**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

Патласова Елизавета Витальевна

(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 Васильев С.Ю.

от учебной организации (подпись) (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

Патласова Елизавета Витальевна

(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. Васильев С.Ю.. (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

Патласова Елизавета Витальевна

(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 Васильев С.Ю.

прохождения практики (подпись) (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 6](#_Toc217302595)

[1. Практическая работа №1 «Настройка внешнего вида и функциональности среды разработки Visual Studio» 7](#_Toc217302596)

[1.1 Задание 1 7](#_Toc217302597)

[1.2. Задание 2 12](#_Toc217302598)

[1.3. Задание 3 17](#_Toc217302599)

[1.4. Задание 4 22](#_Toc217302600)

[1.5. Задание 5 25](#_Toc217302601)

[2. Практическая работа №2 «Отладка в Visual Studio» 28](#_Toc217302602)

[2.1 Задание 1. Навигация по коду с помощью отладчика 28](#_Toc217302603)

[2.2. Задание 2. Управление исключениями 32](#_Toc217302604)

[2.3. Задание 3. Профилирование 34](#_Toc217302605)

[3. Практическая работа №3 «Обеспечение качества кода на C++» 37](#_Toc217302606)

[3.1. Задание 1: Самодокументирующийся код 37](#_Toc217302607)

[3.2. Задание 2: Соглашение о кодировании 38](#_Toc217302608)

[3.3. Задание 3: Метрика Джилба (упрощенный вариант) 44](#_Toc217302609)

[3.3.1. Подсчёт исполняемых строк (η₁): 45](#_Toc217302610)

[3.3.2. Подсчёт логических переходов (η₂): 45](#_Toc217302611)

[3.3.3. Вывод: 45](#_Toc217302612)

[3.3.4. Подсчёт η₁ 46](#_Toc217302613)

[3.3.5. Подсчёт η₂ 46](#_Toc217302614)

[3.3.6. Вычисление метрики Джилба: 46](#_Toc217302615)

[3.3.7. Вывод: 46](#_Toc217302616)

[4. Практическая работа №4 «Работа с реестром ОС Windows» 47](#_Toc217302617)

[4.1 Задание 47](#_Toc217302618)

[5. Практическая работа №5 «Задание для тестировщика» 51](#_Toc217302619)

[5.1 Краткое резюме 51](#_Toc217302620)

[5.2 Список багов 51](#_Toc217302621)

[5.3. Функциональное тестирование (Functional Testing) 52](#_Toc217302622)

[5.4. Юзабилити (Usability) и Пользовательский опыт (UX) 56](#_Toc217302623)

[5.5. Тестирование совместимости (Compatibility Testing) 57](#_Toc217302624)

[5.6. Производительность 59](#_Toc217302625)

[5.7. Тестирование безопасности 59](#_Toc217302626)

[6. Практическая работа №6 «Создание инсталляторов» 61](#_Toc217302627)

[6.1. Создание консольного приложения (ConsoleApp) с классом 61](#_Toc217302628)

[7. Практическая работа №7 «Тестирование» 65](#_Toc217302629)

[7.1. Задание 1 65](#_Toc217302630)

[7.2. Задание 2 68](#_Toc217302631)

[8. Практическая работа 8: «Разработка ТЗ и макетов для сайта» 74](#_Toc217302632)

[8.1. Часть 1: Разработка Технического Задания (ТЗ) 74](#_Toc217302633)

[8.1.1. Цели и описание проекта 74](#_Toc217302634)

[8.1.2. Функциональные требования 74](#_Toc217302635)

[8.1.3. Технологии 75](#_Toc217302636)

[8.1.4. Нефункциональные требования 75](#_Toc217302637)

[8.1.5. Критерии тестирования 76](#_Toc217302638)

[8.1.6. Сроки поэтапной сдачи 76](#_Toc217302639)

[8.2. Часть 2: Разработка Макетов 77](#_Toc217302640)

[8.2.1. Черно-белые макеты (Wireframes) 77](#_Toc217302641)

[8.2.2. Цветные макеты (Mockups) 78](#_Toc217302642)

[ПРИЛОЖЕНИЕ А 79](#_Toc217302643)

[ПРИЛОЖЕНИЕ Б 84](#_Toc217302644)

[9. Практическая работа 9 «Работа с системой контроля версий Git» 89](#_Toc217302645)

[9.1. Задание 1. 89](#_Toc217302646)

[9.1.1. Решение уровней раздела «Введение» 89](#_Toc217302647)

[9.1.2. Раздел «Едем дальше» 91](#_Toc217302648)

[9.1.3. Раздел «Push&Pull» на вкладке «Удаленные репозитории» 93](#_Toc217302649)

ВВЕДЕНИЕ

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

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

Код программы:

#include <iostream>

using namespace std;

int main() {

setlocale(0, "");

int x, y;

cout << "Введите два целых числа через пробел: ";

cin >> x >> y;

int sum = x + y;

cout << "Сумма: " << sum << endl;

return 0;

}

1.1 Задание 1

Открыть Средства, затем Параметры, в поиск ввести Текстовый редактор, выбрать язык С++, Форматирование и дальше менять как на Рисунках 1 – 8. На Рисунках 1-2 включить автоматическое форматирование при вводе { и ;.

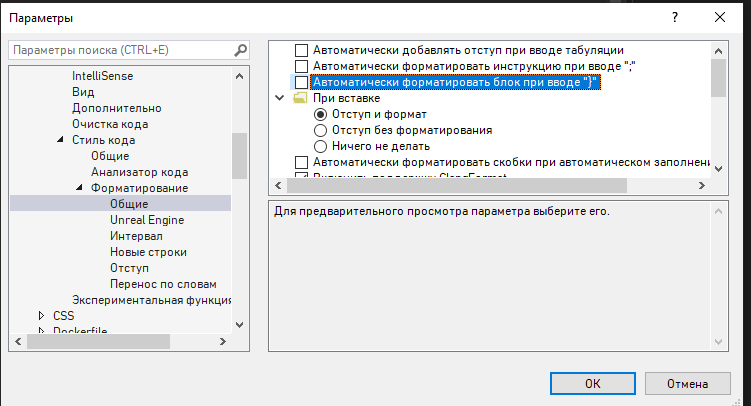


Рисунок 1 - Форматирование(скобки)

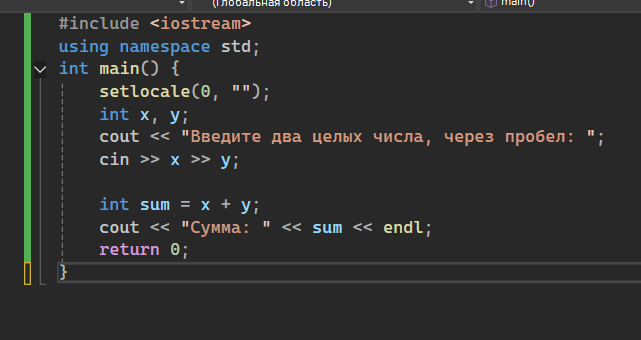


Рисунок 2 – Результат рис.1

Если отключить форматирование, то следовательно все съедет (Рисунок 3-4).

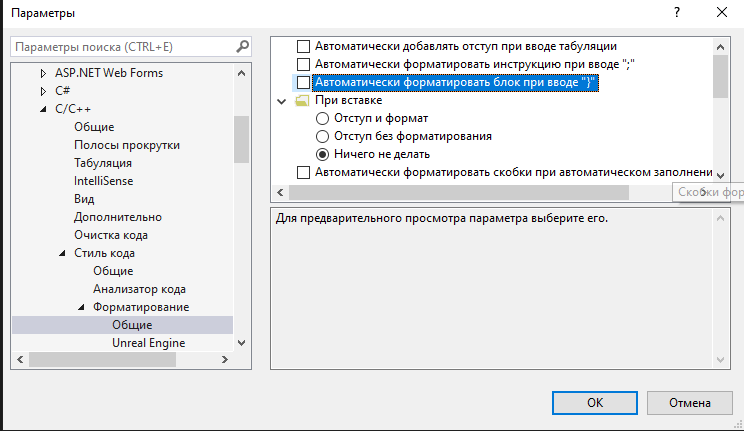


Рисунок 3 - Отключение форматирования

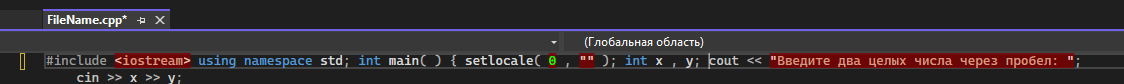


Рисунок 4 – Результат рис.3

Редактировать интервалы, добавить интервал между сводками и в скобках (Рисунок 5-6).

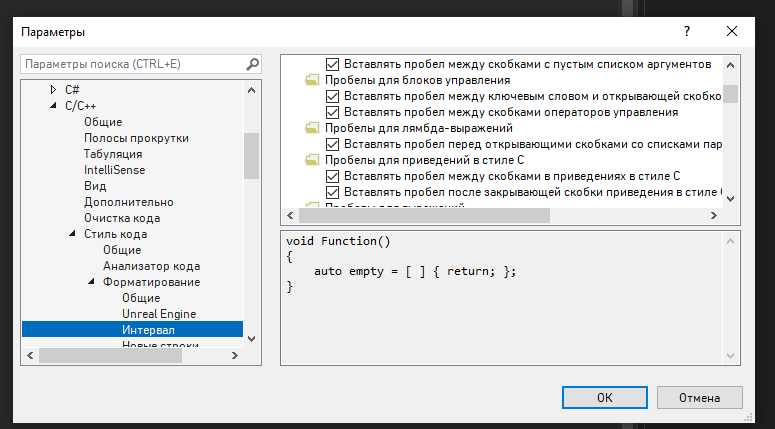


Рисунок 5 – Редактирование интервал

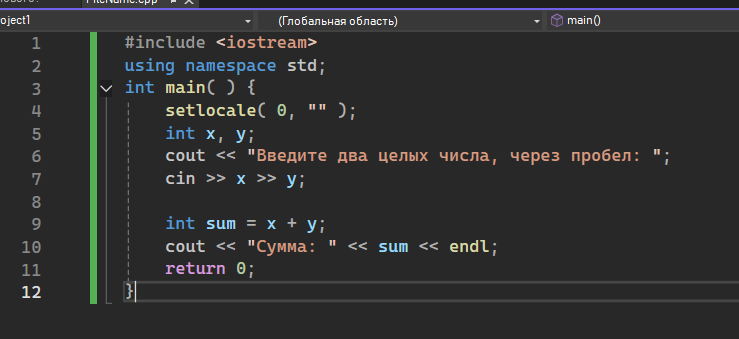


Рисунок 6 – Результат рис.5

Включить нумерацию строк (Рисунок 7-8).

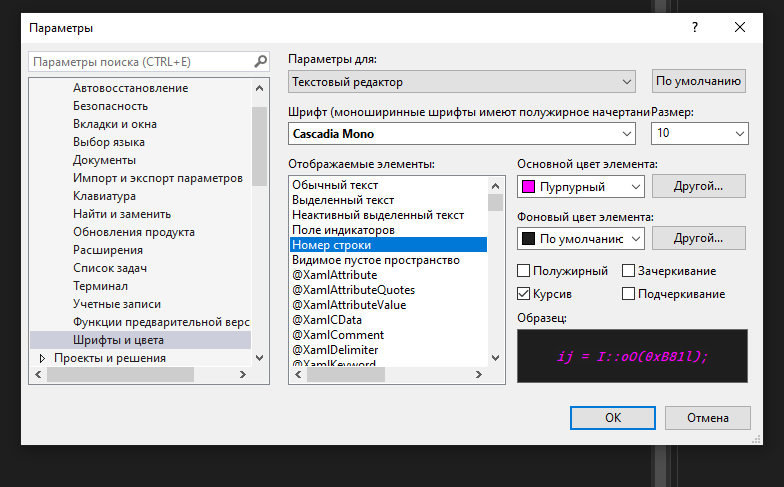


Рисунок 7 - Номера строк

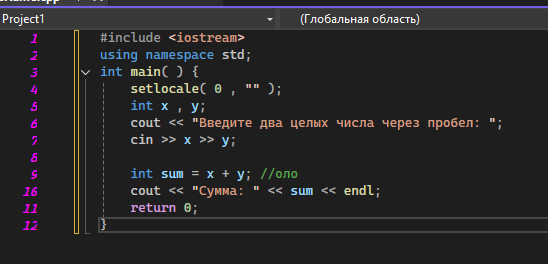


Рисунок 8 – Результат рис.8

1.2. Задание 2

На панели инструментов найти Средства, нажать на Параметры, Окружение, Шрифты и цвета. Изменить в настройках цвет для всплывающих подсказок (Рисунок 9-10), для комментариев поменять цвет (Рисунок 11-12), для строковых литералов поменять цвет (Рисунок 13-14), для текста среды поменять цвет, шрифт, (Рисунок 15-16), для чисел поменять цвет (Рисунок 17-18).

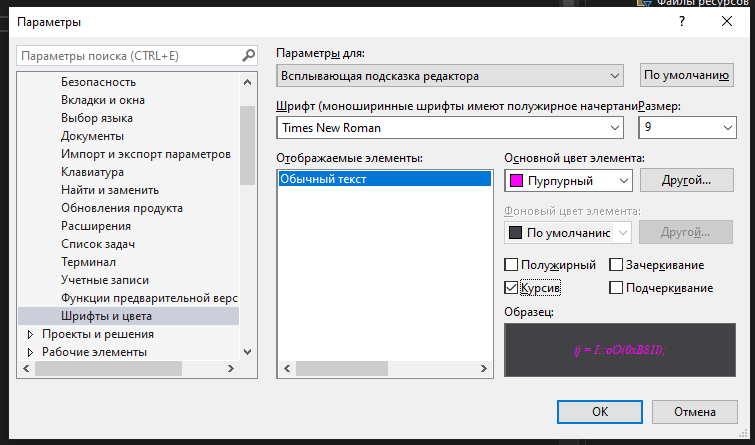


Рисунок 9 - Подсказка

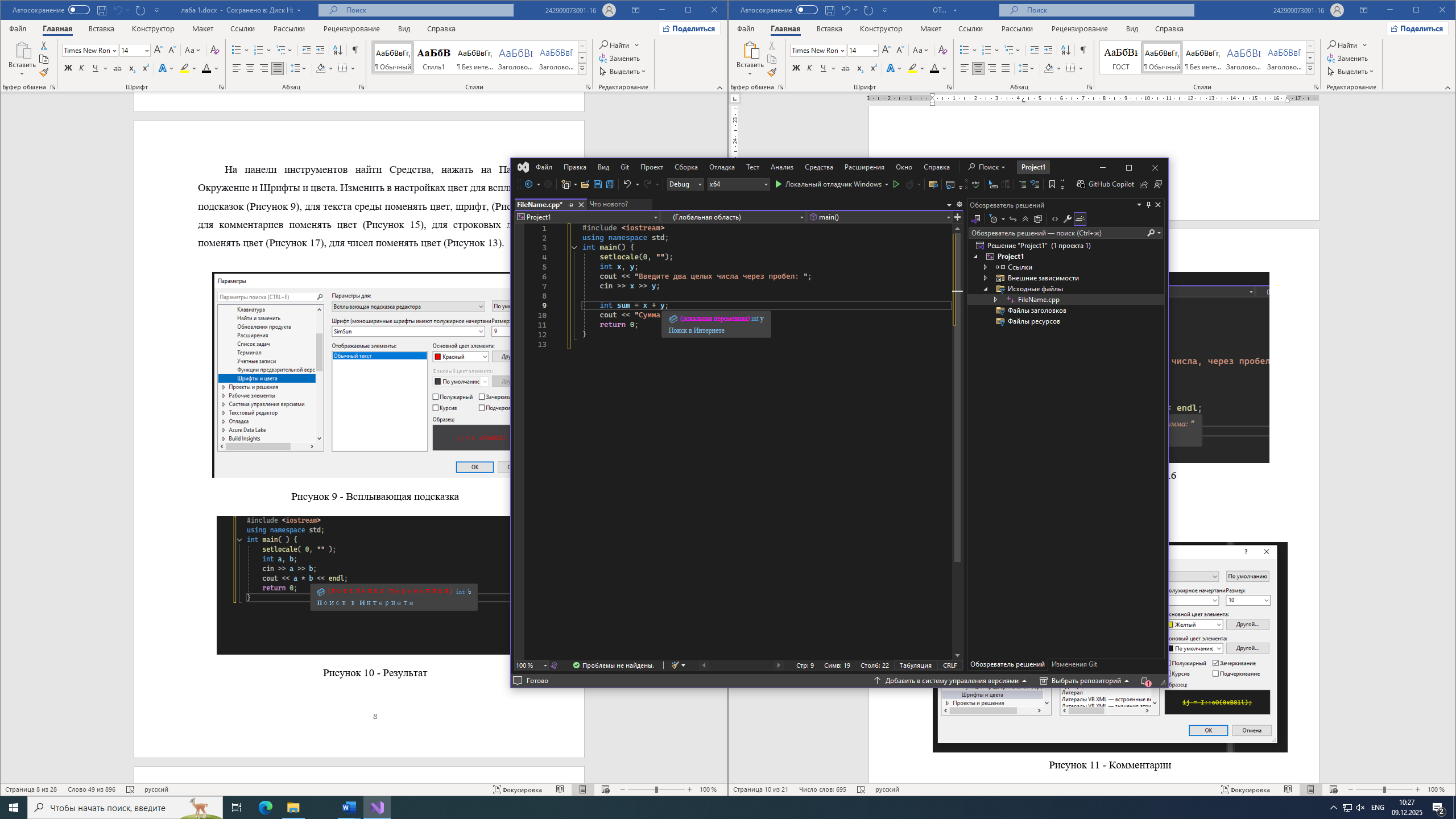


Рисунок 10 – Результат рис.6

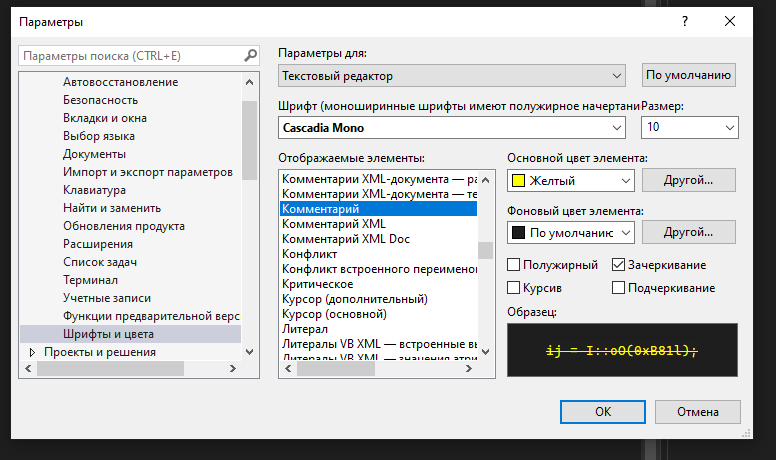


Рисунок 11 - Комментарии

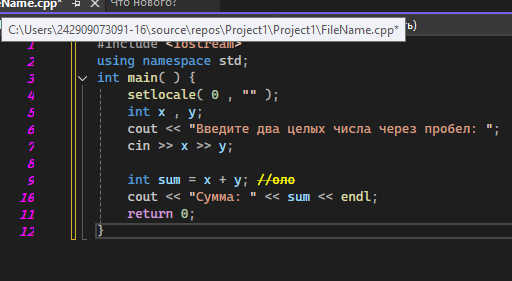


Рисунок 12 – Результат рис.10

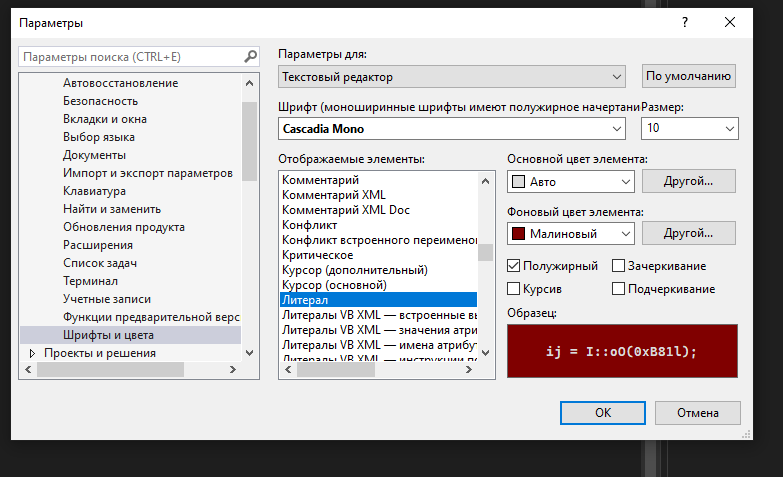


Рисунок 13 - Литералы

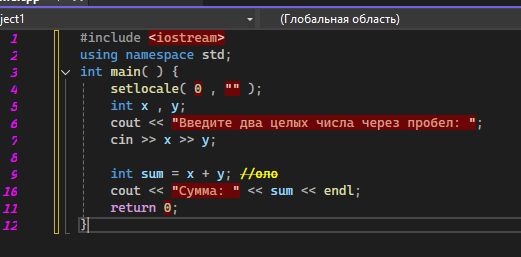


Рисунок 14 – Результат рис.12

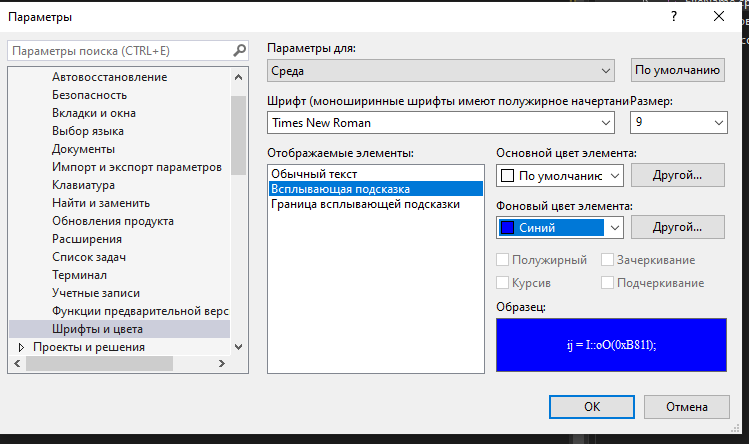


Рисунок 15 – Текст среды

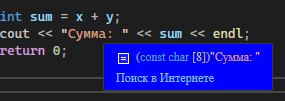


Рисунок 16 – Результат рис.15

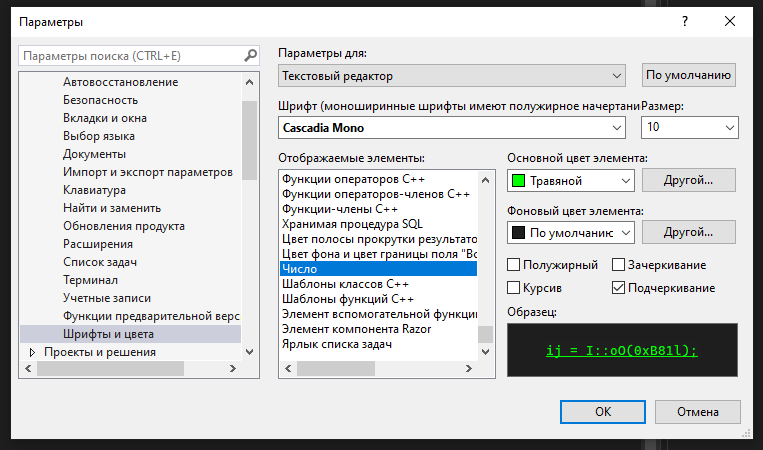


Рисунок 17 - Число

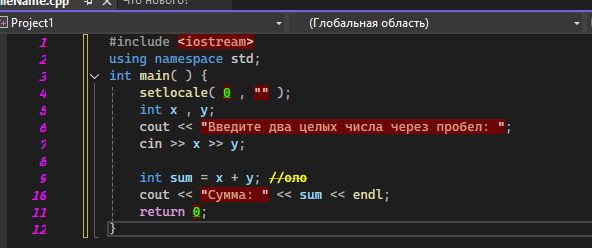


Рисунок 18 – Результат рис.17

1.3. Задание 3

На панели инструментов найти Средства, Настройки. Нажать на создать, сохранить под своей фамилией (Рисунок 19). Добавить команды на панель инструментов (Рисунок 20). Назначить горячие клавиши через Средства → Параметры → Окружение → Клавиатура (Рисунок 21-25).

