Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное образовательное учреждение высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»
(ФГАОУ ВО «СПбПУ»)
Институт среднего профессионального образования

ОТЧЕТ
по учебной практике (по профилю специальности)

по профессиональному модулю ПМ.04 «Сопровождение и обслуживание программного обеспечения компьютерных систем»
(код и наименование)

Специальность 09.02.07 «Информационные системы и программирование»
(код и наименование специальности)

Студент(ка) II курса 24290907/3091 группы

Семененко Наталья Дмитриевна
(ФИО полностью)

Место прохождения практики: ФГАОУ ВО СПбПУ Петра Великого Институт СПО, учебно-вычислительный центр, пр. Энгельса д.23
(наименование и адрес организации)

Период прохождения практики

с «08» декабря 2025 г. по «27» декабря 2025 г.

Руководитель практики от учебной организации _____ (подпись) Курылева А. А. (расшифровка подписи)

Итоговая оценка по практике _____
М.П.

Санкт-Петербург
2025г.

СОГЛАСОВАНО

Председатель ПЦК
_____/_____
«_____» _____ 202__ г.

ЗАДАНИЕ
на учебную практику (по профилю специальности)

по профессиональному модулю ПМ.04 «Сопровождение и обслуживание программного обеспечения компьютерных систем»
(код и наименование)

Специальность 09.02.07 «Информационные системы и программирование»
(код и наименование специальности)

Студент(ка) II курса 24290907/3091 группы

Семененко Наталья Дмитриевна
(ФИО полностью)

Место прохождения практики: ФГАОУ ВО СПбПУ Петра Великого Институт СПО, учебно-вычислительный центр, пр. Энгельса д.23
(наименование и адрес организации)

Период прохождения практики

с «08» декабря 2025 г. по «27» декабря 2025 г.

Виды работ, обязательные для выполнения (переносится из программы, соответствующего ПМ):

1. Персонализация интегрированной среды разработки Visual Studio Community 2022
2. Отладка в IDE Visual Studio Community 2022
3. Обеспечение качества кода
4. Упаковка приложения

Индивидуальное задание:_____

Задание выдал с «8» декабря 2025 г. _____ Курылева А. А.
(подпись) (Ф.И.О.)

Задание получил с «8» декабря 2025 г. _____ _____
(подпись) (Ф.И.О.)

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»
(ФГАОУ ВО «СПбПУ»)
Институт среднего профессионального образования

ДНЕВНИК
прохождения учебной практики
(по профилю специальности)

по профессиональному модулю ПМ.04 «Сопровождение и обслуживание программного обеспечения компьютерных систем»
(код и наименование)

Специальность 09.02.07 «Информационные системы и программирование»
(код и наименование специальности)

Студент(ка) II курса 24290907/3091 группы

Семененко Наталья Дмитриевна
(ФИО полностью)

Место прохождения практики: ФГАОУ ВО СПбПУ Петра Великого Институт СПО, учебно-вычислительный центр, пр. Энгельса д.23
(наименование и адрес организации)

Период прохождения практики
с «08» декабря 2025 г. по «27» декабря 2025 г.

Руководитель с места прохождения практики _____ (подпись) Курылева А. А. (расшифровка подписи)

Санкт-Петербург
2025 г.

Содержание дневника

Дата	Виды выполненных работ и заданий по программе практики	Подпись руководителя практики
1	**2**	**3**
8.12.25	Настройка меню и панели инструментов	
9.12.25	Параметры текстового редактора	
10.12.25	Создание кода и текстового шаблона	
11.12.25	Навигация по коду с помощью отладчика	
12.12.25	Использование точек останова	
13.12.25	Управление исключениями с помощью отладчика	
15.12.25	Использование файлов дампа Использование средств профилирования	
16.12.25	Тестирование в IDE Visual Studio Community 2022	
17.12.25	Документирование кода с помощью XML-комментариев	
18.12.25	Изменение кода в соответствии с соглашением о кодировании	
19.12.25	Анализ качества кода	
20.12.25	Основы системы контроля версий Git	
22.12.25	Технологические подходы программирования	
23.12.25	Методология программирования	
24.12.25	Работа с реестром ОС Windows	
25.12.25	Упаковка классического приложения вручную	
26.12.25	Упаковка приложения с помощью Visual Studio Package Installer	
27.12.25	Создание виртуальной машины	

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Практическая работа №1 «Настройка внешнего вида и функциональности среды разработки Visual Studio»

Цель: Освоить настройку интерфейса и функциональности Visual Studio для комфортной работы.

1 вариант (номер в журнале - 21) - Вам даны два целых числа. Напишите программу, выводящую произведение целых чисел.

1. 1. Параметры текстового редактора

В настройках текстового редактора необходимо настроить автоматическое форматирование при вводе } и ;. Для этого необходимо в разделе «Средства → Параметры → Текстовый редактор» выбрать язык программирования, открыть общие настройки и установить флаги на соответствующих пунктах. Эти настройки позволят автоматически форматировать код во время ввода указанных символов (Рисунок 1).

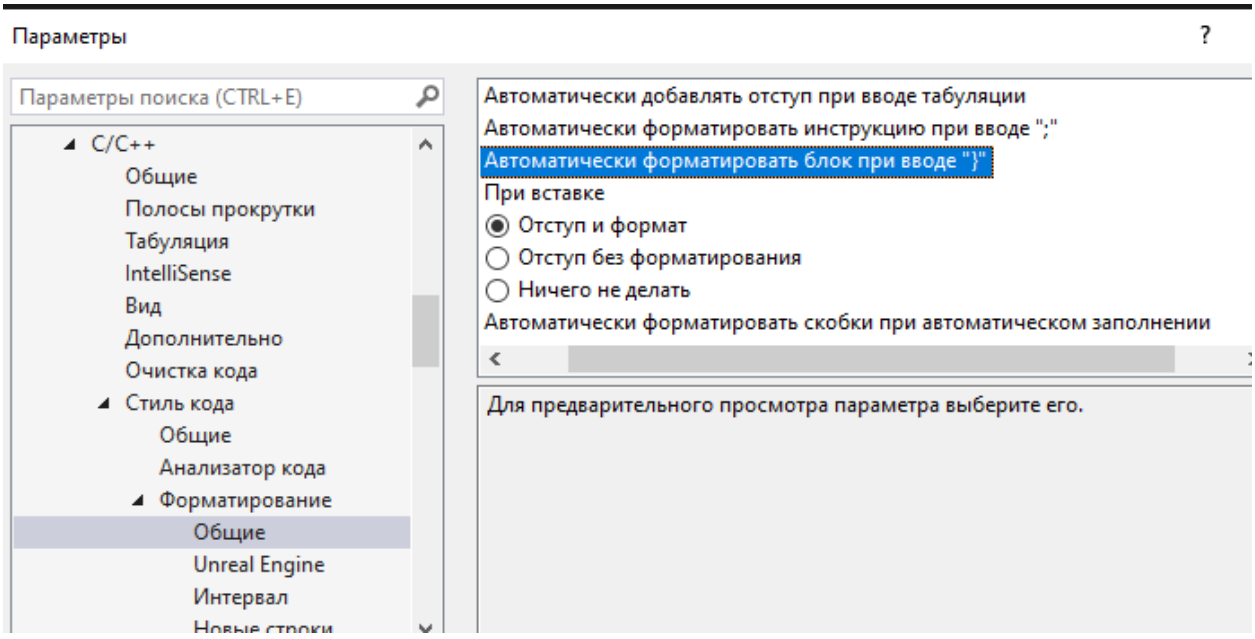


Рисунок 1 - Автоматическое форматирование при вводе } и ;.

Для изменения интервалов в разделе «Средства → Параметры → Текстовый редактор» выбираем раздел «Форматирование» устанавливаем

флаг на пункт «Вставлять пробел между скобками со списком параметров» (Рисунок 2).

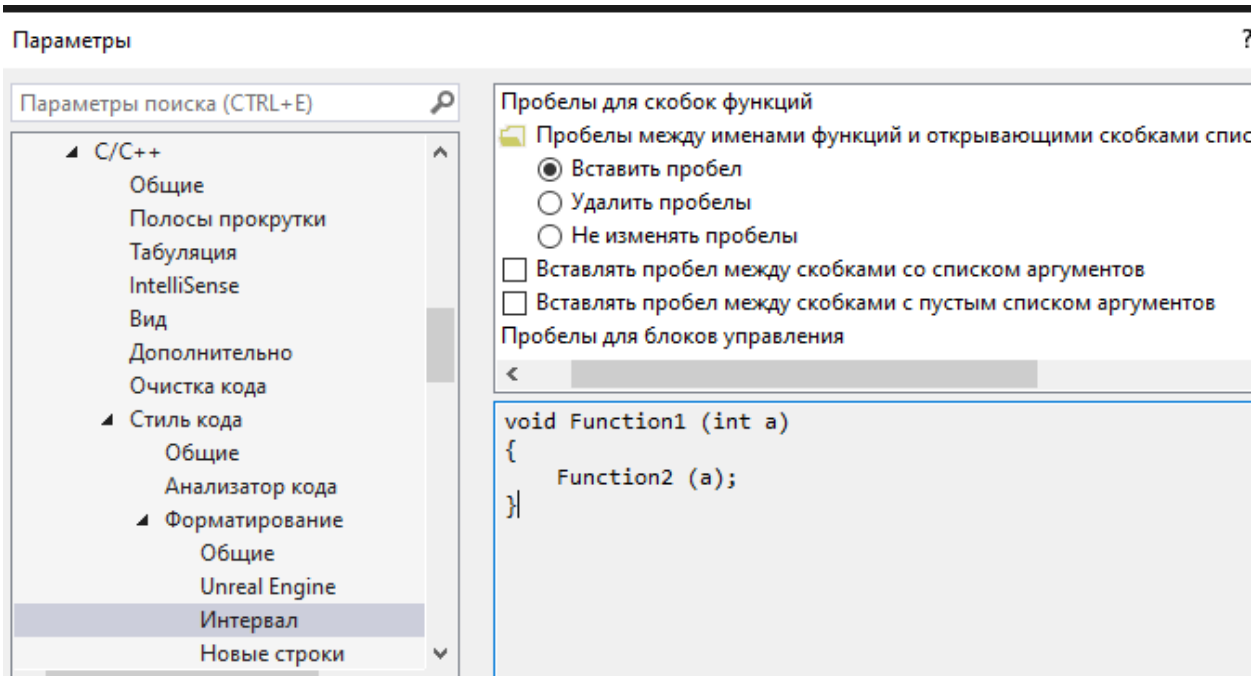


Рисунок 2 - Изменение интервалов

Для изменения интервалов в разделе «Средства → Параметры → Текстовый редактор» выбрать раздел «Общие» и установить флаг на пункт «Номера строк» (Рисунок 3).

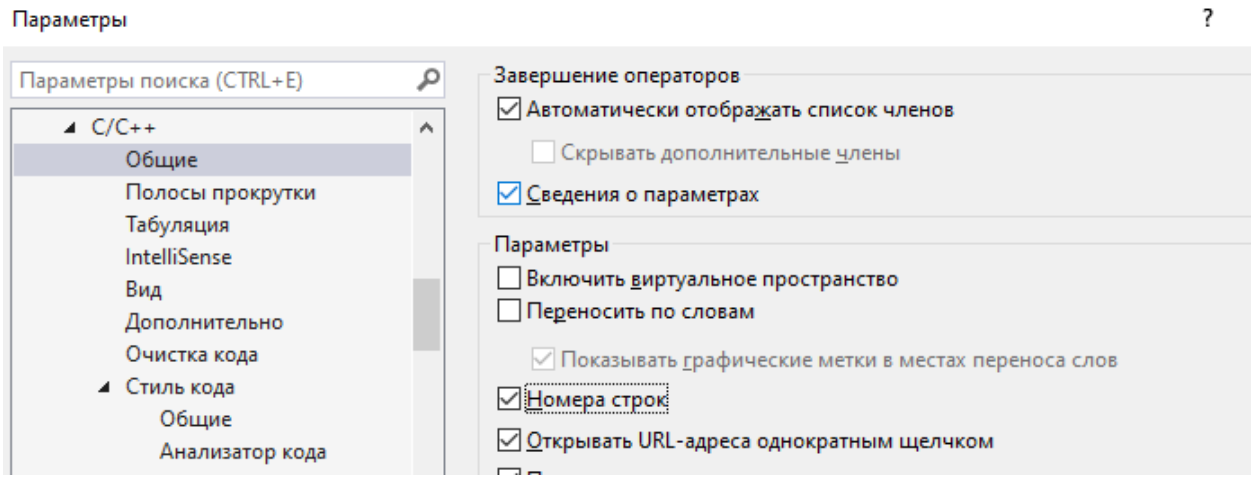


Рисунок 3 - Нумерация строк

1.2. Изменение шрифтов и цветов

Для этого необходимо в разделе «Средства → Параметры → Окружение» выбирать раздел «Шрифты и цвета» и установить необходимые параметры (Рисунок 4 – 9).

