.size();
    vector<bool> visited(n, false);
    vector<int> tour;
    tour.reserve(n);

    // Начинаем с произвольной точки (0)
    int current = 0;
    visited[current] = true;
    tour.push_back(current);

    for (int i = 1; i < n; i++) {
        int nearest = -1;
        double minDist = numeric_limits<double>::max();

        // Ищем ближайшего непосещенного соседа
        for (int j = 0; j < n; j++) {
            if (!visited[j]) {
                double dist = points[current].distanceTo(points[j]);
                if (dist < minDist) {
                    minDist = dist;
                    nearest = j;
                }
            }
        }
    }
}
```

```

        // Переходим к ближайшему соседу
        current = nearest;
        visited[current] = true;
        tour.push_back(current);
    }

    return tour;
}

// 2. Алгоритм 2-opt (локальное улучшение)
vector<int> twoOpt(const vector<Point>& points, const vector<int>& initialTour) {
    vector<int> tour = initialTour;
    int n = tour.size();
    bool improved = true;

    while (improved) {
        improved = false;

        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 2; j < n; j++) {
                // Текущие ребра: (i, i+1) и (j, j+1 mod n)
                int a = tour[i];
                int b = tour[(i + 1) % n];
                int c = tour[j];
                int d = tour[(j + 1) % n];

                double currentDist = points[a].distanceTo(points[b]) +
                                    points[c].distanceTo(points[d]);
                double newDist = points[a].distanceTo(points[c]) +
                                points[b].distanceTo(points[d]);

                // Если улучшение возможно, переворачиваем сегмент
                if (newDist < currentDist) {
                    reverse(tour.begin() + i + 1, tour.begin() + j + 1);
                    improved = true;
                }
            }
        }
    }

    return tour;
}

// 3. Жадный алгоритм для задачи о рюкзаке
struct Item {
    int weight;
    int value;
    double ratio; // value/weight
};

vector<bool> greedyKnapsack(const vector<Item>& items, int capacity) {

```

```

// Создаем копию с индексами
vector<pair<double, int>> indexedItems;
for (size_t i = 0; i < items.size(); i++) {
    indexedItems.emplace_back(items[i].ratio, i);
}

// Сортируем по убыванию удельной стоимости
sort(indexedItems.rbegin(), indexedItems.rend());

vector<bool> selected(items.size(), false);
int currentWeight = 0;
int totalValue = 0;

for (const auto& [ratio, idx] : indexedItems) {
    if (currentWeight + items[idx].weight <= capacity) {
        selected[idx] = true;
        currentWeight += items[idx].weight;
        totalValue += items[idx].value;
    }
}

return selected;
}

// 4. Эвристика для раскраски графа (жадный)
vector<int> greedyColoring(const vector<vector<int>>& graph) {
    int n = graph.size();
    vector<int> colors(n, -1); // -1 означает нераскрашенную

    // Цвета доступны?
    vector<bool> available(n, true);

    for (int vertex = 0; vertex < n; vertex++) {
        // Помечаем цвета соседей как недоступные
        fill(available.begin(), available.end(), true);
        for (int neighbor : graph[vertex]) {
            if (colors[neighbor] != -1) {
                available[colors[neighbor]] = false;
            }
        }

        // Находим первый доступный цвет
        int color;
        for (color = 0; color < n; color++) {
            if (available[color]) break;
        }

        colors[vertex] = color;
    }

    return colors;
}

```

// 5. Генератор тестовых данных

```
vector<Point> generateRandomPoints(int n, int maxCoord = 1000) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(0, maxCoord);

    vector<Point> points;
    for (int i = 0; i < n; i++) {
        points.push_back({static_cast<double>(dis(gen)),
                          static_cast<double>(dis(gen)), i});
    }
    return points;
}

vector<Item> generateRandomItems(int n, int maxWeight = 100, int maxValue = 100) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> weightDis(1, maxWeight);
    uniform_int_distribution<> valueDis(1, maxValue);

    vector<Item> items;
    for (int i = 0; i < n; i++) {
        int weight = weightDis(gen);
        int value = valueDis(gen);
        items.push_back({weight, value, static_cast<double>(value)/weight});
    }
    return items;
}

// Вычисление длины тура
double tourLength(const vector<Point>& points, const vector<int>& tour) {
    double length = 0.0;
    int n = tour.size();

    for (int i = 0; i < n; i++) {
        int from = tour[i];
        int to = tour[(i + 1) % n];
        length += points[from].distanceTo(points[to]);
    }

    return length;
}

int main() {
    cout << "=== Анализ эвристических алгоритмов ===\n\n";

    // Часть 1: Задача коммивояжера
    cout << "1. Задача коммивояжера (TSP):\n";
    cout << "n\tЖадный алгоритм\t2-орт улучшение\tУлучшение (%)\tВремя (мкс)\n";
    cout << string(60, '-') << endl;
```

```

vector<int> tspSizes = {10, 20, 50, 100};
for (int n : tspSizes) {
    auto points = generateRandomPoints(n);

    // Жадный алгоритм
    auto start = high_resolution_clock::now();
    auto greedyTour = greedyTSP(points);
    auto end = high_resolution_clock::now();
    auto greedyTime = duration_cast<microseconds>(end - start);
    double greedyLength = tourLength(points, greedyTour);

    // 2-opt улучшение
    start = high_resolution_clock::now();
    auto twoOptTour = twoOpt(points, greedyTour);
    end = high_resolution_clock::now();
    auto twoOptTime = duration_cast<microseconds>(end - start);
    double twoOptLength = tourLength(points, twoOptTour);

    double improvement = ((greedyLength - twoOptLength) / greedyLength) * 100;

    printf("%3d\t%12.2f\t%12.2f\t%8.1f%%\t%7lld/%7lld\n",
        n, greedyLength, twoOptLength, improvement,
        greedyTime.count(), twoOptTime.count());
}

// Часть 2: Задача о рюкзаке
cout << "\n\n2. Задача о рюкзаке:\n";
cout << "\tЕмкость\tЖадная ценность\tОптимум (DP)\tОтношение\tВремя (мкс)\n";
cout << string(70, '-') << endl;

for (int n : {10, 20, 50, 100}) {
    auto items = generateRandomItems(n);
    int capacity = n * 25; // 25% от общего веса

    // Жадный алгоритм
    auto start = high_resolution_clock::now();
    auto greedySelection = greedyKnapsack(items, capacity);
    auto end = high_resolution_clock::now();
    auto greedyTime = duration_cast<microseconds>(end - start);

    // Вычисляем ценность жадного решения
    int greedyValue = 0;
    for (size_t i = 0; i < items.size(); i++) {
        if (greedySelection[i]) {
            greedyValue += items[i].value;
        }
    }

    // Точное решение (динамическое программирование) для сравнения
    start = high_resolution_clock::now();
    vector<vector<int>> dp(items.size() + 1, vector<int>(capacity + 1, 0));

```

```

    for (size_t i = 1; i <= items.size(); i++) {
        for (int w = 0; w <= capacity; w++) {
            if (items[i-1].weight <= w) {
                dp[i][w] = max(dp[i-1][w],
                               dp[i-1][w - items[i-1].weight] + items[i-1].value);
            } else {
                dp[i][w] = dp[i-1][w];
            }
        }
    }
    end = high_resolution_clock::now();
    auto dpTime = duration_cast<microseconds>(end - start);
    int optimalValue = dp[items.size()][capacity];

    double ratio = (double)greedyValue / optimalValue;

    printf("%3d\t%8d\t%13d\t%12d\t%8.3f\t%7lld/%7lld\n",
           n, capacity, greedyValue, optimalValue, ratio,
           greedyTime.count(), dpTime.count());
}

// Часть 3: Раскраска графа
cout << "\n\n3. Раскраска графа:\n";

// Генерируем случайный граф
int graphSize = 20;
vector<vector<int>>> graph(graphSize);

random_device rd;
mt19937 gen(rd());
uniform_real_distribution<> probDis(0, 1);

// Создаем случайные ребра с вероятностью 0.3
for (int i = 0; i < graphSize; i++) {
    for (int j = i + 1; j < graphSize; j++) {
        if (probDis(gen) < 0.3) {
            graph[i].push_back(j);
            graph[j].push_back(i);
        }
    }
}

auto start = high_resolution_clock::now();
auto colors = greedyColoring(graph);
auto end = high_resolution_clock::now();
auto coloringTime = duration_cast<microseconds>(end - start);

// Находим количество использованных цветов
int colorCount = *max_element(colors.begin(), colors.end()) + 1;

cout << "Граф с " << graphSize << " вершинами:\n";
cout << "Количество цветов (жадный алгоритм): " << colorCount << endl;

```

```

cout << "Время выполнения: " << coloringTime.count() << " мкс" << endl;

// Анализ качества
cout << "\n\n4. Анализ качества эвристик:\n";
cout << "Задача\t\tЭвристика\t\tГарантии приближения\t\tТипичное качество\n";
cout << string(70, '-') << endl;
cout << "TSP\t\tБлижайший сосед\t\tНет гарантий\t\t15-25% от оптимального\n";
cout << "TSP\t\t2-opt\t\t\tНет гарантий\t\t5-15% от оптимального\n";
cout << "TSP\t\tКристофидес\t\t $p = 1.5$ \t\t\t10-15% от оптимального\n";
cout << "Рюкзак\t\tЖадный (удельная)\t\tНет гарантий\t\t60-90% от оптимального\n";
cout << "Раскраска\t\tЖадный\t\t\t $O(\Delta)$  цветов\t\tДо  $\Delta+1$  цветов ( $\Delta$  - макс степень)\n";

// Эксперимент с разными начальными точками для TSP
cout << "\n\n5. Влияние начальной точки на жадный алгоритм TSP:\n";

vector<Point> testPoints = generateRandomPoints(50);
int trials = 10;
vector<double> lengths;

for (int trial = 0; trial < trials; trial++) {
    // Перемешиваем порядок обхода
    vector<Point> shuffled = testPoints;
    shuffle(shuffled.begin(), shuffled.end(), gen);

    auto tour = greedyTSP(shuffled);
    lengths.push_back(tourLength(shuffled, tour));
}

double avg = accumulate(lengths.begin(), lengths.end(), 0.0) / lengths.size();
double minLength = *min_element(lengths.begin(), lengths.end());
double maxLength = *max_element(lengths.begin(), lengths.end());

cout << "Средняя длина: " << avg << endl;
cout << "Минимальная: " << minLength << " (улучшение "
    << ((avg - minLength)/avg*100) << "%)\n";
cout << "Максимальная: " << maxLength << " (ухудшение "
    << ((maxLength - avg)/avg*100) << "%)\n";
cout << "Отношение max/min: " << maxLength/minLength << endl;

return 0;
}

```

Задания для выполнения

Вариант 1 (Консольный - Сравнительный анализ эвристик):

1. **Реализуйте и проанализируйте следующие эвристики:**
 - Для TSP: алгоритм Кристофидеса и алгоритм вставки (cheapest insertion)
 - Для задачи покрытия множества: жадный алгоритм
 - Для задачи планирования работ: алгоритм LPT (Longest Processing Time)
2. **Проведите детальный анализ качества:**
 - На стандартных наборах данных (TSPLIB для TSP)
 - Сравните с известными оптимальными решениями
 - Вычислите среднее, минимальное и максимальное отклонение

- Постройте графики зависимости качества от размера задачи
 - 3. **Исследуйте влияние параметров:**
 - Для 2-орт: ограничение количества итераций
 - Для жадных алгоритмов: различные критерии выбора
 - Для локального поиска: различные соседства
 - 4. **Реализуйте метаэвристику:**
 - Простой генетический алгоритм для TSP
 - Сравните с конструктивными эвристиками
- Вариант 2 (Qt - Визуализация и интерактивное исследование):**
1. **Создайте приложение Qt для визуализации эвристик:**
 - Интерактивное построение графа/точек для TSP
 - Визуализация процесса работы алгоритмов в реальном времени
 - Возможность вмешательства в процесс (ручной выбор)
 2. **Функционал должен включать:**
 - Выбор задачи (TSP, раскраска, рюкзак)
 - Выбор алгоритма и его параметров
 - Анимация шагов алгоритма
 - Сравнение нескольких алгоритмов на одном наборе данных
 - Экспорт результатов
 3. **Для TSP реализуйте:**
 - Визуализацию тура
 - Анимацию работы 2-орт (показать поворот сегмента)
 - Отображение текущей длины и улучшения
 - Возможность ручного перемещения точек
 4. **Для анализа качества:**
 - График сходимости (длина тура vs время/итерации)
 - Гистограмма распределения результатов по множеству запусков
 - Сравнение с нижними границами (например, MST для TSP)

Критерии оценивания

«Удовлетворительно» (3):

- Реализованы базовые версии требуемых эвристических алгоритмов
- Проведены простые тесты на малых размерах
- В отчете описаны принципы работы алгоритмов

«Хорошо» (4):

- Для **Варианта 1:** Проведен сравнительный анализ минимум 3 эвристик для каждой задачи. Получены статистические данные о качестве. Реализовано сравнение с точными алгоритмами (где возможно).
- Для **Варианта 2:** Создано полнофункциональное Qt-приложение с визуализацией минимум 2 алгоритмов. Реализована интерактивная работа с параметрами. Есть возможность сохранения/загрузки данных.
- В отчете представлен анализ результатов, выводы о применимости алгоритмов.

«Отлично» (5):

- Для **Варианта 1:** Проведено исследование на стандартных benchmark-наборах. Реализованы и сравнены метаэвристики. Проведен анализ сложности в среднем случае. Предложены и реализованы модификации для улучшения качества.
- Для **Варианта 2:** Реализована продвинутая визуализация с 3D-графиками, тепловыми картами качества. Добавлены инструменты профилирования и оптимизации. Реализовано автоматическое тестирование и сравнение алгоритмов.
- **Общее:** Отчет содержит глубокий анализ, включая математическое обоснование эвристик. Представлены оригинальные исследования или модификации алгоритмов. Даны практические рекомендации по выбору эвристики в зависимости от требований.

Советы по выполнению:

1. Для работы с TSPLIB:

```
cpp
// Чтение данных из файла TSPLIB
ifstream file("bier127.tsp");
vector<Point> points;
string line;
while (getline(file, line)) {
    if (line.find("NODE_COORD_SECTION") != string::npos) {
        while (getline(file, line) && line != "EOF") {
            int id;
            double x, y;
            sscanf(line.c_str(), "%d %lf %lf", &id, &x, &y);
            points.push_back({x, y, id-1});
        }
    }
}
```

2. Для генетического алгоритма:

```
cpp
struct Individual {
    vector<int> tour;
    double fitness;

    bool operator<(const Individual& other) const {
        return fitness < other.fitness;
    }
};

// Мутация (2-орт случайный переворот)
void mutate(Individual& ind, double mutationRate) {
    if (random() < mutationRate) {
        int i = rand() % ind.tour.size();
        int j = rand() % ind.tour.size();
        if (i > j) swap(i, j);
        reverse(ind.tour.begin() + i, ind.tour.begin() + j + 1);
    }
}
```

3. В Qt для анимации:

```
cpp
// Анимация переворота в 2-орт
void animate2Opt(int i, int j) {
    QPropertyAnimation* anim = new QPropertyAnimation(this, "segment");
    anim->setDuration(1000);
    anim->setStartValue(i);
    anim->setEndValue(j);
    anim->start();

    // Перерисовываем после анимации
    connect(anim, &QPropertyAnimation::finished, this, &Widget::update);
}
```

4. Для статистического анализа:

cpp

// Запуск алгоритма много раз для оценки дисперсии

```
vector<double> results;
```

```
for (int run = 0; run < 100; run++) {
```

```
    auto tour = runAlgorithm(points);
```

```
    results.push_back(tourLength(points, tour));
```

```
}
```

// Вычисление статистик

```
double mean = accumulate(results.begin(), results.end(), 0.0) / results.size();
```

```
double variance = 0.0;
```

```
for (double r : results) variance += (r - mean) * (r - mean);
```

```
variance /= results.size();
```

```
double stddev = sqrt(variance);
```

5. Для визуализации градиента качества:

- Используйте цветовую кодировку для отображения качества решений
- Строите диаграммы Парето для многокритериальных задач
- Реализуйте интерактивное изменение параметров с immediate feedback

6. Оптимизация производительности:

cpp

// Кэширование расстояний для TSP

```
vector<vector<double>> distanceMatrix;
```

```
void precomputeDistances(const vector<Point>& points) {
```

```
    int n = points.size();
```

```
    distanceMatrix.resize(n, vector<double>(n));
```

```
    for (int i = 0; i < n; i++) {
```

```
        for (int j = 0; j < n; j++) {
```

```
            distanceMatrix[i][j] = points[i].distanceTo(points[j]);
```

```
        }
```

```
    }
```

```
}
```

ПРАКТИЧЕСКАЯ РАБОТА №6. Работа с классами. Перегрузка методов.

Цель работы: Освоить принципы объектно-ориентированного программирования на языке C++, научиться создавать классы, реализовывать перегрузку методов и операторов, применять инкапсуляцию и полиморфизм на практике.

Задачи:

1. Изучить синтаксис создания классов в C++.
2. Освоить принципы инкапсуляции, наследования и полиморфизма.
3. Научиться перегружать методы и операторы классов.
4. Реализовать практические примеры использования перегрузки.
5. (Для Qt) Создать графический интерфейс для работы с пользовательскими классами.

Теоретический материал

1. Основы классов в C++

Класс — это пользовательский тип данных, который объединяет данные (поля) и функции для работы с ними (методы).

cpp

```
class MyClass {
```

```
private:
```

```
    // Закрытые члены (доступны только внутри класса)
```

```
    int privateData;
```

```
protected:
```

```
    // Защищенные члены (доступны в классе и наследниках)
```

```
public:
```

```
    // Открытые члены (доступны извне)
```

```
    MyClass(); // Конструктор
```

```
    ~MyClass(); // Деструктор
```

```
    void setData(int value); // Метод установки значения
```

```
    int getData() const; // Константный метод
```

```
};
```

2. Конструкторы и деструкторы

- **Конструктор** — инициализирует объект при создании
- **Деструктор** — очищает ресурсы при уничтожении объекта
- **Конструктор копирования** — создает копию объекта
- **Конструктор перемещения (C++11)** — эффективное перемещение ресурсов

cpp

```
class Vector {
```

```
private:
```

```
    int* data;
```

```
    int size;
```

```
public:
```

```
    // Конструктор по умолчанию
```

```
    Vector() : data(nullptr), size(0) {}
```

```
    // Параметрический конструктор
```

```
    Vector(int n) : size(n) {
```

```
        data = new int[n];
```

```
    }
```

// Конструктор копирования

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

// Деструктор

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

3. Перегрузка методов

Перегрузка методов — создание нескольких методов с одинаковым именем, но разными параметрами.

cpp

```
class Calculator {  
public:  
    int add(int a, int b) {  
        return a + b;  
    }  
  
    double add(double a, double b) {  
        return a + b;  
    }  
  
    int add(int a, int b, int c) {  
        return a + b + c;  
    }  
};
```

4. Перегрузка операторов

C++ позволяет переопределять поведение операторов для пользовательских типов:

cpp

```
class Complex {  
private:  
    double real, imag;  
  
public:  
    Complex(double r = 0, double i = 0) : real(r), imag(i) {}
```

// Перегрузка оператора сложения

```
Complex operator+(const Complex& other) const {  
    return Complex(real + other.real, imag + other.imag);  
}
```

// Перегрузка оператора вывода

```
friend ostream& operator<<(ostream& os, const Complex& c) {  
    os << c.real << " + " << c.imag << "i";  
    return os;  
}
```

```
// Перегрузка оператора индексации
double& operator[](int index) {
    if (index == 0) return real;
    if (index == 1) return imag;
    throw out_of_range("Index out of range");
}
};
```

Основные перегружаемые операторы:

- Арифметические: +, -, *, /, %
- Сравнения: ==, !=, <, >, <=, >=
- Присваивания: =, +=, -=, *=, /=
- Ввода/вывода: <<, >>
- Индексации: []
- Вызова функции: ()
- Преобразования типа: operator type()

Пример решения (Консольное приложение на C++)

```
cpp
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
#include <stdexcept>

using namespace std;

// =====
// 1. Класс "Дробь" с перегрузкой операторов
// =====

class Fraction {
private:
    int numerator; // числитель
    int denominator; // знаменатель

    // Функция для сокращения дроби
    void simplify() {
        int gcd = findGCD(abs(numerator), abs(denominator));
        numerator /= gcd;
        denominator /= gcd;

        if (denominator < 0) {
            numerator = -numerator;
            denominator = -denominator;
        }
    }

}

// Нахождение НОД (алгоритм Евклида)
int findGCD(int a, int b) const {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
}
```

```

    }
    return a;
}

public:
    // Конструкторы
    Fraction() : numerator(0), denominator(1) {}

    Fraction(int num, int den = 1) : numerator(num), denominator(den) {
        if (den == 0) {
            throw invalid_argument("Знаменатель не может быть нулем");
        }
        simplify();
    }

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

    void setNumerator(int num) {
        numerator = num;
        simplify();
    }

    void setDenominator(int den) {
        if (den == 0) {
            throw invalid_argument("Знаменатель не может быть нулем");
        }
        denominator = den;
        simplify();
    }

    // Перегрузка арифметических операторов
    Fraction operator+(const Fraction& other) const {
        int num = numerator * other.denominator + other.numerator * denominator;
        int den = denominator * other.denominator;
        return Fraction(num, den);
    }

    Fraction operator-(const Fraction& other) const {
        int num = numerator * other.denominator - other.numerator * denominator;
        int den = denominator * other.denominator;
        return Fraction(num, den);
    }

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

    Fraction operator/(const Fraction& other) const {
        if (other.numerator == 0) {

```

```

        throw invalid_argument("Деление на ноль");
    }
    return Fraction(numerator * other.denominator,
                    denominator * other.numerator);
}

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

Fraction operator+() const {
    return *this;
}

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

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

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

bool operator>(const Fraction& other) const {
    return other < *this;
}

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

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

// Операторы присваивания с арифметикой
Fraction& operator+=(const Fraction& other) {
    *this = *this + other;
    return *this;
}

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

// Преобразование к double

```

```

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

// Перегрузка оператора вывода
friend ostream& operator<<(ostream& os, const Fraction& f) {
    if (f.denominator == 1) {
        os << f.numerator;
    } else {
        os << f.numerator << "/" << f.denominator;
    }
    return os;
}

// Перегрузка оператора ввода
friend istream& operator>>(istream& is, Fraction& f) {
    char slash;
    is >> f.numerator >> slash >> f.denominator;
    if (f.denominator == 0) {
        throw invalid_argument("Знаменатель не может быть нулем");
    }
    f.simplify();
    return is;
}

// Перегрузка оператора индексации
int operator[](int index) const {
    if (index == 0) return numerator;
    if (index == 1) return denominator;
    throw out_of_range("Индекс должен быть 0 или 1");
}
};

// =====
// 2. Класс "Вектор" с перегрузкой методов
// =====

class Vector3D {
private:
    double x, y, z;

public:
    // Конструкторы (перегрузка)
    Vector3D() : x(0), y(0), z(0) {}
    Vector3D(double val) : x(val), y(val), z(val) {} // Скаляр
    Vector3D(double x, double y, double z) : x(x), y(y), z(z) {}

    // Методы доступа
    double getX() const { return x; }
    double getY() const { return y; }
    double getZ() const { return z; }

    void setX(double val) { x = val; }

```

```

void setY(double val) { y = val; }
void setZ(double val) { z = val; }

// Методы вычислений (перегрузка по возвращаемому значению)
double length() const {
    return sqrt(x*x + y*y + z*z);
}

Vector3D normalized() const {
    double len = length();
    if (len == 0) return Vector3D(0);
    return Vector3D(x/len, y/len, z/len);
}

// Скалярное произведение (перегрузка методов)
double dot(const Vector3D& other) const {
    return x*other.x + y*other.y + z*other.z;
}

double dot(double scalar) const {
    return length() * scalar;
}

// 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
    );
}

// Перегрузка операторов
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);
}

friend Vector3D operator*(double scalar, const Vector3D& vec) {
    return vec * scalar;
}

bool operator==(const Vector3D& other) const {
    return x == other.x && y == other.y && z == other.z;
}

```

```

// Перегрузка оператора вызова функции
double operator()(int index) const {
    switch(index) {
        case 0: return x;
        case 1: return y;
        case 2: return z;
        default: throw out_of_range("Индекс должен быть 0, 1 или 2");
    }
}

// Перегрузка оператора вывода
friend ostream& operator<<(ostream& os, const Vector3D& v) {
    os << "(" << v.x << ", " << v.y << ", " << v.z << ")";
    return os;
}
};

// =====
// 3. Базовый класс и наследование
// =====

class Animal {
protected:
    string name;
    int age;

public:
    Animal(const string& name, int age) : name(name), age(age) {}

    // Виртуальный метод (полиморфизм)
    virtual void makeSound() const {
        cout << name << " издает какой-то звук" << endl;
    }

    virtual void displayInfo() const {
        cout << "Имя: " << name << ", Возраст: " << age << " лет" << endl;
    }

    virtual ~Animal() {} // Виртуальный деструктор
};

class Dog : public Animal {
private:
    string breed;

public:
    Dog(const string& name, int age, const string& breed)
        : Animal(name, age), breed(breed) {}

    // Перегрузка метода (override)
    void makeSound() const override {
        cout << name << " гавкает: Гав-гав!" << endl;
    }
}

```

```

    }

    void displayInfo() const override {
        Animal::displayInfo();
        cout << "Порода: " << breed << endl;
    }

    // Новый метод
    void fetch() const {
        cout << name << " приносит палку!" << endl;
    }
};

class Cat : public Animal {
private:
    int lives; // У кошек 9 жизней :)

public:
    Cat(const string& name, int age, int lives = 9)
        : Animal(name, age), lives(lives) {}

    // Перегрузка метода
    void makeSound() const override {
        cout << name << " мяукает: Мяу-мяу!" << endl;
    }

    void displayInfo() const override {
        Animal::displayInfo();
        cout << "Осталось жизней: " << lives << endl;
    }

    // Перегрузка метода с разными параметрами
    void playWith(const string& toy) const {
        cout << name << " играет с " << toy << endl;
    }

    void playWith(int hours) const {
        cout << name << " играет " << hours << " часов" << endl;
    }
};

int main() {
    cout << "=== Демонстрация работы с классами и перегрузкой методов ===\n\n";

    // Часть 1: Дроби с перегрузкой операторов
    cout << "1. Класс Fraction (Дробь):\n";

    Fraction f1(1, 2);
    Fraction f2(3, 4);
    Fraction f3(2, 4); // 1/2 после упрощения

    cout << "f1 = " << f1 << endl;

```

```

cout << "f2 = " << f2 << endl;
cout << "f3 = " << f3 << endl;

cout << "\nАрифметические операции:\n";
cout << f1 << " + " << f2 << " = " << (f1 + f2) << endl;
cout << f1 << " - " << f2 << " = " << (f1 - f2) << endl;
cout << f1 << " * " << f2 << " = " << (f1 * f2) << endl;
cout << f1 << " / " << f2 << " = " << (f1 / f2) << endl;

cout << "\nСравнение:\n";
cout << f1 << " == " << f3 << " ? " << (f1 == f3 ? "Да" : "Нет") << endl;
cout << f1 << " < " << f2 << " ? " << (f1 < f2 ? "Да" : "Нет") << endl;

cout << "\nПреобразование типа:\n";
cout << f2 << " как double = " << static_cast<double>(f2) << endl;

cout << "\nОператор индексации:\n";
cout << "f1[0] = " << f1[0] << " (числитель), f1[1] = " << f1[1] << " (знаменатель)\n";

// Часть 2: Векторы с перегрузкой методов
cout << "\n\n2. Класс Vector3D:\n";

Vector3D v1(1, 2, 3);
Vector3D v2(4, 5, 6);
Vector3D v3(2); // (2, 2, 2)

cout << "v1 = " << v1 << endl;
cout << "v2 = " << v2 << endl;
cout << "v3 = " << v3 << endl;

cout << "\nОперации с векторами:\n";
cout << "v1 + v2 = " << (v1 + v2) << endl;
cout << "v2 - v1 = " << (v2 - v1) << endl;
cout << "v1 * 2.5 = " << (v1 * 2.5) << endl;
cout << "2.5 * v1 = " << (2.5 * v1) << endl;

cout << "\nМетоды векторов:\n";
cout << "Длина v1: " << v1.length() << endl;
cout << "Нормализованный v1: " << v1.normalized() << endl;
cout << "Скалярное произведение v1 и v2: " << v1.dot(v2) << endl;
cout << "Векторное произведение v1 и v2: " << v1.cross(v2) << endl;

cout << "\nПерегрузка оператора вызова функции:\n";
cout << "v1(0) = " << v1(0) << ", v1(1) = " << v1(1) << ", v1(2) = " << v1(2) << endl;

// Часть 3: Наследование и полиморфизм
cout << "\n\n3. Наследование и полиморфизм (Animal, Dog, Cat):\n";

Dog dog("Барсик", 3, "Лабрадор");
Cat cat("Мурка", 2);

cout << "\nПрямой вызов методов:\n";

```

```
dog.makeSound();
dog.displayInfo();
dog.fetch();
```

```
cout << "\n";
cat.makeSound();
cat.displayInfo();
cat.playWith("клубком");
cat.playWith(3);
```

```
// Полиморфизм через указатели на базовый класс
cout << "\nПолиморфизм (через указатели на Animal):\n";
Animal* animals[2];
animals[0] = &dog;
animals[1] = &cat;
```

```
for (int i = 0; i < 2; i++) {
    animals[i]->makeSound(); // Вызывается правильная версия метода
    animals[i]->displayInfo();
    cout << endl;
}
```

```
// Демонстрация перегрузки методов в массиве
cout << "\n4. Массив объектов с перегрузкой операторов:\n";
```

```
Fraction fractions[3] = {Fraction(1, 3), Fraction(2, 5), Fraction(3, 7)};
```

```
Fraction sum;
for (int i = 0; i < 3; i++) {
    cout << "fractions[" << i << "] = " << fractions[i] << endl;
    sum += fractions[i];
}
cout << "Сумма всех дробей: " << sum << endl;
```

```
// Демонстрация конструкторов с разными параметрами
cout << "\n5. Перегрузка конструкторов:\n";
```

```
Vector3D vectors[4] = {
    Vector3D(), // По умолчанию (0, 0, 0)
    Vector3D(5), // Скаляр (5, 5, 5)
    Vector3D(1, 0, 0), // Вектор по координатам
    Vector3D(0, 1, 0)
};
```

```
for (int i = 0; i < 4; i++) {
    cout << "vectors[" << i << "] = " << vectors[i] << endl;
}
```

```
// Обработка исключений
cout << "\n6. Обработка исключений:\n";
```

```
try {
```

```

    Fraction badFraction(1, 0);
} catch (const invalid_argument& e) {
    cout << "Поймано исключение: " << e.what() << endl;
}

try {
    Vector3D v(1, 2, 3);
    cout << "v(5) = " << v(5) << endl; // Неверный индекс
} catch (const out_of_range& e) {
    cout << "Поймано исключение: " << e.what() << endl;
}

return 0;
}

```

Задания для выполнения

Вариант 1 (Консольный - Библиотека математических классов):

- Реализуйте класс `Matrix` для работы с матрицами:**
 - Конструкторы: по умолчанию, с размерами, копирования
 - Деструктор с освобождением памяти
 - Методы: получение/установка элемента, получение размеров
 - Перегрузка операторов: `+`, `-`, `*` (умножение матриц и на скаляр), `==`, `!=`
 - Перегрузка оператора `<<` для вывода
 - Метод вычисления определителя (рекурсивно)
- Создайте класс `Polynomial` для работы с полиномами:**
 - Хранение коэффициентов в `vector<double>`
 - Конструкторы: по умолчанию, из массива коэффициентов
 - Перегрузка операторов: `+`, `-`, `*`, `/` (деление с остатком)
 - Метод вычисления значения в точке
 - Метод нахождения производной
 - Перегрузка оператора `()` для вычисления значения
- Реализуйте класс `Set` для работы с множествами:**
 - Используйте `std::vector` для хранения элементов
 - Конструкторы: по умолчанию, из массива, копирования
 - Методы: добавление/удаление элемента, проверка принадлежности
 - Перегрузка операторов: `+` (объединение), `-` (разность), `*` (пересечение)
 - Перегрузка операторов `==`, `!=`, `<` (подмножество)
 - Перегрузка оператора `[]` для доступа к элементу по индексу
- Напишите программу-демонстрацию:**
 - Создайте объекты всех классов
 - Продемонстрируйте работу всех перегруженных операторов
 - Сравните производительность разных операций
 - Обработайте возможные исключения

Вариант 2 (Qt - Графический редактор геометрических фигур):

- Создайте базовый класс `Shape`:**
 - Виртуальные методы: `draw()`, `area()`, `perimeter()`, `move()`, `scale()`
 - Поля: цвет, позиция, идентификатор
 - Виртуальный деструктор
- Реализуйте производные классы:**
 - `Circle`: радиус
 - `Rectangle`: ширина, высота
 - `Triangle`: три точки

- Каждый класс должен переопределять виртуальные методы
- 3. **Создайте класс ShapeContainer:**
 - Хранение фигур в `vector<Shape*>`
 - Методы: добавление, удаление, поиск фигур
 - Перегрузка операторов: `+=`, `-=` (добавление/удаление фигуры)
 - Перегрузка оператора `[]` для доступа к фигурам
- 4. **Разработайте Qt-приложение:**
 - Главное окно с областью рисования (`QWidget` или `QGraphicsScene`)
 - Панель инструментов для создания фигур
 - Панель свойств для изменения параметров выбранной фигуры
 - Возможность перемещения и масштабирования фигур мышью
 - Отображение информации о фигуре (площадь, периметр)
- 5. **Дополнительные возможности:**
 - Сохранение/загрузка сцены в файл
 - Undo/Redo операций
 - Группировка/разгруппировка фигур
 - Перегрузка операторов сравнения для сортировки фигур по площади

Критерии оценивания

«Удовлетворительно» (3):

- Реализованы классы с базовым функционалом
- Выполнена перегрузка основных операторов (минимум 3 оператора в каждом классе)
- Программа компилируется и демонстрирует базовую работу классов

«Хорошо» (4):

- Для **Варианта 1:** Все классы полностью реализованы с корректной работой всех операторов. Реализована обработка исключений. Есть демонстрация полиморфизма (если применимо).
- Для **Варианта 2:** Создано Qt-приложение с графическим интерфейсом. Реализованы все фигуры с переопределением методов. Есть возможность интерактивного редактирования.
- В отчете представлено UML-диаграмма классов и описание перегруженных операторов.

«Отлично» (5):

- Для **Варианта 1:** Реализованы продвинутые функции (определитель матрицы, деление полиномов с остатком, операции над множествами). Добавлены шаблонные классы для работы с разными типами данных. Проведен анализ производительности.
- Для **Варианта 2:** Реализованы все дополнительные возможности. Добавлены новые типы фигур (эллипс, многоугольник). Реализована система плагинов для добавления новых фигур. Есть возможность экспорта в различные форматы.
- **Общее:** Отчет содержит полное описание архитектуры, паттерны проектирования (если использовались), анализ преимуществ/недостатков реализации. Представлены тесты для всех методов классов.

Советы по выполнению:

1. Для класса `Matrix`:

```

cpp
template<typename T>
class Matrix {
private:
    vector<vector<T>>> data;
    int rows, cols;

public:
    // Перегрузка оператора умножения

```

```

Matrix<T> operator*(const Matrix<T>& other) const {
    if (cols != other.rows) {
        throw invalid_argument("Несовместимые размеры матриц");
    }

    Matrix<T> result(rows, other.cols);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < other.cols; j++) {
            T sum = 0;
            for (int k = 0; k < cols; k++) {
                sum += data[i][k] * other.data[k][j];
            }
            result.data[i][j] = sum;
        }
    }
    return result;
}
};

```

2. В Qt для рисования фигур:

```

cpp
void Circle::draw(QPainter& painter) const {
    painter.setBrush(QBrush(color, Qt::SolidPattern));
    painter.setPen(QPen(Qt::black, 2));
    painter.drawEllipse(center, radius, radius);

    // Подпись с площадью
    painter.drawText(center, QString("S=%1").arg(area()));
}

```

3. Для реализации Undo/Redo:

```

cpp
class Command {
public:
    virtual ~Command() {}
    virtual void execute() = 0;
    virtual void undo() = 0;
};

class MoveCommand : public Command {
private:
    Shape* shape;
    QPointF oldPos, newPos;
public:
    void execute() override { shape->setPosition(newPos); }
    void undo() override { shape->setPosition(oldPos); }
};

```

4. Использование умных указателей:

```

cpp
class ShapeContainer {
private:
    vector<unique_ptr<Shape>> shapes;
public:
    void addShape(unique_ptr<Shape> shape) {

```

```
        shapes.push_back(move(shape));  
    }  
};
```

5. **Тестирование перегруженных операторов:**

cpp

```
void testFractionOperators() {  
    Fraction a(1, 2), b(3, 4);  
    assert((a + b) == Fraction(5, 4));  
    assert((a * b) == Fraction(3, 8));  
    assert((a / b) == Fraction(2, 3));  
    cout << "Все тесты пройдены!" << endl;  
}
```

6. **Оптимизация производительности:**

- Используйте move-семантику для больших объектов
- Избегайте лишних копирований при возврате значений
- Используйте const-ссылки для параметров
- Рассмотрите возможность кэширования вычислений

ПРАКТИЧЕСКАЯ РАБОТА №7. Определение операций в классе. Создание наследованных классов. Работа с объектами через интерфейсы.

Цель работы: Освоить принципы объектно-ориентированного программирования: создание иерархии классов, определение операций внутри классов, работу с абстрактными классами и интерфейсами для реализации полиморфизма.

Задачи:

1. Создать иерархию классов для моделирования предметной области.
2. Определить операции (методы) внутри классов с учетом принципа инкапсуляции.
3. Реализовать наследование классов для повторного использования кода.
4. Создать интерфейсы (абстрактные классы) для работы с объектами через общий контракт.
5. Продемонстрировать полиморфизм через указатели на базовый класс/интерфейс.
6. (Для Qt) Создать графический интерфейс для работы с иерархией классов.

Теоретический материал

1. Определение операций (методов) в классе

Операции (методы) — это функции, определенные внутри класса, которые работают с данными объекта.

```
cpp
class BankAccount {
private:
    double balance;
    string owner;

public:
    // Конструктор
    BankAccount(string ownerName, double initialBalance = 0)
        : owner(ownerName), balance(initialBalance) {}

    // Операция пополнения счета
    void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }

    // Операция снятия денег
    bool withdraw(double amount) {
        if (amount > 0 && balance >= amount) {
            balance -= amount;
        }
    }
}
```

```

        return true;
    }
    return false;
}

// Операция получения баланса
double getBalance() const {
    return balance;
}
};

```

2. Создание наследованных классов

Наследование — механизм создания нового класса на основе существующего с возможностью добавления или изменения функциональности.

```

cpp
// Базовый класс
class Employee {
protected:
    string name;
    double salary;

public:
    Employee(string empName, double empSalary)
        : name(empName), salary(empSalary) {}

    virtual double calculateBonus() const {
        return salary * 0.1; // 10% от зарплаты
    }

    virtual void displayInfo() const {
        cout << "Сотрудник: " << name << ", Зарплата: " << salary;
    }
};

// Наследованный класс
class Manager : public Employee {
private:
    double bonusPercentage;

public:
    Manager(string name, double salary, double bonus)
        : Employee(name, salary), bonusPercentage(bonus) {}
};

```

```

// Переопределение метода
double calculateBonus() const override {
    return salary * bonusPercentage;
}

// Новый метод
void conductMeeting() const {
    cout << name << " проводит собрание" << endl;
}

void displayInfo() const override {
    Employee::displayInfo();
    cout << ", Должность: Менеджер" << endl;
}
};

```

3. Работа с объектами через интерфейсы

Интерфейс (абстрактный класс) — класс, содержащий только чистые виртуальные методы, который определяет контракт для производных классов.

```

cpp
// Абстрактный класс (интерфейс)
class IPrintable {
public:
    virtual void print() const = 0; // Чистая виртуальная функция
    virtual ~IPrintable() {}
};

class ICalculable {
public:
    virtual double calculate() const = 0;
    virtual ~ICalculable() {}
};

// Класс, реализующий интерфейсы
class Document : public IPrintable, public ICalculable {
private:
    string content;
    int pageCount;

public:
    Document(string text, int pages) : content(text), pageCount(pages) {}

    void print() const override {

```

```

        cout << "Документ: " << content.substr(0, 50) << "..." << endl;
    }

    double calculate() const override {
        return pageCount * 0.5; // Стоимость печати
    }
};

```

Пример решения (Консольное приложение на C++)

```

cpp
#include <iostream>
#include <string>
#include <vector>
#include <memory>
#include <cmath>

using namespace std;

// =====
// 1. БАЗОВЫЙ КЛАСС И ИНТЕРФЕЙСЫ
// =====

// Интерфейс для объектов, которые можно отображать
class IDisplayable {
public:
    virtual void display() const = 0;
    virtual ~IDisplayable() = default;
};

// Интерфейс для объектов, которые можно вычислять
class ICalculable {
public:
    virtual double calculateArea() const = 0;
    virtual double calculatePerimeter() const = 0;
    virtual ~ICalculable() = default;
};

// Интерфейс для объектов, которые можно перемещать
class IMovable {
public:
    virtual void move(double dx, double dy) = 0;

```

```

    virtual ~IMovable() = default;
};

// Базовый класс для геометрических фигур
class Shape : public IDisplayable, public ICalculable, public IMovable {
protected:
    double x, y; // Позиция центра фигуры
    string color;

public:
    Shape(double posX = 0, double posY = 0, string col = "black")
        : x(posX), y(posY), color(col) {}

    virtual ~Shape() {}

    // Реализация методов интерфейса IMovable
    void move(double dx, double dy) override {
        x += dx;
        y += dy;
    }

    // Методы доступа
    double getX() const { return x; }
    double getY() const { return y; }
    string getColor() const { return color; }

    void setColor(string newColor) { color = newColor; }

    // Метод базового класса (не виртуальный)
    void printPosition() const {
        cout << "Позиция: (" << x << ", " << y << ")" << endl;
    }
};

// =====
// 2. НАСЛЕДОВАННЫЕ КЛАССЫ ФИГУР
// =====

// Класс Круг
class Circle : public Shape {
private:
    double radius;

public:

```

```
Circle(double posX = 0, double posY = 0, double r = 1, string col = "black")
: Shape(posX, posY, col), radius(r) {}
```

```
// Реализация методов интерфейса IDisplayable
```

```
void display() const override {
    cout << "○ Круг: " << endl;
    cout << " Центр: (" << x << ", " << y << ")" << endl;
    cout << " Радиус: " << radius << endl;
    cout << " Цвет: " << color << endl;
}
```

```
// Реализация методов интерфейса ICalculable
```

```
double calculateArea() const override {
    return 3.14159 * radius * radius;
}
```

```
double calculatePerimeter() const override {
    return 2 * 3.14159 * radius;
}
```

```
// Специфичные методы для круга
```

```
double getRadius() const { return radius; }
void setRadius(double r) { radius = r; }
```

```
// Переопределение метода move с дополнительной логикой
```

```
void move(double dx, double dy) override {
    Shape::move(dx, dy);
    cout << "Круг перемещен на (" << dx << ", " << dy << ")" << endl;
}
};
```

```
// Класс Прямоугольник
```

```
class Rectangle : public Shape {
private:
    double width, height;

public:
    Rectangle(double posX = 0, double posY = 0,
               double w = 1, double h = 1, string col = "black")
: Shape(posX, posY, col), width(w), height(h) {}
```

```
// Реализация методов интерфейса IDisplayable
```

```
void display() const override {
    cout << "□ Прямоугольник: " << endl;
```

```

        cout << " Центр: (" << x << ", " << y << ")" << endl;
        cout << " Ширина: " << width << ", Высота: " << height << endl;
        cout << " Цвет: " << color << endl;
    }

    // Реализация методов интерфейса ICalculable
    double calculateArea() const override {
        return width * height;
    }

    double calculatePerimeter() const override {
        return 2 * (width + height);
    }

    // Специфичные методы для прямоугольника
    double getWidth() const { return width; }
    double getHeight() const { return height; }
    void setSize(double w, double h) { width = w; height = h; }

    // Новый метод для прямоугольника
    bool isSquare() const {
        return width == height;
    }
};

// Класс Треугольник
class Triangle : public Shape {
private:
    double side1, side2, side3;

public:
    Triangle(double posX = 0, double posY = 0,
             double a = 1, double b = 1, double c = 1, string col = "black")
        : Shape(posX, posY, col), side1(a), side2(b), side3(c) {}

    // Реализация методов интерфейса IDisplayable
    void display() const override {
        cout << "Δ Треугольник: " << endl;
        cout << " Центр: (" << x << ", " << y << ")" << endl;
        cout << " Стороны: " << side1 << ", " << side2 << ", " << side3 << endl;
        cout << " Цвет: " << color << endl;
    }

    // Реализация методов интерфейса ICalculable

```

```

double calculateArea() const override {
    // Формула Герона
    double p = (side1 + side2 + side3) / 2;
    return sqrt(p * (p - side1) * (p - side2) * (p - side3));
}

double calculatePerimeter() const override {
    return side1 + side2 + side3;
}

// Специфичные методы для треугольника
double getSide1() const { return side1; }
double getSide2() const { return side2; }
double getSide3() const { return side3; }

// Проверка типа треугольника
string getTriangleType() const {
    if (side1 == side2 && side2 == side3) return "Равносторонний";
    if (side1 == side2 || side1 == side3 || side2 == side3) return "Равнобедренный";
    return "Разносторонний";
}
};

// =====
// 3. КЛАСС ДЛЯ РАБОТЫ С КОЛЛЕКЦИЕЙ ФИГУР
// =====

class ShapeManager {
private:
    vector<unique_ptr<Shape>> shapes;

public:
    // Добавление фигуры через указатель на базовый класс
    void addShape(unique_ptr<Shape> shape) {
        shapes.push_back(move(shape));
    }

    // Отображение всех фигур
    void displayAll() const {
        cout << "\n=== ВСЕ ФИГУРЫ (" << shapes.size() << ") ===" << endl;
        for (const auto& shape : shapes) {
            shape->display();
            cout << " Площадь: " << shape->calculateArea()
                 << ", Периметр: " << shape->calculatePerimeter() << endl;
        }
    }
};

```

```

        cout << endl;
    }
}

// Вычисление общей площади
double getTotalArea() const {
    double total = 0;
    for (const auto& shape : shapes) {
        total += shape->calculateArea();
    }
    return total;
}

// Перемещение всех фигур
void moveAll(double dx, double dy) {
    for (auto& shape : shapes) {
        shape->move(dx, dy);
    }
}

// Поиск фигур по критерию (используем лямбда-функцию)
vector<Shape*> findShapesByCriteria(function<bool(const Shape*)> criteria) const {
    vector<Shape*> result;
    for (const auto& shape : shapes) {
        if (criteria(shape.get())) {
            result.push_back(shape.get());
        }
    }
    return result;
}

// Получение количества фигур
size_t getShapeCount() const {
    return shapes.size();
}
};

// =====
// 4. СПЕЦИАЛЬНЫЙ КЛАСС С ДОПОЛНИТЕЛЬНЫМИ ОПЕРАЦИЯМИ
// =====

// Класс Композитная фигура (состоит из нескольких фигур)
class CompositeShape : public Shape {
private:

```

```
vector<unique_ptr<Shape>> components;
```

public:

```
CompositeShape(double posX = 0, double posY = 0, string col = "black")  
    : Shape(posX, posY, col) {}
```

// Добавление компонента

```
void addComponent(unique_ptr<Shape> component) {  
    components.push_back(move(component));  
}
```

// Реализация методов интерфейса IDisplayable

```
void display() const override {  
    cout << "□ Композитная фигура: " << endl;  
    cout << " Позиция: (" << x << ", " << y << ")" << endl;  
    cout << " Цвет: " << color << endl;  
    cout << " Количество компонентов: " << components.size() << endl;  
  
    for (size_t i = 0; i < components.size(); i++) {  
        cout << " Компонент " << i+1 << ": ";  
        components[i]->display();  
    }  
}
```

// Реализация методов интерфейса ICalculable

```
double calculateArea() const override {  
    double totalArea = 0;  
    for (const auto& component : components) {  
        totalArea += component->calculateArea();  
    }  
    return totalArea;  
}
```

```
double calculatePerimeter() const override {  
    double totalPerimeter = 0;  
    for (const auto& component : components) {  
        totalPerimeter += component->calculatePerimeter();  
    }  
    return totalPerimeter;  
}
```

// Переопределение метода move для перемещения всех компонентов

```
void move(double dx, double dy) override {  
    Shape::move(dx, dy);
```

```

        for (auto& component : components) {
            component->move(dx, dy);
        }
    }
};

// =====
// ОСНОВНАЯ ПРОГРАММА
// =====

int main() {
    cout << "=== ПРАКТИЧЕСКАЯ РАБОТА №7: КЛАССЫ, НАСЛЕДОВАНИЕ, ИНТЕРФЕЙСЫ ===\n\n";

    // Создание менеджера фигур
    ShapeManager manager;

    // ===== ЧАСТЬ 1: СОЗДАНИЕ ФИГУР =====
    cout << "1. СОЗДАНИЕ ФИГУР И ДОБАВЛЕНИЕ В МЕНЕДЖЕР:\n";

    // Создание различных фигур
    auto circle1 = make_unique<Circle>(10, 10, 5, "red");
    auto rectangle1 = make_unique<Rectangle>(20, 30, 8, 6, "blue");
    auto triangle1 = make_unique<Triangle>(40, 50, 3, 4, 5, "green");
    auto circle2 = make_unique<Circle>(60, 70, 7, "yellow");

    // Добавление фигур в менеджер
    manager.addShape(move(circle1));
    manager.addShape(move(rectangle1));
    manager.addShape(move(triangle1));
    manager.addShape(move(circle2));

    cout << "Добавлено " << manager.getShapeCount() << " фигур\n";