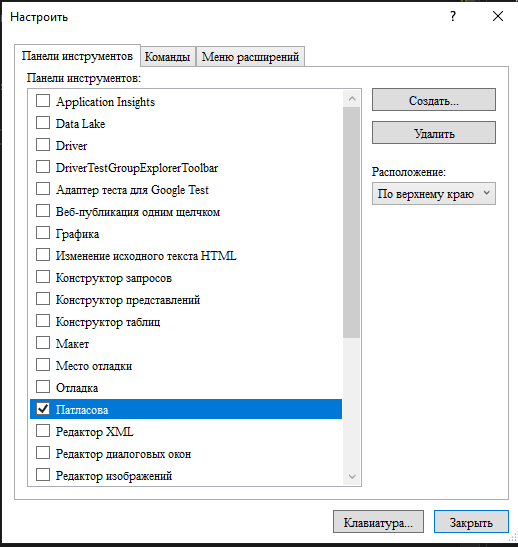


Рисунок 19 – Создание панели

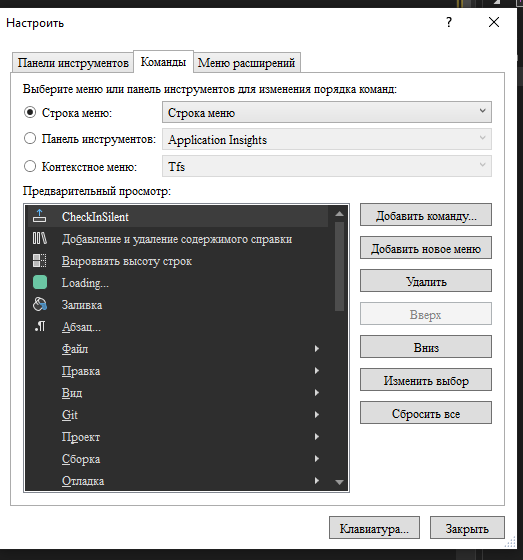


Рисунок 20 – Добавление команд

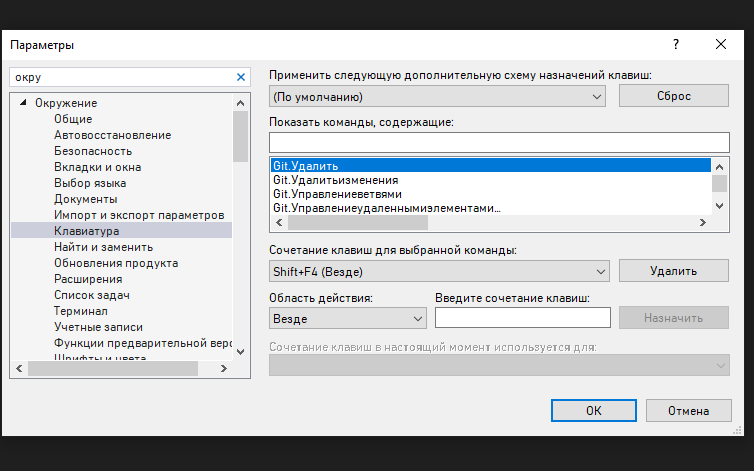


Рисунок 21 - Команда 1

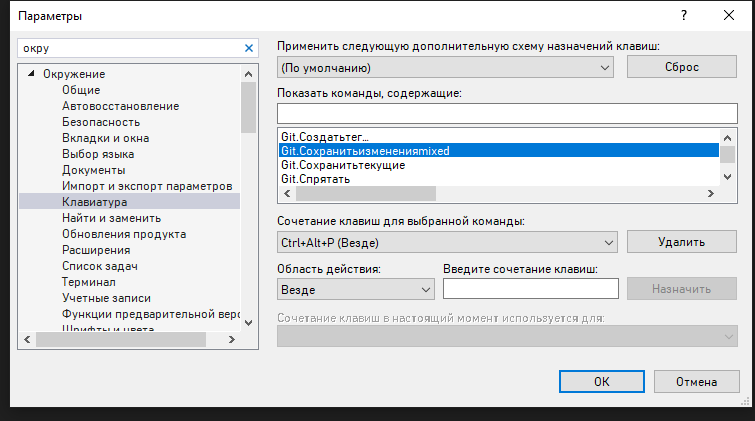


Рисунок 22 - Команда 2

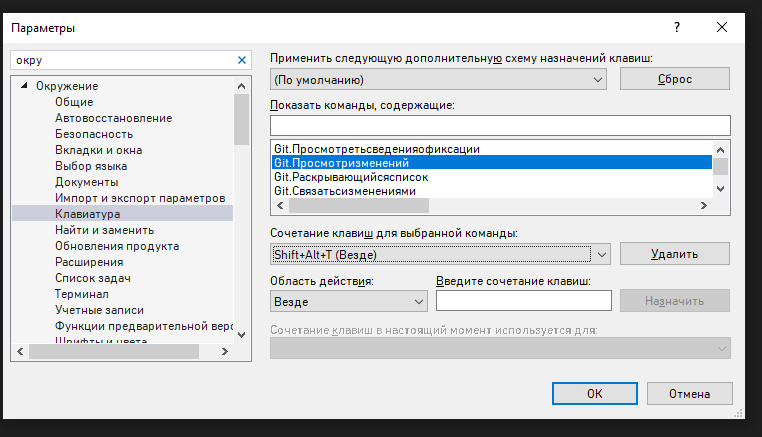


Рисунок 23 - Команда 3

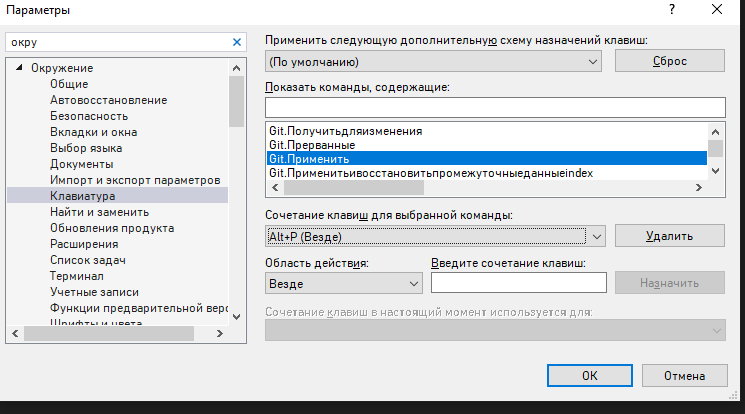


Рисунок 24 - Команда 4

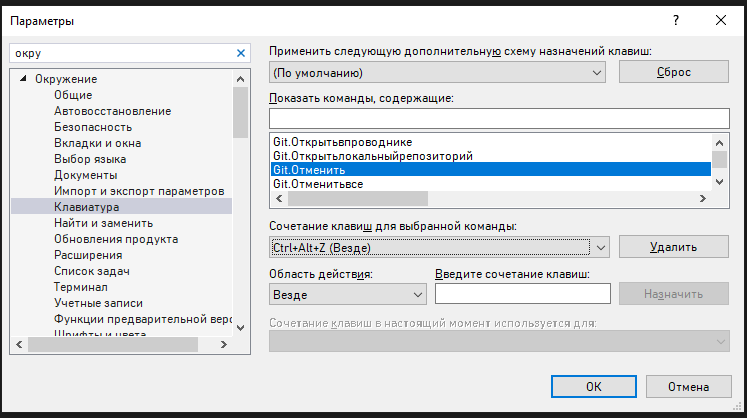
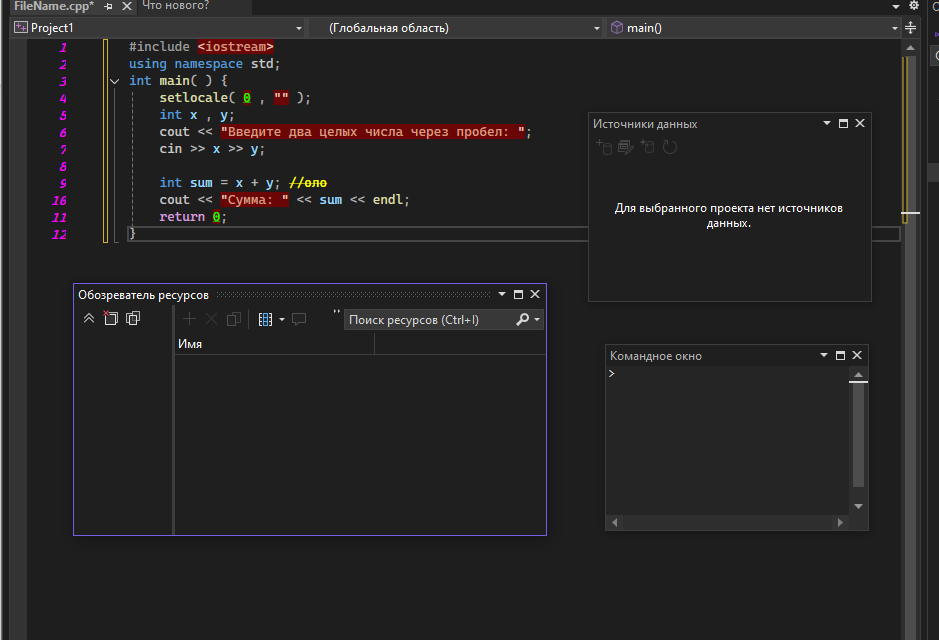


Рисунок 25 - Команда 5

 Рисунок 26 - Добавление окон

1.4. Задание 4

На панели инструментов выбрать Вид, добавить другие окна. В моем случае это: Обозреватель ресурсов, Командное окно, Источники данных (см. рис. 26)

Далее я снова нажимаю Вид на панели инструментов, нахожу «Сохранить макет окна» (Сохраняются нынешние окна и то, как они открыты!) Сохранила под своей фамилией (Рисунок 27). Я применяю макет (Рисунок 28), у меня открываются три окна: Обозреватель ресурсов, Командное окно, Источники данных.

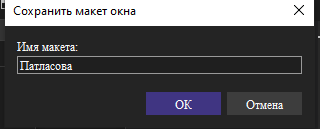


Рисунок 27 - Сохранение макета

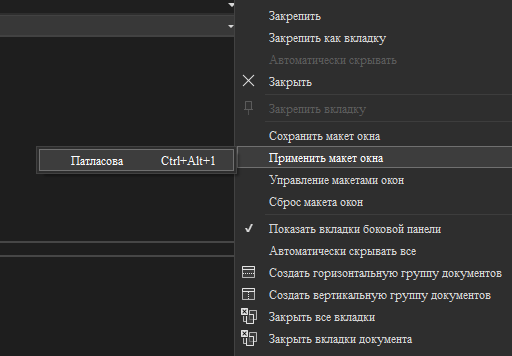


Рисунок 28 - Применение макета Патласова

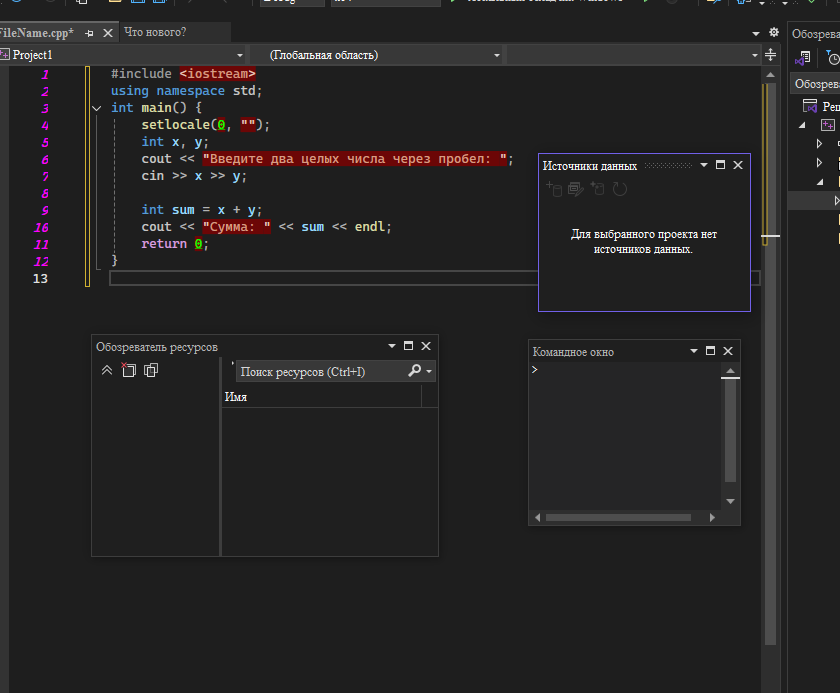


Рисунок 29 – Результат применения макета

Далее я открываю другие окна (операции инструментальных средств, обозреватель ресурсов, структуру), сохраняю Новый макет 1 аналогичным способом. Применяю его (Рисунок 30).

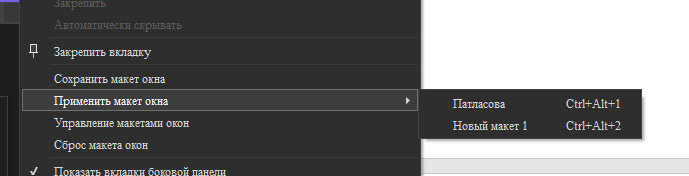


Рисунок 30 - Применение макета

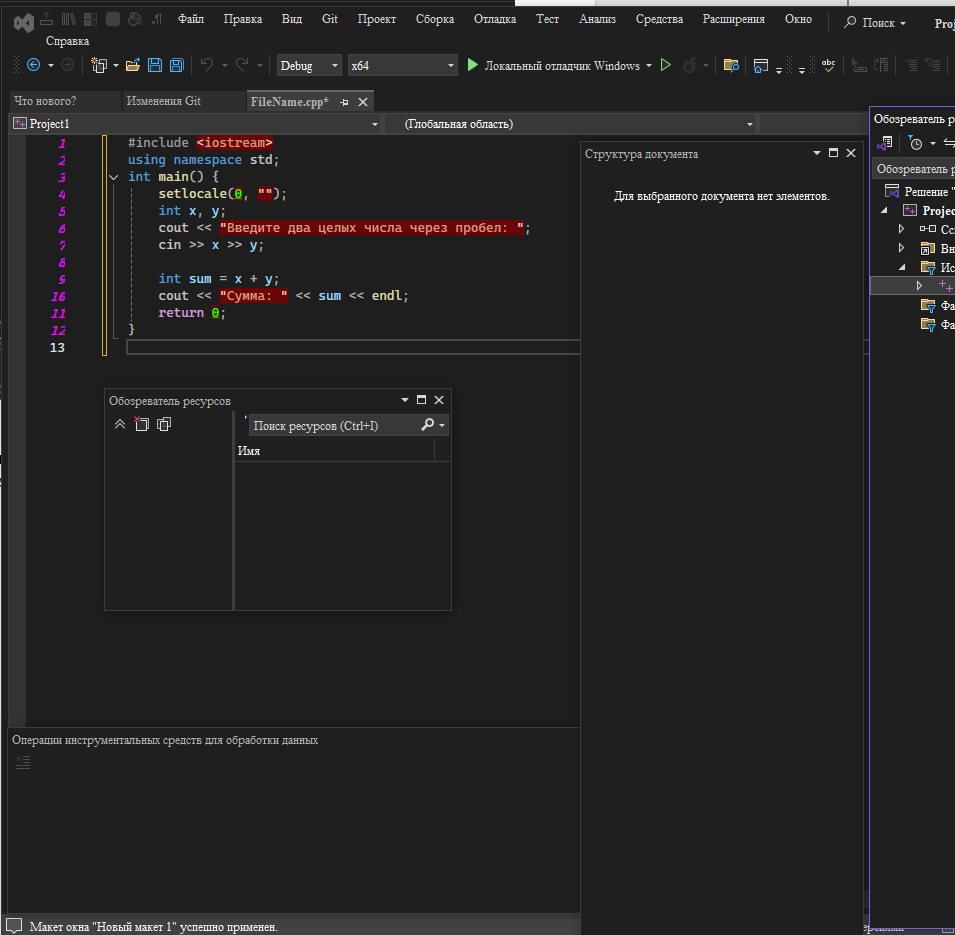


Рисунок 31 - Результат применения нового макета

На рис. 31 наблюдаю, что макет применился, т.к. появились новые окна: Операции инструментальных средств для обработки данных, Обозреватель ресурсов, Структура документа.

1.5. Задание 5

Сделать экспорт, сброс и импорт параметров через окно «Средства» на панели инструментов.

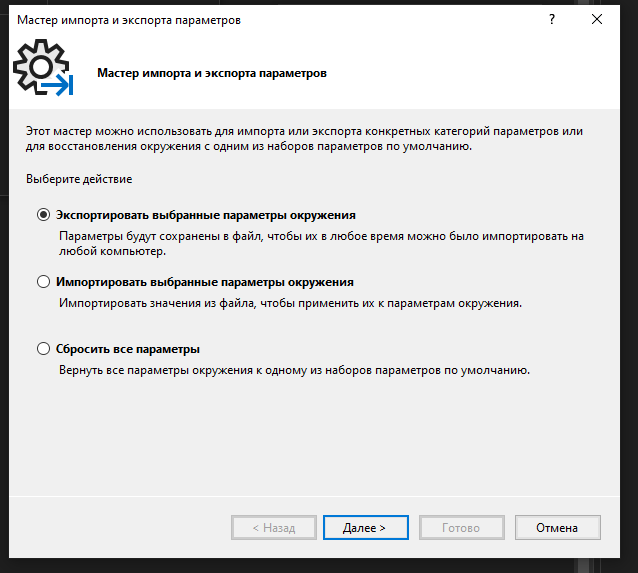


Рисунок 32 - Экспорт параметров



Рисунок 33 - Результат рис.32

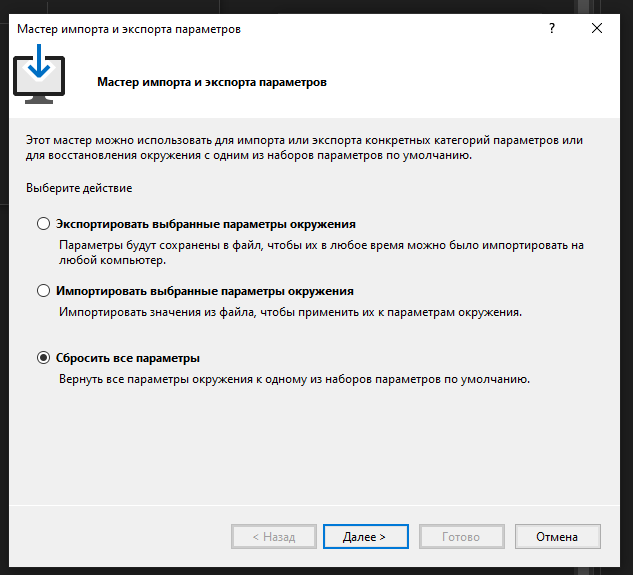


Рисунок 34 - Сброс параметров

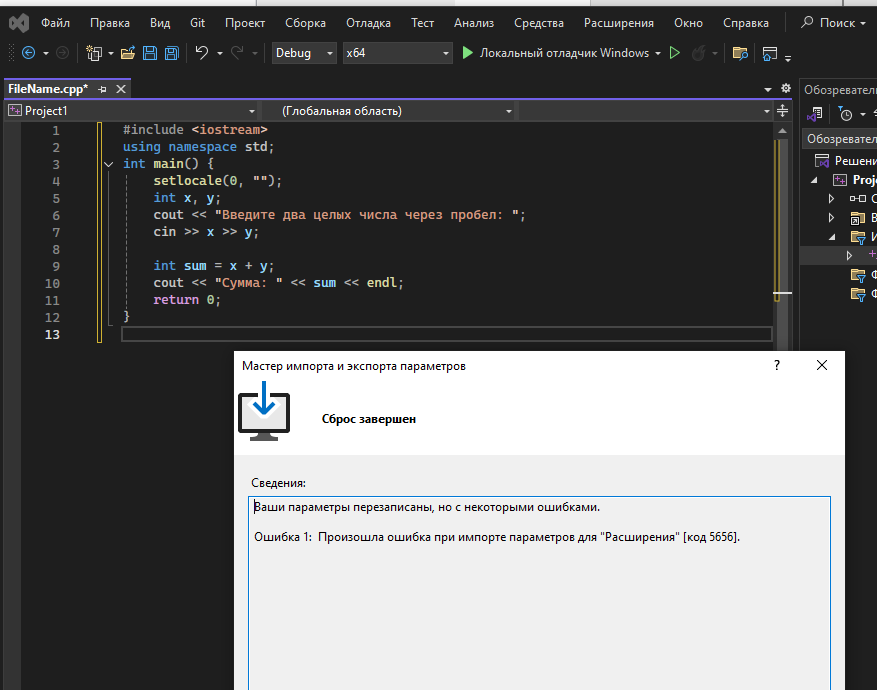


Рисунок 35 - Результат рис.34

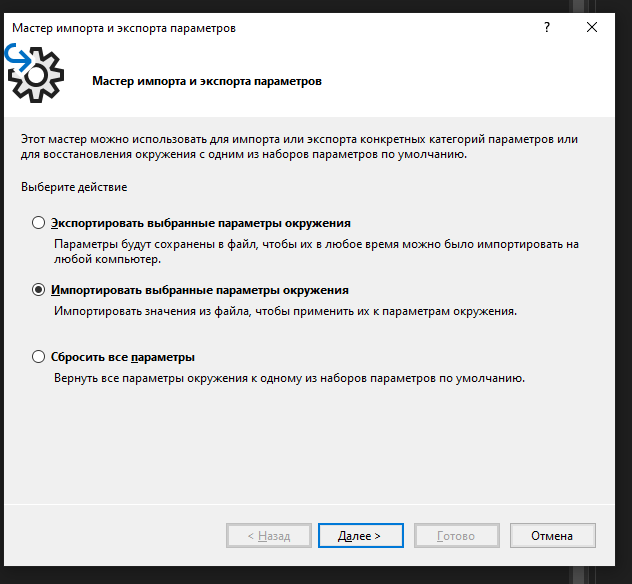


Рисунок 36 - Импорт параметров

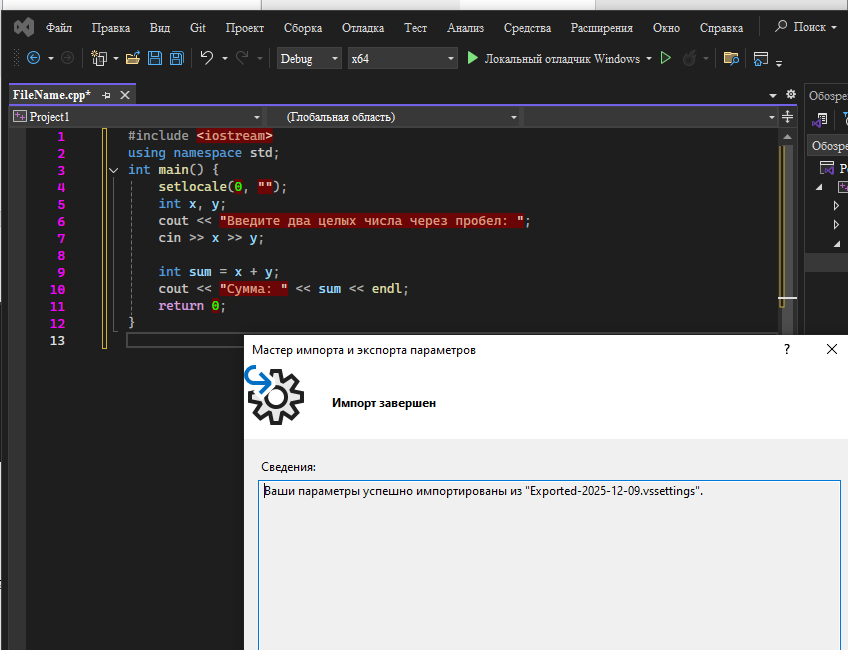


Рисунок 37 - Результат рис.36

2. Практическая работа №2 «Отладка в Visual Studio»

Цель: освоить базовые и продвинутые техники отладки.