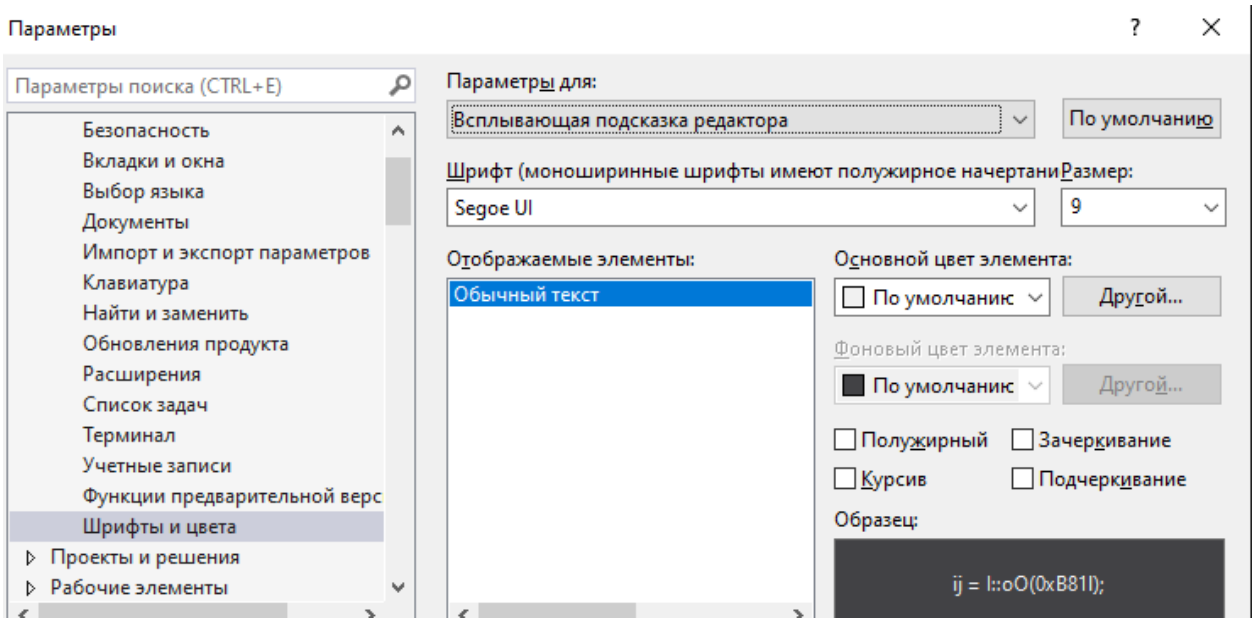


Рисунок 4 - Всплывающие подсказки

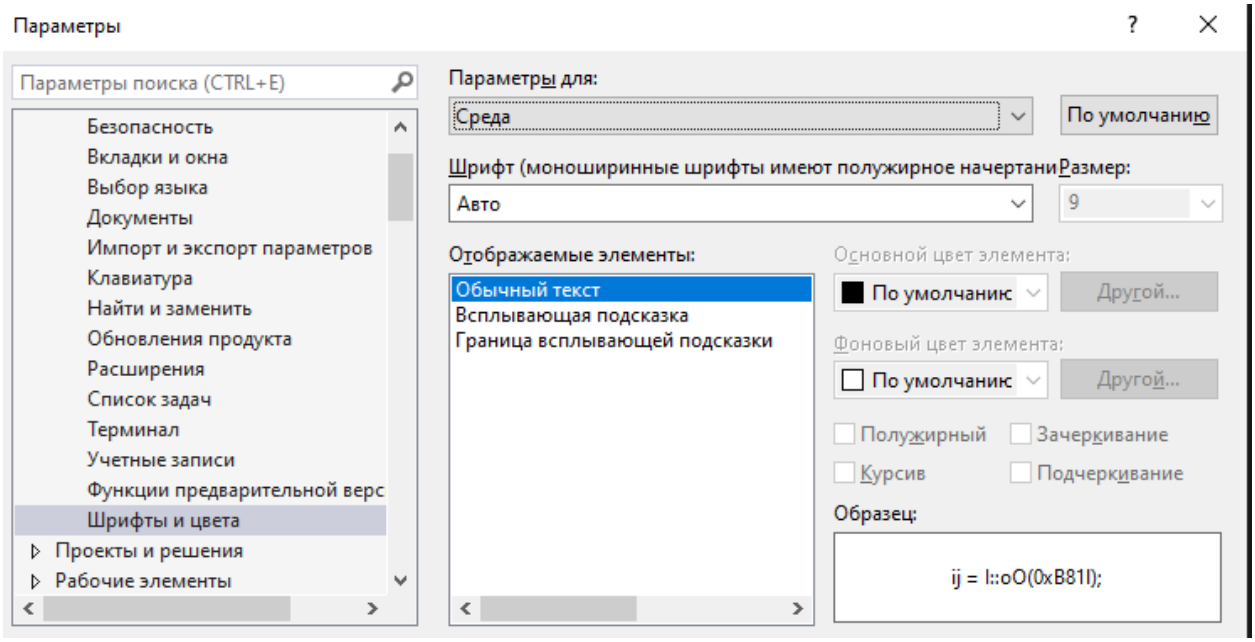


Рисунок 5 – Текст среды

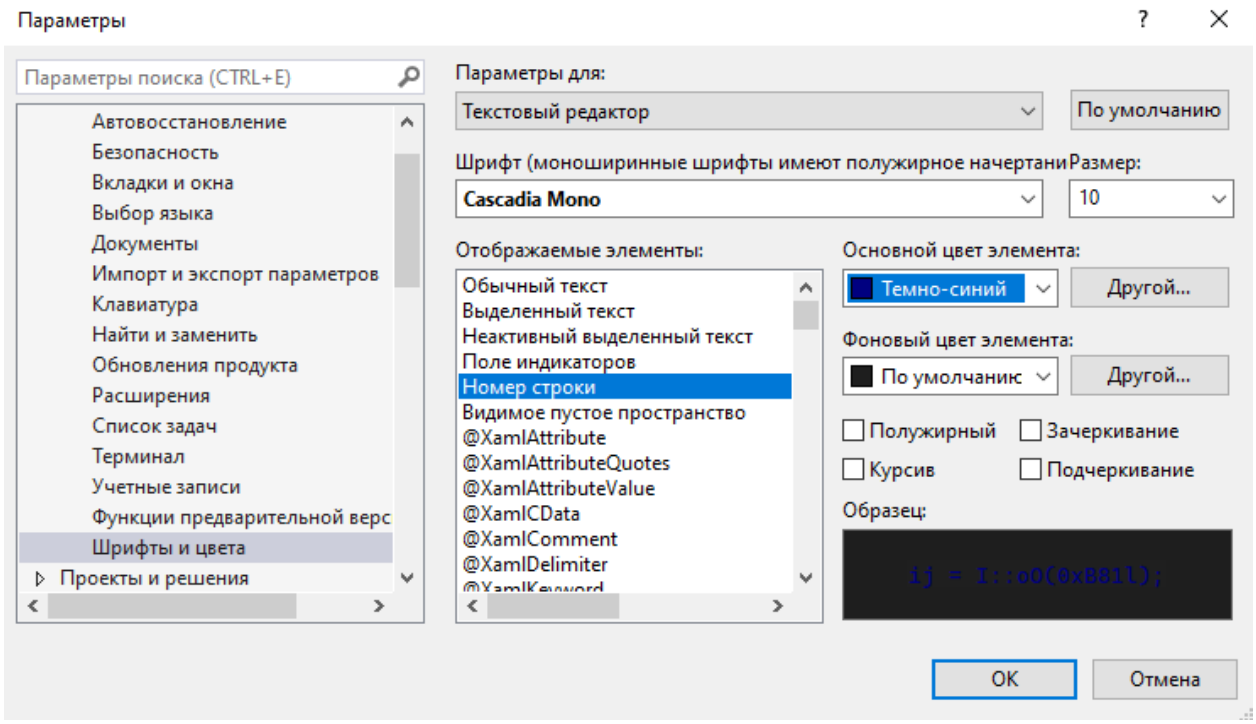


Рисунок 6 – Номера строк

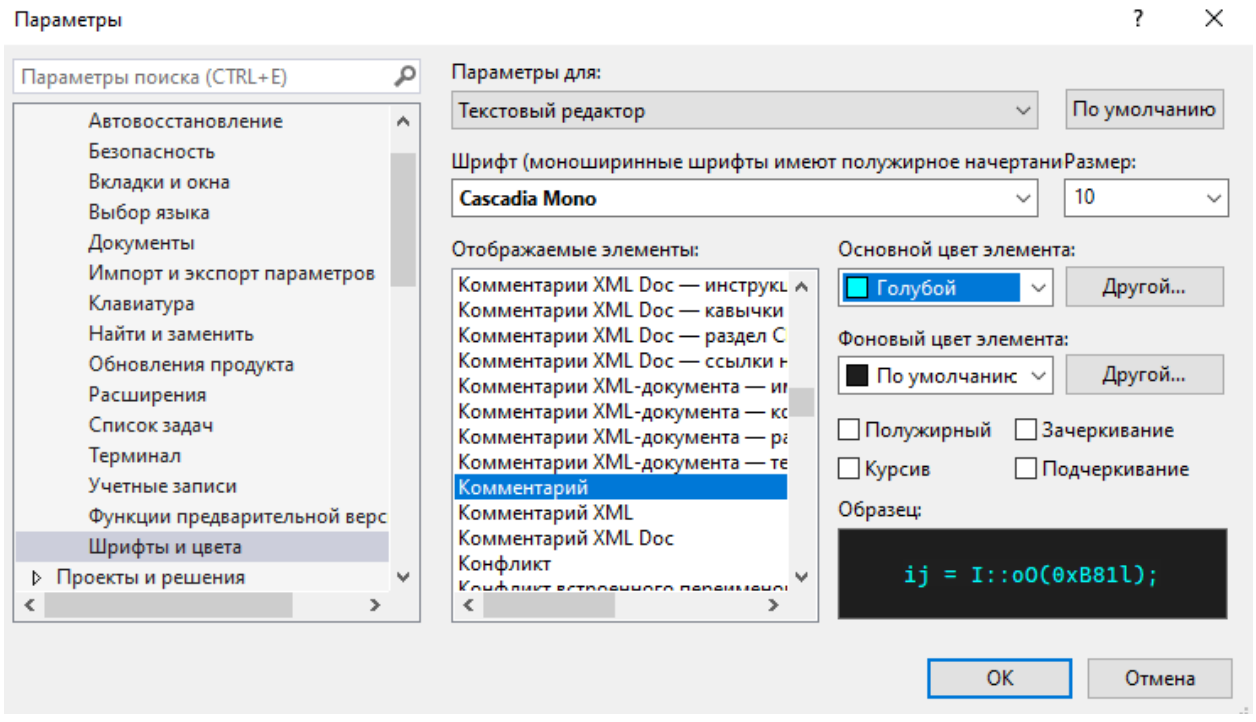


Рисунок 7 – Комментарии

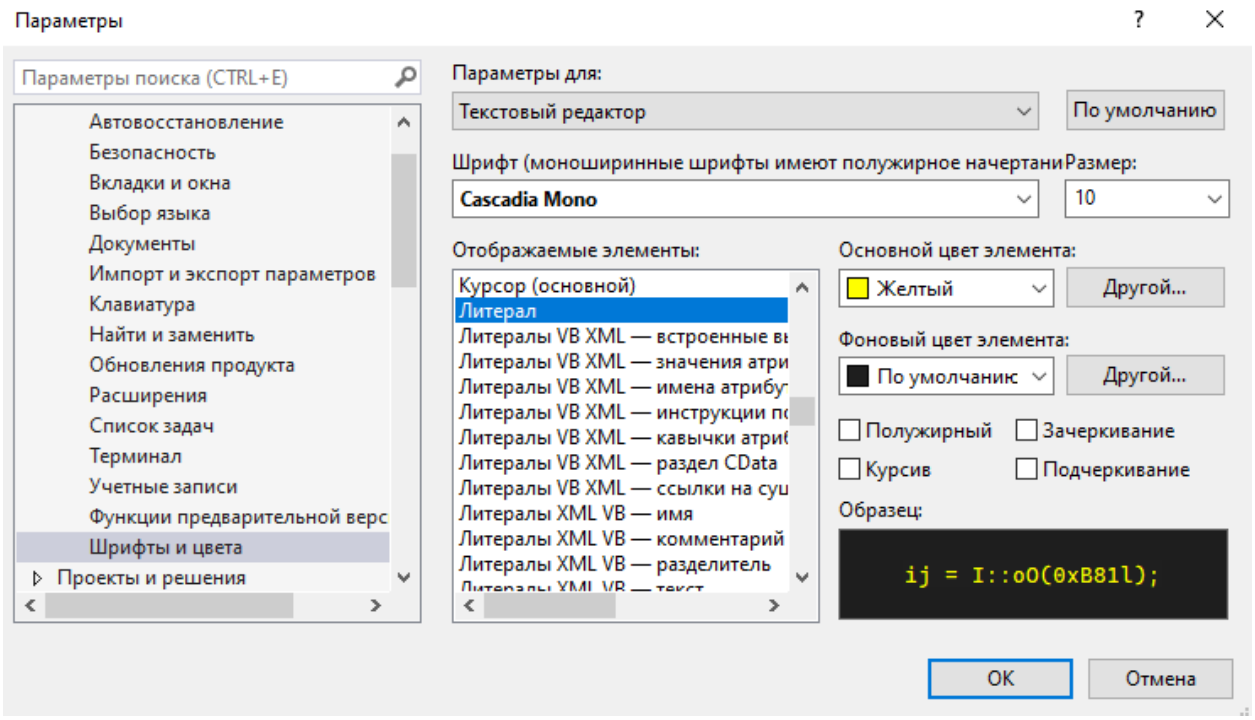


Рисунок 8 – Строковые литералы

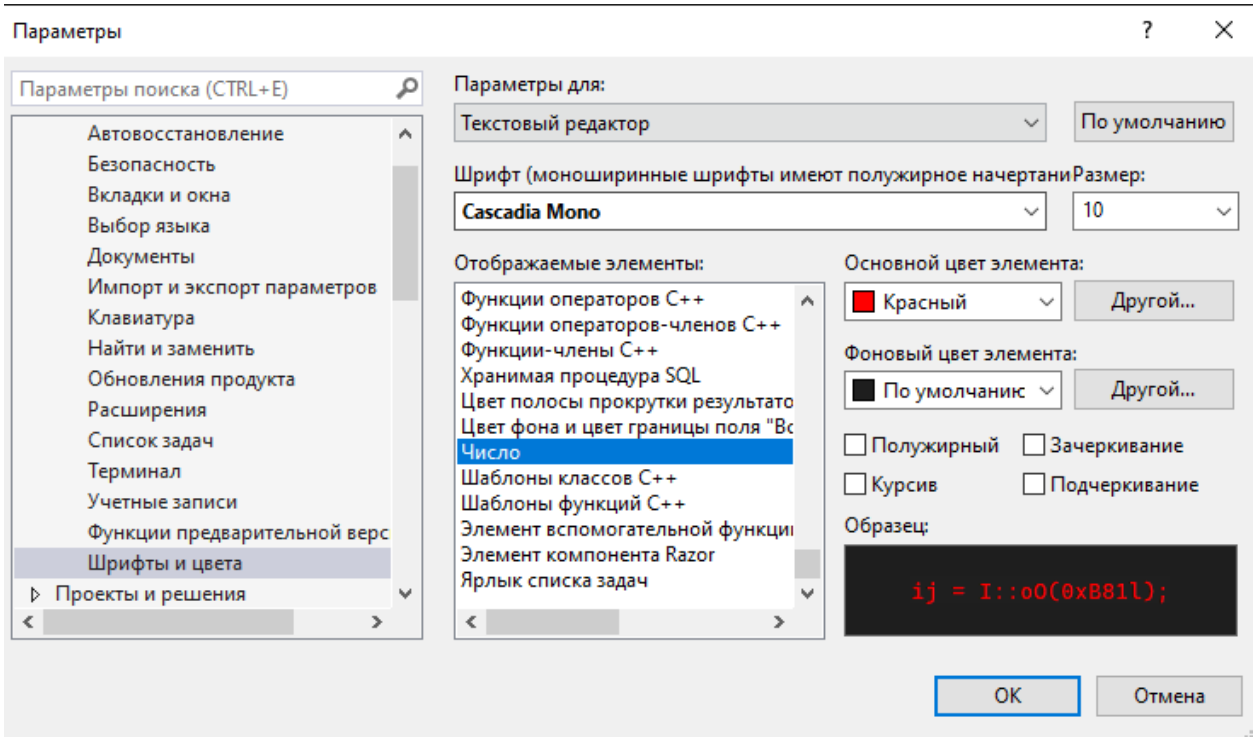


Рисунок 9 – Числа

1.3. Настройка меню и панели инструментов

Для создания своей панели инструментов необходимо в разделе «Средства → Настройка» выбрать раздел «Панели инструментов», нажать кнопку «Создать» и ввести название панели. Затем открыть раздел «Команды», выбрать созданную панель инструментов и добавить нужные команды (Рисунок 10).

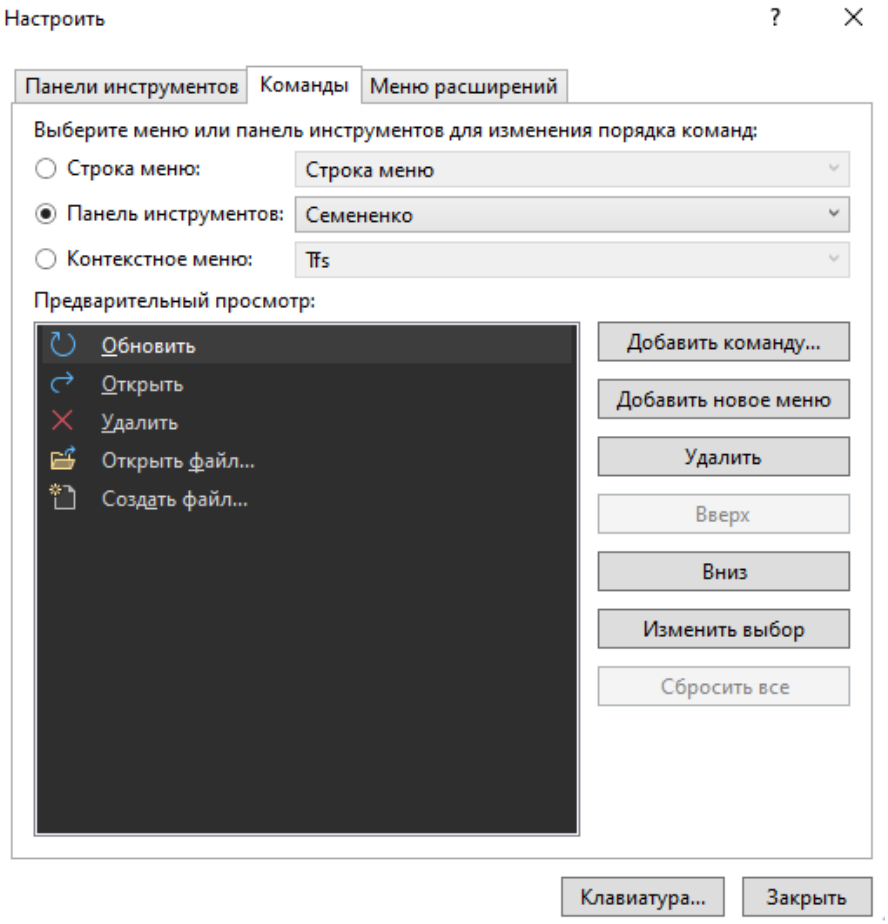


Рисунок 10 - Панель инструментов

Для назначения горячих клавиш нужно в разделе «Средства → Параметры → Окружение» выбрать раздел «Клавиатура» и выбрать необходимые функции (Рисунок 11 – 15).

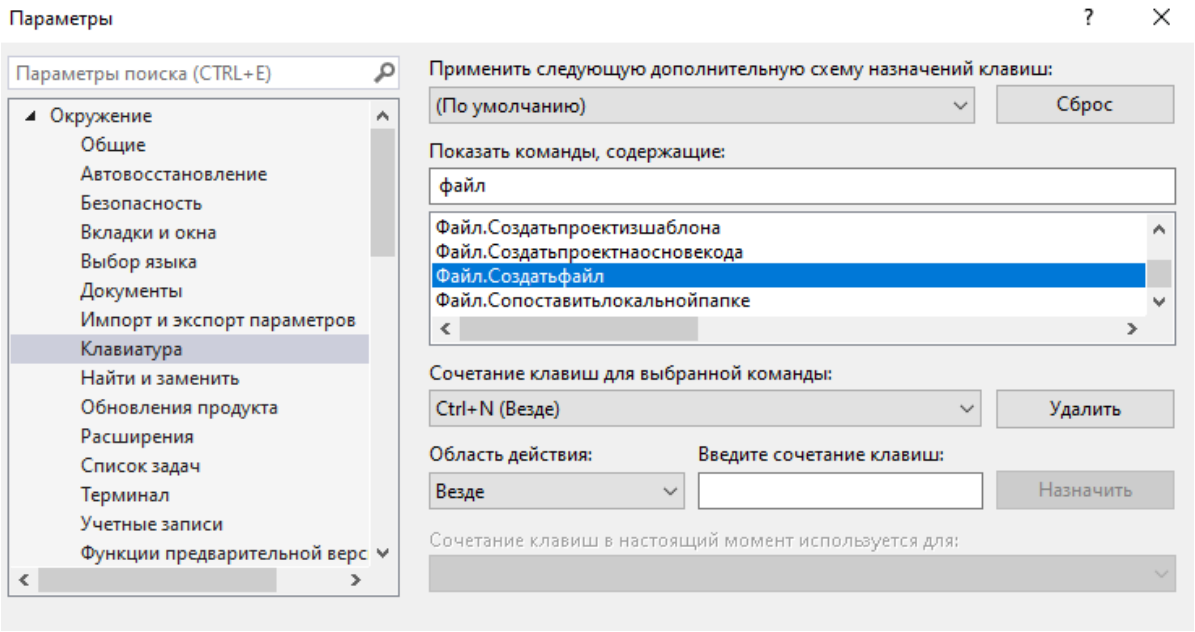


Рисунок 11 - Добавление клавиш

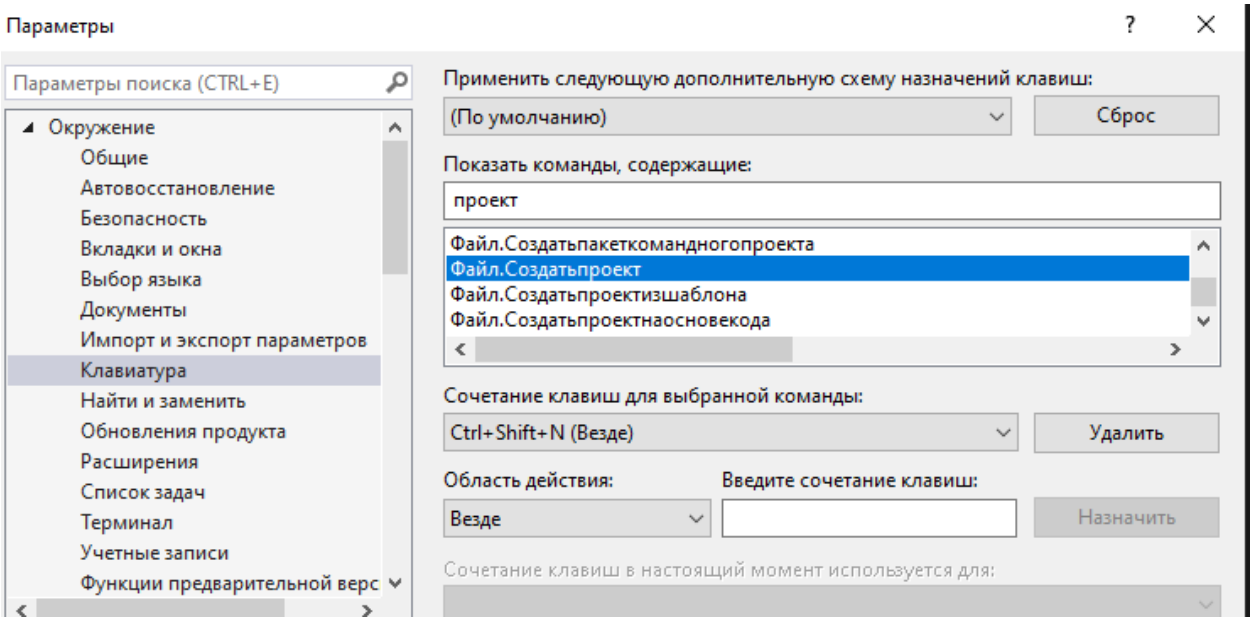


Рисунок 12

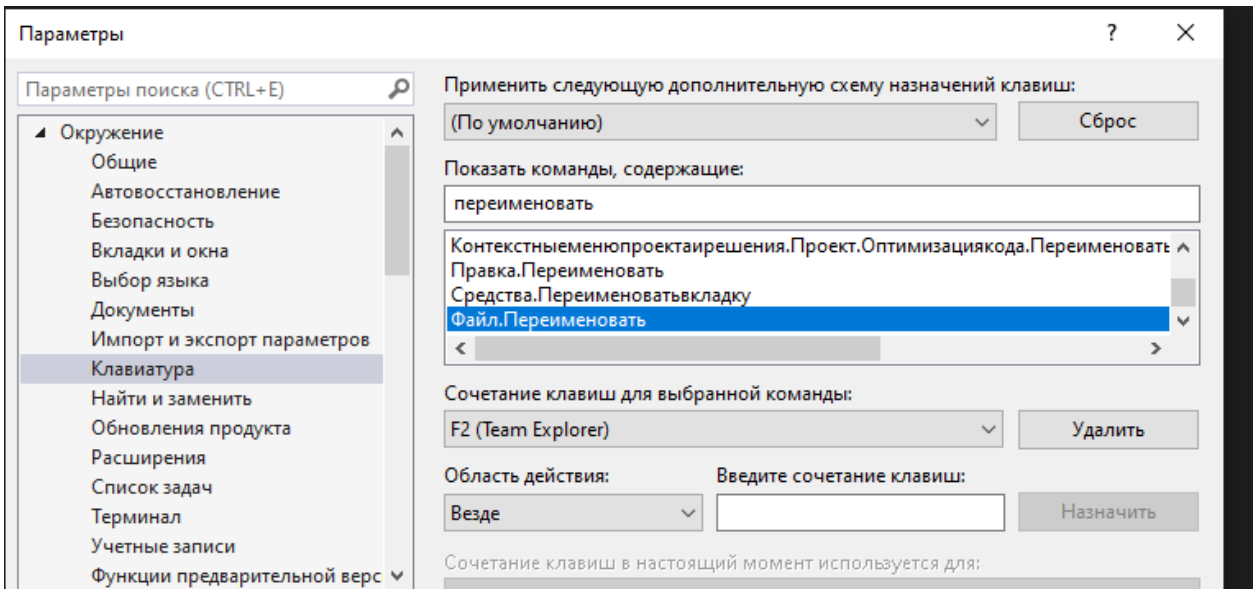


Рисунок 13

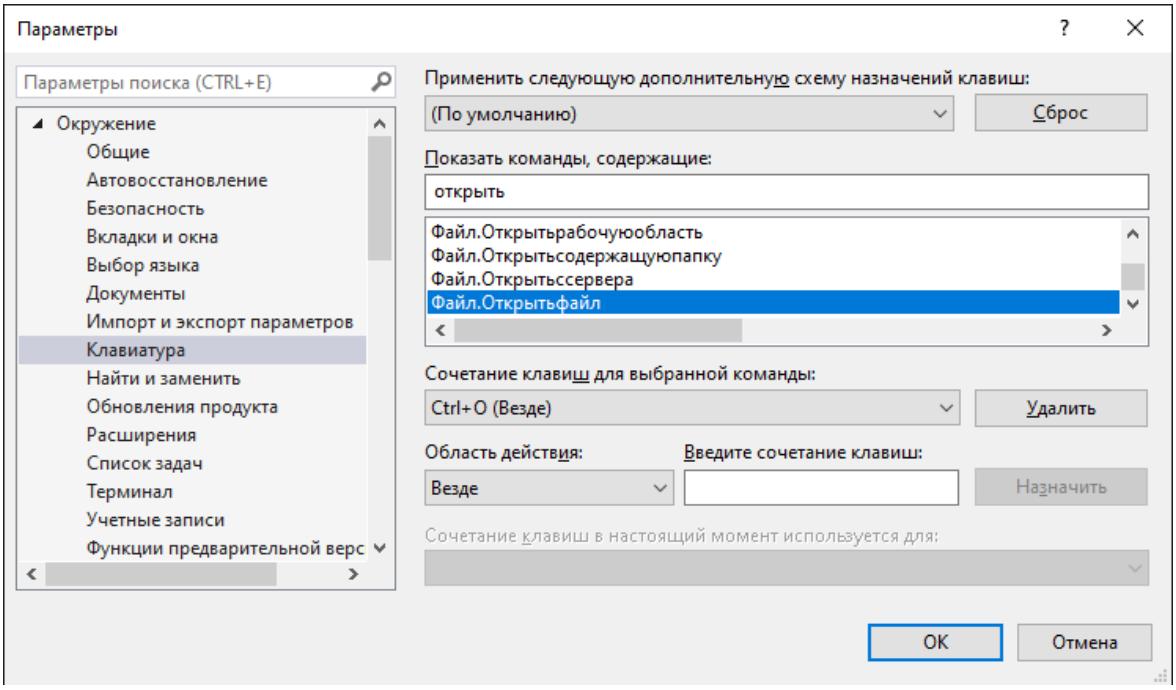


Рисунок 14

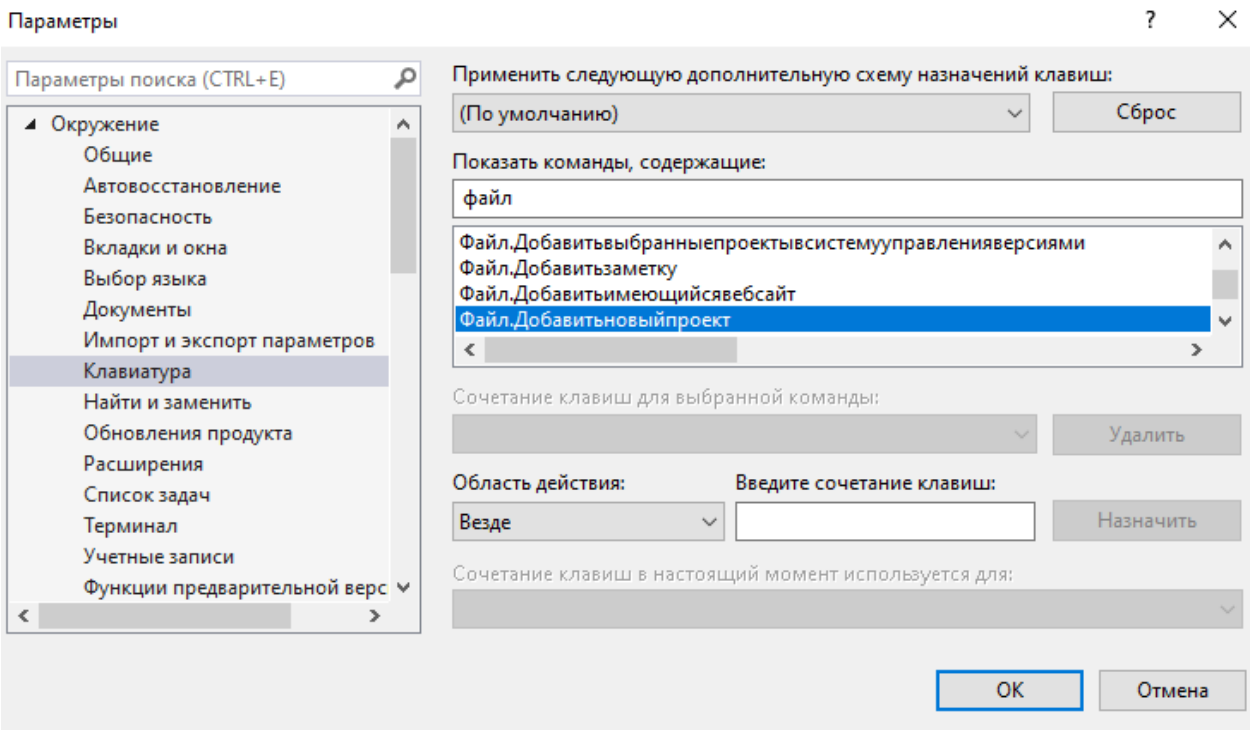


Рисунок 15

1.4. Настройка макетов окон

Для добавления окон необходимо открыть раздел «Вид → Другие окна» и выбрать нужные окна (Рисунок 16). Они отобразятся в макете (Рисунок 17).

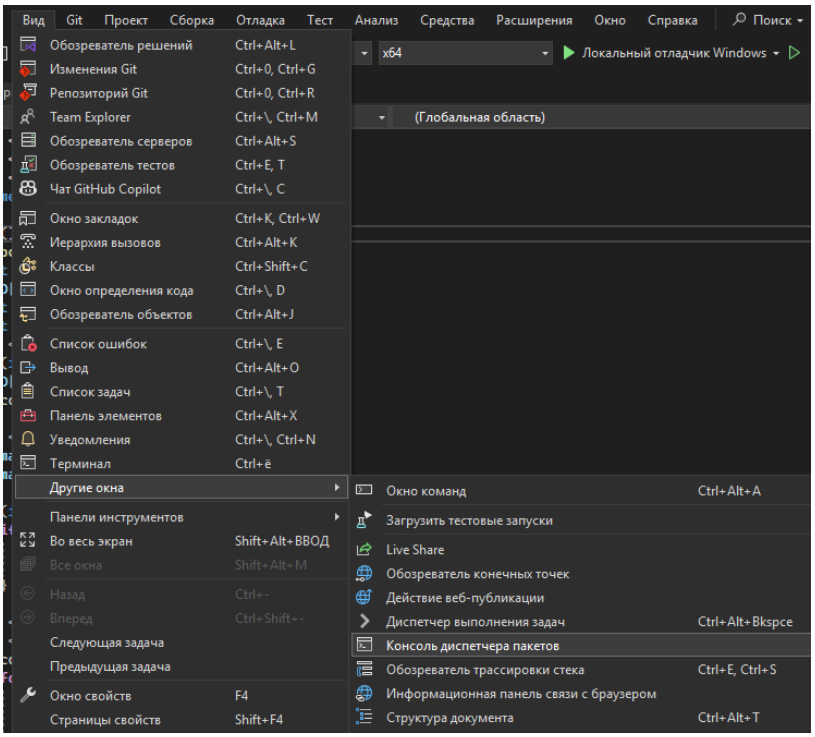


Рисунок 16 - Добавление окон

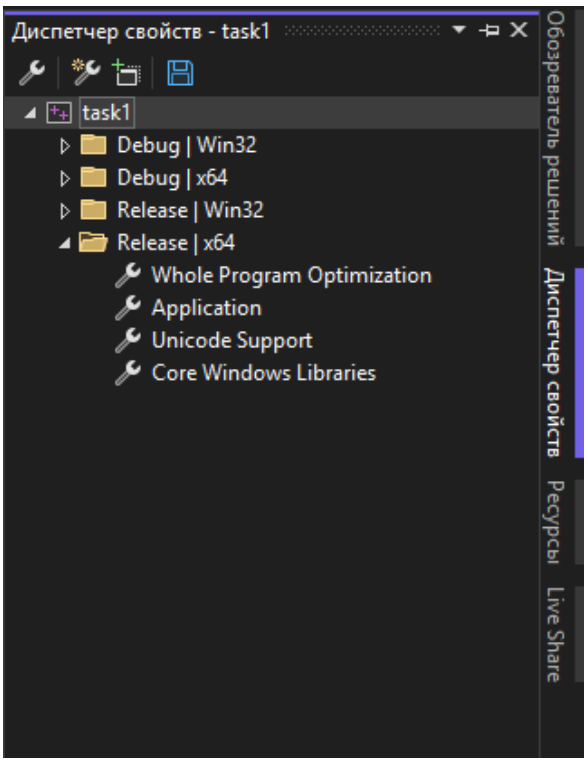


Рисунок 17 - Переключение окон

После настройки макета нужно его сохранить. Для этого необходимо зайти в раздел «Окно → Сохранить макет окна», ввести название и нажать кнопку «Ок» (Рисунок 18).