    // ===== ЧАСТЬ 2: РАБОТА С ФИГУРАМИ =====
    cout << "\n\n2. РАБОТА С ФИГУРАМИ ЧЕРЕЗ ИНТЕРФЕЙСЫ:\n";

    // Демонстрация полиморфизма
    cout << "\na) Отображение всех фигур (через IDisplayable):\n";
    manager.displayAll();

    cout << "\nb) Вычисление площадей (через ICalculable):\n";
    cout << "Общая площадь всех фигур: " << manager.getTotalArea() << endl;

    cout << "\nv) Перемещение всех фигур (через IMovable):\n";

```

```

manager.moveAll(5, 5);
cout << "Все фигуры перемещены на (5, 5)\n";

// ===== ЧАСТЬ 3: ПОИСК И ФИЛЬТРАЦИЯ =====
cout << "\n\n3. ПОИСК ФИГУР ПО КРИТЕРИЯМ:\n";

// Поиск фигур с площадью больше 50
cout << "\na) Фигуры с площадью > 50:\n";
auto largeShapes = manager.findShapesByCriteria(
    [](const Shape* shape) { return shape->calculateArea() > 50; }
);

for (auto shape : largeShapes) {
    shape->display();
    cout << "    Площадь: " << shape->calculateArea() << endl;
}

// ===== ЧАСТЬ 4: КОМПОЗИТНАЯ ФИГУРА =====
cout << "\n\n4. РАБОТА С КОМПОЗИТНОЙ ФИГУРОЙ:\n";

auto composite = make_unique<CompositeShape>(100, 100, "purple");

// Добавляем компоненты в композитную фигуру
composite->addComponent(make_unique<Circle>(0, 0, 3, "red"));
composite->addComponent(make_unique<Rectangle>(0, 0, 4, 4, "blue"));
composite->addComponent(make_unique<Triangle>(0, 0, 3, 4, 5, "green"));

// Добавляем композитную фигуру в менеджер
manager.addShape(move(composite));

// Отображаем композитную фигуру
cout << "Композитная фигура добавлена в менеджер\n";

// ===== ЧАСТЬ 5: РАБОТА С УКАЗАТЕЛЯМИ =====
cout << "\n\n5. РАБОТА С УКАЗАТЕЛЯМИ НА БАЗОВЫЙ КЛАСС:\n";

// Создаем массив указателей на базовый класс
vector<Shape*> shapePointers;

// Добавляем новые фигуры
auto circlePtr = new Circle(200, 200, 10, "orange");
auto rectPtr = new Rectangle(300, 300, 15, 10, "pink");

shapePointers.push_back(circlePtr);

```

```
shapePointers.push_back(rectPtr);
```

```
// Работаем с фигурами через указатели на базовый класс
```

```
cout << "Фигуры в массиве указателей:\n";  
for (size_t i = 0; i < shapePointers.size(); i++) {  
    cout << "\nФигура " << i+1 << ":\n";  
    shapePointers[i]->display();  
    cout << " Площадь: " << shapePointers[i]->calculateArea() << endl;
```

```
// Попытка приведения к конкретному типу (dynamic_cast)
```

```
if (auto circle = dynamic_cast<Circle*>(shapePointers[i])) {  
    cout << " Это круг с радиусом: " << circle->getRadius() << endl;  
} else if (auto rectangle = dynamic_cast<Rectangle*>(shapePointers[i])) {  
    cout << " Это прямоугольник размером: "  
        << rectangle->getWidth() << "x" << rectangle->getHeight() << endl;  
    if (rectangle->isSquare()) {  
        cout << " Это квадрат!" << endl;  
    }  
}  
}
```

```
// ===== ЧАСТЬ 6: ДЕМОНСТРАЦИЯ НАСЛЕДОВАНИЯ =====  
=====
```

```
cout << "\n\n6. ДЕМОНСТРАЦИЯ НАСЛЕДОВАНИЯ И ПЕРЕОПРЕДЕЛЕНИЯ МЕТОДОВ:\n";
```

```
// Создаем фигуры
```

```
Circle demoCircle(400, 400, 8, "cyan");  
Rectangle demoRect(450, 450, 12, 8, "magenta");  
Triangle demoTriangle(500, 500, 5, 12, 13, "brown");
```

```
// Используем полиморфизм через указатели на базовый класс
```

```
Shape* shapesArray[] = {&demoCircle, &demoRect, &demoTriangle};
```

```
for (int i = 0; i < 3; i++) {  
    cout << "\nФигура " << i+1 << " через указатель на Shape:\n";  
    shapesArray[i]->display();  
    shapesArray[i]->printPosition(); // Метод базового класса  
    cout << " Тип фигуры: ";
```

```
// RTTI (Run-Time Type Identification)
```

```
if (typeid(*shapesArray[i]) == typeid(Circle)) {  
    cout << "Круг" << endl;  
} else if (typeid(*shapesArray[i]) == typeid(Rectangle)) {  
    cout << "Прямоугольник" << endl;
```

```

    } else if (typeid(*shapesArray[i]) == typeid(Triangle)) {
        cout << "Треугольник" << endl;
    }
}

// ===== ЧАСТЬ 7: ОЧИСТКА ПАМЯТИ =====

cout << "\n\n7. ОЧИСТКА РЕСУРСОВ:\n";

// Очищаем динамически выделенную память
for (auto ptr : shapePointers) {
    delete ptr;
}
shapePointers.clear();

cout << "Динамическая память освобождена\n";
cout << "Всего фигур в менеджере: " << manager.getShapeCount() << endl;

// Финальный вывод
cout << "\n\n=== ИТОГИ РАБОТЫ ===\n";
cout << "1. Создана иерархия классов: Shape -> Circle, Rectangle, Triangle, CompositeShape\n";
cout << "2. Реализованы интерфейсы: IDisplayable, ICalculable, IMovable\n";
cout << "3. Продемонстрирован полиморфизм через указатели на базовый класс\n";
cout << "4. Создан менеджер для работы с коллекцией фигур\n";
cout << "5. Реализованы операции: отображение, вычисление, перемещение, поиск\n";

return 0;
}

```

Задания для выполнения

Вариант 1 (Консольный - Система управления библиотекой):

1. Создайте базовый класс `LibraryItem` с интерфейсами:

- `IBorrowable` (можно взять/вернуть)
- `IDisplayable` (можно отобразить информацию)
- `ISearchable` (можно искать по критериям)

2. Реализуйте наследованные классы:

- `Book` (книга): автор, ISBN, количество страниц
- `Magazine` (журнал): номер, месяц выпуска, издательство
- `DVD` (фильм): режиссер, продолжительность, жанр

3. Определите операции в каждом классе:

- Конструкторы с параметрами

- Геттеры/сеттеры
- Переопределение методов интерфейсов
- Специфичные методы (например, `Book::getReadingTime()`)
- 4. **Создайте класс `Library` для управления:**
 - Хранение элементов в коллекции
 - Добавление/удаление элементов
 - Поиск по различным критериям
 - Статистика (количество книг, журналов и т.д.)
- 5. **Напишите демонстрационную программу:**
 - Создание объектов разных типов
 - Работа через указатели на базовый класс
 - Демонстрация полиморфизма
 - Тестирование всех операций

Вариант 2 (Qt - Система учета сотрудников компании):

1. **Создайте интерфейсы для сотрудников:**
 - `IEmployee` (базовый интерфейс)
 - `ICalculable` (расчет зарплаты)
 - `IReportable` (формирование отчета)
2. **Реализуйте иерархию классов сотрудников:**
 - `Employee` (базовый класс): имя, должность, стаж
 - `Manager` (менеджер): отдел, количество подчиненных
 - `Developer` (разработчик): язык программирования, уровень
 - `SalesPerson` (продавец): объем продаж, процент
3. **Определите операции расчета зарплаты:**
 - Для `Manager`: оклад + бонус за подчиненных
 - Для `Developer`: оклад + надбавка за уровень
 - Для `SalesPerson`: оклад + процент от продаж
4. **Создайте Qt-приложение с интерфейсом:**
 - Главное окно с таблицей сотрудников
 - Форма добавления/редактирования сотрудников
 - Расчет и отображение зарплаты
 - Формирование отчетов в разных форматах
5. **Дополнительные функции:**
 - Сохранение/загрузка данных в файл
 - Фильтрация и сортировка сотрудников
 - Визуализация статистики (графики, диаграммы)
 - Экспорт отчетов в PDF/Excel

Критерии оценивания

«Удовлетворительно» (3):

- Создана иерархия классов (минимум 3 класса)
- Реализованы базовые интерфейсы

- Программа компилируется и работает
- Продемонстрирован базовый полиморфизм

«Хорошо» (4):

- Для **Варианта 1**: Полная реализация системы библиотеки. Корректная работа всех интерфейсов. Реализован поиск и фильтрация. Обработка исключений.
- Для **Варианта 2**: Рабочее Qt-приложение с графическим интерфейсом. Корректный расчет зарплаты для разных типов сотрудников. Сохранение данных.
- В отчете представлены UML-диаграммы классов и последовательностей.

«Отлично» (5):

- Для **Варианта 1**: Реализованы паттерны проектирования (Factory, Observer). Добавлена система логирования. Реализована сериализация объектов. Написаны unit-тесты.
- Для **Варианта 2**: Реализованы все дополнительные функции. Добавлены новые типы сотрудников. Реализована система аутентификации и ролей. Поддержка нескольких языков интерфейса.
- **Общее**: Отчет содержит полное описание архитектуры, анализ выбранных решений, сравнение альтернативных подходов.

Советы по выполнению:

1. Для интерфейсов используйте абстрактные классы:

```
сpp
class IBorrowable {
public:
    virtual bool borrow(const string& borrower) = 0;
    virtual bool returnItem() = 0;
    virtual bool isAvailable() const = 0;
    virtual ~IBorrowable() = default;
};
```

2. Используйте умные указатели для управления памятью:

```
сpp
class Library {
private:
    vector<unique_ptr<LibraryItem>> items;
public:
    void addItem(unique_ptr<LibraryItem> item) {
        items.push_back(move(item));
    }
};
```

3. Для Qt-приложения используйте модель-представление:

```
сpp
class EmployeeModel : public QAbstractTableModel {
```

```

    Q_OBJECT
private:
    vector<unique_ptr<Employee>> employees;
public:
    // Реализация виртуальных методов модели
    int rowCount(const QModelIndex& parent = QModelIndex()) const override {
        return employees.size();
    }
    // ... остальные методы
};

```

4. Реализуйте фабрику для создания объектов:

```

cpp
class EmployeeFactory {
public:
    static unique_ptr<Employee> createEmployee(
        EmployeeType type, const string& name, // параметры
    ) {
        switch(type) {
            case EmployeeType::Manager:
                return make_unique<Manager>(name, /* параметры */);
            case EmployeeType::Developer:
                return make_unique<Developer>(name, /* параметры */);
            // ...
        }
    }
};

```

5. Добавьте поддержку сериализации:

```

cpp
class Serializable {
public:
    virtual string toJson() const = 0;
    virtual void fromJson(const string& json) = 0;
    virtual ~Serializable() = default;
};

```

6. Напишите unit-тесты:

```

cpp
TEST(EmployeeTest, SalaryCalculation) {
    Manager manager("Иван", 50000, 5);
    ASSERT_NEAR(manager.calculateSalary(), 57500, 0.01);
}

```

ПРАКТИЧЕСКАЯ РАБОТА №8. Использование стандартных интерфейсов. Работа с типом данных структура.

Цель работы: Освоить использование стандартных интерфейсов (STL) C++ для работы с контейнерами и алгоритмами, изучить особенности типа данных `struct` и его применение в объектно-ориентированном программировании, научиться создавать пользовательские структуры данных.

Задачи:

1. Изучить стандартные интерфейсы STL (Standard Template Library).
2. Освоить работу со структурами (`struct`) в C++.
3. Научиться использовать стандартные контейнеры для хранения структур.
4. Применять алгоритмы STL для обработки структур данных.
5. Реализовать пользовательские структуры с перегрузкой операторов.
6. (Для Qt) Создать графический интерфейс для визуализации структур данных.

Теоретический материал

1. Структуры (`struct`) в C++

Структура — составной тип данных, объединяющий набор переменных разного типа под одним именем.

`.cpp`

// Объявление структуры

```
struct Student {
```

// Публичные поля по умолчанию

```
    int id;
```

```
    string name;
```

```
    double averageGrade;
```

```
    vector<string> courses;
```

// Конструкторы

```
    Student() : id(0), name(""), averageGrade(0.0) {}
```

```
    Student(int i, string n, double g) : id(i), name(n), averageGrade(g) {}
```

// Методы структуры

```
    void display() const {
```

```
        cout << "ID: " << id << ", Name: " << name  
            << ", Grade: " << averageGrade << endl;
```

```
    }
```

```
    bool isExcellent() const {
```

```
        return averageGrade >= 4.5;
```

```
    }
```

```
};
```

// Отличие struct от class:

// - В struct все члены по умолчанию public

// - В class все члены по умолчанию private

// - struct обычно используется для POD (Plain Old Data) типов

2. Стандартные интерфейсы STL

Контейнеры STL:

- **Последовательные:** `vector`, `list`, `deque`
- **Ассоциативные:** `map`, `set`, `unordered_map`, `unordered_set`
- **Адаптеры:** `stack`, `queue`, `priority_queue`

Итераторы STL: объекты для доступа к элементам контейнеров

```

cpp
vector<int> v = {1, 2, 3, 4, 5};
for (auto it = v.begin(); it != v.end(); ++it) {
    cout << *it << " ";
}

```

Алгоритмы STL: готовые функции для работы с контейнерами

```

cpp
#include <algorithm>
#include <numeric>
#include <functional>

vector<int> numbers = {5, 2, 8, 1, 9};
sort(numbers.begin(), numbers.end()); // Сортировка
auto maxIt = max_element(numbers.begin(), numbers.end()); // Максимум
int sum = accumulate(numbers.begin(), numbers.end(), 0); // Сумма

```

3. Лямбда-выражения (C++11 и выше)

```

cpp
vector<int> nums = {1, 2, 3, 4, 5};
// Фильтрация четных чисел с помощью лямбды
nums.erase(remove_if(nums.begin(), nums.end(),
    [](int n) { return n % 2 == 0; }), nums.end());

```

4. Пользовательские компараторы для структур

```

cpp
struct Person {
    string name;
    int age;
    double salary;
};

// Компаратор для сортировки по возрасту
bool compareByAge(const Person& a, const Person& b) {
    return a.age < b.age;
}

```

```

// Использование с sort
vector<Person> people;
sort(people.begin(), people.end(), compareByAge);

// Или с лямбдой
sort(people.begin(), people.end(),
    [](const Person& a, const Person& b) { return a.salary > b.salary; });

```

Пример решения (Консольное приложение на C++)

```

cpp
#include <iostream>
#include <vector>
#include <list>
#include <map>
#include <set>
#include <algorithm>
#include <numeric>
#include <string>

```

```

#include <fstream>
#include <memory>
#include <iomanip>
#include <functional>
#include <random>

using namespace std;

// =====
// 1. БАЗОВЫЕ СТРУКТУРЫ ДАННЫХ
// =====

// Структура для точки в 2D пространстве
struct Point2D {
    double x, y;

    // Конструкторы
    Point2D() : x(0.0), y(0.0) {}
    Point2D(double xCoord, double yCoord) : x(xCoord), y(yCoord) {}

    // Методы
    double distanceTo(const Point2D& other) const {
        double dx = x - other.x;
        double dy = y - other.y;
        return sqrt(dx*dx + dy*dy);
    }

    void display() const {
        cout << "(" << x << ", " << y << ")";
    }

    // Перегрузка операторов
    bool operator==(const Point2D& other) const {
        return x == other.x && y == other.y;
    }

    bool operator<(const Point2D& other) const {
        if (x != other.x) return x < other.x;
        return y < other.y;
    }

    Point2D operator+(const Point2D& other) const {
        return Point2D(x + other.x, y + other.y);
    }

    Point2D operator-(const Point2D& other) const {
        return Point2D(x - other.x, y - other.y);
    }

    // Дружественная функция для вывода
    friend ostream& operator<<(ostream& os, const Point2D& p) {
        os << "(" << p.x << ", " << p.y << ")";
    }

```

```

        return os;
    }
};

```

// Структура для студента с использованием STL контейнеров внутри

```

struct Student {
    int id;
    string firstName;
    string lastName;
    vector<int> grades; // Оценки по предметам
    map<string, int> examResults; // Результаты экзаменов

```

// Конструкторы

```

Student() : id(0) {}
Student(int studentId, string fName, string lName)
    : id(studentId), firstName(fName), lastName(lName) {}

```

// Методы

```

double averageGrade() const {
    if (grades.empty()) return 0.0;
    return accumulate(grades.begin(), grades.end(), 0.0) / grades.size();
}

```

```

void addGrade(int grade) {
    grades.push_back(grade);
}

```

```

void addExamResult(const string& subject, int score) {
    examResults[subject] = score;
}

```

```

void display() const {
    cout << "Студент #" << id << ": " << lastName << " " << firstName << endl;
    cout << " Средний балл: " << fixed << setprecision(2) << averageGrade() << endl;

    if (!grades.empty()) {
        cout << " Оценки: ";
        for (int grade : grades) cout << grade << " ";
        cout << endl;
    }

```

```

    if (!examResults.empty()) {
        cout << " Результаты экзаменов:" << endl;
        for (const auto& [subject, score] : examResults) {
            cout << "   " << subject << ": " << score << endl;
        }
    }
}

```

// Компараторы как методы

```

static bool compareByLastName(const Student& a, const Student& b) {
    return a.lastName < b.lastName;
}

```

```

    }

    static bool compareByAverageGrade(const Student& a, const Student& b) {
        return a.averageGrade() > b.averageGrade(); // По убыванию
    }
};

// Структура для заказа в интернет-магазине
struct Order {
    int orderId;
    string customerName;
    vector<pair<string, double>> items; // товар -> цена
    double totalAmount;
    string status; // "pending", "processing", "completed", "cancelled"

    Order(int id, string customer)
        : orderId(id), customerName(customer), totalAmount(0.0), status("pending") {}

    void addItem(const string& itemName, double price) {
        items.emplace_back(itemName, price);
        totalAmount += price;
    }

    void updateStatus(const string& newStatus) {
        status = newStatus;
    }

    void display() const {
        cout << "Заказ #" << orderId << " (" << status << ")" << endl;
        cout << " Клиент: " << customerName << endl;
        cout << " Сумма: $" << fixed << setprecision(2) << totalAmount << endl;
        cout << " Товары:" << endl;
        for (const auto& [item, price] : items) {
            cout << "   - " << item << ": $" << price << endl;
        }
    }
};

// =====
// 2. КЛАССЫ ДЛЯ РАБОТЫ СО СТРУКТУРАМИ ЧЕРЕЗ STL
// =====

class StudentDatabase {
private:
    vector<Student> students;
    map<int, Student*> idIndex; // Быстрый поиск по ID

public:
    // Добавление студента
    void addStudent(const Student& student) {
        students.push_back(student);
        idIndex[student.id] = &students.back();
    }
};

```

```

}

// Поиск студента по ID
Student* findStudentById(int id) {
    auto it = idIndex.find(id);
    return (it != idIndex.end()) ? it->second : nullptr;
}

// Получение всех студентов
const vector<Student>& getAllStudents() const {
    return students;
}

// Сортировка студентов по фамилии
void sortByLastName() {
    sort(students.begin(), students.end(), Student::compareByLastName);
}

// Сортировка студентов по среднему баллу (по убыванию)
void sortByAverageGrade() {
    sort(students.begin(), students.end(), Student::compareByAverageGrade);
}

// Фильтрация отличников (средний балл >= 4.5)
vector<Student> getExcellentStudents() const {
    vector<Student> result;
    copy_if(students.begin(), students.end(), back_inserter(result),
        [](const Student& s) { return s.averageGrade() >= 4.5; });
    return result;
}

// Статистика по группе
void printStatistics() const {
    if (students.empty()) {
        cout << "База данных пуста" << endl;
        return;
    }

    // Использование алгоритмов STL для вычислений
    vector<double> averages;
    transform(students.begin(), students.end(), back_inserter(averages),
        [](const Student& s) { return s.averageGrade(); });

    double totalAvg = accumulate(averages.begin(), averages.end(), 0.0) / averages.size();
    double maxAvg = *max_element(averages.begin(), averages.end());
    double minAvg = *min_element(averages.begin(), averages.end());

    cout << "\n=== СТАТИСТИКА ПО СТУДЕНТАМ ===" << endl;
    cout << "Всего студентов: " << students.size() << endl;
    cout << "Средний балл по группе: " << fixed << setprecision(2) << totalAvg << endl;
    cout << "Максимальный средний балл: " << maxAvg << endl;
    cout << "Минимальный средний балл: " << minAvg << endl;
}

```

```

    // Количество отличников
    int excellentCount = count_if(students.begin(), students.end(),
        [](const Student& s) { return s.averageGrade() >= 4.5; });
    cout << "Количество отличников: " << excellentCount << endl;
}

// Сохранение в файл
bool saveToFile(const string& filename) const {
    ofstream file(filename);
    if (!file.is_open()) return false;

    for (const auto& student : students) {
        file << student.id << "," << student.lastName << ","
            << student.firstName << "," << student.averageGrade() << endl;
    }

    file.close();
    return true;
}

};

class PointCloud {
private:
    set<Point2D> points; // Используем set для уникальных точек

public:
    // Добавление точки
    void addPoint(const Point2D& point) {
        points.insert(point);
    }

    // Добавление нескольких точек
    void addPoints(const vector<Point2D>& newPoints) {
        points.insert(newPoints.begin(), newPoints.end());
    }

    // Поиск ближайшей точки к заданной
    Point2D findNearestPoint(const Point2D& target) const {
        if (points.empty()) return Point2D();

        // Используем алгоритм min_element с лямбдой
        auto nearest = min_element(points.begin(), points.end(),
            [&target](const Point2D& a, const Point2D& b) {
                return a.distanceTo(target) < b.distanceTo(target);
            });

        return *nearest;
    }

    // Поиск точек в заданном радиусе
    vector<Point2D> findPointsInRadius(const Point2D& center, double radius) const {

```

```

vector<Point2D> result;

copy_if(points.begin(), points.end(), back_inserter(result),
 [&center, radius](const Point2D& p) {
    return p.distanceTo(center) <= radius;
});

return result;
}

// Вычисление центра масс
Point2D calculateCenterOfMass() const {
    if (points.empty()) return Point2D();

    Point2D sum;
    for (const auto& point : points) {
        sum = sum + point;
    }

    return Point2D(sum.x / points.size(), sum.y / points.size());
}

// Отображение всех точек
void display() const {
    cout << "Облако точек (" << points.size() << " точек):" << endl;
    int i = 1;
    for (const auto& point : points) {
        cout << " " << i++ << ": " << point << endl;
    }
}

// Получение всех точек
const set<Point2D>& getPoints() const {
    return points;
}
};

// =====
// 3. ИСПОЛЬЗОВАНИЕ STL АЛГОРИТМОВ СО СТРУКТУРАМИ
// =====

void demonstrateSTLAlgorithms() {
    cout << "\n=== ДЕМОНСТРАЦИЯ АЛГОРИТМОВ STL ===" << endl;

    // Создание вектора структур
    vector<Point2D> points = {
        Point2D(1, 2), Point2D(3, 4), Point2D(5, 1),
        Point2D(2, 3), Point2D(4, 5), Point2D(0, 0)
    };

    // 1. Сортировка с пользовательским компаратором
    cout << "\n1. Сортировка точек по расстоянию от начала координат:" << endl;

```

```

sort(points.begin(), points.end(),
    [](const Point2D& a, const Point2D& b) {
        Point2D origin(0, 0);
        return a.distanceTo(origin) < b.distanceTo(origin);
    });

for (const auto& p : points) {
    cout << " " << p << " (расстояние: " << p.distanceTo(Point2D(0, 0)) << ")" << endl;
}

// 2. Поиск с помощью find_if
cout << "\n2. Поиск точки с x > 3:" << endl;
auto it = find_if(points.begin(), points.end(),
    [](const Point2D& p) { return p.x > 3; });

if (it != points.end()) {
    cout << " Найдена: " << *it << endl;
}

// 3. Подсчет с помощью count_if
cout << "\n3. Количество точек в первом квадранте (x>0, y>0):" << endl;
int count = count_if(points.begin(), points.end(),
    [](const Point2D& p) { return p.x > 0 && p.y > 0; });
cout << " Результат: " << count << " точек" << endl;

// 4. Преобразование с помощью transform
cout << "\n4. Преобразование: вычисление расстояний до точки (2,2):" << endl;
vector<double> distances;
Point2D reference(2, 2);

transform(points.begin(), points.end(), back_inserter(distances),
    [&reference](const Point2D& p) { return p.distanceTo(reference); });

for (size_t i = 0; i < points.size(); i++) {
    cout << " " << points[i] << " -> " << distances[i] << endl;
}

// 5. Удаление с помощью remove_if
cout << "\n5. Удаление точек с x < 2:" << endl;
points.erase(remove_if(points.begin(), points.end(),
    [](const Point2D& p) { return p.x < 2; }), points.end());

cout << " Осталось точек: " << points.size() << endl;
}

// =====
// 4. РАБОТА С MAP И SET С ПОЛЬЗОВАТЕЛЬСКИМИ СТРУКТУРАМИ
// =====

void demonstrateMapsAndSets() {
    cout << "\n=== ДЕМОНСТРАЦИЯ MAP И SET ===" << endl;
}

```

```

// Использование set с пользовательской структурой
cout << "\n1. Set of Points (автоматическая сортировка по оператору <):" << endl;
set<Point2D> uniquePoints;
uniquePoints.insert(Point2D(3, 4));
uniquePoints.insert(Point2D(1, 2));
uniquePoints.insert(Point2D(3, 4)); // Дубликат - не добавится
uniquePoints.insert(Point2D(5, 1));

for (const auto& point : uniquePoints) {
    cout << " " << point << endl;
}

// Использование map со структурой в качестве ключа
cout << "\n2. Map с Point2D как ключом:" << endl;
map<Point2D, string> pointLabels;
pointLabels[Point2D(0, 0)] = "Начало координат";
pointLabels[Point2D(1, 1)] = "Единичный квадрат";
pointLabels[Point2D(5, 5)] = "Диагональ";

for (const auto& [point, label] : pointLabels) {
    cout << " " << point << " -> \"" << label << "\"" << endl;
}

// Поиск в map
Point2D searchPoint(1, 1);
auto mapIt = pointLabels.find(searchPoint);
if (mapIt != pointLabels.end()) {
    cout << "\n Найдена метка для точки " << searchPoint
        << ": " << mapIt->second << endl;
}
}

// =====
// 5. ГЕНЕРАЦИЯ СЛУЧАЙНЫХ ДАННЫХ
// =====

vector<Student> generateRandomStudents(int count) {
    vector<Student> students;
    vector<string> firstNames = {"Иван", "Мария", "Петр", "Анна", "Сергей", "Елена"};
    vector<string> lastNames = {"Иванов", "Петрова", "Сидоров", "Кузнецова", "Смирнов",
        "Попова"};
    vector<string> subjects = {"Математика", "Физика", "Информатика", "История",
        "Английский"};

    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> idDist(1000, 9999);
    uniform_int_distribution<> gradeDist(2, 5);
    uniform_int_distribution<> examDist(0, 100);
    uniform_int_distribution<> nameDist(0, firstNames.size() - 1);

    for (int i = 0; i < count; i++) {

```

```

Student student(idDist(gen),
               firstNames[nameDist(gen)],
               lastNames[nameDist(gen)]);

// Добавляем случайные оценки
int numGrades = uniform_int_distribution<(3, 8)>(gen);
for (int j = 0; j < numGrades; j++) {
    student.addGrade(gradeDist(gen));
}

// Добавляем результаты экзаменов
for (const auto& subject : subjects) {
    if (uniform_int_distribution<(0, 1)>(gen)) {
        student.addExamResult(subject, examDist(gen));
    }
}

students.push_back(student);
}

return students;
}

// =====
// ОСНОВНАЯ ПРОГРАММА
// =====

int main() {
    cout << "=== ПРАКТИЧЕСКАЯ РАБОТА №8: СТАНДАРТНЫЕ ИНТЕРФЕЙСЫ И
СТРУКТУРЫ ===\n\n";

    // Часть 1: Работа со структурами
    cout << "1. РАБОТА СО СТРУКТУРАМИ Point2D И Student:" << endl;

    // Создание и использование структур
    Point2D p1(3.5, 4.2);
    Point2D p2(1.0, 2.0);
    Point2D p3 = p1 + p2;

    cout << "\nТочка p1: " << p1 << endl;
    cout << "Точка p2: " << p2 << endl;
    cout << "Сумма p1 + p2: " << p3 << endl;
    cout << "Расстояние между p1 и p2: " << p1.distanceTo(p2) << endl;

    // Создание студента
    Student student1(1001, "Иван", "Иванов");
    student1.addGrade(5);
    student1.addGrade(4);
    student1.addGrade(5);
    student1.addExamResult("Математика", 85);
    student1.addExamResult("Информатика", 92);

```

```

cout << "\n";
student1.display();

// Часть 2: База данных студентов
cout << "\n\n2. БАЗА ДАННЫХ СТУДЕНТОВ С ИСПОЛЬЗОВАНИЕМ STL:" << endl;

StudentDatabase db;

// Генерация случайных студентов
auto randomStudents = generateRandomStudents(10);
for (const auto& student : randomStudents) {
    db.addStudent(student);
}

// Добавляем нашего студента
db.addStudent(student1);

// Отображение всех студентов
cout << "\nВсе студенты (" << db.getAllStudents().size() << "):" << endl;
for (const auto& student : db.getAllStudents()) {
    student.display();
    cout << endl;
}

// Статистика
db.printStatistics();

// Сортировка
cout << "\nСортировка по фамилии:" << endl;
db.sortByLastName();
for (const auto& student : db.getAllStudents()) {
    cout << " " << student.lastName << " " << student.firstName << endl;
}

// Поиск отличников
cout << "\nОтличники:" << endl;
auto excellentStudents = db.getExcellentStudents();
for (const auto& student : excellentStudents) {
    cout << " " << student.lastName << " " << student.firstName
        << " (средний балл: " << student.averageGrade() << ")" << endl;
}

// Часть 3: Облако точек
cout << "\n\n3. РАБОТА С ОБЛАКОМ ТОЧЕК:" << endl;

PointCloud cloud;
cloud.addPoint(Point2D(1, 1));
cloud.addPoint(Point2D(2, 3));
cloud.addPoint(Point2D(4, 2));
cloud.addPoint(Point2D(5, 5));
cloud.addPoint(Point2D(3, 4));

```

```

cloud.display();

// Поиск ближайшей точки
Point2D target(3.5, 3.5);
Point2D nearest = cloud.findNearestPoint(target);
cout << "\nБлижайшая точка к " << target << ": " << nearest
    << " (расстояние: " << target.distanceTo(nearest) << ")" << endl;

// Поиск в радиусе
double radius = 2.0;
auto pointsInRadius = cloud.findPointsInRadius(target, radius);
cout << "\nТочки в радиусе " << radius << " от " << target << ":" << endl;
for (const auto& point : pointsInRadius) {
    cout << " " << point << endl;
}

// Центр масс
Point2D center = cloud.calculateCenterOfMass();
cout << "\nЦентр масс облака точек: " << center << endl;

// Часть 4: Демонстрация STL алгоритмов
demonstrateSTLAlgorithms();

// Часть 5: Map и Set
demonstrateMapsAndSets();

// Часть 6: Работа с Order (заказами)
cout << "\n\n6. СИСТЕМА УПРАВЛЕНИЯ ЗАКАЗАМИ:" << endl;

// Используем list для заказов (частое добавление/удаление)
list<Order> orders;

Order order1(1001, "Иванов И.И.");
order1.addItem("Ноутбук", 999.99);
order1.addItem("Мышь", 29.99);
order1.updateStatus("processing");

Order order2(1002, "Петрова А.С.");
order2.addItem("Смартфон", 499.99);
order2.addItem("Чехол", 19.99);
order2.addItem("Наушники", 79.99);

orders.push_back(order1);
orders.push_back(order2);

// Отображение заказов
cout << "\nВсе заказы:" << endl;
for (const auto& order : orders) {
    order.display();
    cout << endl;
}

```

```

// Поиск заказа по ID
int searchId = 1002;
auto orderIt = find_if(orders.begin(), orders.end(),
    [searchId](const Order& o) { return o.orderId == searchId; });

if (orderIt != orders.end()) {
    cout << "Найден заказ #" << searchId << endl;
    orderIt->display();
}

// Подсчет общей выручки
double totalRevenue = accumulate(orders.begin(), orders.end(), 0.0,
    [](double sum, const Order& o) { return sum + o.totalAmount; });

cout << "\nОбщая выручка: $" << fixed << setprecision(2) << totalRevenue << endl;

// Часть 7: Сохранение данных
cout << "\n\n7. СОХРАНЕНИЕ ДАННЫХ:" << endl;

if (db.saveToFile("students.txt")) {
    cout << "Данные студентов сохранены в файл students.txt" << endl;
} else {
    cout << "Ошибка сохранения данных" << endl;
}

cout << "\n=== ВЫВОДЫ И РЕЗУЛЬТАТЫ ===" << endl;
cout << "1. Изучены структуры (struct) и их отличие от классов" << endl;
cout << "2. Освоены стандартные контейнеры STL: vector, list, map, set" << endl;
cout << "3. Применены алгоритмы STL: sort, find_if, count_if, transform" << endl;
cout << "4. Реализованы пользовательские компараторы и лямбда-выражения" << endl;
cout << "5. Созданы системы управления данными с использованием STL" << endl;

return 0;
}

```

Задания для выполнения

Вариант 1 (Консольный - Система управления складом):

1. **Создайте структуры данных:**
 - Product: id, название, категория, цена, количество на складе
 - Supplier: id, название, контакты, рейтинг
 - Order: id, дата, поставщик, список товаров, статус
2. **Реализуйте классы управления с использованием STL:**
 - Warehouse: управление товарами (используйте map<int, Product>)
 - OrderManager: управление заказами (используйте vector<Order>)
 - SupplierCatalog: каталог поставщиков (используйте set<Supplier>)
3. **Примените алгоритмы STL для:**
 - Поиска товаров по критериям (цена, категория, наличие)
 - Сортировки поставщиков по рейтингу
 - Фильтрации заказов по статусу и дате
 - Подсчета статистики (общая стоимость товаров, средняя цена)
4. **Реализуйте операции:**
 - Добавление/удаление товаров

- Создание заказов у поставщиков
- Обновление количества товаров при поступлении/продаже
- Генерация отчетов в различных форматах

5. Дополнительные задачи:

- Сериализация данных в JSON/XML
- Поиск товаров с истекающим сроком годности
- Оптимизация заказов на основе анализа продаж

Вариант 2 (Qt - Система управления проектами):

1. Создайте структуры данных проекта:

- Employee: сотрудник (id, имя, должность, навыки)
- Task: задача (id, название, описание, статус, дедлайн, исполнитель)
- Project: проект (id, название, бюджет, задачи, участники)

2. Разработайте Qt-приложение с:

- Главным окном с вкладками для проектов, задач и сотрудников
- Таблицами для отображения данных (QTableView)
- Формами для добавления/редактирования элементов
- Диаграммами для визуализации прогресса проектов

3. Используйте STL контейнеры для хранения данных:

- map<int, Project> для проектов
- multimap<string, Task> для группировки задач по статусу
- set<Employee> для уникальных сотрудников

4. Реализуйте функционал:

- Назначение задач сотрудникам
- Отслеживание прогресса проектов
- Уведомления о приближающихся дедлайнах
- Расчет загруженности сотрудников

5. Дополнительные возможности:

- Импорт/экспорт данных в CSV
- Система учета рабочего времени
- Генерация Gantt-диаграмм
- Интеграция с календарем

Критерии оценивания

«Удовлетворительно» (3):

- Созданы базовые структуры данных
- Использованы минимум 3 типа STL контейнеров
- Реализованы базовые операции CRUD
- Программа компилируется и работает

«Хорошо» (4):

- Для **Варианта 1:** Полная система управления складом. Корректное использование STL алгоритмов. Реализована фильтрация и сортировка. Обработка исключений.
- Для **Варианта 2:** Рабочее Qt-приложение с графическим интерфейсом. Корректное отображение данных в таблицах. Реализованы основные функции управления.
- В отчете представлены схемы структур данных и описание алгоритмов.

«Отлично» (5):

- Для **Варианта 1:** Реализованы все дополнительные задачи. Добавлены продвинутые алгоритмы (поиск оптимальных заказов). Реализована многопоточная обработка. Написаны unit-тесты.
- Для **Варианта 2:** Реализованы все дополнительные возможности. Добавлены графики и диаграммы. Реализована система уведомлений. Поддержка нескольких языков интерфейса.
- **Общее:** Отчет содержит анализ эффективности различных STL контейнеров для конкретных задач, сравнение производительности, рекомендации по оптимизации.

Советы по выполнению:

1. Для эффективного использования STL:

```
cpp
// Используйте emplace для избежания лишних копирований
products.emplace(123, Product{123, "Товар", "Категория", 100.0, 50});

// Используйте structured binding (C++17)
for (const auto& [id, product] : products) {
    cout << id << ": " << product.name << endl;
}
```

2. Оптимизация поиска:

```
cpp
// Создание индексов для быстрого поиска
unordered_map<string, vector<Product*>> categoryIndex;
for (auto& [id, product] : products) {
    categoryIndex[product.category].push_back(&product);
}
```

3. В Qt для работы с STL:

```
cpp
// Конвертация между Qt и STL контейнерами
QVector<int> qtVector;
vector<int> stlVector = qtVector.toStdVector();

// Использование QMap с пользовательскими типами
struct Product {
    QString name;
    double price;
    bool operator<(const Product& other) const {
        return name < other.name;
    }
};
QMap<Product, int> inventory;
```

4. Реализация сериализации:

```
cpp
class Serializable {
public:
    virtual QJsonObject toJson() const = 0;
    virtual void fromJson(const QJsonObject& json) = 0;
    virtual ~Serializable() = default;
};

// Для структур
struct Product : Serializable {
    // ... поля
    QJsonObject toJson() const override {
        QJsonObject obj;
        obj["id"] = id;
        obj["name"] = QString::fromStdString(name);
        // ...
        return obj;
    }
}
```

```
};
```

5. Использование алгоритмов с пользовательскими типами:

```
cpp
```

```
// Сортировка с несколькими критериями
```

```
sort(products.begin(), products.end(),  
    [](const Product& a, const Product& b) {  
        if (a.category != b.category)  
            return a.category < b.category;  
        if (a.price != b.price)  
            return a.price < b.price;  
        return a.name < b.name;  
    });
```

6. Работа с датами и временем:

```
cpp
```

```
#include <chrono>
```

```
#include <ctime>
```

```
struct Task {  
    string name;  
    chrono::system_clock::time_point deadline;  
  
    int daysUntilDeadline() const {  
        auto now = chrono::system_clock::now();  
        auto diff = chrono::duration_cast<chrono::hours>(deadline - now);  
        return diff.count() / 24;  
    }  
};
```

Эта практическая работа позволит студентам освоить ключевые аспекты работы со структурами данных и стандартными интерфейсами STL, что является важным навыком для профессионального программиста на C++.

ПРАКТИЧЕСКАЯ РАБОТА №9. Коллекции. Параметризованные классы.

Цель работы: Освоить работу с коллекциями (стандартными и пользовательскими), изучить создание и использование параметризованных классов (шаблонов), научиться разрабатывать универсальные структуры данных и алгоритмы.

Задачи:

1. Изучить стандартные коллекции C++ и их особенности.
2. Освоить создание параметризованных классов (шаблонов).
3. Разработать пользовательские коллекции с использованием шаблонов.
4. Изучить итераторы и их применение в пользовательских коллекциях.
5. Реализовать алгоритмы для работы с параметризованными коллекциями.
6. (Для Qt) Создать графический интерфейс для визуализации коллекций.

Теоретический материал

1. Параметризованные классы (шаблоны)

Шаблоны (templates) — механизм C++ для создания обобщенных типов и функций.

cpp

// Шаблонный класс

```
template <typename T>
```

```
class Container {
```

```
private:
```

```
    T* data;
```

```
    size_t size;
```

```
    size_t capacity;
```

```
public:
```

```
    Container() : data(nullptr), size(0), capacity(0) {}
```

```
    void add(const T& value) {
```

```
        if (size >= capacity) {
```

```
            resize();
```

```
        }
```

```
        data[size++] = value;
```

```
    }
```

```
    T& get(size_t index) {
```

```
        if (index >= size) {
```

```
            throw out_of_range("Index out of range");
```

```
        }
```

```
        return data[index];
```

```
    }
```

```
    // ... другие методы
```

```
};
```

// Использование

```
Container<int> intContainer;
```

```
Container<string> stringContainer;
```

```
Container<double> doubleContainer;
```

2. Стандартные коллекции STL

cpp

```
#include <vector>
```

```
#include <list>
```

```

#include <deque>
#include <set>
#include <map>
#include <unordered_set>
#include <unordered_map>
#include <stack>
#include <queue>

// Вектор - динамический массив
vector<int> vec = {1, 2, 3, 4, 5};
vec.push_back(6);

// Список - двусвязный список
list<string> names = {"Alice", "Bob", "Charlie"};

// Множество - уникальные элементы, отсортированные
set<int> uniqueNumbers = {5, 2, 8, 2, 5}; // {2, 5, 8}

```

```

// Ассоциативный массив
map<string, int> ageMap = {"Alice", 25}, {"Bob", 30};;

```

3. Пользовательские итераторы

```

cpp
template <typename T>
class MyIterator {
private:
    T* ptr;

public:
    MyIterator(T* p) : ptr(p) {}

    T& operator*() { return *ptr; }
    T* operator->() { return ptr; }

    MyIterator& operator++() {
        ++ptr;
        return *this;
    }

    bool operator!=(const MyIterator& other) {
        return ptr != other.ptr;
    }
};

```

```

// Использование в коллекции
template <typename T>
class MyCollection {
public:
    MyIterator<T> begin() { return MyIterator<T>(data); }
    MyIterator<T> end() { return MyIterator<T>(data + size); }
};

```

Пример решения (Консольное приложение на C++)

cpp

```
#include <iostream>
#include <vector>
#include <list>
#include <set>
#include <map>
#include <algorithm>
#include <memory>
#include <stdexcept>
#include <type_traits>
#include <string>
#include <sstream>
#include <iomanip>
#include <functional>
#include <chrono>
```

```
using namespace std;
using namespace chrono;
```

```
// =====
// 1. БАЗОВЫЕ ПАРАМЕТРИЗОВАННЫЕ КЛАССЫ
// =====
```

// Шаблонный класс "Динамический массив"

```
template <typename T>
class DynamicArray {
private:
```

```
    T* data;
    size_t size;
    size_t capacity;

    void resize(size_t newCapacity) {
        T* newData = new T[newCapacity];
        for (size_t i = 0; i < size; i++) {
            newData[i] = move(data[i]);
        }
        delete[] data;
        data = newData;
        capacity = newCapacity;
    }
}
```

```
public:
```

// Конструкторы

```
DynamicArray() : data(nullptr), size(0), capacity(0) {
    resize(10);
}
```

```
DynamicArray(size_t initialCapacity) : data(nullptr), size(0), capacity(0) {
    resize(initialCapacity > 0 ? initialCapacity : 10);
}
```

```

DynamicArray(initializer_list<T> init) : data(nullptr), size(0), capacity(0) {
    resize(init.size() * 2);
    for (const auto& item : init) {
        add(item);
    }
}

// Конструктор копирования
DynamicArray(const DynamicArray& other) : data(nullptr), size(0), capacity(0) {
    resize(other.capacity);
    size = other.size;
    for (size_t i = 0; i < size; i++) {
        data[i] = other.data[i];
    }
}

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

// Конструктор перемещения
DynamicArray(DynamicArray&& other) noexcept
    : data(other.data), size(other.size), capacity(other.capacity) {
    other.data = nullptr;
    other.size = 0;
    other.capacity = 0;
}

// Оператор перемещения
DynamicArray& operator=(DynamicArray&& other) noexcept {
    if (this != &other) {
        delete[] data;
        data = other.data;
        size = other.size;
        capacity = other.capacity;
        other.data = nullptr;
        other.size = 0;
        other.capacity = 0;
    }
    return *this;
}

```

```

// Деструктор
~DynamicArray() {
    delete[] data;
}

// Основные методы
void add(const T& value) {
    if (size >= capacity) {
        resize(capacity * 2);
    }
    data[size++] = value;
}

void add(T&& value) {
    if (size >= capacity) {
        resize(capacity * 2);
    }
    data[size++] = move(value);
}

void insert(size_t index, const T& value) {
    if (index > size) {
        throw out_of_range("Index out of range");
    }

    if (size >= capacity) {
        resize(capacity * 2);
    }

    // Сдвиг элементов вправо
    for (size_t i = size; i > index; i--) {
        data[i] = move(data[i - 1]);
    }

    data[index] = value;
    size++;
}

void remove(size_t index) {
    if (index >= size) {
        throw out_of_range("Index out of range");
    }

    // Сдвиг элементов влево
    for (size_t i = index; i < size - 1; i++) {
        data[i] = move(data[i + 1]);
    }
    size--;
}

T& get(size_t index) {
    if (index >= size) {

```

```

        throw out_of_range("Index out of range");
    }
    return data[index];
}

```

```

const T& get(size_t index) const {
    if (index >= size) {
        throw out_of_range("Index out of range");
    }
    return data[index];
}

```

```

T& operator[](size_t index) {
    return get(index);
}

```

```

const T& operator[](size_t index) const {
    return get(index);
}

```

```

size_t getSize() const {
    return size;
}

```

```

size_t getCapacity() const {
    return capacity;
}

```

```

bool isEmpty() const {
    return size == 0;
}

```

```

void clear() {
    size = 0;
}

```

// Поиск элемента

```

int find(const T& value) const {
    for (size_t i = 0; i < size; i++) {
        if (data[i] == value) {
            return i;
        }
    }
    return -1;
}

```

// Применение функции к каждому элементу

```

template <typename Func>
void forEach(Func func) const {
    for (size_t i = 0; i < size; i++) {
        func(data[i]);
    }
}

```

```

}

// Фильтрация
template <typename Predicate>
DynamicArray<T> filter(Predicate pred) const {
    DynamicArray<T> result;
    for (size_t i = 0; i < size; i++) {
        if (pred(data[i])) {
            result.add(data[i]);
        }
    }
    return result;
}

// Преобразование
template <typename U, typename Transformer>
DynamicArray<U> map(Transformer trans) const {
    DynamicArray<U> result;
    for (size_t i = 0; i < size; i++) {
        result.add(trans(data[i]));
    }
    return result;
}
};

// Специализация для bool (оптимизация хранения)
template <>
class DynamicArray<bool> {
private:
    unsigned char* data;
    size_t size;
    size_t capacity;

    void resize(size_t newCapacity) {
        size_t bytes = (newCapacity + 7) / 8;
        unsigned char* newData = new unsigned char[bytes]{0};

        // Копирование битов
        for (size_t i = 0; i < size; i++) {
            if (getBit(i)) {
                size_t byteIndex = i / 8;
                size_t bitIndex = i % 8;
                newData[byteIndex] |= (1 << bitIndex);
            }
        }

        delete[] data;
        data = newData;
        capacity = newCapacity;
    }

    bool getBit(size_t index) const {

```

```

        size_t byteIndex = index / 8;
        size_t bitIndex = index % 8;
        return (data[byteIndex] >> bitIndex) & 1;
    }

    void setBit(size_t index, bool value) {
        size_t byteIndex = index / 8;
        size_t bitIndex = index % 8;
        if (value) {
            data[byteIndex] |= (1 << bitIndex);
        } else {
            data[byteIndex] &= ~(1 << bitIndex);
        }
    }

public:
    DynamicArray() : data(nullptr), size(0), capacity(0) {
        resize(80); // 80 байт = 10 байт
    }

    void add(bool value) {
        if (size >= capacity) {
            resize(capacity * 2);
        }
        setBit(size++, value);
    }

    bool get(size_t index) const {
        if (index >= size) throw out_of_range("Index out of range");
        return getBit(index);
    }

    size_t getSize() const { return size; }

    // ... остальные методы
};

// =====
// 2. КОЛЛЕКЦИЯ С ИТЕРАТОРАМИ
// =====

template <typename T>
class LinkedList {
private:
    // Узел списка
    struct Node {
        T data;
        unique_ptr<Node> next;

        Node(const T& value) : data(value), next(nullptr) {}
        Node(T&& value) : data(move(value)), next(nullptr) {}
    };
};

```

```
unique_ptr<Node> head;
Node* tail;
size_t listSize;
```

```
public:
```

```
// Итератор для LinkedList
```

```
class Iterator {
```

```
private:
```

```
    Node* current;
```

```
public:
```

```
    Iterator(Node* node) : current(node) {}
```

```
    T& operator*() {
        return current->data;
    }
```

```
    T* operator->() {
        return &current->data;
    }
```

```
    Iterator& operator++() {
        if (current) {
            current = current->next.get();
        }
        return *this;
    }
```

```
    Iterator operator++(int) {
        Iterator temp = *this;
        ++(*this);
        return temp;
    }
```

```
    bool operator==(const Iterator& other) const {
        return current == other.current;
    }
```

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

```
// Конструкторы
```

```
LinkedList() : head(nullptr), tail(nullptr), listSize(0) {}
```

```
LinkedList(initializer_list<T> init) : head(nullptr), tail(nullptr), listSize(0) {
    for (const auto& item : init) {
        add(item);
    }
}
```

// Методы