2.1 Задание 1. Навигация по коду с помощью отладчика

Код программы:

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "");

cout << "=== Задание 1: цикл для отладки ===" << endl;

// Цикл 1..10 — здесь будем ставить обычную и условную точку останова

for (int i = 1; i <= 10; ++i)

{

cout << "i = " << i << endl;

}

cout << "Нажмите Enter для выхода..." << endl;

cin.get();

return 0;

}

Показать установку обычной точки останова на строке цикла в функции main, чтобы зафиксировать место начала пошагового выполнения программы и продемонстрировать использование breakpoints в Visual Studio (Рисунок 38).

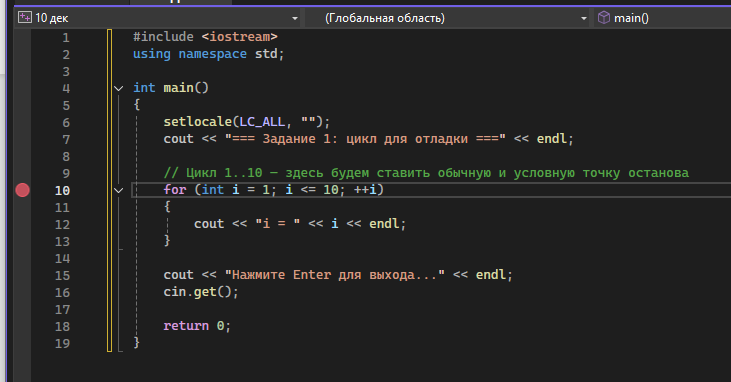


Рисунок 38 - Точка останова

Показать остановку программы на установленной точке останова после запуска отладки клавишей F5 (Рисунок 39).

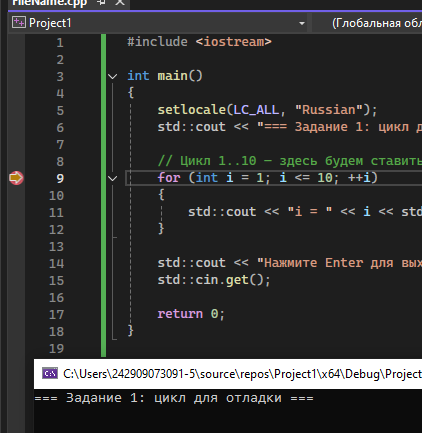


Рисунок 39 - Остановка breakpoints

Далее используем шаг с обходом, то есть выполняется текущая строка. Выполняется нажатием клавиши F10 (Рисунок 40).

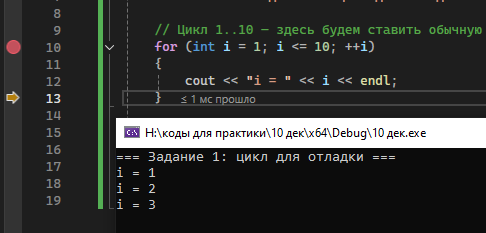


Рисунок 40 - Шаг с обходом(F10)

Используем шаг с заходом, данный шаг дает возможность «Зайти» в сложный оператор, например, перейти к функции, которая выполняется. Выполняется нажатием клавиши F11 (Рисунок 41).

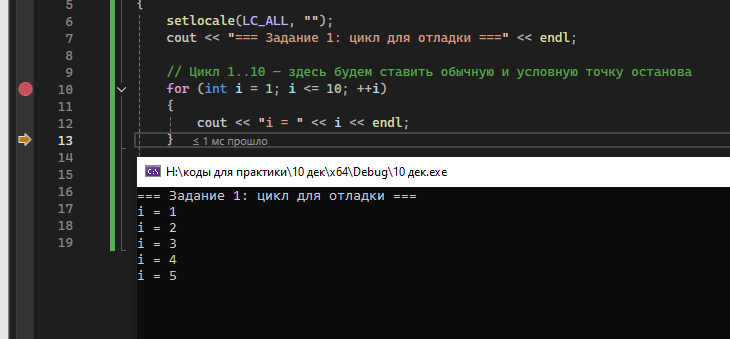


Рисунок 41 - Шаг с заходом(F11)

Шаг с выходом позволяет вернуться на уровень выше, если был совершен шаг с заходом. Выполняется нажатием клавиши Shift*+*F11 (Рисунок 48).

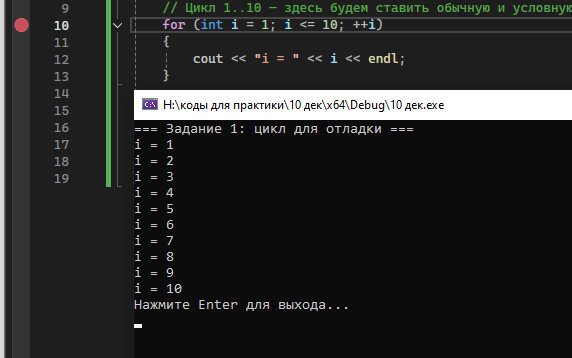


Рисунок 42 - Шаг с выходом (Shift+F11)

Далее навести мышь на красную точку, нажать на правую кнопку мыши, выбрать условие и вписать i == 5, чтобы продемонстрировать остановку программы только при выполнении заданного логического условия для переменной цикла (Рисунок 43).

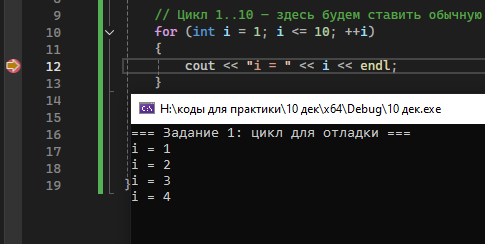


Рисунок 43 - Условная точка останова

Чтобы открыть Локальное окно, нужно зайти в отладку, выбрать окна и в списке найти Локальное окно (Рисунок 44).

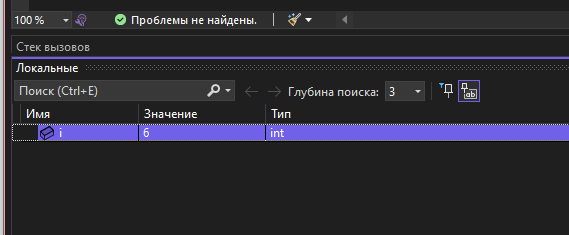


Рисунок 44 – Локальное окно

Чтобы открыть окно Контрольные значения, нужно зайти в отладку, выбрать окна и в списке найти Контрольные значения (Рисунок 45).

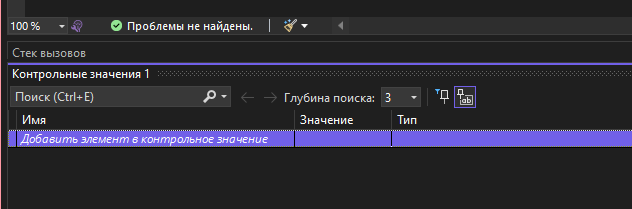


Рисунок 45 - Контрольные значения

Чтобы открыть окно Видимые, нужно зайти в отладку, выбрать окна и в списке найти Видимые (Рисунок 46).

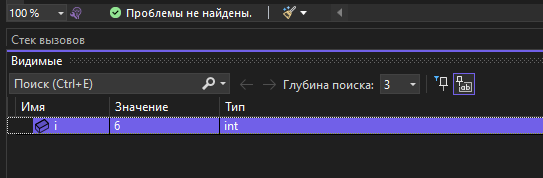


Рисунок 46 – Видимые

Чтобы открыть окно Быстрая проверка, нужно навести курсор мыши на i, нажать на правую кнопку и выбрать Быстрая проверка (Рисунок 47-48).

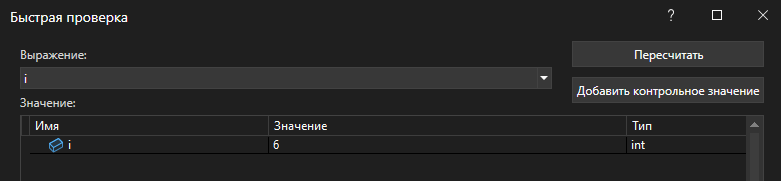


Рисунок 47 - Быстрая проверка

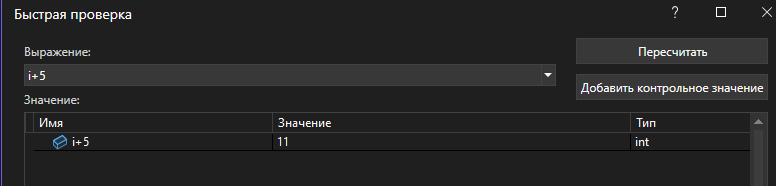


Рисунок 48 - Быстрая проверка 2

2.2. Задание 2. Управление исключениями

Код программы:

#include <iostream>

#include <locale.h>

#include <stdexcept> // для runtime\_error

#include <exception> // для exception

using namespace std;

int main()

{

setlocale(LC\_ALL, "");

cout << "=== Задание 2: исключение (деление на ноль) ===" << endl;

// ЗАДАНИЕ 2: намеренно вызываем исключение при делении на ноль

int a = 10;

int b = 0;

try

{

cout << "a = " << a << ", b = " << b << endl;

// Проверяем делитель

if (b == 0)

{

// Явно генерируем C++-исключение

throw std::runtime\_error("Деление на ноль");

}

int c = a / b; // до сюда не дойдём при b == 0

cout << "Результат деления: c = " << c << endl;

}

catch (const std::exception& ex)

{

cout << "Исключение перехвачено: " << ex.what() << endl;

}

cout << "Нажмите Enter для выхода..." <<endl;

cin.get();

return 0;

}

Для того, чтобы найти Параметры исключений, нужно зайти в Откладку, потом в Окна, выбрать Параметры исключений. В настройках поставить галочку рядом с С++ (Рисунок 49).

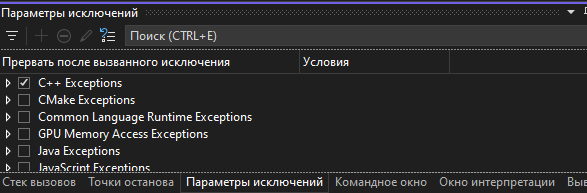


Рисунок 49 - Параметры исключений

Запустить код. На экране выводиться сообщение о том, что вызвано исключение (Рисунок 50).

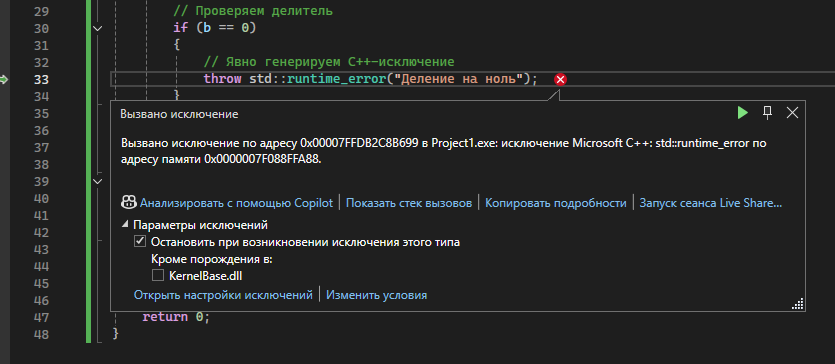


Рисунок 50 - Вызванное исключение

2.3. Задание 3. Профилирование

Код программы:

#include <iostream>

#include <vector>

#include <chrono>

int main()

{

setlocale(LC\_ALL, "");

std::cout << "=== Задание 3: профилирование ===" << std::endl;

const int iterations = 1'000'000; // 1 000 000 итераций

std::vector<int> data;

data.reserve(iterations); // заранее выделяем память

auto start = std::chrono::high\_resolution\_clock::now();

long long sum = 0;

for (int i = 0; i < iterations; ++i)

{

data.push\_back(i); // потребляет память

sum += i; // нагружает CPU

}

auto end = std::chrono::high\_resolution\_clock::now();

auto ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count();

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

std::cout << "Время выполнения = " << ms << " мс" << std::endl;

std::cout << "Нажмите Enter для выхода..." << std::endl;

std::cin.get();

return 0;

}

Запустить код нажатием клавиши F5 (Рисунок 51).

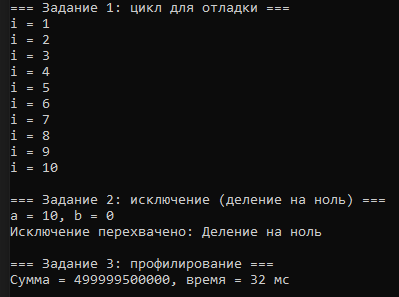


Рисунок 51 - Результат кода

Чтобы найти Профилировщик производительности, надо нажать на Отладку, найти Профилировщик производительности. Далее надо поставить галочку рядом с Использование памяти и ЦП, начать (Рисунок 52).

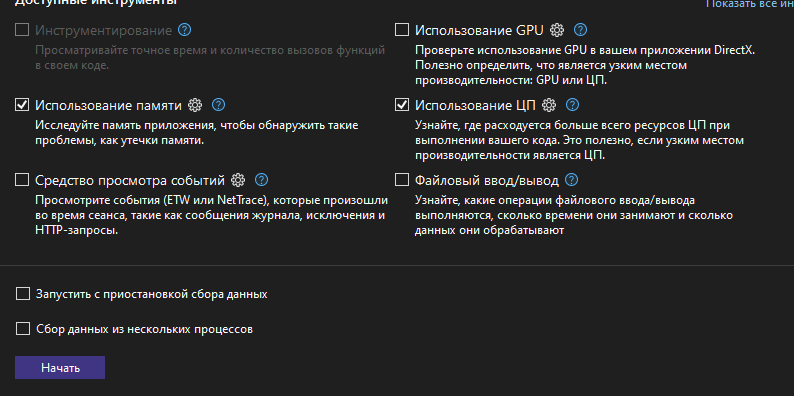


Рисунок 52 - Настройка ЦП

Откроется отчет, где будет Память процессора и % всех процессоров (Рисунок 53).

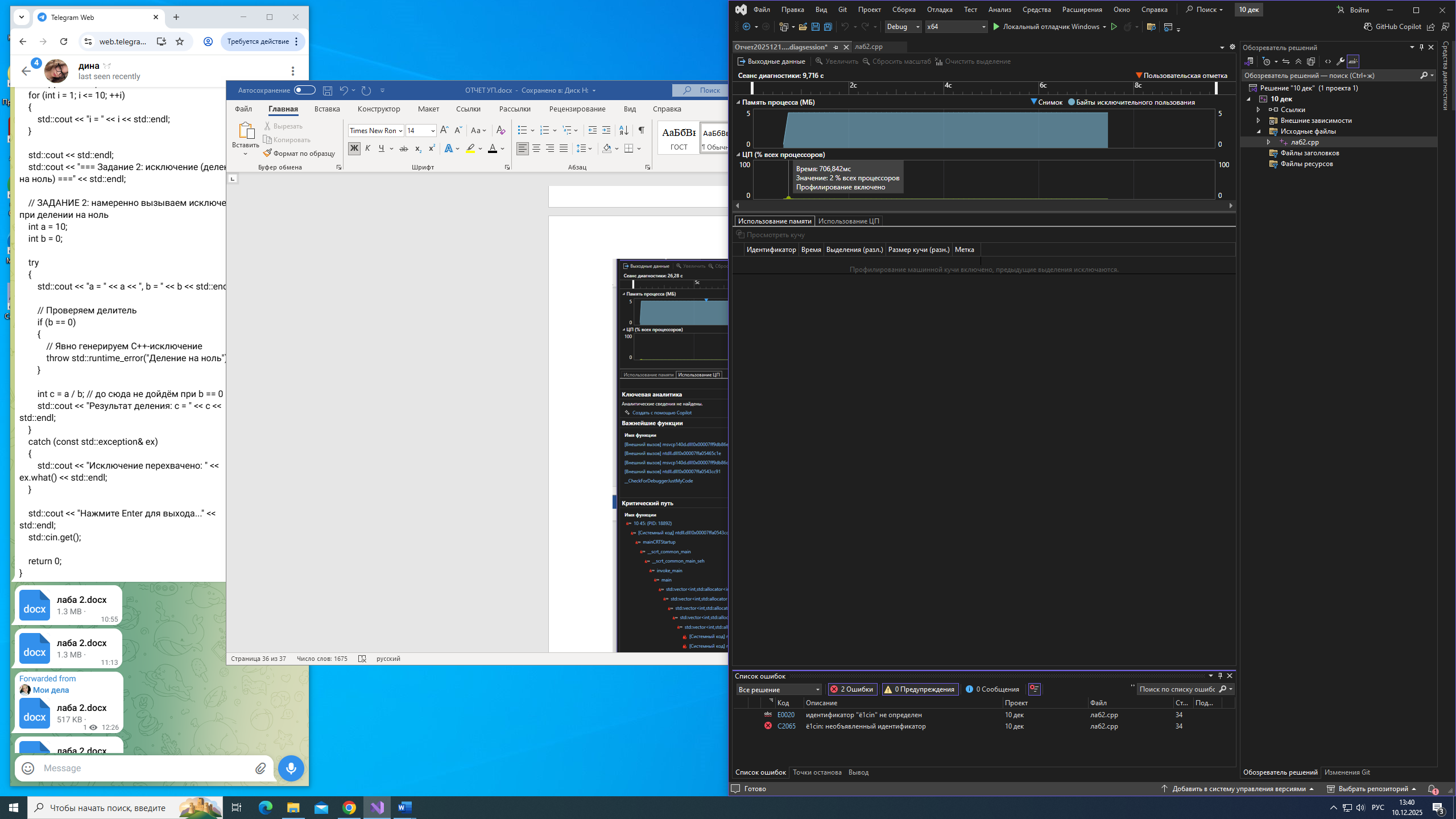


Рисунок 53 – Графики

Для того, чтобы вывести использование ЦП, надо перейти в меню использование ЦП, выведется окно с графиком загрузки процессора и таблицей вызовов, где основное время выполнения приходится на функцию main с большим циклом, чтобы продемонстрировать определение наиболее загруженного участка кода (Рисунок 54).

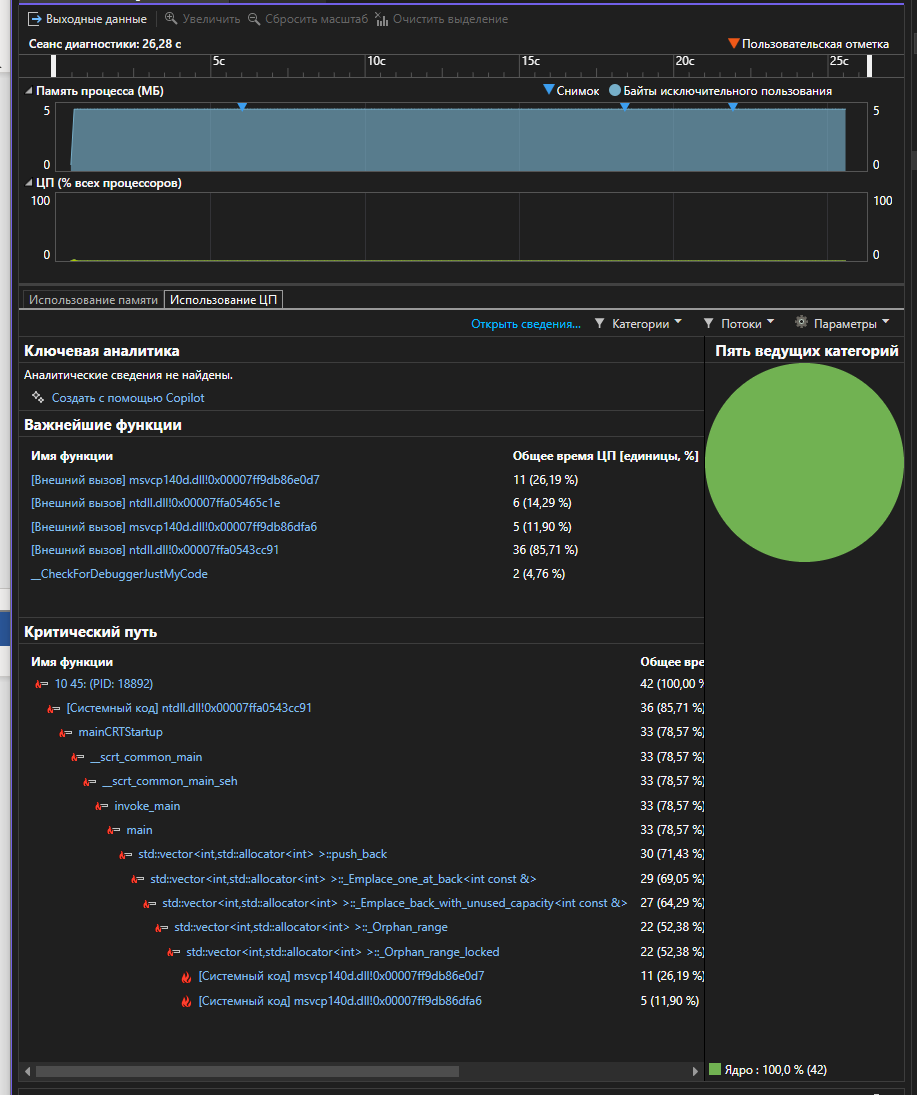


Рисунок 54 – Использование ЦП

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

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

3.1. Задание 1: Самодокументирующийся код

Вариант 5. Создайте функцию для вычисления периметра квадрата. Функция должна принимать длину стороны и возвращать периметр. Продемонстрируйте работу функции в main.

Код программы:

#include <iostream>

// <summary>

// Вычисляет периметр квадрата.

// </summary>

// <param name="side">Длина стороны квадрата.</param>

// <returns>

// Периметр квадрата, если длина стороны положительна.

// Возвращает -1, если длина стороны некорректна (меньше либо равна нулю).

// </returns>

double calculateSquarePerimetr(double side)

{

if (side <= 0.0)

{

return -1.0;

}

return 4. \* side;

}

int main()

{

setlocale(LC\_ALL, "");

double side = 5.0;

double perimetr = calculateSquarePerimetr(side);

std::cout << "Длина стороны квадрата: " << side << std::endl;

std::cout << "Периметр квадрата: " << perimetr << std::endl;

return 0;

}

Запуск программы (Рисунок 55).

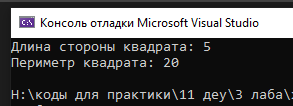


Рисунок 55 - Запуск кода

Чтобы вывести подсказки, которые были изначально вписаны в код можно навести курсор на текст с маленьким красным прямоугольником (например, на имя переменной) - при наведении курсора открывается всплывающая подсказка (Рисунок 56).