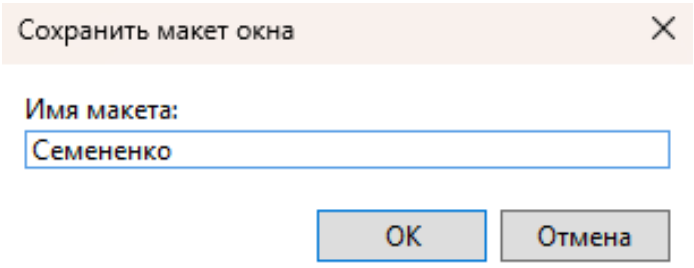


Рисунок 18 - Сохранение макета

1.5. Экспорт настроек

Нужно открыть раздел «Средства → Импорт и экспорт параметров» и поставить флажок на необходимой функции (Рисунок 19–21).

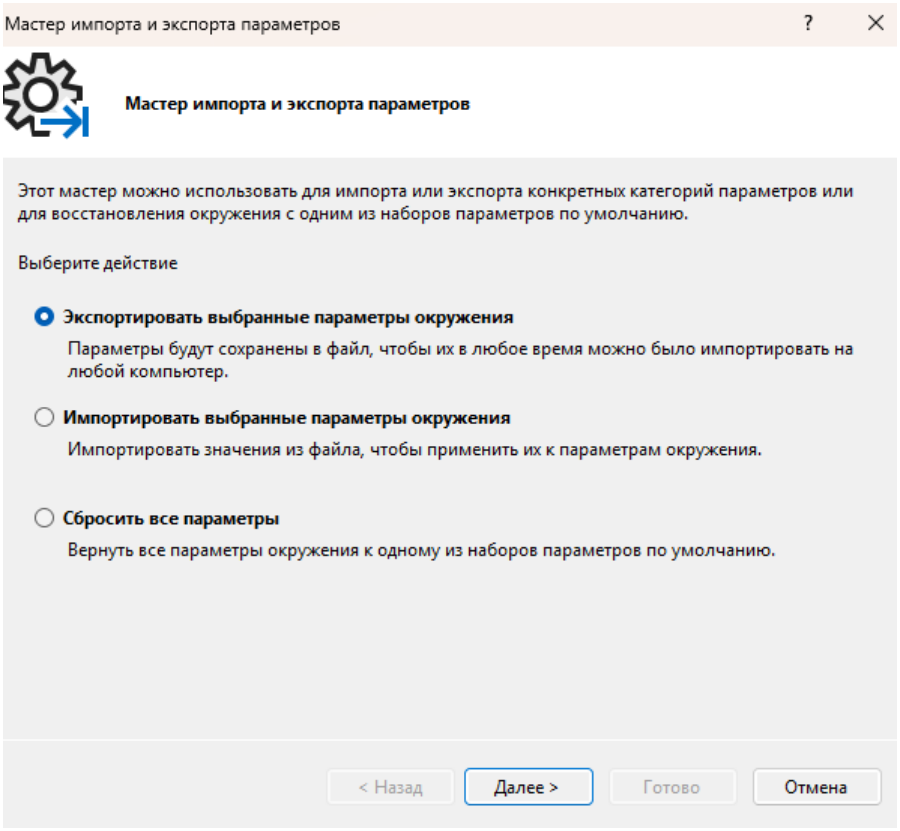


Рисунок 19 - Экспорт настроек

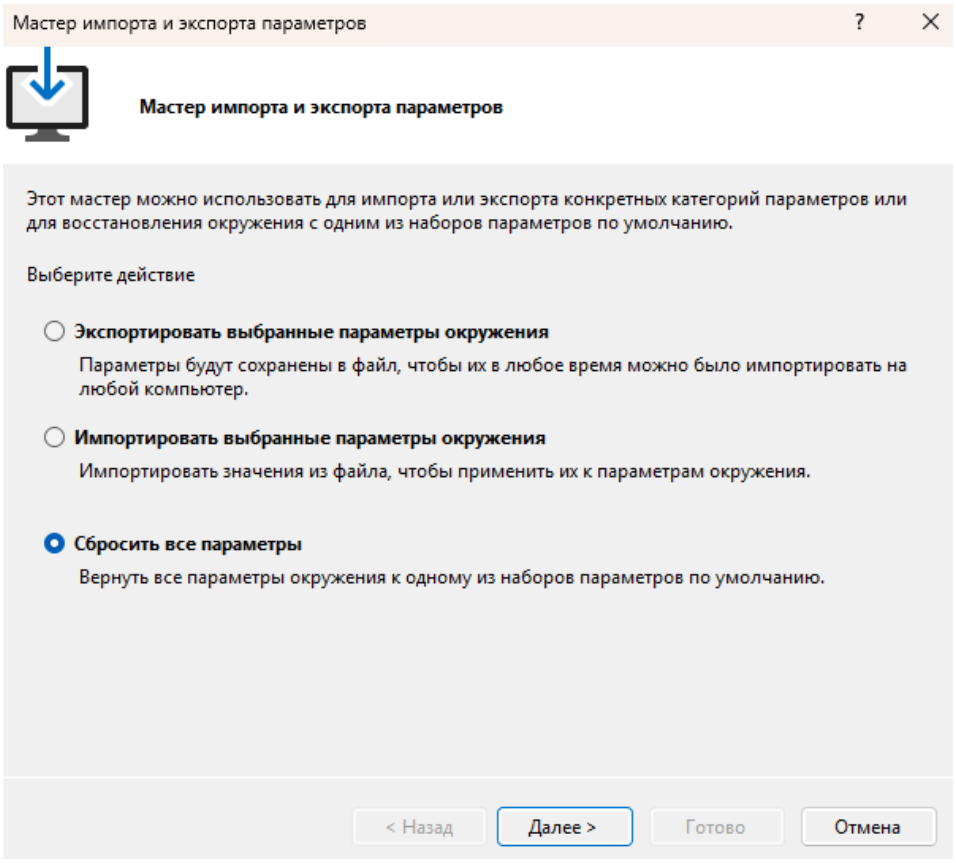


Рисунок 20 - Сброс настроек

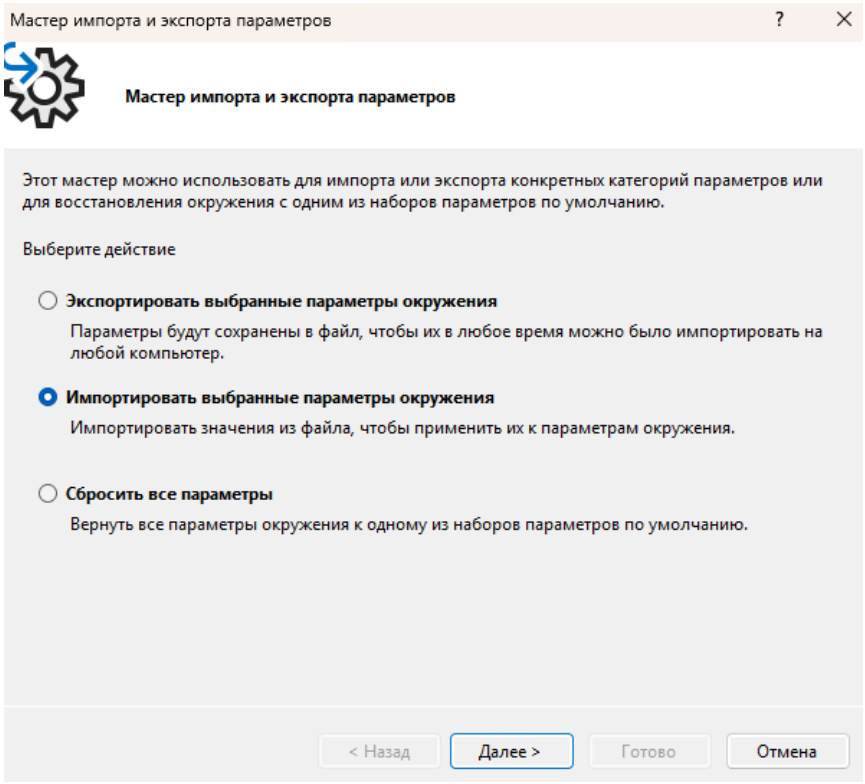


Рисунок 21 - Импорт настроек

Код программы

```
#include<iostream>
using namespace std;
int main () {
	int x, y;
	cin >> x;
	cin >> y;
	cout << x * y;
}
```

Практическая работа №2. «Отладка в Visual Studio»

Цель: освоить базовые и продвинутые техники отладки.

2.1. Навигация по коду с помощью отладчика

Нужно создать консольное приложение на C++, содержащее цикл (Рисунок 22), и установить точку останова на строке с циклом. Для этого необходимо нажать левой кнопкой мыши на область слева от нумерации строк на нужном уровне (Рисунок 23).

Код программы

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

int factorial(int n) {
    setlocale(0, "");
    if (n == 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
int main() {
    setlocale(0, "");
    cout << "Числа от 1 до 10:" << endl;
    for (int i = 1; i <= 10; ++i) {
        cout << i << " ";
    }
    cout << "\n\nФакториал числа 5: " << factorial(5) << endl;
    return 0;
}
```

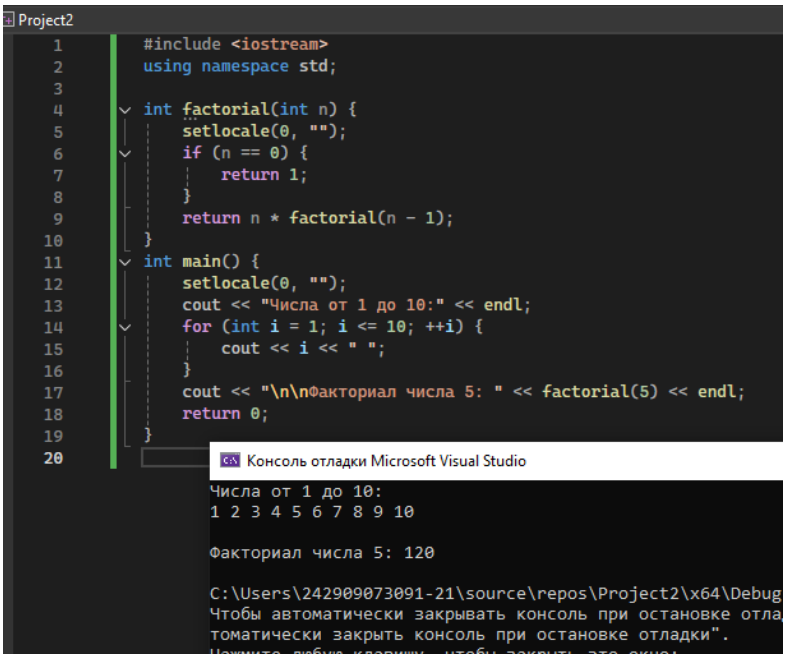


Рисунок 22 - Запуск

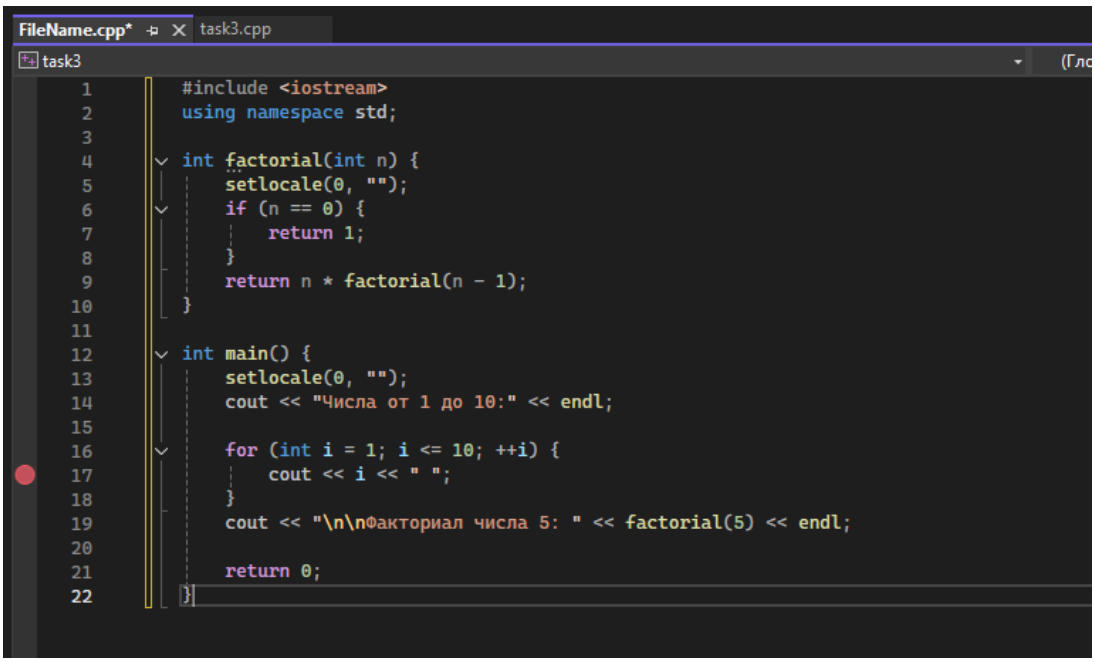


Рисунок 23 – Точка останова

Для запуска отладки нужно нажать F5. Программа запустится и остановится на строке с циклом. В окне локальных переменных будет видно, что переменная i еще не создана (Рисунок 24).

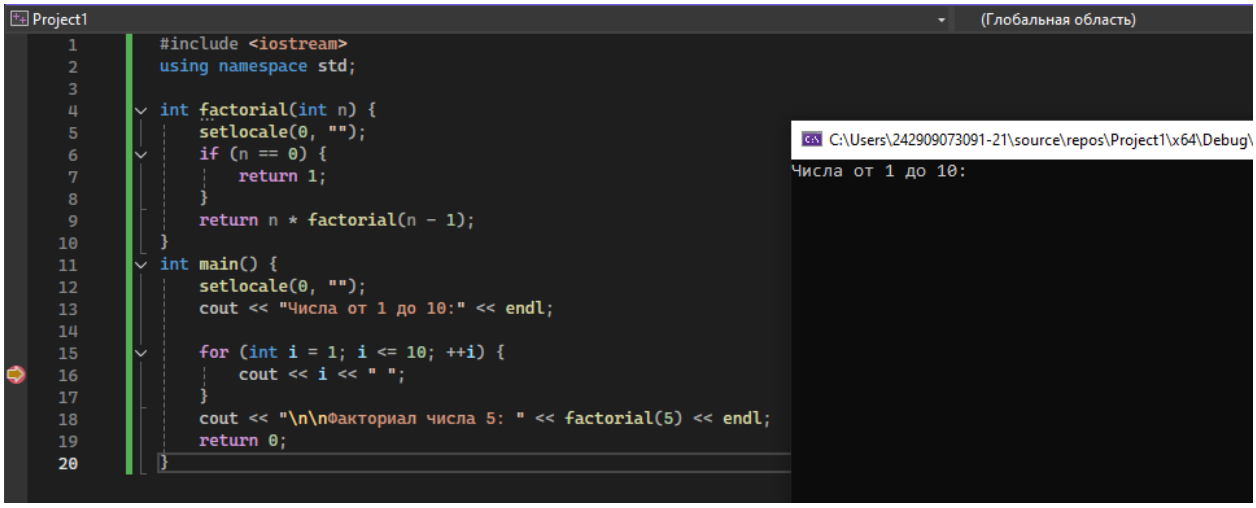


Рисунок 24 – Отладка (F5)

Необходимо использовать шаги отладки.

Шаг с обходом (F10). Он выполняет текущую строку без заглядывания внутрь функции. Если строка содержит вызов функции, то функция

выполнится за один шаг, отладчик остановится на следующей строке после вызова (Рисунок 25).

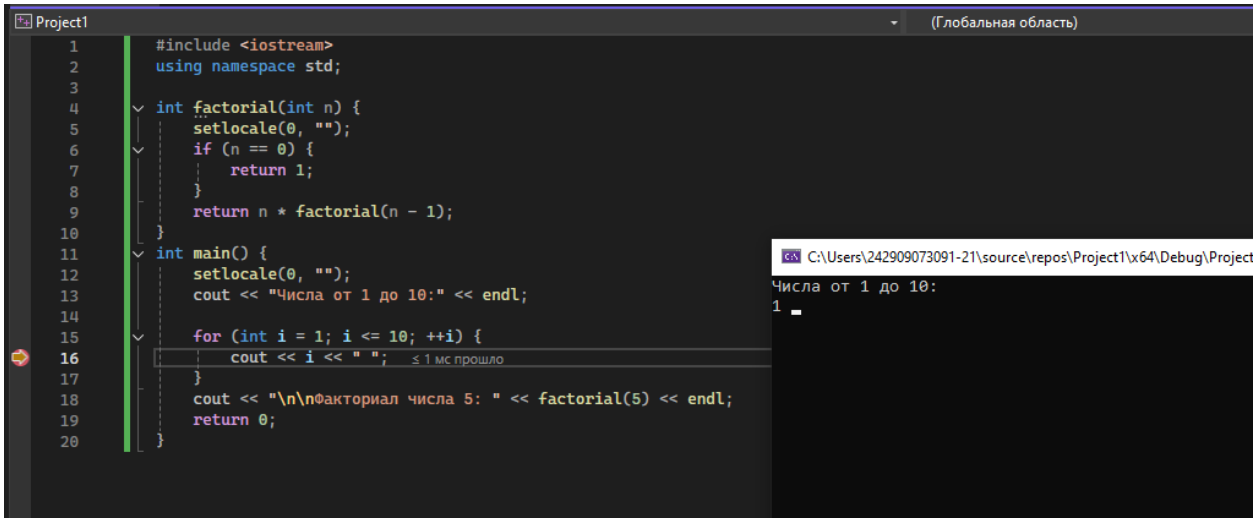


Рисунок 25 – Шаг с обходом (F10)

Шаг с заходом (F11). Он заходит внутрь вызываемой функции, если текущая строка содержит вызов функции. Отладчик переходит на первую строку вызываемой функции (Рисунок 26).

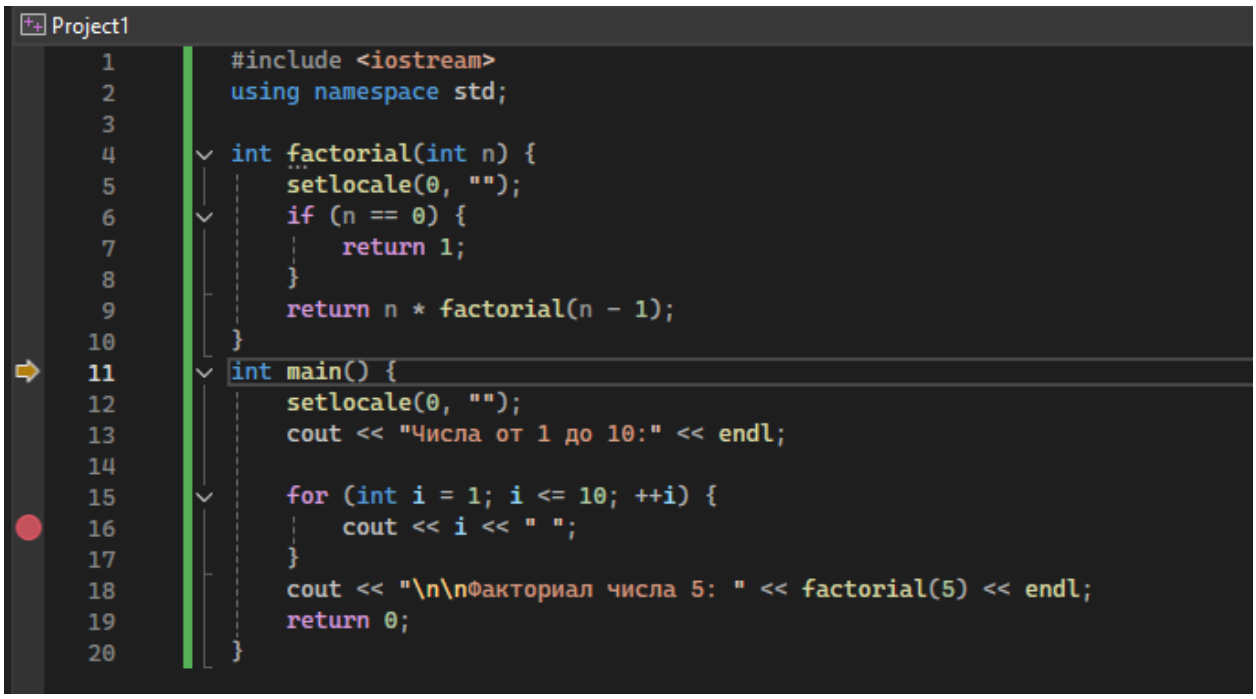


Рисунок 26 – Шаг с заходом (F11)

Шаг с выходом (Shift+F11). Выполняет текущую функцию до возврата и останавливается на строке, которая вызвала функцию (Рисунок 27).

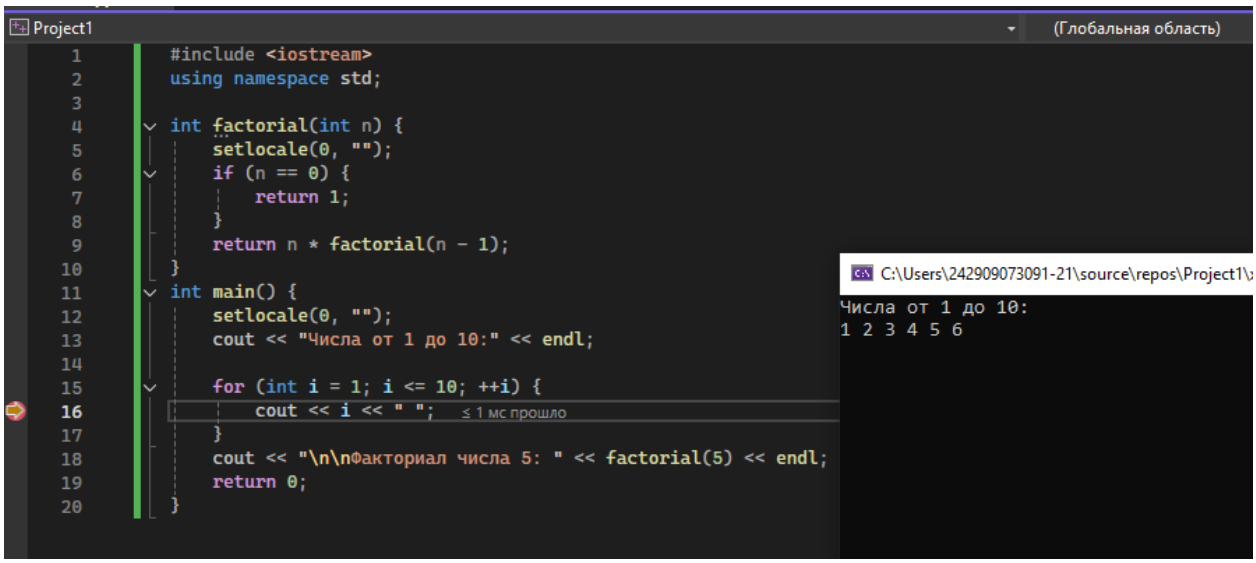


Рисунок 27 – Шаг с выходом (Shift+F11)

Нужно установить условную точку останова. Для этого необходимо нажать правой кнопкой мыши на точке останова в строке цикла. Выбрать «Условие» и в открывшемся окне ввести «i == 5» (Рисунок 28).

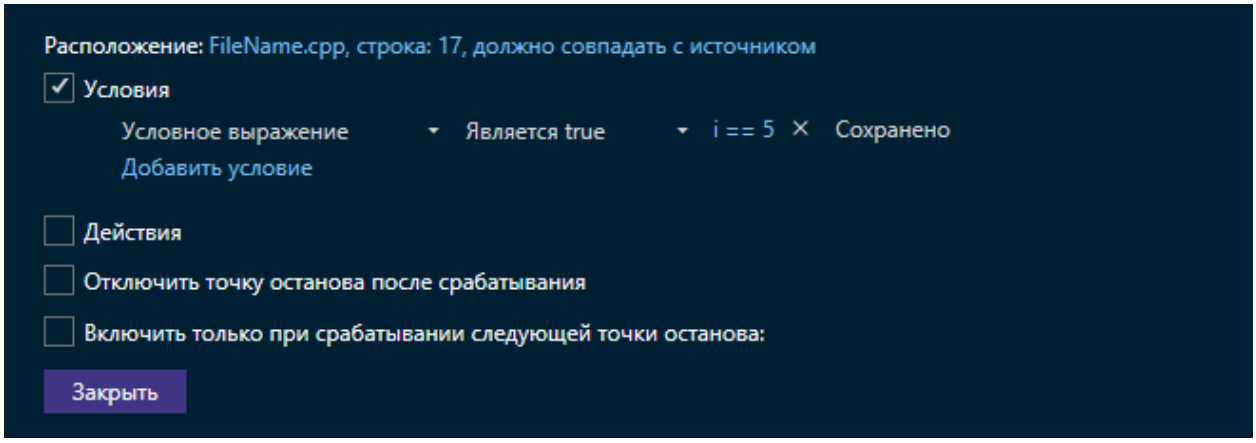


Рисунок 28 – Условная точка

Наблюдение за значениями переменных в окнах.

Видимые – показывает переменные, используемые в текущей и предыдущей строках кода. Показывает значения всех переменных и выражений, имеющихся втекущей выполняющейся строке кода или в предыдущей строке (Рисунок 29).

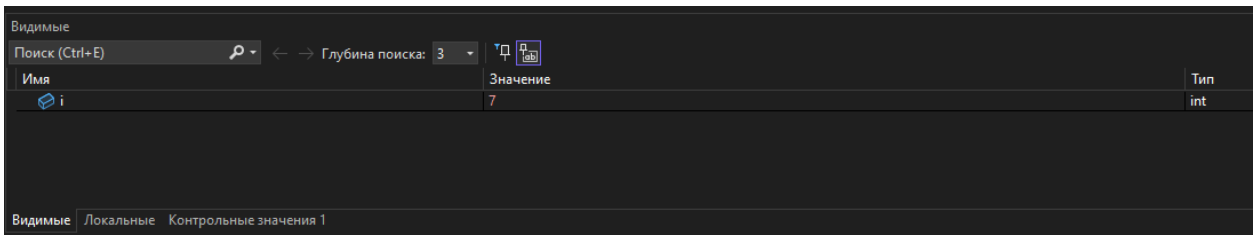


Рисунок 29 – Видимое окно

Локальные окна - отображают локальные переменные и их значения в текущей области выполнения кода. Включает переменные, которые объявлены в текущем методе или блоке, а также аргументы метода (Рисунок 30).

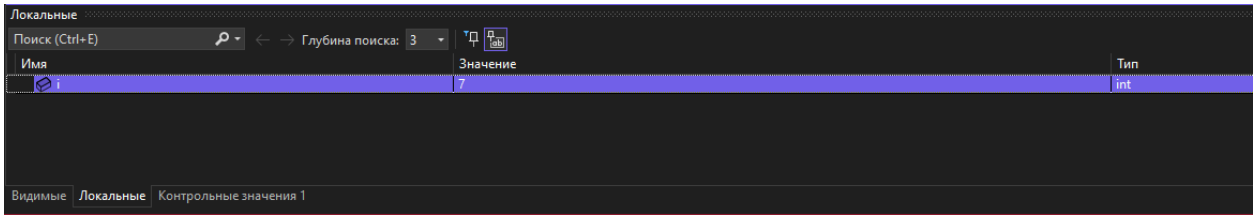


Рисунок 30 – Локальное окно

Контрольные значения - отслеживают значения переменных и выражений во время отладки программы. Отличие от других окон переменных в том, что в окне «Контрольные значения» всегда отображаются просматриваемые переменные (Рисунок 31).

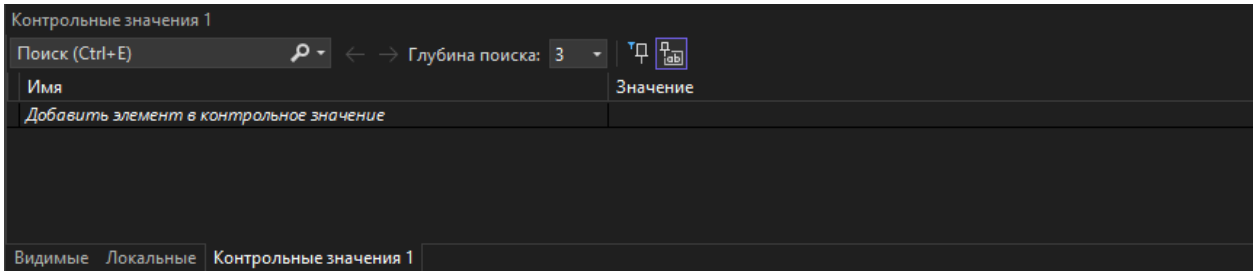


Рисунок 31 – Контрольные значения

Быстрая проверка – отображает одну переменную за раз, в отличие от «Контрольные значения». Предназначена для наблюдения за одной переменной или выражением. Доступна только во время сеанса отладки (Рисунок 32).

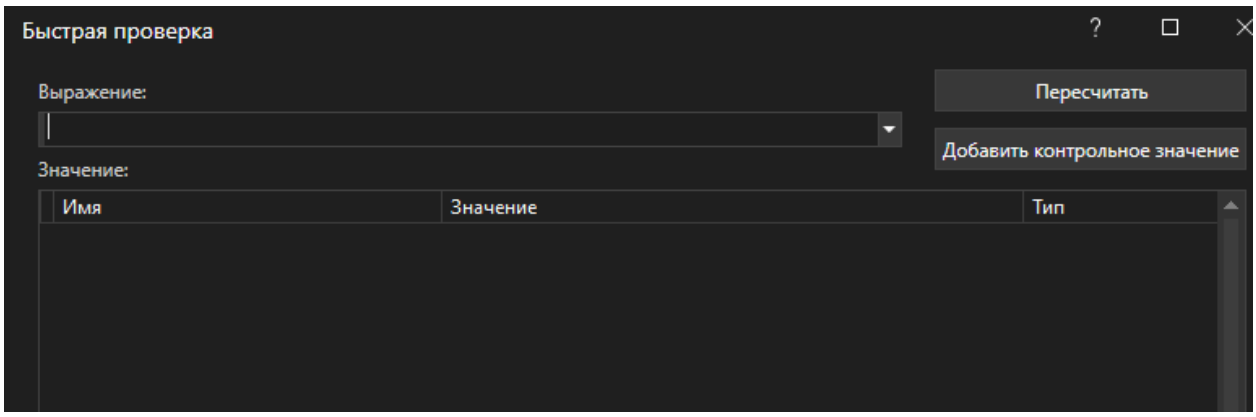


Рисунок 32 – Быстрая проверка

2.2. Управление исключениями

Добавьте код, генерирующий исключение (например, деление на ноль (System.DivideByZeroException)). (Рисунок 33).

Код программы

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

int safe_divide(int a, int b) {
    if (b == 0) {
        throw std::runtime_error("DivisionbyzeroException");
    }
    return a / b;
}
int main() {
    try {
        int x = 10;
        int y = 0;
        std::cout << "Result: " << safe_divide(x, y) << "\n";
    }
    catch (const std::exception& ex) {
        std::cerr << "Caught exception: " << ex.what() << "\n";
    }
    return 0;
}
```