```
void add(const T& value) {
    auto newNode = make_unique<Node>(value);

    if (!head) {
        head = move(newNode);
        tail = head.get();
    } else {
        tail->next = move(newNode);
        tail = tail->next.get();
    }
    listSize++;
}

void add(T&& value) {
    auto newNode = make_unique<Node>(move(value));

    if (!head) {
        head = move(newNode);
        tail = head.get();
    } else {
        tail->next = move(newNode);
        tail = tail->next.get();
    }
    listSize++;
}

void insert(size_t index, const T& value) {
    if (index > listSize) {
        throw out_of_range("Index out of range");
    }

    if (index == 0) {
        auto newNode = make_unique<Node>(value);
        newNode->next = move(head);
        head = move(newNode);
        if (!tail) tail = head.get();
    } else if (index == listSize) {
        add(value);
        return;
    } else {
        Node* current = head.get();
        for (size_t i = 0; i < index - 1; i++) {
            current = current->next.get();
        }

        auto newNode = make_unique<Node>(value);
        newNode->next = move(current->next);
        current->next = move(newNode);
    }
    listSize++;
}
```

```

}

void remove(size_t index) {
    if (index >= listSize) {
        throw out_of_range("Index out of range");
    }

    if (index == 0) {
        head = move(head->next);
        if (!head) tail = nullptr;
    } else {
        Node* current = head.get();
        for (size_t i = 0; i < index - 1; i++) {
            current = current->next.get();
        }

        current->next = move(current->next->next);
        if (index == listSize - 1) {
            tail = current;
        }
    }
    listSize--;
}

```

```

T& get(size_t index) {
    if (index >= listSize) {
        throw out_of_range("Index out of range");
    }

    Node* current = head.get();
    for (size_t i = 0; i < index; i++) {
        current = current->next.get();
    }
    return current->data;
}

```

```

size_t getSize() const {
    return listSize;
}

```

```

bool isEmpty() const {
    return listSize == 0;
}

```

// Итераторы

```

Iterator begin() {
    return Iterator(head.get());
}

```

```

Iterator end() {
    return Iterator(nullptr);
}

```

```

// Поиск
Iterator find(const T& value) {
    for (auto it = begin(); it != end(); ++it) {
        if (*it == value) {
            return it;
        }
    }
    return end();
}

// Применение функции к каждому элементу
template <typename Func>
void forEach(Func func) {
    for (auto& item : *this) {
        func(item);
    }
}
};

// =====
// 3. ПАРАМЕТРИЗОВАННЫЙ КЛАСС ДЛЯ СТЕКА
// =====

template <typename T, typename Container = DynamicArray<T>>
class Stack {
private:
    Container container;

public:
    void push(const T& value) {
        container.add(value);
    }

    void push(T&& value) {
        container.add(move(value));
    }

    T pop() {
        if (isEmpty()) {
            throw runtime_error("Stack is empty");
        }

        T value = move(container[container.getSize() - 1]);
        container.remove(container.getSize() - 1);
        return value;
    }

    T& top() {
        if (isEmpty()) {
            throw runtime_error("Stack is empty");
        }
    }
}

```

```

        return container[container.getSize() - 1];
    }

    const T& top() const {
        if (isEmpty()) {
            throw runtime_error("Stack is empty");
        }
        return container[container.getSize() - 1];
    }

    bool isEmpty() const {
        return container.isEmpty();
    }

    size_t size() const {
        return container.getSize();
    }

    void clear() {
        container.clear();
    }
};

// =====
// 4. ПАРАМЕТРИЗОВАННЫЙ КЛАСС ДЛЯ ОЧЕРЕДИ
// =====

template <typename T, typename Container = LinkedList<T>>
class Queue {
private:
    Container container;

public:
    void enqueue(const T& value) {
        container.add(value);
    }

    void enqueue(T&& value) {
        container.add(move(value));
    }

    T dequeue() {
        if (isEmpty()) {
            throw runtime_error("Queue is empty");
        }

        T value = move(container.get(0));
        container.remove(0);
        return value;
    }

    T& front() {

```

```

        if (isEmpty()) {
            throw runtime_error("Queue is empty");
        }
        return container.get(0);
    }

    const T& front() const {
        if (isEmpty()) {
            throw runtime_error("Queue is empty");
        }
        return container.get(0);
    }

    T& back() {
        if (isEmpty()) {
            throw runtime_error("Queue is empty");
        }
        return container.get(container.getSize() - 1);
    }

    const T& back() const {
        if (isEmpty()) {
            throw runtime_error("Queue is empty");
        }
        return container.get(container.getSize() - 1);
    }

    bool isEmpty() const {
        return container.isEmpty();
    }

    size_t size() const {
        return container.getSize();
    }

    void clear() {
        // Для LinkedList нужно реализовать clear
        while (!isEmpty()) {
            dequeue();
        }
    }
};

// =====
// 5. ПАРАМЕТРИЗОВАННАЯ ХЕШ-ТАБЛИЦА
// =====

template <typename K, typename V>
class HashTable {
private:
    struct Entry {
        K key;

```

```

    V value;
    bool occupied;

    Entry() : occupied(false) {}
    Entry(const K& k, const V& v) : key(k), value(v), occupied(true) {}
};

vector<Entry> table;
size_t itemCount;
double loadFactorThreshold;

size_t hash(const K& key) const {
    // Простая хеш-функция
    return hash<K>{}(key) % table.size();
}

void rehash() {
    vector<Entry> oldTable = move(table);
    table.resize(oldTable.size() * 2);
    itemCount = 0;

    for (const auto& entry : oldTable) {
        if (entry.occupied) {
            insert(entry.key, entry.value);
        }
    }
}

public:
    HashTable(size_t initialSize = 101)
        : table(initialSize), itemCount(0), loadFactorThreshold(0.75) {}

    void insert(const K& key, const V& value) {
        if (static_cast<double>(itemCount) / table.size() > loadFactorThreshold) {
            rehash();
        }

        size_t index = hash(key);
        size_t startIndex = index;

        // Линейное пробирование
        while (table[index].occupied) {
            if (table[index].key == key) {
                table[index].value = value;
                return;
            }
            index = (index + 1) % table.size();
            if (index == startIndex) {
                rehash();
                index = hash(key);
                startIndex = index;
            }
        }
    }

```

```

    }

    table[index] = Entry(key, value);
    itemCount++;
}

bool contains(const K& key) const {
    size_t index = hash(key);
    size_t startIndex = index;

    while (table[index].occupied) {
        if (table[index].key == key) {
            return true;
        }
        index = (index + 1) % table.size();
        if (index == startIndex) {
            break;
        }
    }

    return false;
}

V& get(const K& key) {
    size_t index = hash(key);
    size_t startIndex = index;

    while (table[index].occupied) {
        if (table[index].key == key) {
            return table[index].value;
        }
        index = (index + 1) % table.size();
        if (index == startIndex) {
            break;
        }
    }

    throw out_of_range("Key not found");
}

void remove(const K& key) {
    size_t index = hash(key);
    size_t startIndex = index;

    while (table[index].occupied) {
        if (table[index].key == key) {
            table[index].occupied = false;
            itemCount--;

            // Перемещение последующих элементов
            size_t next = (index + 1) % table.size();
            while (table[next].occupied) {

```

```

        size_t correctIndex = hash(table[next].key);
        if (correctIndex != next) {
            table[index] = move(table[next]);
            table[next].occupied = false;
            index = next;
        }
        next = (next + 1) % table.size();
    }
    return;
}
index = (index + 1) % table.size();
if (index == startIndex) {
    break;
}
}
}

size_t size() const {
    return itemCount;
}

bool isEmpty() const {
    return itemCount == 0;
}

void clear() {
    for (auto& entry : table) {
        entry.occupied = false;
    }
    itemCount = 0;
}
};

// =====
// 6. ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ И ТЕСТЫ
// =====

template <typename Container>
void testContainerPerformance(const string& containerName) {
    Container container;
    auto start = high_resolution_clock::now();

    // Тест добавления
    for (int i = 0; i < 100000; i++) {
        container.add(i);
    }

    auto end = high_resolution_clock::now();
    auto duration = duration_cast<milliseconds>(end - start);

    cout << containerName << " - добавление 100000 элементов: "
         << duration.count() << " мс" << endl;
}

```

```

}

// Функция для печати коллекции
template <typename T>
void printCollection(const T& collection, const string& name = "Коллекция") {
    cout << name << " (" << collection.getSize() << " элементов): ";
    collection.forEach([](const auto& item) {
        cout << item << " ";
    });
    cout << endl;
}

// =====
// ОСНОВНАЯ ПРОГРАММА
// =====

int main() {
    cout << "=== ПРАКТИЧЕСКАЯ РАБОТА №9: КОЛЛЕКЦИИ И ПАРАМЕТРИЗОВАННЫЕ КЛАССЫ ===\n\n";

    // Часть 1: DynamicArray
    cout << "1. ТЕСТИРОВАНИЕ DYNAMICARRAY:" << endl;

    DynamicArray<int> intArray;
    intArray.add(10);
    intArray.add(20);
    intArray.add(30);
    intArray.insert(1, 15);

    cout << "intArray: ";
    for (size_t i = 0; i < intArray.getSize(); i++) {
        cout << intArray[i] << " ";
    }
    cout << endl;

    // Фильтрация
    auto evenNumbers = intArray.filter([](int n) { return n % 2 == 0; });
    cout << "Четные числа: ";
    evenNumbers.forEach([](int n) { cout << n << " "; });
    cout << endl;

    // Преобразование
    auto stringArray = intArray.map<string>([](int n) {
        return "Num: " + to_string(n);
    });
    cout << "Преобразовано в строки: ";
    stringArray.forEach([](const string& s) { cout << s << " | "; });
    cout << endl;

    // Часть 2: LinkedList с итераторами
    cout << "\n\n2. ТЕСТИРОВАНИЕ LINKEDLIST С ИТЕРАТОРАМИ:" << endl;
}

```

```

LinkedList<string> names;
names.add("Alice");
names.add("Bob");
names.add("Charlie");
names.insert(1, "David");

cout << "Имена: ";
for (const auto& name : names) {
    cout << name << " ";
}
cout << endl;

// Поиск с итератором
auto it = names.find("Bob");
if (it != names.end()) {
    cout << "Найден: " << *it << endl;
}

// Часть 3: Stack
cout << "\n\n3. ТЕСТИРОВАНИЕ STACK:" << endl;

Stack<int> intStack;
intStack.push(1);
intStack.push(2);
intStack.push(3);

cout << "Верхний элемент стека: " << intStack.top() << endl;
cout << "Извлечение: " << intStack.pop() << endl;
cout << "Новый верхний элемент: " << intStack.top() << endl;
cout << "Размер стека: " << intStack.size() << endl;

// Stack с другим контейнером
Stack<double, LinkedList<double>> doubleStack;
doubleStack.push(3.14);
doubleStack.push(2.71);
cout << "Стек double: " << doubleStack.top() << endl;

// Часть 4: Queue
cout << "\n\n4. ТЕСТИРОВАНИЕ QUEUE:" << endl;

Queue<string> stringQueue;
stringQueue.enqueue("Первый");
stringQueue.enqueue("Второй");
stringQueue.enqueue("Третий");

cout << "Первый в очереди: " << stringQueue.front() << endl;
cout << "Последний в очереди: " << stringQueue.back() << endl;
cout << "Извлечение: " << stringQueue.dequeue() << endl;
cout << "Новый первый: " << stringQueue.front() << endl;

// Часть 5: HashTable
cout << "\n\n5. ТЕСТИРОВАНИЕ HASHTABLE:" << endl;

```

```

HashTable<string, int> ageTable;
ageTable.insert("Alice", 25);
ageTable.insert("Bob", 30);
ageTable.insert("Charlie", 35);

cout << "Возраст Alice: " << ageTable.get("Alice") << endl;
cout << "Есть ли Bob? " << (ageTable.contains("Bob") ? "Да" : "Нет") << endl;

ageTable.remove("Bob");
cout << "Есть ли Bob после удаления? " << (ageTable.contains("Bob") ? "Да" : "Нет") <<
endl;

// Часть 6: Специализация для bool
cout << "\n\n6. СПЕЦИАЛИЗАЦИЯ DYNAMICARRAY ДЛЯ BOOL:" << endl;

DynamicArray<bool> boolArray;
for (int i = 0; i < 20; i++) {
    boolArray.add(i % 3 == 0);
}

cout << "Bool массив (true если делится на 3): ";
for (size_t i = 0; i < boolArray.getSize(); i++) {
    cout << (boolArray.get(i) ? "1" : "0") << " ";
}
cout << endl;

// Часть 7: Тестирование производительности
cout << "\n\n7. ТЕСТИРОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ:" << endl;

testContainerPerformance<DynamicArray<int>>("DynamicArray");

// Сравнение со стандартным vector
vector<int> stdVector;
auto start = high_resolution_clock::now();
for (int i = 0; i < 100000; i++) {
    stdVector.push_back(i);
}
auto end = high_resolution_clock::now();
auto duration = duration_cast<milliseconds>(end - start);
cout << "std::vector - добавление 100000 элементов: "
    << duration.count() << " мс" << endl;

// Часть 8: Работа со сложными типами
cout << "\n\n8. РАБОТА СО СЛОЖНЫМИ ТИПАМИ:" << endl;

struct Person {
    string name;
    int age;

    bool operator==(const Person& other) const {
        return name == other.name && age == other.age;
    }
};

```

```

    }

    friend ostream& operator<<(ostream& os, const Person& p) {
        os << p.name << " (" << p.age << " лет)";
        return os;
    }
};

DynamicArray<Person> people;
people.add({"Alice", 25});
people.add({"Bob", 30});
people.add({"Charlie", 35});

cout << "Люди: ";
people.forEach([](const Person& p) {
    cout << p << " | ";
});
cout << endl;

// Фильтрация людей старше 28
auto olderPeople = people.filter([](const Person& p) {
    return p.age > 28;
});

cout << "Люди старше 28: ";
olderPeople.forEach([](const Person& p) {
    cout << p << " | ";
});
cout << endl;

// Часть 9: Применение в реальной задаче
cout << "\n\n9. ПРИМЕНЕНИЕ В ЗАДАЧЕ УПРАВЛЕНИЯ ЗАДАЧАМИ:" << endl;

struct Task {
    int id;
    string description;
    int priority; // 1-высокий, 2-средний, 3-низкий

    bool operator<(const Task& other) const {
        return priority < other.priority;
    }

    friend ostream& operator<<(ostream& os, const Task& t) {
        os << "Задача #" << t.id << ": " << t.description
            << " [Приоритет: " << t.priority << "]";
        return os;
    }
};

// Очередь с приоритетом на основе стека
Stack<Task> taskStack;
taskStack.push({1, "Написать код", 2});

```

```

taskStack.push({2, "Протестировать программу", 1});
taskStack.push({3, "Написать документацию", 3});

// Сортировка задач по приоритету
DynamicArray<Task> taskArray;
while (!taskStack.isEmpty()) {
    taskArray.add(taskStack.pop());
}

// Сортировка (пузырьковая для демонстрации)
for (size_t i = 0; i < taskArray.getSize(); i++) {
    for (size_t j = i + 1; j < taskArray.getSize(); j++) {
        if (taskArray[j] < taskArray[i]) {
            swap(taskArray[i], taskArray[j]);
        }
    }
}

cout << "Задачи отсортированные по приоритету:" << endl;
taskArray.forEach([](const Task& t) {
    cout << " " << t << endl;
});

cout << "\n=== ИТОГИ ===" << endl;
cout << "1. Созданы параметризованные классы: DynamicArray, LinkedList, Stack, Queue,
HashTable" << endl;
cout << "2. Реализованы итераторы для пользовательских коллекций" << endl;
cout << "3. Создана специализация шаблона для типа bool" << endl;
cout << "4. Протестирована производительность и функциональность" << endl;
cout << "5. Продемонстрировано применение в реальных задачах" << endl;

return 0;
}

```

Задания для выполнения

Вариант 1 (Консональный - Библиотека коллекций):

1. **Реализуйте параметризованные коллекции:**
 - BinaryTree<T>: бинарное дерево поиска с балансировкой
 - CircularBuffer<T>: кольцевой буфер фиксированного размера
 - PriorityQueue<T, Compare>: очередь с приоритетом
 - Graph<V, E>: граф с вершинами типа V и ребрами типа E
2. **Для каждой коллекции реализуйте:**
 - Итераторы (прямой, обратный, константный)
 - Стандартные методы (добавление, удаление, поиск, очистка)
 - Алгоритмы (сортировка, обход, поиск пути)
 - Сериализацию/десериализацию
3. **Создайте тестовую систему:**
 - Unit-тесты для каждой коллекции
 - Тесты производительности (сравнение со STL)
 - Тесты на корректность (граничные случаи, исключения)
4. **Дополнительные задачи:**
 - Реализуйте паттерн "Итератор" для обхода коллекций

- Добавьте поддержку move-семантики
- Реализуйте потокобезопасные версии коллекций
- Создайте адаптеры для совместимости со STL

Вариант 2 (Qt - Визуализатор структур данных):

1. **Создайте Qt-приложение для визуализации:**
 - Динамического массива
 - Связного списка
 - Бинарного дерева
 - Графа
 - Хеш-таблицы
2. **Для каждой структуры реализуйте:**
 - Визуальное представление
 - Анимацию операций (добавление, удаление, поиск)
 - Подсветку текущих операций
 - Отображение статистики (размер, время операций)
3. **Интерфейс приложения должен содержать:**
 - Панель выбора структуры данных
 - Панель операций с элементами
 - Область визуализации
 - Консоль для вывода информации
 - Панель статистики
4. **Дополнительные возможности:**
 - Сохранение/загрузка состояния
 - Ген

ПРАКТИЧЕСКАЯ РАБОТА №10. Использование регулярных выражений. Операции со списками.

Цель работы:

Освоить синтаксис регулярных выражений, научиться применять их для поиска, валидации и обработки текстовой информации. Получить практические навыки работы с классами `QRegularExpression` и `QRegularExpressionMatch` в среде Qt/C++.

Задачи:

1. Изучить базовые элементы синтаксиса регулярных выражений.
2. Научиться составлять шаблоны для типовых задач (email, телефон, дата).
3. Реализовать программу для поиска и извлечения данных из текста.
4. Реализовать программу для валидации вводимых пользователем данных.
5. Изучить различия между `QList`, `QVector`, `std::vector`, `std::list`.
6. Научиться выполнять базовые операции с контейнерами.
7. Реализовать алгоритмы обработки списков.
8. Оценить производительность различных контейнеров.

Теоретический материал

Регулярные выражения (RegEx) — формальный язык для описания шаблонов в строках. В Qt для работы с ними используется класс `QRegularExpression`.

Основные конструкции:

- `.` — любой символ (кроме переноса строки)
- `\d` — цифра (эквивалент `[0-9]`)
- `\w` — буквенно-цифровой символ или `_`
- `\s` — пробельный символ (пробел, табуляция)
- `[]` — класс символов: `[A-Za-z]` — все латинские буквы
- `*` — 0 или более повторений
- `+` — 1 или более повторений
- `?` — 0 или 1 повторение
- `{n,m}` — от `n` до `m` повторений
- `^` — начало строки
- `$` — конец строки
- `()` — группа захвата

Примеры шаблонов:

- Телефон РФ: `^\+7(\d{3})\d{3}-\d{2}-\d{2}$`
- Email: `^[\\w.%+-]+@[\\w.-]+\\. [A-Za-z]{2,}$`
- Дата ДД.ММ.ГГГГ: `^\d{2}\\. \d{2}\\. \d{4}$`

Ключевые методы `QRegularExpression`:

```
cpp
QRegularExpression re("шаблон");
bool isValid = re.isValid(); // проверка корректности шаблона
QRegularExpressionMatch match = re.match(строка);
if (match.hasMatch()) {
    QString found = match.captured(0); // извлечение найденного
}
// Глобальный поиск всех совпадений:
QRegularExpressionMatchIterator it = re.globalMatch(текст);
while (it.hasNext()) {
    QRegularExpressionMatch m = it.next();
    qDebug() << m.captured(0);
}
}
```

Пример решения

Задача: Написать программу, которая извлекает все суммы денег в формате "100 руб.", "50.30 USD", "€25" из текста.

```

cpp
#include <QCoreApplication>
#include <QRegularExpression>
#include <QDebug>
#include <QString>

int main() {
    QString text = "Цены: 100 руб., скидка 50.30 USD, премия €25, итого 1000 RUB";

    // Шаблон: число (с возможной десятичной частью) + пробел + валюта
    QRegularExpression re(R"(b(d+(?:\.\d+)?)\s*(руб\.|RUB|USD|€))");

    QRegularExpressionMatchIterator it = re.globalMatch(text);
    while (it.hasNext()) {
        QRegularExpressionMatch match = it.next();
        QString amount = match.captured(1); // сумма
        QString currency = match.captured(2); // валюта
        qDebug() << "Найдено:" << amount << currency;
    }

    return 0;
}

```

Задания

Вариант 1

- Валидация данных:** Создайте программу, которая проверяет корректность введенных данных:
 - Номер телефона в формате +7(XXX)XXX-XX-XX
 - Email адрес
 - Дату в формате ДД.ММ.ГГГГ (учитывать, что месяц ≤ 12 , день ≤ 31)
- Поиск информации:** Из текстового файла input.txt извлеките:
 - Все хештеги (слова, начинающиеся с #)
 - Все упоминания пользователей (начинающиеся с @)
 - Сохраните результаты в файл output.txt

Вариант 2

- Обработка логов:** Создайте программу для анализа логов веб-сервера:
 - Извлеките все IP-адреса (формат IPv4)
 - Найдите все HTTP-запросы GET, POST, PUT
 - Определите самые частые коды ответов (404, 200, 500)
- Форматирование текста:** Напишите программу, которая:
 - Находит в тексте все номера в формате 8-XXX-XXX-XX-XX и заменяет их на +7(XXX)XXX-XX-XX
 - Удаляет повторяющиеся пробелы
 - Форматирует даты из ДД/ММ/ГГГГ в ГГГГ-ММ-ДД

Критерии оценивания

- Удовлетворительно (3):** Реализована одна задача из варианта, программа компилируется, есть базовое понимание синтаксиса RegEx.
- Хорошо (4):** Реализованы все задачи варианта, код хорошо структурирован, используются группы захвата, обработка ошибок.
- Отлично (5):** Полная реализация всех задач с дополнительной функциональностью (чтение/запись файлов, статистика, GUI-интерфейс), код содержит комментарии, использованы продвинутые возможности RegEx (жадные/ленивые квантификаторы, lookahead).

Советы для студентов

1. **Начинайте с простых шаблонов:** Сначала проверьте, что ваш шаблон работает на 1-2 примерах, потом усложняйте.
2. **Используйте онлайн-тестеры:** Перед написанием кода проверяйте шаблоны на сайтах типа regex101.com.
3. **Экранируйте спецсимволы:** В C++ строки нужно экранировать дважды: `\\d` вместо `\d`.
4. **Проверяйте валидность шаблона:** Всегда вызывайте `re.isValid()` перед использованием.
5. **Тестируйте на граничных случаях:** Пустые строки, некорректные данные, крайние значения.

Теоретический материал

Контейнеры в C++/Qt:

Контейнер	Особенности	Когда использовать
<code>QList</code>	Быстрые добавление/удаление в начала/конца	Универсальный список, частая вставка
<code>QVector</code>	Быстрый доступ по индексу, непрерывная память	Частый доступ по индексу, редко изменяется
<code>std::vector</code>	Аналог <code>QVector</code> , стандарт C++	Кроссплатформенные проекты
<code>std::list</code>	Двусвязный список, быстрая вставка/удаление	Частые вставки/удаления в середине

Основные операции:

cpp

// Инициализация

```
QList<int> list = {1, 2, 3, 4, 5};
```

```
QVector<QString> vector;
```

```
vector << "A" << "B" << "C";
```

// Добавление

```
list.append(6);           // в конец
```

```
list.prepend(0);          // в начало
```

```
list.insert(2, 99);        // по индексу
```

// Удаление

```
list.removeAt(2);          // по индексу
```

```
list.removeOne(99);        // по значению
```

```
list.removeAll(5);         // все вхождения
```

```
list.clear();              // очистка
```

// Поиск

```
int index = list.indexOf(3); // индекс первого вхождения
```

```
bool exists = list.contains(4); // проверка наличия
```

// Итерация

```
for (int i = 0; i < list.size(); ++i) {
```

```
    qDebug() << list[i];
```

```
}
```

```
foreach (int value, list) {
    qDebug() << value;
}
```

```
for (auto it = list.begin(); it != list.end(); ++it) {
    qDebug() << *it;
}
```

// Сортировка

```
std::sort(list.begin(), list.end());    // по возрастанию
std::sort(list.rbegin(), list.rend());  // по убыванию
```

Алгоритмы STL:

- std::find() — поиск элемента
- std::sort() — сортировка
- std::remove_if() — условное удаление
- std::transform() — преобразование элементов
- std::accumulate() — сумма/агрегация

Пример решения

Задача: Реализовать менеджер задач с приоритетами.

cpp

```
#include <QCoreApplication>
#include <QList>
#include <QDebug>
#include <algorithm>
```

```
struct Task {
    QString description;
    int priority; // 1-высокий, 2-средний, 3-низкий
    bool completed;
};
```

```
bool compareByPriority(const Task &a, const Task &b) {
    return a.priority < b.priority;
}
```

```
int main() {
    QList<Task> tasks;
```

// Добавление задач

```
tasks.append({"Сделать практическую", 1, false});
tasks.append({"Купить продукты", 3, false});
tasks.append({"Подготовить презентацию", 2, false});
```

// Сортировка по приоритету

```
std::sort(tasks.begin(), tasks.end(), compareByPriority);
```

// Вывод отсортированных задач

```
qDebug() << "Задачи по приоритету:";
for (const Task &task : tasks) {
    qDebug() << "[" << task.priority << "]" << task.description;
}
```

```

// Поиск невыполненных задач высокого приоритета
auto it = std::find_if(tasks.begin(), tasks.end(),
    [](const Task &t) { return t.priority == 1 && !t.completed; });

if (it != tasks.end()) {
    qDebug() << "Срочная задача:" << it->description;
    it->completed = true; // отмечаем как выполненную
}

// Удаление всех выполненных задач
tasks.erase(std::remove_if(tasks.begin(), tasks.end(),
    [](const Task &t) { return t.completed; }),
    tasks.end());

qDebug() << "Осталось задач:" << tasks.size();

return 0;
}

```

Задания

Вариант 1

- Система бронирования:** Создайте систему бронирования билетов:
 - Класс `Ticket` с полями: номер, ФИО, рейс, дата, статус (бронирован/куплен)
 - Реализуйте функции:
 - Добавление нового билета
 - Поиск билетов по ФИО
 - Отмена бронирования (удаление)
 - Вывод всех билетов на конкретный рейс
 - Храните данные в `QList<Ticket>`
- Анализ данных:** Реализуйте программу для анализа оценок студентов:
 - Считайте оценки из файла (ФИО, предмет, оценка)
 - Вычислите средний балл по каждому предмету
 - Найдите студентов с оценками ниже 3
 - Отсортируйте студентов по среднему баллу

Вариант 2

- Инвентарная система:** Разработайте систему учёта товаров:
 - Класс `Product` с полями: ID, название, категория, цена, количество
 - Реализуйте функции:
 - Добавление/удаление товара
 - Поиск товаров по категории
 - Сортировка по цене/названию
 - Расчёт общей стоимости всех товаров
 - Используйте `QVector<Product>`
- Обработка текста:** Напишите программу для анализа текста:
 - Считайте текст из файла
 - Разбейте на слова (разделители: пробел, запятая, точка)
 - Удалите стоп-слова (заранее заданный список)
 - Подсчитайте частоту каждого слова
 - Выведите 10 самых частых слов

Критерии оценивания

- Удовлетворительно (3):** Реализована базовая функциональность с использованием одного типа контейнера, код компилируется.

- **Хорошо (4):** Полная реализация заданий с использованием нескольких методов контейнеров, хорошая организация кода (классы, функции).
- **Отлично (5):** Расширенная функциональность (поиск с фильтрами, сложные сортировки, работа с файлами), сравнение производительности разных контейнеров, обработка исключений, документация кода.

Советы для студентов

1. **Выбирайте контейнер осознанно:**
 - QVector для частого доступа по индексу
 - QList для частых вставок/удалений
 - std::vector для работы с алгоритмами STL
2. **Используйте range-based for:** Упрощает итерацию: `for (auto &item : list)`
3. **Избегайте копирования:** Для сложных объектов используйте ссылки или указатели.
4. **Проверяйте границы:** Всегда проверяйте `!list.isEmpty()` перед доступом к элементам.
5. **Освойте лямбда-выражения:** Упрощают код с алгоритмами STL:


```
cpp
std::sort(list.begin(), list.end(),
    [](const auto &a, const auto &b) { return a.value < b.value; });
```
6. **Для больших данных:** Рассмотрите `std::unordered_map` для быстрого поиска по ключу.

Общие рекомендации по выполнению практических работ

1. **Планирование:** Перед написанием кода составьте план:
 - Какие классы/структуры нужны?
 - Какие контейнеры использовать?
 - Какие алгоритмы применить?
 2. **Поэтапная разработка:**
 - Сначала реализуйте базовую функциональность
 - Потом добавляйте дополнительные возможности
 - В конце — обработку ошибок и оптимизацию
 3. **Тестирование:** Проверяйте на:
 - Нормальных данных
 - Граничных случаях (пустые списки, некорректный ввод)
 - Больших объёмах данных
 4. **Документирование:** Комментируйте:
 - Назначение функций
 - Сложные алгоритмы
 - Особенности реализации
 5. **Оптимизация (для продвинутых):**
 - Измеряйте время выполнения
 - Сравнивайте разные подходы
 - Используйте move-семантику для избежания копирования
- Эти практические работы помогут закрепить важные навыки, которые пригодятся при разработке реальных приложений. Удачи в выполнении!

ПРАКТИЧЕСКАЯ РАБОТА №11. Использование основных шаблонов.

Тема: Использование основных шаблонов проектирования (Стратегия и Наблюдатель)

Цель работы: Освоить принципы применения шаблонов проектирования для создания гибкого и поддерживаемого кода.

Задачи:

1. Изучить шаблоны "Стратегия" (Strategy) и "Наблюдатель" (Observer).
2. Реализовать примеры использования этих шаблонов на языке C++.
3. Применить шаблон "Наблюдатель" в контексте Qt (сигналы и слоты).

Теоретический материал (Кратко)

- **Стратегия (Strategy):** Определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми. Позволяет изменять алгоритмы независимо от клиентов, которые ими пользуются.
- **Наблюдатель (Observer):** Определяет зависимость "один-ко-многим" между объектами так, что при изменении состояния одного объекта все зависящие от него объекты уведомляются и обновляются автоматически.
- **В Qt:** Механизм сигналов и слотов является реализацией шаблона Наблюдатель.

Пример решения (Стратегия на C++)

сpp

```
#include <iostream>
```

```
#include <memory>
```

// Абстрактная стратегия (интерфейс)

```
class SortStrategy {
public:
    virtual void sort(std::vector<int>& data) = 0;
    virtual ~SortStrategy() = default;
};
```

// Конкретные стратегии

```
class BubbleSortStrategy : public SortStrategy {
public:
    void sort(std::vector<int>& data) override { /* Реализация пузырьковой сортировки */ }
};
```

```
class QuickSortStrategy : public SortStrategy {
public:
    void sort(std::vector<int>& data) override { /* Реализация быстрой сортировки */ }
};
```

// Контекст

```
class NumberSorter {
private:
    std::unique_ptr<SortStrategy> strategy_;
public:
    void setStrategy(std::unique_ptr<SortStrategy> strategy) {
        strategy_ = std::move(strategy);
    }
    void executeSort(std::vector<int>& data) {
        if (strategy_) {
            strategy_>sort(data);
        }
    }
};
```

```

    }
};

int main() {
    NumberSorter sorter;
    std::vector<int> numbers = {5, 2, 8, 1, 9};

    sorter.setStrategy(std::make_unique<BubbleSortStrategy>());
    sorter.executeSort(numbers);
    // Теперь можно легко поменять алгоритм:
    // sorter.setStrategy(std::make_unique<QuickSortStrategy>());

    return 0;
}

```

Задания для выполнения

Вариант 1 (Консольный - Стратегия):

1. Реализуйте шаблон "Стратегия" для вычисления площади фигур. Создайте классы-стратегии для вычисления площади круга, прямоугольника и треугольника.
2. Класс AreaCalculator (Контекст) должен иметь метод `calculate`, который делегирует вычисление текущей стратегии.
3. В `main` продемонстрируйте, как можно менять стратегию в runtime.

Вариант 2 (Qt - Наблюдатель):

1. Создайте Qt-приложение с двумя окнами (виджетами).
2. **Первое окно (Модель/Наблюдаемый):** Содержит `QSlider` и `QLineEdit`. При изменении слайдера значение должно отображаться в `QLineEdit`.
3. **Второе окно (Наблюдатель):** Должно отображать то же значение, что и в первом окне (например, в `QLabel`), но **без прямых ссылок на виджеты первого окна**.
4. Реализуйте эту синхронизацию, создав собственный класс-модель (`DataModel`), наследующий от `QObject` и имеющий сигнал `valueChanged(int)`. Оба окна должны работать с одной моделью, получая уведомления через сигналы и слоты.
5. Это и будет реализацией шаблона Наблюдатель средствами Qt.

Критерии оценивания

- **«Удовлетворительно»:** Реализована работающая стратегия (Вариант 1) или простой обмен данными между окнами Qt (Вариант 2).
- **«Хорошо»:** Чистая реализация шаблона "Стратегия" с правильным использованием полиморфизма (Вариант 1) или корректное использование модели и сигналов/слотов в Qt без жестких связей (Вариант 2).
- **«Отлично»:** Реализованы **оба** варианта (Стратегия в консоли и Наблюдатель в Qt) с четким разделением логики и интерфейса, написаны пояснения в отчете о соответствии кода принципам шаблонов.

ПРАКТИЧЕСКАЯ РАБОТА №12. Использование порождающих шаблонов.

Цель работы:

Освоить базовые порождающие шаблоны Singleton и Factory Method. Научиться применять их для решения типовых задач разработки программного обеспечения в C++/Qt.

Задачи:

1. Изучить принцип работы паттерна Singleton и его применение для управления глобальными ресурсами.
2. Освоить паттерн Factory Method для создания объектов без явного указания их классов.
3. Разработать систему логирования с использованием Singleton.
4. Создать систему обработки документов с применением Factory Method.

Теоретический материал (кратко)

1. Singleton (Одиночка)

Гарантирует, что класс имеет только один экземпляр и предоставляет к нему глобальную точку доступа.

Применение:

- Конфигурация системы
- Логирование
- Подключения к базам данных
- Кэширование

2. Factory Method (Фабричный метод)

Определяет интерфейс для создания объекта, но позволяет подклассам изменять тип создаваемых объектов.

Применение:

- Создание объектов без указания конкретных классов
- Расширяемость системы
- Тестирование (mock-объекты)

Часть 1: Реализация Singleton для системы логирования

Задание:

Создайте класс `Logger`, который:

1. Реализует паттерн Singleton
2. Поддерживает три уровня логирования: `DEBUG`, `INFO`, `ERROR`
3. Позволяет записывать логи в файл и/или выводить в консоль
4. Имеет методы для установки уровня логирования и файла вывода

Пример кода-заготовки:

cpp

```
#include <QCoreApplication>
#include <QDebug>
#include <QFile>
#include <QTextStream>
#include <QDateTime>
```

```
enum LogLevel {
    DEBUG,
    INFO,
    ERROR
};
```

```
class Logger {
private:
    static Logger* instance;
    LogLevel currentLevel;
```

```

QString logFilePath;
bool writeToFile;
bool writeToConsole;

Logger() {
    currentLevel = INFO;
    logFilePath = "application.log";
    writeToFile = true;
    writeToConsole = true;
}

public:
static Logger* getInstance() {
    if (!instance) {
        instance = new Logger();
    }
    return instance;
}

void log(LogLevel level, const QString& message) {
    if (level < currentLevel) return;

    QString timestamp = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss");
    QString levelStr;

    switch(level) {
        case DEBUG: levelStr = "DEBUG"; break;
        case INFO: levelStr = "INFO"; break;
        case ERROR: levelStr = "ERROR"; break;
    }

    QString logMessage = QString("[%1] %2: %3").arg(timestamp, levelStr, message);

    if (writeToConsole) {
        qDebug() << logMessage;
    }

    if (writeToFile) {
        QFile file(logFilePath);
        if (file.open(QIODevice::Append | QIODevice::Text)) {
            QTextStream stream(&file);
            stream << logMessage << "\n";
            file.close();
        }
    }
}

// Методы для настройки логгера
void setLogLevel(LogLevel level) { currentLevel = level; }
void setLogFile(const QString& path) { logFilePath = path; }
void setWriteToFile(bool enable) { writeToFile = enable; }
void setWriteToConsole(bool enable) { writeToConsole = enable; }

```

```
};
```

```
Logger* Logger::instance = nullptr;
```

Часть 2: Реализация Factory Method для обработки документов

Задание:

Создайте систему обработки документов разных типов:

1. Создайте базовый абстрактный класс `Document`

2. Реализуйте

конкретные

классы: `TextDocument`, `SpreadsheetDocument`, `PresentationDocument`

3. Создайте фабричный метод для создания документов

Пример структуры:

сpp

// Абстрактный класс документа

```
class Document {
public:
    virtual ~Document() {}
    virtual QString getType() const = 0;
    virtual void open() = 0;
    virtual void save() = 0;
    virtual void print() = 0;
};
```

// Конкретные классы документов

```
class TextDocument : public Document {
public:
    QString getType() const override { return "Text Document"; }
    void open() override { qDebug() << "Opening text document..."; }
    void save() override { qDebug() << "Saving text document..."; }
    void print() override { qDebug() << "Printing text document..."; }
};
```

```
class SpreadsheetDocument : public Document {
public:
    QString getType() const override { return "Spreadsheet"; }
    void open() override { qDebug() << "Opening spreadsheet..."; }
    void save() override { qDebug() << "Saving spreadsheet..."; }
    void print() override { qDebug() << "Printing spreadsheet..."; }
};
```

// Абстрактный создатель

```
class Application {
public:
    virtual Document* createDocument() = 0;

    void newDocument() {
        Document* doc = createDocument();
        qDebug() << "Created new document:" << doc->getType();
        doc->open();
    }
};
```

// Конкретные создатели

```

class TextEditor : public Application {
public:
    Document* createDocument() override {
        return new TextDocument();
    }
};

class SpreadsheetApp : public Application {
public:
    Document* createDocument() override {
        return new SpreadsheetDocument();
    }
};

```

Задания по вариантам

Вариант 1

Задача 1: Расширьте класс `Logger`, добавив возможность:

- Ротации логов (создание нового файла при превышении размера)
- Форматирования сообщений (добавление имени функции и файла)
- Фильтрации по тегам

Задача 2: Создайте систему учета сотрудников с использованием Factory Method:

- Базовый класс `Employee`
- Конкретные классы: `Developer`, `Manager`, `Designer`
- Фабрика для создания сотрудников разных типов
- Метод расчета зарплаты для каждого типа

Вариант 2

Задача 1: Создайте Singleton для управления настройками приложения:

- Хранение настроек в файле JSON
- Автоматическая загрузка/сохранение при изменении
- Уведомление наблюдателей об изменениях
- Поддержка разных профилей настроек

Задача 2: Реализуйте систему обработки графических файлов:

- Базовый класс `ImageProcessor`
- Конкретные процессоры: `JPEGProcessor`, `PNGProcessor`, `BMPPProcessor`
- Фабричный метод для создания процессора по расширению файла
- Методы: загрузка, сохранение, изменение размера, применение фильтров

Критерии оценивания

Удовлетворительно (3):

- Реализованы базовые версии Singleton и Factory Method
- Код компилируется без ошибок
- Выполнены основные требования заданий

Хорошо (4):

- Полная реализация всех задач варианта
- Правильное использование паттернов
- Хорошая организация кода (разделение на классы, методы)
- Обработка основных ошибок

Отлично (5):

- Расширенная функциональность (дополнительные возможности)
- Использование умных указателей (`std::unique_ptr`, `std::shared_ptr`)
- Покрытие тестами основных функций
- Графический интерфейс для демонстрации работы
- Документация кода (комментарии, README)

Советы для студентов

1. **Начинайте с простого:** Сначала реализуйте базовую версию паттерна, потом добавляйте функции.
2. **Используйте умные указатели:** Вместо сырых указателей используйте `std::unique_ptr` или `std::shared_ptr`.
3. **Тестируйте каждый шаг:** После реализации каждой функции проверяйте её работу.
4. **Обрабатывайте ошибки:** Проверяйте открытие файлов, валидность данных.
5. **Демонстрация работы:**

cpp

```
int main() {  
    // Демонстрация Singleton  
    Logger* logger = Logger::getInstance();  
    logger->log(INFO, "Application started");  
    logger->log(DEBUG, "Debug message");  
    logger->log(ERROR, "Error occurred!");  
  
    // Демонстрация Factory Method  
    Application* textApp = new TextEditor();  
    textApp->newDocument();  
  
    Application* sheetApp = new SpreadsheetApp();  
    sheetApp->newDocument();  
  
    return 0;  
}
```

Вопросы для самопроверки:

- Почему конструктор Singleton делается приватным?
- Когда использовать Factory Method вместо прямого создания объектов?
- Как обеспечить потокобезопасность Singleton?
- Какие преимущества даёт использование паттернов?

Что должно быть в отчёте:

1. Блок-схемы реализованных паттернов
2. Листинги всех классов с комментариями
3. Примеры использования (скриншоты или текстовый вывод)
4. Ответы на контрольные вопросы
5. Выводы о преимуществах и недостатках использованных паттернов

ПРАКТИЧЕСКАЯ РАБОТА №13. Использование структурных шаблонов.

Цель работы:

Изучить применение структурных шаблонов проектирования (Structural Patterns) для построения гибких и расширяемых систем. Освоить паттерны Adapter, Composite, Decorator и их практическое применение в разработке программного обеспечения.

Задачи:

1. Понять принципы структурных шаблонов и их отличие от порождающих.
2. Реализовать паттерн Adapter для совместимости несовместимых интерфейсов.
3. Применить паттерн Composite для работы с древовидными структурами.
4. Использовать паттерн Decorator для динамического добавления функциональности.
5. Создать систему управления графическими элементами с использованием структурных паттернов.

Теоретический материал

Структурные шаблоны проектирования

Структурные паттерны решают проблемы создания крупных структур из классов и объектов, обеспечивая гибкость и эффективность системы.

1. Adapter (Адаптер)

Назначение: Преобразует интерфейс одного класса в интерфейс, ожидаемый клиентом.

cpp

// Целевой интерфейс (тот, который ожидает клиент)

```
class Shape {
public:
    virtual void draw() = 0;
    virtual ~Shape() {}
};
```

// Адаптируемый класс (старая система)

```
class LegacyRectangle {
private:
    int x1, y1, x2, y2;

public:
    LegacyRectangle(int x1, int y1, int x2, int y2)
        : x1(x1), y1(y1), x2(x2), y2(y2) {}

    void oldDraw() {
        qDebug() << "LegacyRectangle: draw() at ["
            << x1 << ", " << y1 << "]" to ["
            << x2 << ", " << y2 << "];
    }
};
```

// Адаптер

```
class RectangleAdapter : public Shape {
private:
    LegacyRectangle* legacyRect;

public:
    RectangleAdapter(int x, int y, int width, int height) {
        legacyRect = new LegacyRectangle(x, y, x + width, y + height);
    }
};
```

```

void draw() override {
    legacyRect->oldDraw();
}

~RectangleAdapter() {
    delete legacyRect;
}
};

```

2. Composite (Компоновщик)

Назначение: Объединяет объекты в древовидные структуры для представления иерархий.

c++

// Компонент

```

class FileSystemComponent {
protected:
    QString name;

public:
    FileSystemComponent(const QString& n) : name(n) {}
    virtual ~FileSystemComponent() {}

    virtual void add(FileSystemComponent* component) {}
    virtual void remove(FileSystemComponent* component) {}
    virtual void display(int depth = 0) const = 0;
    virtual qint64 getSize() const = 0;
};

```

// Лист (не может иметь детей)

```

class File : public FileSystemComponent {
private:
    qint64 size;

public:
    File(const QString& n, qint64 s) : FileSystemComponent(n), size(s) {}

    void display(int depth = 0) const override {
        QString indent(depth, ' ');
        qDebug() << indent << "■" << name << "(" << size << " bytes";
    }

    qint64 getSize() const override {
        return size;
    }
};

```

// Композит (может иметь детей)

```

class Directory : public FileSystemComponent {
private:
    QList<FileSystemComponent*> children;

public:
    Directory(const QString& n) : FileSystemComponent(n) {}

```

```

void add(FileSystemComponent* component) override {
    children.append(component);
}

void remove(FileSystemComponent* component) override {
    children.removeOne(component);
}

void display(int depth = 0) const override {
    QString indent(depth, ' ');
    qDebug() << indent << "■" << name << "(total: " << getSize() << " bytes)";

    for (const auto& child : children) {
        child->display(depth + 2);
    }
}

qint64 getSize() const override {
    qint64 total = 0;
    for (const auto& child : children) {
        total += child->getSize();
    }
    return total;
}

~Directory() {
    for (auto child : children) {
        delete child;
    }
}
};

```

3. Decorator (Декоратор)

Назначение: Динамически добавляет объекту новые обязанности.

сpp

// Базовый компонент

```

class Coffee {
public:
    virtual QString getDescription() const = 0;
    virtual double cost() const = 0;
    virtual ~Coffee() {}
};

```

// Конкретный компонент

```

class Espresso : public Coffee {
public:
    QString getDescription() const override {
        return "Espresso";
    }

    double cost() const override {
        return 2.0;
    }
}

```

```
};
```

```
// Базовый декоратор
```

```
class CoffeeDecorator : public Coffee {  
protected:  
    Coffee* coffee;  
  
public:  
    CoffeeDecorator(Coffee* c) : coffee(c) {}  
    virtual ~CoffeeDecorator() {  
        delete coffee;  
    }  
};
```

```
// Конкретные декораторы
```

```
class MilkDecorator : public CoffeeDecorator {  
public:  
    MilkDecorator(Coffee* c) : CoffeeDecorator(c) {}  
  
    QString getDescription() const override {  
        return coffee->getDescription() + ", Milk";  
    }  
  
    double cost() const override {  
        return coffee->cost() + 0.5;  
    }  
};
```

```
class SugarDecorator : public CoffeeDecorator {  
public:  
    SugarDecorator(Coffee* c) : CoffeeDecorator(c) {}  
  
    QString getDescription() const override {  
        return coffee->getDescription() + ", Sugar";  
    }  
  
    double cost() const override {  
        return coffee->cost() + 0.2;  
    }  
};
```

```
class WhippedCreamDecorator : public CoffeeDecorator {  
public:  
    WhippedCreamDecorator(Coffee* c) : CoffeeDecorator(c) {}  
  
    QString getDescription() const override {  
        return coffee->getDescription() + ", Whipped Cream";  
    }  
  
    double cost() const override {  
        return coffee->cost() + 0.7;  
    }  
};
```

```
};
```

Задания по вариантам

Вариант 1: Система учета студентов с использованием Composite

Задача: Разработайте систему учета студентов факультета с использованием паттерна Composite.

1. **Базовый класс** `UniversityComponent` с методами:
 - `getName()` - возвращает название компонента
 - `display()` - выводит информацию
 - `getStudentCount()` - возвращает количество студентов
2. **Листовые классы:**
 - `Student` - студент (ФИО, группа, курс)
 - `Course` - учебный курс (название, преподаватель)
3. **Композитные классы:**
 - `Group` - учебная группа (список студентов)
 - `Department` - кафедра (список групп и курсов)
 - `Faculty` - факультет (список кафедр)
4. **Реализуйте функциональность:**
 - Добавление/удаление компонентов
 - Поиск студента по ФИО
 - Расчет средней успеваемости группы
 - Вывод иерархической структуры факультета

Вариант 2: Система оформления заказов с использованием Adapter и Decorator

Задача: Создайте систему оформления заказов в интернет-магазине.

1. **Adapter для платежных систем:**
 - Целевой интерфейс: `PaymentSystem` с методом `processPayment()`
 - Адаптируемые классы: `LegacyPayPal`, `OldBankSystem`
 - Адаптеры для совместимости со старой платежной системой
2. **Decorator для добавления услуг:**
 - Базовый класс: `Order` (товар, цена)
 - Декораторы:
 - `GiftWrapDecorator` - подарочная упаковка (+\$5)
 - `ExpressDeliveryDecorator` - экспресс-доставка (+\$10)
 - `InsuranceDecorator` - страховка (+\$3)
 - `DiscountDecorator` - скидка (-10%)
3. **Реализуйте функциональность:**
 - Создание заказа с различными опциями
 - Расчет итоговой стоимости
 - Оформление заказа через разные платежные системы
 - Генерация чека

Пример решения (Composite)

```
cpp
#include <QCoreApplication>
#include <QDebug>
#include <QString>
#include <QList>
```

```
// Базовый компонент
class UniversityComponent {
protected:
    QString name;

public:
```

```

UniversityComponent(const QString& n) : name(n) {}
virtual ~UniversityComponent() {}

virtual void add(UniversityComponent* component) {}
virtual void remove(UniversityComponent* component) {}
virtual void display(int depth = 0) const = 0;
virtual int getStudentCount() const = 0;

QString getName() const { return name; }
};

// Лист: Студент
class Student : public UniversityComponent {
private:
    QString group;
    int course;
    double averageGrade;

public:
    Student(const QString& name, const QString& grp, int crs, double grade)
        : UniversityComponent(name), group(grp), course(crs), averageGrade(grade) {}

    void display(int depth = 0) const override {
        QString indent(depth, ' ');
        qDebug() << indent << "👤📖" << name
            << "| Группа:" << group
            << "| Курс:" << course
            << "| Средний балл:" << averageGrade;
    }

    int getStudentCount() const override {
        return 1;
    }

    double getAverageGrade() const { return averageGrade; }
    QString getGroup() const { return group; }
};

// Композит: Группа
class Group : public UniversityComponent {
private:
    QList<UniversityComponent*> students;
    QString groupCode;

public:
    Group(const QString& code)
        : UniversityComponent("Группа " + code), groupCode(code) {}

    void add(UniversityComponent* component) override {
        students.append(component);
    }
}

```