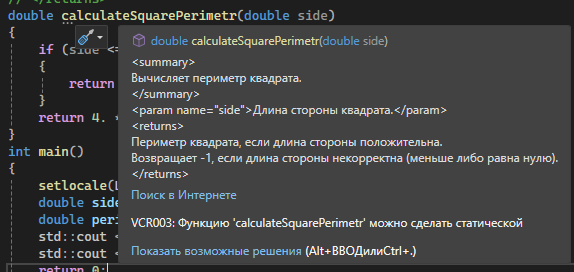


Рисунок 56 – Подсказки

3.2. Задание 2: Соглашение о кодировании

Цель: Оформить свой код согласно правилам С++

На Рисунке 57 абсолютно рабочий код, однако неправильно оформлены отступы, нет пробелов вокруг операторов, нечитаемая структура.

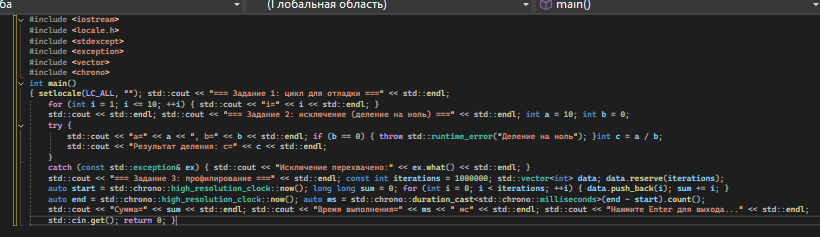


Рисунок 57 – Код не по правилам С++

Код до:

#include <iostream>

#include <locale.h>

#include <stdexcept>

#include <exception>

#include <vector>

#include <chrono>

int main() { setlocale(LC\_ALL, ""); std::cout << "=== Задание 1: цикл для отладки ===" << std::endl;

for (int i = 1; i <= 10; ++i) { std::cout << "i=" << i << std::endl; }

std::cout << std::endl; std::cout << "=== Задание 2: исключение (деление на ноль) ===" << std::endl; int a = 10; int b = 0;

try {

std::cout << "a=" << a << ", b=" << b << std::endl; if (b == 0) { throw std::runtime\_error("Деление на ноль"); }int c = a / b;

std::cout << "Результат деления: c=" << c << std::endl;

}

catch (const std::exception& ex) { std::cout << "Исключение перехвачено:" << ex.what() << std::endl; }

std::cout << "=== Задание 3: профилирование ===" << std::endl; const int iterations = 1000000; std::vector<int> data; data.reserve(iterations);

auto start = std::chrono::high\_resolution\_clock::now(); long long sum = 0; for (int i = 0; i < iterations; ++i) { data.push\_back(i); sum += i; }

auto end = std::chrono::high\_resolution\_clock::now(); auto ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count();

std::cout << "Сумма=" << sum << std::endl; std::cout << "Время выполнения=" << ms << " мс" << std::endl; std::cout << "Нажмите Enter для выхода..." << std::endl;

std::cin.get(); return 0;}

На Рисунке 58 тот же код, но уже оформленный в соответствии с правилами code style C++ (правильно оформлены отступы, есть пробелы вокруг операторов, единый стиль фигурных скобок, читаемая структура).

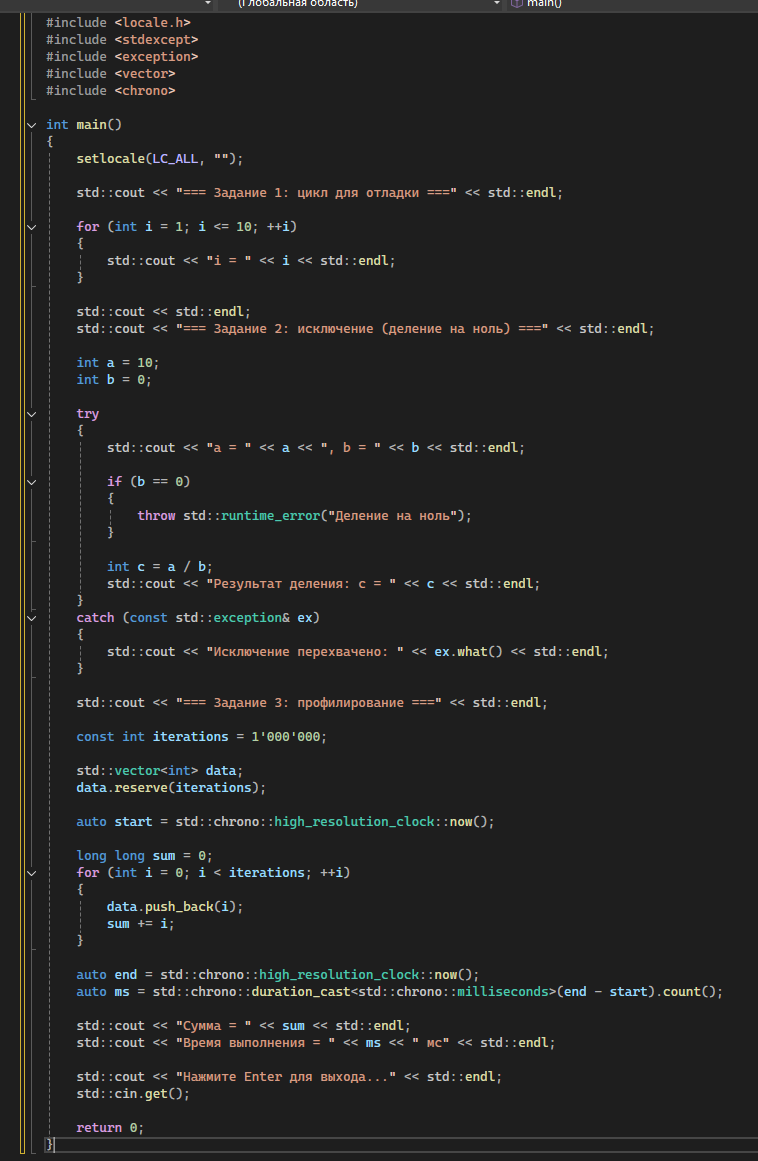


Рисунок 58 - Код по правилам С++

Код после:

#include <locale.h>

#include <stdexcept>

#include <exception>

#include <vector>

#include <chrono>

int main()

{

setlocale(LC\_ALL, "");

std::cout << "=== Задание 1: цикл для отладки ===" << std::endl;

for (int i = 1; i <= 10; ++i)

{

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

}

std::cout << std::endl;

std::cout << "=== Задание 2: исключение (деление на ноль) ===" << std::endl;

int a = 10;

int b = 0;

try

{

std::cout << "a = " << a << ", b = " << b << std::endl;

if (b == 0)

{

throw std::runtime\_error("Деление на ноль");

}

int c = a / b;

std::cout << "Результат деления: c = " << c << std::endl;

}

catch (const std::exception& ex)

{

std::cout << "Исключение перехвачено: " << ex.what() << std::endl;

}

std::cout << "=== Задание 3: профилирование ===" << std::endl;

const int iterations = 1'000'000;

std::vector<int> data;

data.reserve(iterations);

auto start = std::chrono::high\_resolution\_clock::now();

long long sum = 0;

for (int i = 0; i < iterations; ++i)

{

data.push\_back(i);

sum += i;

}

auto end = std::chrono::high\_resolution\_clock::now();

auto ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count();

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

std::cout << "Время выполнения = " << ms << " мс" << std::endl;

std::cout << "Нажмите Enter для выхода..." << std::endl;

std::cin.get();

return 0;

}

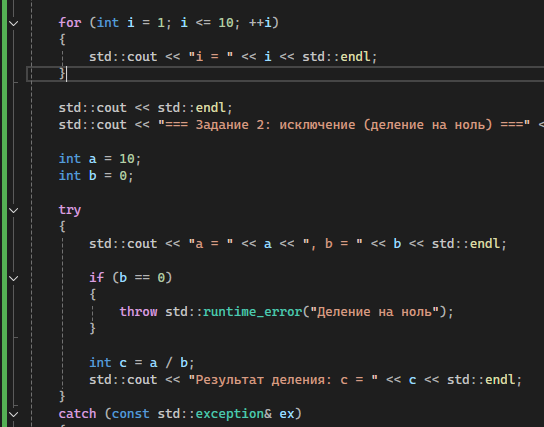


Рисунок 59 - Выравнивание отступов

Во всём коде выполнено выравнивание отступов, чтобы каждая вложенная конструкция (for, try, catch, if) имела чёткую структуру. (см. рис. 59).

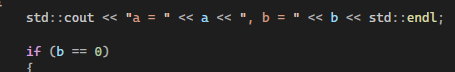


Рисунок 60 - Оформление операторов

Исправлено оформление операторов (=, <<, +, -), чтобы код читался легче и соответствовал стилевым стандартам (см. рис. 60).

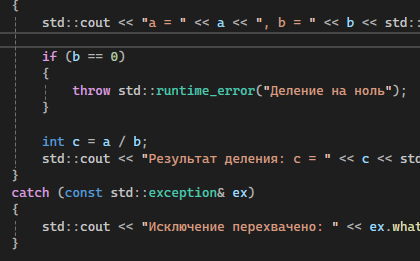


Рисунок 61 - Фигурные скобки

Фигурные скобки приведены к единому стилю (см. рис. 61).

Разделение логических блоков. Каждая часть программы оформлена как самостоятельный блок:

1. Задание 1 (цикл)
2. Задание 2 (исключение)
3. Задание 3 (профилирование)

Между блоками оставлены пустые строки (Рисунок 62-64).

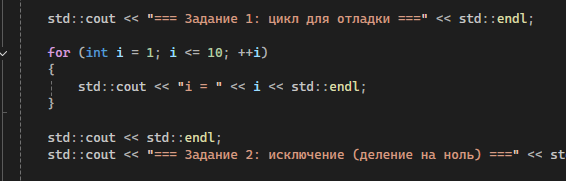


Рисунок 62 – Задание 1

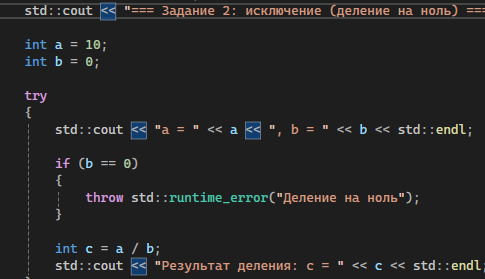


Рисунок 63 – Задание 2

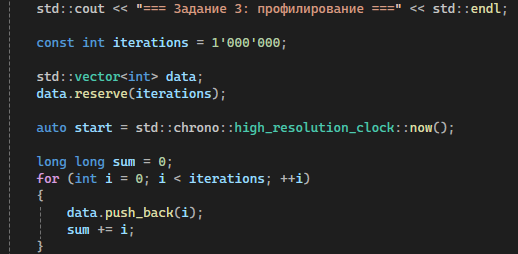


Рисунок 64 – Задание 3

3.3. Задание 3: Метрика Джилба (упрощенный вариант)

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

Исходный код функции:

double calculateSquarePerimeter(double side)

{

if (side <= 0.0)

{

return -1.0;

}

return 4.0 \* side;

}

3.3.1. Подсчёт исполняемых строк (η₁):

Исполняемые строки:

1. return -1.0;
2. return 4.0 \* side, следовательно η₁ = 2

3.3.2. Подсчёт логических переходов (η₂):

Логические конструкции:

один оператор if, следовательно η₂ = 1

Общая метрика Джилба: Gillb = η₁ + η₂ = 2 + 1 = 3

3.3.3. Вывод:

Метрика Джилба равна 3, что относится к **низкому уровню сложности.**  
Функция простая, легко понимаемая и хорошо тестируемая.

Расчёт метрики Джилба для функции calculateSquarePerimeter:

Код функции:

double calculateSquarePerimeter(double side)

{

if (side <= 0.0)

{

return -1.0;

}

return 4.0 \* side;

}

3.3.4. Подсчёт η₁ - количество исполняемых строк кода (исполняемыми строками считаем операторы, которые выполняются во время работы программы):

1. return -1.0;
2. return 4.0 \* side;

Следовательно:

η₁ = 2.

3.3.5. Подсчёт η₂ - количество логических переходов.

В функции используется один оператор ветвления: if (side <= 0.0). Циклов и дополнительных логических операций (&&, ||) нет.

Следовательно:

η₂ = 1.

3.3.6. Вычисление метрики Джилба:

Gillb = η₁ + η₂ = 2 + 1 = 3.

3.3.7. Вывод:

Значение Gillb = 3 попадает в диапазон низкой сложности (Gillb ≤ 10). Функция имеет простую линейную структуру, содержит одну проверку условия и две исполняемые строки. Код легко понимать, тестировать и поддерживать.

4. Практическая работа №4 «Работа с реестром ОС Windows»

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

Вариант 16

Сохранить и вывести любимый музыкальный инструмент.

Сохранить цвет фона консоли при выводе инструмента.

4.1 Задание

Для написания программы по данному заданию, нужно добавить к базовым библиотекам библиотеку <fstream> ( для работы с файлами ifstream, ofstream ), <string> (удобные строковые переменные). Они нужны для того, чтобы программа могла записывать данные в файл, работать со строками. Основные переменные: string instrument (любимый музыкальный инструмент), string color (цвет консоли). Для проверки открыт ли файл используется if, else if. Если он существует, значит это не первый запуск, если нет, то это первый запуск программы. Далее через char спрашивается у пользователя хочет ли он поменять настройки. y – меняет музыкальный инструмент и цвет консоли, n – оставляет все как есть, и программа завершается.

Код программы:

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

// функция1, установка цвета фона консоли

void setConsoleColor(const string& color) {

if (color == "black") system("color 07");

else if (color == "blue") system("color 17");

else if (color == "green") system("color 27");

else if (color == "red") system("color 47");

else if (color == "yellow") system("color 67");

else if (color == "white") system("color 77");

else system("color 0F"); // по умолчанию

}

// функция2, загрузка и сохранение данных

void loadAndSave() {

setlocale(LC\_ALL, "Russian");

const string filename = "settings.txt";

string instrument;

string color;

ifstream fin(filename);

// если файл существует

if (fin.is\_open()) {

fin >> instrument >> color;

fin.close();

// установка цвета сразу после загрузки

setConsoleColor(color);

cout << "Сохранённые данные:\n";

cout << "Любимый музыкальный инструмент: " << instrument << "\n";

cout << "Цвет фона консоли: " << color << "\n\n";

cout << "Хотите изменить данные? (y/n): ";

char choice;

cin >> choice;

if (choice == 'y' || choice == 'Y') {

cout << "Введите новый музыкальный инструмент (одно слово на англ): ";

cin >> instrument;

cout << "Введите цвет фона (black, blue, green, red, yellow, white): ";

cin >> color;

ofstream fout(filename);

fout << instrument << " " << color;

fout.close();

cout << "Данные обновлены!\n";

}

}

// если файл не найден

else {

cout << "Файл не найден. Введите данные впервые.\n";

cout << "Введите любимый музыкальный инструмент (одно слово на англ): ";

cin >> instrument;

cout << "Введите цвет фона (black, blue, green, red, yellow, white): ";

cin >> color;

ofstream fout(filename);

fout << instrument << " " << color;

fout.close();

cout << "Данные сохранены!\n";

setConsoleColor(color);

}

}

int main() {

loadAndSave();

return 0;

}

Запуск программы нажатием клавиши F5. Пользователь вводит настройки (Рисунок 65).

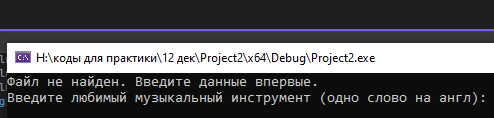


Рисунок 65 - Запуск программы 1

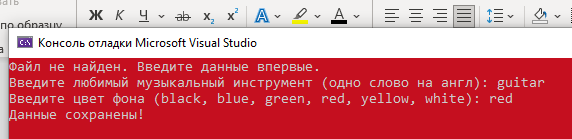


Рисунок 66 - Результат запуска 1

На втором запуске высвечивается ранее сохраненные настройки и предложение изменить их (Рисунок 67).

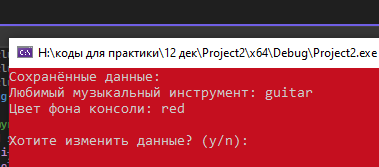


Рисунок 67 – Запуск программы 2

Далее пользователь набирает «y» и меняет настройки, когда доходит до смены цвета, пользователь выбирает нужный цвет из представленных вариантов: black, blue, green, red, yellow, white (Рисунок 68). Если пользователь не хочет менять настройки, то набирает «n» (Рисунок 69).

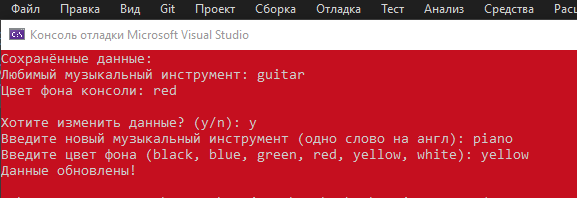


Рисунок 68 – Результат запуска 2

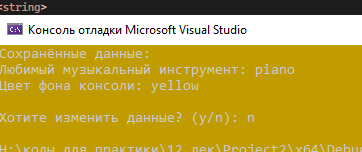


Рисунок 69 – Результат запуска 3

Настройки сохраняются в папке как settings.txt (Рисунок 67).

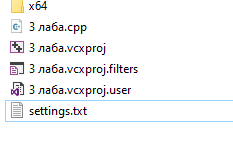


Рисунок 70 - Файл settings.txt

Открыть файл можно в самом Microsoft Visual Studio (Рисунок 72).

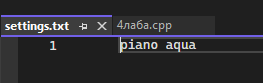


Рисунок 71 - Открытие файла, Visual Studio

5. Практическая работа №5 «Задание для тестировщика»

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

Задача: необходимо проверить сайт как с точки зрения рядового пользователя, так и с технической стороны. По итогам тестирования предоставить отчет, включающий в себя:

Тестирование сайта ([Разговоры о важном — Официальный сайт](https://xn--80aafadvc9bifbaeqg0p.xn--p1ai/)). Окружение: ПК: Windows + Chrome.

5.1 Краткое резюме

Сайт Разговоры о важном – образовательный портал с тематическими материалами. ЦА: обучающиеся, педагоги. Сайт выполнен в современном стиле, адаптирован под мобильные устройства.

5.2 Список багов

Баги отсутствуют.

Тест 1 «Битая ссылка».

Шаги: перейти на сайт по поиску ломанных ссылок, вставить ссылку на сайт, нажать на эту ссылку, проверить рабочая она или нет.

Ожидаемый результат: ссылка нерабочая.

Фактический результат: ссылка рабочая.

Окружение: ПК, Windows + Chrome.

Вывод: Ломанных ссылок нет (Рисунок 72).

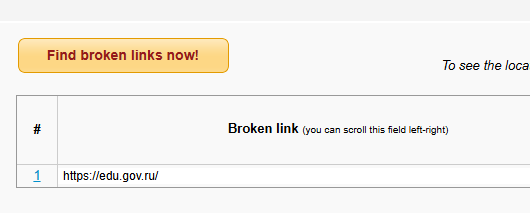


Рисунок 72 - Тест 1

Тест 2 «Скорость загрузки изображений».

Шаги: перейти на сайт для определения скорости загрузки изображения, вставить ссылку на сайт, проверить время загрузки изображений.

Ожидаемый результат: изображения грузятся моментально, за мс.

Фактический результат: изображения грузятся моментально, за мс.

Окружение: ПК, Windows + Chrome.

Вывод: изображения грузятся моментально, за мс (Рисунок 73).

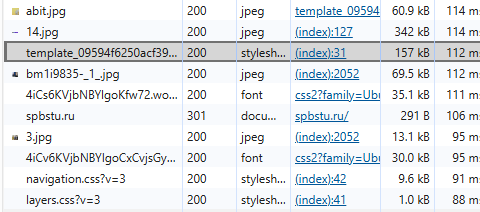


Рисунок 73 - Тест 2

5.3. Функциональное тестирование (Functional Testing)

Тест 3 «Ссылки».

Шаги: перейти на главную, выбрать методичку, нажать на нее, проверить открывается ли она.

Ожидаемый результат: ссылка рабочая, открывается новое окно.

Фактический результат: ссылка рабочая, открывается новое окно.

Окружение: ПК, Windows + Chrome.

Вывод: ссылка функционирует (Рисунок 74).

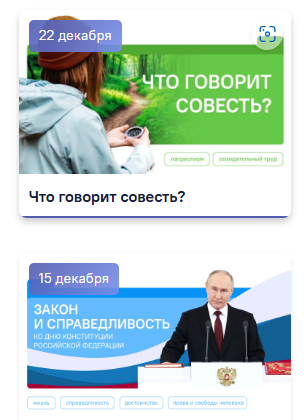


Рисунок 74 - Тест 3

Тест 4 «Кликабельность иконок».

Шаги: перейти на главную страницу, прокрутить до конца, нажать на иконку, проверить рабочая она или нет.

Ожидаемый результат: перекидывает в тг.

Фактический результат: перекидывает в тг.

Окружение: ПК, Windows + Chrome.

Вывод: Иконка ТГ рабочая (Рисунок 75).



Рисунок 75 - Тест 4

Тест 5 «Фильтры».

Шаги: перейти на главную страницу, выбрать период, нажать на фильтр с выбранным периодом, проверить работает ли фильтр.

Ожидаемый результат: фильтр выдает материалы за выбранный период.

Фактический результат: фильтр выдает материалы за выбранный период.

Окружение: ПК, Windows + Chrome.

Вывод: Фильтры рабочие (Рисунок 76).

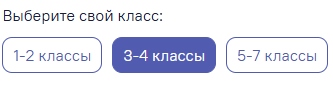


Рисунок 76 - Тест 5

Тест 6 «Кликабельность кнопок».

Шаги: перейти на главную страницу, нажать кнопку домашние питомцы, проверить рабочая она или нет.

Ожидаемый результат: кнопка рабочая, открывается новое окно.

Фактический результат: кнопка рабочая, открывается новое окно.

Окружение: ПК, Windows + Chrome.

Вывод: Кнопки рабочие (Рисунок 77).



Рисунок 77 - Тест 6

Тест 7 «Валидация».

Шаги: перейти на главную страницу, нажать на конверт, тем самым осуществив переход на форму, ввести неправильные данные, понять проверяет ли форма правильность данных.

Ожидаемый результат: вывод сообщения о том, что поле заполнено неверно.

Фактический результат: вывод сообщения о том, что поле заполнено неверно.

Окружение: ПК, Windows + Chrome.

Вывод: Форма проверяет на правильность данных (Рисунок 78).

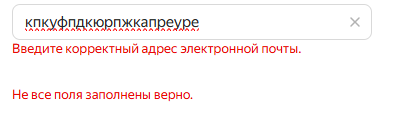


Рисунок 78 - Тест 7

Тест 8 «Интерактивные элементы».

Шаги: перейти на главную страницу, нажать на ссылку с методичкой, в открывшемся окне, выбрать Интерактивные элементы, выполнить задание, чтобы понять работают ли элементы.

Ожидаемый результат: элементы рабочие, после переноса текста в квадратик появляется картинка.

Фактический результат: элементы рабочие, после переноса текста в квадратик появляется картинка.

Окружение: ПК, Windows + Chrome.

Вывод: элементы рабочие (Рисунок 79).