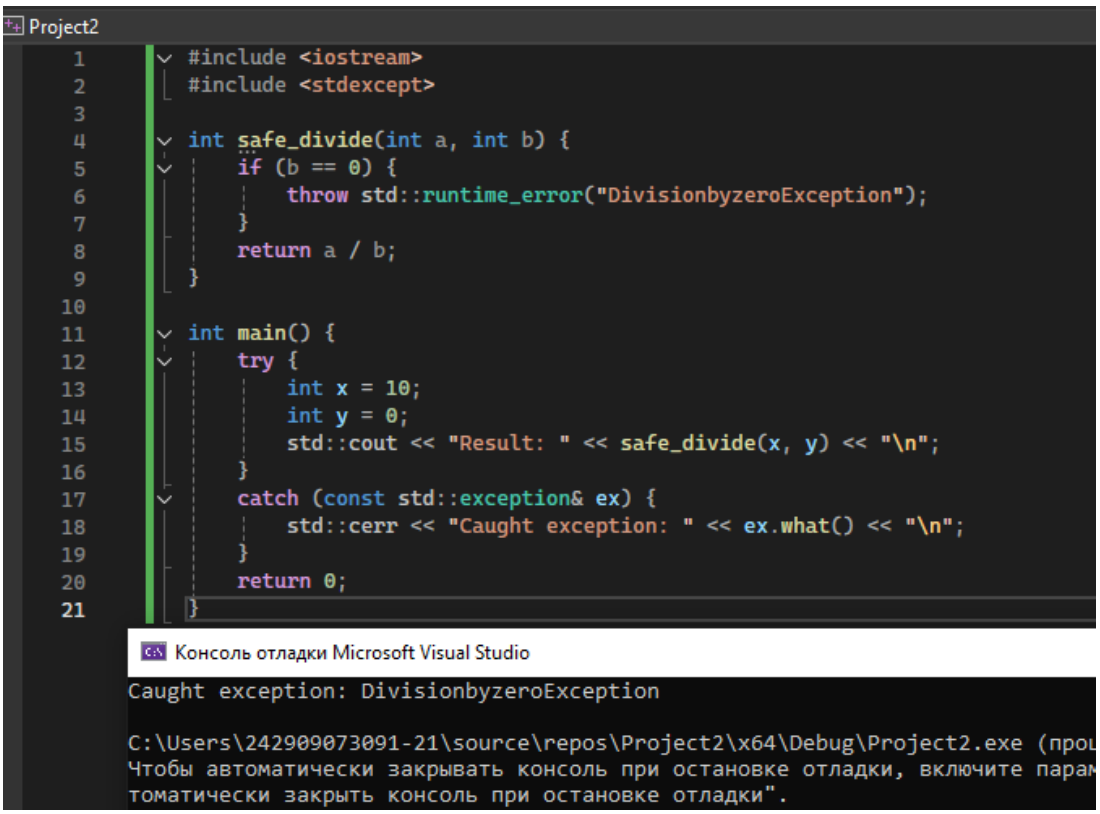


Рисунок 33

Настройте отладчик для прерывания при определённых исключениях: Отладка → Окна → Параметры исключения (только когда код запущен). Найдите ветку «C++ Exceptions». Убедитесь, что галочки для всех исключений установлены (отладчик будет прерываться в момент генерации исключения). (Рисунок 34)

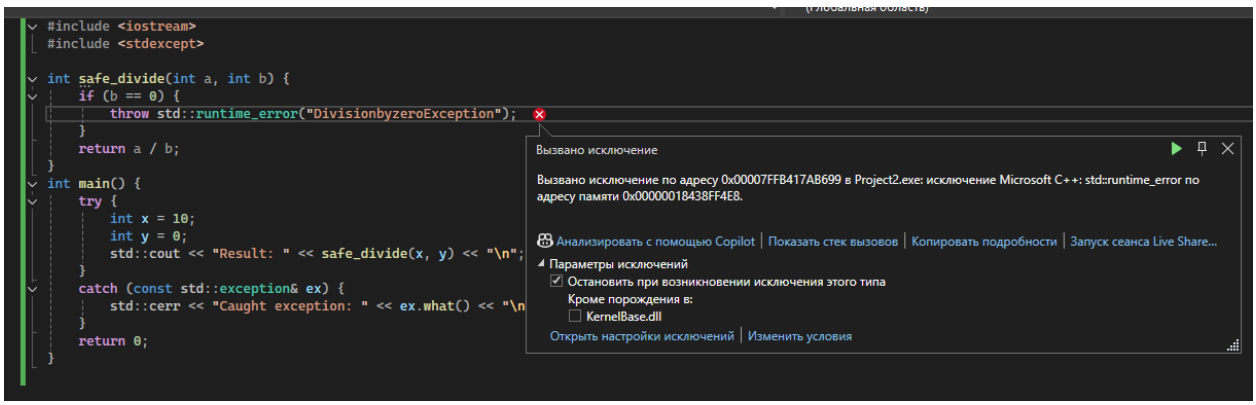


Рисунок 34 – Отладчик

2.3. Профилирование

Добавьте код с интенсивными вычислениями (например, цикл с большим количеством итераций (1000000) или работа с коллекциями). (Рисунок 35)

Код программы

```
#include<iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <chrono>
bool isPrime(int n) {
    setlocale(0, "");
    if (n <= 1) return false;
    for (int i = 2; i <= std::sqrt(n); ++i) {
        if (n % i == 0) return false;
    }
    return true;
}
int main() {
    setlocale(0, "");
    const int VECTOR_SIZE = 10'000'000;
    const int PRIME_LIMIT = 500'000;

    auto start = std::chrono::high_resolution_clock::now();

    std::cout << "Генерация и счётчиквектора " << VECTOR_SIZE << "
элементы..." << std::endl;
    std::vector<int>data(VECTOR_SIZE);

    for (int i = 0; i < VECTOR_SIZE; ++i) {
        data[i] = rand() % 1000000;
    }
    std::sort(data.begin(), data.end());

    std::cout << "Вычисление простыхчисел с точностью до " << PRIME_LIMIT <<
"..." << std::endl;
    int primeCount = 0;
    for (int i = 0; i < PRIME_LIMIT; ++i) {
        if (isPrime(i)) {
            primeCount++;
        }
    }
    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double>elapsed = end - start;
    std::cout << "\n--- Результат---" << std::endl;
    std::cout << "Наименьший элемент: " << data[0] << std::endl;
    std::cout << "Наибольший элемент: " << data[VECTOR_SIZE - 1] <<
std::endl;
    std::cout << "Найденные простые числа: " << primeCount << std::endl;
    std::cout << "Общее время выполнения: " << elapsed.count() << " секунд"
<< std::endl;
    return 0;
}
```

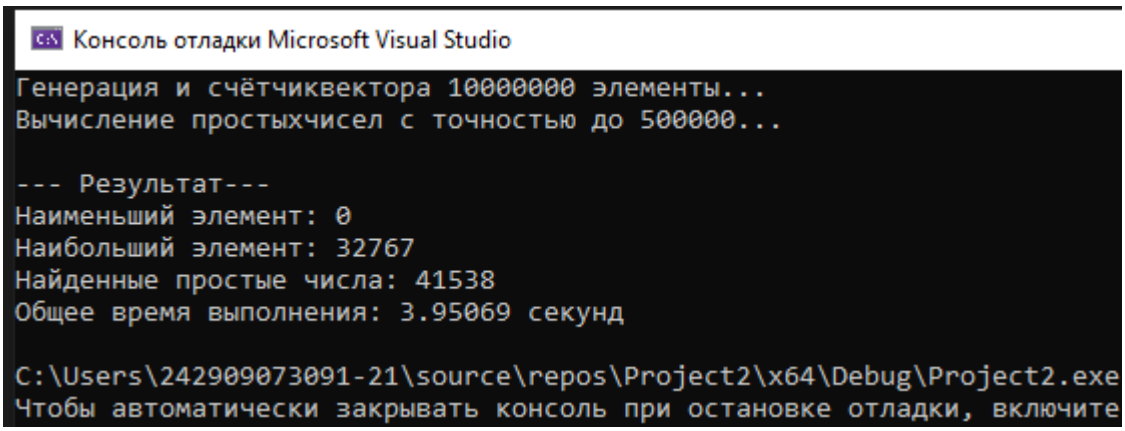


Рисунок 35

В разделе «Отладка → Профилировщик производительности» запустите:

- Анализ **Использование ЦП** для определения нагрузочных функций (Рисунок 36).
- Анализ **Использование памяти** для отслеживания выделения памяти и утечек (Рисунок 37).

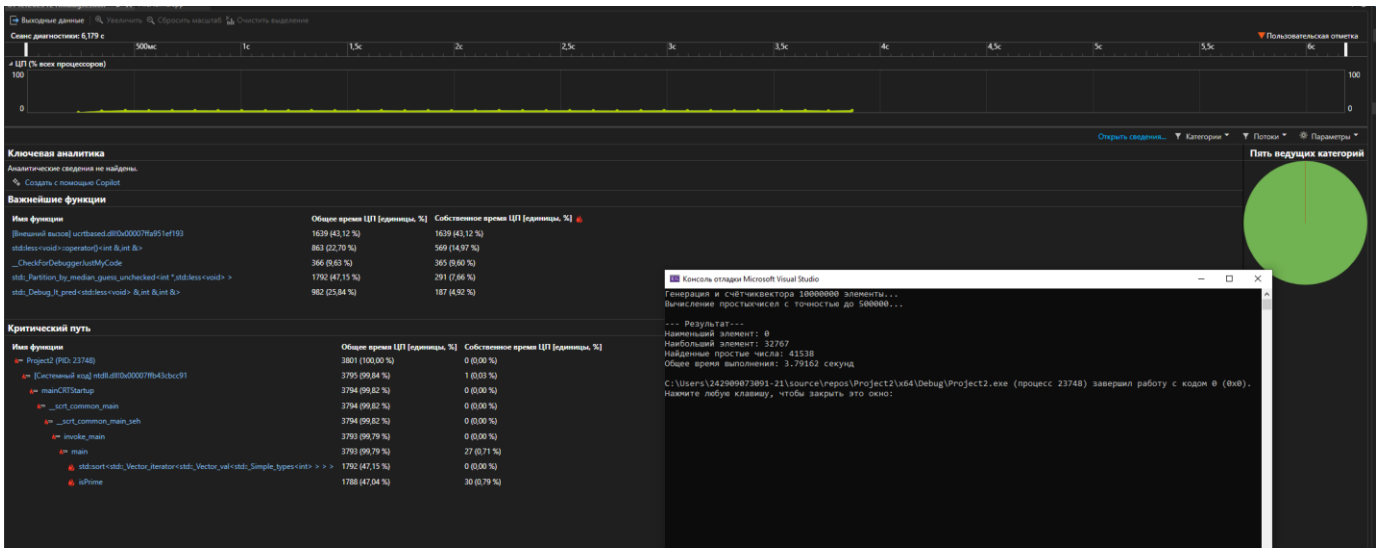


Рисунок 36 - Использование ЦП

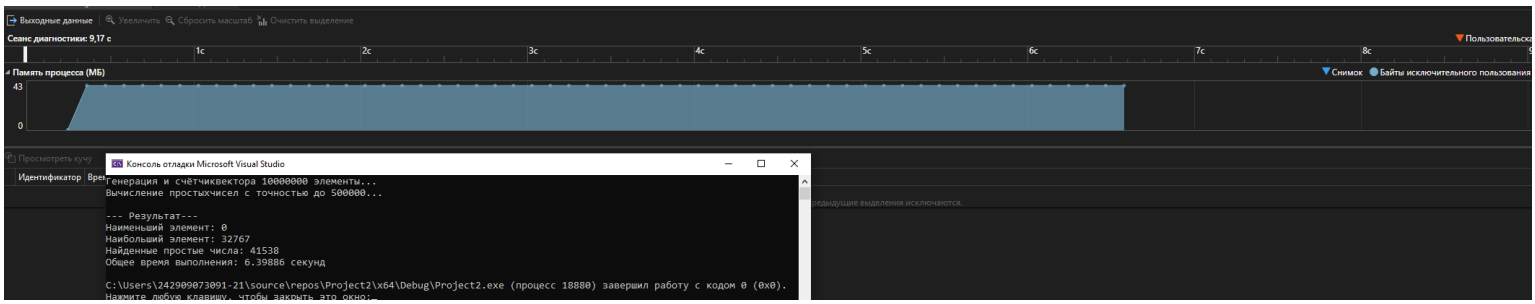


Рисунок 37 - Использование памяти

Использование ЦП.diagsession

Использование памяти.diagsession

Практическая работа №3: «Обеспечение качества кода на C++»

Цель: научиться писать чистый, понятный и хорошо документированный код.

Вариант 4 (по таблице). Создайте функцию для вычисления площади круга. Используйте математическую константу π. Функция принимает радиус, возвращает площадь. Протестируйте в main.

3.1. Самодокументирующийся код

Создайте новый консольный проект C++. Напишите свою функцию по варианту. Добавьте к функции XML-комментарии.

Код программы

```
#include <iostream>
#include <iomanip>
using namespace std;
double calculateCircleArea(double r) {

    const double PI = 3.14159265358979323846;
    return PI * r * r;
}
int main() {

    setlocale(0, "");
    double r;
    cout << "Введите радиус круга: ";
    cin >> r;

    if (r < 0) {
        cout << "Ошибка! Радиус не может быть отрицательным!" << endl;
    }

    else {
        double area = calculateCircleArea(r);


        cout << fixed << setprecision(2);
        cout << "Площадь круга = " << area << endl;
    }
    cout << fixed << setprecision(2);
    cout << "Тестовые значения: " << endl;


    cout << "Радиус = 4: Площадь = " << calculateCircleArea(4) << endl;
    cout << "Радиус = 7: Площадь = " << calculateCircleArea(7) << endl;
    cout << "Радиус = 11: Площадь = " << calculateCircleArea(11) << endl;
    return 0;
}
```

Скриншот всплывающей подсказки в Visual Studio при наведении на вашу функцию по варианту, комментарии соответствуют функции (Рисунок 38).

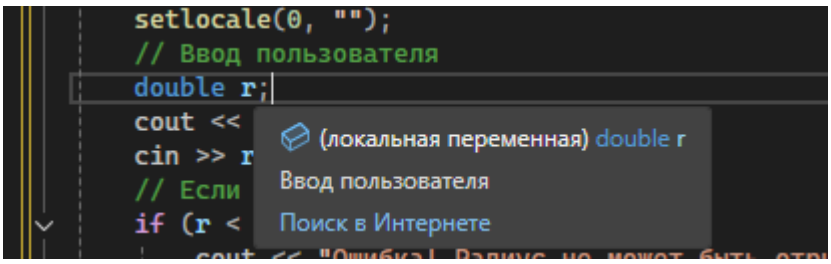


Рисунок 38 – Подсказка

3.2. Соглашение о кодировании

Цель: Оформить свой код согласно правилам C++

Код программы

```
#include <iostream>
#include <iomanip>
#include <clocale>

double calculateCircleArea(double radius)
{
    const double PI = 3.14159265358979323846;
    return PI * radius * radius;
}

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

    double radius;
    std::cout << "Введите радиус круга: ";
    std::cin >> radius;

    if (radius < 0) {
        std::cout << "Ошибка! Радиус не может быть отрицательным!" <<
std::endl;
    }
    else {
        double area = calculateCircleArea(radius);
        std::cout << std::fixed << std::setprecision(2);
        std::cout << "Площадь круга = " << area << std::endl;
    }

    std::cout << std::fixed << std::setprecision(2);
    std::cout << "Тестовые значения: " << std::endl;
    std::cout << "Радиус = 4: Площадь = " << calculateCircleArea(4) <<
std::endl;
    std::cout << "Радиус = 7: Площадь = " << calculateCircleArea(7) <<
std::endl;
    std::cout << "Радиус = 11: Площадь = " << calculateCircleArea(11) <<
std::endl;

    return 0;
}
```

3.3. Метрика Джилба (упрощенный вариант)\

Цель: Для вашей функции из Задания 1, рассчитать сложность по метрике Джилба:

Анализ функции calculateCircleArea:

```
double calculateCircleArea(double r) {
  const double PI = 3.14159265358979323846;  // объявление константы (не считается)
  return PI * r * r;                    // исполняемая строка (1)
}
```

Анализ функции main:

```
int main() {
  setlocale(LC_ALL, "Russian");
    double r;      // исполняемая строка (1)


  cout << "Введите радиус круга: ";
    cin >> r;
  if (r < 0) {                    // условный переход (считаем в η2)
    cout << "Ошибка! Радиус не может быть отрицательным!" << endl;
  }
  else {
    double area = calculateCircleArea(r);    // исполняемая строка (1)


    cout << fixed << setprecision(2);
    cout << "Площадь круга = " << area << endl;
  }
  cout << fixed << setprecision(2);
  cout << "Тестовые значения: " << endl;
  cout << "Радиус = 4: Площадь = " << calculateCircleArea(4) << endl;
  cout << "Радиус = 7: Площадь = " << calculateCircleArea(7) << endl;
  cout << "Радиус = 11: Площадь = " << calculateCircleArea(11) << endl;
  return 0;                       // исполняемая строка (1)
}
```

Расчет метрики Джилба:

для функции calculateCircleArea**:**

- η1 = 1 (одна исполняемая строка — return)
- η2 = 0 (нет условных переходов)
- **Gillb = 1 + 0 = 1**

Для функции main**:**

- η1 = 3 (все исполняемые строки)
- η2 = 1 (один условный переход if-else)
- **Gillb = 3 + 1 = 4**

Для всей программы:

- η1 = 4 (все исполняемые строки)
- η2 = 1 (один условный переход)
- **Gillb = 4 + 1 = 5**

Вывод:

- Функция calculateCircleArea имеет **низкую сложность** (Gillb = 1)
- Функция main имеет **низкую сложность** (Gillb = 4)
- Программа в целом имеет **низкую сложность** (Gillb = 5), что приемлемо для учебной программы.

Практическая работа №4 «Работа с реестром ОС Windows»

Цель: Понять что такое реестр, для чего нужен. Научиться работать с файлами.

Вариант 4

1. Сохранить и вывести любимый город.

2. Сохранить цвет консоли для текста приветствия.

4.1. Задание

1. **Создать консольное приложение**.

2. **Реализовать функционал** для сохранения и восстановления из файла, смотреть свой вариант.

3. **Обеспечить корректный вывод сохраненных данных** при его перезапуске.