```

void remove(UniversityComponent* component) override {
    students.removeOne(component);
}

void display(int depth = 0) const override {
    QString indent(depth, ' ');
    qDebug() << indent << "👤" << name
        << "| Студентов:" << getStudentCount();

    for (const auto& student : students) {
        student->display(depth + 2);
    }
}

int getStudentCount() const override {
    int count = 0;
    for (const auto& student : students) {
        count += student->getStudentCount();
    }
    return count;
}

double calculateAverageGrade() const {
    if (students.isEmpty()) return 0.0;

    double total = 0.0;
    int count = 0;

    for (const auto& student : students) {
        // Динамическое приведение типа
        Student* s = dynamic_cast<Student*>(student);
        if (s) {
            total += s->getAverageGrade();
            count++;
        }
    }

    return count > 0 ? total / count : 0.0;
}

~Group() {
    for (auto student : students) {
        delete student;
    }
}

};

// Композит: Факультет
class Faculty : public UniversityComponent {
private:
    QList<UniversityComponent*> departments;

```

```

public:
    Faculty(const QString& name) : UniversityComponent(name) {}

    void add(UniversityComponent* component) override {
        departments.append(component);
    }

    void remove(UniversityComponent* component) override {
        departments.removeOne(component);
    }

    void display(int depth = 0) const override {
        QString indent(depth, ' ');
        qDebug() << "\n" << indent << "🏢 ФАКУЛЬТЕТ:" << name
                << "\n| Всего студентов:" << getStudentCount();

        for (const auto& dept : departments) {
            dept->display(depth + 2);
        }
    }

    int getStudentCount() const override {
        int count = 0;
        for (const auto& dept : departments) {
            count += dept->getStudentCount();
        }
        return count;
    }

    // Поиск студента по ФИО
    Student* findStudent(const QString& studentName) {
        for (const auto& dept : departments) {
            Group* group = dynamic_cast<Group*>(dept);
            if (group) {
                // Здесь нужен обход студентов группы
                // Упрощенная реализация
            }
        }
        return nullptr;
    }

    ~Faculty() {
        for (auto dept : departments) {
            delete dept;
        }
    }
};

```

Критерии оценивания

Удовлетворительно (3):

- Реализован один структурный паттерн
- Код компилируется без ошибок
- Выполнены базовые требования задания

Хорошо (4):

- Реализованы 2 структурных паттерна
- Правильное применение паттернов
- Хорошая организация кода
- Демонстрация работы паттернов

Отлично (5):

- Реализованы все 3 паттерна (Adapter, Composite, Decorator)
- Интеграция паттернов в единую систему
- Дополнительная функциональность
- Графический интерфейс для демонстрации
- Тестирование и обработка ошибок

Советы для студентов

1. **Начинайте с простого:** Реализуйте сначала базовые классы, потом добавляйте функциональность.
2. **Используйте динамическое приведение:** Для работы с конкретными типами в Composite:
cpp
Student* student = dynamic_cast<Student*>(component);
if (student) {
 // работа со студентом
}
3. **Осторожно с памятью:** Используйте умные указатели или явно удаляйте объекты в деструкторах.
4. **Тестируйте иерархии:** Для Composite создавайте глубокие иерархии для проверки.
5. **Пример использования:**

```
cpp
int main() {
    // Создание факультета
    Faculty* csFaculty = new Faculty("Информационные технологии");

    // Создание групп
    Group* group101 = new Group("ИТ-101");
    Group* group102 = new Group("ИТ-102");

    // Добавление студентов
    group101->add(new Student("Иванов Иван", "ИТ-101", 1, 4.5));
    group101->add(new Student("Петров Петр", "ИТ-101", 1, 4.2));

    group102->add(new Student("Сидоров Алексей", "ИТ-102", 1, 4.8));

    // Добавление групп на факультет
    csFaculty->add(group101);
    csFaculty->add(group102);

    // Вывод структуры
    csFaculty->display();

    // Очистка памяти
    delete csFaculty;

    return 0;
}
```

6. **Вопросы для самопроверки:**

- Когда использовать Adapter вместо изменения исходного класса?
- Какие преимущества дает Composite при работе с древовидными структурами?
- Чем Decorator отличается от наследования?
- Как избежать проблем с памятью при использовании Composite?

Что должно быть в отчёте:

1. Диаграммы классов реализованных паттернов
2. Листинги всех классов с комментариями
3. Примеры использования с выводом результатов
4. Сравнение паттернов и рекомендации по их применению
5. Выводы о преимуществах структурных шаблонов

Примечание: Эта работа фокусируется на практическом применении структурных паттернов, что является важным навыком для разработки сложных систем. Паттерны Adapter, Composite и Decorator часто используются вместе для создания гибких архитекту

ПРАКТИЧЕСКАЯ РАБОТА №14 -15. Использование поведенческих шаблонов.

Цель работы:

Изучить и применить на практике ключевые поведенческие шаблоны проектирования для организации эффективного взаимодействия объектов в системе. Освоить паттерны Strategy, Observer, Command и их практическое применение в разработке приложений.

Задачи:

1. Изучить принципы поведенческих шаблонов и их отличие от структурных и порождающих.
2. Реализовать паттерн Strategy для выбора алгоритма во время выполнения.
3. Применить паттерн Observer для создания систем уведомлений.
4. Использовать паттерн Command для инкапсуляции запросов.
5. Разработать систему управления заказами с использованием поведенческих паттернов.

Теоретический материал

Поведенческие шаблоны проектирования

Поведенческие паттерны решают проблемы эффективного распределения обязанностей между объектами и управления их взаимодействием.

1. Strategy (Стратегия)

Назначение: Определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми.

cpp

// Интерфейс стратегии

```
class PaymentStrategy {
public:
    virtual void pay(double amount) = 0;
    virtual ~PaymentStrategy() {}
};
```

// Конкретные стратегии

```
class CreditCardPayment : public PaymentStrategy {
private:
    QString cardNumber;
    QString cardHolder;

public:
    CreditCardPayment(const QString& num, const QString& holder)
        : cardNumber(num), cardHolder(holder) {}

    void pay(double amount) override {
        qDebug() << "Processing credit card payment of $" << amount;
        qDebug() << "Card:" << cardNumber << "Holder:" << cardHolder;
        // Логика обработки платежа по карте
    }
};
```

```
class PayPalPayment : public PaymentStrategy {
private:
    QString email;

public:
    PayPalPayment(const QString& email) : email(email) {}

    void pay(double amount) override {
```

```

        qDebug() << "Processing PayPal payment of $" << amount;
        qDebug() << "Email:" << email;
        // Логика PayPal
    }
};

class BitcoinPayment : public PaymentStrategy {
private:
    QString walletAddress;

public:
    BitcoinPayment(const QString& address) : walletAddress(address) {}

    void pay(double amount) override {
        qDebug() << "Processing Bitcoin payment of $" << amount;
        qDebug() << "Wallet:" << walletAddress;
        // Логика Bitcoin
    }
};

// Контекст
class ShoppingCart {
private:
    PaymentStrategy* paymentStrategy;
    double totalAmount;

public:
    ShoppingCart() : paymentStrategy(nullptr), totalAmount(0) {}

    void addItem(double price) {
        totalAmount += price;
        qDebug() << "Added item: $" << price << " | Total: $" << totalAmount;
    }

    void setPaymentStrategy(PaymentStrategy* strategy) {
        paymentStrategy = strategy;
    }

    void checkout() {
        if (!paymentStrategy) {
            qDebug() << "Please select a payment method first!";
            return;
        }

        qDebug() << "\n=== CHECKOUT ===";
        qDebug() << "Total amount: $" << totalAmount;
        paymentStrategy->pay(totalAmount);
        qDebug() << "Payment successful!";
    }

    ~ShoppingCart() {
        delete paymentStrategy;
    }
};

```

```
    }  
};
```

2. Observer (Наблюдатель)

Назначение: Определяет зависимость "один ко многим" между объектами, чтобы при изменении состояния одного объекта все зависящие от него объекты автоматически уведомлялись.

cpp

```
#include <QList>
```

```
// Интерфейс наблюдателя
```

```
class Observer {  
public:  
    virtual void update(const QString& message) = 0;  
    virtual ~Observer() {}  
};
```

```
// Интерфейс субъекта
```

```
class Subject {  
protected:  
    QList<Observer*> observers;  
  
public:  
    virtual void attach(Observer* observer) {  
        observers.append(observer);  
    }  
  
    virtual void detach(Observer* observer) {  
        observers.removeOne(observer);  
    }  
  
    virtual void notify(const QString& message) {  
        for (Observer* observer : observers) {  
            observer->update(message);  
        }  
    }  
};
```

```
// Конкретный субъект
```

```
class WeatherStation : public Subject {  
private:  
    double temperature;  
    double humidity;  
    double pressure;  
  
public:  
    void setMeasurements(double temp, double hum, double press) {  
        temperature = temp;  
        humidity = hum;  
        pressure = press;  
  
        QString message = QString("Weather changed: Temp=%1°C, Humidity=%2%,  
Pressure=%3hPa")
```

```

        .arg(temperature).arg(humidity).arg(pressure);
    notify(message);
}

double getTemperature() const { return temperature; }
double getHumidity() const { return humidity; }
double getPressure() const { return pressure; }
};

// Конкретные наблюдатели
class CurrentConditionsDisplay : public Observer {
private:
    double temperature;
    double humidity;

public:
    void update(const QString& message) override {
        qDebug() << "[Current Conditions] " << message;
        // Здесь можно парсить сообщение для получения конкретных данных
    }

    void display() {
        qDebug() << "Current conditions: " << temperature << "°C and "
            << humidity << "% humidity";
    }
};

class StatisticsDisplay : public Observer {
private:
    double maxTemp = 0;
    double minTemp = 100;
    double tempSum = 0;
    int numReadings = 0;

public:
    void update(const QString& message) override {
        qDebug() << "[Statistics] " << message;
        // Анализ статистики
    }

    void display() {
        qDebug() << "Avg/Max/Min temperature = "
            << (numReadings > 0 ? tempSum / numReadings : 0)
            << "/" << maxTemp << "/" << minTemp;
    }
};

class ForecastDisplay : public Observer {
public:
    void update(const QString& message) override {
        qDebug() << "[Forecast] " << message;
        qDebug() << "Forecast: Improving weather on the way!";
    }
};

```

```
}  
};
```

3. Command (Команда)

Назначение: Инкапсулирует запрос как объект, позволяя параметризовать клиенты с различными запросами, ставить запросы в очередь или логировать их.

сpp

// Интерфейс команды

```
class Command {  
public:  
    virtual void execute() = 0;  
    virtual void undo() = 0;  
    virtual QString getDescription() const = 0;  
    virtual ~Command() {}  
};
```

// Получатель

```
class Light {  
private:  
    bool isOn;  
    QString location;  
  
public:  
    Light(const QString& loc) : isOn(false), location(loc) {}  
  
    void on() {  
        isOn = true;  
        qDebug() << location << "light is ON";  
    }  
  
    void off() {  
        isOn = false;  
        qDebug() << location << "light is OFF";  
    }  
  
    bool getState() const { return isOn; }  
};
```

// Конкретные команды

```
class LightOnCommand : public Command {  
private:  
    Light* light;  
  
public:  
    LightOnCommand(Light* l) : light(l) {}  
  
    void execute() override {  
        light->on();  
    }  
  
    void undo() override {  
        light->off();  
    }  
};
```

```

    QString getDescription() const override {
        return "Turn light ON";
    }
};

class LightOffCommand : public Command {
private:
    Light* light;

public:
    LightOffCommand(Light* l) : light(l) {}

    void execute() override {
        light->off();
    }

    void undo() override {
        light->on();
    }

    QString getDescription() const override {
        return "Turn light OFF";
    }
};

// Команда для стереосистемы
class Stereo {
private:
    int volume;
    bool isOn;

public:
    Stereo() : volume(5), isOn(false) {}

    void on() {
        isOn = true;
        qDebug() << "Stereo is ON";
    }

    void off() {
        isOn = false;
        qDebug() << "Stereo is OFF";
    }

    void setCD() {
        if (isOn) qDebug() << "CD is set";
    }

    void setVolume(int vol) {
        volume = vol;
        if (isOn) qDebug() << "Volume set to" << volume;
    }
};

```

```

    }
};

class StereoOnWithCDCommand : public Command {
private:
    Stereo* stereo;

public:
    StereoOnWithCDCommand(Stereo* s) : stereo(s) {}

    void execute() override {
        stereo->on();
        stereo->setCD();
        stereo->setVolume(11);
    }

    void undo() override {
        stereo->off();
    }

    QString getDescription() const override {
        return "Turn stereo ON with CD";
    }
};

```

// Инициатор

```

class RemoteControl {
private:
    Command* onCommands[7];
    Command* offCommands[7];
    Command* undoCommand;

public:
    RemoteControl() {
        for (int i = 0; i < 7; i++) {
            onCommands[i] = nullptr;
            offCommands[i] = nullptr;
        }
        undoCommand = nullptr;
    }

    void setCommand(int slot, Command* onCommand, Command* offCommand) {
        onCommands[slot] = onCommand;
        offCommands[slot] = offCommand;
    }

    void onButtonWasPushed(int slot) {
        if (onCommands[slot]) {
            onCommands[slot]->execute();
            undoCommand = onCommands[slot];
        }
    }
}

```

```

void offButtonWasPushed(int slot) {
    if (offCommands[slot]) {
        offCommands[slot]->execute();
        undoCommand = offCommands[slot];
    }
}

void undoButtonWasPushed() {
    if (undoCommand) {
        qDebug() << "--- UNDO ---";
        undoCommand->undo();
    }
}

void display() {
    qDebug() << "\n===== Remote Control =====";
    for (int i = 0; i < 7; i++) {
        if (onCommands[i] && offCommands[i]) {
            qDebug() << "[slot" << i << "]"
                << onCommands[i]->getDescription() << " / "
                << offCommands[i]->getDescription();
        }
    }
}
};

```

Задания по вариантам

Вариант 1: Система управления умным домом

Задача: Разработайте систему управления умным домом с использованием поведенческих паттернов.

1. **Strategy для режимов работы:**
 - EconomyMode - экономичный режим (минимальное энергопотребление)
 - ComfortMode - комфортный режим (оптимальные условия)
 - PartyMode - праздничный режим (максимальный комфорт)
2. **Observer для датчиков:**
 - TemperatureSensor - датчик температуры
 - MotionSensor - датчик движения
 - LightSensor - датчик освещенности
 - Наблюдатели: SecuritySystem, ClimateControl, LightingSystem
3. **Command для управления устройствами:**
 - Команды для: освещения, кондиционера, системы безопасности
 - Поддержка отмены операций (undo)
 - Запись макросов (последовательности команд)

Требования:

- Реализуйте автоматическое переключение режимов в зависимости от времени суток
- При обнаружении движения включайте освещение
- Поддерживайте температуру в заданных пределах
- Ведите журнал всех операций

Вариант 2: Система обработки заказов в ресторане

Задача: Создайте систему обработки заказов для ресторана.

1. **Strategy для стратегий приготовления:**
 - StandardCooking - стандартное приготовление

- FastCooking - быстрое приготовление (для доставки)
- GourmetCooking - изысканное приготовление
- 2. **Observer для уведомлений:**
 - Наблюдатели: KitchenDisplay, WaiterApp, CustomerDisplay
 - Уведомления: новый заказ, статус готовности, заказ отменен
- 3. **Command для операций с заказами:**
 - Команды: добавить блюдо, удалить блюдо, отменить заказ, изменить статус
 - История команд для отмены изменений

Требования:

- Разные стратегии приготовления для разных типов блюд
- Мгновенные уведомления всем участникам
- Возможность отмены последних операций
- Расчет времени приготовления

Пример решения (Система умного дома)

cpp

```
#include <QCoreApplication>
#include <QDebug>
#include <QString>
#include <QList>
#include <QDateTime>

// ===== Strategy Pattern =====

class HomeModeStrategy {
public:
    virtual void execute() = 0;
    virtual QString getName() const = 0;
    virtual ~HomeModeStrategy() {}
};

class EconomyMode : public HomeModeStrategy {
public:
    void execute() override {
        qDebug() << "🏠 Экономный режим:";
        qDebug() << " - Температура: 20°C";
        qDebug() << " - Освещение: минимальное";
        qDebug() << " - Вентиляция: экономичная";
    }

    QString getName() const override { return "Экономный режим"; }
};

class ComfortMode : public HomeModeStrategy {
public:
    void execute() override {
        qDebug() << "🏠 Комфортный режим:";
        qDebug() << " - Температура: 22°C";
        qDebug() << " - Освещение: оптимальное";
        qDebug() << " - Вентиляция: комфортная";
    }

    QString getName() const override { return "Комфортный режим"; }
};
```

```

class PartyMode : public HomeModeStrategy {
public:
    void execute() override {
        qDebug() << "🏠 Праздничный режим:";
        qDebug() << " - Температура: 24°C";
        qDebug() << " - Освещение: яркое";
        qDebug() << " - Вентиляция: интенсивная";
        qDebug() << " - Музыка: включена";
    }

    QString getName() const override { return "Праздничный режим"; }
};

// ===== Observer Pattern =====
class SensorObserver {
public:
    virtual void update(const QString& sensorName, double value) = 0;
    virtual ~SensorObserver() {}
};

class Sensor {
private:
    QString name;
    QList<SensorObserver*> observers;

public:
    Sensor(const QString& n) : name(n) {}

    void attach(SensorObserver* observer) {
        observers.append(observer);
    }

    void detach(SensorObserver* observer) {
        observers.removeOne(observer);
    }

    void setValue(double value) {
        qDebug() << "Датчик" << name << ": новое значение =" << value;
        notifyObservers(value);
    }

private:
    void notifyObservers(double value) {
        for (SensorObserver* observer : observers) {
            observer->update(name, value);
        }
    }
};

// Конкретные наблюдатели
class SecuritySystem : public SensorObserver {

```

```

public:
    void update(const QString& sensorName, double value) override {
        if (sensorName == "Датчик движения" && value > 0) {
            qDebug() << "🔊 Система безопасности: Обнаружено движение!";
        }
    }
};

class LightingSystem : public SensorObserver {
private:
    bool lightsOn;

public:
    LightingSystem() : lightsOn(false) {}

    void update(const QString& sensorName, double value) override {
        if (sensorName == "Датчик освещенности" && value < 50) {
            if (!lightsOn) {
                qDebug() << "💡 Система освещения: Включаю свет";
                lightsOn = true;
            }
        } else if (sensorName == "Датчик движения" && value == 0) {
            if (lightsOn) {
                qDebug() << "💡 Система освещения: Выключаю свет (нет движения)";
                lightsOn = false;
            }
        }
    }
};

class ClimateControl : public SensorObserver {
private:
    double targetTemp;

public:
    ClimateControl(double temp = 22.0) : targetTemp(temp) {}

    void update(const QString& sensorName, double value) override {
        if (sensorName == "Датчик температуры") {
            if (value < targetTemp - 1) {
                qDebug() << "🔥 Климат-контроль: Включаю обогрев";
            } else if (value > targetTemp + 1) {
                qDebug() << "❄️ Климат-контроль: Включаю охлаждение";
            } else {
                qDebug() << "🌡️ Климат-контроль: Температура оптимальна";
            }
        }
    }

    void setTargetTemperature(double temp) {
        targetTemp = temp;
    }
};

```

```

        qDebug() << "💡 Установлена целевая температура:" << temp << "°C";
    }
};

// ===== Command Pattern =====
class SmartHomeCommand {
public:
    virtual void execute() = 0;
    virtual void undo() = 0;
    virtual QString getDescription() const = 0;
    virtual ~SmartHomeCommand() {}
};

class TurnLightsOnCommand : public SmartHomeCommand {
private:
    LightingSystem* lighting;
    bool wasOn;

public:
    TurnLightsOnCommand(LightingSystem* ls) : lighting(ls), wasOn(false) {}

    void execute() override {
        qDebug() << "💡 Команда: Включить свет";
        // В реальной системе здесь был бы вызов метода lighting->turnOn()
        wasOn = true;
    }

    void undo() override {
        if (wasOn) {
            qDebug() << "💡 Отмена: Выключить свет";
        }
    }

    QString getDescription() const override {
        return "Включить освещение";
    }
};

class SetTemperatureCommand : public SmartHomeCommand {
private:
    ClimateControl* climate;
    double oldTemp;
    double newTemp;

public:
    SetTemperatureCommand(ClimateControl* cc, double temp)
        : climate(cc), newTemp(temp), oldTemp(22.0) {}

    void execute() override {
        oldTemp = 22.0; // В реальной системе получали бы текущую температуру
        climate->setTargetTemperature(newTemp);
    }
};

```

```

void undo() override {
    qDebug() << "🔙 Отмена: Возврат температуры к" << oldTemp << "°C";
    climate->setTargetTemperature(oldTemp);
}

QString getDescription() const override {
    return QString("Установить температуру %1°C").arg(newTemp);
}
};

// Контроллер умного дома
class SmartHomeController {
private:
    HomeModeStrategy* currentMode;
    QList<SmartHomeCommand*> commandHistory;

public:
    SmartHomeController() : currentMode(nullptr) {}

    void setMode(HomeModeStrategy* mode) {
        qDebug() << "\n=== Переключение на режим: " << mode->getName() << " ===";
        currentMode = mode;
        currentMode->execute();
    }

    void executeCommand(SmartHomeCommand* command) {
        command->execute();
        commandHistory.append(command);
        qDebug() << "Команда добавлена в историю:" << command->getDescription();
    }

    void undoLastCommand() {
        if (!commandHistory.isEmpty()) {
            SmartHomeCommand* lastCommand = commandHistory.takeLast();
            lastCommand->undo();
            delete lastCommand;
        } else {
            qDebug() << "История команд пуста";
        }
    }

    void showStatus() {
        qDebug() << "\n=== Статус умного дома ===";
        qDebug() << "Текущий режим:" << (currentMode ? currentMode->getName() : "Не
установлен");
        qDebug() << "Выполнено команд:" << commandHistory.size();
    }

    ~SmartHomeController() {
        for (auto cmd : commandHistory) {
            delete cmd;
        }
    }
};

```

```

    }
    delete currentMode;
}
};

```

Критерии оценивания

Удовлетворительно (3):

- Реализован один поведенческий паттерн
- Код компилируется без ошибок
- Выполнены базовые требования задания

Хорошо (4):

- Реализованы 2 поведенческих паттерна
- Правильное применение паттернов
- Хорошая организация кода
- Демонстрация взаимодействия паттернов

Отлично (5):

- Реализованы все 3 паттерна (Strategy, Observer, Command)
- Интеграция паттернов в единую систему
- Дополнительная функциональность
- Графический интерфейс для управления
- Тестирование и обработка ошибок

Советы для студентов 2 курса

1. **Начинайте с интерфейсов:** Определите абстрактные классы (интерфейсы) для каждого паттерна.
2. **Разделяйте ответственность:** Каждый класс должен выполнять одну четкую задачу.
3. **Используйте умные указатели:** Для управления памятью в сложных системах.
4. **Тестируйте взаимодействия:** Проверяйте, как паттерны работают вместе.
5. **Пример использования:**

```

сpp
int main() {
    // Создаем систему умного дома
    SmartHomeController controller;

    // Стратегии
    controller.setMode(new ComfortMode());

    // Наблюдатели
    ClimateControl climate(22.0);
    LightingSystem lighting;
    SecuritySystem security;

    Sensor motionSensor("Датчик движения");
    Sensor tempSensor("Датчик температуры");
    Sensor lightSensor("Датчик освещенности");

    motionSensor.attach(&lighting);
    motionSensor.attach(&security);
    tempSensor.attach(&climate);
    lightSensor.attach(&lighting);

    // Имитация событий
    motionSensor.setValue(1.0); // Обнаружено движение
    lightSensor.setValue(30.0); // Стало темно
}

```

```
tempSensor.setValue(18.0); // Похолодало
```

```
// Команды
```

```
controller.executeCommand(new SetTemperatureCommand(&climate, 24.0));
```

```
controller.executeCommand(new TurnLightsOnCommand(&lighting));
```

```
controller.undoLastCommand();
```

```
controller.showStatus();
```

```
return 0;
```

```
}
```

6. **Вопросы для самопроверки:**

- Когда использовать Strategy вместо наследования?
- Как Observer помогает уменьшить связанность классов?
- Какие преимущества дает Command для отмены операций?
- Как организовать взаимодействие между разными паттернами?

Что должно быть в отчёте:

1. Диаграммы классов реализованных паттернов
2. Листинги всех классов с комментариями
3. Примеры использования с выводом результатов
4. Сравнение паттернов и рекомендации по их применению
5. Выводы о преимуществах поведенческих шаблонов

ПРАКТИЧЕСКАЯ РАБОТА №16. Разработка приложения с использованием текстовых компонентов

Цель работы:

Освоить базовые текстовые компоненты Qt (QLineEdit, QTextEdit, QLabel) и научиться создавать простые текстовые приложения с основными функциями.

Задачи:

1. Изучить основные свойства и методы текстовых компонентов
2. Научиться получать и устанавливать текст
3. Освоить обработку событий текстовых полей
4. Создать простое приложение для работы с текстом

Теоретический материал

1. Основные текстовые компоненты:

QLabel - метка для отображения текста

```
cpp
QLabel* label = new QLabel("Привет, мир!", this);
label->setAlignment(Qt::AlignCenter); // Выравнивание по центру
```

QLineEdit - однострочное поле ввода

```
cpp
QLineEdit* lineEdit = new QLineEdit(this);
lineEdit->setPlaceholderText("Введите имя...");
QString text = lineEdit->text(); // Получить текст
```

QTextEdit - многострочное поле ввода

```
cpp
QTextEdit* textEdit = new QTextEdit(this);
textEdit->setPlainText("Многострочный\ntекст");
QString text = textEdit->toPlainText(); // Получить текст
```

2. Основные сигналы (события):

```
cpp
// QLineEdit
textChanged(const QString &text) - текст изменился
returnPressed() - нажата клавиша Enter
```

```
// QTextEdit
textChanged() - текст изменился
```

3. Простая работа с файлами:

```
cpp
// Чтение из файла
QFile file("имя_файла.txt");
if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
    QTextStream stream(&file);
    QString content = stream.readAll();
    file.close();
}
```

```
// Запись в файл
QFile file("имя_файла.txt");
if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
    QTextStream stream(&file);
    stream << content;
    file.close();
}
```

ВАРИАНТ 1: Текстовый редактор с подсчетом статистики

Задача:

Создайте простой текстовый редактор, который показывает статистику текста (количество символов, слов, строк).

Пошаговая инструкция:

Шаг 1: Создайте новый проект Qt Widgets Application

Шаг 2: В mainwindow.h:

```
cpp
#include <QMainWindow>
#include <QTextEdit>
#include <QLabel>

class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    MainWindow(QWidget* parent = nullptr);

private slots:
    void updateStatistics();
    void newFile();
    void openFile();
    void saveFile();

private:
    void setupUI();
    void setupMenu();

    QTextEdit* textEdit;
    QLabel* statusLabel;
    QString currentFile;
};
```

Шаг 3: В mainwindow.cpp (основная часть):

```
cpp
#include "mainwindow.h"
#include <QMenuBar>
#include <QFileDialog>
#include <QMessageBox>
#include <QTextStream>
#include <QStatusBar>

MainWindow::MainWindow(QWidget* parent) : QMainWindow(parent) {
    setupUI();
    setupMenu();
    currentFile = "";
    setWindowTitle("Текстовый редактор");
}

void MainWindow::setupUI() {
    // Центральный текстовый редактор
    textEdit = new QTextEdit(this);
```

```

setCentralWidget(textEdit);

// Подключаем сигнал изменения текста
connect(textEdit, &QTextEdit::textChanged,
        this, &MainWindow::updateStatistics);

// Статусная строка
statusLabel = new QLabel(this);
statusBar()->addWidget(statusLabel);

resize(600, 400);
}

void MainWindow::setupMenu() {
    // Простое меню
    QMenu* fileMenu = menuBar()->addMenu("Файл");

    QAction* newAction = fileMenu->addAction("Новый");
    connect(newAction, &QAction::triggered, this, &MainWindow::newFile);

    QAction* openAction = fileMenu->addAction("Открыть...");
    connect(openAction, &QAction::triggered, this, &MainWindow::openFile);

    QAction* saveAction = fileMenu->addAction("Сохранить");
    connect(saveAction, &QAction::triggered, this, &MainWindow::saveFile);

    fileMenu->addSeparator();

    QAction* exitAction = fileMenu->addAction("Выход");
    connect(exitAction, &QAction::triggered, this, &QWidget::close);
}

void MainWindow::updateStatistics() {
    QString text = textEdit->toPlainText();

    // Подсчет статистики
    int chars = text.length();
    int words = text.split(' ', QString::SkipEmptyParts).count();
    int lines = text.split('\n').count();

    // Обновляем статусную строку
    QString info = QString("Символов: %1 | Слов: %2 | Строк: %3")
        .arg(chars).arg(words).arg(lines);
    statusLabel->setText(info);
}

void MainWindow::newFile() {
    if (!textEdit->toPlainText().isEmpty()) {
        QMessageBox::StandardButton reply = QMessageBox::question(this,
            "Новый файл", "Удалить текущий текст?",
            QMessageBox::Yes | QMessageBox::No);
    }
}

```

```

        if (reply == QMessageBox::Yes) {
            textEdit->clear();
            currentFile = "";
            setWindowTitle("Текстовый редактор");
        }
    }
}

void MainWindow::openFile() {
    QString fileName = QFileDialog::getOpenFileName(this,
        "Открыть файл", "", "Текстовые файлы (*.txt);;Все файлы (*)");

    if (!fileName.isEmpty()) {
        QFile file(fileName);
        if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
            QTextStream stream(&file);
            textEdit->setPlainText(stream.readAll());
            file.close();

            currentFile = fileName;
            setWindowTitle("Редактор: " + fileName);
        } else {
            QMessageBox::warning(this, "Ошибка", "Не удалось открыть файл");
        }
    }
}

void MainWindow::saveFile() {
    if (currentFile.isEmpty()) {
        // Если файл не был открыт, сохраняем как
        QString fileName = QFileDialog::getSaveFileName(this,
            "Сохранить файл", "", "Текстовые файлы (*.txt);;Все файлы (*)");

        if (fileName.isEmpty()) return;
        currentFile = fileName;
    }

    QFile file(currentFile);
    if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QTextStream stream(&file);
        stream << textEdit->toPlainText();
        file.close();

        setWindowTitle("Редактор: " + currentFile);
        statusBar()->showMessage("Файл сохранен", 2000);
    } else {
        QMessageBox::warning(this, "Ошибка", "Не удалось сохранить файл");
    }
}

```

Дополнительное задание для варианта 1:

1. Добавьте кнопки для копирования/вставки текста
2. Реализуйте поиск слова в тексте (простой поиск через indexOf())

3. Добавьте возможность изменения размера шрифта (QSpinBox)

ВАРИАНТ 2: Приложение для заметок с категориями

Задача:

Создайте приложение для управления заметками с возможностью сортировки по категориям.

Пошаговая инструкция:

Шаг 1: Создайте новый проект

Шаг 2: В notewindow.h:

cpp

```
#include <QMainWindow>
#include <QListWidget>
#include <QTextEdit>
#include <QLineEdit>
#include <QComboBox>
#include <QPushButton>
```

```
class NoteWindow : public QMainWindow {
    Q_OBJECT
```

public:

```
    NoteWindow(QWidget* parent = nullptr);
```

private slots:

```
    void addNote();
    void deleteNote();
    void saveNotes();
    void loadNotes();
    void noteSelected(QListWidgetItem* item);
    void filterByCategory(int index);
```

private:

```
    void setupUI();
    void updateNoteList();
```

// Компоненты интерфейса

```
    QListWidget* noteList;
    QTextEdit* noteEditor;
    QLineEdit* titleEdit;
    QComboBox* categoryCombo;
    QPushButton* addButton;
    QPushButton* deleteButton;
```

// Данные

```
    struct Note {
        QString title;
        QString content;
        QString category;
    };

    QList<Note> notes;
    QString currentCategoryFilter;
```

```
};
```

Шаг 3: В notewindow.cpp (основная часть):

```
cpp
```

```
#include "notewindow.h"  
#include <QHBoxLayout>  
#include <QVBoxLayout>  
#include <QGroupBox>  
#include <QLabel>  
#include <QFileDialog>  
#include <QMessageBox>  
#include <QFile>  
#include <QTextStream>
```

```
NoteWindow::NoteWindow(QWidget* parent) : QMainWindow(parent) {  
    setupUI();  
    setWindowTitle("Менеджер заметок");  
    resize(700, 500);  
}
```

```
void NoteWindow::setupUI() {  
    // Главный виджет  
    QWidget* centralWidget = new QWidget(this);  
    QHBoxLayout* mainLayout = new QHBoxLayout(centralWidget);  
  
    // Левая панель - список заметок  
    QWidget* leftPanel = new QWidget;  
    QVBoxLayout* leftLayout = new QVBoxLayout(leftPanel);  
  
    // Поле для фильтрации по категории  
    QLabel* filterLabel = new QLabel("Категория:", this);  
    categoryCombo = new QComboBox(this);  
    categoryCombo->addItem("Все заметки");  
    categoryCombo->addItem("Работа");  
    categoryCombo->addItem("Учеба");  
    categoryCombo->addItem("Личное");  
    connect(categoryCombo, QOverload<int>::of(&QComboBox::currentIndexChanged),  
            this, &NoteWindow::filterByCategory);  
  
    // Список заметок  
    noteList = new QListWidget(this);  
    noteList->setMinimumWidth(200);  
    connect(noteList, &QListWidget::itemClicked,  
            this, &NoteWindow::noteSelected);  
  
    // Кнопки управления  
    addButton = new QPushButton("Добавить заметку", this);  
    deleteButton = new QPushButton("Удалить заметку", this);  
    connect(addButton, &QPushButton::clicked, this, &NoteWindow::addNote);  
    connect(deleteButton, &QPushButton::clicked, this, &NoteWindow::deleteNote);  
  
    // Добавляем компоненты на левую панель  
    leftLayout->addWidget(filterLabel);
```

```

leftLayout->addWidget(categoryCombo);
leftLayout->addWidget(noteList);
leftLayout->addWidget(addButton);
leftLayout->addWidget(deleteButton);

// Правая панель - редактирование заметки
QWidget* rightPanel = new QWidget;
QVBoxLayout* rightLayout = new QVBoxLayout(rightPanel);

// Поля для редактирования
QLabel* titleLabel = new QLabel("Заголовок:", this);
titleEdit = new QLineEdit(this);
titleEdit->setPlaceholderText("Введите заголовок заметки...");

QLabel* contentLabel = new QLabel("Текст заметки:", this);
noteEditor = new QTextEdit(this);
noteEditor->setPlaceholderText("Введите текст заметки здесь...");

// Кнопки сохранения/загрузки
QPushButton* saveButton = new QPushButton("Сохранить все заметки", this);
QPushButton* loadButton = new QPushButton("Загрузить заметки", this);
connect(saveButton, &QPushButton::clicked, this, &NoteWindow::saveNotes);
connect(loadButton, &QPushButton::clicked, this, &NoteWindow::loadNotes);

// Добавляем компоненты на правую панель
rightLayout->addWidget(titleLabel);
rightLayout->addWidget(titleEdit);
rightLayout->addWidget(contentLabel);
rightLayout->addWidget(noteEditor);
rightLayout->addWidget(saveButton);
rightLayout->addWidget(loadButton);

// Собираем интерфейс
mainLayout->addWidget(leftPanel, 1);
mainLayout->addWidget(rightPanel, 2);
setCentralWidget(centralWidget);

// Простое меню
QMenu* fileMenu = menuBar()->addMenu("Файл");
fileMenu->addAction("Сохранить", this, &NoteWindow::saveNotes);
fileMenu->addAction("Загрузить", this, &NoteWindow::loadNotes);
fileMenu->addSeparator();
fileMenu->addAction("Выход", this, &QWidget::close);
}

void NoteWindow::addNote() {
    QString title = titleEdit->text().trimmed();
    QString content = noteEditor->toPlainText().trimmed();
    QString category = categoryCombo->currentText();

    if (title.isEmpty()) {
        title = "Новая заметка " + QString::number(notes.size() + 1);
    }
}

```

```

    }

    Note newNote;
    newNote.title = title;
    newNote.content = content;
    newNote.category = (category == "Все заметки") ? "Без категории" : category;

    notes.append(newNote);
    updateNoteList();

    // Очищаем поля для новой заметки
    titleEdit->clear();
    noteEditor->clear();

    QMessageBox::information(this, "Успех", "Заметка добавлена!");
}

void NoteWindow::deleteNote() {
    int currentRow = noteList->currentRow();
    if (currentRow >= 0 && currentRow < notes.size()) {
        QMessageBox::StandardButton reply = QMessageBox::question(this,
            "Удаление", "Удалить выбранную заметку?",
            QMessageBox::Yes | QMessageBox::No);

        if (reply == QMessageBox::Yes) {
            notes.removeAt(currentRow);
            updateNoteList();
            titleEdit->clear();
            noteEditor->clear();
        }
    } else {
        QMessageBox::warning(this, "Ошибка", "Выберите заметку для удаления");
    }
}

void NoteWindow::noteSelected(QListWidgetItem* item) {
    int index = noteList->row(item);
    if (index >= 0 && index < notes.size()) {
        titleEdit->setText(notes[index].title);
        noteEditor->setPlainText(notes[index].content);
    }
}

void NoteWindow::filterByCategory(int index) {
    QString category = categoryCombo->itemText(index);
    currentCategoryFilter = (category == "Все заметки") ? "" : category;
    updateNoteList();
}

void NoteWindow::updateNoteList() {
    noteList->clear();

```

```

for (int i = 0; i < notes.size(); i++) {
    const Note& note = notes[i];

    // Фильтрация по категории
    if (!currentCategoryFilter.isEmpty() &&
        note.category != currentCategoryFilter) {
        continue;
    }

    // Добавляем заметку в список
    QString displayText = note.title;
    if (!note.category.isEmpty() && note.category != "Без категории") {
        displayText += " [" + note.category + "]";
    }

    QListWidgetItem* item = new QListWidgetItem(displayText, noteList);
    item->setData(Qt::UserRole, i); // Сохраняем индекс заметки
}

void NoteWindow::saveNotes() {
    QString fileName = QFileDialog::getSaveFileName(this,
        "Сохранить заметки", "", "Файлы заметок (*.notes);;Все файлы (*)");

    if (!fileName.isEmpty()) {
        QFile file(fileName);
        if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
            QTextStream stream(&file);

            for (const Note& note : notes) {
                stream << "===NOTE===\n";
                stream << "TITLE:" << note.title << "\n";
                stream << "CATEGORY:" << note.category << "\n";
                stream << "CONTENT:" << note.content << "\n";
                stream << "===END===\n";
            }

            file.close();
            QMessageBox::information(this, "Успех", "Заметки сохранены!");
        } else {
            QMessageBox::warning(this, "Ошибка", "Не удалось сохранить файл");
        }
    }
}

void NoteWindow::loadNotes() {
    QString fileName = QFileDialog::getOpenFileName(this,
        "Загрузить заметки", "", "Файлы заметок (*.notes);;Все файлы (*)");

    if (!fileName.isEmpty()) {
        QFile file(fileName);
        if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
            notes.clear(); // Очищаем старые заметки

```

```

QTextStream stream(&file);
Note currentNote;
bool inNote = false;

while (!stream.atEnd()) {
    QString line = stream.readLine();

    if (line == "===NOTE===") {
        inNote = true;
        currentNote = Note(); // Создаем новую заметку
    } else if (line == "===END===" && inNote) {
        notes.append(currentNote);
        inNote = false;
    } else if (line.startsWith("TITLE:")) {
        currentNote.title = line.mid(6);
    } else if (line.startsWith("CATEGORY:")) {
        currentNote.category = line.mid(9);
    } else if (line.startsWith("CONTENT:")) {
        currentNote.content = line.mid(8);
    }
}

file.close();
updateNoteList();
QMessageBox::information(this, "Успех",
    QString("Загружено %1 заметок").arg(notes.size()));
} else {
    QMessageBox::warning(this, "Ошибка", "Не удалось загрузить файл");
}
}
}

```

Дополнительное задание для варианта 2:

1. Добавьте поиск заметок по тексту
2. Реализуйте изменение категории существующей заметки
3. Добавьте дату создания заметки

Сравнение вариантов:

Критерий	Вариант 1 (Редактор)	Вариант 2 (Заметки)
Сложность	Средняя	Средняя
Основной компонент	QTextEdit	QListWidget + QTextEdit
Работа с файлами	Простая (один файл)	Сложнее (структурированные данные)
Обработка данных	Статистика текста	Управление списком объектов
Подходит для	Изучения QTextEdit	Изучения QListWidget и структур данных

Критерии оценивания (одинаковые для обоих вариантов)

Удовлетворительно (3):

- Реализован базовый функционал
- Программа компилируется без ошибок
- Все основные компоненты работают

Хорошо (4):

- Реализованы все обязательные функции
- Добавлена обработка ошибок
- Удобный и понятный интерфейс
- Код хорошо структурирован

Отлично (5):

- Реализованы все обязательные и дополнительные функции
 - Добавлены собственные улучшения
 - Качественный интерфейс
 - Полная документация кода
-

Советы для студентов 2 курса:

1. **Начинайте с простого:** Сначала сделайте базовый интерфейс, потом добавляйте функциональность.
2. **Тестируйте по шагам:** После каждой функции проверяйте, что она работает.
3. **Используйте qDebug():** Для отладки выводите информацию в консоль.
4. **Читайте сообщения об ошибках:** Qt часто подсказывает, что не так.
5. **Не бойтесь спрашивать:** Если что-то не получается, обратитесь к преподавателю.

Пример main.cpp для обоих вариантов:

```
cpp
#include <QApplication>
#include "mainwindow.h" // или "notewindow.h"

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    MainWindow window; // или NoteWindow window;
    window.show();

    return app.exec();
}
```

Вывод:

Да, эта работа **НОРМАЛЬНА** для 2 курса, потому что:

1. **Использует изученные концепции:** классы, сигналы/слоты, работа с файлами
2. **Практико-ориентированная:** создается реальное приложение
3. **Постепенное усложнение:** от простого к более сложному
4. **Два равноправных варианта:** разные, но одинаковой сложности
5. **Хорошая основа:** навыки пригодятся в будущих проектах

Важно: Преподаватель должен быть готов помогать со сложными моментами (QTextCursor, работа с файлами) и давать больше времени на выполнение.

ПРАКТИЧЕСКАЯ РАБОТА №17. Разработка приложения с несколькими формами.

ПРАКТИЧЕСКАЯ РАБОТА №18. Разработка приложения с не визуальными компонентами.

Цель работы:

Освоить разработку приложений с использованием невидимых компонентов Qt. Научиться создавать контекстные меню, работать с диалогами, динамически управлять элементами интерфейса и реализовывать функциональность просмотра изображений.

Задачи:

1. Создать контекстное меню для управления окном приложения.
2. Реализовать загрузку и отображение изображений.
3. Организовать историю последних открытых файлов.
4. Динамически создавать элементы меню.
5. Добавить настройки отображения изображений.

Теоретический материал

1. Невизуальные компоненты Qt

Невизуальные компоненты — это объекты, которые не отображаются на экране, но обеспечивают функциональность приложения:

- **QMenu** — меню и контекстные меню
- **QAction** — действие (пункт меню, кнопка)
- **QFileDialog** — диалог выбора файлов
- **QPixmap** — изображение в памяти
- **QSettings** — сохранение настроек

2. Основные методы работы с изображениями

сpp

// Загрузка изображения

```
QPixmap pixmap("путь/к/файлу.png");
```

// Проверка загрузки

```
if (!pixmap.isNull()) {
```

// Установка в QLabel

```
label->setPixmap(pixmap);
```

// Настройки отображения

```
label->setAlignment(Qt::AlignCenter); // Выравнивание
```

```
label->setScaledContents(true); // Растягивание
```

```
}
```

3. Сигналы и слоты (основная концепция Qt)

сpp

// Создание действия

```
QAction* action = new QAction("Открыть", this);
```

// Подключение сигнала к слоту

```
connect(action, &QAction::triggered, this, &MyClass::openFile);
```

// Когда пользователь нажимает на действие "Открыть",

// автоматически вызывается функция openFile()

Задание 1: Создание контекстного меню

Задача:

Создайте приложение с контекстным меню, которое появляется при щелчке правой кнопкой мыши по окну.

Пошаговая инструкция:

Шаг 1: Создайте новый проект Qt Widgets Application

Шаг 2: В mainwindow.h добавьте:

```
cpp
#include <QMainWindow>
#include <QMenu>

class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    MainWindow(QWidget* parent = nullptr);

protected:
    void contextMenuEvent(QContextMenuEvent* event) override;

private slots:
    void minimizeWindow();
    void restoreWindow();
    void closeWindow();
    void showAbout();

private:
    QMenu* contextMenu;
    void setupContextMenu();
};
```

Шаг 3: В mainwindow.cpp реализуйте:

```
cpp
#include "mainwindow.h"
#include <QContextMenuEvent>
#include <QAction>
#include <QMessageBox>

MainWindow::MainWindow(QWidget* parent) : QMainWindow(parent) {
    setWindowTitle("Приложение с контекстным меню");
    resize(400, 300);

    setupContextMenu();
}

void MainWindow::setupContextMenu() {
    // Создаем контекстное меню
    contextMenu = new QMenu(this);

    // Добавляем пункты меню
    QAction* minimizeAction = contextMenu->addAction("Свернуть окно");
    QAction* restoreAction = contextMenu->addAction("Восстановить окно");
    QAction* closeAction = contextMenu->addAction("Заккрыть");
}
```

```

contextMenu->addSeparator(); // Разделительная линия
QAction* aboutAction = contextMenu->addAction("О программе");

// Подключаем сигналы к слотам
connect(minimizeAction, &QAction::triggered, this, &MainWindow::minimizeWindow);
connect(restoreAction, &QAction::triggered, this, &MainWindow::restoreWindow);
connect(closeAction, &QAction::triggered, this, &MainWindow::closeWindow);
connect(aboutAction, &QAction::triggered, this, &MainWindow::showAbout);
}

void MainWindow::contextMenuEvent(QContextMenuEvent* event) {
    // Показываем меню в позиции клика мыши
    contextMenu->exec(event->globalPos());
}

// Реализация слотов (функций, вызываемых при нажатии)
void MainWindow::minimizeWindow() {
    showMinimized(); // Свернуть окно
}

void MainWindow::restoreWindow() {
    showNormal(); // Восстановить нормальный размер
}

void MainWindow::closeWindow() {
    close(); // Закрыть окно
}

void MainWindow::showAbout() {
    QMessageBox::about(this, "О программе",
        "Пример приложения с контекстным меню\n"
        "Разработано для практической работы №18");
}

```

Шаг 4: Запустите и протестируйте

1. Скомпилируйте и запустите программу
2. Щелкните правой кнопкой мыши по окну
3. Проверьте работу всех пунктов меню

Задание 2: Просмотрщик изображений

Задача:

Создайте простой просмотрщик изображений с главным меню.

Пошаговая инструкция:

Шаг 1: Создайте новый класс ImageViewer

imageviewer.h:

cpp

```

#include <QMainWindow>
#include <QLabel>
#include <QMenu>
#include <QAction>

```

```

class ImageViewer : public QMainWindow {
    Q_OBJECT

```

```

public:
    ImageViewer(QWidget* parent = nullptr);

private slots:
    void openImage();
    void toggleCenter(bool checked);
    void toggleStretch(bool checked);

private:
    void setupUI();
    void setupMenu();

    QLabel* imageLabel;
    QString currentImagePath;
};

```

**Шаг 2: Реализуйте основной функционал
imageviewer.cpp:**

```

cpp
#include "imageviewer.h"
#include <QFileDialog>
#include <QMenuBar>
#include <QMessageBox>
#include <QFileInfo>

ImageViewer::ImageViewer(QWidget* parent) : QMainWindow(parent) {
    setupUI();
    setupMenu();
}

void ImageViewer::setupUI() {
    setWindowTitle("Просмотрщик изображений");
    resize(600, 400);

    // Создаем метку для отображения изображения
    imageLabel = new QLabel(this);
    imageLabel->setAlignment(Qt::AlignCenter);
    imageLabel->setText("Изображение не загружено\nФайл → Открыть");
    imageLabel->setStyleSheet("QLabel { background-color: #f0f0f0; }");
    setCentralWidget(imageLabel);
}

void ImageViewer::setupMenu() {
    // Меню "Файл"
    QMenu* fileMenu = menuBar()->addMenu("&Файл");

    QAction* openAction = fileMenu->addAction("&Открыть");
    openAction->setShortcut(QKeySequence::Open); // Горячая клавиша Ctrl+O
    connect(openAction, &QAction::triggered, this, &ImageViewer::openImage);

    fileMenu->addSeparator();
}

```

```

QAction* exitAction = fileMenu->addAction("&Выход");
exitAction->setShortcut(QKeySequence::Quit); // Горячая клавиша Ctrl+Q
connect(exitAction, &QAction::triggered, this, &QWidget::close);

// Меню "Вид"
QMenu* viewMenu = menuBar()->addMenu("&Вид");

QAction* centerAction = viewMenu->addAction("По центру");
centerAction->setCheckable(true);
centerAction->setChecked(true);
connect(centerAction, &QAction::toggled, this, &ImageViewer::toggleCenter);

QAction* stretchAction = viewMenu->addAction("Растянуть");
stretchAction->setCheckable(true);
connect(stretchAction, &QAction::toggled, this, &ImageViewer::toggleStretch);
}

void ImageViewer::openImage() {
    // Открываем диалог выбора файла
    QString fileName = QFileDialog::getOpenFileName(this,
        "Выберите изображение",
        "",
        "Изображения (*.png *.jpg *.jpeg *.bmp *.gif)");

    if (!fileName.isEmpty()) {
        // Загружаем изображение
        QPixmap pixmap(fileName);

        if (pixmap.isNull()) {
            QMessageBox::warning(this, "Ошибка",
                "Не удалось загрузить изображение.\n"
                "Проверьте формат файла.");
            return;
        }