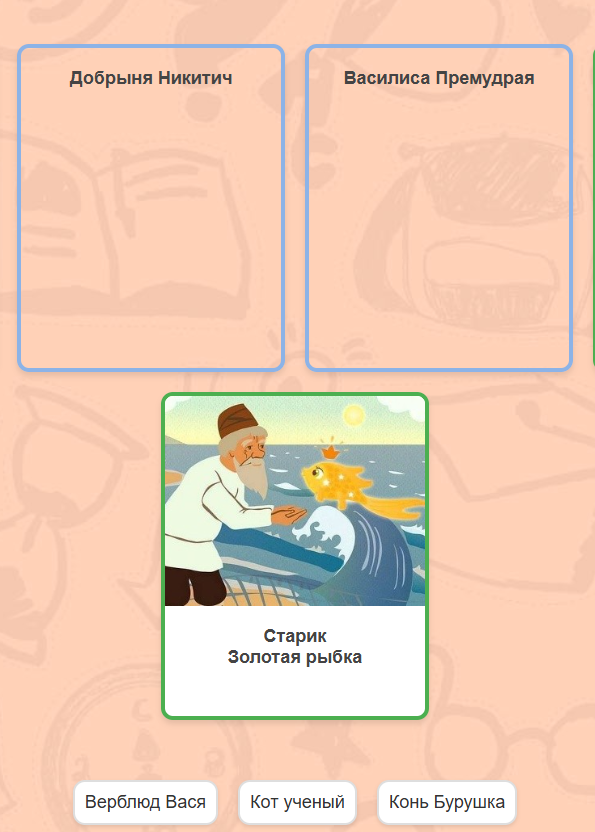


Рисунок 79 - Тест 8

5.4. Юзабилити (Usability) и Пользовательский опыт (UX)

Тест 9 «Контент».

Шаги: перейти на главную страницу, проверить читабельность текста, проверить насколько понятна цель существования сайта, нет ли грамматических ошибок.

Ожидаемый результат: текст читабелен, цель ясна, грамматических ошибок нет.

Фактический результат текст читабелен, цель ясна, грамматических ошибок нет.

Окружение: ПК, Windows + Chrome.

Вывод: текст читабелен, цель ясна, грамматических ошибок нет (Рисунок 80-81).

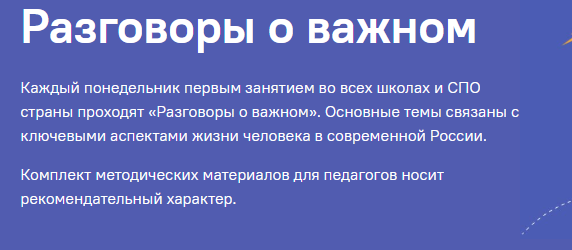


Рисунок 80 - Тест 9

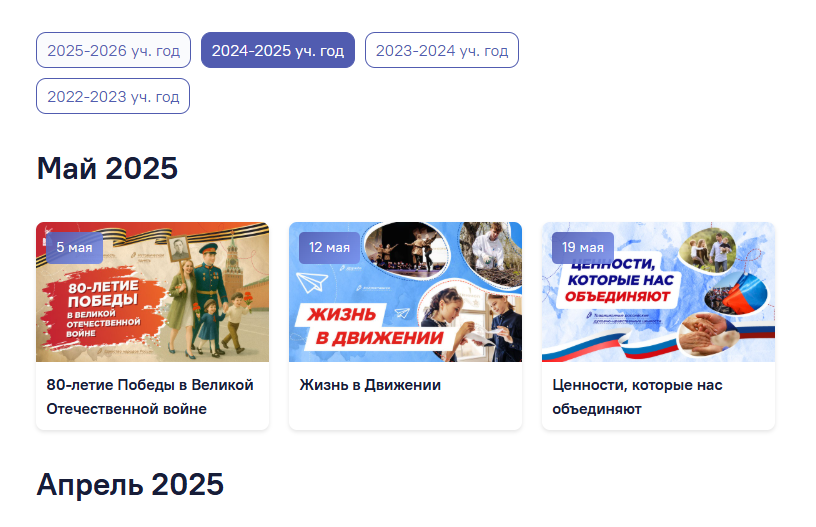


Рисунок 81 - Тест 9

Тест 10 «Структура и Навигация».

Шаги: перейти на главную страницу, проверить легко ли понять, где пользователь находится на сайте, можно ли добраться до любой важной страницы за 2-3 клика.

Ожидаемый результат: легко понять, где пользователь находится, можно добраться до любой важной страницы за 2-3 клика.

Фактический результат: легко понять, где пользователь находится, можно добраться до любой важной страницы за 2-3 клика.

Окружение: ПК, Windows + Chrome.

Вывод: структура логична, навигация на хорошем уровне (Рисунок 85).

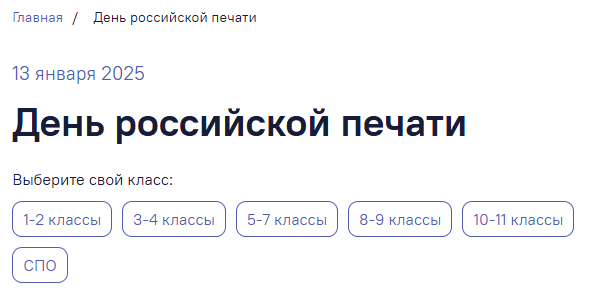


Рисунок 82 - Тест 10

5.5. Тестирование совместимости (Compatibility Testing)

Тест 11 «Совместимость».

Шаги: открыть страницу в разных браузерах, выглядит и работает ли сайт одинаково.

Ожидаемый результат: все работает и выглядит одинаково.

Фактический результат: все работает и выглядит одинаково.

Окружение: ПК, Windows + Chrome, Firefox, Safari, Edge.

Вывод: все работает и выглядит одинаково.

Тест 12 «Адаптивность».

Шаги: открыть страницу на мобильном устройстве, выглядит и работает ли сайт одинаково, не наезжают ли элементы друг друга.

Ожидаемый результат: все работает и выглядит одинаково.

Фактический результат: все работает и выглядит одинаково.

Окружение: смартфон.

Разрешения экрана: проверка на Full HD, все исправно.

Вывод: на моб. устройстве (Рисунок 84), и на разрешении Full HD (Рисунок 83) все работает и выглядит одинаково, элементы не съезжают.

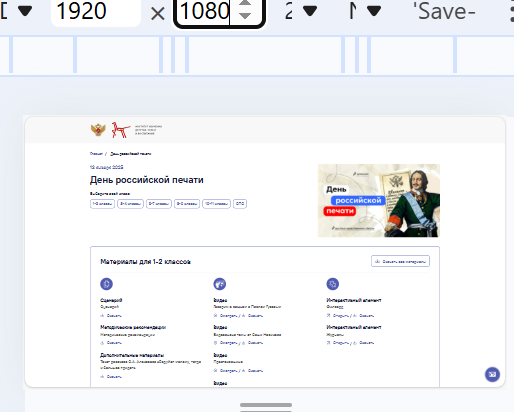


Рисунок 83 - Тест 12 (Full HD)

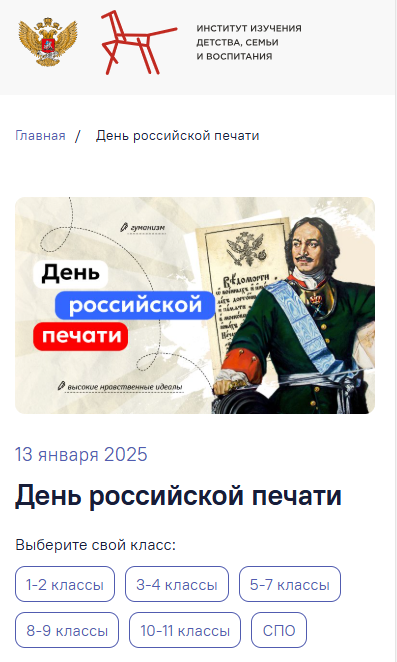


Рисунок 84 - Тест 12 (Моб. устройство)

5.6. Производительность

Тест 13 «Производительность».

Скорость загрузки: моментально.

Скорость реакции: моментально.

Загрузка контента: контент подгружается моментально, сайт не подвисает. Тяжелых видео и изображений нет.

5.7. Тестирование безопасности

Тест 14 «Безопасность».

HTTPS: сайт использует защищенное соединение https:// и замок в адресной строке (Рисунок 85).

Формы: данные не передаются из форм в открытом виде (http).



Рисунок 85 - Тест 14(1)

Попытка обхода: выдалась ошибка (Рисунок 86)



Рисунок 86 - Тест 14(2)

6. Практическая работа №6 «Создание инсталляторов»

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

**Задание 1.** Создание консольного приложения (ConsoleApp) с классом.

Класс «СумматорМассива» (ArraySum).

Задание: Создать класс для работы с массивом чисел. SetArray(arr, size) - установить массив, calculateSum() - вычислить сумму всех элементов, findMax() - найти максимальный элемент, findMin() - найти минимальный элемент

6.1. Создание консольного приложения (ConsoleApp) с классом

Код программы:

#include <iostream>

#include <locale>

using namespace std;

class ArraySum {

private:

double arr[100]; // обычный массив

int size;

public:

// Конструктор по умолчанию

ArraySum() {

size = 0;

}

// Установка массива (без void)

bool setArray(double a[], int s) {

if (s <= 0 || s > 100) return false;

size = s;

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

arr[i] = a[i];

}

return true;

}

// Сумма элементов

double calculateSum() {

double sum = 0.0;

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

sum += arr[i];

}

return sum;

}

// Максимальный элемент

double findMax() {

double max = arr[0];

for (int i = 1; i < size; i++) {

if (arr[i] > max) {

max = arr[i];

}

}

return max;

}

// Минимальный элемент

double findMin() {

double min = arr[0];

for (int i = 1; i < size; i++) {

if (arr[i] < min) {

min = arr[i];

}

}

return min;

}

};

int main() {

setlocale(0, "");

double data[] = { 5.5, 2.1, 9.3, 1.8, 7.0 };

int size = 5;

ArraySum arraySum;

if (!arraySum.setArray(data, size)) {

cout << "Ошибка установки массива!" << endl;

return 1;

}

cout << "Массив чисел: ";

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

cout << data[i] << " ";

}

cout << endl;

cout << "Сумма элементов: " << arraySum.calculateSum() << endl;

cout << "Максимальный элемент: " << arraySum.findMax() << endl;

cout << "Минимальный элемент: " << arraySum.findMin() << endl;

system("pause");

return 0;

}

Запуск кода совершается нажатием клавиши F5 (Рисунок 87).

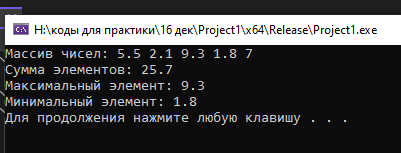


Рисунок 87 - Запуск кода

Далее нужно поменять конфигурацию решения на Release (Рисунок 88).



Рисунок 88 – Release

Следующим пунктом нужно собрать решение. Нужно нажать на Сборка, выбрать Собрать решение (Рисунок 89).

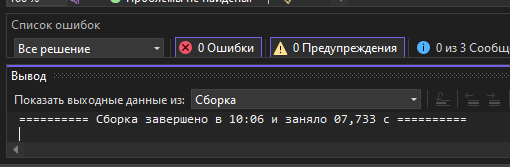


Рисунок 89 – Сборка

Найти файл в папке Release, выбрать файл .exe, вынести на рабочий стол (Рисунок 90).

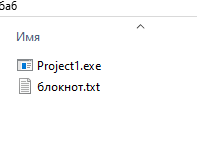


Рисунок 90 - Файл .exe и .txt

На рабочем столе создать папку и назвать своим именем. В именную папку перенести текстовый файл (тут содержится инструкция и описание) и файл .exe. Папку следует заархивировать в zip и отправить другу. Друг, следуя инструкции из .txt (Рисунок 91) извлечет файл из архива (Рисунок 92).

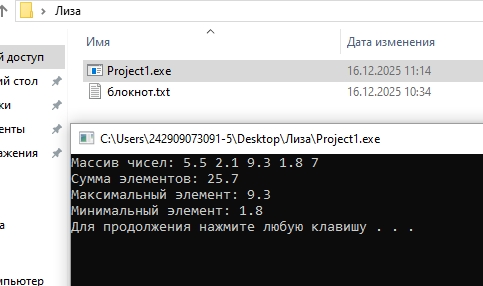


Рисунок 91 – Тестирование (1)

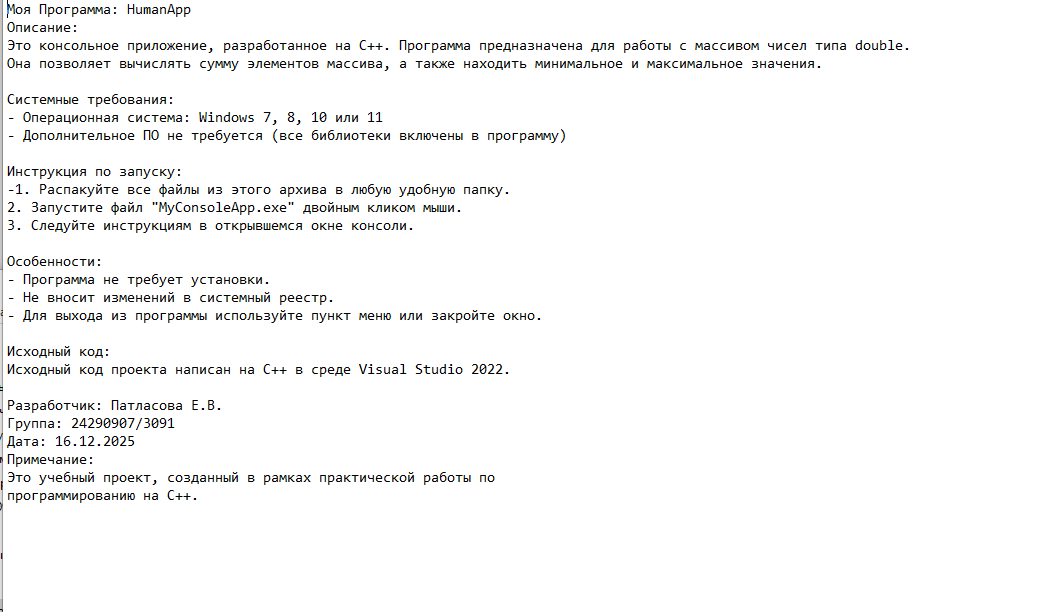


Рисунок 92 – Тестирование (2)

7. Практическая работа №7 «Тестирование»

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

7.1. Задание 1

Создать свою структуру по варианту. Реализовать функции (методы) структуры по варианту. Изучить и расписать отличия каждого вида тестирования.

Вариант 1. Структура для операций с дробями.

Написать структуру для операций с дробями, реализовать методы:

1. static void addFractions() - сложение двух дробей
2. static void multiplyFractions() - умножение двух дробей
3. static void simplifyFraction() - упрощение дроби
4. static double toDecimal() - преобразование дроби в десятичное число.

Код программы:

#include <iostream>

#include <stdexcept>

struct Fraction

{

int numerator; // числитель

int denominator; // знаменатель

// сложение дробей

static void addFractions(const Fraction& a, const Fraction& b, Fraction& result)

{

if (a.denominator == 0 || b.denominator == 0)

throw std::invalid\_argument("Знаменатель не может быть равен 0");

result.numerator = a.numerator \* b.denominator

+ b.numerator \* a.denominator;

result.denominator = a.denominator \* b.denominator;

simplifyFraction(result);

}

// умножение дробей

static void multiplyFractions(const Fraction& a, const Fraction& b, Fraction& result)

{

if (a.denominator == 0 || b.denominator == 0)

throw std::invalid\_argument("Знаменатель не может быть равен 0");

result.numerator = a.numerator \* b.numerator;

result.denominator = a.denominator \* b.denominator;

simplifyFraction(result);

}

// упрощение дроби

static void simplifyFraction(Fraction& f)

{

if (f.denominator == 0)

throw std::invalid\_argument("Знаменатель не может быть равен 0");

int a = f.numerator < 0 ? -f.numerator : f.numerator;

int b = f.denominator < 0 ? -f.denominator : f.denominator;

while (b != 0)

{

int t = b;

b = a % b;

a = t;

}

f.numerator /= a;

f.denominator /= a;

// только положительный знаменатель

if (f.denominator < 0)

{

f.denominator = -f.denominator;

f.numerator = -f.numerator;

}

}

// преобразование в десятичное число

static void toDecimal(const Fraction& f, double& result)

{

if (f.denominator == 0)

throw std::invalid\_argument("Знаменатель не может быть равен 0");

result = static\_cast<double>(f.numerator) / f.denominator;

}

};

int main()

{

setlocale(0, "");

Fraction f1{ 1, 2 };

Fraction f2{ 3, 4 };

Fraction res;

double dec;

Fraction::addFractions(f1, f2, res);

std::cout << "Сумма: " << res.numerator << "/" << res.denominator << std::endl;

Fraction::multiplyFractions(f1, f2, res);

std::cout << "Произведение: " << res.numerator << "/" << res.denominator << std::endl;

Fraction::toDecimal(f1, dec);

std::cout << "Десятичное значение: " << dec << std::endl;

return 0;

}

Запуск кода совершается нажатием клавиши F5 (Рисунок 93).

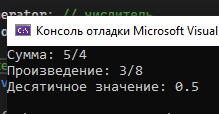


Рисунок 93 – Запуск кода

Тестирование – проверка программы на ошибки и правильность работы. Целью тестирования являются: найти ошибки до того, как программу начнут использовать, доказать, что программа работает ожидаемым образом.

Виды тестирования: модульное, интеграционное, системное, позитивное тестирование, негативное тестирование.

Модульное тестирование – это проверка всей программы целиком. Запуск у данного тестирования ручной.

Интеграционное тестирование – это проверка взаимодействия нескольких модулей между собой. Цель – убедиться, что разные части программы корректно работают вместе.

Системное тестирование – это проверка отдельных частей программы (модулей), например: одной функции, одного метода, одной структуры или класса. В МТ проверяется не вся программа целиком, а маленький фрагмент, тесты пишутся отдельно от кода, тестирование происходит автоматически.

Позитивное тестирование – это проверка работы программы с корректными входными данными. Цель ПТ – убедиться, что функция возвращает правильный результат, если все введено верно.

Негативное тестирование – проверка работы программы с некорректными входными данными. Цель НТ – убедиться, что программа правильно обрабатывает ошибки и сообщает о ней, а не падает.

И в итоге получается, что разница между ПТ и НТ вводимых данных, также в ПТ проверяется логика вычислений, ожидается правильный результат, а в НТ ожидается ошибка, проверяется как обрабатывается ошибка, не падает ли программа.

7.2. Задание 2

Обменяться проектами и написать тесты для каждой функции структуры варианта напарника 2 позитивных и 2 негативных теста.

Создаем пустой проект для кода напарника (Рисунок 94).

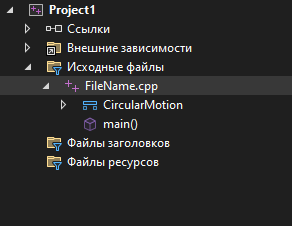


Рисунок 94 - Создание проекта

Через окно Обозреватель решений создаем проект для модульного тестирования машинного модульного теста (Рисунок 95-96).

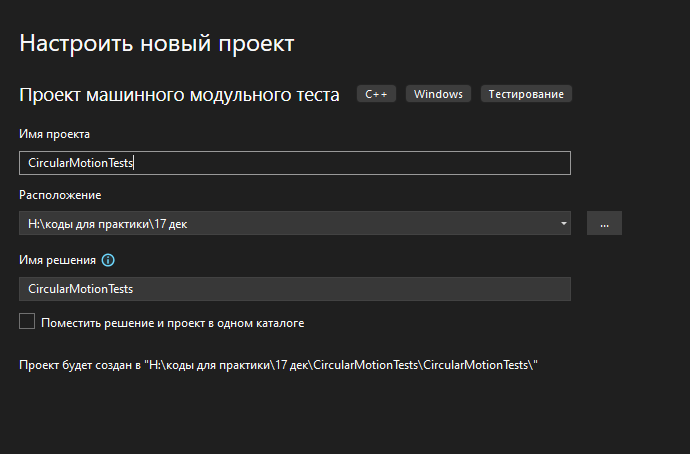


Рисунок 95 - Проект машинного модульного теста

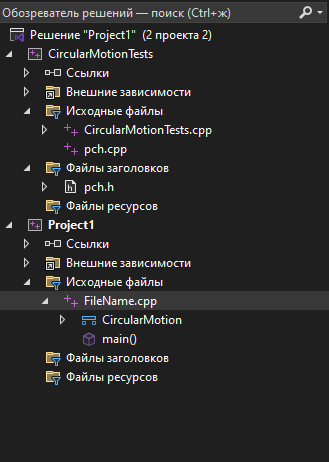


Рисунок 96 – Результат в обозревателе решений

Вставляем код напарника в Project1.

**Код напарника:**

#include <iostream>

#include <stdexcept>

using namespace std;

// Вариант 5: движение по окружности

struct CircularMotion

{

// Угловая скорость ω = v / r

static double angularVelocity(double v, double r)

{

if (r <= 0)

throw invalid\_argument("Радиус должен быть больше 0");

return v / r;

}

// Центростремительное ускорение a = v^2 / r

static double centripetalAcceleration(double v, double r)

{

if (r <= 0)

throw invalid\_argument("Радиус должен быть больше 0");

return (v \* v) / r;

}

// Частота f = 1 / T

static double frequency(double T)

{

if (T <= 0)

throw invalid\_argument("Период должен быть больше 0");

return 1.0 / T;

}

// Период T = 1 / f

static double period(double f)

{

if (f <= 0)

throw invalid\_argument("Частота должна быть больше 0");

return 1.0 / f;

}

};

int main()

{

setlocale(0, "");

try

{

double v, r;

cout << "Введите скорость v (м/с): ";

cin >> v;

cout << "Введите радиус r (м): ";

cin >> r;

cout << "Угловая скорость ω = " << CircularMotion::angularVelocity(v, r) << " рад/с" << endl;

cout << "Центростремительное ускорение a = " << CircularMotion::centripetalAcceleration(v, r) << " м/с^2" << endl;

double T;

cout << "\nВведите период T (с): ";

cin >> T;

cout << "Частота f = " << CircularMotion::frequency(T) << " 1/с" << endl;

double f;

cout << "\nВведите частоту f (1/с): ";

cin >> f;

cout << "Период T = " << CircularMotion::period(f) << " с" << endl;

}

catch (const exception& ex)

{

cout << "Ошибка: " << ex.what() << endl;

}

system("pause");

return 0;

}

Запускаем код напарника нажатием клавиши F5 (Рисунок 97).

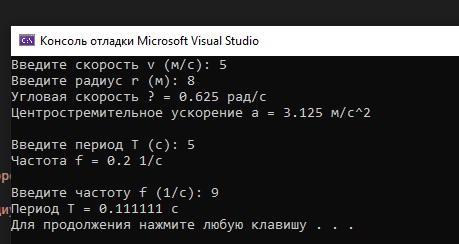


Рисунок 97 - Запуск кода до проверки

Вставляем код программы для тестирования кода напарника в CircularMotionTests.

**Код тестирования:**

#include "pch.h"

#include "CppUnitTest.h"

#include <stdexcept>

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

struct CircularMotion

{

static double angularVelocity(double v, double r)

{

if (r <= 0)

throw std::invalid\_argument("Радиус должен быть больше 0");

return v / r;

}

static double centripetalAcceleration(double v, double r)

{

if (r <= 0)

throw std::invalid\_argument("Радиус должен быть больше 0");

return (v \* v) / r;

}

static double frequency(double T)

{

if (T <= 0)

throw std::invalid\_argument("Период должен быть больше 0");

return 1.0 / T;

}

static double period(double f)

{

if (f <= 0)

throw std::invalid\_argument("Частота должна быть больше 0");

return 1.0 / f;

}

};

namespace CircularMotionUnitTests

{

TEST\_CLASS(CircularMotionTests)

{

public:

// позитивный тест 1

TEST\_METHOD(AngularVelocity\_ValidData)

{

double result = CircularMotion::angularVelocity(10.0, 2.0);

Assert::AreEqual(5.0, result, 0.0001);

}

// позитивный тест 2

TEST\_METHOD(Frequency\_ValidData)

{

double result = CircularMotion::frequency(2.0);

Assert::AreEqual(0.5, result, 0.0001);

}

// негативный тест 1

TEST\_METHOD(AngularVelocity\_InvalidRadius)

{

Assert::ExpectException<std::invalid\_argument>([] {

CircularMotion::angularVelocity(10.0, 0);

});

}

// негативный тест 2

TEST\_METHOD(Period\_InvalidFrequency)

{

Assert::ExpectException<std::invalid\_argument>([] {

CircularMotion::period(-1.0);

});

}

};

}

Нажимаем на Тесты, выбираем Обозреватель тестов, запустить все тесты (Рисунок 98-99).