Код программы

```
#include <iostream>
#include <windows.h>
#include <string>
#include <vector>

const char* REG_PATH = "Software\\MyFavoriteCityApp";

void SaveToRegistry(const std::string& city, DWORD color) {
    HKEY hKey;
    if (RegCreateKeyExA(HKEY_CURRENT_USER, REG_PATH, 0, NULL,
REG_OPTION_NON_VOLATILE, KEY_WRITE, NULL, &hKey, NULL) == ERROR_SUCCESS) {
        RegSetValueExA(hKey, "FavoriteCity", 0, REG_SZ, (const
BYTE*)city.c_str(), (DWORD)(city.length() + 1));
        RegSetValueExA(hKey, "ConsoleColor", 0, REG_DWORD, (const
BYTE*)&color, sizeof(DWORD));
        RegCloseKey(hKey);
        std::cout << "Данные сохранены в реестр.\n";
    }
    else {
        std::cerr << "Ошибка при открытии реестра для записи.\n";
    }
}

bool LoadFromRegistry(std::string& city, DWORD& color) {
    HKEY hKey;
    if (RegOpenKeyExA(HKEY_CURRENT_USER, REG_PATH, 0, KEY_READ, &hKey) ==
ERROR_SUCCESS) {
        char buffer[255];
        DWORD bufferSize = sizeof(buffer);
        DWORD colorSize = sizeof(DWORD);

        if (RegQueryValueExA(hKey, "FavoriteCity", NULL, NULL,
(LPBYTE)buffer, &bufferSize) == ERROR_SUCCESS) {
```

```
                city = buffer;
            }
            else {
                city = "Неизвестно";
            }

            if (RegQueryValueExA(hKey, "ConsoleColor", NULL, NULL,
(LPBYTE)&color, &colorSize) != ERROR_SUCCESS) {
                color = 7; // По умолчанию стандартный
            }
            RegCloseKey(hKey);
            return true;
        }
        return false;
    }

    void SetColor(DWORD color) {
        HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
        SetConsoleTextAttribute(hConsole, (WORD)color);
    }

    int main() {
        setlocale(LC_ALL, "Russian");
        std::string city;
        DWORD color = 7;

        if (LoadFromRegistry(city, color)) {
            SetColor(color);
            std::cout << "Введите Ваш любимый город: " << city << std::endl;
            SetColor(7); // Сброс цвета на стандартный
        }
        else {
            std::cout << "Сохраненные данные не найдены.\n";
        }

        int choice;
        do {
            std::cout << "\n--- Меню ---\n";
            std::cout << "1. Изменить любимый город и цвет\n";
            std::cout << "2. Выход\n";
            std::cout << "Выбор: ";
            std::cin >> choice;

            if (choice == 1) {
                std::cout << "Введите название города: ";
                std::cin.ignore();
                std::getline(std::cin, city);

                std::cout << "Выберите цвет текста (1 - Синий, 2 - Зеленый, 3 -
голубой, 4 - Красный, 5 - фиолетовый, 6 - Желтый, 7 - Белый): ";
                std::cin >> color;

                SaveToRegistry(city, color);

                SetColor(color);
                std::cout << "\nДанные обновлены. Ваш любимый город: " << city <<
"!\n";
                SetColor(7);
            }
        } while (choice != 2);
        return 0;
    }
```

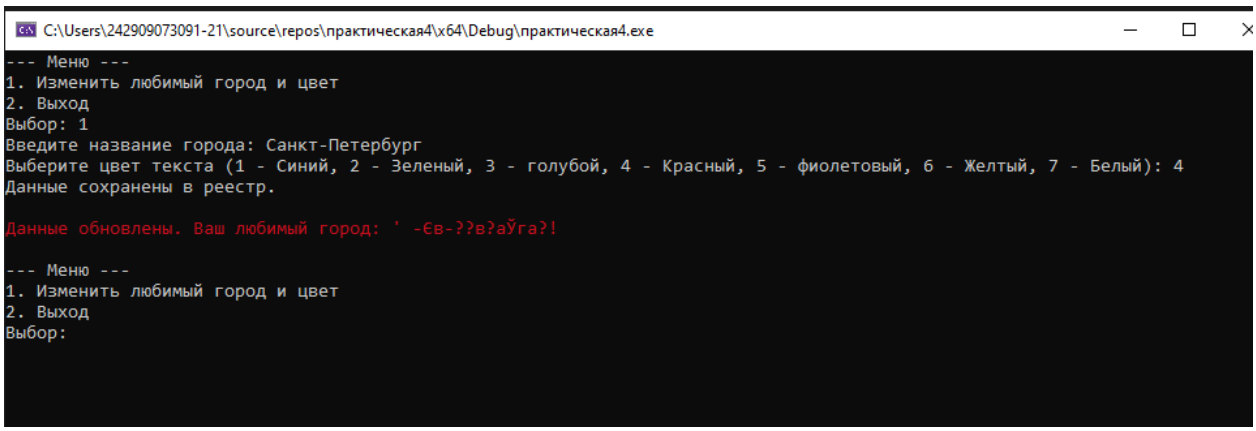


Рисунок 39 – Запуск

Практическая работа №5 «Задание для тестировщика»

Цель: Провести комплексное тестирование сайта с целью оценки его качества, удобства использования и выявления потенциальных дефектов.

Задача: Вам необходимо проверить сайт как с точки зрения рядового пользователя, так и с технической стороны. (Вариант 4 - https://www.spbstu.ru/).

5.1. Функциональное тестирование (Functional Testing)

- **Навигация:**

Все элементы кликабельны, ссылки переводят на другие страницы сайта, предлагающие большую информацию об учреждении.

Навигационное меню работает исправно. Каждый раздел открывается, выдает нужную информацию (Рисунок 40).

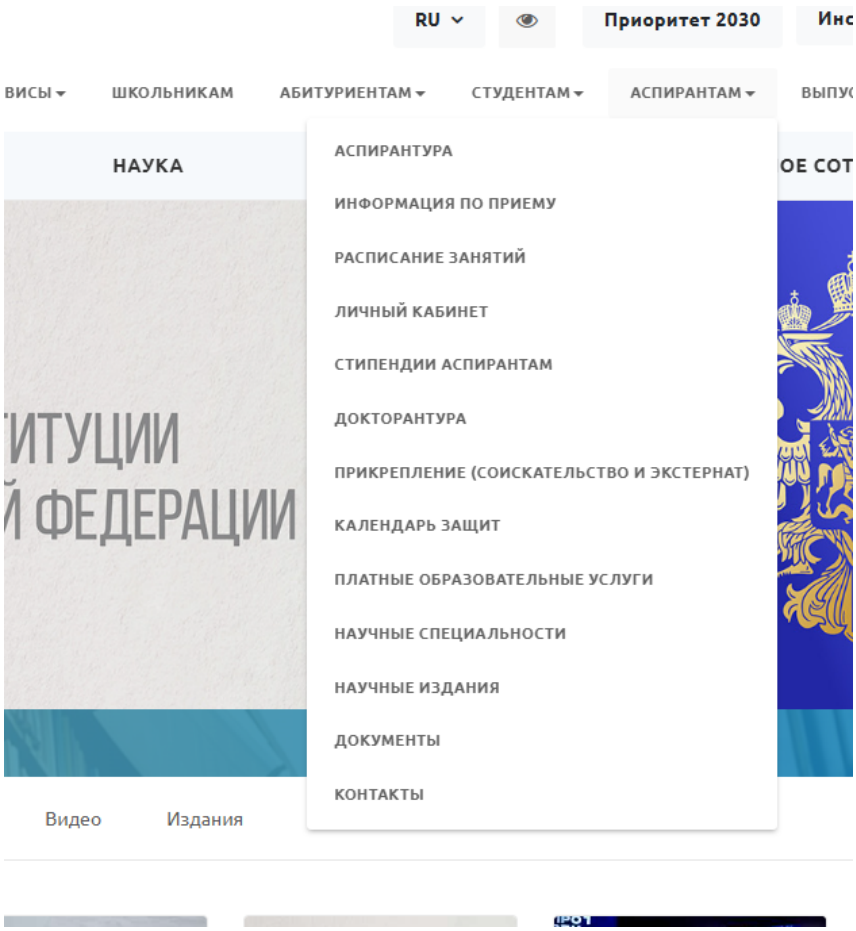


Рисунок 40 – Навигационное меню

Все ссылки в футере кликабельны. При нажатии открывается новый сайт с информацией (например, при нажатии на иконку «ВКонтакте» пользователя переводят на страницу сайта в этой соц. сети). (Рисунок 41)

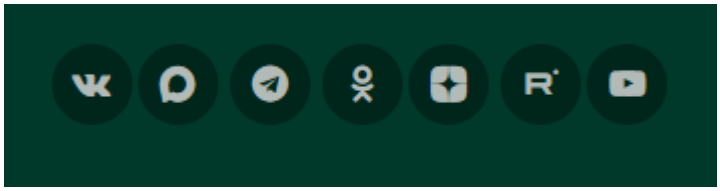


Рисунок 41 – Ссылки в футере

Поисковая строка помогает пользователю быстро найти нужную информацию на сайте. Предлагает все новости и разделы, имеющие в названии (или кратком описании) введенное слово. (Рисунок 42)

Рисунок 42 – Поисковая строка

- **Формы:**

При некорректном вводе данных сайт указывает на ошибку (Рисунок 43).

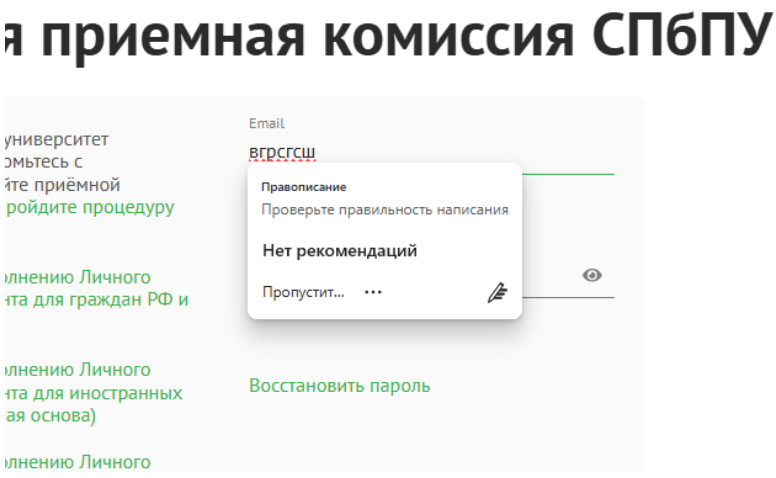


Рисунок 43 – Проверка правильности ввода

Если пользователь не заполняет обязательные поля, сайт не дает совершить операцию. (Рисунок 44)

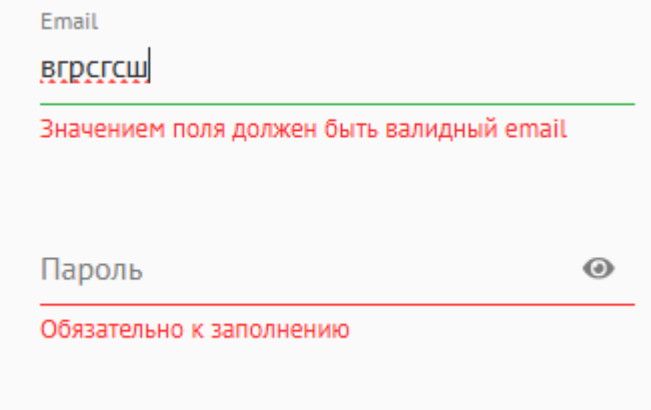


Рисунок 44 – Поля

При правильном заполнении формы на посту приходит уведомление о действии на сайте. После отправки формы поля очищаются.

- **Интерактивные элементы:**

Все карусели, аккордеоны работают корректно. При нажатии на один из предлагающихся разделов, пользователь попадает на нужную страницу сайта.

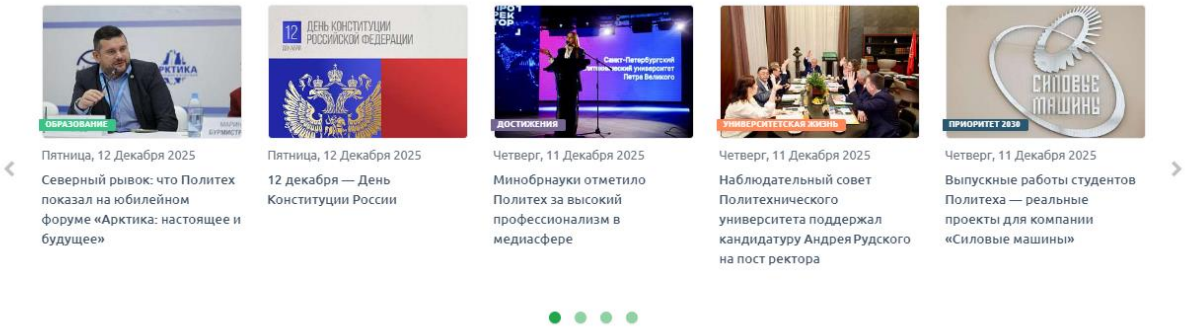


Рисунок 45 – Карусель

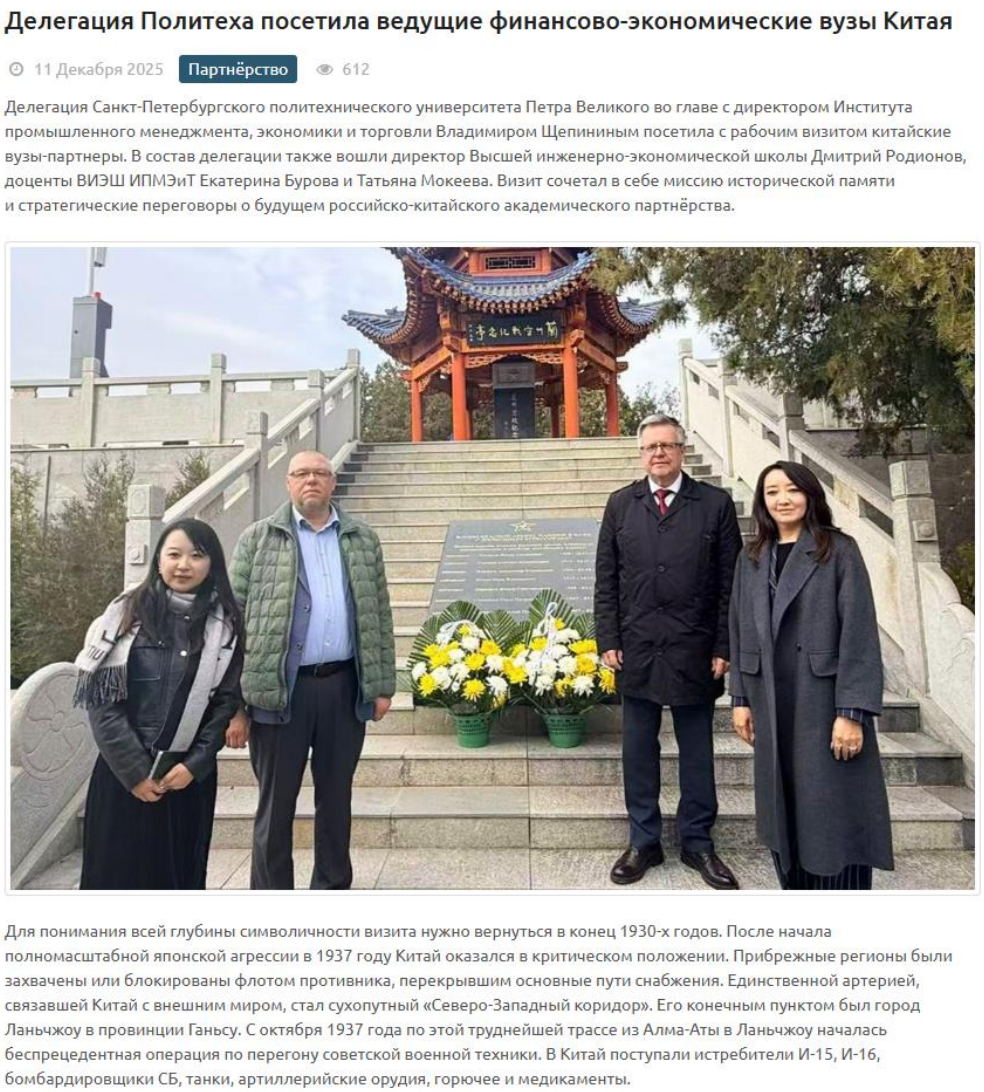
5. 2. Юзабилити (Usability) и Пользовательский опыт (UX)

- **Первое впечатление:**

Вся информация на сайте предоставляется понятно и доступно. Кнопки, выполняющие похожие функции, располагаются в одном месте и выполнены в одинаковом стиле. Заголовки выполнены жирным шрифтом, что позволяет быстро ориентироваться в разделах.

- **Контент:**

Текст читабелен, написан грамотно, изображения не искажены. (Рисунок 46)



Делегация Политеха посетила ведущие финансово-экономические вузы Китая

11 Декабря 2025 Партнёрство 612

Делегация Санкт-Петербургского политехнического университета Петра Великого во главе с директором Института промышленного менеджмента, экономики и торговли Владимиром Щепининым посетила с рабочим визитом китайские вузы-партнеры. В состав делегации также вошли директор Высшей инженерно-экономической школы Дмитрий Родионов, доценты ВИЭШ ИПМЭиТ Екатерина Бурова и Татьяна Мокеева. Визит сочетал в себе миссию исторической памяти и стратегические переговоры о будущем российско-китайского академического партнёрства.

Для понимания всей глубины символичности визита нужно вернуться в конец 1930-х годов. После начала полномасштабной японской агрессии в 1937 году Китай оказался в критическом положении. Прибрежные регионы были захвачены или блокированы флотом противника, перекрывшим основные пути снабжения. Единственной артерией, связавшей Китай с внешним миром, стал сухопутный «Северо-Западный коридор». Его конечным пунктом был город Ланьчжоу в провинции Ганьсу. С октября 1937 года по этой труднейшей трассе из Алма-Аты в Ланьчжоу началась беспрецедентная операция по перегону советской военной техники. В Китай поступали истребители И-15, И-16, бомбардировщики СБ, танки, артиллерийские орудия, горючее и медикаменты.

Рисунок 46 – Контент

- **Навигация и структура:**

До основных страниц можно добраться за 2 клика – навести курсор на нужный раздел и выбрать одну из предложенных страниц. Заголовки страниц написаны крупно, выделяются на фоне остального текста. (Рисунок 47)

Рисунок 47

- **Обратная связь:**

Сайт дает обратную связь пользователю. Например, при наведении курсора на белую иконку «Глаз» - переключение на версию для слабовидящих, она подсвечивается зеленым цветом. (Рисунок 48)

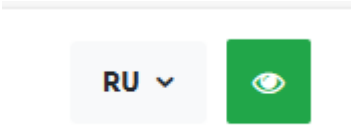


Рисунок 48

5.3. Тестирование совместимости (Compatibility Testing)

- **Кросс-браузерность:** Сайт легко открывается во всех браузерах. Работа и интерфейс сайта не меняются.
- **Кросс-платформенность: Сайт свободно работает как на** компьютерах, так и на смартфонах. Интерфейс сайта на мобильных устройствах не меняется, все функции работают корректно.

5. 4. Производительность (Performance)

Сайт загружается быстро, вся информация появляется моментально, без долгих загрузок отдельных элементов. Функции срабатывают быстро, при нажатии на кнопку пользователь сразу попадает на другую страницу. При загрузке «тяжелых» изображений сайт не виснет, загружает все четко, без искажений (Рисунок 49).

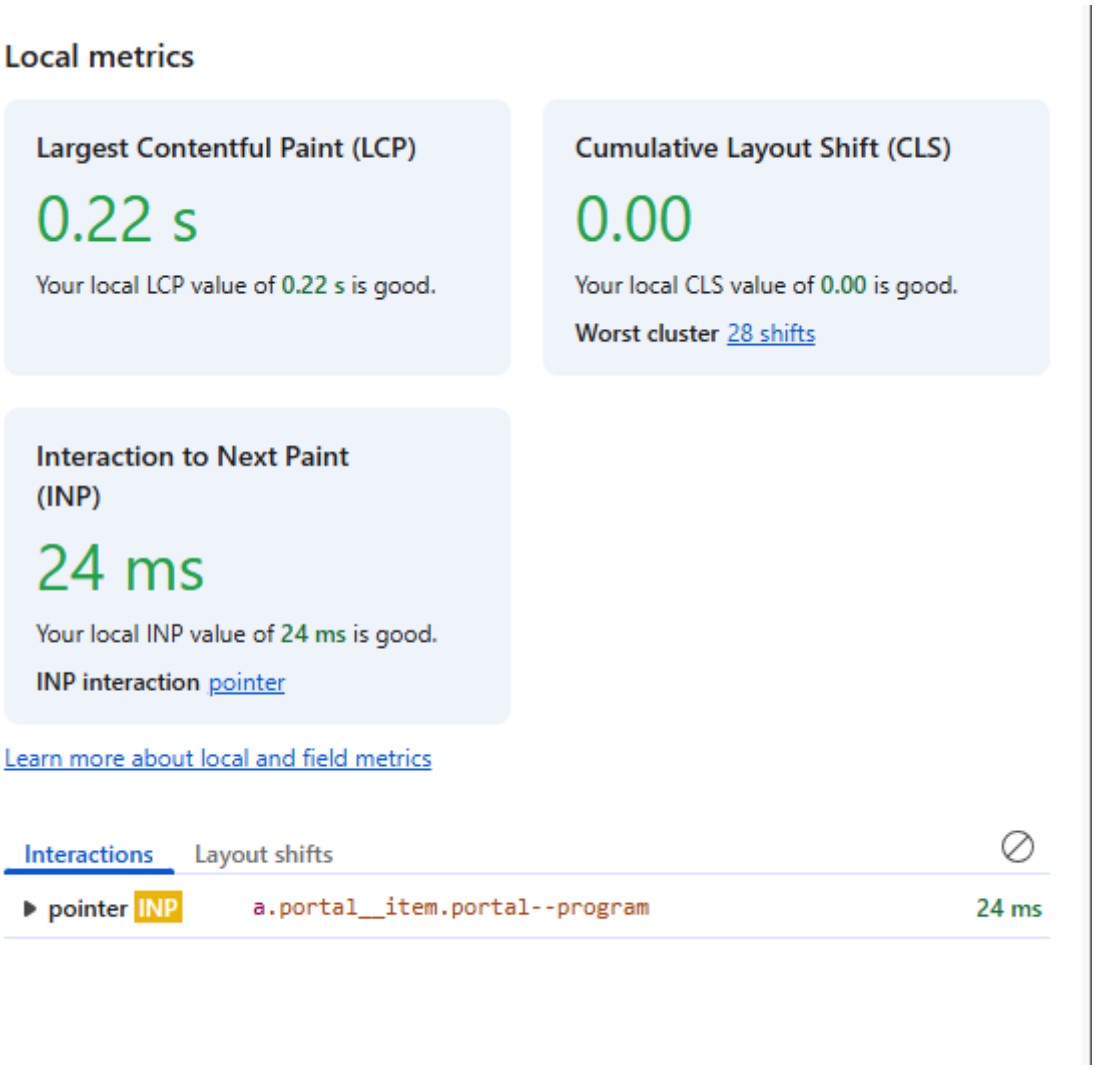


Рисунок 49 – Скорость работы

5.5. Тестирование безопасности (Security Testing) - (Базовая проверка)

Сайт использует защищенное соединение.

Рисунок 50 – HTTPS

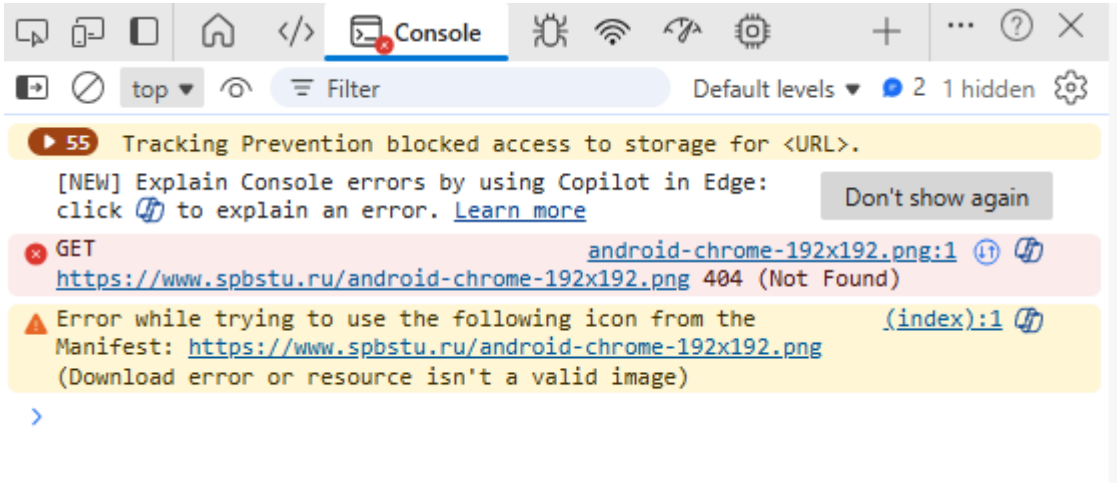


Рисунок 51 - Ошибки

К закрытым для всех пользователей данным получить доступ нельзя. Например, данные в личном кабинете доступны только его владельцу, другие пользователи никак не смогут их просмотреть.