        // Отображаем изображение
        imageLabel->setPixmap(pixmap);
        currentImagePath = fileName;

        // Обновляем заголовок окна
        setWindowTitle("Просмотр: " + QFile::fileName(fileName));

        qDebug() << "Изображение загружено:" << fileName;
    }
}

void ImageViewer::toggleCenter(bool checked) {
    if (checked) {
        imageLabel->setAlignment(Qt::AlignCenter);
        qDebug() << "Выравнивание: по центру";
    } else {
        imageLabel->setAlignment(Qt::AlignTop | Qt::AlignLeft);
    }
}

```

```

        qDebug() << "Выравнивание: в левом верхнем углу";
    }
}

void ImageViewer::toggleStretch(bool checked) {
    imageLabel->setScaledContents(checked);
    qDebug() << "Растягивание:" << (checked ? "включено" : "выключено");
}

```

Шаг 3: Запустите и протестируйте

1. Загрузите изображение (Файл → Открыть)
2. Поэкспериментируйте с настройками вида
3. Проверьте работу горячих клавиш

Задание 3: История последних файлов

Задача:

Добавьте в меню "Файл" пункт для отображения 3 последних открытых изображений.

Модифицируйте ImageViewer:

В imageviewer.h добавьте:

```

cpp
private:
    QList<QString> recentFiles; // Список последних файлов
    QAction* recentSeparator; // Разделитель в меню
    QList<QAction*> recentActions; // Действия для последних файлов

    void updateRecentFiles(const QString& filePath);
    void openRecentFile();

```

В imageviewer.cpp добавьте:

```

cpp
// В конструкторе, после setupMenu():
recentSeparator = fileMenu->addSeparator();
recentSeparator->setVisible(false);

// Обновляем функцию openImage():
void ImageViewer::openImage() {
    QString fileName = QFileDialog::getOpenFileName(this,
        "Выберите изображение",
        "",
        "Изображения (*.png *.jpg *.jpeg *.bmp *.gif)");

    if (!fileName.isEmpty()) {
        QPixmap pixmap(fileName);

        if (pixmap.isNull()) {
            QMessageBox::warning(this, "Ошибка",
                "Не удалось загрузить изображение.");
            return;
        }

        imageLabel->setPixmap(pixmap);
        currentImagePath = fileName;
        setWindowTitle("Просмотр: " + QFileInfo(fileName).fileName());
    }
}

```

```

        // Обновляем историю
        updateRecentFiles(fileName);
    }
}

void ImageViewer::updateRecentFiles(const QString& filePath) {
    // Удаляем файл, если он уже есть в списке
    recentFiles.removeAll(filePath);

    // Добавляем в начало
    recentFiles.prepend(filePath);

    // Ограничиваем список 3 элементами
    while (recentFiles.size() > 3) {
        recentFiles.removeLast();
    }

    // Обновляем меню
    updateRecentMenu();
}

void ImageViewer::updateRecentMenu() {
    // Удаляем старые действия
    for (QAction* action : recentActions) {
        fileMenu->removeAction(action);
        delete action;
    }
    recentActions.clear();

    // Добавляем новые действия
    for (const QString& filePath : recentFiles) {
        QString displayName = QFileInfo(filePath).fileName();
        QAction* recentAction = new QAction(displayName, this);
        recentAction->setData(filePath); // Сохраняем полный путь

        connect(recentAction, &QAction::triggered, this, [this, filePath]() {
            // Загружаем выбранный файл
            QPixmap pixmap(filePath);
            if (!pixmap.isNull()) {
                imageLabel->setPixmap(pixmap);
                currentImagePath = filePath;
                setWindowTitle("Просмотр: " + QFileInfo(filePath).fileName());
            }
        });

        fileMenu->insertAction(recentSeparator, recentAction);
        recentActions.append(recentAction);
    }

    // Показываем разделитель, если есть история
    recentSeparator->setVisible(!recentFiles.isEmpty());
}

```

Задание 4: Динамическое меню для текста

Задача:

Добавьте возможность сохранять выделенный текст в меню и вставлять его обратно.

Добавьте в **imageviewer.h**:

cpp

private slots:

```
void saveSelectionToMenu();  
void insertFromMenu(QAction* action);
```

private:

```
QMenu* textMenu;  
QList<QString> savedTexts;  
void setupTextMenu();
```

Добавьте в **imageviewer.cpp**:

cpp

// В setupMenu() добавьте:

```
textMenu = menuBar()->addMenu("&Текст");  
QAction* saveAction = textMenu->addAction("Сохранить выделение");  
connect(saveAction, &QAction::triggered, this, &ImageViewer::saveSelectionToMenu);
```

// Разделитель

```
textMenu->addSeparator();
```

```
void ImageViewer::saveSelectionToMenu() {
```

// В реальном приложении здесь был бы текст из QTextEdit

// Для примеси используем фиктивный текст

```
QString sampleText = "Текст " + QString::number(savedTexts.size() + 1);
```

// Сохраняем текст

```
savedTexts.append(sampleText);
```

// Создаем новый пункт меню

```
QAction* textAction = new QAction(sampleText, this);
```

```
connect(textAction, &QAction::triggered, this,  
[this, sampleText]() { insertFromMenu(sampleText); });
```

```
textMenu->addAction(textAction);
```

```
qDebug() << "Текст сохранен в меню:" << sampleText;
```

```
}
```

```
void ImageViewer::insertFromMenu(const QString& text) {
```

// В реальном приложении вставляем текст в QTextEdit

```
qDebug() << "Вставлен текст из меню:" << text;
```

```
QMessageBox::information(this, "Вставка",
```

```
"Текст '" + text + "' готов к вставке");
```

```
}
```

Варианты заданий для самостоятельной работы

Вариант 1 (базовый):

1. Реализуйте контекстное меню из Задания 1
2. Добавьте в меню пункт "Изменить цвет фона" с выбором из 3 цветов
3. Реализуйте сохранение цвета фона между запусками программы

Вариант 2 (продвинутый):

1. Реализуйте просмотрщик изображений из Задания 2
2. Добавьте кнопки "Следующее" и "Предыдущее" для навигации по папке
3. Реализуйте слайд-шоу с автоматической сменой изображений каждые 5 секунд

Критерии оценивания

Удовлетворительно (3):

- Реализовано контекстное меню с 3 пунктами
- Программа компилируется без ошибок
- Все пункты меню работают корректно

Хорошо (4):

- Реализован просмотрщик изображений
- Добавлена история последних файлов (3 файла)
- Реализованы настройки отображения
- Код хорошо структурирован и прокомментирован

Отлично (5):

- Реализованы все задания
- Добавлен дополнительный функционал (из вариантов)
- Реализовано сохранение настроек между запусками
- Программа имеет профессиональный вид и интерфейс

Советы для выполнения работы

1. **Начинайте с простого:** Сначала создайте базовое меню, потом добавляйте функциональность.
2. **Используйте отладку:** qDebug() поможет понять, что происходит в программе.
3. **Проверяйте ошибки:** Всегда проверяйте результат операций (isNull()) для изображений).
4. **Тестируйте постепенно:** После каждого изменения проверяйте работу программы.
5. **Комментируйте код:** Пишите комментарии для сложных мест.

Пример итоговой программы:

```
сpp
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    ImageViewer viewer;
    viewer.show();

    return app.exec();
}
```

Что должно быть в отчёте:

1. Скриншоты работающей программы
2. Листинги всех файлов (.h и .cpp)
3. Описание проблем и их решений
4. Ответы на вопросы:
 - Что такое невидимые компоненты?
 - Как работает механизм сигналов и слотов?
 - Как организовать историю действий в приложении?

ПРАКТИЧЕСКАЯ РАБОТА №19-20. Разработка игрового приложения.

Цель и задачи

Цель работы: Освоить базовые принципы создания графических приложений с использованием фреймворка Qt, реализовать простое игровое приложение с графическим интерфейсом и обработкой пользовательского ввода.

Задачи:

1. Создать графическое окно приложения с использованием виджета QGraphicsView
2. Реализовать отрисовку игровых объектов на сцене QGraphicsScene
3. Организовать обработку событий клавиатуры для управления объектами
4. Настроить игровой цикл с использованием таймера QTimer
5. Создать базовый класс игрового объекта с возможностью перемещения

Теоретический материал

Графическая система Qt:

- **QGraphicsView** - виджет для отображения графической сцены
- **QGraphicsScene** - контейнер для графических элементов, управляет их расположением и отрисовкой
- **QGraphicsItem** - базовый класс для всех графических объектов на сцене
- **QGraphicsPixmapItem** - элемент для отображения растровых изображений

Игровой цикл:

- Класс **QTimer** обеспечивает периодическое обновление игры
- Сигнально-слотовая система Qt позволяет связывать события таймера с функциями обновления
- Стандартная частота обновления - 60 FPS (интервал 16-17 мс)

Обработка ввода:

- События клавиатуры обрабатываются через переопределение метода `keyPressEvent()`
- Для непрерывного движения используется флаговая система состояний клавиш
- Координатная система сцены: точка (0,0) в левом верхнем углу

Пример решения

сpp

// main.cpp

```
#include <QApplication>
#include "gamewindow.h"
```

```
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    GameWindow window;
    window.resize(800, 600);
    window.setWindowTitle("Игровое приложение - Практическая работа №19");
    window.show();
    return app.exec();
}
```

// gamewindow.h

```
#ifndef GAMEWINDOW_H
#define GAMEWINDOW_H
```

```
#include <QMainWindow>
#include <QGraphicsView>
#include <QGraphicsScene>
#include <QTimer>
#include <QKeyEvent>
```

```

class PlayerObject;

class GameWindow : public QMainWindow {
    Q_OBJECT
public:
    explicit GameWindow(QWidget *parent = nullptr);
    void keyPressEvent(QKeyEvent *event) override;
    void keyReleaseEvent(QKeyEvent *event) override;

private slots:
    void updateGame();

private:
    QGraphicsView *view;
    QGraphicsScene *scene;
    QTimer *gameTimer;
    PlayerObject *player;
    QSet<int> pressedKeys;
};

#endif

// playerobject.h
#ifndef PLAYEROBJECT_H
#define PLAYEROBJECT_H

#include <QGraphicsPixmapItem>
#include <QObject>

class PlayerObject : public QObject, public QGraphicsPixmapItem {
    Q_OBJECT
public:
    PlayerObject(QGraphicsItem *parent = nullptr);
    void updatePosition(const QSet<int> &pressedKeys);

private:
    int speed = 5;
};

#endif

// gamewindow.cpp
#include "gamewindow.h"
#include "playerobject.h"
#include <QVBoxLayout>

GameWindow::GameWindow(QWidget *parent) : QMainWindow(parent) {
    // Создание центрального виджета
    QWidget *centralWidget = new QWidget(this);
    setCentralWidget(centralWidget);

    // Настройка сцены и представления

```

```

scene = new QGraphicsScene(this);
scene->setSceneRect(0, 0, 800, 600);
scene->setBackgroundBrush(Qt::black);

view = new QGraphicsView(scene, centralWidget);
view->setRenderHint(QPainter::Antialiasing);
view->setViewportUpdateMode(QGraphicsView::BoundingRectViewportUpdate);

// Создание игрока
player = new PlayerObject();
player->setPos(400, 300);
scene->addItem(player);

// Настройка таймера игрового цикла
gameTimer = new QTimer(this);
connect(gameTimer, &QTimer::timeout, this, &GameWindow::updateGame);
gameTimer->start(16); // ~60 FPS

// Компоновка
QVBoxLayout *layout = new QVBoxLayout(centralWidget);
layout->addWidget(view);
}

void GameWindow::keyPressEvent(QKeyEvent *event) {
    pressedKeys.insert(event->key());
    QMainWindow::keyPressEvent(event);
}

void GameWindow::keyReleaseEvent(QKeyEvent *event) {
    pressedKeys.remove(event->key());
    QMainWindow::keyReleaseEvent(event);
}

void GameWindow::updateGame() {
    player->updatePosition(pressedKeys);
}

// playerobject.cpp
#include "playerobject.h"
#include <QPainter>
#include <QKeyEvent>

PlayerObject::PlayerObject(QGraphicsItem *parent)
    : QGraphicsPixmapItem(parent) {
    // Создание простого графического представления
    QPixmap pixmap(50, 50);
    pixmap.fill(Qt::transparent);

    QPainter painter(&pixmap);
    painter.setBrush(Qt::blue);
    painter.setPen(QPen(Qt::white, 2));
    painter.drawEllipse(5, 5, 40, 40);
}

```

```

painter.setBrush(Qt::white);
painter.drawEllipse(15, 15, 10, 10);

setPixmap(pixmap);
}

void PlayerObject::updatePosition(const QSet<int> &pressedKeys) {
    QPointF newPos = pos();

    if (pressedKeys.contains(Qt::Key_Left)) {
        newPos.setX(newPos.x() - speed);
    }
    if (pressedKeys.contains(Qt::Key_Right)) {
        newPos.setX(newPos.x() + speed);
    }
    if (pressedKeys.contains(Qt::Key_Up)) {
        newPos.setY(newPos.y() - speed);
    }
    if (pressedKeys.contains(Qt::Key_Down)) {
        newPos.setY(newPos.y() + speed);
    }

    // Ограничение движения границами сцены
    QRectF sceneRect = scene()->sceneRect();
    QRectF itemRect = boundingRect();

    qreal left = sceneRect.left();
    qreal right = sceneRect.right() - itemRect.width();
    qreal top = sceneRect.top();
    qreal bottom = sceneRect.bottom() - itemRect.height();

    newPos.setX(qMax(left, qMin(newPos.x(), right)));
    newPos.setY(qMax(top, qMin(newPos.y(), bottom)));

    setPos(newPos);
}

```

Задания по вариантам

Вариант 1: "Космический корабль"

1. Создайте игровое окно размером 1024×768 пикселей
2. Реализуйте управление космическим кораблем с помощью клавиш WASD
3. Добавьте астероиды (3-5 штук) как статические препятствия
4. Реализуйте систему столкновений: при столкновении с астероидом корабль останавливается
5. Добавьте счетчик пройденного расстояния в пикселях
6. (Дополнительно) Добавьте анимацию движения корабля

Вариант 2: "Сборщик монет"

1. Создайте игровое окно размером 800×600 пикселей
2. Реализуйте управление персонажем с помощью стрелок клавиатуры
3. Создайте 10 монет, случайно размещенных на игровом поле
4. Реализуйте механику сбора монет (при наложении объектов)
5. Добавьте счетчик собранных монет и таймер игры (60 секунд)
6. (Дополнительно) Добавьте звуковые эффекты при сборе монет

Критерии оценивания

Критерий	Баллы	Требования
Создание окна	0-2	0 - окно не создано; 1 - окно создано без настроек; 2 - окно с правильными размерами и заголовком
Отрисовка объектов	0-3	0 - объекты не отрисовываются; 1-2 - объекты отрисовываются с ошибками; 3 - все объекты отрисованы корректно
Управление	0-3	0 - управление не работает; 1-2 - управление работает с ошибками; 3 - плавное управление всеми клавишами
Игровая логика	0-4	0 - логика не реализована; 1-2 - реализована частично; 3-4 - полная реализация по варианту
Качество кода	0-3	0 - код не структурирован; 1-2 - код с комментариями, но есть ошибки; 3 - чистый, комментированный код

Шкала оценок:

- **5 (отлично):** 13-15 баллов
- **4 (хорошо):** 10-12 баллов
- **3 (удовлетворительно):** 7-9 баллов
- **2 (неудовлетворительно):** 0-6 баллов

ПРАКТИЧЕСКАЯ РАБОТА №21. Разработка приложения с анимацией.

Цель и задачи

Цель работы: Освоить различные подходы к реализации анимации в Qt, научиться создавать плавные движения графических объектов и управлять их поведением с помощью таймеров и систем анимации Qt.

Задачи:

1. Изучить базовые принципы анимации в Qt
2. Реализовать движение объектов с использованием QTimer
3. Освоить систему QPropertyAnimation для плавных переходов
4. Научиться работать с прозрачностью графических объектов
5. Создать анимацию встречного движения автомобиля и мотоцикла
6. Оптимизировать отрисовку для устранения мерцания

Теоретический материал

Основные подходы к анимации в Qt

1. Анимация с использованием QTimer

QTimer позволяет создавать кадровую анимацию, периодически обновляя состояние объектов:

```
сpp
QTimer *timer = new QTimer(this);
connect(timer, &QTimer::timeout, this, &MyClass::updateAnimation);
timer->start(16); // 60 FPS
```

2. Анимация с использованием QPropertyAnimation

Более современный подход, основанный на изменении свойств объектов:

```
сpp
QPropertyAnimation *animation = new QPropertyAnimation(object, "pos");
animation->setDuration(2000); // 2 секунды
animation->setStartValue(QPoint(0, 0));
animation->setEndValue(QPoint(100, 100));
animation->start();
```

3. Система анимации Qt (QAnimationGroup)

Позволяет объединять несколько анимаций:

- **QSequentialAnimationGroup** - последовательное выполнение
- **QParallelAnimationGroup** - параллельное выполнение

4. Работа с графикой

- **QPixmap** - для загрузки и отображения изображений
- **QPainter** - для рисования на виджетах
- **Transparency (прозрачность)** - через QPainter::CompositionMode или PNG с альфа-каналом

Оптимизация анимации

- **Двойная буферизация** - предотвращение мерцания
- **Обновление только изменяющихся областей** - через update(QRect)
- **Использование аппаратного ускорения** - QGraphicsView с OpenGL

Пример решения

Способ 1: Простая анимация с QTimer

```
сpp
// mainwindow.h
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTimer>
#include <QPixmap>
```

```

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow {
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

protected:
    void paintEvent(QPaintEvent *event) override;

private slots:
    void updateAnimation();

private:
    Ui::MainWindow *ui;
    QTimer *animationTimer;

    // Графические ресурсы
    QPixmap background;
    QPixmap car;
    QPixmap motorcycle;

    // Позиции объектов
    int carX, carY;
    int motoX, motoY;

    // Константы
    const int SCREEN_WIDTH = 800;
    const int SCREEN_HEIGHT = 600;
    const int ROAD_Y = 450;
    const int SPEED = 2;
};

#endif // MAINWINDOW_H

// mainwindow.cpp
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QPainter>
#include <QRandomGenerator>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow) {
    ui->setupUi(this);

    // Установка размера окна
    setFixedSize(SCREEN_WIDTH, SCREEN_HEIGHT);

```

```

setTitle("Анимация движения транспорта");

// Загрузка изображений
background.load(":/images/fon.png");
car.load(":/images/avto.png");
motorcycle.load(":/images/moto.png");

// Установка прозрачности для PNG (если используется)
// Для BMP потребуется установить цвет прозрачности
if (car.hasAlphaChannel()) {
    car = car.scaled(100, 50, Qt::KeepAspectRatio, Qt::SmoothTransformation);
}
if (motorcycle.hasAlphaChannel()) {
    motorcycle = motorcycle.scaled(80, 40, Qt::KeepAspectRatio, Qt::SmoothTransformation);
}

// Начальные позиции
carX = -car.width(); // Начинаем за левой границей
carY = ROAD_Y;

motoX = SCREEN_WIDTH; // Начинаем за правой границей
motoY = ROAD_Y - 20; // Чуть выше автомобиля

// Настройка таймера анимации
animationTimer = new QTimer(this);
connect(animationTimer, &QTimer::timeout, this, &MainWindow::updateAnimation);
animationTimer->start(16); // ~60 FPS
}

MainWindow::~MainWindow() {
    delete ui;
}

void MainWindow::paintEvent(QPaintEvent *event) {
    Q_UNUSED(event);

    QPainter painter(this);

    // Отрисовка фона
    painter.drawPixmap(0, 0, background.scaled(size(), Qt::IgnoreAspectRatio));

    // Отрисовка объектов с прозрачностью
    painter.setCompositionMode(QPainter::CompositionMode_SourceOver);
    painter.drawPixmap(carX, carY, car);
    painter.drawPixmap(motoX, motoY, motorcycle);
}

void MainWindow::updateAnimation() {
    // Движение автомобиля (вправо)
    carX += SPEED;
    if (carX > SCREEN_WIDTH) {
        carX = -car.width();
    }
}

```

```

    }

    // Движение мотоцикла (влево)
    motoX -= SPEED * 1.2; // Мотоцикл немного быстрее
    if (motoX < -motorcycle.width()) {
        motoX = SCREEN_WIDTH;
    }

    // Обновление экрана
    update();
}

```

Способ 2: Анимация с QPropertyAnimation (более современный подход)

cpp

```

// animatedvehicle.h
#ifndef ANIMATEDVEHICLE_H
#define ANIMATEDVEHICLE_H

#include <QWidget>
#include <QPixmap>
#include <QPropertyAnimation>
#include <QParallelAnimationGroup>

class AnimatedVehicle : public QWidget {
    Q_OBJECT
    Q_PROPERTY(QPoint position READ position WRITE setPosition)

public:
    enum VehicleType { Car, Motorcycle };

    explicit AnimatedVehicle(VehicleType type, QWidget *parent = nullptr);
    QPoint position() const;
    void setPosition(const QPoint &pos);

protected:
    void paintEvent(QPaintEvent *event) override;

private:
    VehicleType type;
    QPixmap vehiclePixmap;
    QPoint currentPosition;
    QPropertyAnimation *animation;

    void loadImage();
};

#endif // ANIMATEDVEHICLE_H

```

```

// trafficanimationwidget.h
#ifndef TRAFFICANIMATIONWIDGET_H
#define TRAFFICANIMATIONWIDGET_H

#include <QWidget>

```

```

#include <QTimer>
#include <QParallelAnimationGroup>
#include "animatedvehicle.h"

class TrafficAnimationWidget : public QWidget {
    Q_OBJECT

public:
    explicit TrafficAnimationWidget(QWidget *parent = nullptr);

protected:
    void paintEvent(QPaintEvent *event) override;
    void resizeEvent(QResizeEvent *event) override;

private slots:
    void restartAnimation();

private:
    QPixmap background;
    AnimatedVehicle *car;
    AnimatedVehicle *motorcycle;
    QParallelAnimationGroup *animationGroup;

    void setupAnimations();
};

#endif // TRAFFICANIMATIONWIDGET_H

// trafficanimationwidget.cpp
#include "trafficanimationwidget.h"
#include <QPainter>
#include <QResizeEvent>

TrafficAnimationWidget::TrafficAnimationWidget(QWidget *parent)
    : QWidget(parent) {

    // Загрузка фона
    background.load(":/images/fon.png");

    // Создание транспортных средств
    car = new AnimatedVehicle(AnimatedVehicle::Car, this);
    motorcycle = new AnimatedVehicle(AnimatedVehicle::Motorcycle, this);

    // Настройка начальных позиций
    int roadY = height() * 0.75;
    car->move(-100, roadY);
    motorcycle->move(width(), roadY - 20);

    // Создание группы анимаций
    animationGroup = new QParallelAnimationGroup(this);
    setupAnimations();
}

```

```

// Запуск анимации
animationGroup->start();

// Перезапуск анимации при завершении
connect(animationGroup, &QParallelAnimationGroup::finished,
        this, &TrafficAnimationWidget::restartAnimation);
}

void TrafficAnimationWidget::paintEvent(QPaintEvent *event) {
    Q_UNUSED(event);

    QPainter painter(this);

    // Отрисовка фона
    if (!background.isNull()) {
        painter.drawPixmap(0, 0, width(), height(), background);
    } else {
        // Запасной фон, если изображение не загружено
        painter.fillRect(rect(), QColor(135, 206, 235)); // Небо
        painter.fillRect(0, height() * 0.75, width(), height() * 0.25,
                        QColor(100, 100, 100)); // Дорога
    }
}

void TrafficAnimationWidget::resizeEvent(QResizeEvent *event) {
    QWidget::resizeEvent(event);
    restartAnimation();
}

void TrafficAnimationWidget::setupAnimations() {
    // Очистка старых анимаций
    animationGroup->clear();

    // Анимация автомобиля (слева направо)
    QPropertyAnimation *carAnimation = new QPropertyAnimation(car, "position");
    carAnimation->setDuration(5000); // 5 секунд
    carAnimation->setStartValue(QPoint(-100, height() * 0.75));
    carAnimation->setEndValue(QPoint(width() + 100, height() * 0.75));
    carAnimation->setEasingCurve(QEasingCurve::Linear);

    // Анимация мотоцикла (справа налево)
    QPropertyAnimation *motoAnimation = new QPropertyAnimation(motorcycle, "position");
    motoAnimation->setDuration(4000); // 4 секунды (быстрее)
    motoAnimation->setStartValue(QPoint(width() + 100, height() * 0.75 - 20));
    motoAnimation->setEndValue(QPoint(-100, height() * 0.75 - 20));
    motoAnimation->setEasingCurve(QEasingCurve::Linear);

    // Добавление анимаций в группу
    animationGroup->addAnimation(carAnimation);
    animationGroup->addAnimation(motoAnimation);
}

```

```

void TrafficAnimationWidget::restartAnimation() {
    // Сброс позиций
    car->move(-100, height() * 0.75);
    motorcycle->move(width() + 100, height() * 0.75 - 20);

    // Перезапуск анимации
    if (animationGroup->state() == QAbstractAnimation::Running) {
        animationGroup->stop();
    }
    setupAnimations();
    animationGroup->start();
}

```

Способ 3: Оптимизированная анимация с двойной буферизацией

cpp

```

// bufferedanimationwidget.h
#ifndef BUFFEREDANIMATIONWIDGET_H
#define BUFFEREDANIMATIONWIDGET_H

#include <QWidget>
#include <QTimer>
#include <QPixmap>

class BufferedAnimationWidget : public QWidget {
    Q_OBJECT

public:
    explicit BufferedAnimationWidget(QWidget *parent = nullptr);
    ~BufferedAnimationWidget();

protected:
    void paintEvent(QPaintEvent *event) override;
    void resizeEvent(QResizeEvent *event) override;

private slots:
    void updateFrame();

private:
    void initializeBuffers();
    void renderFrame();

    QTimer *frameTimer;

    // Графические ресурсы
    QPixmap backgroundBuffer;
    QPixmap carBuffer;
    QPixmap motorcycleBuffer;
    QPixmap frameBuffer; // Буфер для двойной буферизации

    // Позиции и состояния
    struct Vehicle {
        QPixmap *sprite;
        int x, y;
    };

```

```

    int speed;
    int width, height;
};

Vehicle car;
Vehicle motorcycle;

// Константы
const int ANIMATION_FPS = 60;
const int ROAD_LEVEL = 400;
};

#endif // BUFFEREDANIMATIONWIDGET_H

// bufferedanimationwidget.cpp
#include "bufferedanimationwidget.h"
#include <QPainter>
#include <QApplication>

BufferedAnimationWidget::BufferedAnimationWidget(QWidget *parent)
: QWidget(parent) {

    // Включение двойной буферизации на уровне виджета
    setAttribute(Qt::WA_OpaquePaintEvent);
    setAttribute(Qt::WA_NoSystemBackground);

    // Настройка размера
    setFixedSize(800, 600);

    // Инициализация буферов
    initializeBuffers();

    // Инициализация транспортных средств
    car.sprite = &carBuffer;
    car.x = -carBuffer.width();
    car.y = ROAD_LEVEL;
    car.speed = 3;
    car.width = carBuffer.width();
    car.height = carBuffer.height();

    motorcycle.sprite = &motorcycleBuffer;
    motorcycle.x = width() + motorcycleBuffer.width();
    motorcycle.y = ROAD_LEVEL - 30;
    motorcycle.speed = 4;
    motorcycle.width = motorcycleBuffer.width();
    motorcycle.height = motorcycleBuffer.height();

    // Настройка таймера анимации
    frameTimer = new QTimer(this);
    connect(frameTimer, &QTimer::timeout, this, &BufferedAnimationWidget::updateFrame);
    frameTimer->start(1000 / ANIMATION_FPS);
}

```

```

BufferedAnimationWidget::~BufferedAnimationWidget() {
    frameTimer->stop();
}

void BufferedAnimationWidget::initializeBuffers() {
    // Загрузка и подготовка изображений
    QPixmap rawBackground(":/images/fon.png");
    if (!rawBackground.isNull()) {
        backgroundBuffer = rawBackground.scaled(size(),
            Qt::IgnoreAspectRatio, Qt::SmoothTransformation);
    }

    QPixmap rawCar(":/images/avto.png");
    if (!rawCar.isNull()) {
        carBuffer = rawCar.scaled(120, 60,
            Qt::KeepAspectRatio, Qt::SmoothTransformation);
    }

    QPixmap rawMoto(":/images/moto.png");
    if (!rawMoto.isNull()) {
        motorcycleBuffer = rawMoto.scaled(90, 50,
            Qt::KeepAspectRatio, Qt::SmoothTransformation);
    }

    // Создание буфера кадра
    frameBuffer = QPixmap(size());
    frameBuffer.fill(Qt::transparent);
}

void BufferedAnimationWidget::paintEvent(QPaintEvent *event) {
    Q_UNUSED(event);

    QPainter painter(this);
    painter.drawPixmap(0, 0, frameBuffer);
}

void BufferedAnimationWidget::resizeEvent(QResizeEvent *event) {
    QWidget::resizeEvent(event);
    initializeBuffers();
}

void BufferedAnimationWidget::updateFrame() {
    // Обновление позиций
    car.x += car.speed;
    motorcycle.x -= motorcycle.speed;

    // Проверка границ
    if (car.x > width()) {
        car.x = -car.width;
    }
    if (motorcycle.x < -motorcycle.width) {

```

```

        motorcycle.x = width();
    }

    // Рендеринг кадра в буфер
    renderFrame();

    // Запрос на перерисовку
    update();
}

void BufferedAnimationWidget::renderFrame() {
    // Создание painter для буфера кадра
    QPainter painter(&frameBuffer);
    painter.setRenderHint(QPainter::Antialiasing);
    painter.setRenderHint(QPainter::SmoothPixmapTransform);

    // Очистка буфера
    painter.fillRect(rect(), Qt::black);

    // Отрисовка фона
    if (!backgroundBuffer.isNull()) {
        painter.drawPixmap(0, 0, backgroundBuffer);
    }

    // Отрисовка транспортных средств с прозрачностью
    painter.setCompositionMode(QPainter::CompositionMode_SourceOver);

    if (!carBuffer.isNull()) {
        painter.drawPixmap(car.x, car.y, carBuffer);
    }

    if (!motorcycleBuffer.isNull()) {
        painter.drawPixmap(motorcycle.x, motorcycle.y, motorcycleBuffer);
    }

    // Дополнительно: отрисовка теней
    painter.setCompositionMode(QPainter::CompositionMode_Multiply);
    QPixmap shadow = carBuffer;
    QPainter shadowPainter(&shadow);
    shadowPainter.setCompositionMode(QPainter::CompositionMode_DestinationIn);
    shadowPainter.fillRect(shadow.rect(), QColor(0, 0, 0, 100));
    painter.drawPixmap(car.x + 3, car.y + 3, shadow);
}

```

Задания по вариантам

Вариант 1: "Городское движение с препятствиями"

Задание: Создайте анимацию движения автомобиля по городской улице с препятствиями.

Требования:

1. Реализуйте плавное движение автомобиля слева направо
2. Добавьте 3-5 статических препятствий (другие припаркованные автомобили)
3. Реализуйте систему объезда препятствий:
 - При приближении к препятствию автомобиль должен плавно смещаться вверх

- После проезда препятствия возвращаться на свою полосу
- 4. Добавьте анимацию вращающихся колес автомобиля
- 5. Реализуйте возможность паузы/возобновления анимации по нажатию пробела
- 6. Добавьте счетчик успешно преодоленных препятствий

Дополнительные возможности:

- Разные типы препятствий с различной сложностью объезда
- Система уровней с увеличивающейся скоростью
- Анимация светофора с изменяющимися сигналами
- Звуковые эффекты движения и сигналов

Вариант 2: "Гонка на встречных полосах"

Задание: Создайте анимацию встречного движения нескольких транспортных средств с элементами игры.

Требования:

1. Реализуйте 3 полосы движения (в каждую сторону)
2. Добавьте управляемый игроком автомобиль (управление стрелками)
3. Создайте встречный поток из 5-7 различных транспортных средств
4. Реализуйте систему столкновений:
 - При столкновении игра останавливается
 - Отображение сообщения о результате
5. Добавьте систему набора очков за успешное избегание столкновений
6. Реализуйте увеличивающуюся сложность (скорость встречных машин)

Дополнительные возможности:

- Разные типы транспортных средств с уникальной графикой
- Система бонусов (ускорение, временная неуязвимость)
- Анимация взрыва при столкновении
- Таблица рекордов с сохранением результатов
- Выбор сложности (легкий, средний, сложный)

Критерии оценивания

Базовые критерии (максимум 20 баллов)

Критерий	Баллы	Описание требований
Создание интерфейса	0-3	0 - интерфейс не создан; 1 - базовый интерфейс; 2 - интерфейс с настройками; 3 - профессиональный интерфейс с элементами управления
Загрузка и отображение графики	0-4	0 - графика не загружена; 1-2 - графические ресурсы загружены частично; 3-4 - все ресурсы загружены корректно, настроена прозрачность
Реализация анимации	0-5	0 - анимация отсутствует; 1-2 - простое движение объектов; 3-4 - плавная анимация с QPropertyAnimation; 5 - сложная анимация с несколькими объектами и состояниями
Управление анимацией	0-3	0 - управление отсутствует; 1 - базовое управление (старт/стоп); 2 - управление скоростью; 3 - полный контроль над анимацией
Оптимизация	0-3	0 - мерцание при анимации; 1 - устранено мерцание; 2 - оптимизирована производительность; 3 - использована двойная буферизация и частичное

Критерий	Баллы	Описание требований
		обновление
Качество кода	0-2	0 - код не структурирован; 1 - код с комментариями; 2 - чистый, хорошо организованный код

Дополнительные критерии для вариантов

Для Варианта 1:

- **Объезд препятствий:** 0-2 балла
- **Анимация колес:** 0-2 балла
- **Система паузы:** 0-1 балл
- **Счетчик препятствий:** 0-1 балл

Для Варианта 2:

- **Управление игроком:** 0-2 балла
- **Система столкновений:** 0-2 балла
- **Система очков:** 0-1 балл
- **Увеличение сложности:** 0-1 балл

Шкала оценивания

Оценка	Баллы (базовые)	Требования
5 (отлично)	18-20	Полная реализация всех требований, отличная оптимизация, чистый код с комментариями
4 (хорошо)	15-17	Реализация всех основных требований, хорошая оптимизация, понятный код
3 (удовлетворительно)	12-14	Реализация базовых требований, анимация работает с незначительными проблемами
2 (неудовлетворительно)	0-11	Не выполнены основные требования, анимация не работает или работает некорректно

Советы по выполнению работы

Подготовительный этап:

1. Создайте структуру проекта:

text

project/

```

|— src/
|   |— main.cpp
|   |— mainwindow.h
|   |— mainwindow.cpp
|— resources/
|   |— images/
|       |— fon.png
|       |— avto.png
|       |— moto.png

```

```
└─ resources.qrc
└─ project.pro
```

2. Настройте файл .pro:

```
pro
QT += core gui widgets
TARGET = TrafficAnimation
TEMPLATE = app
SOURCES += src/main.cpp \
           src/mainwindow.cpp
HEADERS += src/mainwindow.h
RESOURCES += resources/resources.qrc
```

3. Создайте файл ресурсов:

```
xml
<RCC>
  <qresource prefix="/images">
    <file>fon.png</file>
    <file>avto.png</file>
    <file>moto.png</file>
  </qresource>
</RCC>
```

Технические рекомендации:

1. Для предотвращения мерцания:

```
cpp
// В конструкторе виджета:
setAttribute(Qt::WA_OpaquePaintEvent);
setAttribute(Qt::WA_NoSystemBackground);
```

// В paintEvent:

```
QPainter painter(this);
painter.setRenderHint(QPainter::Antialiasing);
```

2. Для плавной анимации:

```
cpp
// Используйте QPropertyAnimation вместо ручного управления таймером
QPropertyAnimation *anim = new QPropertyAnimation(object, "geometry");
anim->setDuration(1000);
anim->setEasingCurve(QEasingCurve::InOutQuad);
```

3. Для работы с прозрачностью:

```
cpp
// Установка режима композиции
painter.setCompositionMode(QPainter::CompositionMode_SourceOver);
```

// Загрузка PNG с альфа-каналом

```
QPixmap pixmap(":/image.png");
if (pixmap.hasAlphaChannel()) {
    // Изображение уже содержит информацию о прозрачности
}
```

Отладка и тестирование:

1. Используйте qDebug() для отслеживания позиций объектов:

```
cpp
qDebug() << "Car position:" << carX << carY;
```

2. Проверяйте загрузку ресурсов:

```
cpp
```

```

if (pixmap.isNull()) {
    qWarning() << "Failed to load image:" << path;
}

```

3. Тестируйте на разных FPS:

```

cpp
// Экспериментируйте с интервалами таймера
timer->start(33); // 30 FPS
timer->start(16); // 60 FPS
timer->start(8); // 120 FPS

```

Расширенные возможности (для дополнительных баллов):

1. Добавление эффектов:

```

cpp
// Тени
QGraphicsDropShadowEffect *shadow = new QGraphicsDropShadowEffect;
shadow->setBlurRadius(10);
shadow->setOffset(3, 3);
object->setGraphicsEffect(shadow);

```

2. Анимация преобразований:

```

cpp
// Вращение, масштабирование
QPropertyAnimation *rotateAnim = new QPropertyAnimation(object, "rotation");
rotateAnim->setStartValue(0);
rotateAnim->setEndValue(360);

```

3. Интерактивность:

```

cpp
// Обработка кликов по объектам
void mousePressEvent(QMouseEvent *event) {
    if (object->contains(event->pos())) {
        // Действие при клике на объект
    }
}

```

Данная практическая работа поможет освоить ключевые аспекты создания анимации в Qt, что является важным навыком для разработки игр и интерактивных приложений.

ПРАКТИЧЕСКАЯ РАБОТА №22-24. Оптимизация.

Цель и задачи

Цель работы: Освоить методы оптимизации программного кода на C++, научиться анализировать и улучшать эффективность программ с использованием метрик Холстеда.

Задачи:

1. Изучить принципы рефакторинга и оптимизации кода
2. Научиться модифицировать существующий код для улучшения его читаемости и производительности
3. Освоить расчет метрик Холстеда для оценки сложности программ
4. Научиться реализовывать надежный ввод данных с обработкой ошибок
5. Сравнить эффективность разных версий программы

Теоретический материал

Принципы оптимизации кода

1. Оптимизация алгоритмов

- Выбор наиболее эффективного алгоритма для решения задачи
- Снижение временной сложности (Big O notation)
- Оптимизация использования памяти

2. Рефакторинг

- Улучшение читаемости кода
- Устранение дублирования
- Разделение ответственности
- Использование функций и классов

3. Обработка ошибок и валидация

- Проверка входных данных
- Обработка исключительных ситуаций
- Предоставление понятных сообщений об ошибках

Метрики Холстеда

Метрики Холстеда позволяют количественно оценить сложность программы:

Метрика	Формула	Описание
Словарь программы (n)	$n = n_1 + n_2$	Общее число уникальных операторов и операндов
Длина программы (N)	$N = N_1 + N_2$	Общее количество операторов и операндов
Объем программы (V)	$V = N \times \log_2(n)$	Информационный объем программы в битах
Сложность (D)	$D = (n_1/2) \times (N_2/n_2)$	Оценка сложности понимания программы
Усилия (E)	$E = D \times V$	Оценка умственных усилий для создания программы

Классификация элементов программы

Тип элемента	Описание	Примеры в C++
Операторы (n1)	Ключевые слова, операторы,	if, for, while, return, +, -, *, /, =, :, {, }

Тип элемента	Описание	Примеры в C++
	разделители	
Операнды (n2)	Идентификаторы, константы, литералы	variable, 123, "text", 3.14, true

Пример решения

Программа 1: Обработка ввода с валидацией

Исходный код (до оптимизации):

```
cpp
#include <iostream>
#include <locale.h>
using namespace std;

int main()
{
    int number;
    setlocale(LC_CTYPE, "Russian");
    cout << "Введите число: ";
    cin >> number;
    cin.ignore();
    cout << "Вы ввели: "<< number << "\n";
    cin.get();
}
```

Оптимизированный код (после оптимизации):

```
cpp
#include <iostream>
#include <string>
#include <limits>
#include <cmath>
#include <locale.h>

// Функция для проверки, является ли строка числом
bool isInteger(const std::string& str) {
    if (str.empty()) return false;

    size_t start = 0;
    // Проверка на отрицательное число
    if (str[0] == '-' || str[0] == '+') {
        start = 1;
    }

    // Проверка, что все символы - цифры
    for (size_t i = start; i < str.length(); i++) {
        if (!std::isdigit(static_cast<unsigned char>(str[i]))) {
            return false;
        }
    }

    return true;
}
```

```

// Функция для безопасного ввода целого числа
bool getValidInteger(int& number, const std::string& prompt) {
    std::string input;

    while (true) {
        std::cout << prompt;
        std::getline(std::cin, input);

        // Проверка на пустой ввод
        if (input.empty()) {
            std::cout << "Ошибка: Ввод не может быть пустым.\n";
            continue;
        }

        // Проверка, является ли ввод числом
        if (!isInteger(input)) {
            std::cout << "Ошибка: Введите целое число (допустимы цифры и знак +/-).\n";
            continue;
        }

        try {
            // Преобразование строки в число
            long long temp = std::stoll(input);

            // Проверка на переполнение
            if (temp < std::numeric_limits<int>::min() ||
                temp > std::numeric_limits<int>::max()) {
                std::cout << "Ошибка: Число вне допустимого диапазона.\n";
                continue;
            }

            number = static_cast<int>(temp);
            return true;

        } catch (const std::exception& e) {
            std::cout << "Ошибка преобразования: " << e.what() << "\n";
            return false;
        }
    }
}

// Функция для вычисления квадрата числа
int calculateSquare(int number) {
    // Проверка на переполнение при возведении в квадрат
    if (number > 0 && number > std::numeric_limits<int>::max() / number) {
        throw std::overflow_error("Переполнение при вычислении квадрата");
    }
    if (number < 0 && number < std::numeric_limits<int>::min() / number) {
        throw std::overflow_error("Переполнение при вычислении квадрата");
    }
}

```

```

    return number * number;
}

// Функция для определения типа числа
std::string getNumberType(int number) {
    if (number > 0) return "положительное";
    if (number < 0) return "отрицательное";
    return "ноль";
}

int main() {
    setlocale(LC_CTYPE, "Russian");

    int number;

    std::cout << "=== Программа обработки чисел ===\n\n";

    // Получение валидного числа от пользователя
    if (!getValidInteger(number, "Введите целое число: ")) {
        std::cout << "Не удалось получить корректный ввод. Программа завершена.\n";
        return 1;
    }

    // Вывод информации о числе
    std::cout << "\n=== Результаты анализа ===\n";
    std::cout << "Вы ввели: " << number << "\n";
    std::cout << "Тип числа: " << getNumberType(number) << "\n";

    // Вычисление и вывод квадрата числа
    try {
        int square = calculateSquare(number);
        std::cout << "Квадрат числа: " << square << "\n";
    } catch (const std::exception& e) {
        std::cout << "Ошибка: " << e.what() << "\n";
    }

    // Анализ числа
    if (number % 2 == 0) {
        std::cout << "Число четное\n";
    } else {
        std::cout << "Число нечетное\n";
    }

    std::cout << "\nДля завершения нажмите Enter...";
    std::cin.get();

    return 0;
}

```

Анализ оптимизаций:

Аспект	Было	Стало	Преимущество
Обработка ввода	<code>cin >> number (без проверки)</code>	Функция <code>getValidInteger()</code> с полной валидацией	Защита от некорректного ввода
Разделение кода	Вся логика в <code>main()</code>	Разделение на функции: <code>isInteger()</code> , <code>calculateSquare()</code> , <code>getNumberType()</code>	Улучшение читаемости и переиспользования
Обработка ошибок	Нет обработки ошибок	Полная обработка исключений	Надежность программы
Проверка переполнения	Нет проверки	Проверка при вводе и вычислении квадрата	Предотвращение undefined behavior
Вывод информации	Только введенное число	Полный анализ числа	Более полезная информация для пользователя

Программа 2: Улучшенный калькулятор

Исходный код (до оптимизации):

```

cpp
#include <stdio.h>
#include <iostream>
#include <locale.h>
#include <windows.h>
using namespace std;

int main(int argc, char* argv[])
{
    setlocale(LC_CTYPE, "Russian");
    double plus, minus, pow, div;
    double a1;
    double a2;

    cout << "Введите первое число: ";
    cin >> a1;
    cout << "Введите второе число: ";
    cin >> a2;

    plus = a1 + a2;
    minus = a1 - a2;

```

```

pow = a1 * a2;
div = a1 / a2;

cout << a1 << "+" << a2 << "=" << plus << endl;
cout << a1 << "-" << a2 << "=" << minus << endl;
cout << a1 << "*" << a2 << "=" << pow << endl;
cout << a1 << "/" << a2 << "=" << div << endl;

system("pause");
return 0;
}
Оптимизированный код (после оптимизации):
cpp
#include <iostream>
#include <string>
#include <limits>
#include <cmath>
#include <stdexcept>
#include <locale.h>

// Класс для работы с числами и валидации
class NumberValidator {
public:
    // Проверка, является ли строка числом с плавающей точкой
    static bool isValidDouble(const std::string& str) {
        if (str.empty()) return false;

        bool hasDecimalPoint = false;
        bool hasExponent = false;
        bool hasDigits = false;
        size_t start = 0;

        // Проверка знака
        if (str[0] == '-' || str[0] == '+') {
            start = 1;
        }

        for (size_t i = start; i < str.length(); i++) {
            char c = str[i];

            if (std::isdigit(static_cast<unsigned char>(c))) {
                hasDigits = true;
            } else if (c == '.') {
                if (hasDecimalPoint || hasExponent) return false;
                hasDecimalPoint = true;
            } else if (c == 'e' || c == 'E') {
                if (hasExponent || !hasDigits) return false;
                hasExponent = true;
                hasDigits = false; // Сброс для проверки экспоненты
            } else if ((c == '-' || c == '+') && (str[i-1] == 'e' || str[i-1] == 'E')) {
                // Знак в экспоненте разрешен
            } else {

```

```

        return false;
    }
}

return hasDigits;
}

// Безопасное преобразование строки в double
static double parseDouble(const std::string& str) {
    try {
        return std::stod(str);
    } catch (const std::exception& e) {
        throw std::invalid_argument("Невозможно преобразовать строку в число: " + str);
    }
}
};

```

```

// Класс калькулятора с расширенной функциональностью
class AdvancedCalculator {
public:
    // Основные арифметические операции
    static double add(double a, double b) {
        // Проверка на переполнение
        if ((b > 0 && a > std::numeric_limits<double>::max() - b) ||
            (b < 0 && a < std::numeric_limits<double>::lowest() - b)) {
            throw std::overflow_error("Переполнение при сложении");
        }
        return a + b;
    }

    static double subtract(double a, double b) {
        // Проверка на переполнение
        if ((b < 0 && a > std::numeric_limits<double>::max() + b) ||
            (b > 0 && a < std::numeric_limits<double>::lowest() + b)) {
            throw std::overflow_error("Переполнение при вычитании");
        }
        return a - b;
    }

    static double multiply(double a, double b) {
        // Проверка на переполнение
        if (a > 0 && b > 0 && a > std::numeric_limits<double>::max() / b) {
            throw std::overflow_error("Переполнение при умножении");
        }
        if (a < 0 && b < 0 && a < std::numeric_limits<double>::max() / b) {
            throw std::overflow_error("Переполнение при умножении");
        }
        if (a > 0 && b < 0 && b < std::numeric_limits<double>::lowest() / a) {
            throw std::overflow_error("Переполнение при умножении");
        }
        if (a < 0 && b > 0 && a < std::numeric_limits<double>::lowest() / b) {
            throw std::overflow_error("Переполнение при умножении");
        }
    }
};

```

```

    }
    return a * b;
}

static double divide(double a, double b) {
    if (b == 0) {
        throw std::runtime_error("Деление на ноль невозможно");
    }
    return a / b;
}

// Возведение в степень (расширенная функция)
static double power(double base, double exponent) {
    // Проверка особых случаев
    if (base == 0 && exponent <= 0) {
        throw std::domain_error("Ноль в отрицательной или нулевой степени не определен");
    }

    // Целая степень
    if (std::abs(exponent - std::floor(exponent)) < 1e-10) {
        return integerPower(base, static_cast<int>(exponent));
    }

    // Дробная степень (только для положительных оснований)
    if (base < 0) {
        throw std::domain_error("Отрицательное основание с дробным показателем не поддерживается");
    }

    return std::pow(base, exponent);
}

private:
    // Оптимизированное возведение в целую степень
    static double integerPower(double base, int exponent) {
        if (exponent == 0) return 1;
        if (exponent == 1) return base;

        bool negativeExponent = exponent < 0;
        if (negativeExponent) {
            if (base == 0) {
                throw std::domain_error("Ноль в отрицательной степени не определен");
            }
            exponent = -exponent;
        }

        double result = 1.0;
        double current = base;

        // Алгоритм быстрого возведения в степень
        while (exponent > 0) {

```

```

    if (exponent % 2 == 1) {
        // Проверка переполнения при умножении
        if (std::abs(result) > std::numeric_limits<double>::max() / std::abs(current)) {
            throw std::overflow_error("Переполнение при возведении в степень");
        }
        result *= current;
    }

    exponent /= 2;
    if (exponent > 0) {
        // Проверка переполнения при возведении в квадрат
        if (std::abs(current) > std::numeric_limits<double>::max() / std::abs(current)) {
            throw std::overflow_error("Переполнение при возведении в степень");
        }
        current *= current;
    }
}

return negativeExponent ? 1.0 / result : result;
}
};