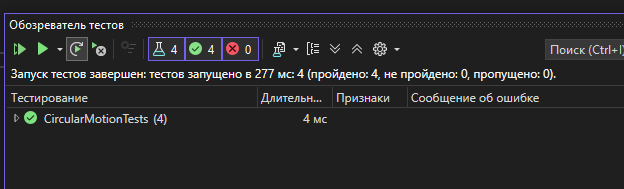


Рисунок 98 - Проверка кода друга

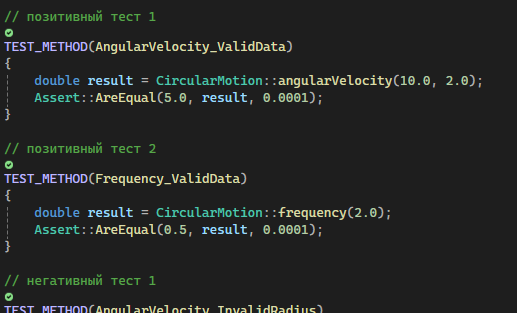


Рисунок 99 – Создание модульного теста

8. 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.

**Проект: Сайт музыкального исполнителя.**

8.1. Часть 1: Разработка Технического Задания (ТЗ)

Описание проекта от клиента:

«Креативный сайт с музыкой, видео, афишей концертов и формой для бронирования выступлений.»

Страницы: Главная, Биография, Музыка (аудиоплеер), Видео (галерея), Концерты (расписание).

JS-элементы: Аудиоплеер, слайдер с фото группы, счётчик до следующего концерта.

8.1.1. Цели и описание проекта

Ключевая цель: Продвижение творчества музыкального исполнителя, публикация музыкального и видеоконтента, информирование о концертах и прием заявок на выступления.

Целевая аудитория: Поклонники исполнителя, молодая аудитория 14-30 лет, организаторы концертов и мероприятий.

Описание проекта: Разрабатывается статический адаптивный сайт музыкального исполнителя Вани Дмитриенко. Сайт использует базу данных и реализуется с применением HTML, CSS и JavaScript.

8.1.2. Функциональные требования

Главная страница: имя исполнителя, герой-блок, слайдер фотографий, анонс ближайшего концерта.

Биография: информация о творческом пути и музыкальном стиле.

Музыка: список треков с встроенным аудиоплеером.

Видео: галерея видеоклипов со встроенным плеером.

Концерты: расписание выступлений (дата, город, площадка).

Контакты: форма обратной связи для бронирования выступлений.

8.1.3. Технологии

Фронтенд: HTML5 – структура страниц сайта, CSS3 – стилизация и адаптивный дизайн, JavaScript – интерактивные элементы (аудиоплеер).

Хостинг: Статический хостинг (GitHub Pages или Netlify).

Интеграции:

Аудиоплеер: HTML5 <audio> для воспроизведения музыкальных треков.

Видео: встроенный видеоплеер YouTube или RUTUBE.

Форма обратной связи: интеграция с внешним сервисом (Formspree или EmailJS)

Социальные сети: иконки со ссылками на официальные страницы исполнителя.

8.1.4. Нефункциональные требования

Производительность:

* 1. Время загрузки главной страницы – не более 3 секунд при скорости интернета 10Мбит/с.
  2. Интерактивные элементы (слайдер, плеер) должны реагировать без заметных задержек.

Адаптивность: Корректное отображение сайта на устройствах с шириной экрана от 320px до 1920 px.

Кроссбраузерность: Сайт должен одинаково работать в последних версиях Chrome, Firefox, Safari и Edge.

Удобство использования (Usability):

* 1. Интуитивно понятная навигация.
  2. Визуальный отклик кнопок и ссылок при наведении.
  3. Читаемый шрифт и контрастный текст.

Поддержка и масшабируемость:

* 1. Структурированный и прокомментированный код.
  2. Возможность добавления новых треков, видео и концертов без изменения структуры сайта.

8.1.5. Критерии тестирования

Проверка отображения сайта на расширениях: 320px, 768px, 1024px, 1920px.

Проверка корректной работы аудиоплеера.

Проверка открытия и воспроизведения видео.

Проверка отправки формы обратной связи.

Проверка кликабельности свех пунктов меню и ссылок.

8.1.6. Сроки поэтапной сдачи

Этап 1 (1 неделя): разработка и согласование технического задания.

Этап 2 (2 неделя): создание черно-белых макетов страниц (wireframes).

Этап 3 (3 неделя): разработка цветных макетов (mpckups).

Этап 4 (4 неделя): верстка сайта, подключение интерактивных элементов и финальное тестирование.

8.2. Часть 2: Разработка Макетов

8.2.1. Черно-белые макеты (Wireframes)

Главная страница:

1. Шапка с логотипом «Ваня Дмитриенко» в меню.
2. Герой-блок с именем исполнителя.
3. Оповещение о концерте (туре).

Отсчет: отсчет времени до ближайшего концерта.

Концерты(тур):

1. Список концертов (дата, город, площадка).
2. Большой город визуально выделен.

Социальные сети:

1. Заголовок.
2. Ссылки на официальные соц. сети.

Мерч:

1. Заголовок.
2. Поля для заполнения контактной информации.

Аудиоплеер(карусель):

1. Картинка.
2. Название.
3. Исполнитель.
4. Год выпуска.

Галерея: Карусель изображений исполнителя.

Видео: Карусель видеоклипов.

Обратная связь:

1. Концерты и реклама.
2. PR/PROMO.

Макеты расположены в Приложении А.

8.2.2. Цветные макеты (Mockups)

Цветовая палитра: белый, черный, акцентный розовый.

Шрифты: современные sans-serif.

Главная: крупная фотография исполнителя, акцент на ближайшем концерте.

Отсчет: циферблат (дни, часы, минуты).

Соц. сети: ссылки на официальные социальные сети исполнителя.

Музыка: карусель карточки треков.

Видео: карусель клипов.

Макеты расположены в Приложении Б.