Практическая работа №6 «Создание инсталляторов»

Цель работы: Создать консольное приложение с классом и сформировать итоговый установочный пакет.

6.1. Создание консольного приложения (ConsoleApp) с классом.

Код программы

```
#include <iostream>
#include <vector>
#include <random>
#include <algorithm>

class RandomNumberGenerator {
private:
    std::vector<int> generatedNumbers;
    std::mt19937 generator;

public:
    RandomNumberGenerator() {
        std::random_device rd;
        generator.seed(rd());
    }

    std::vector<int> generateRandom(int min, int max) {
        std::uniform_int_distribution<int> distribution(min, max);
        std::vector<int> result;

        for (int i = min; i <= max; ++i) {
            int value = distribution(generator);
            result.push_back(value);
            generatedNumbers.push_back(value);
        }

        return result;
    }

    std::vector<int> generateEven(int min, int max) {
        if (min % 2 != 0) {
            min++;
        }
        if (min > max) {
            return {};
        }

        std::vector<int> result;
        int count = (max - min) / 2 + 1;

        for (int i = 0; i < count; ++i) {
            int value = min + i * 2;
            result.push_back(value);
            generatedNumbers.push_back(value);
        }

        return result;
    }

    std::vector<int> generateOdd(int min, int max) {
        if (min % 2 == 0) {
            min++;
```

```
        }
        if (min > max) {
            return {};
        }

        std::vector<int> result;
        int count = (max - min) / 2 + 1;

        for (int i = 0; i < count; ++i) {
            int value = min + i * 2;
            result.push_back(value);
            generatedNumbers.push_back(value);
        }

        return result;
    }


    std::vector<int> getGeneratedArray() const {
        return generatedNumbers;
    }
};

int main() {
    setlocale(0, "");
    RandomNumberGenerator generator;


    std::vector<int> randomNumbers = generator.generateRandom(1, 10);
    std::cout << "Случайные числа: ";
    for (int num : randomNumbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;


    std::vector<int> evenNumbers = generator.generateEven(2, 10);
    std::cout << "Четные числа: ";
    for (int num : evenNumbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    std::vector<int> oddNumbers = generator.generateOdd(1, 9);
    std::cout << "Нечетные числа: ";
    for (int num : oddNumbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    std::vector<int> allGenerated = generator.getGeneratedArray();
    std::cout << "Все сгенерированные числа: ";
    for (int num : allGenerated) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;

}
```

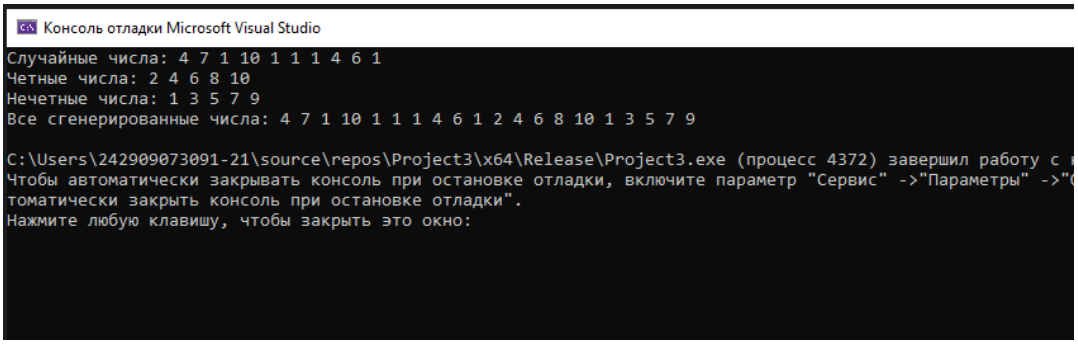


Рисунок 52 - Запуск

6.2. Создание инсталлятора

Zip.файл(рисунок 53).

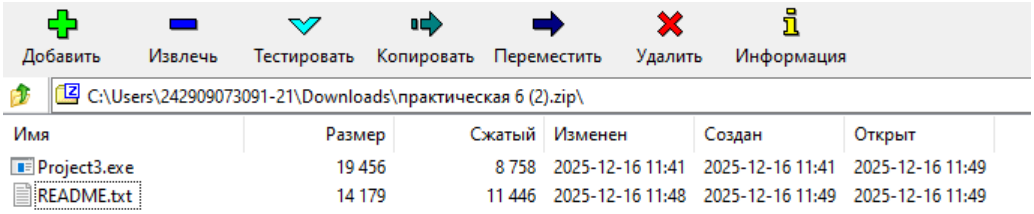


Рисунок 53 - Zip.файл

Файл README.txt (рисунок 54).

```
Моя Программа: RandomNumberGenerator
Описание:
Это консольное приложение, разработанное на C++. Программа генерирует случайные числа, затем по отдельности
четные и нечетные. В результате из всех чисел выводится массив.

Системные требования:
- Операционная система: Windows 7, 8, 10 или 11
- Дополнительное ПО не требуется (все библиотеки включены в программу)

Инструкция по запуску:
1. Распакуйте все файлы из этого архива в любую удобную папку.
2. Запустите файл "MyConsoleApp.exe" двойным кликом мыши.
3. Следуйте инструкциям в открывшемся окне консоли.

Особенности:
- Программа не требует установки.
- Не вносит изменений в системный реестр.
- Для выхода из программы используйте пункт меню или закройте окно.

Исходный код:
Исходный код проекта написан на C++ в среде Visual Studio 2022.

Разработчик: Наталья
Группа: 24290907/3091
Дата: 16.12.2025
Примечание:
Это учебный проект, созданный в рамках практической работы по
программированию на C++.
```

Рисунок 54 - Файл README.txt

Практическая работа №7 «Тестирование»

Цель: Научиться работать со структурами, понять разницу между классом и структурой, научиться тестировать программу с помощью unit-тестов.

Задания:

1. Создать свою структуру по варианту.

2. Реализовать функции (методы) структуры по варианту.

3. Изучить и расписать отличия каждого вида тестирования

4. Написать 2 позитивных и 2 негативных теста

5. Написать 1 тест для функции напарника (обменятся проектами, разобраться в коде напарника, протестировать)

Вариант 4. Структура для операций со степенями и корнями

Написать структуру для операций со степенями и корнями, реализовать методы:

- static double square() - возведение в квадрат
- static double cube() - возведение в куб
- static double power() - возведение в степень
- static double nthRoot() - извлечение корня n-ной степени

7.1. Создание структуры

```
#include <iostream>
#include <cmath>
#include <limits> // Для очистки буфера ввода
struct PowerRootOperations {
    static double square(double a) {
        return a * a;
    }
    static double cube(double a) {
        return a * a * a;
    }

    static double power(double a, double n) {
        return std::pow(a, n);
    }

    static double nthRoot(double a, int n) {
        if (n <= 0) {
            throw std::invalid_argument("Показатель корня должен быть
положительным числом.");
        }

        if (a < 0 && n % 2 == 0) {
```

```
                throw std::invalid_argument("Извлечение четного корня из
отрицательного числа невозможно.");
            }
            return std::pow(a, 1.0 / n);
        }

    };

    void clearInputBuffer() {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    int main() {
        setlocale(0, "");

        int choice;
        double number, exponent;
        int rootDegree;
        do {
            std::cout << "Меню операций: " << std::endl;
            std::cout << "1. Возвести число в квадрат" << std::endl;
            std::cout << "2. Возвести число в куб" << std::endl;
            std::cout << "3. Возвести число в произвольную степень" << std::endl;
            std::cout << "4. Извлечь корень произвольной степени из числа" <<
std::endl;
            std::cout << "5. Выход" << std::endl;
            std::cout << "Выберите пункт меню: ";

            if (!(std::cin >> choice)) {
                std::cout << "Ошибка ввода! Введите число." << std::endl;
                clearInputBuffer();
                continue;
            }

            switch (choice) {
            case 1: // Квадрат
                std::cout << "Введите число: ";
                if (!(std::cin >> number)) {
                    std::cout << "Ошибка ввода числа!" << std::endl;
                    clearInputBuffer();
                    break;
                }
                std::cout << "Квадрат введенного числа" << number << " = " <<
PowerRootOperations::square(number) << std::endl;
                break;
            case 2: // Куб
                std::cout << "Введите число: ";
                if (!(std::cin >> number)) {
                    std::cout << "Ошибка ввода числа!" << std::endl;
                    clearInputBuffer();
                    break;
                }
                std::cout << "Куб введенного числа" << number << " = " <<
PowerRootOperations::cube(number) << std::endl;
                break;
            case 3: // Произвольная степень
                std::cout << "Введите число: ";
                if (!(std::cin >> number)) {
                    std::cout << "Ошибка ввода числа!" << std::endl;
                    clearInputBuffer();
                    break;
                }
                std::cout << "Введите степень: ";
                if (!(std::cin >> exponent)) {
                    std::cout << "Ошибка ввода степени!" << std::endl;
```

```
                    clearInputBuffer();
                    break;
                }
                std::cout << number << " в степени " << exponent << " = " <<
PowerRootOperations::power(number, exponent) << std::endl;
                break;
            case 4: // Корень произвольной степени
                std::cout << "Введите число: ";
                if (!(std::cin >> number)) {
                    std::cout << "Ошибка ввода числа!" << std::endl;
                    clearInputBuffer();
                    break;
                }
                std::cout << "Введите степень корня (целое положительное число):
";
                if (!(std::cin >> rootDegree)) {
                    std::cout << "Ошибка ввода степени корня!" << std::endl;
                    clearInputBuffer();
                    break;
                }

                try {
                    std::cout << "Корень степени " << rootDegree << " из " <<
number << " = " << PowerRootOperations::nthRoot(number, rootDegree) << std::endl;
                }
                catch (const std::invalid_argument& e) {
                    std::cout << "Ошибка: " << e.what() << std::endl;
                }
                break;


            case 5: // Выход
                std::cout << "Выход из программы." << std::endl;
                break;

            default:
                std::cout << "Неверный пункт меню! Попробуйте снова." <<
std::endl;
                break;
            }
        } while (choice != 0);
        return 0;
    }
```

7.2. Реализация функций (методов) структуры

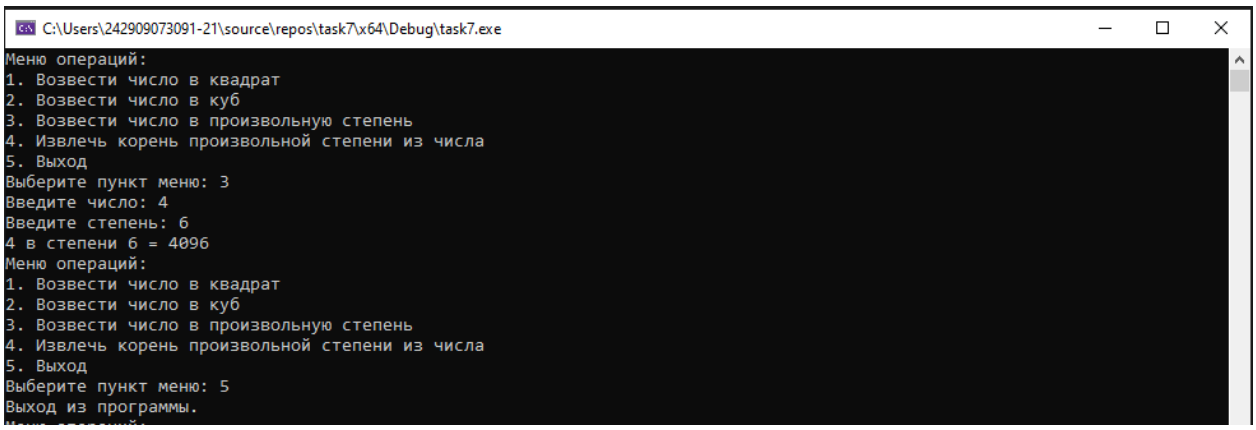


Рисунок 55 – Запуск

7.3. Отличия каждого вида тестирования

По уровню:

- Unit-тесты (модульные) – Тестируют отдельные функции/методы в изоляции, быстрые, запускаются при каждом коммите.
- Интеграционные – Тестируют взаимодействия компонентов, средняя скорость, запускаются перед мерджем.
- Системные – Тестируют всю систему целиком (как пользователь), медленные, запускаются перед релизом.

По подходу:

- Black-box – Без знания кода (тестируется ввод – вывод), фокус на требованиях и спецификациях.
- White-box – С полным знанием кода (тест строится на логике программы), фокус на условиях кода.
- Gray-box – Комбинация (тестировщик знает архитектуру, но не детали кода), учитывает и требования, и структуру системы.

7.4. Тестирование функции

Для создания проекта модельного тестирование кликаем правой кнопкой мыши по решению и Добавить -> Создать проект -> Проект машинного модульного теста (Рисунок 56).

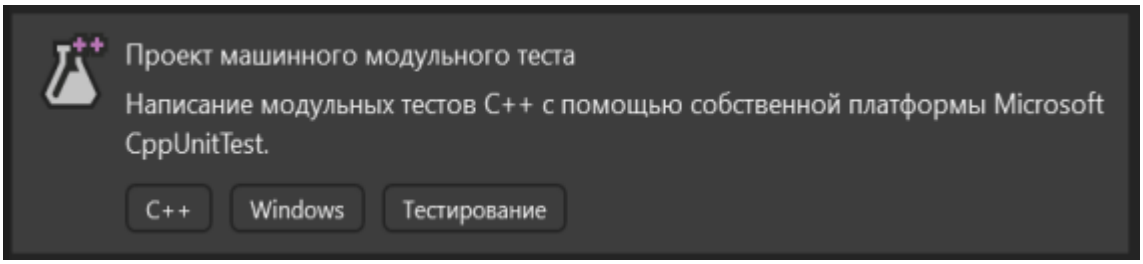


Рисунок 56 – Проект тестирования

Код теста

```
#include "pch.h"
#include "CppUnitTest.h"
#include "..\task7\task7.cpp"
using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1
{
```

```
        TEST_CLASS(PowerRootOperationsTests)
        {
        public:

            // Позитивный тест: square
            TEST_METHOD(TestSquare_Positive)
            {
                double value = 3.0;
                double expected = 9.0;
                double result = PowerRootOperations::square(value);
                Assert::AreEqual(expected, result, 0.0001);
            }

            // Позитивный тест: nthRoot (кубический корень)
            TEST_METHOD(TestNthRoot_Positive)
            {
                double value = 27.0;
                double n = 3.0;
                double expected = 3.0;
                double result = PowerRootOperations::nthRoot(value, n);
                Assert::AreEqual(expected, result, 0.0001);
            }

            // Негативный тест: корень из отрицательного значения (чётная
степень)
            TEST_METHOD(TestNthRoot_NegativeNumber_ThrowsException)
            {
                // Arrange: Пытаемся извлечь квадратный корень (2) из -16
                double number = -16.0;
                double n = 2.0;
                Assert::ExpectException<std::invalid_argument>(
                    [this, number, n]() {
                        // Это действие должно вызвать ошибку
                        PowerRootOperations::nthRoot(number, n);
                    }
                );
            }

            // Негативный тест: корень нулевой степени
            TEST_METHOD(TestNthRoot_ZeroDegree_ThrowsException)
            {
                // Arrange: Пытаемся извлечь корень 0-й степени (математически
невозможно по формуле 1/n)
                double number = 10.0;
                double n = 0.0;

                // Assert: Ожидаем исключение
                Assert::ExpectException<std::invalid_argument>(
                    [this, number, n]() {
                        PowerRootOperations::nthRoot(number, n);
                    }
                );
            }
        };
    }
```

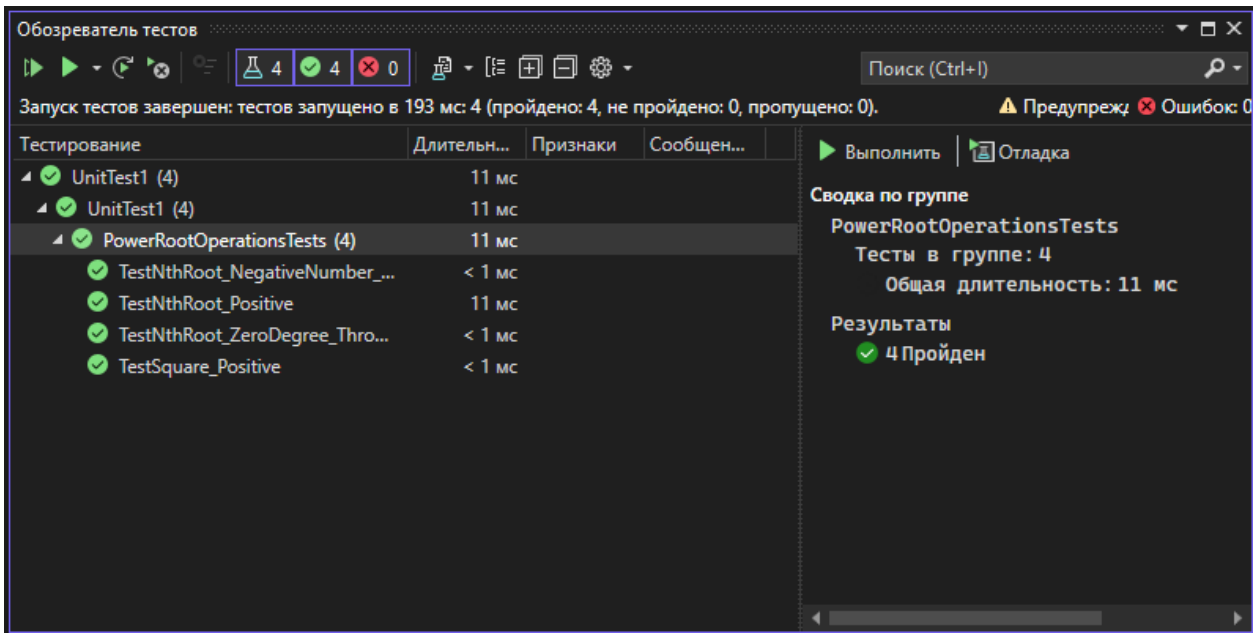


Рисунок 57 - Тестирование

7.5. Тест для функции напарника

Код программы

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

struct PerimeterOperations {
    static double squarePerimeter(double a) {
        if (a <= 0) throw std::invalid_argument("Сторона квадрата должна быть
> 0");
        return 4.0 * a;
    }

    static double rectanglePerimeter(double a, double b) {
        if (a <= 0 || b <= 0) throw std::invalid_argument("Стороны
прямоугольника должны быть > 0");
        return 2.0 * (a + b);
    }

    static double trianglePerimeter(double a, double b, double c) {
        if (a <= 0 || c <= 0) throw std::invalid_argument("Стороны
треугольника должны быть > 0");
        if (a + b <= c || b + c <= a)
            throw std::invalid_argument("Треугольник с такими сторонами не
существует");
        return a + b + c;
    }

    static double circleCircumference(double r) {
        if (r <= 0) throw std::invalid_argument("Радиус должен быть > 0");
        const double PI = 3.14159265358979323846;
        return 2.0 * PI * r;
    }
};

// безопасное чтение числа double с проверкой ввода
```

```
double readDouble(const char* prompt) {
    while (true) {
        std::cout << prompt;
        double x;
        if (std::cin >> x) return x;

        std::cout << "Ошибка ввода. Введите число.\n";
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
}

// безопасное чтение пункта меню (int)
int readInt(const char* prompt) {
    while (true) {
        std::cout << prompt;
        int x;
        if (std::cin >> x) return x;

        std::cout << "Ошибка ввода. Введите целое число.\n";
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
}

int main() {
    setlocale(0, "");

    while (true) {
        std::cout << "\n=== Вычисление периметров ===\n";
        std::cout << "1) Периметр квадрата\n";
        std::cout << "2) Периметр прямоугольника\n";
        std::cout << "3) Периметр треугольника\n";
        std::cout << "4) Длина окружности\n";
        std::cout << "0) Выход\n";

        int choice = readInt("Выберите пункт: ");

        try {
            if (choice == 0) {
                std::cout << "Выход.\n";
                break;
            }
            else if (choice == 1) {
                double a = readDouble("Введите сторону квадрата a: ");
                double p = PerimeterOperations::squarePerimeter(a);
                std::cout << "Периметр квадрата: " << p << "\n";
            }
            else if (choice == 2) {
                double a = readDouble("Введите сторону прямоугольника a: ");
                double b = readDouble("Введите сторону прямоугольника b: ");
                double p = PerimeterOperations::rectanglePerimeter(a, b);
                std::cout << "Периметр прямоугольника: " << p << "\n";
            }
            else if (choice == 3) {
                double a = readDouble("Введите сторону треугольника a: ");
                double b = readDouble("Введите сторону треугольника b: ");
                double c = readDouble("Введите сторону треугольника c: ");
                double p = PerimeterOperations::trianglePerimeter(a, b, c);
                std::cout << "Периметр треугольника: " << p << "\n";
            }
            else if (choice == 4) {
                double r = readDouble("Введите радиус окружности r: ");
                double c = PerimeterOperations::circleCircumference(r);
                std::cout << "Длина окружности: " << c << "\n";
```

```
            }
            else {
                std::cout << "Нет такого пункта меню. Повторите ввод.\n";
            }
        }
        catch (const std::invalid_argument& e) {
            std::cout << "Ошибка: " << e.what() << "\n";
        }
    }

    return 0;
}
```

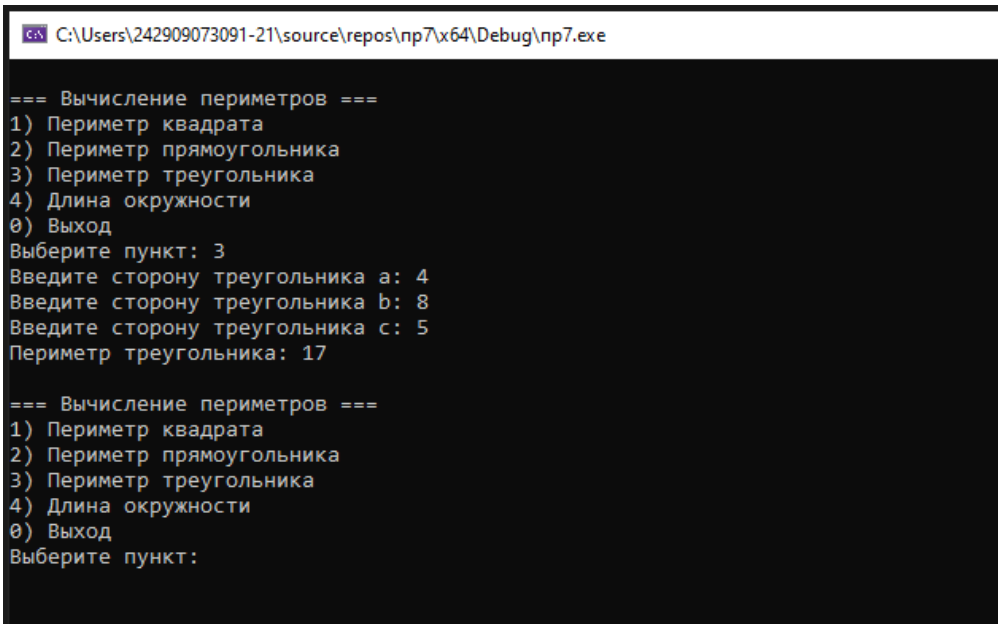


Рисунок 58 – Запуск функции напарника

Код теста