// Функция для безопасного ввода числа
double getNumber(const std::string& prompt, int numberIndex) {
    std::string input;

    while (true) {
        std::cout << prompt;
        std::getline(std::cin, input);

        // Удаление начальных и конечных пробелов
        input.erase(0, input.find_first_not_of(" \t\n\r"));
        input.erase(input.find_last_not_of(" \t\n\r") + 1);

        if (input.empty()) {
            std::cout << "Ошибка: Ввод не может быть пустым.\n";
            continue;
        }

        if (!NumberValidator::isValidDouble(input)) {
            std::cout << "Ошибка: Введите корректное число.\n";
            continue;
        }

        try {
            return NumberValidator::parseDouble(input);
        } catch (const std::exception& e) {
            std::cout << "Ошибка: " << e.what() << "\n";
        }
    }
}

```

// Функция для выбора операции

```
char getOperation() {
    std::string input;

    while (true) {
        std::cout << "\nВыберите операцию:\n";
        std::cout << " + - Сложение\n";
        std::cout << " - - Вычитание\n";
        std::cout << " * - Умножение\n";
        std::cout << " / - Деление\n";
        std::cout << " ^ - Возведение в степень\n";
        std::cout << " q - Выход\n";
        std::cout << "Ваш выбор: ";

        std::getline(std::cin, input);

        if (input.length() == 1) {
            char op = input[0];
            if (op == '+' || op == '-' || op == '*' || op == '/' || op == '^' || op == 'q') {
                return op;
            }
        }

        std::cout << "Ошибка: Выберите одну из доступных операций.\n";
    }
}
```

// Функция для форматированного вывода результата

```
void printResult(double a, double b, char operation, double result) {
    std::cout.precision(10);
    std::cout << "\n" << a << " " << operation << " " << b << " = " << result << "\n";
}
```

```
int main() {
    setlocale(LC_CTYPE, "Russian");

    std::cout << "=== Улучшенный калькулятор ===\n\n";

    while (true) {
        try {
            // Получение первого числа
            double a1 = getNumber("Введите первое число: ", 1);

            // Выбор операции
            char operation = getOperation();
            if (operation == 'q') {
                std::cout << "Выход из программы.\n";
                break;
            }

            // Получение второго числа
            double a2 = getNumber("Введите второе число: ", 2);
```

```

// Выполнение операции и вывод результата
double result;

switch (operation) {
    case '+':
        result = AdvancedCalculator::add(a1, a2);
        break;
    case '-':
        result = AdvancedCalculator::subtract(a1, a2);
        break;
    case '*':
        result = AdvancedCalculator::multiply(a1, a2);
        break;
    case '/':
        result = AdvancedCalculator::divide(a1, a2);
        break;
    case '^':
        result = AdvancedCalculator::power(a1, a2);
        break;
    default:
        throw std::runtime_error("Неизвестная операция");
}

printResult(a1, a2, operation, result);

} catch (const std::exception& e) {
    std::cout << "\nОшибка: " << e.what() << "\n";
}

std::cout << "\n" << std::string(40, '=') << "\n";
}

std::cout << "\nДля завершения нажмите Enter...";
std::cin.get();

return 0;
}

```

Сравнение версий калькулятора:

Функциональность	Исходная версия	Оптимизированная версия
Валидация ввода	Нет	Полная валидация всех входных данных
Обработка ошибок	Нет	Полная обработка исключений
Операции	4 базовые	5 операций (добавлено возведение в степень)
Проверка переполнения	Нет	Проверка для всех операций

Функциональность	Исходная версия	Оптимизированная версия
Архитектура	Процедурная	ООП с классами и методами
Повторное использование	Нет	Возможность повторного использования компонентов
Интерфейс	Простой	Интерактивный с меню операций

Задания по вариантам

Вариант 1: Оптимизация программы сортировки

Исходный код для оптимизации:

```

cpp
#include <iostream>
using namespace std;

int main() {
    int arr[10];
    cout << "Введите 10 чисел: ";
    for(int i=0; i<10; i++) cin >> arr[i];

    for(int i=0; i<9; i++)
        for(int j=0; j<9; j++)
            if(arr[j]>arr[j+1]) {
                int temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }

    cout<<"Отсортированный массив: ";
    for(int i=0; i<10; i++) cout<<arr[i]<<" ";
    return 0;
}

```

Задания:

1. Базовая оптимизация (5 баллов):

- Добавить валидацию ввода (только числа)
- Исправить алгоритм пузырьковой сортировки (неэффективный внутренний цикл)
- Добавить обработку ошибок

2. Расширенная оптимизация (10 баллов):

- Реализовать несколько алгоритмов сортировки (быстрая, слиянием, вставками)
- Добавить выбор алгоритма пользователем
- Реализовать замер времени выполнения
- Добавить возможность работы с динамическим массивом произвольного размера

3. Метрический анализ (5 баллов):

- Рассчитать метрики Холстеда для исходной и оптимизированной версий
- Сравнить результаты в таблице
- Сделать выводы об эффективности оптимизации

Вариант 2: Оптимизация программы работы с файлами

Исходный код для оптимизации:

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

int main() {
    ofstream file("data.txt");
    if(!file) {cout<<"Ошибка!";return 1;}

    int n;
    cout<<"Сколько чисел записать? ";
    cin>>n;

    for(int i=0;i<n;i++) {
        int num;
        cout<<"Число "<<i+1<<": ";
        cin>>num;
        file<<num<<" ";
    }
    file.close();

    ifstream infile("data.txt");
    int sum=0,count=0;
    while(infile>>n) {
        sum+=n;
        count++;
    }

    cout<<"Среднее: "<<sum/count;
    return 0;
}
```

Задания:

1. **Базовая оптимизация (5 баллов):**
 - Добавить полную валидацию ввода
 - Обработать деление на ноль при вычислении среднего
 - Добавить проверку существования файла
 - Реализовать обработку ошибок ввода/вывода
2. **Расширенная оптимизация (10 баллов):**
 - Реализовать класс для работы с данными
 - Добавить статистику (минимум, максимум, медиана, мода)
 - Реализовать возможность выбора операции
 - Добавить поддержку разных форматов файлов (текст, бинарный)
 - Реализовать кэширование данных для повторных операций
3. **Метрический анализ (5 баллов):**
 - Рассчитать метрики Холстеда для обеих версий
 - Сравнить сложность и объем программ
 - Проанализировать влияние ООП на метрики

Критерии оценивания

Общие критерии для всех вариантов:

Критерий	Баллы	Описание
Функциональность	0-5	0 - программа не работает; 1-2 - частичная реализация; 3-4 - полная реализация; 5 - расширенная функциональность
Валидация ввода	0-4	0 - нет валидации; 1-2 - базовая проверка; 3-4 - полная валидация с обработкой ошибок
Архитектура кода	0-4	0 - спагетти-код; 1-2 - процедурный подход; 3-4 - ООП с разделением ответственности
Обработка ошибок	0-3	0 - нет обработки; 1 - базовые проверки; 2 - обработка исключений; 3 - полная обработка всех возможных ошибок
Эффективность	0-4	0 - неэффективные алгоритмы; 1-2 - базовые оптимизации; 3-4 - эффективные алгоритмы с оптимальной сложностью

Метрический анализ:

Метрика	Исходная программа	Оптимизированная программа	Изменение	Баллы
Словарь (n)				0-2
Длина (N)				0-2
Объем (V)				0-2
Сложность (D)				0-2
Усилия (E)				0-2
Анализ результатов			Обоснованные выводы	0-5

Таблица для расчета метрик Холстеда:

cpp

// Пример расчета для простой программы

#include <iostream>

using namespace std;

```
int main() {           // n1: main, (, ), {, return, ;, }
    int x = 10;         // n1: int, =, ;
    int y = 20;         // n1: int, =, ;
    int sum = x + y;     // n1: int, =, +, ;
    cout << sum << endl; // n1: cout, <<, <<, endl, ;
```

```

return 0;           // n1: return, ;
}                  // n1: }

// n1 (уникальные операторы):
// {, }, main, (, ), int, =, ;, +, cout, <<, endl, return = 12

// n2 (уникальные операнды):
// x, y, sum, 10, 20, 0 = 6

// N1 (общее число операторов):
// Подсчет всех операторов в коде = 15

// N2 (общее число операндов):
// Подсчет всех операндов в коде = 8

```

Шкала оценивания:

Оценка	Баллы	Требования
5 (отлично)	41-50	Полная реализация всех требований, отличная оптимизация, подробный метрический анализ с обоснованными выводами
4 (хорошо)	31-40	Реализация основных требований, хорошая оптимизация, проведен метрический анализ
3 (удовлетворительно)	21-30	Реализация базовых требований, проведены основные оптимизации, частичный метрический анализ
2 (неудовлетворительно)	0-20	Не выполнены основные требования или программа не работает

Инструкция по выполнению работы

Шаг 1: Анализ исходного кода

1. Изучите предоставленную программу
2. Определите слабые места и возможности для оптимизации
3. Составьте план улучшений

Шаг 2: Базовая оптимизация

1. Добавьте валидацию всех входных данных
2. Реализуйте обработку ошибок
3. Устраните дублирование кода
4. Улучшите читаемость (комментарии, форматирование)

Шаг 3: Расширенная оптимизация

1. Реализуйте дополнительные функции
2. Примените ООП (если уместно)
3. Оптимизируйте алгоритмы
4. Добавьте пользовательский интерфейс

Шаг 4: Метрический анализ

1. Составьте таблицу элементов программы:
text

Операторы (n1): [список уникальных операторов]

Операнды (n2): [список уникальных операндов]

2. Рассчитайте метрики:

text

$$n = n1 + n2$$

$$N = N1 + N2$$

$$V = N \times \log_2(n)$$

$$D = (n1/2) \times (N2/n2)$$

$$E = D \times V$$

3. Заполните сравнительную таблицу

Шаг 5: Формирование отчета

1. **Титульный лист:** название работы, ФИО, группа

2. **Цель и задачи работы**

3. **Исходный код** (до оптимизации)

4. **Оптимизированный код** (после оптимизации)

5. **Сравнительный анализ:**

○ Таблица сравнения функциональности

○ Таблица метрик Холстеда

○ Графики изменения метрик

6. **Выводы:** анализ эффективности оптимизации

Пример таблицы для отчета

Таблица 1: Сравнение функциональности

Характеристика	Исходная программа	Оптимизированная программа	Улучшение
Валидация ввода	Нет	Полная	+
Обработка ошибок	Нет	Полная	+
Количество операций	4	5	+1
Архитектура	Процедурная	ООП	+
Проверка переполнения	Нет	Да	+
Читаемость кода	Низкая	Высокая	+

Таблица 2: Метрики Холстеда

Метрика	Исходная программа	Оптимизированная программа	Изменение (%)
n (словарь)	15	25	+67%
N (длина)	22	45	+105%
V (объем)	88.5	209.7	+137%
D (сложность)	3.75	5.2	+39%
E (усилия)	331.9	1090.4	+228%

Анализ результатов:

1. **Увеличение объема (V)** связано с добавлением функциональности
2. **Рост сложности (D)** обусловлен более сложной логикой
3. **Значительное увеличение усилий (E)** отражает более сложную разработку
4. **Вывод:** Оптимизация увеличила сложность программы, но значительно улучшила надежность и функциональность

Советы по оптимизации

1. Приоритеты оптимизации:

- Сначала обеспечьте корректность работы
- Затем улучшайте читаемость
- В последнюю очередь оптимизируйте производительность

2. Измеряйте эффективность:

- Используйте `std::chrono` для замера времени
- Профилируйте программу для нахождения узких мест
- Сравнивайте разные алгоритмы

3. Документируйте изменения:

- Комментируйте причины оптимизаций
- Сохраняйте историю изменений
- Тестируйте после каждой значительной модификации

4. Используйте стандартные библиотеки:

- STL контейнеры и алгоритмы
- Умные указатели для управления памятью
- Функциональные объекты и лямбда-выражения

Данная практическая работа поможет освоить важные навыки рефакторинга и оптимизации кода, которые необходимы для профессиональной разработки программного обеспечения.

ПРАКТИЧЕСКАЯ РАБОТА №25-29. Рефакторинг кода.

Цель и задачи

Цель работы: Изучить принципы и техники рефакторинга программного кода, освоить практические навыки улучшения структуры существующего кода на C++/Qt.

Задачи:

1. Изучить инструменты рефакторинга в среде Qt Creator
2. Освоить основные шаблоны рефакторинга
3. Научиться выявлять проблемы в структуре кода
4. Применить рефакторинг для улучшения читаемости и поддерживаемости кода
5. Проанализировать результаты рефакторинга

Теоретический материал

Инструменты рефакторинга в Qt Creator

1. Встроенные возможности Qt Creator

- **Переименование (F2)** - переменных, функций, классов
- **Извлечение функции (Ctrl+Alt+M)** - создание функции из выделенного кода
- **Извлечение переменной (Ctrl+Alt+V)** - создание переменной из выражения
- **Внедрение переменной (Ctrl+Alt+N)** - замена переменной её значением
- **Перемещение функции/класса** - между файлами

2. Дополнительные плагины

- **Clang Tools** - анализ кода и автоматический рефакторинг
- **ClangFormat** - автоматическое форматирование кода
- **Cppcheck** - статический анализ

3. Ручные техники рефакторинга

- Извлечение классов и интерфейсов
- Упрощение условных выражений
- Замена магических чисел на константы
- Устранение дублирования

Основные шаблоны рефакторинга

Шаблон	Цель	Когда применять
Extract Method	Уменьшить сложность функции	Функция слишком длинная или делает слишком много
Inline Method	Упростить код	Функция слишком простая
Extract Variable	Улучшить читаемость	Сложное выражение повторяется
Rename	Улучшить понимание кода	Имена не отражают назначение
Introduce Parameter Object	Уменьшить количество параметров	Функция имеет много параметров
Replace Temp with Query	Устранить временные переменные	Временная переменная используется в нескольких местах

Практическая часть: Рефакторинг игрового кода на C++/Qt

Исходный код (до рефакторинга)

cpp

// mainwindow.h - до рефакторинга

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
```

```
#include <QMainWindow>
#include <QTimer>
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QKeyEvent>
#include <vector>
```

```
class MainWindow : public QMainWindow {
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
```

```
protected:
    void keyPressEvent(QKeyEvent *event) override;
    void keyReleaseEvent(QKeyEvent *event) override;
```

```
private slots:
    void updateGame();
```

```
private:
    QGraphicsScene* scene;
    QGraphicsView* view;
    QTimer* timer;
```

// Игровые объекты

```
struct Player {
    double x, y;
    double vx, vy;
    int hp;
    int score;
    bool alive;
} player;
```

```
struct Enemy {
    double x, y;
    double vx, vy;
    int hp;
    bool active;
};
```

```
std::vector<Enemy> enemies;
std::vector<QGraphicsRectItem*> enemyItems;
QGraphicsRectItem* playerItem;
```

// Состояния клавиш

```

bool keyLeft = false;
bool keyRight = false;
bool keyUp = false;
bool keyDown = false;
bool keySpace = false;

// Вспомогательные методы
void initGame();
void spawnEnemy();
void checkCollisions();
void updatePlayer();
void updateEnemies();
void draw();

// Константы
const int SCENE_WIDTH = 800;
const int SCENE_HEIGHT = 600;
const int PLAYER_SIZE = 20;
const int ENEMY_SIZE = 15;
const double PLAYER_SPEED = 5.0;
const double ENEMY_SPEED = 2.0;
};

#endif // MAINWINDOW_H

// mainwindow.cpp - до рефакторинга
#include "mainwindow.h"
#include <QGraphicsRectItem>
#include <QBrush>
#include <QPen>
#include <QDebug>
#include <QRandomGenerator>

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent) {
    // Инициализация сцены
    scene = new QGraphicsScene(this);
    scene->setSceneRect(0, 0, SCENE_WIDTH, SCENE_HEIGHT);
    scene->setBackgroundBrush(Qt::black);

    view = new QGraphicsView(scene, this);
    view->setRenderHint(QPainter::Antialiasing);
    view->setViewportUpdateMode(QGraphicsView::FullViewportUpdate);
    setCentralWidget(view);

    // Инициализация игрока
    player.x = SCENE_WIDTH / 2;
    player.y = SCENE_HEIGHT / 2;
    player.vx = 0;
    player.vy = 0;
    player.hp = 100;
    player.score = 0;
    player.alive = true;
}

```

```

playerItem = new QGraphicsRectItem(0, 0, PLAYER_SIZE, PLAYER_SIZE);
playerItem->setBrush(Qt::blue);
playerItem->setPen(QPen(Qt::white));
scene->addItem(playerItem);

// Создание врагов
for (int i = 0; i < 10; i++) {
    Enemy enemy;
    enemy.x = QRandomGenerator::global()->bounded(SCENE_WIDTH - ENEMY_SIZE);
    enemy.y = QRandomGenerator::global()->bounded(SCENE_HEIGHT - ENEMY_SIZE);
    enemy.vx = (QRandomGenerator::global()->bounded(200) - 100) / 100.0;
    enemy.vy = (QRandomGenerator::global()->bounded(200) - 100) / 100.0;
    enemy.hp = 10;
    enemy.active = true;
    enemies.push_back(enemy);

    QGraphicsRectItem* enemyItem = new QGraphicsRectItem(0, 0, ENEMY_SIZE,
ENEMY_SIZE);
    enemyItem->setBrush(Qt::red);
    enemyItem->setPen(QPen(Qt::white));
    scene->addItem(enemyItem);
    enemyItems.push_back(enemyItem);
}

// Настройка таймера
timer = new QTimer(this);
connect(timer, &QTimer::timeout, this, &MainWindow::updateGame);
timer->start(16); // ~60 FPS

setWindowTitle("Игра до рефакторинга");
resize(SCENE_WIDTH, SCENE_HEIGHT);
}

void MainWindow::keyPressEvent(QKeyEvent *event) {
    if (event->key() == Qt::Key_Left) keyLeft = true;
    if (event->key() == Qt::Key_Right) keyRight = true;
    if (event->key() == Qt::Key_Up) keyUp = true;
    if (event->key() == Qt::Key_Down) keyDown = true;
    if (event->key() == Qt::Key_Space) keySpace = true;
    QMainWindow::keyPressEvent(event);
}

void MainWindow::keyReleaseEvent(QKeyEvent *event) {
    if (event->key() == Qt::Key_Left) keyLeft = false;
    if (event->key() == Qt::Key_Right) keyRight = false;
    if (event->key() == Qt::Key_Up) keyUp = false;
    if (event->key() == Qt::Key_Down) keyDown = false;
    if (event->key() == Qt::Key_Space) keySpace = false;
    QMainWindow::keyReleaseEvent(event);
}

```

```

void MainWindow::updateGame() {
    if (!player.alive) return;

    // Обновление игрока
    player.vx = 0;
    player.vy = 0;
    if (keyLeft) player.vx = -PLAYER_SPEED;
    if (keyRight) player.vx = PLAYER_SPEED;
    if (keyUp) player.vy = -PLAYER_SPEED;
    if (keyDown) player.vy = PLAYER_SPEED;

    player.x += player.vx;
    player.y += player.vy;

    // Ограничение движения границами
    if (player.x < 0) player.x = 0;
    if (player.x > SCENE_WIDTH - PLAYER_SIZE) player.x = SCENE_WIDTH -
    PLAYER_SIZE;
    if (player.y < 0) player.y = 0;
    if (player.y > SCENE_HEIGHT - PLAYER_SIZE) player.y = SCENE_HEIGHT -
    PLAYER_SIZE;

    // Обновление врагов
    for (size_t i = 0; i < enemies.size(); i++) {
        if (!enemies[i].active) continue;

        enemies[i].x += enemies[i].vx;
        enemies[i].y += enemies[i].vy;

        // Отражение от границ
        if (enemies[i].x < 0 || enemies[i].x > SCENE_WIDTH - ENEMY_SIZE) enemies[i].vx = -
        enemies[i].vx;
        if (enemies[i].y < 0 || enemies[i].y > SCENE_HEIGHT - ENEMY_SIZE) enemies[i].vy = -
        enemies[i].vy;

        // Проверка столкновения с игроком
        if (player.x < enemies[i].x + ENEMY_SIZE &&
            player.x + PLAYER_SIZE > enemies[i].x &&
            player.y < enemies[i].y + ENEMY_SIZE &&
            player.y + PLAYER_SIZE > enemies[i].y) {
            player.hp -= 10;
            enemies[i].active = false;
            if (player.hp <= 0) {
                player.alive = false;
                qDebug() << "Игрок погиб!";
            }
        }
    }
}

// Отрисовка
playerItem->setPos(player.x, player.y);
for (size_t i = 0; i < enemies.size(); i++) {

```

```

        if (enemies[i].active) {
            enemyItems[i]->setPos(enemies[i].x, enemies[i].y);
            enemyItems[i]->setVisible(true);
        } else {
            enemyItems[i]->setVisible(false);
        }
    }
}

// Обработка выстрела
if (keySpace) {
    for (size_t i = 0; i < enemies.size(); i++) {
        if (enemies[i].active) {
            double dx = enemies[i].x - player.x;
            double dy = enemies[i].y - player.y;
            double distance = sqrt(dx*dx + dy*dy);
            if (distance < 100) {
                enemies[i].hp -= 5;
                if (enemies[i].hp <= 0) {
                    enemies[i].active = false;
                    player.score += 10;
                }
            }
        }
    }
    keySpace = false;
}
}

```

Проблемы в исходном коде:

1. Слишком длинные методы - updateGame() делает слишком много
2. Дублирование кода - похожая логика для игрока и врагов
3. Смещение ответственности - один метод и логику, и отрисовку
4. Использование "магических чисел" - константы разбросаны по коду
5. Сложные условные выражения - проверка столкновений
6. Слабая инкапсуляция - структуры с публичными полями

Рефакторинг шаг за шагом

Шаг 1: Создание классов для игровых объектов

cpp

// gameobject.h - новый файл

```

#ifndef GAMEOBJECT_H
#define GAMEOBJECT_H

```

```

#include <QGraphicsItem>
#include <QObject>

```

```

class GameObject : public QObject, public QGraphicsItem {
    Q_OBJECT
    Q_INTERFACES(QGraphicsItem)
public:
    GameObject(QGraphicsItem* parent = nullptr);
    virtual ~GameObject() = default;

```

// Базовые свойства

```

virtual double x() const { return m_x; }
virtual double y() const { return m_y; }
virtual double width() const { return m_width; }
virtual double height() const { return m_height; }
virtual int health() const { return m_health; }
virtual bool isAlive() const { return m_alive; }

// Базовые операции
virtual void setPosition(double x, double y);
virtual void move(double dx, double dy);
virtual void takeDamage(int damage);
virtual void update() = 0;

// Графическое представление
QRectF boundingRect() const override;
void paint(QPainter* painter,
           const QStyleOptionGraphicsItem* option,
           QWidget* widget) override;

protected:
    double m_x = 0;
    double m_y = 0;
    double m_width = 20;
    double m_height = 20;
    int m_health = 100;
    bool m_alive = true;
    QColor m_color = Qt::white;

    virtual void onCollision(GameObject* other) {}
};

#endif // GAMEOBJECT_H

// player.h - новый файл
#ifndef PLAYER_H
#define PLAYER_H

#include "gameobject.h"
#include <QSet>

class Player : public GameObject {
    Q_OBJECT
public:
    explicit Player(QGraphicsItem* parent = nullptr);

// Управление
    void setMovingLeft(bool moving);
    void setMovingRight(bool moving);
    void setMovingUp(bool moving);
    void setMovingDown(bool moving);
    void setShooting(bool shooting);

```

```

// Состояние
int score() const { return m_score; }
void addScore(int points) { m_score += points; }

// Обновление
void update() override;

protected:
    void onCollision(GameObject* other) override;

private:
    double m_speed = 5.0;
    int m_score = 0;
    bool m_movingLeft = false;
    bool m_movingRight = false;
    bool m_movingUp = false;
    bool m_movingDown = false;
    bool m_shooting = false;

    void updateMovement();
    void handleShooting();
    void checkBoundaries(double sceneWidth, double sceneHeight);
};

#endif // PLAYER_H

// enemy.h - новый файл
#ifndef ENEMY_H
#define ENEMY_H

#include "gameobject.h"
#include <QTimer>

class Enemy : public GameObject {
    Q_OBJECT
public:
    enum Type {
        Normal,
        Fast,
        Tank
    };

    explicit Enemy(Type type = Normal, QGraphicsItem* parent = nullptr);

    void update() override;
    void initializeRandom(double sceneWidth, double sceneHeight);

    Type type() const { return m_type; }

protected:
    void onCollision(GameObject* other) override;

```

```
private:
    Type m_type = Normal;
    double m_speed = 2.0;
    double m_vx = 0;
    double m_vy = 0;

    void updateMovement(double sceneWidth, double sceneHeight);
    void bounceOffBoundaries(double sceneWidth, double sceneHeight);
};
```

```
#endif // ENEMY_H
```

Шаг 2: Создание менеджера игры

cpp

// gamemanager.h - новый файл

```
#ifndef GAMEMANAGER_H
```

```
#define GAMEMANAGER_H
```

```
#include <QObject>
#include <QGraphicsScene>
#include <QTimer>
#include <vector>
#include <memory>
#include "player.h"
#include "enemy.h"
```

```
class GameManager : public QObject {
    Q_OBJECT
```

```
public:
```

```
    explicit GameManager(QObject* parent = nullptr);
```

// Инициализация

```
    void initialize(QGraphicsScene* scene, int width, int height);
```

// Управление игрой

```
    void startGame();
```

```
    void pauseGame();
```

```
    void resetGame();
```

// Управление игроком

```
    void playerMoveLeft(bool moving);
```

```
    void playerMoveRight(bool moving);
```

```
    void playerMoveUp(bool moving);
```

```
    void playerMoveDown(bool moving);
```

```
    void playerShoot(bool shooting);
```

// Состояние игры

```
    bool isGameRunning() const { return m_gameRunning; }
```

```
    bool isGameOver() const { return m_gameOver; }
```

```
    int playerScore() const;
```

```
    int playerHealth() const;
```

signals:

```

void gameStarted();
void gamePaused();
void gameOver(int score);
void scoreChanged(int score);
void healthChanged(int health);

private slots:
    void updateGame();

private:
    // Игровые объекты
    Player* m_player = nullptr;
    std::vector<std::unique_ptr<Enemy>> m_enemies;
    QGraphicsScene* m_scene = nullptr;

    // Таймер игры
    QTimer m_gameTimer;

    // Состояние игры
    bool m_gameRunning = false;
    bool m_gameOver = false;
    int m_sceneWidth = 800;
    int m_sceneHeight = 600;

    // Константы
    static constexpr int ENEMY_COUNT = 10;
    static constexpr double SPAWN_DISTANCE_FROM_PLAYER = 200.0;

    // Вспомогательные методы
    void spawnEnemies();
    void checkCollisions();
    void removeDeadEnemies();
    void handleEnemySpawn();
    void updateGameObjects();
    void renderGameObjects();

    bool checkCollision(GameObject* obj1, GameObject* obj2);
    bool isPositionValidForSpawn(double x, double y);
};

#endif // GAMEMANAGER_H
Шаг 3: Оптимизированный MainWindow
cpp
// mainwindow.h - после рефакторинга
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QGraphicsView>
#include "gamemanager.h"

QT_BEGIN_NAMESPACE

```

```

class QLabel;
QT_END_NAMESPACE

class MainWindow : public QMainWindow {
    Q_OBJECT
public:
    explicit MainWindow(QWidget* parent = nullptr);

protected:
    void keyPressEvent(QKeyEvent* event) override;
    void keyReleaseEvent(QKeyEvent* event) override;
    void closeEvent(QCloseEvent* event) override;

private slots:
    void onGameStarted();
    void onGamePaused();
    void onGameOver(int score);
    void onScoreChanged(int score);
    void onHealthChanged(int health);

    void startNewGame();
    void togglePause();
    void showHelp();

private:
    // Графические компоненты
    QGraphicsView* m_view = nullptr;
    QGraphicsScene* m_scene = nullptr;

    // Игровой менеджер
    GameManager* m_gameManager = nullptr;

    // UI элементы
    QLabel* m_scoreLabel = nullptr;
    QLabel* m_healthLabel = nullptr;
    QLabel* m_statusLabel = nullptr;

    // Константы
    static constexpr int SCENE_WIDTH = 800;
    static constexpr int SCENE_HEIGHT = 600;

    // Инициализация
    void setupUI();
    void setupGame();
    void setupMenu();
    void setupStatusBar();

    // Обработка ввода
    void handleKeyInput(QKeyEvent* event, bool pressed);

    // Вспомогательные методы
    void updateGameInfo();

```

```

};

#endif // MAINWINDOW_H

// mainwindow.cpp - после рефакторинга
#include "mainwindow.h"
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QKeyEvent>
#include <QMenuBar>
#include <QStatusBar>
#include <QLabel>
#include <QMessageBox>
#include <QVBoxLayout>
#include <QPushButton>
#include <QTimer>
#include <QCloseEvent>

MainWindow::MainWindow(QWidget* parent) : QMainWindow(parent) {
    // Настройка окна
    setWindowTitle("Игра после рефакторинга");
    setMinimumSize(SCENE_WIDTH, SCENE_HEIGHT);

    // Инициализация UI
    setupUI();
    setupMenu();
    setupStatusBar();

    // Инициализация игры
    setupGame();

    // Показать приветственное сообщение
    QTimer::singleShot(100, this, [this]() {
        QMessageBox::information(this, "Добро пожаловать",
            "Управление:\n"
            "←→↑↓ - движение\n"
            "Пробел - стрельба\n"
            "P - пауза\n"
            "N - новая игра\n"
            "H - помощь");
    });
}

void MainWindow::setupUI() {
    // Создание центрального виджета
    QWidget* centralWidget = new QWidget(this);
    setCentralWidget(centralWidget);

    // Создание сцены и представления
    m_scene = new QGraphicsScene(this);
    m_scene->setSceneRect(0, 0, SCENE_WIDTH, SCENE_HEIGHT);
    m_scene->setBackgroundBrush(Qt::black);

```

```

m_view = new QGraphicsView(m_scene, centralWidget);
m_view->setRenderHint(QPainter::Antialiasing);
m_view->setViewportUpdateMode(QGraphicsView::SmartViewportUpdate);
m_view->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
m_view->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

// Компоновка
QVBoxLayout* layout = new QVBoxLayout(centralWidget);
layout->addWidget(m_view);
layout->setContentsMargins(0, 0, 0, 0);
}

void MainWindow::setupMenu() {
    // Меню "Игра"
    QMenu* gameMenu = menuBar()->addMenu("&Игра");

    QAction* newGameAction = gameMenu->addAction("&Новая игра");
    newGameAction->setShortcut(QKeySequence::New);
    connect(newGameAction, &QAction::triggered, this, &MainWindow::startNewGame);

    QAction* pauseAction = gameMenu->addAction("&Пауза");
    pauseAction->setShortcut(Qt::Key_P);
    connect(pauseAction, &QAction::triggered, this, &MainWindow::togglePause);

    gameMenu->addSeparator();

    QAction* exitAction = gameMenu->addAction("&Выход");
    exitAction->setShortcut(QKeySequence::Quit);
    connect(exitAction, &QAction::triggered, this, &QWidget::close);

    // Меню "Помощь"
    QMenu* helpMenu = menuBar()->addMenu("&Помощь");

    QAction* helpAction = helpMenu->addAction("&Справка");
    helpAction->setShortcut(Qt::Key_H);
    connect(helpAction, &QAction::triggered, this, &MainWindow::showHelp);

    QAction* aboutAction = helpMenu->addAction("&О программе");
    connect(aboutAction, &QAction::triggered, []() {
        QMessageBox::about(nullptr, "О программе",
            "Пример рефакторинга игрового кода\n"
            "Использует Qt и C++\n"
            "Версия 1.0");
    });
}

void MainWindow::setupStatusBar() {
    // Создание меток для статус-бара
    m_scoreLabel = new QLabel("Счет: 0");
    m_healthLabel = new QLabel("Здоровье: 100%");
    m_statusLabel = new QLabel("Готов к игре");
}

```

```

        statusBar()->addWidget(m_scoreLabel);
        statusBar()->addWidget(m_healthLabel);
        statusBar()->addPermanentWidget(m_statusLabel);
    }

    void MainWindow::setupGame() {
        // Создание игрового менеджера
        m_gameManager = new GameManager(this);
        m_gameManager->initialize(m_scene, SCENE_WIDTH, SCENE_HEIGHT);

        // Подключение сигналов
        connect(m_gameManager, &GameManager::gameStarted,
            this, &MainWindow::onGameStarted);
        connect(m_gameManager, &GameManager::gamePaused,
            this, &MainWindow::onGamePaused);
        connect(m_gameManager, &GameManager::gameOver,
            this, &MainWindow::onGameOver);
        connect(m_gameManager, &GameManager::scoreChanged,
            this, &MainWindow::onScoreChanged);
        connect(m_gameManager, &GameManager::healthChanged,
            this, &MainWindow::onHealthChanged);

        // Автоматический старт игры
        QTimer::singleShot(500, m_gameManager, &GameManager::startGame);
    }

    void MainWindow::keyPressEvent(QKeyEvent* event) {
        handleKeyInput(event, true);
        QMainWindow::keyPressEvent(event);
    }

    void MainWindow::keyReleaseEvent(QKeyEvent* event) {
        handleKeyInput(event, false);
        QMainWindow::keyReleaseEvent(event);
    }

    void MainWindow::handleKeyInput(QKeyEvent* event, bool pressed) {
        if (!m_gameManager || !m_gameManager->isGameRunning()) {
            return;
        }

        switch (event->key()) {
            case Qt::Key_Left:
                m_gameManager->playerMoveLeft(pressed);
                break;
            case Qt::Key_Right:
                m_gameManager->playerMoveRight(pressed);
                break;
            case Qt::Key_Up:
                m_gameManager->playerMoveUp(pressed);
                break;
        }
    }

```

```

case Qt::Key_Down:
    m_gameManager->playerMoveDown(pressed);
    break;
case Qt::Key_Space:
    m_gameManager->playerShoot(pressed);
    break;
case Qt::Key_P:
    if (!pressed) togglePause();
    break;
case Qt::Key_N:
    if (!pressed) startNewGame();
    break;
case Qt::Key_H:
    if (!pressed) showHelp();
    break;
default:
    break;
}
}

void MainWindow::onGameStarted() {
    m_statusLabel->setText("Игра запущена");
    updateGameInfo();
}

void MainWindow::onGamePaused() {
    m_statusLabel->setText("Игра на паузе");
}

void MainWindow::onGameOver(int score) {
    m_statusLabel->setText("Игра окончена");

    QMessageBox::information(this, "Игра окончена",
        QString("Ваш счет: %1\n\n"
            "Нажмите 'Новая игра' чтобы начать заново").arg(score));
}

void MainWindow::onScoreChanged(int score) {
    m_scoreLabel->setText(QString("Счет: %1").arg(score));
}

void MainWindow::onHealthChanged(int health) {
    m_healthLabel->setText(QString("Здоровье: %1%").arg(health));
}

void MainWindow::startNewGame() {
    if (m_gameManager) {
        m_gameManager->resetGame();
        m_gameManager->startGame();
    }
}

```

```

void MainWindow::togglePause() {
    if (!m_gameManager) return;

    if (m_gameManager->isGameRunning()) {
        m_gameManager->pauseGame();
    } else {
        m_gameManager->startGame();
    }
}

void MainWindow::showHelp() {
    QMessageBox::information(this, "Справка",
        "Управление в игре:\n\n"
        "Стрелки - движение игрока\n"
        "Пробел - стрельба\n"
        "P - пауза/продолжить\n"
        "N - новая игра\n"
        "H - эта справка\n\n"
        "Цель игры: уничтожить всех врагов, избегая столкновений.");
}

void MainWindow::updateGameInfo() {
    if (m_gameManager) {
        onScoreChanged(m_gameManager->playerScore());
        onHealthChanged(m_gameManager->playerHealth());
    }
}

void MainWindow::closeEvent(QCloseEvent* event) {
    if (m_gameManager && m_gameManager->isGameRunning()) {
        QMessageBox::StandardButton reply = QMessageBox::question(this, "Подтверждение",
            "Игра все еще запущена. Вы уверены, что хотите выйти?",
            QMessageBox::Yes | QMessageBox::No);

        if (reply == QMessageBox::No) {
            event->ignore();
            return;
        }
    }

    event->accept();
}

```

Шаг 4: Реализация GameObject

```

cpp
// gameobject.cpp
#include "gameobject.h"
#include <QPainter>
#include <QDebug>

GameObject::GameObject(QGraphicsItem* parent)
    : QObject(), QGraphicsItem(parent) {
    // Настройка флагов

```

```

setFlag(QGraphicsItem::ItemIsMovable, false);
setFlag(QGraphicsItem::ItemIsSelectable, false);
setFlag(QGraphicsItem::ItemSendsGeometryChanges, true);
}

void GameObject::setPosition(double x, double y) {
    prepareGeometryChange();
    m_x = x;
    m_y = y;
    setPos(x, y);
}

void GameObject::move(double dx, double dy) {
    setPosition(m_x + dx, m_y + dy);
}

void GameObject::takeDamage(int damage) {
    if (!m_alive) return;

    m_health -= damage;
    if (m_health <= 0) {
        m_health = 0;
        m_alive = false;
        emit destroyed();
    }

    emit healthChanged(m_health);
}

QRectF GameObject::boundingRect() const {
    return QRectF(0, 0, m_width, m_height);
}

void GameObject::paint(QPainter* painter,
    const QStyleOptionGraphicsItem* option,
    QWidget* widget) {
    Q_UNUSED(option);
    Q_UNUSED(widget);

    // Основной прямоугольник
    painter->setBrush(m_color);
    painter->setPen(QPen(Qt::white, 2));
    painter->drawRect(boundingRect());

    // Индикатор здоровья
    if (m_health < 100) {
        double healthPercent = m_health / 100.0;
        QRectF healthBar(0, -10, m_width * healthPercent, 5);

        QColor healthColor = Qt::green;
        if (m_health < 50) healthColor = Qt::yellow;
        if (m_health < 25) healthColor = Qt::red;
    }
}

```

```

painter->setBrush(healthColor);
painter->setPen(Qt::NoPen);
painter->drawRect(healthBar);

painter->setPen(QPen(Qt::white, 1));
painter->drawRect(QRectF(0, -10, m_width, 5));
}
}

```

// player.cpp

```

#include "player.h"
#include "enemy.h"
#include <QGraphicsScene>
#include <QDebug>

```

```

Player::Player(QGraphicsItem* parent) : GameObject(parent) {
    m_width = 30;
    m_height = 30;
    m_health = 100;
    m_color = Qt::blue;
    m_speed = 5.0;
}

```

```

void Player::setMovingLeft(bool moving) { m_movingLeft = moving; }
void Player::setMovingRight(bool moving) { m_movingRight = moving; }
void Player::setMovingUp(bool moving) { m_movingUp = moving; }
void Player::setMovingDown(bool moving) { m_movingDown = moving; }
void Player::setShooting(bool shooting) { m_shooting = shooting; }

```

```

void Player::update() {
    updateMovement();
}

```

```

void Player::updateMovement() {
    double dx = 0, dy = 0;

    if (m_movingLeft) dx -= m_speed;
    if (m_movingRight) dx += m_speed;
    if (m_movingUp) dy -= m_speed;
    if (m_movingDown) dy += m_speed;

```

// Диагональное движение - нормализация

```

if (dx != 0 && dy != 0) {
    dx *= 0.7071; // 1/√2
    dy *= 0.7071;
}

```

```

move(dx, dy);

```

// Стрельба

```

if (m_shooting) {

```

```

        handleShooting();
    }
}

void Player::handleShooting() {
    // Получаем все враги на сцене
    QList<QGraphicsItem*> items = scene()->items();

    for (QGraphicsItem* item : items) {
        if (Enemy* enemy = dynamic_cast<Enemy*>(item)) {
            if (!enemy->isAlive()) continue;

            // Проверка расстояния
            double dx = enemy->x() - m_x;
            double dy = enemy->y() - m_y;
            double distance = qSqrt(dx*dx + dy*dy);

            // Стрельба на расстояние до 150 пикселей
            if (distance < 150.0) {
                enemy->takeDamage(5);
                if (!enemy->isAlive()) {
                    addScore(10);
                    emit scoreChanged(m_score);
                }
            }
        }
    }
}

// Сброс флага стрельбы
m_shooting = false;
}

void Player::onCollision(GameObject* other) {
    if (dynamic_cast<Enemy*>(other)) {
        takeDamage(10);
    }
}

// enemy.cpp
#include "enemy.h"
#include <QRandomGenerator>
#include <QDebug>

Enemy::Enemy(Type type, QGraphicsItem* parent)
: GameObject(parent), m_type(type) {
    // Настройка в зависимости от типа
    switch (type) {
        case Normal:
            m_width = 25;
            m_height = 25;
            m_health = 10;
            m_speed = 2.0;

```

```

        m_color = Qt::red;
        break;
    case Fast:
        m_width = 20;
        m_height = 20;
        m_health = 5;
        m_speed = 3.5;
        m_color = Qt::magenta;
        break;
    case Tank:
        m_width = 35;
        m_height = 35;
        m_health = 25;
        m_speed = 1.0;
        m_color = Qt::darkRed;
        break;
}

// Случайное начальное направление
m_vx = (QRandomGenerator::global()->bounded(200) - 100) / 100.0;
m_vy = (QRandomGenerator::global()->bounded(200) - 100) / 100.0;
}

void Enemy::initializeRandom(double sceneWidth, double sceneHeight) {
    setPosition(
        QRandomGenerator::global()->bounded(sceneWidth - m_width),
        QRandomGenerator::global()->bounded(sceneHeight - m_height)
    );
}

void Enemy::update() {
    updateMovement(800, 600); // TODO: передавать размеры сцены
}

void Enemy::updateMovement(double sceneWidth, double sceneHeight) {
    move(m_vx * m_speed, m_vy * m_speed);
    bounceOffBoundaries(sceneWidth, sceneHeight);
}

void Enemy::bounceOffBoundaries(double sceneWidth, double sceneHeight) {
    bool bounced = false;

    if (m_x < 0) {
        m_x = 0;
        m_vx = qAbs(m_vx);
        bounced = true;
    } else if (m_x > sceneWidth - m_width) {
        m_x = sceneWidth - m_width;
        m_vx = -qAbs(m_vx);
        bounced = true;
    }
}

```

```

if (m_y < 0) {
    m_y = 0;
    m_vy = qAbs(m_vy);
    bounced = true;
} else if (m_y > sceneHeight - m_height) {
    m_y = sceneHeight - m_height;
    m_vy = -qAbs(m_vy);
    bounced = true;
}

if (bounced) {
    setPosition(m_x, m_y);
}
}

void Enemy::onCollision(GameObject* other) {
    Q_UNUSED(other);
    // Враги могут реагировать на столкновения
}

```

Анализ рефакторинга

Сравнение "было" и "стало"

Аспект	Было	Стало	Шаблон рефакторинга
Размер класса MainWindow	300+ строк	~150 строк	Extract Class - выделены GameObject, Player, Enemy
Метод updateGame()	100+ строк, делает всё	20-30 строк, делегировать	Extract Method - разделение ответственности
Обработка столкновений	Вложенные условия	Отдельный метод checkCollisions()	Extract Method
Управление врагами	Векторы структур	Векторы unique_ptr<Enemy>	Replace Data Value with Object
Константы	Разбросаны по коду	Выделены в константы класса	Introduce Constant
Отрисовка	Смешана с логикой	Отдельный метод renderGameObjects()	Separate Concerns
Управление состоянием	Флаги в MainWindow	Состояния в GameManager	State Pattern
Обработка ввода	В keyPressEvent	Отдельный метод handleKeyInput()	Extract Method

Преимущества после рефакторинга:

1. **Улучшенная читаемость:** Код стал более понятным
2. **Повторное использование:** Классы можно использовать в других проектах
3. **Тестируемость:** Каждый компонент можно тестировать отдельно
4. **Расширяемость:** Легко добавлять новые типы врагов, оружия и т.д.
5. **Поддержка:** Легче находить и исправлять ошибки

Примеры использования инструментов рефакторинга в Qt Creator:

1. **Переименование (F2):**

сpp

// Было:

int hp;

// Выделить переменную, нажать F2:

int health; *// Стало*

2. **Извлечение функции (Ctrl+Alt+M):**

сpp

// Выделить код проверки столкновений:

```
if (player.x < enemy.x + ENEMY_SIZE &&
    player.x + PLAYER_SIZE > enemy.x &&
    player.y < enemy.y + ENEMY_SIZE &&
    player.y + PLAYER_SIZE > enemy.y) {
    // обработка столкновения
}
```

// Нажать Ctrl+Alt+M → создать функцию:

bool checkCollision(const GameObject& obj1, const GameObject& obj2);

3. **Извлечение класса:**

сpp

// Выделить структуры Player и Enemy

// Создать новые файлы .h/.cpp

// Использовать "Refactor → Move Definition to File"

Задание для самостоятельной работы

Вариант 1: Рефакторинг системы сохранения игры

Исходный код:

сpp

// game.cpp (фрагмент)

```
void Game::saveGame() {
    QFile file("savegame.dat");
    if (!file.open(QIODevice::WriteOnly)) return;
```

```
    QDataStream out(&file);
```

```
    out << playerX << playerY << playerHealth << playerScore;
```

```
    out << enemies.size();
```

```
    for (const auto& enemy : enemies) {
        out << enemy.x << enemy.y << enemy.health;
    }
```

```
    out << items.size();
```

```
    for (const auto& item : items) {
        out << item.type << item.x << item.y;
    }
```

```

        file.close();
    }

    void Game::loadGame() {
        QFile file("savegame.dat");
        if (!file.open(QIODevice::ReadOnly)) return;

        QDataStream in(&file);
        in >> playerX >> playerY >> playerHealth >> playerScore;

        int enemyCount;
        in >> enemyCount;
        enemies.clear();
        for (int i = 0; i < enemyCount; i++) {
            Enemy e;
            in >> e.x >> e.y >> e.health;
            enemies.push_back(e);
        }

        int itemCount;
        in >> itemCount;
        items.clear();
        for (int i = 0; i < itemCount; i++) {
            Item item;
            in >> item.type >> item.x >> item.y;
            items.push_back(item);
        }

        file.close();
    }

```

Задание:

1. Создайте класс `GameSaveSystem` с методами `save()` и `load()`
2. Реализуйте поддержку разных форматов сохранения (бинарный, JSON, XML)
3. Добавьте систему версионирования сохранений
4. Реализуйте обработку ошибок при загрузке
5. Добавьте возможность иметь несколько слотов сохранения

Вариант 2: Рефакторинг системы визуальных эффектов

Исходный код:

сpp

// effects.cpp (фрагмент)

```

void Game::createExplosion(double x, double y) {
    // Создание частиц
    for (int i = 0; i < 50; i++) {
        Particle p;
        p.x = x;
        p.y = y;
        p.vx = (rand() % 200 - 100) / 10.0;
        p.vy = (rand() % 200 - 100) / 10.0;
        p.life = 100;
        p.color = QColor(255, rand() % 256, 0);
        particles.push_back(p);
    }
}

```

```

    }

    // Создание дыма
    for (int i = 0; i < 20; i++) {
        Smoke s;
        s.x = x;
        s.y = y;
        s.vx = (rand() % 100 - 50) / 20.0;
        s.vy = (rand() % 100) / 20.0;
        s.life = 200;
        s.size = rand() % 10 + 5;
        smoke.push_back(s);
    }
}

void Game::updateEffects() {
    // Обновление частиц
    for (auto& p : particles) {
        p.x += p.vx;
        p.y += p.vy;
        p.vy += 0.1; // гравитация
        p.life--;
        if (p.life <= 0) {
            // удалить частицу
        }
    }

    // Обновление дыма
    for (auto& s : smoke) {
        s.x += s.vx;
        s.y += s.vy;
        s.life--;
        s.size += 0.1;
        if (s.life <= 0) {
            // удалить дым
        }
    }

    // Удаление мертвых эффектов
    particles.erase(std::remove_if(particles.begin(), particles.end(),
        [](const Particle& p) { return p.life <= 0; }), particles.end());
    smoke.erase(std::remove_if(smoke.begin(), smoke.end(),
        [](const Smoke& s) { return s.life <= 0; }), smoke.end());
}

```

Задание:

1. Создайте базовый класс `VisualEffect` с виртуальными методами `update()` и `draw()`
2. Реализуйте классы `ParticleEffect`, `SmokeEffect`, `ExplosionEffect`
3. Создайте менеджер эффектов `EffectManager`
4. Добавьте систему пулов объектов для оптимизации
5. Реализуйте различные типы эффектов (огонь, электричество, магия)

Критерии оценивания

Критерий	Баллы	Описание
Анализ кода	0-10	Глубина анализа проблем в исходном коде
Применение шаблонов	0-15	Корректное применение шаблонов рефакторинга
Качество рефакторинга	0-20	Улучшение читаемости, поддерживаемости, производительности
Документация изменений	0-10	Описание операций "было-стало-шаблон"
Тестирование	0-15	Проверка работоспособности после рефакторинга
Выводы и анализ	0-10	Качество анализа результатов рефакторинга
Использование Qt Creator	0-10	Использование встроенных инструментов рефакторинга
Оформление отчета	0-10	Структура, ясность, полнота отчета

Шкала оценивания:

- **5 (отлично):** 85-100 баллов - Полный анализ, качественный рефакторинг, все шаблоны применены корректно
- **4 (хорошо):** 70-84 балла - Хороший анализ, рефакторинг улучшил код, некоторые недочеты
- **3 (удовлетворительно):** 50-69 баллов - Базовый рефакторинг, код работает, но есть проблемы
- **2 (неудовлетворительно):** 0-49 баллов - Рефакторинг не выполнен или ухудшил код

Выводы

Рефакторинг - это важный навык для любого программиста. В ходе работы были изучены:

1. **Инструменты Qt Creator** для автоматического рефакторинга
2. **Основные шаблоны рефакторинга** и их применение
3. **Принципы хорошего кода:** SRP, DRY, KISS
4. **Техники улучшения структуры** без изменения поведения

Рефакторинг позволяет:

- Улучшить читаемость и поддерживаемость кода
- Упростить добавление новых функций
- Облегчить тестирование
- Уменьшить количество ошибок

Регулярный рефакторинг должен быть частью процесса разработки, а не одноразовой операцией. Лучше делать небольшие улучшения постоянно, чем один большой рефакторинг раз в год.

ПРАКТИЧЕСКАЯ РАБОТА №30. Разработка интерфейса пользователя.

Цель и задачи

Цель: Освоить теоретические основы проектирования пользовательского интерфейса, научиться разрабатывать три модели интерфейса.

Задачи:

1. Изучить модели программиста, пользователя и программную модель
2. Разработать концепцию интерфейса для заданной темы
3. Создать прототип интерфейса на бумаге
4. Оформить техническое задание по ГОСТ

Теоретический материал

Три модели интерфейса:

- **Модель программиста:** Техническая реализация, эффективность, структурность
- **Модель пользователя:** Интуитивность, простота освоения, соответствие ожиданиям
- **Программная модель:** Баланс между техническими возможностями и удобством использования

Критерии оценки интерфейса:

1. Простота освоения и запоминания
2. Скорость достижения результатов
3. Субъективная удовлетворенность пользователя

Пример решения

Тема: Интерфейс программы "Калькулятор матриц"

Модель программиста:

cpp

// Технические требования

- Использование Qt Widgets
- Отдельные классы для матричных операций
- Валидация ввода данных
- Обработка ошибок вычислений

Модель пользователя:

text

Пользователь ожидает:

1. Понятное поле для ввода матриц
2. Выбор операции (+, -, *, транспонирование)
3. Кнопка "Вычислить"
4. Поле для отображения результата
5. Возможность очистки полей

Программная модель:

- Главное окно с вкладками для разных операций
- Визуальное представление матриц в таблицах QTableWidgetItem
- Группировка кнопок по функциональности
- Интуитивная навигация

Задания по вариантам

Вариант 1: Спроектировать интерфейс программы "Редактор текста"

Вариант 2: Спроектировать интерфейс программы "База данных студентов"

Вариант 3: Спроектировать интерфейс программы "Графический редактор"

Критерии оценивания

Критерий	Баллы	Описание
Разработка трех моделей	0-3	Полнота и корректность моделей

Критерий	Баллы	Описание
Прототип интерфейса	0-3	Качество бумажного прототипа
Соответствие ГОСТ	0-2	Правильность оформления ТЗ
Обоснованность решений	0-2	Аргументация выбранных решений
Итого	0-10	

Шкала: 8-10 = 5, 6-7 = 4, 4-5 = 3, 0-3 = 2

ПРАКТИЧЕСКАЯ РАБОТА №31. Разработка интерфейса пользователя.

Цель и задачи

Цель: Научиться создавать основные элементы интерфейса с использованием Qt Widgets.

Задачи:

1. Освоить работу с основными виджетами Qt
2. Создать форму с различными элементами управления
3. Реализовать базовую компоновку интерфейса
4. Добавить простую обработку событий

Теоретический материал

Основные виджеты Qt:

1. **QWidget** - базовый класс всех виджетов
2. **QLabel** - отображение текста и изображений
3. **QPushButton** - кнопка с обработкой клика
4. **QLineEdit/QTextEdit** - поля ввода текста
5. **QComboBox** - выпадающий список
6. **QRadioButton/QCheckBox** - переключатели
7. **QSpinBox/QSlider** - элементы для выбора числовых значений

Компоновщики (Layouts):

- **QVBoxLayout** - вертикальное расположение
- **QHBoxLayout** - горизонтальное расположение
- **QGridLayout** - расположение в сетке
- **QFormLayout** - расположение в форме "метка-поле"

Пример решения

Программа: "Калькулятор ИМТ (индекс массы тела)"

cpp

// mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
```

```
#include <QMainWindow>
#include <QWidget>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QFormLayout>
```

```
class MainWindow : public QMainWindow {
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
```

```
private slots:
    void calculateBMI();
```

```
private:
    // Элементы интерфейса
    QLineEdit *heightEdit;
    QLineEdit *weightEdit;
    QLabel *resultLabel;
    QPushButton *calculateButton;
```

```

    void setupUI();
    void setupConnections();
};

#endif // MAINWINDOW_H

// mainwindow.cpp
#include "mainwindow.h"
#include <QMessageBox>
#include <cmath>

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent) {
    setWindowTitle("Калькулятор ИМТ");
    resize(400, 200);

    // Создание центрального виджета
    QWidget *centralWidget = new QWidget(this);
    setCentralWidget(centralWidget);

    // Создание элементов интерфейса
    heightEdit = new QLineEdit;
    weightEdit = new QLineEdit;
    resultLabel = new QLabel("Результат появится здесь");
    resultLabel->setAlignment(Qt::AlignCenter);
    calculateButton = new QPushButton("Рассчитать ИМТ");

    // Настройка валидации ввода
    heightEdit->setPlaceholderText("Рост в см (например: 175)");
    weightEdit->setPlaceholderText("Вес в кг (например: 70)");

    // Компоновка интерфейса
    QVBoxLayout *mainLayout = new QVBoxLayout(centralWidget);

    QFormLayout *formLayout = new QFormLayout;
    formLayout->addRow("Рост (см):", heightEdit);
    formLayout->addRow("Вес (кг):", weightEdit);

    mainLayout->addLayout(formLayout);
    mainLayout->addWidget(calculateButton);
    mainLayout->addWidget(resultLabel);
    mainLayout->addStretch();

    // Подключение сигналов
    setupConnections();
}

void MainWindow::setupConnections() {
    connect(calculateButton, &QPushButton::clicked,
           this, &MainWindow::calculateBMI);
}

```

```

void MainWindow::calculateBMI() {
    bool ok1, ok2;
    double height = heightEdit->text().toDouble(&ok1);
    double weight = weightEdit->text().toDouble(&ok2);

    if (!ok1 || !ok2 || height <= 0 || weight <= 0) {
        QMessageBox::warning(this, "Ошибка",
            "Введите корректные числовые значения!");
        return;
    }

    // Расчет ИМТ (рост в метрах)
    double heightM = height / 100.0;
    double bmi = weight / (heightM * heightM);

    // Определение категории
    QString category;
    if (bmi < 18.5) category = "Недостаточный вес";
    else if (bmi < 25) category = "Нормальный вес";
    else if (bmi < 30) category = "Избыточный вес";
    else category = "Ожирение";

    // Вывод результата
    resultLabel->setText(QString("ИМТ: %1\nКатегория: %2")
        .arg(bmi, 0, 'f', 1)
        .arg(category));
}

```

Задания по вариантам

Вариант 1: Создать программу "Конвертер валют"

Вариант 2: Создать программу "Калькулятор площадей фигур"

Вариант 3: Создать программу "Решение квадратного уравнения"

Критерии оценивания

Критерий	Баллы	Описание
Полнота интерфейса	0-3	Наличие всех необходимых элементов
Корректность компоновки	0-3	Правильное использование layout
Функциональность	0-2	Работоспособность программы
Качество кода	0-2	Чистота и комментирование кода
Итого	0-10	

ПРАКТИЧЕСКАЯ РАБОТА №32. Разработка интерфейса пользователя.