ПРИЛОЖЕНИЕ А

**Wireframes макеты.**

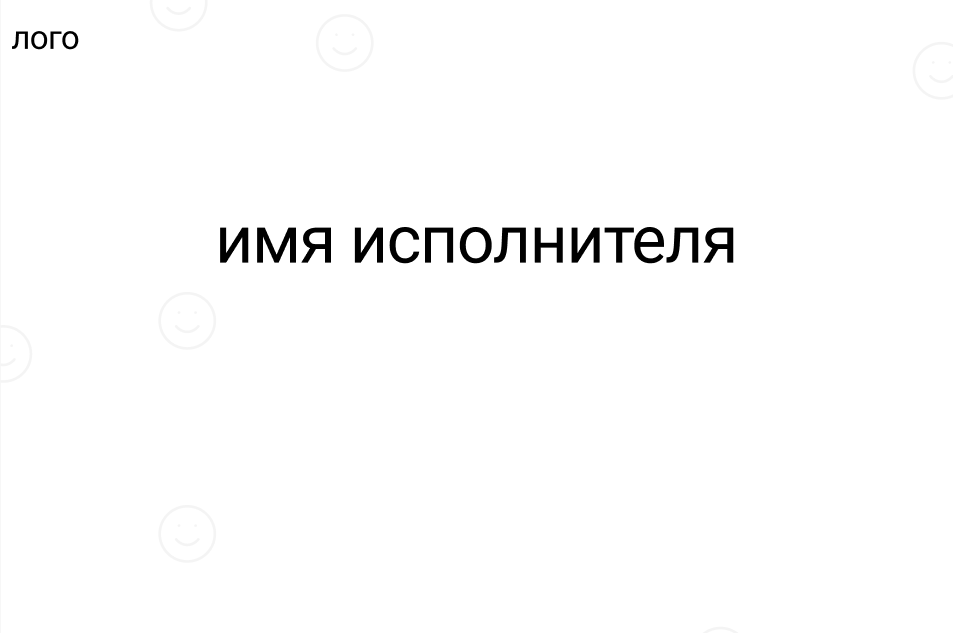


Рисунок А1 - Wireframe Главная страница

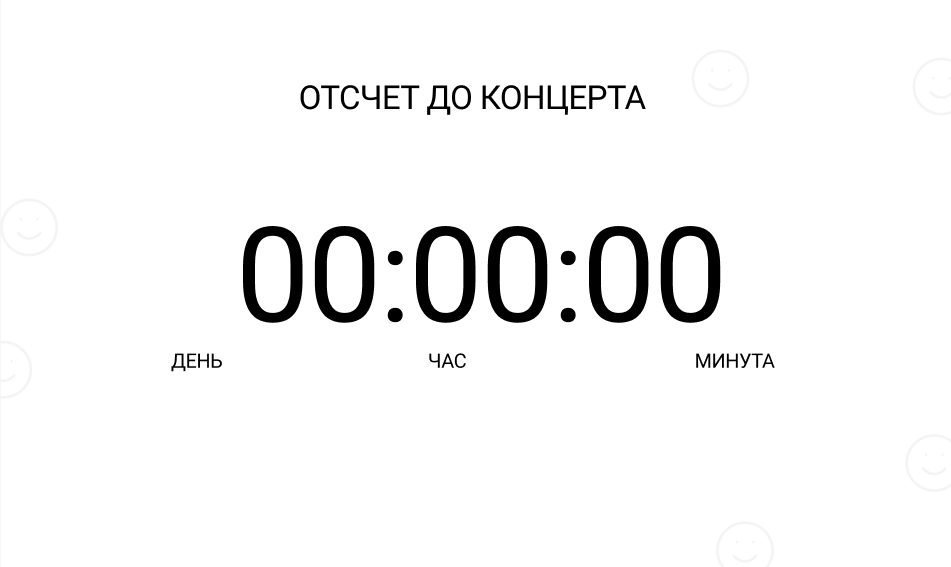


Рисунок А2 - Wireframe Счетчик до ближайшего концерта



Рисунок А3 - Wireframe Расписание концертов

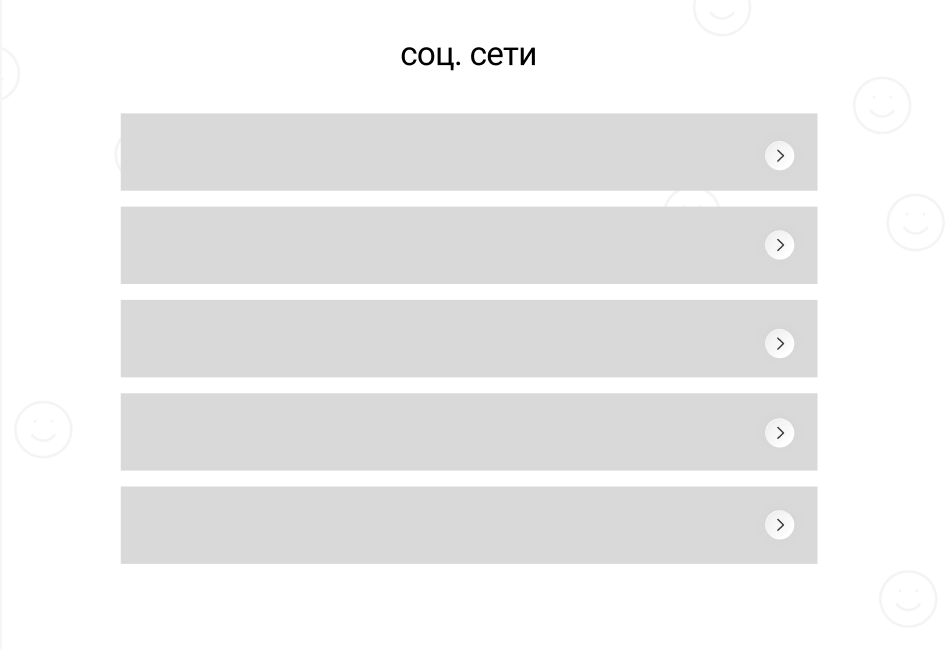


Рисунок А4 - Wireframe Официальные соц. сети

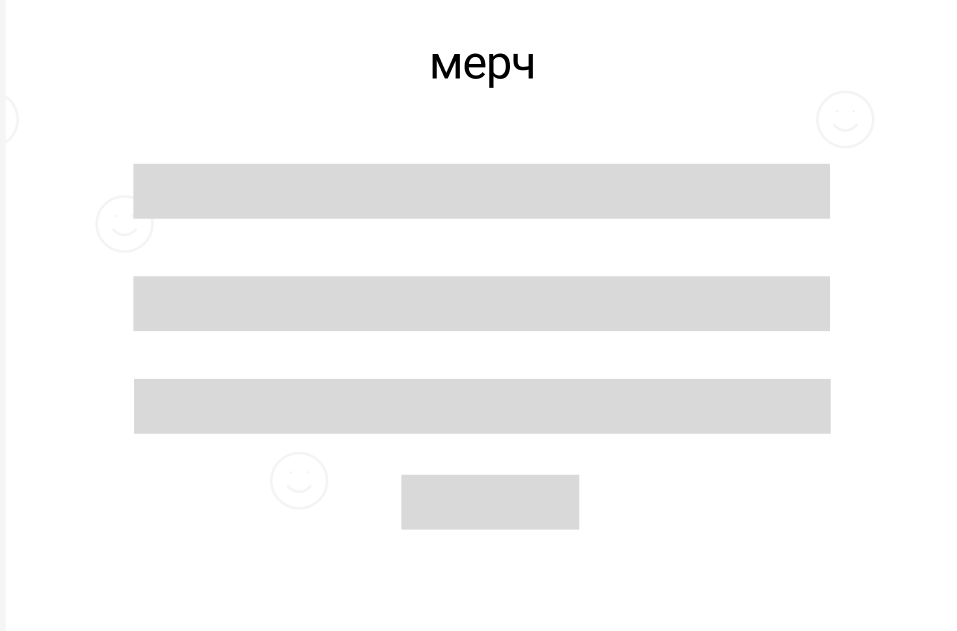


Рисунок А5 - Wireframe Мерч

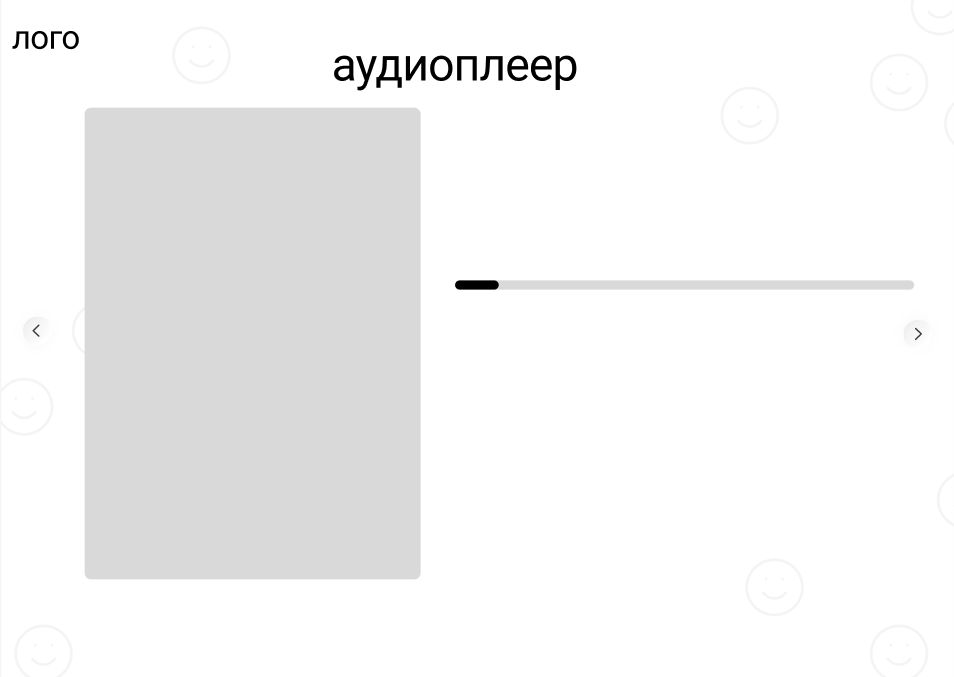


Рисунок А6 - Wireframe Карусель из песен

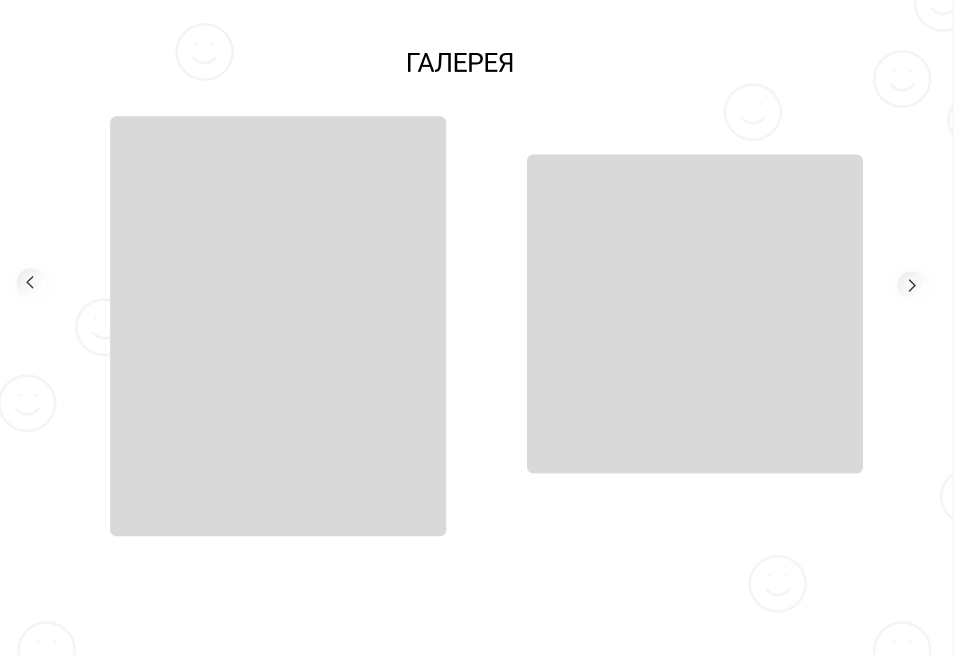


Рисунок А7 - Wireframe Галерея

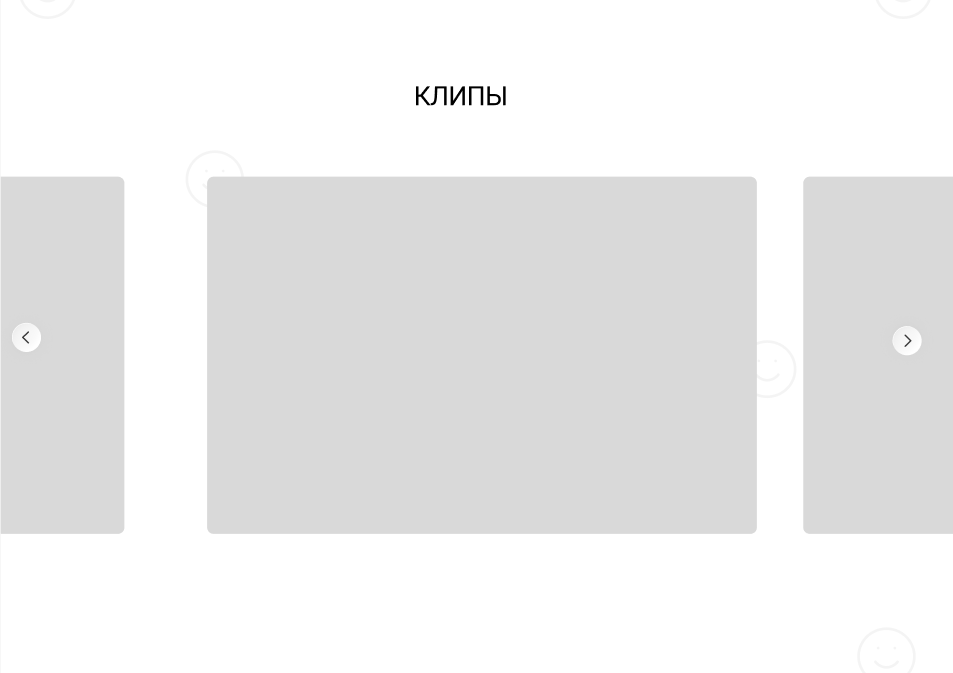


Рисунок А8 - Wireframe Карусель из клипов

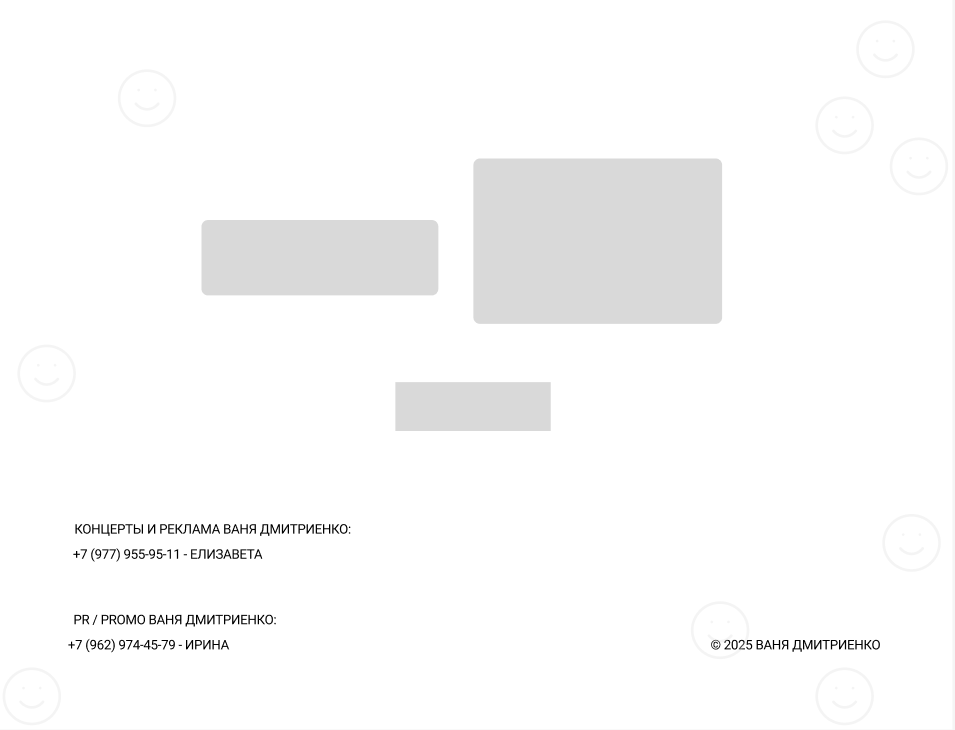


Рисунок А9 - Wireframe Контакты

ПРИЛОЖЕНИЕ Б

**Цветные макеты.**



Рисунок Б1 - Главная страница

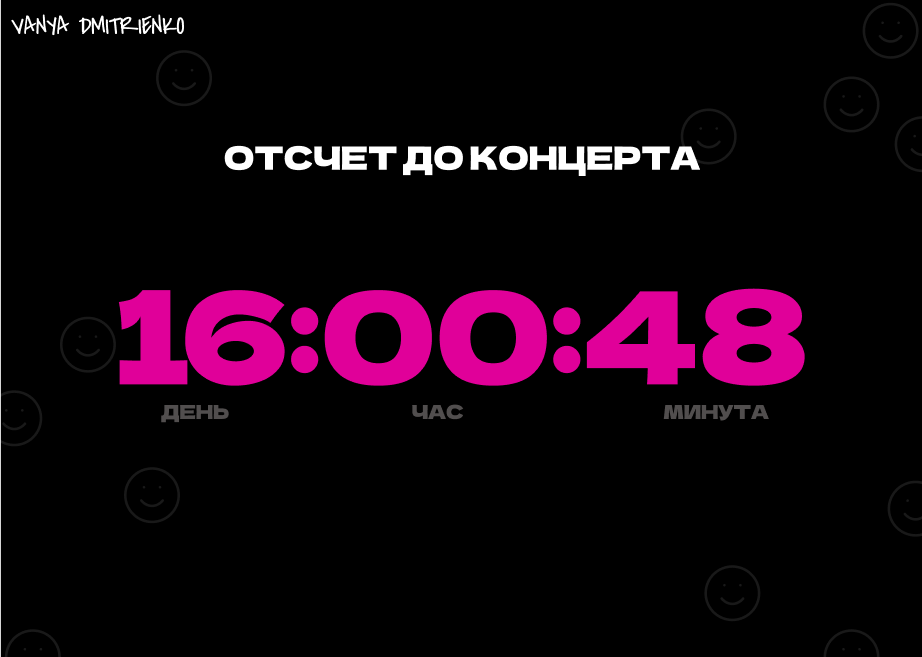


Рисунок Б2 - Счетчик до ближайшего концерта



Рисунок Б3 - Расписание концертов

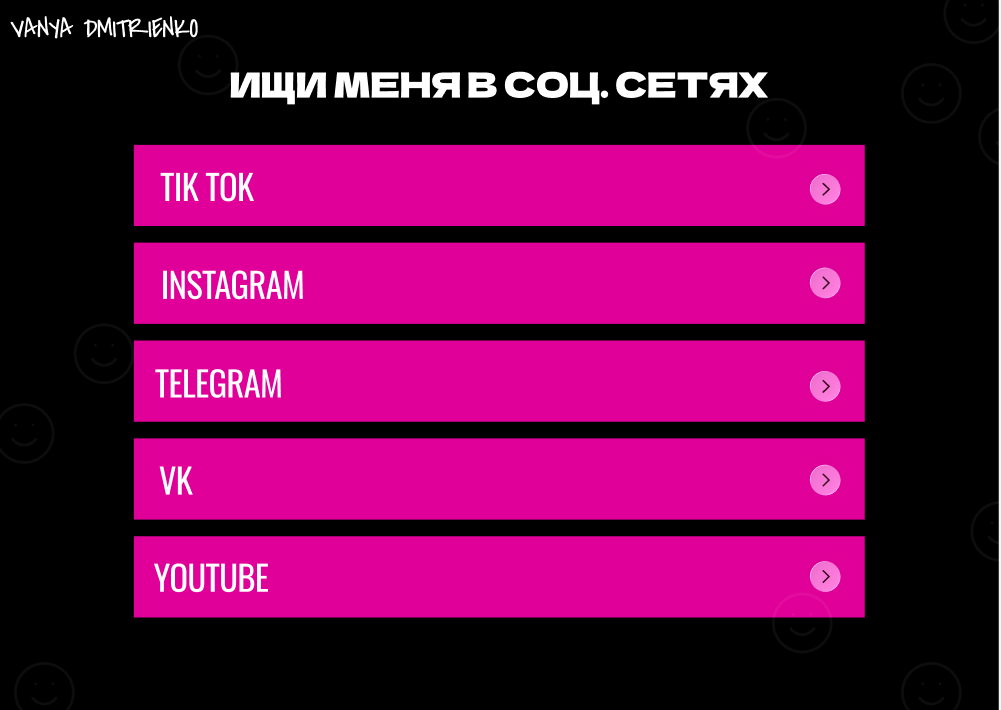


Рисунок Б4 - Официальные соц. сети

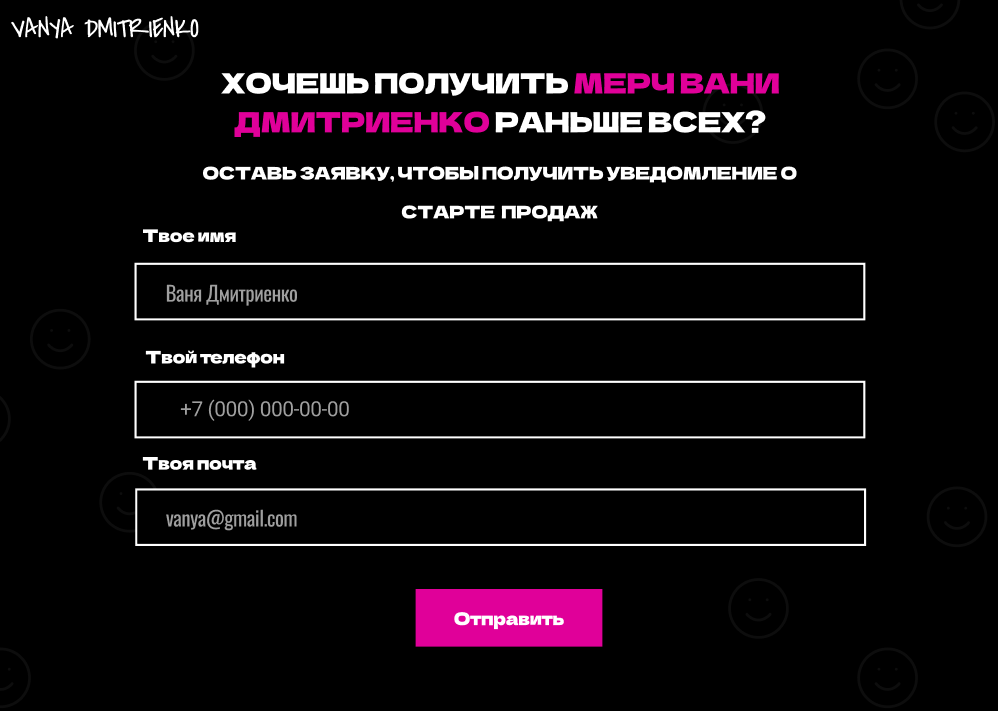


Рисунок Б5 - Мерч



Рисунок Б6 - Карусель песен 1

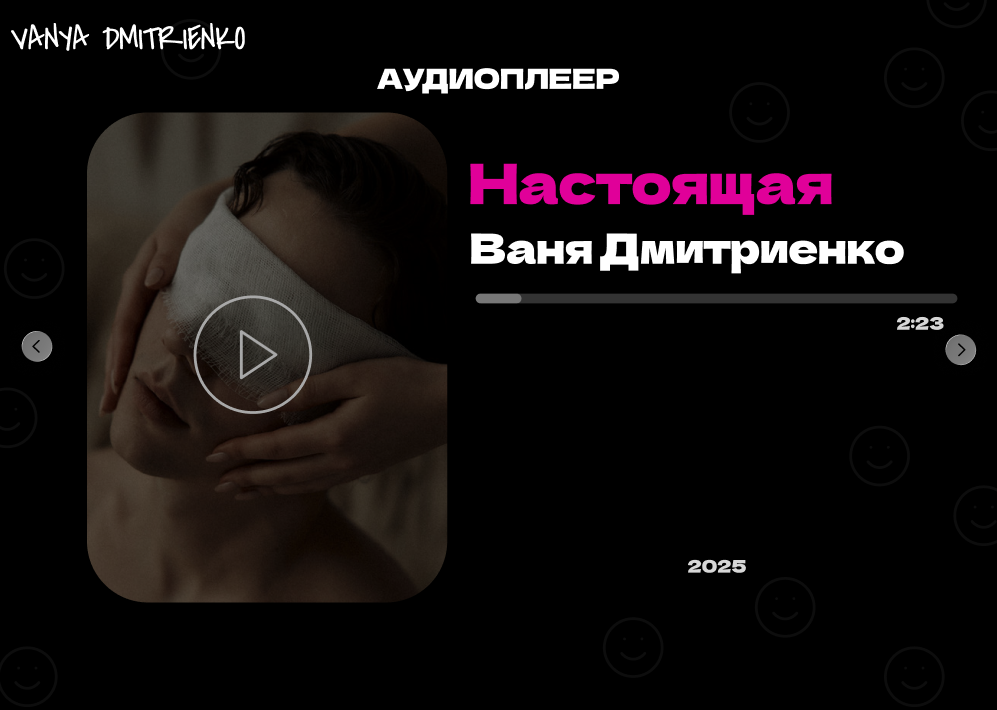


Рисунок Б7 - Карусель песен 2



Рисунок Б8 - Галерея

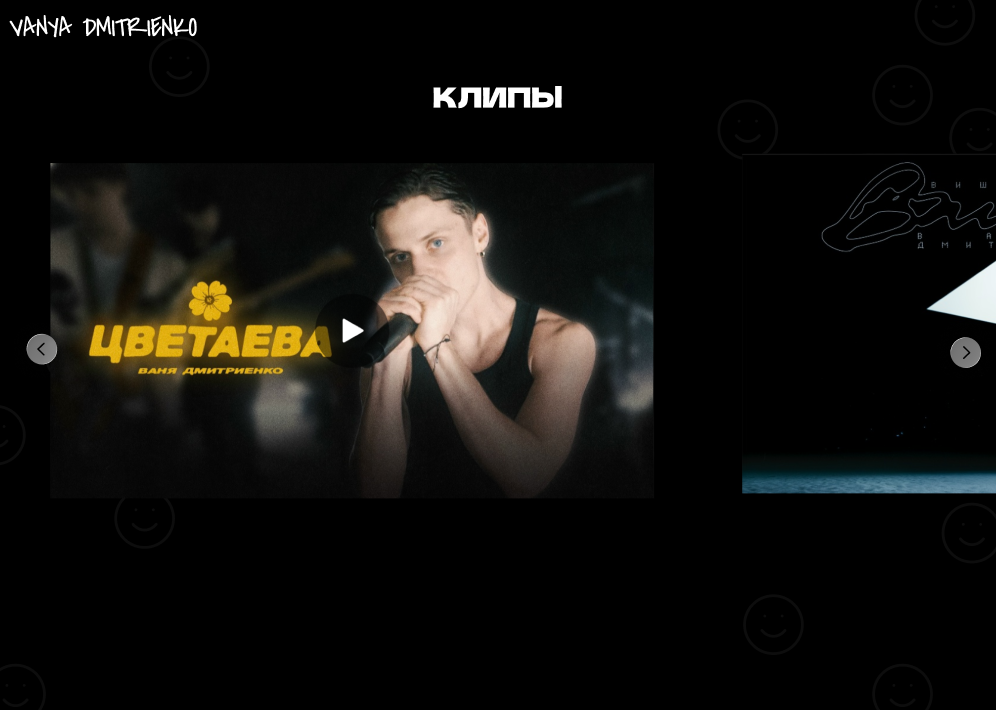


Рисунок Б9 - Карусель клипов



Рисунок Б10 - Контакты

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

**9.1. Задание 1.**

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](https://learngitbranching.js.org/?locale=ru_RU).

9.1.1. Решение уровней раздела «Введение»

Прошла 1 уровень раздела «Введение» (Рисунок 100).

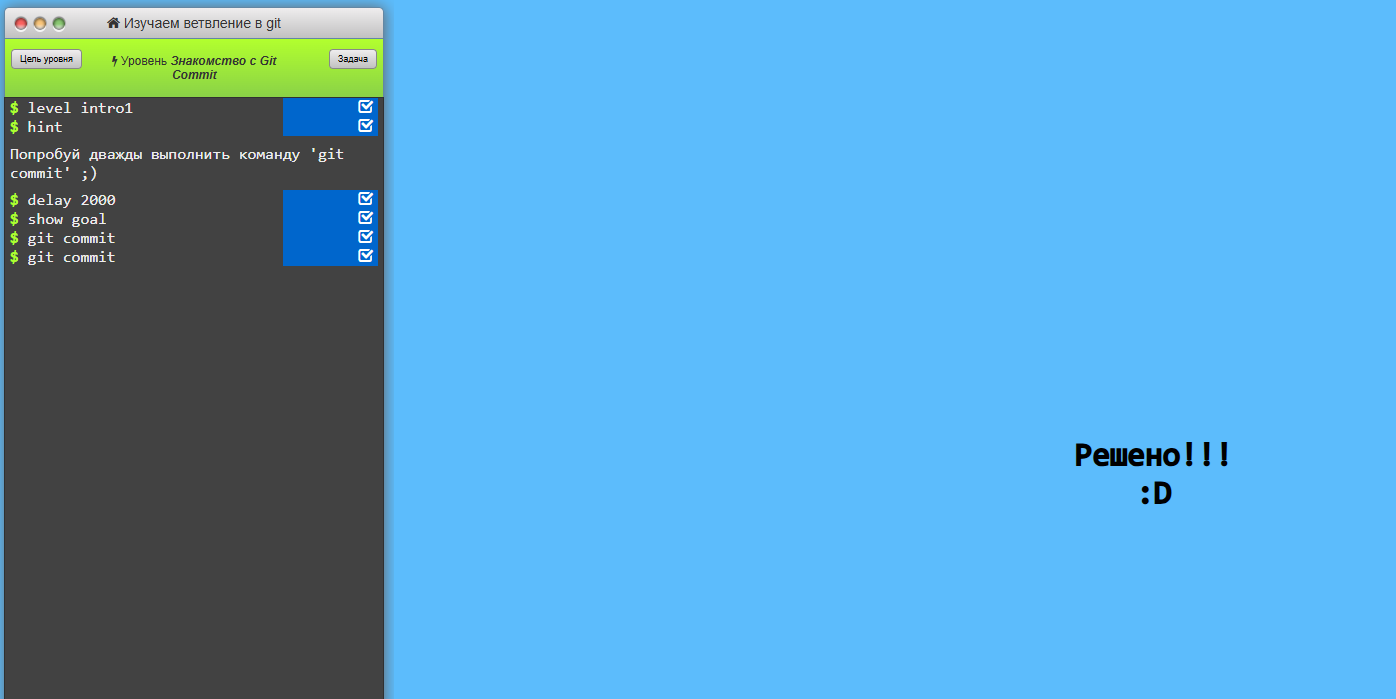


Рисунок 100 - Решение 1-го уровня

Используемые команды git commit, git commit (создание двух коммитов).

Прошла 2 уровень из раздела «Введение» (Рисунок 101).

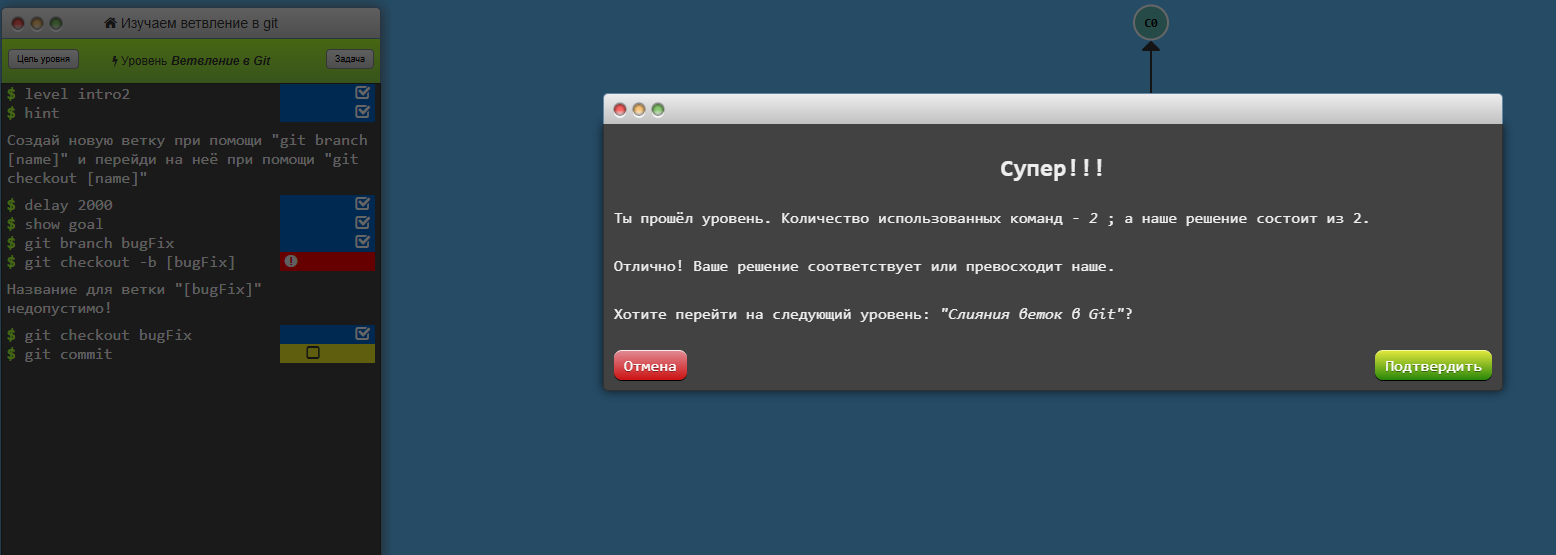


Рисунок 101 - Решение 2-го уровня

Используемые команды: git branch bugFix, git checkout bugFix (создала и переключилась на новую ветку bugFix).

Прошла 3 уровень из раздела «Введение» (Рисунок 102).

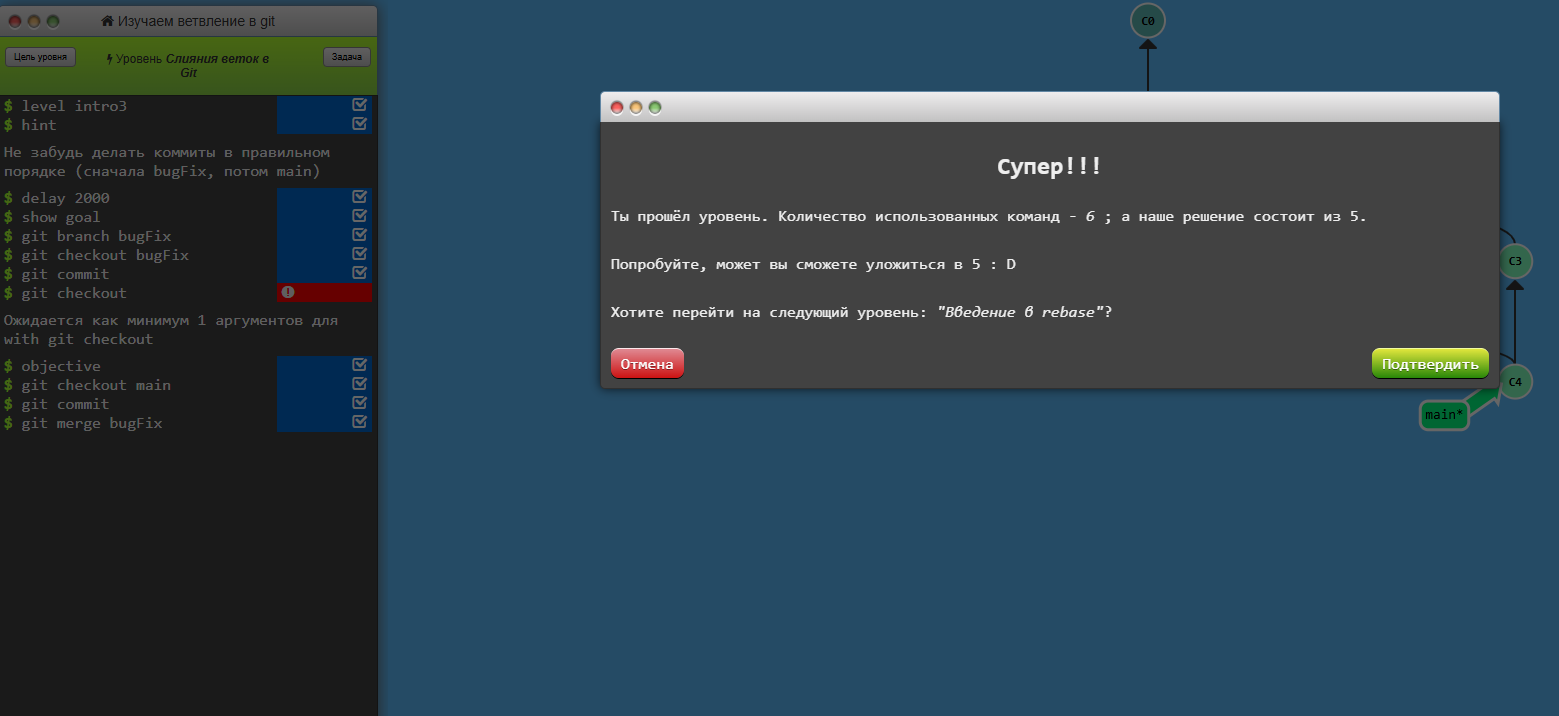


Рисунок 102 - Решение 3-го уровня

Используемые команды: git commit; git checkout main; git commit; git merge bugfix (Объединила изменения из двух разных веток. Использовала команду git merge).

Прошла 4 уровень из раздела «Введение» (Рисунок 103).

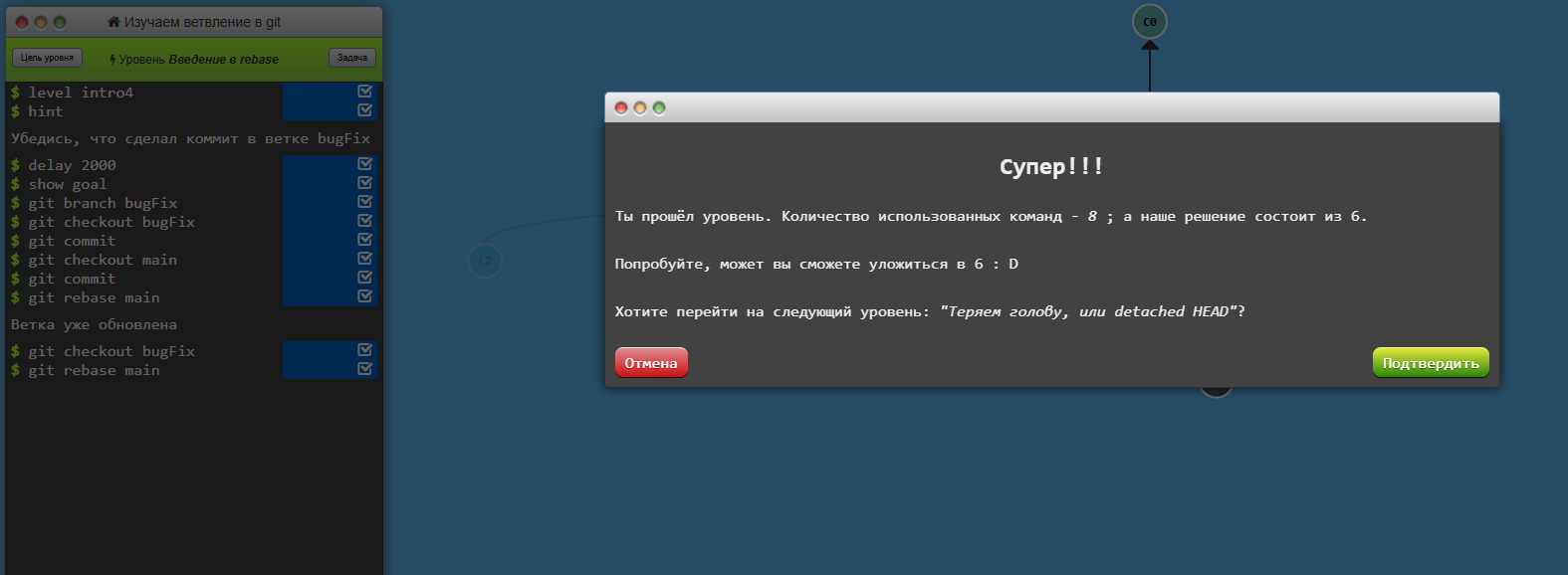


Рисунок 103 - Решение 4-го уровня

Используемые команды: git commit; git checkout main; git commit; git checkout bugfix; git rebase main (Второй способ объединения изменений в двух ветках. При «rebase» Git копирует набор коммитов и переносит их в другое место).