```
#include "pch.h"
#include "CppUnitTest.h"
#include "..\пр7\пр7.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1
{
    TEST_CLASS(PerimeterOperationsTests)
    {
    public:
        // Тест 1: квадрат
        TEST_METHOD(TestSquarePerimeter_ValidInput_ReturnsCorrectValue)
        {
            double a = 5.0;
            double result = PerimeterOperations::squarePerimeter(a);
            Assert::AreEqual(20.0, result, 0.0001);
        }
    };
}
```

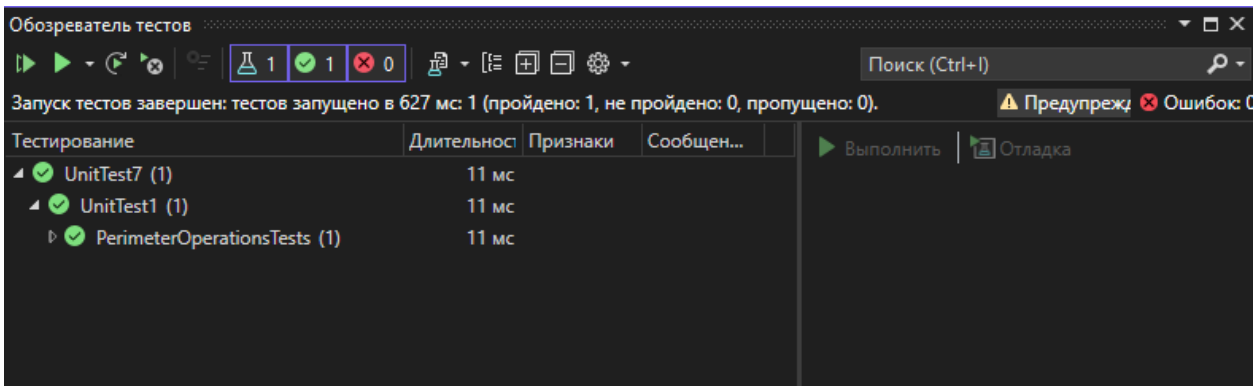


Рисунок 59 - Тестирование функции напарника

Практическая работа 8: «Разработка ТЗ и макетов для сайта»

Цель работы: Научиться создавать четкое техническое задание и разрабатывать к нему визуальные макеты (как черно-белые, так и цветные) для статического сайта.

Вариант 4. Сайт мастера по маникюру

Описание: "Сайт-визитка для привлечения клиентов. Покажите мои работы, прайс-лист и чтобы была запись на услугу."

Страницы: Главная (популярные работы), Мои работы (галерея), Услуги и цены, Обо мне, Запись (форма выбора даты/услуги).

JS-элементы: Слайдер примеров работ, простой калькулятор стоимости (при выборе услуги подставляется цена).

8. 1. Разработка Технического Задания (ТЗ)

1. Цели и описание проекта

Ключевая цель: Создать современный, визуально привлекательный и удобный статический сайт-визитку для мастера маникюра. Он должен привлекать новых клиентов, демонстрировать портфолио мастера, информировать об услугах и ценах, упрощать процесс записи клиентов.

Целевая аудитория: Женщины 16+, интересующиеся уходом за руками и дизайном ногтей.

Общее описание: Сайт представляет собой статический многостраничный лендинг (без базы данных и сложной серверной логики). Он служит цифровой визитной карточкой мастера, акцентируя внимание на визуальной составляющей (портфолио) и предоставляя клиентам всю необходимую информацию для принятия решения и записи.

2. Функциональные требования

Главная страница: Статическая страница с приветственным блоком: Заголовок, короткий текст-подзаголовок о мастере и подходе к работе, блок с 5–7 лучшими работами в виде автоматического слайдера, кнопка «Записаться

на маникюр» (ведёт на страницу Запись), кнопка «Смотреть все работы» (ведёт на страницу Мои работы).

Страница «Мои работы»: Страница с галереей фотографий выполненных работ (минимум 30–50 фото), фильтры по категориям (например: «Все», «Классический маникюр», «Дизайн», «Наращивание», «Педикюр», «Сложный дизайн»), фильтрация работает без перезагрузки страницы (JavaScript): при выборе категории показываются только соответствующие фото, адаптивная сетка (3–4 колонки на десктопе, 2 на планшете, 1 на мобильных), при клике на фото открывается полноэкранное модальное окно (lightbox) с возможностью листать фото стрелками влево/вправо.

Страница «Услуги и цены»: Страница с списком всех услуг в виде аккуратных карточек или аккордеона, при выборе услуги автоматически появляется цена, кнопка «Записаться на выбранные услуги» переносит выбранные позиции в форму записи (страница Запись) через localStorage или URL-параметры.

Страница «Обо мне»: Страница с текстом о мастере: опыт работы, образование, сертификаты, подход к стерильности и материалам, блок «Мои сертификаты» — небольшие изображения сертификатов (по клику открываются в lightbox), блок с 1 – 3 личными фотографиями.

Страница «Запись»: Страница с формой онлайн-записи с пошаговым или одностраничным интерфейсом, календарь, показывающий только рабочие дни и свободные слоты, после успешной записи — модальное окно «Спасибо! Я свяжусь с вами в ближайшее время для подтверждения».

Шапка и подвал сайта: Шапка содержит логотип (имя мастера или графический логотип) в левом углу, кликабельный (ведет на главную), навигационное меню (Главная, Мои работы, Услуги и цены, Обо мне, Запись). Подвал содержит копирайт (имя мастера, текущий год), иконки социальных сетей (Instagram, WhatsApp, VK, Telegram - по наличию у мастера), краткую контактную информацию (телефон, email).

3. Техстек и интеграция

Фронтенд: HTML5, CSS3, чистый JavaScript (ES6+) для интерактивности (меню-бургер, фильтры портфолио, модальные окна, формы).

Хостинг: Статический хостинг (Netlify, GitHub Pages).

Интеграции:

Формы обратной связи: Formspree.io или EmailJS для обработки данных из форм записи и обратной связи.

Карты: Статическая карта через API Яндекс.Карт или Google Maps (бесплатный тариф).

Соцсети: Иконки со ссылками на профили мастера.

4. Нефункциональные требования

a) Производительность:

Время полной загрузки главной страницы — не более 3 секунд (при скорости 10 Мбит/с)

Открытие модальных окон, переключение фильтров — без задержек (< 100 мс)

Оптимизация изображений (WebP, сжатие, отложенная загрузка)

b) Совместимость:

Адаптивность: Полная корректность отображения на всех разрешениях от 320 до 1920 пикселей+.

Кроссбраузерность: Корректная работа в последних версиях Chrome, Firefox, Safari, Edge.

c) Удобство использования:

Стиль: Чистый, современный, женственный минимализм. Акцент на фотографиях работ. Много свободного пространства.

Цветовая палитра: Нежная, пастельная (например, пудрово-розовый, бежевый, персиковый, цвет слоновой кости). Акцентный цвет для кнопок призыва к действию (например, приглушённый коралловый или золотистый).

Шрифты: Акцентный шрифт для заголовков, читабельный шрифт для основного текста.

d) Масштабируемость и сопровождение:

Чистый, структурированный и прокомментированный код

Организация CSS по методологии БЭМ

Изображения в нескольких размерах (обычные + Retina)