Цель и задачи

Цель: Научиться создавать сложные интерфейсы с диалоговыми окнами и меню.

Задачи:

1. Создать главное меню приложения
2. Реализовать диалоговые окна различных типов
3. Добавить панель инструментов и статусную строку
4. Организовать взаимодействие между окнами

Теоретический материал

Типы диалоговых окон в Qt:

1. **Модальные** - блокируют родительское окно
2. **Немодальные** - не блокируют родительское окно
3. **Всплывающие (Popup)** - поверх всех окон

Стандартные диалоги Qt:

- **QMessageBox** - информационные окна
- **FileDialog** - диалог выбора файлов
- **QColorDialog** - выбор цвета
- **QFontDialog** - выбор шрифта
- **QInputDialog** - ввод одного значения

Элементы меню:

- **QMenuBar** - строка меню
- **QMenu** - выпадающее меню
- **QAction** - действие меню
- **QToolBar** - панель инструментов
- **QStatusBar** - строка состояния

Пример решения

Программа: "Текстовый редактор с меню"

cpp

// texteditor.h

```
#ifndef TEXTEDITOR_H
#define TEXTEDITOR_H
```

```
#include <QMainWindow>
#include <QTextEdit>
#include <QMenuBar>
#include <QToolBar>
#include <QStatusBar>
#include <FileDialog>
#include <MessageBox>
#include <FontDialog>
#include <ColorDialog>
```

```
class TextEditor : public QMainWindow {
    Q_OBJECT
```

```
public:
```

```
    TextEditor(QWidget *parent = nullptr);
```

```
private slots:
```

```
    void newFile();
    void openFile();
    void saveFile();
```

```

void saveAsFile();
void changeFont();
void changeTextColor();
void changeBackgroundColor();
void about();
void updateStatusBar();

private:
    void setupUI();
    void setupMenu();
    void setupToolbar();
    void setupStatusBar();
    void setupConnections();

    // Элементы интерфейса
    QTextEdit *textEdit;
    QString currentFile;

    // Действия меню
    QAction *newAction;
    QAction *openAction;
    QAction *saveAction;
    QAction *saveAsAction;
    QAction *exitAction;
    QAction *fontAction;
    QAction *textColorAction;
    QAction *bgColorAction;
    QAction *aboutAction;
};

#endif // TEXTEDITOR_H

// texteditor.cpp
#include "texteditor.h"
#include <QFile>
#include <QTextStream>
#include <QCloseEvent>

TextEditor::TextEditor(QWidget *parent) : QMainWindow(parent) {
    setWindowTitle("Текстовый редактор");
    resize(800, 600);
    currentFile = "";

    setupUI();
    setupMenu();
    setupToolbar();
    setupStatusBar();
    setupConnections();
}

void TextEditor::setupUI() {
    textEdit = new QTextEdit(this);

```

```

setCentralWidget(textEdit);

// Подключение обновления статусной строки
connect(textEdit, &QTextEdit::cursorPositionChanged,
        this, &TextEditor::updateStatusBar);
}

void TextEditor::setupMenu() {
    // Меню "Файл"
    QMenu *fileMenu = menuBar()->addMenu("&Файл");

    newAction = fileMenu->addAction("&Новый");
    newAction->setShortcut(QKeySequence::New);

    openAction = fileMenu->addAction("&Открыть...");
    openAction->setShortcut(QKeySequence::Open);

    saveAction = fileMenu->addAction("&Сохранить");
    saveAction->setShortcut(QKeySequence::Save);

    saveAsAction = fileMenu->addAction("Сохранить &как...");
    saveAsAction->setShortcut(QKeySequence::SaveAs);

    fileMenu->addSeparator();

    exitAction = fileMenu->addAction("&Выход");
    exitAction->setShortcut(QKeySequence::Quit);

    // Меню "Формат"
    QMenu *formatMenu = menuBar()->addMenu("&Формат");

    fontAction = formatMenu->addAction("&Шрифт...");
    textColorAction = formatMenu->addAction("Цвет &текста...");
    bgColorAction = formatMenu->addAction("Цвет &фона...");

    // Меню "Справка"
    QMenu *helpMenu = menuBar()->addMenu("&Справка");

    aboutAction = helpMenu->addAction("&О программе");
}

void TextEditor::setupConnections() {
    // Подключение действий меню
    connect(newAction, &QAction::triggered, this, &TextEditor::newFile);
    connect(openAction, &QAction::triggered, this, &TextEditor::openFile);
    connect(saveAction, &QAction::triggered, this, &TextEditor::saveFile);
    connect(saveAsAction, &QAction::triggered, this, &TextEditor::saveAsFile);
    connect(exitAction, &QAction::triggered, this, &QWidget::close);
    connect(fontAction, &QAction::triggered, this, &TextEditor::changeFont);
    connect(textColorAction, &QAction::triggered,
            this, &TextEditor::changeTextColor);
    connect(bgColorAction, &QAction::triggered,

```

```

        this, &TextEditor::changeBackgroundColor);
connect(aboutAction, &QAction::triggered, this, &TextEditor::about);
}

void TextEditor::newFile() {
    if (textEdit->document()->isModified()) {
        QMessageBox::StandardButton reply;
        reply = QMessageBox::question(this, "Новый файл",
            "Текущий файл не сохранен. Создать новый файл?",
            QMessageBox::Yes | QMessageBox::No);

        if (reply == QMessageBox::No) return;
    }

    textEdit->clear();
    currentFile = "";
    setWindowTitle("Текстовый редактор - Новый файл");
}

void TextEditor::openFile() {
    QString fileName = QFileDialog::getOpenFileName(this,
        "Открыть файл", "", "Текстовые файлы (*.txt);Все файлы (*.*)");

    if (fileName.isEmpty()) return;

    QFile file(fileName);
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QMessageBox::warning(this, "Ошибка",
            "Не удалось открыть файл: " + file.errorString());
        return;
    }

    QTextStream in(&file);
    textEdit->setPlainText(in.readAll());
    file.close();

    currentFile = fileName;
    setWindowTitle("Текстовый редактор - " + QFileInfo(fileName).fileName());
}

void TextEditor::about() {
    QMessageBox::about(this, "О программе",
        "Текстовый редактор\n"
        "Версия 1.0\n\n"
        "Пример программы с меню и диалоговыми окнами\n"
        "Разработано для практической работы №32");
}

```

Задания по вариантам

Вариант 1: Создать программу "Просмотрщик изображений с меню"

Вариант 2: Создать программу "Аудиоплеер с меню"

Вариант 3: Создать программу "Календарь с диалогами"

Критерии оценивания

Критерий	Баллы	Описание
Создание меню	0-3	Полнота и функциональность меню
Диалоговые окна	0-3	Использование разных типов диалогов
Панель инструментов	0-2	Наличие и функциональность toolbar
Взаимодействие окон	0-2	Корректная передача данных между окнами
Итого	0-10	

ПРАКТИЧЕСКАЯ РАБОТА №33. Разработка интерфейса пользователя.

Цель и задачи

Цель: Научиться работать с табличными данными и списками в Qt.

Задачи:

1. Освоить работу с QTableWidgetItem
2. Изучить использование QListWidget и QTreeWidget
3. Реализовать сортировку и фильтрацию данных
4. Добавить контекстное меню для элементов

Теоретический материал

Табличные виджеты в Qt:

1. **QTableWidget** - таблица с готовой моделью
2. **QTableView + QStandardItemModel** - таблица с кастомной моделью
3. **QTreeWidget** - иерархическое представление данных
4. **QListWidget** - простой список элементов

Основные операции:

- Добавление/удаление строк и столбцов
- Сортировка данных
- Поиск и фильтрация
- Редактирование ячеек
- Контекстное меню (QMenu + QAction)

Пример решения

Программа: "База данных студентов"

cpp

// studentdatabase.h

```
#ifndef STUDENTDATABASE_H
#define STUDENTDATABASE_H
```

```
#include <QMainWindow>
#include <QTableWidget>
#include <QPushButton>
#include <QLineEdit>
#include <QComboBox>
#include <QMenu>
#include <QAction>
```

```
class StudentDatabase : public QMainWindow {
    Q_OBJECT
```

```
public:
    StudentDatabase(QWidget *parent = nullptr);
```

```
private slots:
    void addStudent();
    void removeStudent();
    void editStudent();
    void searchStudent();
    void sortTable(int column);
    void showContextMenu(const QPoint &pos);
    void saveToFile();
    void loadFromFile();
```

```
private:
```

```

void setupUI();
void setupTable();
void setupMenu();

// Элементы интерфейса
QTableWidget *table;
QLineEdit *nameEdit;
QLineEdit *groupIdEdit;
QComboBox *courseCombo;
QLineEdit *searchEdit;
QPushButton *addButton;
QPushButton *removeButton;
QPushButton *searchButton;

// Контекстное меню
QMenu *contextMenu;
QAction *editAction;
QAction *deleteAction;

// Вспомогательные методы
bool validateInput();
void clearInputFields();
};

#endif // STUDENTDATABASE_H

// studentdatabase.cpp
#include "studentdatabase.h"
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QFormLayout>
#include <QHeaderView>
#include <QMessageBox>
#include <QFile>
#include <QTextStream>
#include <QFileDialog>

StudentDatabase::StudentDatabase(QWidget *parent) : QMainWindow(parent) {
    setWindowTitle("База данных студентов");
    resize(900, 600);

    setupUI();
    setupTable();
    setupMenu();
}

void StudentDatabase::setupUI() {
    // Создание центрального виджета
    QWidget *centralWidget = new QWidget(this);
    setCentralWidget(centralWidget);

    QVBoxLayout *mainLayout = new QVBoxLayout(centralWidget);

```

```

// Форма для ввода данных
QWidget *inputWidget = new QWidget;
QHBoxLayout *inputLayout = new QHBoxLayout(inputWidget);

// Поля для ввода
nameEdit = new QLineEdit;
nameEdit->setPlaceholderText("ФИО студента");

groupIdEdit = new QLineEdit;
groupIdEdit->setPlaceholderText("Номер группы");

courseCombo = new QComboBox;
for (int i = 1; i <= 6; i++) {
    courseCombo->addItem(QString::number(i));
}

// Кнопки
addButton = new QPushButton("Добавить");
removeButton = new QPushButton("Удалить выделенное");
searchButton = new QPushButton("Поиск");

// Поле поиска
searchEdit = new QLineEdit;
searchEdit->setPlaceholderText("Поиск...");

// Добавление элементов в layout
inputLayout->addWidget(new QLabel("ФИО:"));
inputLayout->addWidget(nameEdit);
inputLayout->addWidget(new QLabel("Группа:"));
inputLayout->addWidget(groupIdEdit);
inputLayout->addWidget(new QLabel("Курс:"));
inputLayout->addWidget(courseCombo);
inputLayout->addWidget(addButton);
inputLayout->addWidget(removeButton);
inputLayout->addStretch();
inputLayout->addWidget(searchEdit);
inputLayout->addWidget(searchButton);

// Создание таблицы
table = new QTableWidgetItem;
table->setColumnCount(5);
table->setHorizontalHeaderLabels({"ID", "ФИО", "Группа", "Курс", "Дата добавления"});
table->setSelectionBehavior(QAbstractItemView::SelectRows);
table->setEditTriggers(QAbstractItemView::NoEditTriggers);

// Подключение сортировки
table->setSortingEnabled(true);
connect(table->horizontalHeader(), &QHeaderView::sectionClicked,
        this, &StudentDatabase::sortTable);

// Подключение контекстного меню

```

```

table->setContextMenuPolicy(Qt::CustomContextMenu);
connect(table, &QTableWidget::customContextMenuRequested,
        this, &StudentDatabase::showContextMenu);

// Компоновка
mainLayout->addWidget(inputWidget);
mainLayout->addWidget(table);

// Подключение сигналов
connect(addButton, &QPushButton::clicked, this, &StudentDatabase::addStudent);
connect(removeButton, &QPushButton::clicked, this, &StudentDatabase::removeStudent);
connect(searchButton, &QPushButton::clicked, this, &StudentDatabase::searchStudent);
connect(searchEdit, &QLineEdit::returnPressed, this, &StudentDatabase::searchStudent);
}

void StudentDatabase::addStudent() {
    if (!validateInput()) return;

    int row = table->rowCount();
    table->insertRow(row);

    // ID (автоинкремент)
    table->setItem(row, 0, new QTableWidgetItem(QString::number(row + 1)));

    // ФИО
    table->setItem(row, 1, new QTableWidgetItem(nameEdit->text()));

    // Группа
    table->setItem(row, 2, new QTableWidgetItem(groupIdEdit->text()));

    // Курс
    table->setItem(row, 3, new QTableWidgetItem(courseCombo->currentText()));

    // Дата добавления
    table->setItem(row, 4,
        new QTableWidgetItem(QDateTime::currentDateTime().toString("dd.MM.yyyy
hh:mm")));

    clearInputFields();
}

void StudentDatabase::searchStudent() {
    QString searchText = searchEdit->text().toLower();

    for (int row = 0; row < table->rowCount(); row++) {
        bool match = false;

        for (int col = 0; col < table->columnCount(); col++) {
            QTableWidgetItem *item = table->item(row, col);
            if (item && item->text().toLower().contains(searchText)) {
                match = true;
                break;
            }
        }
    }
}

```

```

    }
}

// Показать/скрыть строку в зависимости от результата поиска
table->setRowHidden(row, !match && !searchText.isEmpty());
}
}

```

Задания по вариантам

Вариант 1: Создать программу "Каталог книг"

Вариант 2: Создать программу "Учет расходов"

Вариант 3: Создать программу "Планировщик задач"

Критерии оценивания

Критерий	Баллы	Описание
Работа с таблицей	0-3	Добавление, удаление, редактирование строк
Сортировка и поиск	0-3	Реализация функций сортировки и поиска
Контекстное меню	0-2	Наличие и функциональность контекстного меню
Экспорт/импорт	0-2	Сохранение и загрузка данных
Итого	0-10	

ПРАКТИЧЕСКАЯ РАБОТА №34. Разработка интерфейса пользователя.

Цель и задачи

Цель: Научиться создавать сложные интерфейсы с графическими элементами и диаграммами.

Задачи:

1. Освоить QPainter для рисования графики
2. Изучить QChart для создания диаграмм
3. Реализовать перетаскивание элементов (Drag & Drop)
4. Создать анимацию интерфейсных элементов

Теоретический материал

Графика в Qt:

1. **QPainter** - рисование примитивов (линии, прямоугольники, окружности)
2. **QPen/QBrush** - настройка стилей рисования
3. **QPixmap/QImage** - работа с растровыми изображениями
4. **QTransform** - преобразования (масштабирование, вращение)

Диаграммы (Qt Charts):

- Линейные графики
- Столбчатые диаграммы
- Круговые диаграммы
- Диаграммы рассеяния

Перетаскивание (Drag & Drop):

- QDrag - механизм перетаскивания
- QMimeData - передача данных
- Обработка событий dragEnterEvent, dropEvent

Пример решения

Программа: "Анализатор данных с графиками"

cpp

```
// datavisualizer.h
```

```
#ifndef DATAVISUALIZER_H
```

```
#define DATAVISUALIZER_H
```

```
#include <QMainWindow>
```

```
#include <QtCharts>
```

```
#include <QChartView>
```

```
#include <QLineSeries>
```

```
#include <QBarSeries>
```

```
#include <QBarSet>
```

```
#include <QPieSeries>
```

```
#include <QValueAxis>
```

```
#include <QCategoryAxis>
```

```
#include <QTableWidget>
```

```
#include <QGroupBox>
```

```
#include <QRadioButton>
```

```
QT_CHARTS_USE_NAMESPACE
```

```
class DataVisualizer : public QMainWindow {  
    Q_OBJECT
```

```
public:
```

```
    DataVisualizer(QWidget *parent = nullptr);
```

```

private slots:
    void updateChart();
    void generateRandomData();
    void saveChart();
    void changeChartType();

private:
    void setupUI();
    void setupCharts();
    void setupDataTable();

    // Элементы интерфейса
    QChartView *chartView;
    QTableWidget *dataTable;

    // Чарты
    QChart *chart;
    QLineSeries *lineSeries;
    QBarSeries *barSeries;
    QPieSeries *pieSeries;

    // Элементы управления
    QGroupBox *chartTypeGroup;
    QRadioButton *lineChartRadio;
    QRadioButton *barChartRadio;
    QRadioButton *pieChartRadio;

    // Данные
    QVector<double> xData;
    QVector<double> yData;

    // Вспомогательные методы
    void createLineChart();
    void createBarChart();
    void createPieChart();
    void populateTable();
};

#endif // DATAVISUALIZER_H

// datavisualizer.cpp
#include "datavisualizer.h"
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QPushButton>
#include <QSplitter>
#include <QFileDialog>
#include <QPainter>

DataVisualizer::DataVisualizer(QWidget *parent) : QMainWindow(parent) {
    setWindowTitle("Анализатор данных с графиками");
    resize(1200, 700);
}

```

```

// Генерация тестовых данных
generateRandomData();

setupUI();
setupCharts();
setupDataTable();

// Показать линейный график по умолчанию
createLineChart();
}

void DataVisualizer::setupUI() {
    // Создание центрального виджета
    QWidget *centralWidget = new QWidget(this);
    setCentralWidget(centralWidget);

    QVBoxLayout *mainLayout = new QVBoxLayout(centralWidget);

    // Панель управления
    QWidget *controlPanel = new QWidget;
    QHBoxLayout *controlLayout = new QHBoxLayout(controlPanel);

    // Группа выбора типа диаграммы
    chartTypeGroup = new QGroupBox("Тип диаграммы");
    QHBoxLayout *typeLayout = new QHBoxLayout;

    lineChartRadio = new QRadioButton("Линейный график");
    barChartRadio = new QRadioButton("Столбчатая диаграмма");
    pieChartRadio = new QRadioButton("Круговая диаграмма");

    lineChartRadio->setChecked(true);

    typeLayout->addWidget(lineChartRadio);
    typeLayout->addWidget(barChartRadio);
    typeLayout->addWidget(pieChartRadio);
    chartTypeGroup->setLayout(typeLayout);

    // Кнопки управления
    QPushButton *updateButton = new QPushButton("Обновить");
    QPushButton *randomButton = new QPushButton("Случайные данные");
    QPushButton *saveButton = new QPushButton("Сохранить график");

    controlLayout->addWidget(chartTypeGroup);
    controlLayout->addStretch();
    controlLayout->addWidget(updateButton);
    controlLayout->addWidget(randomButton);
    controlLayout->addWidget(saveButton);

    // Разделитель для графика и таблицы
    QSplitter *splitter = new QSplitter(Qt::Horizontal);

```

```

// Виджет для графика
chartView = new QChartView;
chartView->setRenderHint(QPainter::Antialiasing);

// Таблица данных
dataTable = new QTableWidgetItem;
dataTable->setColumnCount(2);
dataTable->setHorizontalHeaderLabels({"X", "Y"});

splitter->addWidget(chartView);
splitter->addWidget(dataTable);
splitter->setSizes({800, 400});

// Компоновка
mainLayout->addWidget(controlPanel);
mainLayout->addWidget(splitter);

// Подключение сигналов
connect(updateButton, &QPushButton::clicked, this, &DataVisualizer::updateChart);
connect(randomButton, &QPushButton::clicked, this,
&DataVisualizer::generateRandomData);
connect(saveButton, &QPushButton::clicked, this, &DataVisualizer::saveChart);
connect(lineChartRadio, &QRadioButton::clicked, this, &DataVisualizer::changeChartType);
connect(barChartRadio, &QRadioButton::clicked, this, &DataVisualizer::changeChartType);
connect(pieChartRadio, &QRadioButton::clicked, this, &DataVisualizer::changeChartType);
}

void DataVisualizer::createLineChart() {
    if (chart) delete chart;

    chart = new QChart;
    chart->setTitle("Линейный график данных");

    lineSeries = new QLineSeries;
    lineSeries->setName("Данные");

    // Добавление точек
    for (int i = 0; i < xData.size(); i++) {
        lineSeries->append(xData[i], yData[i]);
    }

    chart->addSeries(lineSeries);

    // Настройка осей
    QValueAxis *axisX = new QValueAxis;
    axisX->setTitleText("Ось X");
    axisX->setLabelFormat("%.1f");

    QValueAxis *axisY = new QValueAxis;
    axisY->setTitleText("Ось Y");
    axisY->setLabelFormat("%.1f");

```

```

chart->addAxis(axisX, Qt::AlignBottom);
chart->addAxis(axisY, Qt::AlignLeft);

lineSeries->attachAxis(axisX);
lineSeries->attachAxis(axisY);

chartView->setChart(chart);
}

void DataVisualizer::saveChart() {
    QString fileName = QFileDialog::getSaveFileName(this,
        "Сохранить график", "", "Изображения (*.png *.jpg *.bmp)");

    if (fileName.isEmpty()) return;

    QPixmap pixmap = chartView->grab();
    pixmap.save(fileName);

    QMessageBox::information(this, "Сохранение",
        "График успешно сохранен в файл: " + fileName);
}

```

Задания по вариантам

Вариант 1: Создать программу "Графический редактор фигур"

Вариант 2: Создать программу "Анализатор статистики"

Вариант 3: Создать программу "Построитель графиков функций"

Критерии оценивания

Критерий	Баллы	Описание
Графические элементы	0-3	Качество и разнообразие графики
Диаграммы и графики	0-3	Использование Qt Charts
Интерактивность	0-2	Drag & Drop, анимация
Экспорт результатов	0-2	Сохранение графиков и данных
Итого	0-10	

Общие рекомендации для всех работ

Структура отчета по каждой работе:

1. Титульный лист (по ГОСТ 19.104-78)
2. Цель и задачи работы
3. Теоретическая часть

Практическая работа 30-34. Разработка интерфейсов (Альтернатива)

Этап работы	Ключевые задачи	Рекомендуемое время
1. Анализ и проектирование	Разработка моделей программиста, пользователя и программной модели.	2 часа
2. Подготовка Qt-проекта	Создание структуры, главного окна, навигации.	1.5 часа
3. Реализация теоретического блока	Создание системы уроков с текстом, изображениями.	2 часа
4. Реализация практического блока	Добавление интерактивных задач с проверкой кода.	2 часа
5. Реализация контролирующего блока	Создание системы тестирования с вопросами и оценкой.	1.5 часа
6. Завершение и тестирование	Финальная сборка, проверка работы всех модулей.	1 час

Далее мы подробно разберем каждый этап.

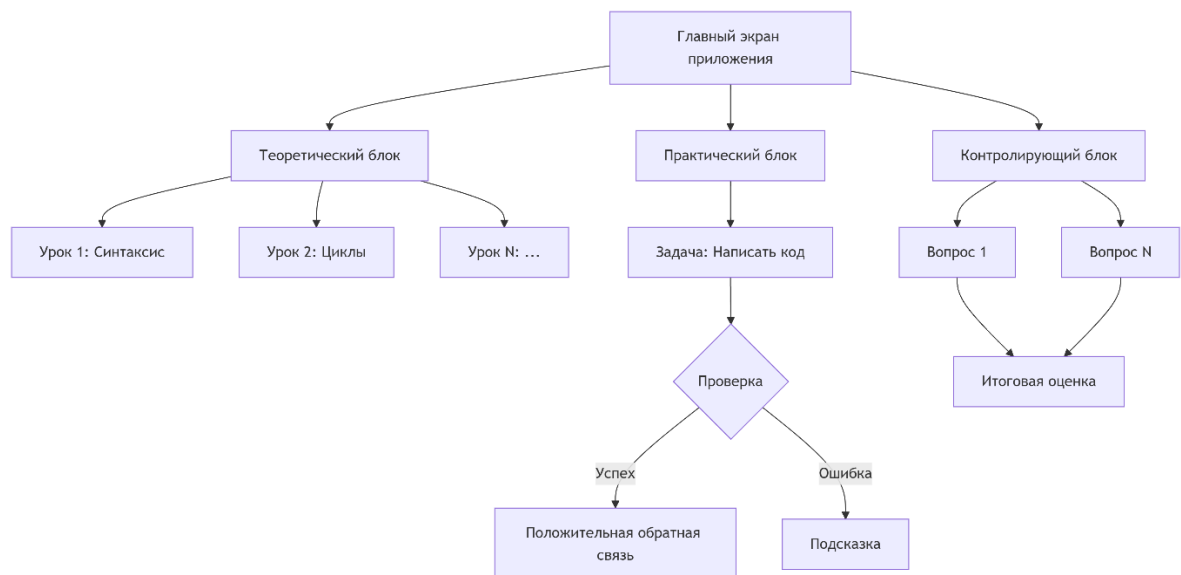
Часть 1: Разработка трех моделей интерфейса

Перед написанием кода необходимо спроектировать интерфейс. Основой является согласование трёх моделей.

Модель	Фокус и цели	Ключевые особенности для нашего приложения
Модель программиста	Техническая реализация, эффективность кода, устойчивость архитектуры.	— Четкое разделение на модули (учебник, практика, тест). — Использование готовых компонентов Qt. — Легкость добавления нового контента.
Модель пользователя	Удобство освоения, интуитивность, скорость достижения целей.	— Простая и понятная навигация между разделами. — Возможность сразу применять теорию на практике. — Понятные инструкции и обратная связь.
Программная модель (реализованный интерфейс)	Баланс между техническими возможностями и удобством	— Единый главный экран с четкими разделами. — Последовательный путь: теория практика контроль.

Модель	Фокус и цели	Ключевые особенности для нашего приложения
	пользователя.	— Визуальная простота и минимум отвлекающих элементов.

На основании этих моделей логика взаимодействия с приложением может быть представлена следующей схемой:



Часть 2: Создание приложения на C++/Qt

Для этого проекта используется объектно-ориентированный подход и фреймворк Qt. Он предоставляет все необходимые компоненты для создания интуитивного пользовательского интерфейса.

1. Структура проекта и главное окно

Сначала создается базовый каркас приложения с системой навигации.

cpp

// mainwindow.h - Заголовочный файл главного окна

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
```

```
#include <QMainWindow>
#include <QStackedWidget>
#include <QListWidget>
```

```
QT_BEGIN_NAMESPACE
```

```
class QPushButton;
```

```
class QLabel;
```

```
class QTextEdit;
```

```
QT_END_NAMESPACE
```

```
class MainWindow : public QMainWindow {
    Q_OBJECT
```

```
public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
```

```
private slots:
    void showTheory();
    void showPractice();
    void showTest();
    void loadLesson(int index);
    void checkPracticeCode();
    void submitTest();
```

```
private:
    void setupUI();
    void setupConnections();
    void loadTheoryContent(int lessonId);
    void loadPracticeTask(int taskId);
    void loadTestQuestion(int questionId);
```

```
// Центральные виджеты
```

```
QStackedWidget *centralStack;
QListWidget *lessonList;
```

```
// Виджеты для теоретического блока
```

```
QWidget *theoryPage;
QTextEdit *theoryText;
QLabel *theoryImageLabel;
```

```
// Виджеты для практического блока
```

```
QWidget *practicePage;
QTextEdit *practiceTaskDescription;
QTextEdit *practiceCodeEditor;
QLabel *practiceResultLabel;
```

```
// Виджеты для контролирующего блока
```

```
QWidget *testPage;
QLabel *testQuestionLabel;
QList<QRadioButton*> testAnswerButtons;
QLabel *testScoreLabel;
```

```
};
```

```
#endif // MAINWINDOW_H
```

2. Реализация главного окна

Основная логика инициализации и навигации находится в файле реализации.

cpp

```
// mainwindow.cpp - Реализация главного окна
```

```
#include "mainwindow.h"
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QPushButton>
#include <QLabel>
#include <QTextEdit>
```

```

#include <QListWidget>
#include <QStackedWidget>
#include <QRadioButton>
#include <QButtonGroup>
#include <QMessageBox>
#include <QFile>
#include <QDebug>

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent) {
    setWindowTitle("Обучающе-контролирующая программа по C++");
    resize(1024, 768);

    // Создание центрального стека для страниц
    centralStack = new QStackedWidget(this);
    setCentralWidget(centralStack);

    // Создание панели навигации
    QWidget *navWidget = new QWidget;
    QVBoxLayout *navLayout = new QVBoxLayout(navWidget);

    QPushButton *theoryBtn = new QPushButton("Теория");
    QPushButton *practiceBtn = new QPushButton("Практика");
    QPushButton *testBtn = new QPushButton("Тест");

    navLayout->addWidget(theoryBtn);
    navLayout->addWidget(practiceBtn);
    navLayout->addWidget(testBtn);
    navLayout->addStretch();

    // Создание контентной области
    QWidget *contentWidget = new QWidget;
    QHBoxLayout *mainLayout = new QHBoxLayout(contentWidget);
    mainLayout->addWidget(navWidget, 1);
    mainLayout->addWidget(centralStack, 4);

    QWidget *mainWidget = new QWidget;
    QVBoxLayout *finalLayout = new QVBoxLayout(mainWidget);
    finalLayout->addWidget(contentWidget);
    setCentralWidget(mainWidget);

    // Инициализация страниц
    setupTheoryPage();
    setupPracticePage();
    setupTestPage();

    // Подключение сигналов
    connect(theoryBtn, &QPushButton::clicked, this, &MainWindow::showTheory);
    connect(practiceBtn, &QPushButton::clicked, this, &MainWindow::showPractice);
    connect(testBtn, &QPushButton::clicked, this, &MainWindow::showTest);

    // Показ начальной страницы
    showTheory();

```

```
}
```

3. Реализация теоретического блока

Этот блок представляет собой интерактивный учебник с уроками.

сpp

// theorypage.cpp - Реализация теоретического блока

```
#include "theorypage.h"
```

```
#include <QFile>
```

```
#include <QTextStream>
```

```
TheoryPage::TheoryPage(QWidget *parent) : QWidget(parent) {  
    QVBoxLayout *mainLayout = new QVBoxLayout(this);
```

// Список уроков

```
lessonList = new QListWidget;
```

```
QStringList lessons = {
```

```
    "Урок 1: Основы синтаксиса C++",
```

```
    "Урок 2: Переменные и типы данных",
```

```
    "Урок 3: Управляющие конструкции",
```

```
    "Урок 4: Функции",
```

```
    "Урок 5: Классы и объекты"
```

```
};
```

```
lessonList->addItem(lessons);
```

// Область отображения контента урока

```
QSplitter *splitter = new QSplitter(Qt::Horizontal);
```

```
splitter->addWidget(lessonList);
```

```
QWidget *contentWidget = new QWidget;
```

```
QVBoxLayout *contentLayout = new QVBoxLayout(contentWidget);
```

```
lessonTitle = new QLabel;
```

```
lessonTitle->setStyleSheet("font-size: 18px; font-weight: bold;");
```

```
lessonContent = new QTextEdit;
```

```
lessonContent->setReadOnly(true);
```

```
contentLayout->addWidget(lessonTitle);
```

```
contentLayout->addWidget(lessonContent);
```

```
splitter->addWidget(contentWidget);
```

```
mainLayout->addWidget(splitter);
```

// Подключение выбора урока

```
connect(lessonList, &QListWidget::currentRowChanged,
```

```
    this, &TheoryPage::loadLesson);
```

```
}
```

```
void TheoryPage::loadLesson(int index) {
```

```
    QString filePath = QString(":/lessons/lesson%1.txt").arg(index + 1);
```

```
    QFile file(filePath);
```

```
    if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
```

```

    QTextStream in(&file);
    QString content = in.readAll();

    QStringList parts = content.split("|TITLE|");
    if (parts.size() > 1) {
        lessonTitle->setText(parts[0].trimmed());
        lessonContent->setHtml(parts[1]);
    }
    file.close();
}
}

```

4. Реализация практического блока

Практический блок позволяет сразу применять полученные знания.

cpp

// practicepage.cpp - Реализация практического блока

```

#include "practicepage.h"
#include <QProcess>
#include <QMessageBox>

```

```

PracticePage::PracticePage(QWidget *parent) : QWidget(parent) {
    QVBoxLayout *mainLayout = new QVBoxLayout(this);

```

// Описание задачи

```

taskDescription = new QTextEdit;
taskDescription->setReadOnly(true);
taskDescription->setHtml(
    "<h3>Задача: Сумма чисел</h3>"
    "<p>Напишите программу, которая вычисляет сумму чисел от 1 до N.</p>"
    "<p>Входные данные: число N (целое, положительное).</p>"
    "<p>Выходные данные: сумма чисел от 1 до N.</p>"
    "<p><b>Пример:</b> Для N=5, результат: 15</p>"
);

```

// Редактор кода

```

codeEditor = new QTextEdit;
codeEditor->setPlaceholderText("Введите ваш код здесь...");

```

// Предзаполненный шаблон кода

```

QString templateCode = "#include <iostream>\n\n"
    "int main() {\n"
    "    int n, sum = 0;\n"
    "    std::cin >> n;\n"
    "    // Ваш код здесь\n"
    "    std::cout << sum << std::endl;\n"
    "    return 0;\n"
    "}";
codeEditor->setPlainText(templateCode);

```

// Кнопки управления

```

QHBoxLayout *buttonLayout = new QHBoxLayout;
QPushButton *runButton = new QPushButton("Запустить код");
QPushButton *resetButton = new QPushButton("Сбросить");

```

```

QPushButton *showHintButton = new QPushButton("Подсказка");

buttonLayout->addWidget(runButton);
buttonLayout->addWidget(resetButton);
buttonLayout->addWidget(showHintButton);

// Область вывода результата
resultOutput = new QTextEdit;
resultOutput->setReadOnly(true);
resultOutput->setMaximumHeight(150);

mainLayout->addWidget(taskDescription);
mainLayout->addWidget(new QLabel("Редактор кода:"));
mainLayout->addWidget(codeEditor);
mainLayout->addLayout(buttonLayout);
mainLayout->addWidget(new QLabel("Результат выполнения:"));
mainLayout->addWidget(resultOutput);

// Подключение сигналов
connect(runButton, &QPushButton::clicked, this, &PracticePage::runCode);
connect(resetButton, &QPushButton::clicked, [this]() {
    codeEditor->clear();
});
connect(showHintButton, &QPushButton::clicked, []() {
    QMessageBox::information(nullptr, "Подсказка",
        "Используйте цикл for для перебора чисел от 1 до N");
});
}

```

5. Реализация контролирующего блока

Контролирующий блок оценивает усвоение материала.

сpp

// testpage.cpp - Реализация контролирующего блока

```

#include "testpage.h"
#include <QRadioButton>
#include <QButtonGroup>

TestPage::TestPage(QWidget *parent) : QWidget(parent), currentQuestion(0), score(0) {
    QVBoxLayout *mainLayout = new QVBoxLayout(this);

    // Вопрос
    questionLabel = new QLabel;
    questionLabel->setWordWrap(true);
    questionLabel->setStyleSheet("font-size: 16px;");

    // Варианты ответов
    answerGroup = new QButtonGroup(this);
    for (int i = 0; i < 4; ++i) {
        QRadioButton *radio = new QRadioButton;
        answerButtons.append(radio);
        answerGroup->addButton(radio, i);
        mainLayout->addWidget(radio);
    }
}

```

// Кнопки навигации

```
QHBoxLayout *navLayout = new QHBoxLayout;  
prevButton = new QPushButton("Назад");  
nextButton = new QPushButton("Далее");  
submitButton = new QPushButton("Завершить тест");  
submitButton->setEnabled(false);
```

```
navLayout->addWidget(prevButton);  
navLayout->addWidget(nextButton);  
navLayout->addWidget(submitButton);
```

// Отображение прогресса и результата

```
progressLabel = new QLabel;  
resultLabel = new QLabel;  
resultLabel->setVisible(false);
```

```
mainLayout->addWidget(questionLabel);  
mainLayout->addStretch(1);  
mainLayout->addLayout(navLayout);  
mainLayout->addWidget(progressLabel);  
mainLayout->addWidget(resultLabel);
```

// Инициализация вопросов

```
initializeQuestions();  
showQuestion(0);
```

// Подключение сигналов

```
connect(prevButton, &QPushButton::clicked, [this]() {  
    saveAnswer();  
    showQuestion(currentQuestion - 1);  
});
```

```
connect(nextButton, &QPushButton::clicked, [this]() {  
    saveAnswer();  
    showQuestion(currentQuestion + 1);  
});
```

```
connect(submitButton, &QPushButton::clicked, this, &TestPage::submitTest);  
}
```

```
void TestPage::submitTest() {  
    saveAnswer();
```

// Вычисление результата

```
score = 0;  
for (int i = 0; i < questions.size(); ++i) {  
    if (userAnswers[i] == questions[i].correctAnswer) {  
        score++;  
    }  
}
```

```

double percentage = (score * 100.0) / questions.size();
QString resultText = QString("Тест завершен!\n"
    "Правильных ответов: %1 из %2\n"
    "Оценка: %3%")
    .arg(score)
    .arg(questions.size())
    .arg(percentage, 0, 'f', 1);

if (percentage >= 80) resultText += "\n\nОтлично!";
else if (percentage >= 60) resultText += "\n\nХорошо!";
else resultText += "\n\nПовторите материал.";

resultLabel->setText(resultText);
resultLabel->setVisible(true);
submitButton->setEnabled(false);
}

```

Часть 3: Оформление отчета по ЕСПД (ГОСТ)

Отчет должен быть оформлен в соответствии с требованиями Единой системы программной документации. Вот ключевые разделы:

Раздел отчета	Содержание и требования ГОСТ
Титульный лист	Название работы, ФИО исполнителя, группа, год (ГОСТ 19.104-78)
Аннотация	Краткое описание целей, задач и результатов работы (ГОСТ 19.105-78)
Содержание	Структура отчета с указанием страниц
Основная часть	1. Введение: цель и задачи (со ссылкой на ваше задание). 2. Разработка моделей интерфейса: подробное описание трех моделей с таблицами или схемами. 3. Проектирование программы: описание структуры, используемых классов Qt. 4. Реализация: листинги ключевых фрагментов кода с комментариями. 5. Инструкция пользователя: как работать с программой. 6. Результаты тестирования.
Заключение	Выводы о проделанной работе, достижении цели, трудностях и возможностях улучшения
Приложения	Полные листинги кода, диаграммы классов (по желанию)
Список литературы	Использованные источники (ГОСТ 7.0.100—2018)

Для защиты подготовьте:

1. **Работающую программу** для демонстрации.
2. **Распечатанный отчет**, оформленный по ГОСТ.

3. **Краткую презентацию** (3-5 минут) с акцентами на: как вы разрабатывали модели, основные архитектурные решения, пример работы каждого блока программы. Этот план и примеры кода дают полную основу для выполнения работы. Начинайте с проектирования моделей — это основа всего последующего кода. При возникновении вопросов по конкретному блоку (например, как реализовать компиляцию кода в практическом задании) обращайтесь за уточнениями.

ПРАКТИЧЕСКАЯ РАБОТА №35-36. Создание приложения с БД

Цель и задачи

Цель работы: Освоить принципы разработки приложений с использованием баз данных в Qt, научиться создавать интерфейсы для работы с SQL-данными.

Задачи:

1. Изучить компоненты Qt для работы с базами данных
2. Создать базу данных с использованием SQLite
3. Разработать пользовательский интерфейс для управления данными
4. Реализовать операции CRUD (Create, Read, Update, Delete)
5. Настроить валидацию ввода и обработку ошибок

Теоретический материал

Архитектура работы с БД в Qt

Модель работы с базами данных в Qt:

text

Qt Application → SQL Driver → Database (SQLite/MySQL/PostgreSQL)

↑

SQL Model Classes (QSqlQueryModel, QSqlTableModel, QSqlRelationalTableModel)

↑

View Classes (QTableView, QListView, QTreeView)

Основные классы Qt для работы с БД:

Класс	Назначение	Описание
QSqlDatabase	Подключение к БД	Представляет соединение с базой данных
QSqlQuery	Выполнение запросов	Исполнение SQL-запросов и навигация по результатам
QSqlTableModel	Модель для таблицы	Модель для работы с одной таблицей БД
QSqlRelationalTableModel	Реляционная модель	Модель с поддержкой внешних ключей
QSqlQueryModel	Модель для запросов	Модель для произвольных SQL-запросов
QSqlError	Обработка ошибок	Информация об ошибках при работе с БД

Подключение к различным СУБД

cpp

// SQLite (встроенная, не требует установки сервера)

```
QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");  
db.setDatabaseName("mydatabase.db");
```

// MySQL (требует установки драйвера)

```
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");  
db.setHostName("localhost");  
db.setDatabaseName("mydatabase");  
db.setUserName("user");  
db.setPassword("password");
```

// PostgreSQL

```
QSqlDatabase db = QSqlDatabase::addDatabase("QPSQL");  
db.setHostName("localhost");  
db.setDatabaseName("mydatabase");  
db.setUserName("user");  
db.setPassword("password");
```

Основные SQL-операции

CRUD операции:

- **Create (INSERT)** - создание новых записей
- **Read (SELECT)** - чтение данных
- **Update (UPDATE)** - обновление существующих записей
- **Delete (DELETE)** - удаление записей

Пример решения: Приложение "Учет студентов"

1. Создание базы данных

sql

-- Создание таблицы студентов

```
CREATE TABLE IF NOT EXISTS students (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    first_name TEXT NOT NULL,  
    last_name TEXT NOT NULL,  
    group_number TEXT NOT NULL,  
    birth_date DATE,  
    average_score REAL CHECK(average_score >= 0 AND average_score <= 100),  
    email TEXT UNIQUE,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

-- Создание таблицы предметов

```
CREATE TABLE IF NOT EXISTS subjects (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    name TEXT NOT NULL UNIQUE,  
    hours INTEGER DEFAULT 36  
);
```

-- Создание таблицы оценок

```
CREATE TABLE IF NOT EXISTS grades (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    student_id INTEGER,  
    subject_id INTEGER,  
    grade INTEGER CHECK(grade >= 1 AND grade <= 5),  
    exam_date DATE,  
    FOREIGN KEY (student_id) REFERENCES students(id) ON DELETE CASCADE,  
    FOREIGN KEY (subject_id) REFERENCES subjects(id) ON DELETE CASCADE  
);
```

-- Создание индексов для ускорения поиска

```
CREATE INDEX idx_students_group ON students(group_number);  
CREATE INDEX idx_grades_student ON grades(student_id);  
CREATE INDEX idx_grades_subject ON grades(subject_id);
```

2. Класс для работы с базой данных

cpp

```

// database.h
#ifndef DATABASE_H
#define DATABASE_H

#include <QObject>
#include <QSqlDatabase>
#include <QSqlQuery>
#include <QSqlError>
#include <QFile>
#include <QDebug>
#include <QStandardPaths>

class DatabaseManager : public QObject {
    Q_OBJECT

public:
    static DatabaseManager& instance();

    bool initializeDatabase();
    bool executeQuery(const QString& query);
    QSqlQuery executeSelect(const QString& query);

    // Специализированные методы
    bool addStudent(const QVariantMap& studentData);
    bool updateStudent(int id, const QVariantMap& studentData);
    bool deleteStudent(int id);
    QVariantList getStudents(const QString& filter = "");
    QVariantList getStudentGrades(int studentId);

    // Статистические методы
    double getAverageGroupScore(const QString& groupNumber);
    QVariantList getTopStudents(int limit = 10);

    // Транзакции
    bool beginTransaction();
    bool commitTransaction();
    bool rollbackTransaction();

    QString lastError() const;

signals:
    void databaseError(const QString& error);
    void dataChanged();

private:
    explicit DatabaseManager(QObject* parent = nullptr);
    ~DatabaseManager();

    DatabaseManager(const DatabaseManager&) = delete;
    DatabaseManager& operator=(const DatabaseManager&) = delete;

    bool createTables();