9.1.2. Раздел «Едем дальше»

Прошла 1 уровень из раздела «Едем дальше» (Рисунок 104).



Рисунок 104 - Решение 1-го уровня

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

Прошла 2 уровень из раздела «Едем дальше» (Рисунок 105).

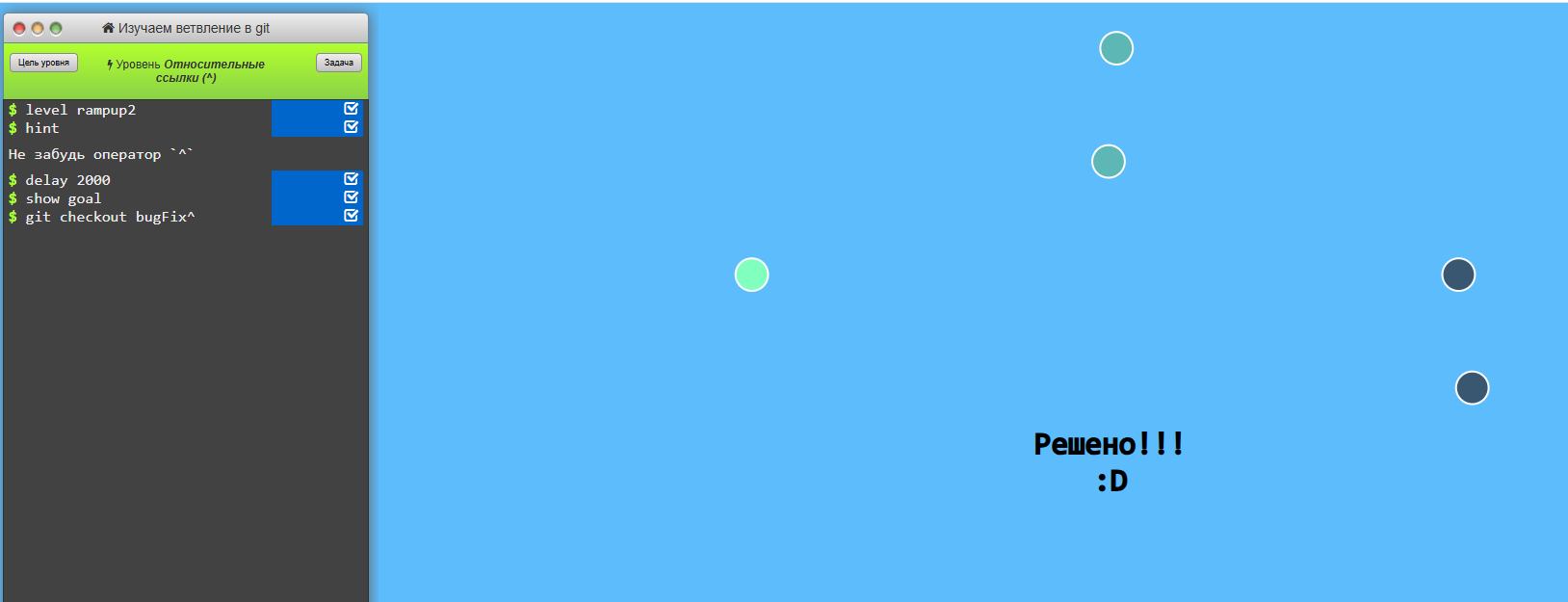


Рисунок 105 - Решение 2-го уровня

Используемые команды: git checkout HEAD^ (Переместила на первого родителя ветки).

Прошла 3 уровень из раздела «Едем дальше» (Рисунок 106).

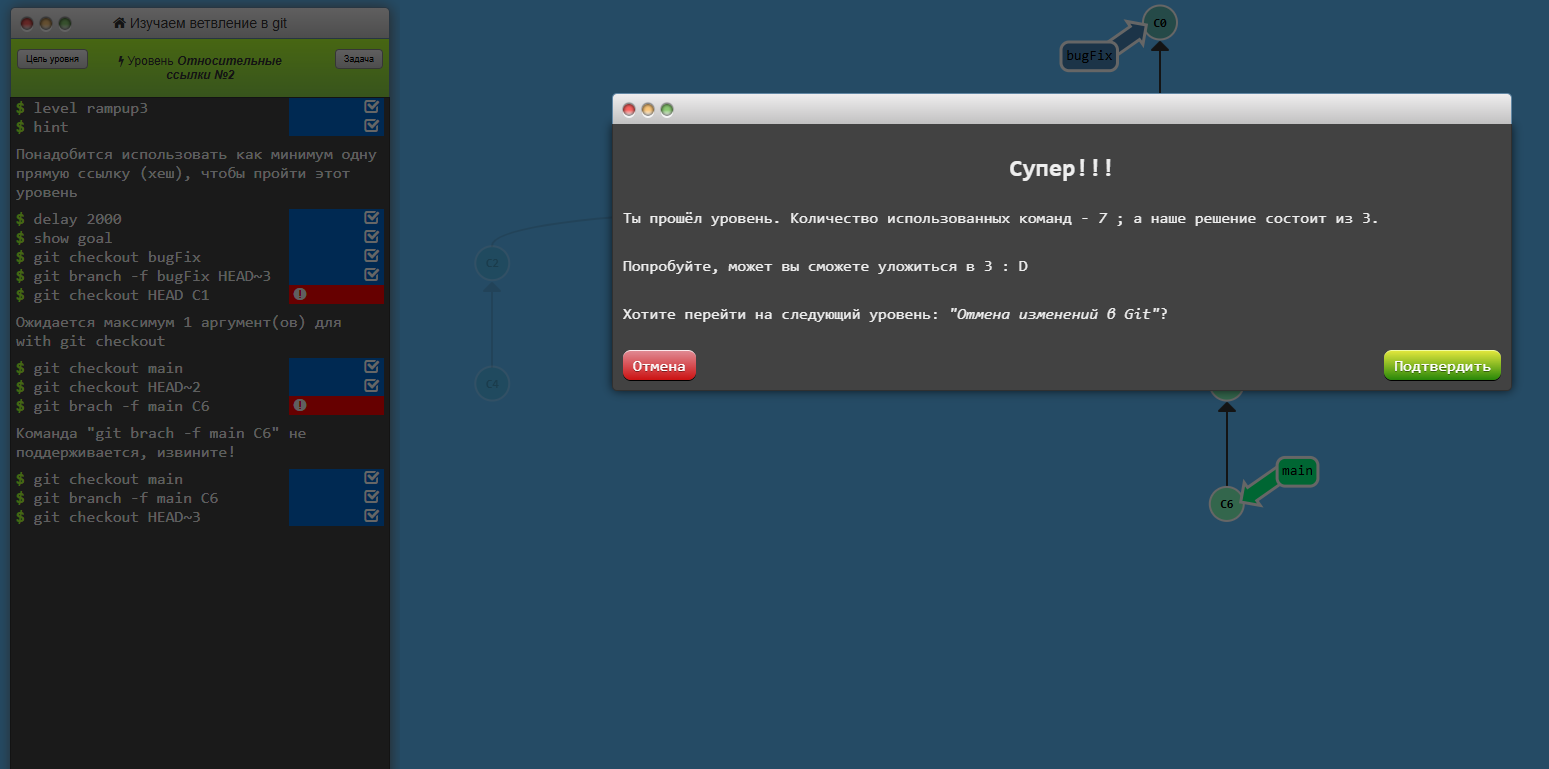


Рисунок 106 - Решение 3-го уровня

Используемые команды: git branch -f main C6; git checkout HEAD~1; git branch -f bugfix HEAD~1 (Создала ветку main с именем C6. Перешла к предыдущему коммиту, создала ветку bugFix с именем, соответствующим предыдущей ветке HEAD~1).

Прошла 4 уровень из раздела «Едем дальше» (Рисунок 107).

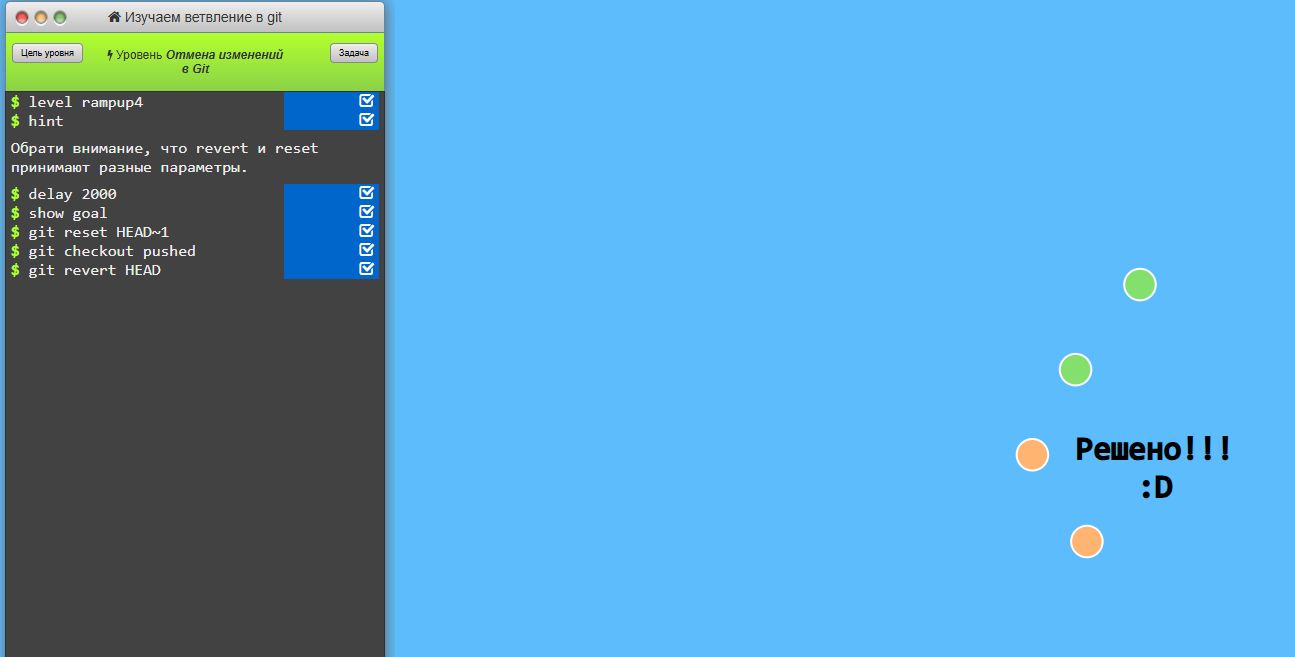


Рисунок 107 - Решение 4-го уровня

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).

Вкладка «Основы» завершена (по заданию) (Рисунок 108).

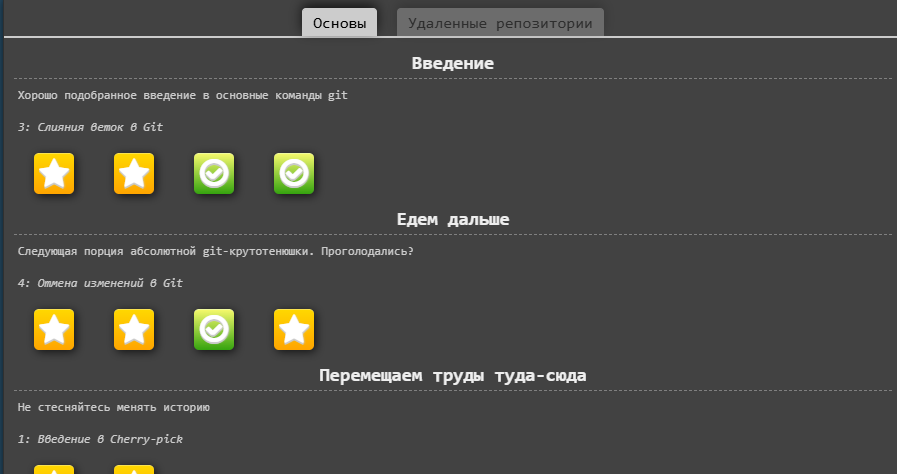


Рисунок 108 - Вкладка основы

9.1.3. Раздел «Push&Pull» на вкладке «Удаленные репозитории»

Прошла 1 уровень из раздела «Push&Pull» (Рисунок 109).

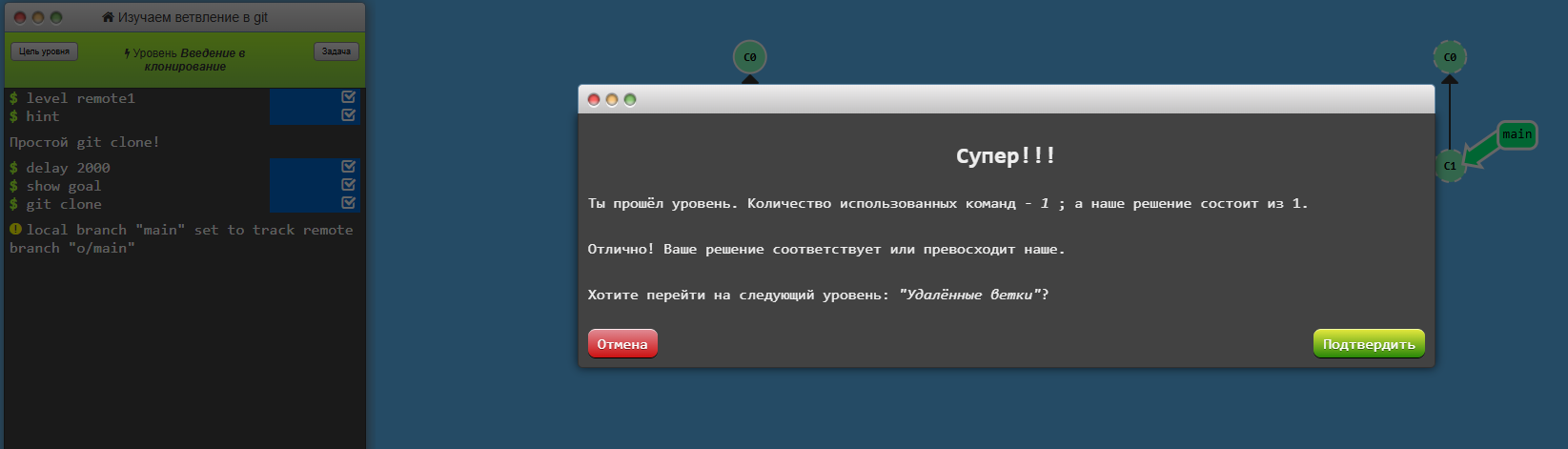


Рисунок 109 - Решение 1-го уровня

Используемые команды: git clone (Создала клон репозитория).

Прошла 2 уровень из раздела «Push&Pull» (Рисунок 110).

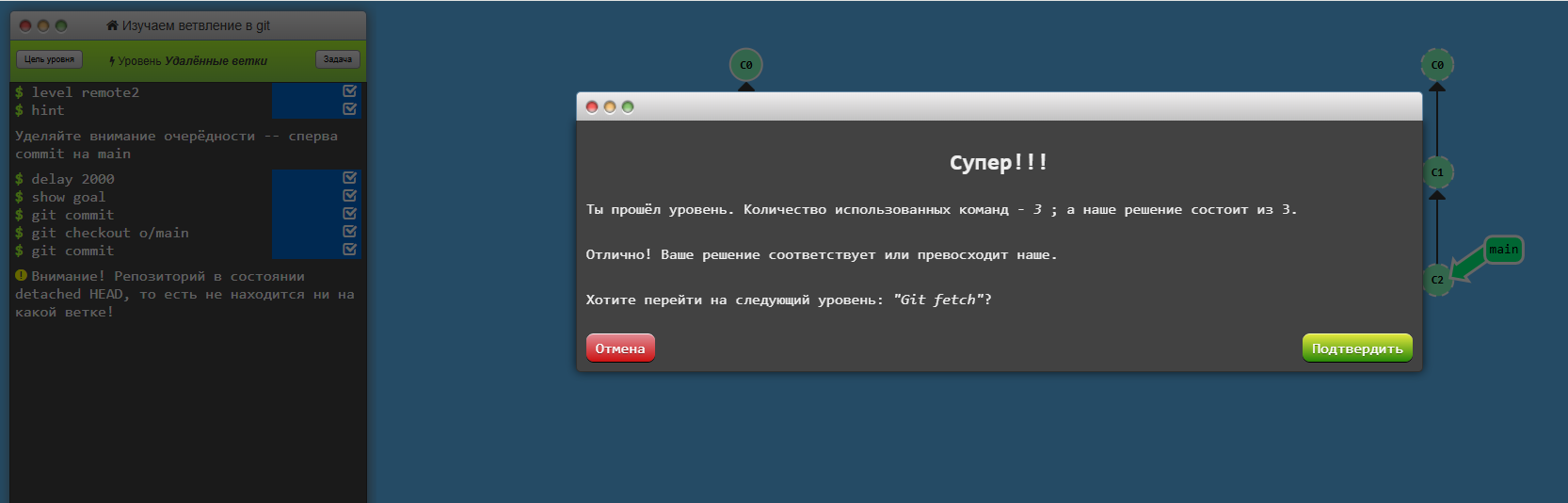


Рисунок 110 - Решение 2-го уровня

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

Прошла 3 уровень из раздела «Push&Pull» (Рисунок 111).

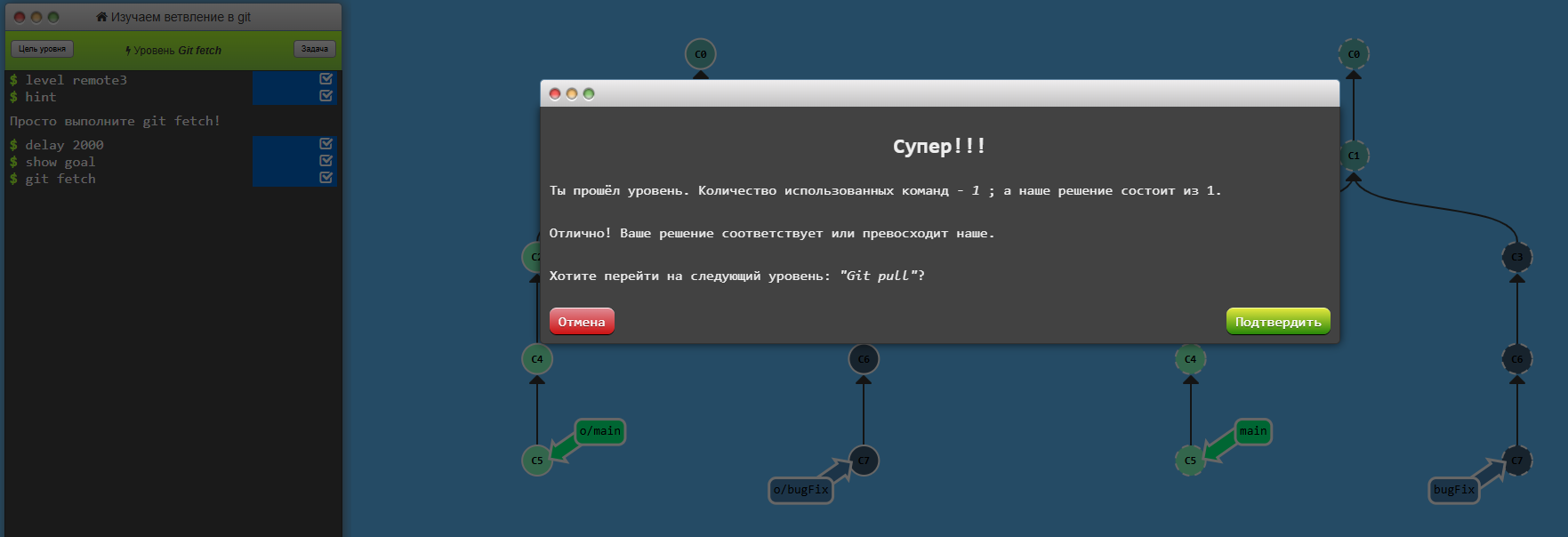


Рисунок 111 - Решение 3-го уровня

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

Прошла 4 уровень из раздела «Push&Pull» (Рисунок 112).

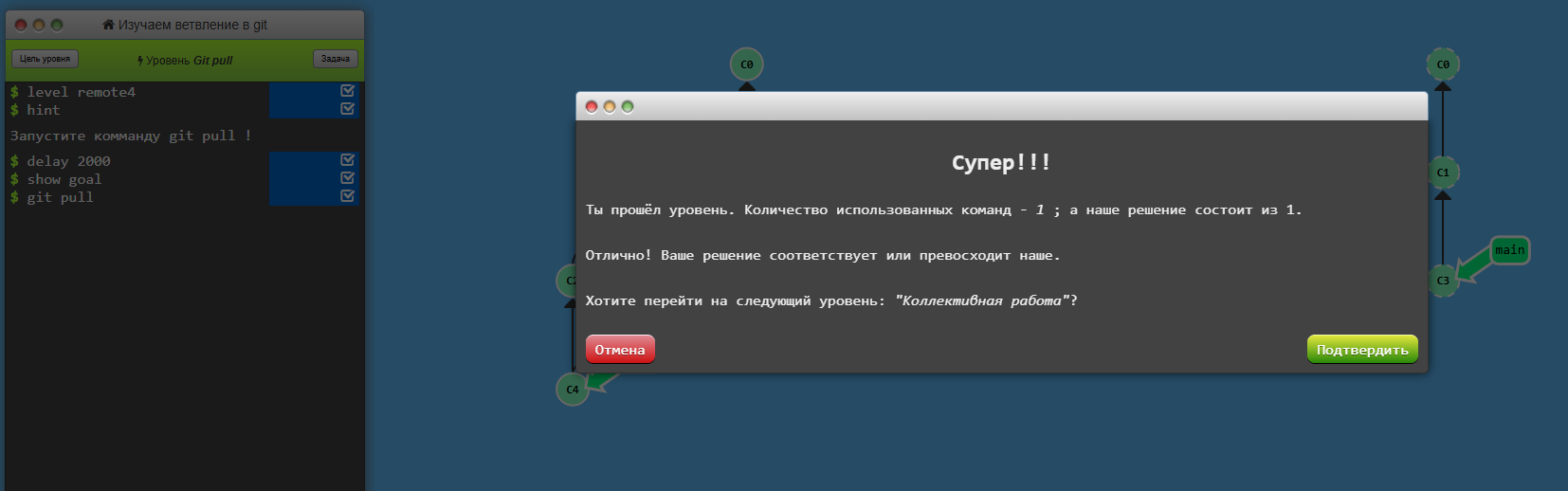


Рисунок 112 - Решение 4-го уровня

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

**АТТЕСТАЦИОННЫЙ ЛИСТ**

**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

Патласова Елизавета Витальевна

(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 «8» декабря 2025 г. по «27» декабря 2025 г.

**Виды и качество выполнения работ**

|  |  |  |
| --- | --- | --- |
| **Виды выполненных работ обучающимся**  **во время практики** | **Объем работ, час.** | **Качество выполнения работ (оценка по пятибалльной системе)** |
| **Тема 1** Персонализация интегрированной среды разработки Visual Studio Community 2022 | 18 |  |
| **Тема 2** Отладка в IDE Visual Studio Community 2022 | 30 |  |
| **Тема 3** Обеспечение качества кода | 36 |  |
| **Тема 4** Упаковка приложения | 24 |  |

Характеристика учебной/профессиональной деятельности обучающегося во время учебной практики (по профилю специальности):

Общие и профессиональные компетенции, предусмотренные программой практики, *освоены* ***/*** *не освоены*.

(нужное подчеркнуть)

Итоговая оценка по практике \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель практики     Васильев С.Ю. \_\_\_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О.) (подпись)