Простота обновления прайс-листа и добавления новых работ

README-файл с инструкцией по редактированию контента

5. Критерии приемки и тестирования

- Все страницы корректно отображаются на тестовых разрешениях: 320 пикселей, 768 пикселей, 1024 пикселя, 1440 пикселей.
- Навигационное меню работает на всех устройствах.
- Фильтры на странице «Мои работы» корректно скрывают/показывают работы.
- Все ссылки (меню, социальные сети) ведут на нужные страницы/ресурсы.
- Формы записи и обратной связи проверяют обязательные поля.

6. Этапы и сроки сдачи

Этап 1 (1 неделя): Согласование черно-белых макетов всех страниц.

Этап 2 (2 неделя): Согласование цветных макетов всех страниц.

Этап 3 (3 неделя): Создание полностью сверстанного, адаптивного и интерактивного сайта. Выполнение всех функциональных требований.

Этап 4 (4 неделя): Подключение всех интеграций (формы, карты). Финальное тестирование.

6. 2. Разработка Макетов

Задача: Создайте визуальные макеты для сайта, описанного в ТЗ.

Инструменты: Вы можете использовать Figma или Canva (Заблокано в РФ).

Черно-белые макеты (Wireframes) – Рисунок 60 - 64

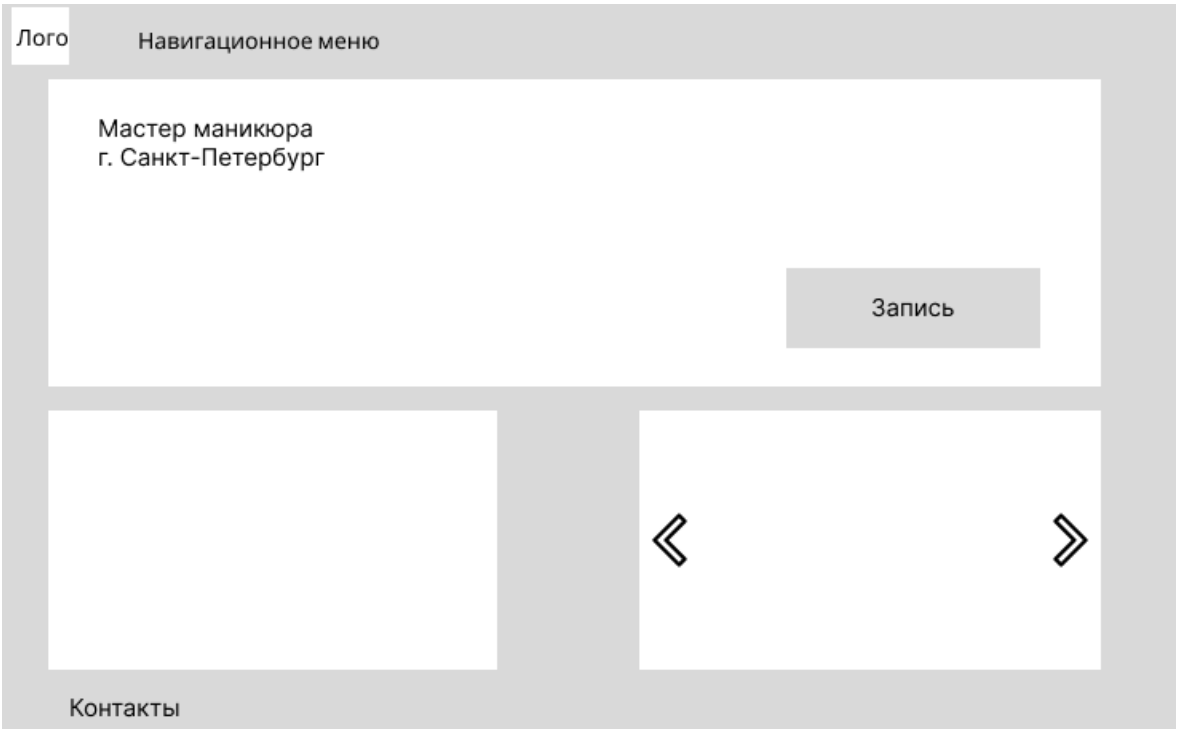


Рисунок 60 – «Главная страница»

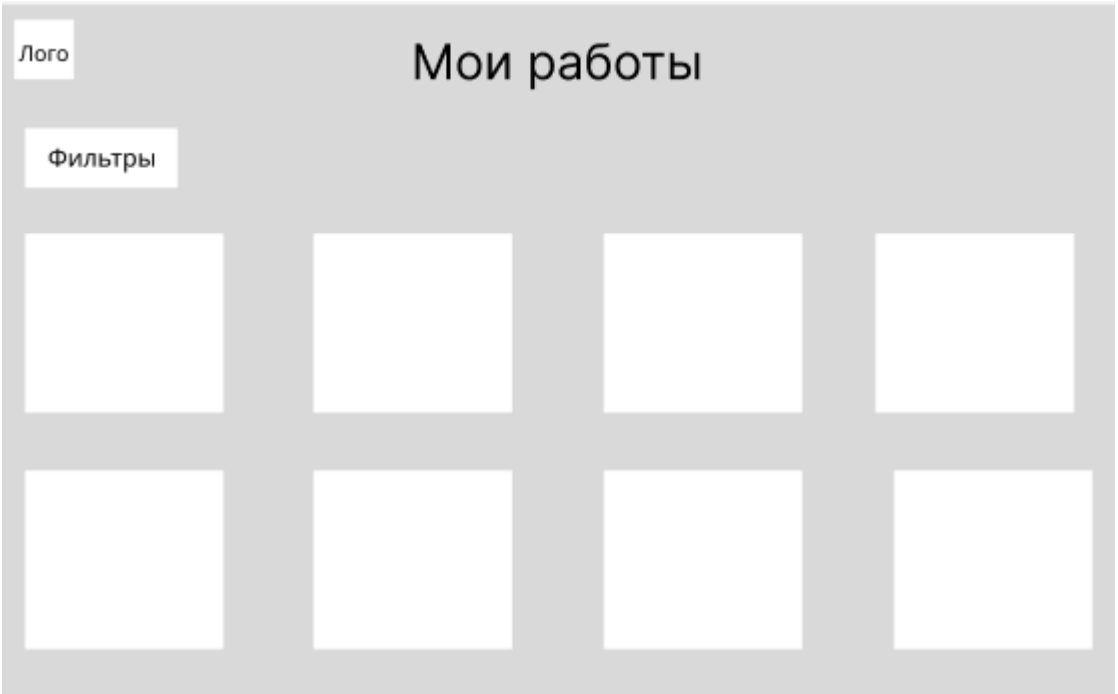


Рисунок 61 – «Мои работы»

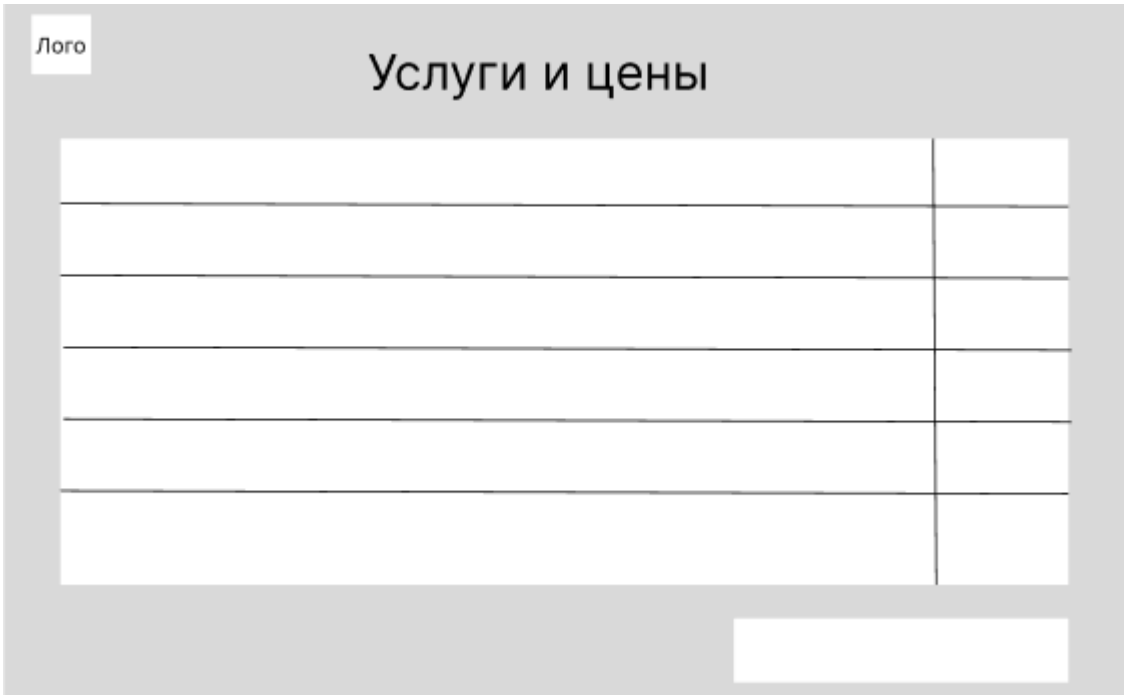


Рисунок 62 – «Услуги и цены»

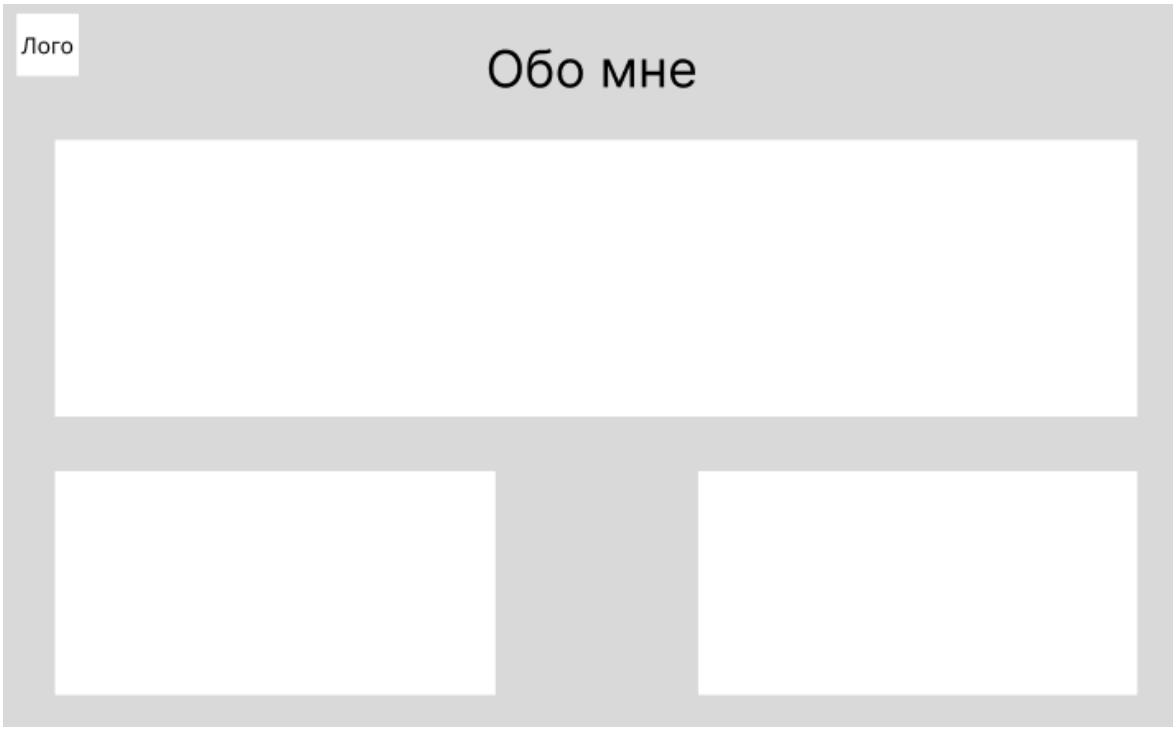


Рисунок 63 – «Обо мне»

Рисунок 64 – «Запись»

Цветные макеты (Mockups) – Рисунок 65 - 69

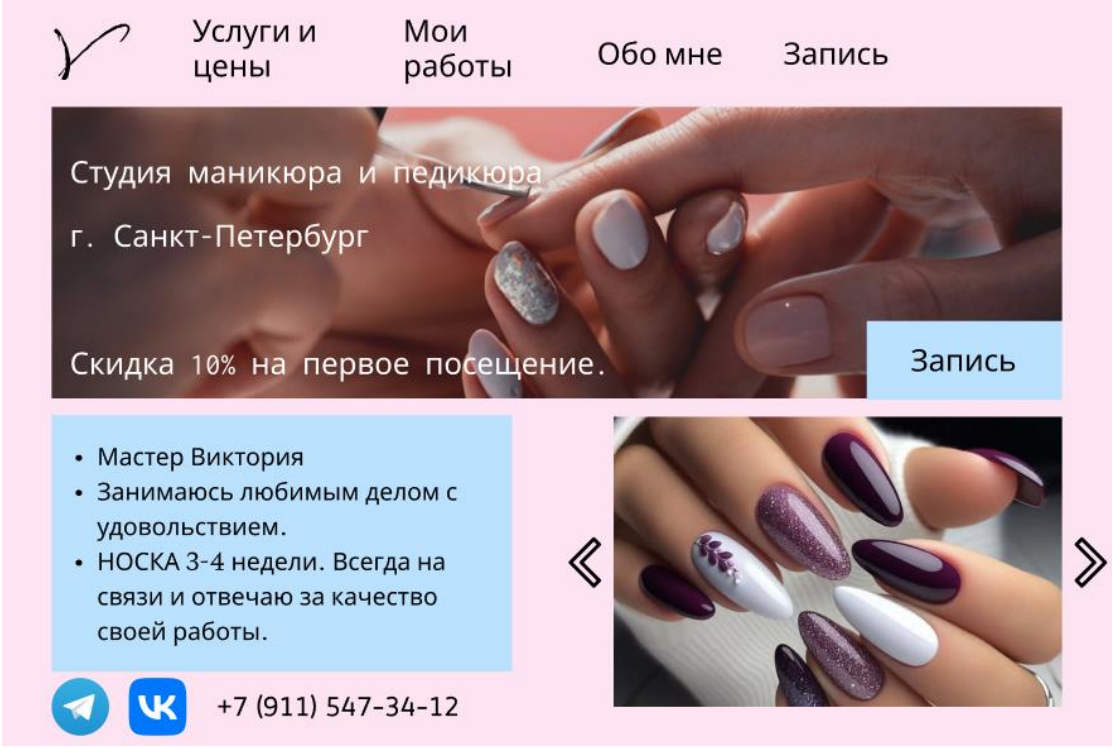


Рисунок 65 – «Главная»

Рисунок 66 – «Мои работы»

Рисунок 67 – «Услуги и цены»

Рисунок 68 – «Обо мне»

Рисунок 69 – «Запись»

Практическая работа 9. «Работа с системой контроля версий Git»

9. 1. Прохождение уровней

Пройти 4 уровня из раздела «Введение», 4 уровня из раздела «Едем дальше» и 4 уровня на вкладке «Удаленные репозитории» из раздела «Push & Pull» в игре Learn Git Branching.

Пройти 4 уровня из раздела «Введение».

1 уровень. Используемые функции: git commit, git commit (Создание коммитов).

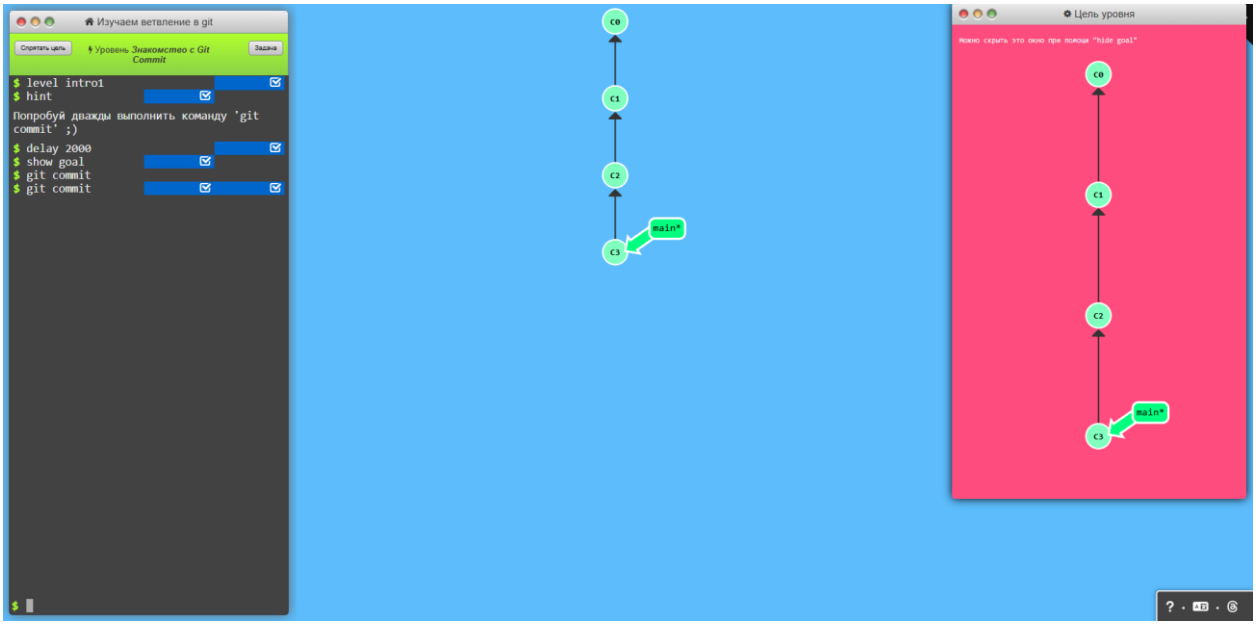


Рисунок 70 - 1 уровень «Введение»

2 уровень. Используемые команды: git branch bugFix, git checkout bugFix (Создать и переключиться на новую ветку bugFix).

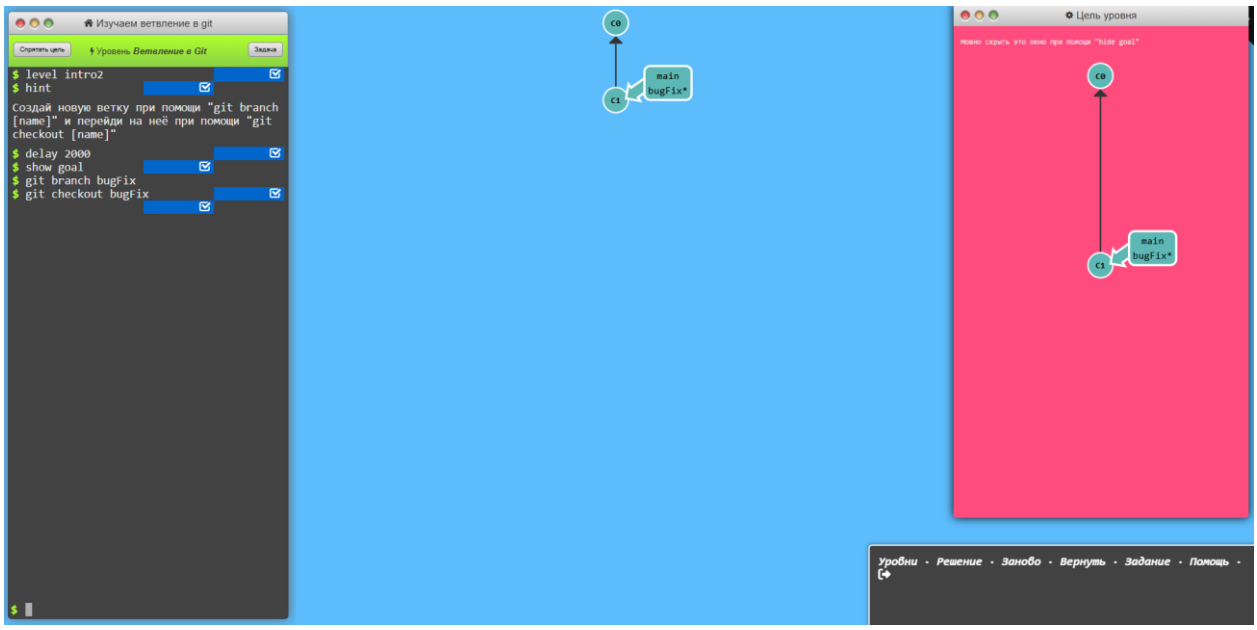


Рисунок 71 - 2 уровень «Введение»

3 уровень. Используемые команды: git branch bugFix, git checkout bugfix; git commit; git checkout main; git commit; git merge bugfix (Создать и переключиться на новую ветку bugfix. Объединение изменений из двух веток. Использование команды git merge).

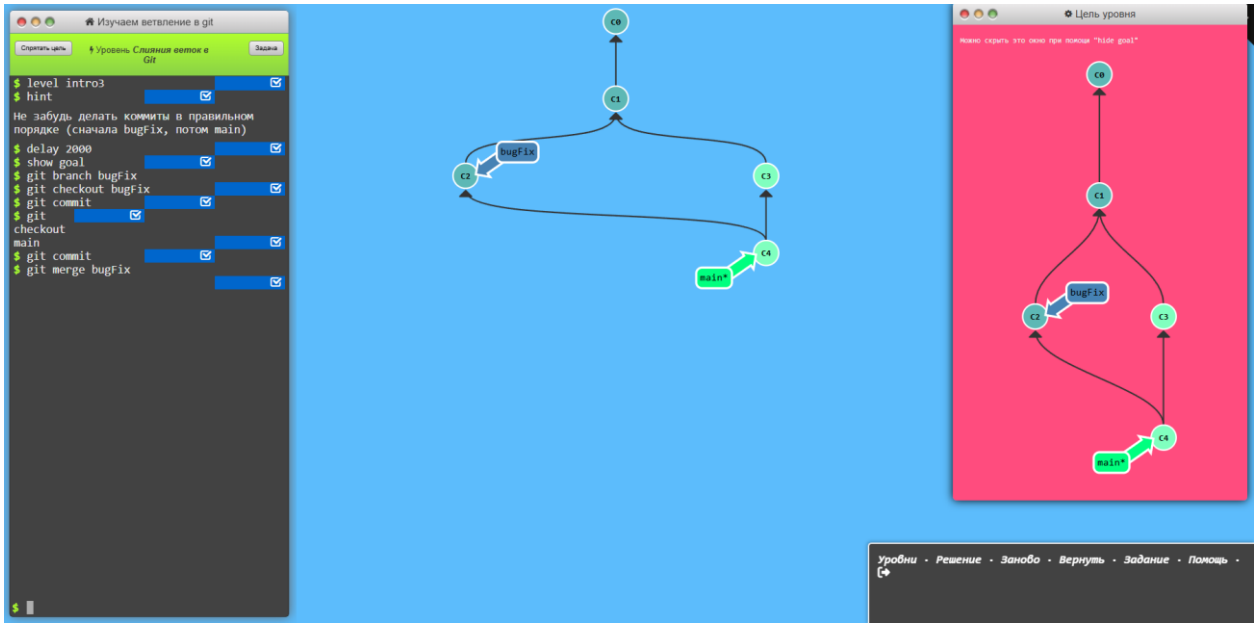


Рисунок 72 - 3 уровень «Введение»

4 уровень. Используемые команды: git branch bugFix, git checkout bugfix; git commit; git checkout main; git commit; git checkout bugfix; git rebase bugfix (Объединение изменений из двух веток).

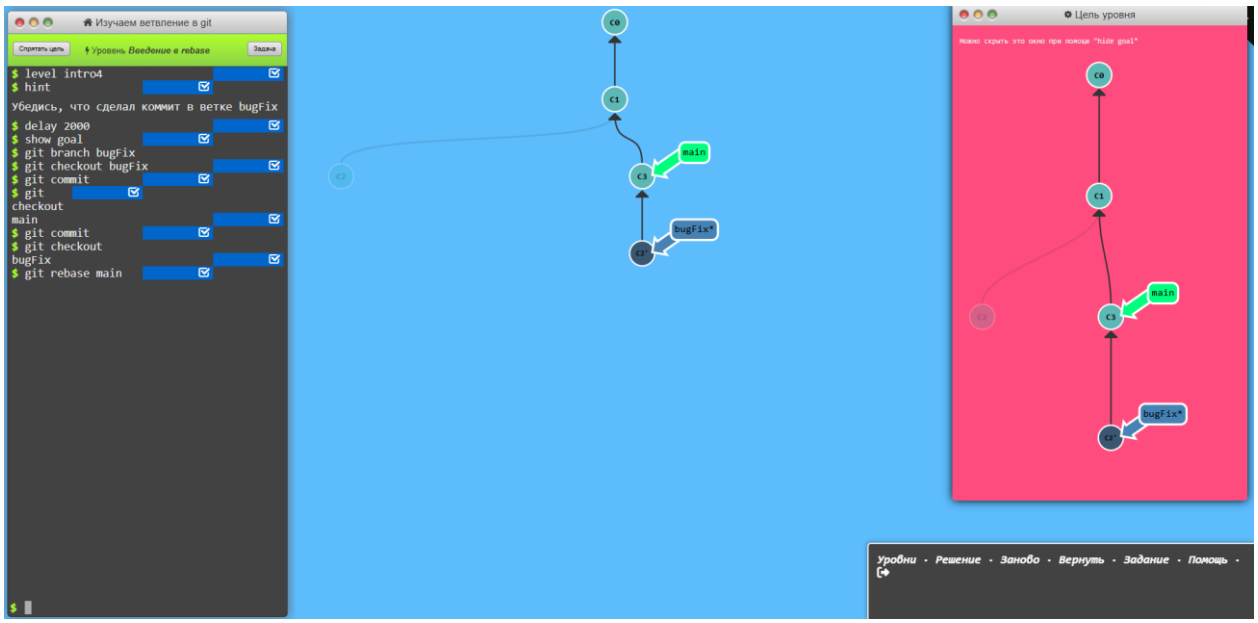


Рисунок 73 - 4 уровень «Введение»

Пройти 4 уровня из раздела «Едем дальше».

1 уровень. Используемые команды: git checkout C4 (Отделила HEAD от ветки bugFix и присвоила его последнему коммиту в этой же ветке).

Рисунок 74 - 1 уровень «Едем дальше»

2 уровень. Используемые команды: git checkout bugfix^ (Переместила на первого родителя ветки).

Рисунок 75 - 2 уровень «Едем дальше»

3 уровень. Используемые команды: git checkout C1; git brunch -f main C6; git brunch -f bugfix bugfix~3 (Переключение на ветку C1. Передвижение указателя ветки main на коммит C6. Сброс текущей ветки bugfix на коммит bugfix~3).

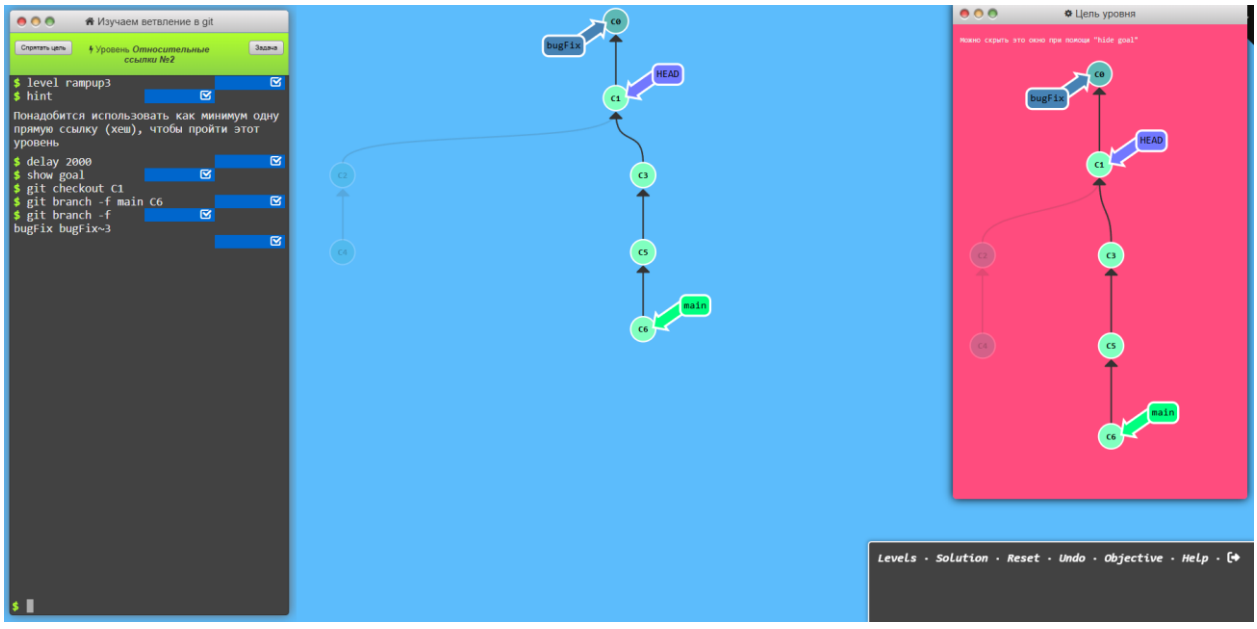


Рисунок 76 - 3 уровень «Едем дальше»

4 уровень. Используемые команды: git reset HEAD~1; git checkout pushed; git revert HEAD (Заменила файлы в рабочей копии на файлы предка переданного коммита, далее создала коммит, чтобы сохранить изменения, вернулась к прежнему состоянию файлов в рабочей области).

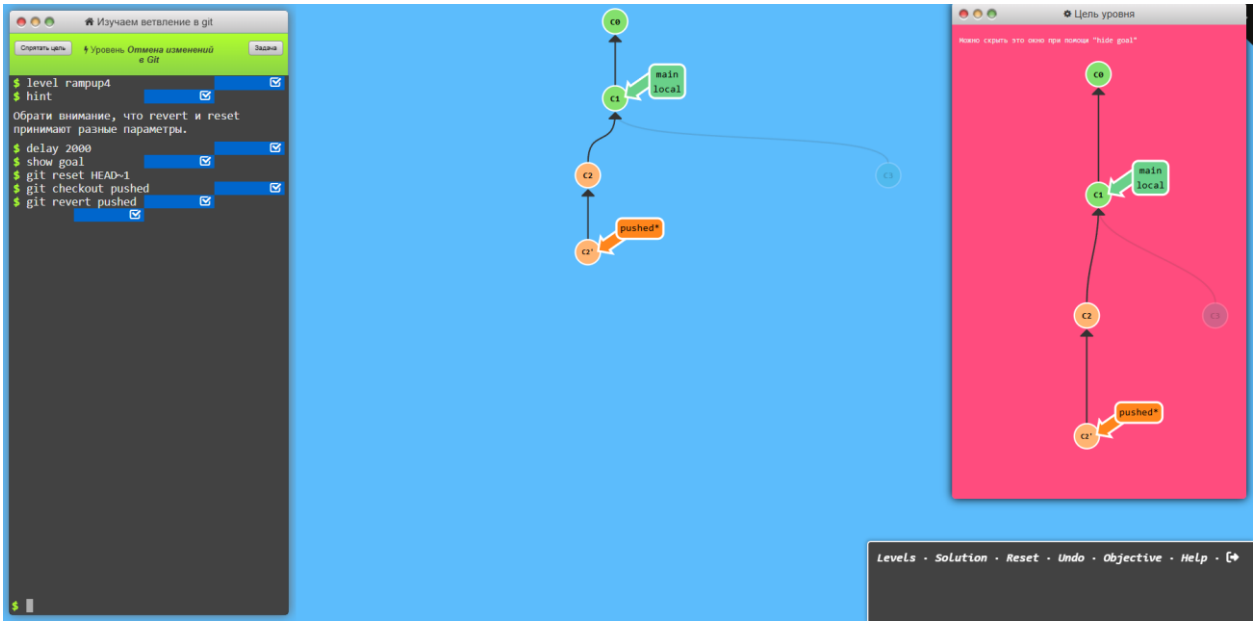


Рисунок 77 - 4 уровень «Едем дальше»

Пройти 4 уровня на вкладке «Удаленные репозитории» из раздела «Push & Pull».

1 уровень. Используемые команды: git clone (Создала клон репозитория).

Рисунок 78 - 1 уровень «Push & Pull»

2 уровень. Используемые команды: git commit; git checkout o/main; git commit (Создала коммит, перешла к удаленным веткам, создала коммит).

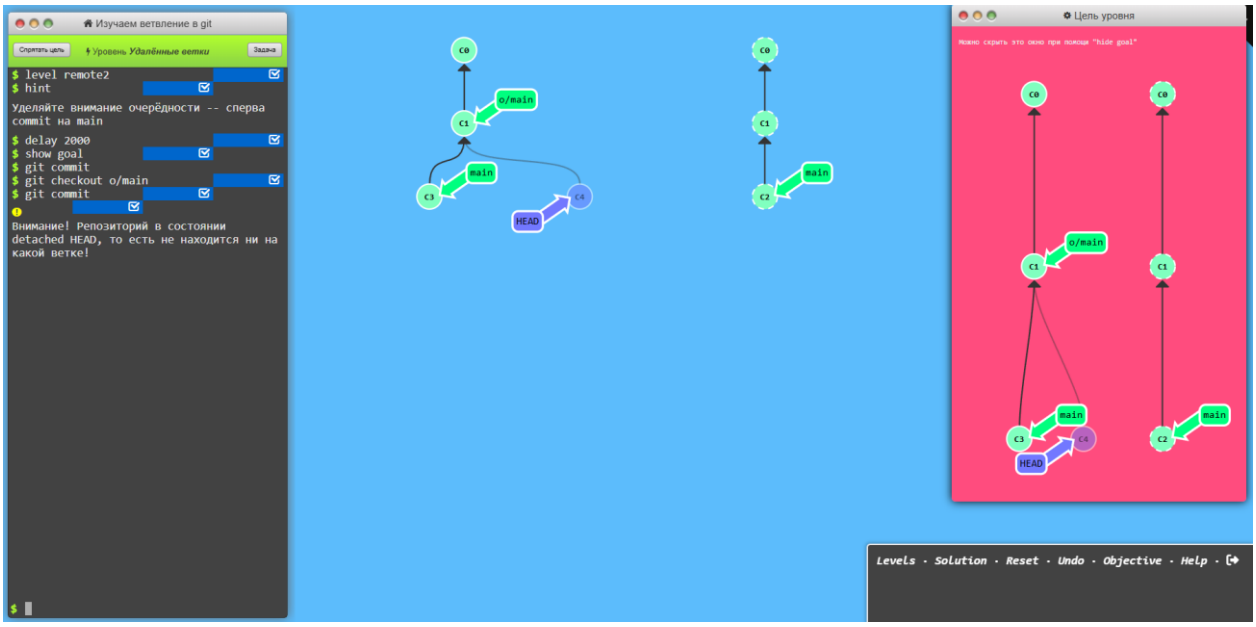


Рисунок 79 - 2 уровень «Push & Pull»

3 уровень. Используемые команды: git fetch **(**Загрузила актуальные данные из удалённого репозитория в локальное хранилище).

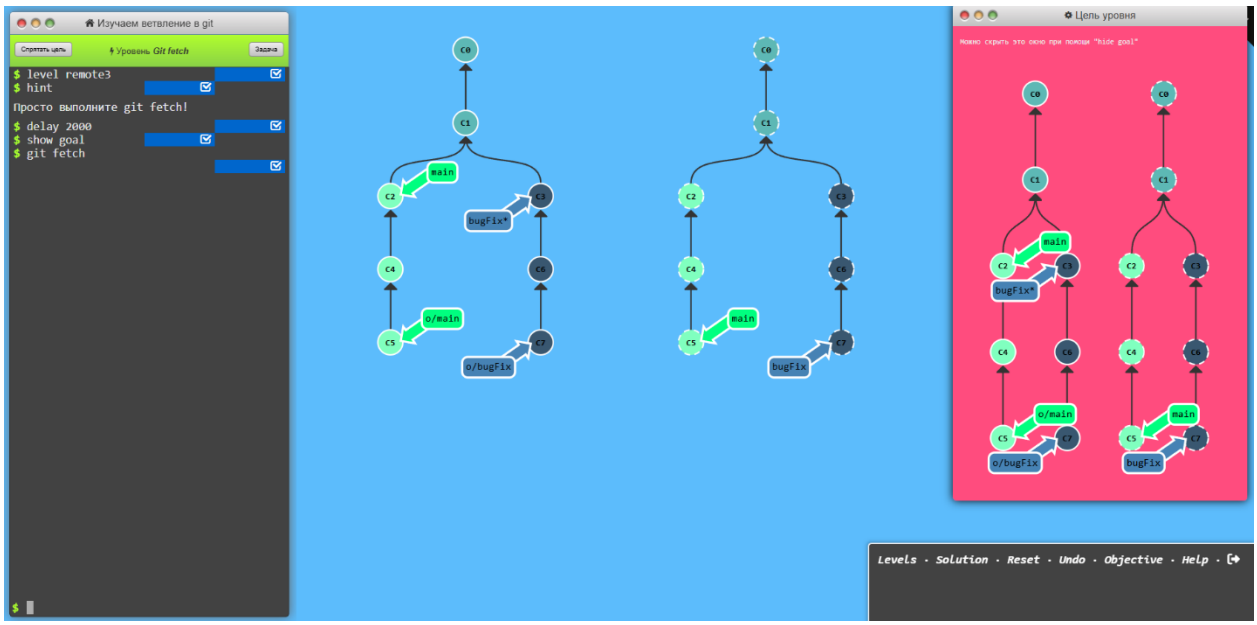


Рисунок 80 - 3 уровень «Push & Pull»

4 уровень. Используемые команды: git pull (Загрузила последние изменения из удалённого репозитория, и команда автоматически слила их с текущей локальной веткой).

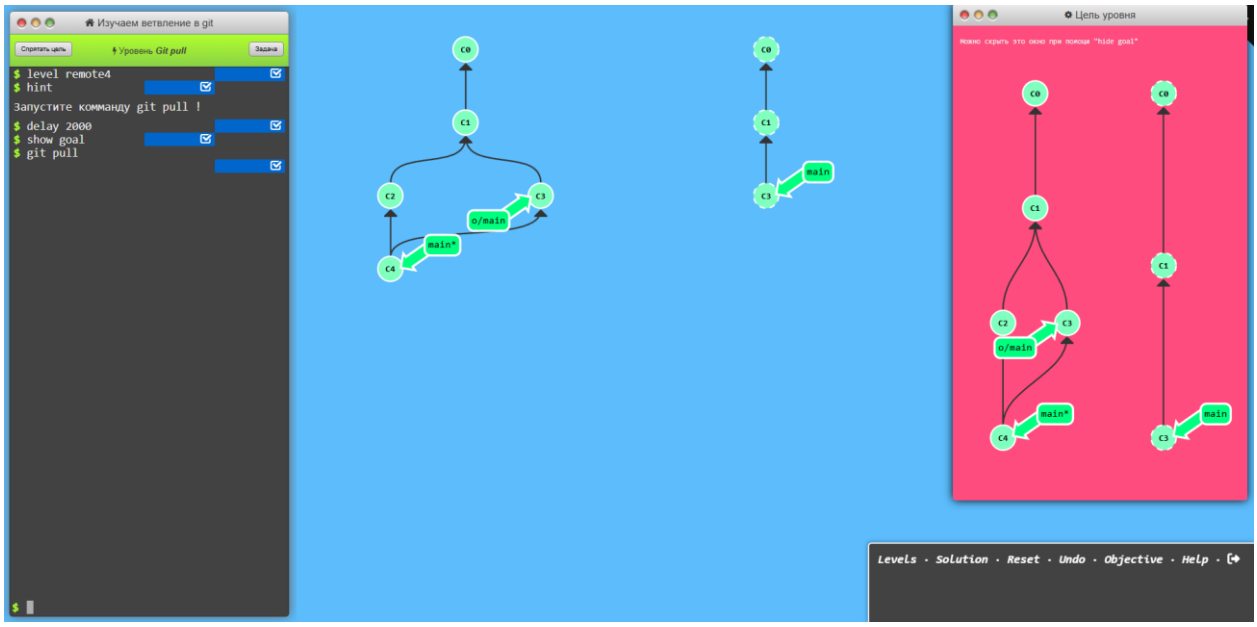


Рисунок 81 - 4 уровень «Push & Pull»

9.2. GitHub

Регистрация:

1. Нажать кнопку «Sign up» справа вверху.

2. Ввести электронную почту, придумать пароль и имя пользователя. Также можно пройти быструю регистрацию через аккаунт Google.

3. Подтвердить почту по ссылке из письма.

Создание репозитория на GitHub:

1. После входа в аккаунт GitHub нажать на «+» в правом верхнем углу.
2. Выбрать «New repository».
3. Заполнить форму: указать имя и краткое описание проекта.
4. Нажать «Create repository».

Добавление проектов:

1. В открытом репозитории выбрать «+», затем «Загрузить файл».
2. На компьютере открыть папку с проектом и перетащить все файлы и папки в браузер.
3. Внизу страницы в разделе «Зафиксировать изменения».

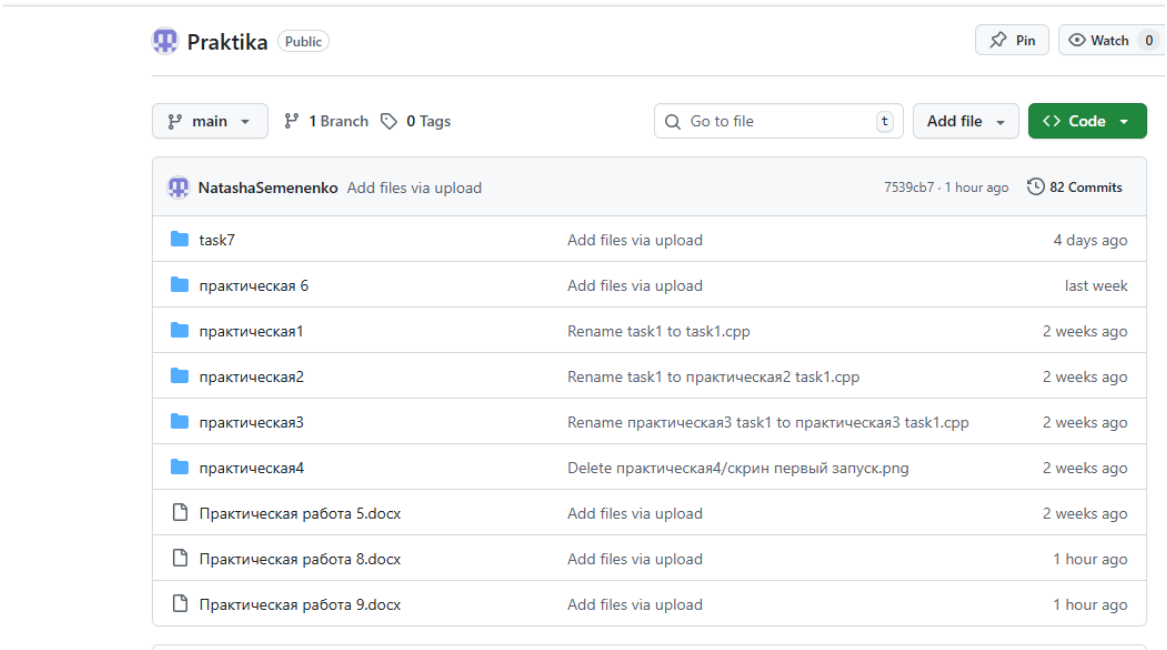


Рисунок 82 - Репозиторий