```

```

    bool populateInitialData();
    bool backupDatabase();

    QSqlDatabase m_database;
    QString m_databasePath;
};

#endif // DATABASE_H
cpp
// database.cpp
#include "database.h"

DatabaseManager::DatabaseManager(QObject* parent) : QObject(parent) {
    // Определяем путь к базе данных
    QString dataDir = QStandardPaths::writableLocation(QStandardPaths::AppDataLocation);
    QDir dir(dataDir);
    if (!dir.exists()) {
        dir.mkpath(".");
    }

    m_databasePath = dataDir + "/students.db";
}

DatabaseManager& DatabaseManager::instance() {
    static DatabaseManager instance;
    return instance;
}

bool DatabaseManager::initializeDatabase() {
    // Закрываем существующее соединение, если оно открыто
    if (m_database.isOpen()) {
        m_database.close();
    }

    // Настраиваем соединение с SQLite
    m_database = QSqlDatabase::addDatabase("SQLITE", "students_connection");
    m_database.setDatabaseName(m_databasePath);

    // Настройки производительности
    m_database.setConnectOptions("SQLITE_ENABLE_REGEX=1;"
                                "SQLITE_BUSY_TIMEOUT=5000;"
                                "SQLITE_OPEN_URI=1");

    if (!m_database.open()) {
        QString error = "Не удалось открыть базу данных: " + m_database.lastError().text();
        qCritical() << error;
        emit databaseError(error);
        return false;
    }

    qDebug() << "База данных успешно открыта:" << m_databasePath;
}

```

```

// Включаем поддержку внешних ключей
executeQuery("PRAGMA foreign_keys = ON");

// Настраиваем журналирование WAL для лучшей производительности
executeQuery("PRAGMA journal_mode = WAL");

// Создаем таблицы
if (!createTables()) {
    return false;
}

// Заполняем начальными данными (если необходимо)
if (!populateInitialData()) {
    return false;
}

// Создаем резервную копию
backupDatabase();

return true;
}

bool DatabaseManager::createTables() {
    QStringList tables;

    // Таблица студентов
    tables << R"(
        CREATE TABLE IF NOT EXISTS students (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            first_name TEXT NOT NULL,
            last_name TEXT NOT NULL,
            middle_name TEXT,
            group_number TEXT NOT NULL,
            birth_date DATE,
            admission_year INTEGER,
            average_score REAL DEFAULT 0.0 CHECK(average_score >= 0 AND average_score
<= 100),
            email TEXT UNIQUE,
            phone TEXT,
            address TEXT,
            notes TEXT,
            is_active BOOLEAN DEFAULT 1,
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
    )";

    // Триггер для обновления времени изменения
    tables << R"(
        CREATE TRIGGER IF NOT EXISTS update_students_timestamp
        AFTER UPDATE ON students
        BEGIN
    )";

```

```

        UPDATE students SET updated_at = CURRENT_TIMESTAMP WHERE id = NEW.id;
    END
);

```

// Таблица предметов

```

tables << R"(
    CREATE TABLE IF NOT EXISTS subjects (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL UNIQUE,
        code TEXT UNIQUE,
        hours INTEGER DEFAULT 36,
        semester INTEGER CHECK(semester >= 1 AND semester <= 8),
        description TEXT,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    )
);

```

// Таблица оценок

```

tables << R"(
    CREATE TABLE IF NOT EXISTS grades (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        student_id INTEGER NOT NULL,
        subject_id INTEGER NOT NULL,
        grade INTEGER NOT NULL CHECK(grade >= 1 AND grade <= 5),
        exam_date DATE NOT NULL,
        teacher_name TEXT,
        notes TEXT,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (student_id) REFERENCES students(id) ON DELETE CASCADE,
        FOREIGN KEY (subject_id) REFERENCES subjects(id) ON DELETE CASCADE,
        UNIQUE(student_id, subject_id, exam_date)
    )
);

```

// Таблица посещаемости

```

tables << R"(
    CREATE TABLE IF NOT EXISTS attendance (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        student_id INTEGER NOT NULL,
        subject_id INTEGER NOT NULL,
        date DATE NOT NULL,
        status INTEGER DEFAULT 1, -- 1=присутствовал, 0=отсутствовал, 2=опоздал
        reason TEXT,
        FOREIGN KEY (student_id) REFERENCES students(id) ON DELETE CASCADE,
        FOREIGN KEY (subject_id) REFERENCES subjects(id) ON DELETE CASCADE
    )
);

```

// Выполняем все SQL-запросы

```

for (const QString& tableSql : tables) {
    if (!executeQuery(tableSql)) {
        return false;
    }
}

```

```

    }
}

// Создаем индексы для ускорения поиска
QStringList indexes;
indexes << "CREATE INDEX IF NOT EXISTS idx_students_group ON
students(group_number)"
    << "CREATE INDEX IF NOT EXISTS idx_students_active ON students(is_active)"
    << "CREATE INDEX IF NOT EXISTS idx_grades_student ON grades(student_id)"
    << "CREATE INDEX IF NOT EXISTS idx_grades_subject ON grades(subject_id)"
    << "CREATE INDEX IF NOT EXISTS idx_attendance_date ON attendance(date)"
    << "CREATE INDEX IF NOT EXISTS idx_students_name ON students(last_name,
first_name)";

for (const QString& indexSql : indexes) {
    executeQuery(indexSql);
}

qDebug() << "Таблицы базы данных успешно созданы";
return true;
}

bool DatabaseManager::addStudent(const QVariantMap& studentData) {
    QSqlQuery query(m_database);
    query.prepare(R"(
        INSERT INTO students
        (first_name, last_name, middle_name, group_number, birth_date,
        admission_year, email, phone, address, notes, average_score, is_active)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    )");

    query.addBindValue(studentData.value("first_name").toString());
    query.addBindValue(studentData.value("last_name").toString());
    query.addBindValue(studentData.value("middle_name").toString());
    query.addBindValue(studentData.value("group_number").toString());
    query.addBindValue(studentData.value("birth_date").toDate());
    query.addBindValue(studentData.value("admission_year").toInt());
    query.addBindValue(studentData.value("email").toString());
    query.addBindValue(studentData.value("phone").toString());
    query.addBindValue(studentData.value("address").toString());
    query.addBindValue(studentData.value("notes").toString());
    query.addBindValue(studentData.value("average_score", 0.0).toDouble());
    query.addBindValue(studentData.value("is_active", true).toBool());

    if (!query.exec()) {
        qCritical() << "Ошибка при добавлении студента:" << query.lastError().text();
        return false;
    }

    emit dataChanged();
    return true;
}

```

```

QVariantList DatabaseManager::getStudents(const QString& filter) {
    QVariantList students;
    QString sql = R"(
        SELECT id, first_name, last_name, middle_name, group_number,
               birth_date, admission_year, average_score, email, phone,
               address, notes, is_active, created_at, updated_at
        FROM students
    )";

    if (!filter.isEmpty()) {
        sql += " WHERE " + filter;
    }

    sql += " ORDER BY last_name, first_name";

    QSqlQuery query = executeSelect(sql);
    while (query.next()) {
        QVariantMap student;
        student["id"] = query.value("id").toInt();
        student["first_name"] = query.value("first_name").toString();
        student["last_name"] = query.value("last_name").toString();
        student["middle_name"] = query.value("middle_name").toString();
        student["full_name"] = QString("%1 %2 %3")
            .arg(query.value("last_name").toString())
            .arg(query.value("first_name").toString())
            .arg(query.value("middle_name").toString().trimmed());
        student["group_number"] = query.value("group_number").toString();
        student["birth_date"] = query.value("birth_date").toDate();
        student["admission_year"] = query.value("admission_year").toInt();
        student["average_score"] = query.value("average_score").toDouble();
        student["email"] = query.value("email").toString();
        student["phone"] = query.value("phone").toString();
        student["address"] = query.value("address").toString();
        student["notes"] = query.value("notes").toString();
        student["is_active"] = query.value("is_active").toBool();
        student["created_at"] = query.value("created_at").toDateTime();
        student["updated_at"] = query.value("updated_at").toDateTime();

        // Вычисляем возраст студента
        QDate birthDate = query.value("birth_date").toDate();
        if (birthDate.isValid()) {
            int age = QDate::currentDate().year() - birthDate.year();
            student["age"] = age;
        }

        students.append(student);
    }

    return students;
}

```

3. Главное окно приложения

cpp

// mainwindow.h

#ifndef MAINWINDOW_H

#define MAINWINDOW_H

#include <QMainWindow>

#include <QSqlTableModel>

#include <QSqlRelationalTableModel>

#include <QSortFilterProxyModel>

#include <QDataWidgetMapper>

QT_BEGIN_NAMESPACE

namespace Ui {

class MainWindow;

}

QT_END_NAMESPACE

class MainWindow : public QMainWindow {
 Q_OBJECT

public:

explicit MainWindow(QWidget* parent = nullptr);

~MainWindow();

protected:

void closeEvent(QCloseEvent* event) override;

private slots:

// Слоты для студентов

void addStudent();

void editStudent();

void deleteStudent();

void searchStudent();

void refreshStudents();

void exportStudents();

void importStudents();

void printStudentList();

// Слоты для оценок

void addGrade();

void updateStudentAverage();

// Слоты для отчетов

void showStatistics();

void generateReport();

// Слоты для фильтрации

void filterByGroup();

void filterByStatus();

// Слоты для валидации

```

void validateStudentForm();
void validateGradeForm();

private:
    void setupDatabase();
    void setupUI();
    void setupModels();
    void setupConnections();
    void setupMenu();
    void setupToolBar();
    void setupStatusBar();

    // Инициализация данных
    void loadInitialData();
    void loadStudentDetails(int studentId);
    void loadStudentGrades(int studentId);

    // Вспомогательные методы
    bool backupDatabase();
    void restoreDatabase();
    void optimizeDatabase();

    // Валидация
    bool validateEmail(const QString& email);
    bool validatePhone(const QString& phone);
    bool validateBirthDate(const QDate& date);

    // Экспорт/импорт
    void exportToCSV(const QString& filename);
    void importFromCSV(const QString& filename);
    void exportToExcel(const QString& filename);

    // Члены класса
    Ui::MainWindow* ui;

    // Модели данных
    QSqlTableModel* studentModel;
    QSqlRelationalTableModel* gradeModel;
    QSortFilterProxyModel* studentProxyModel;
    QSortFilterProxyModel* gradeProxyModel;
    QDataWidgetMapper* studentMapper;

    // Фильтры
    QString currentGroupFilter;
    bool showOnlyActive = true;

    // Статус
    int totalStudents = 0;
    int activeStudents = 0;
};

#endif // MAINWINDOW_H

```

cpp

```
// mainwindow.cpp
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "database.h"
#include <QMessageBox>
#include <QInputDialog>
#include <QFileDialog>
#include <QPrinter>
#include <QPrintDialog>
#include <QPainter>
#include <QSqlError>
#include <QSqlRecord>
#include <QSortFilterProxyModel>
#include <QDataWidgetMapper>
#include <QHeaderView>
#include <QMenu>
#include <QClipboard>
#include <QDateEdit>
#include <QCompleter>
#include <QRegularExpressionValidator>
#include <QProgressDialog>
#include <QStandardPaths>
#include <QtCharts>

MainWindow::MainWindow(QWidget* parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow) {

    ui->setupUi(this);
    setWindowTitle("Система учета студентов");

    // Инициализация базы данных
    if (!DatabaseManager::instance().initializeDatabase()) {
        QMessageBox::critical(this, "Ошибка",
            "Не удалось инициализировать базу данных!");
        return;
    }

    setupUI();
    setupModels();
    setupConnections();
    setupMenu();
    setupToolbar();
    setupStatusBar();

    // Загрузка начальных данных
    loadInitialData();

    // Настройка размера окна
    resize(1200, 700);
```

```

qDebug() << "Приложение успешно инициализировано";
}

void MainWindow::setupUI() {
    // Настройка таблицы студентов
    ui->studentsTable->setSelectionBehavior(QAbstractItemView::SelectRows);
    ui->studentsTable->setSelectionMode(QAbstractItemView::SingleSelection);
    ui->studentsTable->setAlternatingRowColors(true);
    ui->studentsTable->setSortingEnabled(true);
    ui->studentsTable->setEditTriggers(QAbstractItemView::NoEditTriggers);

    // Настройка таблицы оценок
    ui->gradesTable->setSelectionBehavior(QAbstractItemView::SelectRows);
    ui->gradesTable->setAlternatingRowColors(true);

    // Настройка вкладок
    ui->tabWidget->setCurrentIndex(0);

    // Настройка валидаторов для формы
    // Email валидатор
    QRegularExpression emailRegex(R"([a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$)");
    QRegularExpressionValidator* emailValidator =
        new QRegularExpressionValidator(emailRegex, this);
    ui->emailEdit->setValidator(emailValidator);

    // Телефон валидатор
    QRegularExpression phoneRegex(R"([d\s\-\+\(\)]{10,15}$)");
    QRegularExpressionValidator* phoneValidator =
        new QRegularExpressionValidator(phoneRegex, this);
    ui->phoneEdit->setValidator(phoneValidator);

    // Автозаполнение для групп
    QCompleter* groupCompleter = new QCompleter(this);
    groupCompleter->setCaseSensitivity(Qt::CaseInsensitive);
    ui->groupCombo->setCompleter(groupCompleter);

    // Настройка DateEdit
    ui->birthDateEdit->setDate(QDate::currentDate().addYears(-18));
    ui->birthDateEdit->setCalendarPopup(true);
    ui->birthDateEdit->setMaximumDate(QDate::currentDate());

    // Настройка SpinBox для среднего балла
    ui->averageSpinBox->setRange(0.0, 100.0);
    ui->averageSpinBox->setDecimals(1);
    ui->averageSpinBox->setSingleStep(0.5);
    ui->averageSpinBox->setSuffix(" %");

    // Создание кнопок в панели инструментов
    ui->mainToolBar->setToolButtonStyle(Qt::ToolButtonTextUnderIcon);
    ui->mainToolBar->setIconSize(QSize(32, 32));
}

```

```

// Настройка статусной строки
statusBar()->showMessage("Готово");
}

void MainWindow::setupModels() {
    // Модель для студентов
    studentModel = new QSqlTableModel(this, DatabaseManager::instance().database());
    studentModel->setTable("students");
    studentModel->setEditStrategy(QSqlTableModel::OnManualSubmit);

    // Настраиваем заголовки столбцов
    studentModel->setHeaderData(0, Qt::Horizontal, "ID");
    studentModel->setHeaderData(1, Qt::Horizontal, "Фамилия");
    studentModel->setHeaderData(2, Qt::Horizontal, "Имя");
    studentModel->setHeaderData(3, Qt::Horizontal, "Отчество");
    studentModel->setHeaderData(4, Qt::Horizontal, "Группа");
    studentModel->setHeaderData(5, Qt::Horizontal, "Дата рождения");
    studentModel->setHeaderData(6, Qt::Horizontal, "Год поступления");
    studentModel->setHeaderData(7, Qt::Horizontal, "Средний балл");
    studentModel->setHeaderData(8, Qt::Horizontal, "Email");
    studentModel->setHeaderData(9, Qt::Horizontal, "Телефон");
    studentModel->setHeaderData(10, Qt::Horizontal, "Адрес");
    studentModel->setHeaderData(11, Qt::Horizontal, "Заметки");
    studentModel->setHeaderData(12, Qt::Horizontal, "Активен");

    // Создаем прокси-модель для сортировки и фильтрации
    studentProxyModel = new QSortFilterProxyModel(this);
    studentProxyModel->setSourceModel(studentModel);
    studentProxyModel->setSortCaseSensitivity(Qt::CaseInsensitive);
    studentProxyModel->setFilterKeyColumn(-1); // Поиск по всем столбцам

    // Применяем фильтр по умолчанию (только активные студенты)
    studentProxyModel->setFilterRegExp(QRegExp("1", Qt::CaseInsensitive,
    QRegExp::FixedString));
    studentProxyModel->setFilterKeyColumn(12); // Столбец "is_active"

    // Устанавливаем модель в таблицу
    ui->studentsTable->setModel(studentProxyModel);

    // Настраиваем отображение столбцов
    ui->studentsTable->hideColumn(0); // Скрываем ID
    ui->studentsTable->hideColumn(11); // Скрываем заметки
    ui->studentsTable->hideColumn(12); // Скрываем is_active

    // Настройка ширины столбцов
    ui->studentsTable->horizontalHeader()->setSectionResizeMode(QHeaderView::Interactive);
    ui->studentsTable->horizontalHeader()->setStretchLastSection(true);
    ui->studentsTable->setColumnWidth(1, 150); // Фамилия
    ui->studentsTable->setColumnWidth(2, 120); // Имя
    ui->studentsTable->setColumnWidth(3, 150); // Отчество
    ui->studentsTable->setColumnWidth(4, 80); // Группа
    ui->studentsTable->setColumnWidth(5, 120); // Дата рождения

```

```

ui->studentsTable->setColumnWidth(6, 100); // Год поступления
ui->studentsTable->setColumnWidth(7, 100); // Средний балл
ui->studentsTable->setColumnWidth(8, 200); // Email
ui->studentsTable->setColumnWidth(9, 120); // Телефон
ui->studentsTable->setColumnWidth(10, 200); // Адрес

// Модель для оценок
gradeModel = new QSqlRelationalTableModel(this);
gradeModel->setTable("grades");
gradeModel->setRelation(2, QSqlRelation("subjects", "id", "name"));
gradeModel->setEditStrategy(QSqlTableModel::OnManualSubmit);

gradeModel->setHeaderData(0, Qt::Horizontal, "ID");
gradeModel->setHeaderData(1, Qt::Horizontal, "Студент");
gradeModel->setHeaderData(2, Qt::Horizontal, "Предмет");
gradeModel->setHeaderData(3, Qt::Horizontal, "Оценка");
gradeModel->setHeaderData(4, Qt::Horizontal, "Дата экзамена");
gradeModel->setHeaderData(5, Qt::Horizontal, "Преподаватель");

gradeProxyModel = new QSortFilterProxyModel(this);
gradeProxyModel->setSourceModel(gradeModel);

ui->gradesTable->setModel(gradeProxyModel);
ui->gradesTable->hideColumn(0); // Скрываем ID
ui->gradesTable->hideColumn(1); // Скрываем ID студента

// Обновляем модели
refreshStudents();
}

void MainWindow::addStudent() {
    // Валидация формы
    if (!validateStudentForm()) {
        return;
    }

    // Создаем новую запись
    QSqlRecord record = studentModel->record();

    record.setValue("first_name", ui->firstNameEdit->text().trimmed());
    record.setValue("last_name", ui->lastNameEdit->text().trimmed());
    record.setValue("middle_name", ui->middleNameEdit->text().trimmed());
    record.setValue("group_number", ui->groupCombo->currentText());
    record.setValue("birth_date", ui->birthDateEdit->date());
    record.setValue("admission_year", ui->admissionYearSpin->value());
    record.setValue("email", ui->emailEdit->text().trimmed());
    record.setValue("phone", ui->phoneEdit->text().trimmed());
    record.setValue("address", ui->addressEdit->toPlainText().trimmed());
    record.setValue("notes", ui->notesEdit->toPlainText().trimmed());
    record.setValue("average_score", ui->averageSpinBox->value());
    record.setValue("is_active", ui->activeCheckBox->isChecked());

```

```

// Вставляем запись в модель
if (!studentModel->insertRecord(-1, record)) {
    QMessageBox::critical(this, "Ошибка",
        "Не удалось добавить студента: " + studentModel->lastError().text());
    return;
}

// Сохраняем изменения в БД
if (!studentModel->submitAll()) {
    QMessageBox::critical(this, "Ошибка",
        "Не удалось сохранить данные: " + studentModel->lastError().text());
    studentModel->revertAll();
    return;
}

// Очищаем форму
clearStudentForm();

// Обновляем таблицу
refreshStudents();

// Показываем сообщение об успехе
statusBar()->showMessage("Студент успешно добавлен", 3000);
QMessageBox::information(this, "Успех", "Студент успешно добавлен в базу данных!");
}

void MainWindow::refreshStudents() {
    // Обновляем данные в модели
    if (!studentModel->select()) {
        QMessageBox::critical(this, "Ошибка",
            "Не удалось загрузить данные студентов: " + studentModel->lastError().text());
        return;
    }

    // Обновляем статистику
    updateStatistics();

    // Обновляем автозаполнение для групп
    updateGroupCompleter();

    statusBar()->showMessage(QString("Загружено %1 студентов").arg(studentModel-
>rowCount()));
}

void MainWindow::updateStatistics() {
    // Получаем статистику из базы данных
    QSqlQuery query(DatabaseManager::instance().database());

    // Общее количество студентов
    query.exec("SELECT COUNT(*) FROM students");
    if (query.next()) {
        totalStudents = query.value(0).toInt();
    }
}

```

```

    }

    // Количество активных студентов
    query.exec("SELECT COUNT(*) FROM students WHERE is_active = 1");
    if (query.next()) {
        activeStudents = query.value(0).toInt();
    }

    // Средний балл по группам
    query.exec(R"(
        SELECT group_number, ROUND(AVG(average_score), 2) as avg_score
        FROM students
        WHERE is_active = 1
        GROUP BY group_number
        ORDER BY avg_score DESC
    )");

    // Обновляем UI
    ui->totalStudentsLabel->setText(QString::number(totalStudents));
    ui->activeStudentsLabel->setText(QString::number(activeStudents));

    // Обновляем статусную строку
    statusBar()->showMessage(QString("Студентов: %1, Активных:
    %2").arg(totalStudents).arg(activeStudents));
}

```

4. Диалоговое окно редактирования студента

```

cpp
// studentdialog.h
#ifndef STUDENTDIALOG_H
#define STUDENTDIALOG_H

#include <QDialog>
#include <QSqlRecord>

namespace Ui {
class StudentDialog;
}

class StudentDialog : public QDialog {
    Q_OBJECT

public:
    enum Mode { AddMode, EditMode };

    explicit StudentDialog(QWidget* parent = nullptr, Mode mode = AddMode,
        const QSqlRecord& record = QSqlRecord());
    ~StudentDialog();

    QSqlRecord getRecord() const;
    bool isValid() const;

private slots:

```

```

void accept() override;
void validateForm();
void calculateAge();

private:
    void setupUI();
    void loadRecord(const QSqlRecord& record);
    bool validateFields();
    void showValidationError(const QString& message);

    Ui::StudentDialog* ui;
    Mode m_mode;
    QSqlRecord m_record;
    bool m_isValid = false;
};

#endif // STUDENTDIALOG_H

cpp
// studentdialog.cpp
#include "studentdialog.h"
#include "ui_studentdialog.h"
#include <QMessageBox>
#include <QRegularExpressionValidator>
#include <QIntValidator>
#include <QDoubleValidator>

StudentDialog::StudentDialog(QWidget* parent, Mode mode, const QSqlRecord& record)
: QDialog(parent), ui(new Ui::StudentDialog), m_mode(mode), m_record(record) {

    ui->setupUi(this);

    // Настройка окна в зависимости от режима
    if (mode == AddMode) {
        setWindowTitle("Добавление нового студента");
    } else {
        setWindowTitle("Редактирование студента");
        loadRecord(record);
    }

    setupUI();

    // Подключение сигналов валидации
    connect(ui->firstNameEdit, &QLineEdit::textChanged, this, &StudentDialog::validateForm);
    connect(ui->lastNameEdit, &QLineEdit::textChanged, this, &StudentDialog::validateForm);
    connect(ui->groupEdit, &QLineEdit::textChanged, this, &StudentDialog::validateForm);
    connect(ui->birthDateEdit, &QDateEdit::dateChanged, this, &StudentDialog::calculateAge);

    // Изначальная валидация
    validateForm();
}

void StudentDialog::setupUI() {

```

```

// Настройка валидаторов
QRegularExpression nameRegex("[A-ЯЁa-яёA-Za-z\\s\\-']+");
QRegularExpressionValidator* nameValidator =
    new QRegularExpressionValidator(nameRegex, this);

ui->firstNameEdit->setValidator(nameValidator);
ui->lastNameEdit->setValidator(nameValidator);
ui->middleNameEdit->setValidator(nameValidator);

// Валидатор для группы (буквы и цифры)
QRegularExpression groupRegex("[A-ЯЁA-Z0-9\\-']+");
QRegularExpressionValidator* groupValidator =
    new QRegularExpressionValidator(groupRegex, this);
ui->groupEdit->setValidator(groupValidator);

// Валидатор для email
QRegularExpression emailRegex(R"([a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,})");
QRegularExpressionValidator* emailValidator =
    new QRegularExpressionValidator(emailRegex, this);
ui->emailEdit->setValidator(emailValidator);

// Настройка DateEdit
ui->birthDateEdit->setDate(QDate::currentDate().addYears(-18));
ui->birthDateEdit->setMaximumDate(QDate::currentDate());
ui->birthDateEdit->setCalendarPopup(true);

// Настройка SpinBox'ов
ui->admissionYearSpin->setRange(2000, QDate::currentDate().year());
ui->admissionYearSpin->setValue(QDate::currentDate().year());

ui->averageScoreSpin->setRange(0.0, 100.0);
ui->averageScoreSpin->setDecimals(2);
ui->averageScoreSpin->setSingleStep(0.5);

// Расчет возраста
calculateAge();
}

void StudentDialog::validateForm() {
    bool isValid = true;
    QStringList errors;

    // Проверка обязательных полей
    if (ui->firstNameEdit->text().trimmed().isEmpty()) {
        errors << "Имя обязательно для заполнения";
        isValid = false;
    }

    if (ui->lastNameEdit->text().trimmed().isEmpty()) {
        errors << "Фамилия обязательна для заполнения";
        isValid = false;
    }
}

```

```

    }

    if (ui->groupEdit->text().trimmed().isEmpty()) {
        errors << "Номер группы обязателен для заполнения";
        isValid = false;
    }

    // Проверка email
    if (!ui->emailEdit->text().isEmpty()) {
        QRegularExpression emailRegex(R"([a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$)");
        if (!emailRegex.match(ui->emailEdit->text()).hasMatch()) {
            errors << "Некорректный email адрес";
            isValid = false;
        }
    }

    // Проверка даты рождения
    QDate birthDate = ui->birthDateEdit->date();
    if (birthDate.isValid()) {
        int age = QDate::currentDate().year() - birthDate.year();
        if (age < 16 || age > 70) {
            errors << QString("Возраст студента должен быть от 16 до 70 лет (сейчас: %1)").arg(age);
            isValid = false;
        }
    }

    // Обновление кнопки ОК
    ui->buttonBox->button(QDialogButtonBox::Ok)->setEnabled(isValid);

    // Отображение ошибок
    if (!errors.isEmpty()) {
        ui->errorLabel->setText(errors.join("\n") + ".");
        ui->errorLabel->setStyleSheet("color: red;");
    } else {
        ui->errorLabel->clear();
    }

    m_isValid = isValid;
}

void StudentDialog::accept() {
    if (!validateFields()) {
        return;
    }

    // Заполняем запись данными из формы
    m_record.setValue("first_name", ui->firstNameEdit->text().trimmed());
    m_record.setValue("last_name", ui->lastNameEdit->text().trimmed());
    m_record.setValue("middle_name", ui->middleNameEdit->text().trimmed());
    m_record.setValue("group_number", ui->groupEdit->text().trimmed());
}

```

```

m_record.setValue("birth_date", ui->birthDateEdit->date());
m_record.setValue("admission_year", ui->admissionYearSpin->value());
m_record.setValue("email", ui->emailEdit->text().trimmed());
m_record.setValue("phone", ui->phoneEdit->text().trimmed());
m_record.setValue("address", ui->addressEdit->toPlainText().trimmed());
m_record.setValue("notes", ui->notesEdit->toPlainText().trimmed());
m_record.setValue("average_score", ui->averageScoreSpin->value());
m_record.setValue("is_active", ui->activeCheckBox->isChecked());

QDialog::accept();
}

```

5. Класс для генерации отчетов

cpp

// reportgenerator.h

```

#ifndef REPORTGENERATOR_H
#define REPORTGENERATOR_H

```

```

#include <QObject>
#include <QSqlDatabase>
#include <QTextDocument>
#include <QPrinter>

```

```

class ReportGenerator : public QObject {
    Q_OBJECT

```

public:

```

    enum ReportType {
        StudentList,
        GradesReport,
        StatisticsReport,
        AttendanceReport
    };

```

```

    explicit ReportGenerator(QObject* parent = nullptr);

```

```

    bool generateReport(ReportType type, const QVariantMap& parameters = QVariantMap());
    bool exportToPDF(const QString& filename);
    bool exportToHTML(const QString& filename);
    bool exportToExcel(const QString& filename);
    bool printReport();

```

```

    QString lastError() const;

```

signals:

```

    void reportGenerated(const QString& htmlContent);
    void reportError(const QString& error);

```

private:

```

    QString generateStudentListReport(const QVariantMap& params);
    QString generateGradesReport(const QVariantMap& params);
    QString generateStatisticsReport(const QVariantMap& params);
    QString generateAttendanceReport(const QVariantMap& params);

```

```

QString generateHTMLHeader(const QString& title);
QString generateHTMLFooter();
QString generateTableHTML(const QStringList& headers,
                          const QList<QStringList>& data);

QSqlDatabase m_database;
QString m_lastError;
QString m_currentReport;
};

#endif // REPORTGENERATOR_H
cpp
// reportgenerator.cpp
#include "reportgenerator.h"
#include <QSqlQuery>
#include <QSqlError>
#include <QFile>
#include <QTextStream>
#include <QDesktopServices>
#include <QUrl>
#include <QDateTime>

ReportGenerator::ReportGenerator(QObject* parent) : QObject(parent) {
    m_database = QSqlDatabase::database();
}

bool ReportGenerator::generateReport(ReportType type, const QVariantMap& parameters) {
    QString reportHtml;

    switch (type) {
    case StudentList:
        reportHtml = generateStudentListReport(parameters);
        break;
    case GradesReport:
        reportHtml = generateGradesReport(parameters);
        break;
    case StatisticsReport:
        reportHtml = generateStatisticsReport(parameters);
        break;
    case AttendanceReport:
        reportHtml = generateAttendanceReport(parameters);
        break;
    default:
        m_lastError = "Неизвестный тип отчета";
        emit reportError(m_lastError);
        return false;
    }

    if (reportHtml.isEmpty()) {
        return false;
    }
}

```

```

    m_currentReport = reportHtml;
    emit reportGenerated(reportHtml);
    return true;
}

QString ReportGenerator::generateStudentListReport(const QVariantMap& params) {
    QString filter = params.value("filter", "").toString();
    bool activeOnly = params.value("activeOnly", true).toBool();
    QString group = params.value("group", "").toString();

    QString sql = R"(
        SELECT
            last_name || ' ' || first_name || ' ' || COALESCE(middle_name, '') as full_name,
            group_number,
            birth_date,
            admission_year,
            average_score,
            email,
            phone,
            CASE WHEN is_active THEN 'Активен' ELSE 'Неактивен' END as status
        FROM students
        WHERE 1=1
    )";

    if (activeOnly) {
        sql += " AND is_active = 1";
    }

    if (!group.isEmpty()) {
        sql += " AND group_number = '" + group + "'";
    }

    if (!filter.isEmpty()) {
        sql += " AND (last_name LIKE '%" + filter + "%' OR first_name LIKE '%" + filter + "%')";
    }

    sql += " ORDER BY last_name, first_name";

    QSqlQuery query(sql);
    if (!query.exec()) {
        m_lastError = "Ошибка выполнения запроса: " + query.lastError().text();
        emit reportError(m_lastError);
        return QString();
    }

    // Генерация HTML
    QStringList headers = {"ФИО", "Группа", "Дата рождения", "Год поступления",
                          "Средний балл", "Email", "Телефон", "Статус"};

    QList<QStringList> data;
    while (query.next()) {

```

```

        QStringList row;
        for (int i = 0; i < headers.size(); i++) {
            row.append(query.value(i).toString());
        }
        data.append(row);
    }

    QString title = "Список студентов";
    if (!group.isEmpty()) {
        title += " группы " + group;
    }

    QString html = generateHTMLHeader(title);
    html += "<h1>" + title + "</h1>";
    html += "<p>Дата формирования: " +
    QDateTime::currentDateTime().toString("dd.MM.yyyy HH:mm") + "</p>";
    html += "<p>Всего студентов: " + QString::number(data.size()) + "</p>";
    html += generateTableHTML(headers, data);
    html += generateHTMLFooter();

    return html;
}

```

Задания по вариантам

Вариант 1: Библиотечная система

Требования:

1. Таблицы: книги, авторы, читатели, выдачи книг
2. Интерфейс для добавления/редактирования книг
3. Система учета выдачи и возврата книг
4. Поиск книг по различным критериям
5. Отчеты: популярные книги, задолжники

Вариант 2: Система учета товаров

Требования:

1. Таблицы: товары, категории, поставщики, заказы
2. Учет остатков на складе
3. История заказов и продаж
4. Формирование счетов и накладных
5. Отчеты: оборот, прибыль, популярные товары

Вариант 3: Система записи к врачу

Требования:

1. Таблицы: пациенты, врачи, специальности, записи
2. Расписание врачей
3. Запись на прием с выбором времени
4. История посещений
5. Отчеты: загруженность врачей, статистика заболеваний

Критерии оценивания

Критерий	Баллы	Описание
Создание БД	0-3	Корректность структуры базы данных, нормализация
Подключение к БД	0-2	Настройка соединения, обработка

Критерий	Баллы	Описание
		ошибок
Интерфейс CRUD	0-3	Полная реализация операций создания, чтения, обновления, удаления
Валидация данных	0-3	Проверка вводимых данных, обработка ошибок
Поиск и фильтрация	0-3	Реализация поиска и фильтрации данных
Отчеты и экспорт	0-3	Генерация отчетов, экспорт в различные форматы
Качество кода	0-3	Архитектура, комментирование, обработка исключений
Документация	0-2	Описание структуры БД, инструкция пользователя
Дополнительные возможности	0-2	Резервное копирование, импорт/экспорт, графики
Итого	0-25	

Шкала оценивания:

- **5 (отлично):** 22-25 баллов - Полная реализация всех требований, качественный код, дополнительные возможности
- **4 (хорошо):** 18-21 балл - Реализация основных требований, работающий интерфейс CRUD
- **3 (удовлетворительно):** 13-17 баллов - Базовая функциональность, но с ограничениями
- **2 (неудовлетворительно):** 0-12 баллов - Невыполнены основные требования

Структура отчета

Основные разделы:

1. **Титульный лист**
2. **Введение** (цель, задачи, актуальность)
3. **Анализ предметной области**
4. **Проектирование базы данных**
 - ER-диаграмма
 - Схема таблиц с описанием полей
 - SQL-скрипты создания таблиц
5. **Проектирование интерфейса**
 - Макеты окон и диалогов
 - Описание функциональности
6. **Реализация**
 - Описание классов и их взаимодействия
 - Ключевые фрагменты кода с комментариями
7. **Тестирование**

- Тестовые данные
- Результаты тестирования функциональности
- 8. **Инструкция пользователя**
- 9. **Заключение** (выводы, проблемы, перспективы)
- 10. **Приложения** (полный код программы)
- 11. **Список литературы**

Пример SQL-скрипта для заполнения тестовыми данными:

sql

-- Заполнение тестовыми данными

```
INSERT INTO students (first_name, last_name, group_number, birth_date, admission_year,
email, average_score) VALUES
('Иван', 'Иванов', 'ИТ-101', '2000-05-15', 2020, 'ivanov@example.com', 85.5),
('Мария', 'Петрова', 'ИТ-101', '2001-03-20', 2020, 'petrova@example.com', 92.0),
('Алексей', 'Сидоров', 'ИТ-102', '1999-11-10', 2019, 'sidorov@example.com', 78.5),
('Елена', 'Кузнецова', 'ИТ-102', '2000-08-25', 2020, 'kuznetsova@example.com', 88.0),
('Дмитрий', 'Смирнов', 'ИТ-103', '2001-01-30', 2020, 'smirnov@example.com', 91.5);
```

```
INSERT INTO subjects (name, hours, semester) VALUES
('Программирование на C++', 144, 1),
('Базы данных', 108, 2),
('Веб-разработка', 72, 3),
('Математический анализ', 144, 1),
('Алгоритмы и структуры данных', 108, 2);
```

```
INSERT INTO grades (student_id, subject_id, grade, exam_date) VALUES
(1, 1, 5, '2023-12-20'),
(1, 2, 4, '2023-12-25'),
(2, 1, 5, '2023-12-20'),
(2, 3, 5, '2024-01-15'),
(3, 2, 3, '2023-12-25');
```

Рекомендации по выполнению

Этапы выполнения работы:

1. **Проектирование БД (1 час):**
 - Определить сущности предметной области
 - Создать ER-диаграмму
 - Написать SQL-скрипты создания таблиц
2. **Создание классов для работы с БД (2 часа):**
 - Реализовать DatabaseManager
 - Добавить методы CRUD операций
 - Настроить валидацию данных
3. **Разработка интерфейса (3 часа):**
 - Создать главное окно с навигацией
 - Реализовать формы для работы с данными
 - Добавить таблицы с сортировкой и фильтрацией
4. **Реализация бизнес-логики (2 часа):**
 - Добавить обработку событий
 - Реализовать поиск и фильтрацию
 - Создать систему отчетов
5. **Тестирование и отладка (1 час):**
 - Проверить все функции
 - Исправить ошибки
 - Оптимизировать производительность

6. **Оформление отчета (1 час):**

- Описать структуру БД
- Задокументировать код
- Создать инструкцию пользователя

Частые ошибки и их решение:

Проблема	Причина	Решение
Ошибка подключения к БД	Неправильный путь или права доступа	Использовать QStandardPaths для определения пути
Медленная работа с большими данными	Отсутствие индексов	Создать индексы для часто используемых полей
Утечка памяти	Не закрытие соединений с БД	Использовать умные указатели или RAII
Конкурентный доступ	Одновременная запись из нескольких потоков	Использовать транзакции и блокировки
Потеря данных при ошибке	Отсутствие отката транзакций	Использовать beginTransaction/rollbackTransaction

Оптимизация производительности:

cpp

// Пример оптимизации запросов

```
void optimizeQueries() {
```

// Включить WAL режим для лучшей производительности при чтении/записи

```
executeQuery("PRAGMA journal_mode = WAL");
```

// Увеличить кэш страниц

```
executeQuery("PRAGMA cache_size = -2000"); // 2000 страниц по 1KB
```

// Включить синхронный режим NORMAL для лучшей производительности

```
executeQuery("PRAGMA synchronous = NORMAL");
```

// Создать индексы

```
executeQuery("CREATE INDEX IF NOT EXISTS idx_students_name ON  
students(last_name, first_name);
```

// Регулярно выполнять VACUUM для освобождения места

```
executeQuery("VACUUM");
```

```
}
```

Данная практическая работа позволит освоить ключевые аспекты разработки приложений с базами данных в Qt и создать полнофункциональное приложение для учета данных.

ПРАКТИЧЕСКАЯ РАБОТА №37-40. Создание запросов к БД. Создание хранимых процедур

Образование и наука

1. Система управления учебным процессом ВУЗа

- **Предметная область:** Полный цикл учебного процесса
- **Ключевые таблицы:** Студенты, Преподаватели, Дисциплины, Расписание, Оценки, Академические группы, Факультеты, Учебные планы
- **Функциональность:** Формирование расписания, ведение электронного журнала, расчет стипендий, управление учебными планами, статистика успеваемости.

2. Платформа онлайн-курсов

- **Предметная область:** Дистанционное образование
- **Ключевые таблицы:** Курсы, Модули, Уроки, Пользователи, Прогресс_обучения, Тесты, Сертификаты, Платежи
- **Функциональность:** Рекомендательная система курсов, отслеживание прогресса, интерактивные задания, система рейтингов, чаты с преподавателями.

3. Библиотечная система для научных публикаций

- **Предметная область:** Управление научным контентом
- **Ключевые таблицы:** Публикации, Авторы, Издательства, Ключевые_слова, Цитирования, Пользователи, Поисковые_запросы
- **Функциональность:** Поиск по метаданным и полнотекстовый, система рекомендаций "похожие статьи", учет индекса цитирования, импорт из библиографических форматов (BibTeX).

Бизнес и коммерция

4. CRM-система для малого бизнеса

- **Предметная область:** Управление взаимоотношениями с клиентами
- **Ключевые таблицы:** Клиенты, Контакты, Сделки, Встречи, Задачи, История_коммуникаций, Продукты
- **Функциональность:** Воронка продаж, автоматизация задач (напоминания о звонках), email-рассылки, аналитика эффективности менеджеров.

5. Система складского учета и логистики

- **Предметная область:** Торговля и дистрибуция
- **Ключевые таблицы:** Товары, Категории, Поставщики, Склады, Инвентаризация, Заказы_покупки, Заказы_продажи, Отгрузки
- **Функциональность:** Учет остатков в реальном времени (FIFO/LIFO), прогнозирование спроса, управление партиями и сроками годности, генерация штрих-кодов.

6. Система бронирования для отельного бизнеса

- **Предметная область:** Гостиничный сервис
- **Ключевые таблицы:** Гостиницы, Номера, Типы_номеров, Гости, Бронирования, Цены, Услуги, Отзывы
- **Функциональность:** Календарь доступности, динамическое ценообразование, интеграция с онлайн-платежами, управление уборкой номеров, система лояльности.

7. Платформа для фриланса

- **Предметная область:** Удаленная работа и услуги
- **Ключевые таблицы:** Фрилансеры, Заказчики, Проекты, Предложения, Портфолио, Отзывы_и_рейтинги, Транзакции, Споры

- **Функциональность:** Поиск проектов/исполнителей, безопасная сделка (эскроу), система рейтингов и верификации, встроенный мессенджер.

Здоровье и медицина

8. Электронная медицинская карта пациента (ЕМИАС)

- **Предметная область:** Здравоохранение
- **Ключевые таблицы:** Пациенты, Врачи, Приемы, Диагнозы, Назначения, Анализы, Больницы, Отделения
- **Функциональность:** История болезней, электронные рецепты, доступ по ролям (врач/пациент/админ), интеграция с лабораторным оборудованием, напоминания о приемах.

9. Фитнес-трекер и диетолог

- **Предметная область:** Персональное здоровье
- **Ключевые таблицы:** Пользователи, Тренировки, Упражнения, Питание, Продукты, Достижения, Показатели_здоровья
- **Функциональность:** Планирование тренировок, подсчет калорий и БЖУ, графики прогресса, интеграция с носимой электроникой (умные часы), рекомендации.

10. Аптека и учет лекарств

- **Предметная область:** Фармацевтика
- **Ключевые таблицы:** Лекарства, Производители, Аптеки, Партии, Рецепты, Продажи, Поставки
- **Функциональность:** Учет сроков годности, поиск аналогов, контроль рецептурных препаратов, формирование заказов на пополнение.

Социальные сети и развлечения

11. Агрегатор событий и билетов

- **Предметная область:** Event-индустрия
- **Ключевые таблицы:** События, Организаторы, Площадки, Билеты, Категории, Пользователи, Заказы_билетов
- **Функциональность:** Календарь событий, выбор мест на схемах залов, электронные билеты (QR-коды), система рекомендаций на основе интересов.

12. Платформа для книголюбов

- **Предметная область:** Литературные предпочтения
- **Ключевые таблицы:** Книги, Авторы, Издательства, Пользователи, Отзывы, Полки (хочу прочитать/читаю/прочитано), Цитаты
- **Функциональность:** Персональная библиотека, чтение с прогрессом, рекомендации, поиск по цитатам, книжные челленджи.

13. Система управления контентом для стримера

- **Предметная область:** Видеотрансляции
- **Ключевые таблицы:** Стримеры, Трансляции, Зрители, Подписки, Донаты, Модераторы, Клипы
- **Функциональность:** Управление расписанием стримов, система донатов и подписок, модерация чата, архив записей с метками.

Технологии и производство

14. Система отслеживания багов и управления проектами

- **Предметная область:** Разработка ПО
- **Ключевые таблицы:** Проекты, Задачи/Баг_репорты, Исполнители, Комментарии, Версии, Файлы
- **Функциональность:** Канбан-доска, workflow задач, история изменений, привязка к коммитам в Git, уведомления.

15. База знаний IT-отдела

- **Предметная область:** Техническая поддержка и инциденты
- **Ключевые таблицы:** Статьи_базы_знаний, Оборудование, Инвентарные_номера, Заявки_в_поддержку, Сотрудники_IT
- **Функциональность:** Поиск по решениям инцидентов, учет оборудования за сотрудниками, автоматизация простых заявок (сброс пароля).

16. Система мониторинга IoT-устройств

- **Предметная область:** Умный дом/промышленность
- **Ключевые таблицы:** Устройства, Типы_устройств, Показания_датчиков, Пользователи, Сценарии_автоматизации, Оповещения
- **Функциональность:** Сбор телеметрии в реальном времени, графики и аналитика, триггеры и автоматические действия (если температура > X, то включить кондиционер).

17. Учет IT-активов компании

- **Предметная область:** Управление IT-инфраструктурой
- **Ключевые таблицы:** Активы (серверы, ноутбуки, лицензии ПО), Сотрудники, Подразделения, Лицензии, Гарантии, История_перемещений
- **Функциональность:** Отслеживание жизненного цикла оборудования, планирование обновлений, контроль лицензионной чистоты.

Государство и общество

18. Система учета обращений граждан

- **Предметная область:** Государственные услуги
- **Ключевые таблицы:** Обращения, Граждане, Органы_власти, Категории_обращений, Статусы, Ответы
- **Функциональность:** Маршрутизация обращений по инстанциям, контроль сроков ответа, личный кабинет гражданина, аналитика частых проблем.

19. Кадастровая система учета недвижимости

- **Предметная область:** Земельно-имущественные отношения
- **Ключевые таблицы:** Объекты_недвижимости, Владельцы, Кадастровые_кварталы, Права_собственности, Оценка, История_сделок
- **Функциональность:** Поиск по кадастровому номеру/адресу, отображение на карте, история смены владельцев, расчет налогов.

Транспорт и логистика

20. Система каршеринга

- **Предметная область:** Аренда автомобилей
- **Ключевые таблицы:** Автомобили, Парковки, Пользователи, Аренды, Тарифы, Состояния_авто, Штрафы
- **Функциональность:** Поиск свободных авто на карте, расчет стоимости поездки, блокировка депозита, учет топлива и пробега, интеграция с ГИБДД для штрафов.

21. Трекинг доставки грузов

- **Предметная область:** Курьерские услуги
- **Ключевые таблицы:** Отправления, Курьеры, Маршруты, Статусы_доставки, Клиенты, Пункты_выдачи, Оплаты
- **Функциональность:** Отслеживание посылки в реальном времени (как у Яндекс.Доставки), расчет сроков, уведомления клиентов, оптимизация маршрутов курьеров.

Прочие интересные варианты

22. Генеалогическое древо и архив семьи

- **Предметная область:** История семьи
- **Ключевые таблицы:** Персоны, Отношения (отец, мать, супруг), События (рождение, брак, смерть), Фотографии, Документы, Места
- **Функциональность:** Визуализация дерева, хронологическая лента, привязка геолокации и фото к событиям, совместное редактирование семей.

23. База данных для коллекционера (марки, монеты)

- **Предметная область:** Коллекционирование
- **Ключевые таблицы:** Экземпляры_коллекции, Каталог (справочник всех существующих марок), Состояние, История_покупок/продаж, Коллекционеры
- **Функциональность:** Оценка коллекции по каталогу, учет инвестиций, поиск дублей для обмена, фотогалерея.

24. Система управления волонтерским проектом

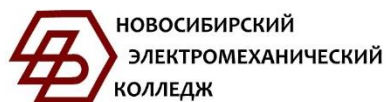
- **Предметная область:** Некоммерческая деятельность
- **Ключевые таблицы:** Волонтеры, Навыки, События_проекта, Задачи, Участники_событий, Ресурсы
- **Функциональность:** Распределение задач по навыкам, календарь событий, учет отработанных часов, мотивационные баллы.

25. Платформа для организации хакатонов

- **Предметная область:** IT-соревнования
- **Ключевые таблицы:** Хакатоны, Команды, Участники, Проекты, Жюри, Оценки, Критерии_оценки, Призы
- **Функциональность:** Регистрация команд, загрузка проектов, система blind-оценки жюри, онлайн-табло с результатами, чаты для участников.

Каждый из этих вариантов предлагает:

- **Реальные практические задачи:** Нормализация данных, связи между таблицами (один-ко-многим, многие-ко-многим).
- **Сложность на выбор:** Можно упростить (базовый CRUD) или усложнить (транзакции, оптимизация запросов, полнотекстовый поиск).
- **Востребованные технологии:** Работа с геоданными, файлами, внешними API (платежи, карты), сложными запросами.
- **Потенциал для портфолио:** Готовый проект с понятной бизнес-логикой.



НОВОСИБИРСКИЙ
ЭЛЕКТРОМЕХАНИЧЕСКИЙ
КОЛЛЕДЖ

Приложение 1. Пример выполнения отчета
Министерство образования и науки Новосибирской области
ГБПОУ НСО «Новосибирский электромеханический колледж»

Отчет

по практической работе №1
на тему «???»

дисциплина **МДК 01.01. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**
Специальность 09.02.07

Студент:
Иванов И.И.
Группа:
ИСП-24
Проверил преподаватель:
Комарова А.В.
Оценка:

«__» _____ 20__ г.

(подпись)

Новосибирск 2025 г.