Министерство образования Республики Беларусь

Учреждение образования
«Полоцкий государственный университет имени Евфросинии Полоцкой»

Р. П. Богуш, Д. М. Васильева

ОСНОВЫ КОНСТРУИРОВАНИЯ ПРОГРАММ

Методические указания к лабораторным работам
для студентов специальности 6-05-0611-01
«Информационные системы и технологии»

Текстовое электронное издание

Новополоцк
Полоцкий государственный университет
имени Евфросинии Полоцкой
2024

Об издании – 1, 2

1 – дополнительный титульный экран – сведения об издании

УДК 004.415

Рекомендовано к изданию
методической комиссией факультета информационных технологий
(протокол № 5 от 29.05.2024 г.)

2 – дополнительный титульный экран – производственно-технические сведения

Для создания текстового электронного издания «Основы конструирования программ. Методические указания к лабораторным работам для студентов специальности 6-05-0611-01 «Информационные системы и технологии» Р. П. Богуша, Д. М. Васильевой использованы текстовый процессор Microsoft Office Word и программа Adobe Acrobat XI Pro для создания и просмотра электронных публикаций в формате PDF.

Редактор С. Е. Рясова

Подписано к использованию 02.09.2024.
Объем издания: 3,2 Мб. Заказ 285.

Свидетельство о государственной регистрации
издателя, изготовителя, распространителя печатных изданий
№ 1/305 от 22.04.2014.

211440, Ул. Блохина, 29,
г. Новополоцк,
Тел. 8 (0214) 59-95-41, 59-95-44
http://www.psu.by

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА № 1
Создание и отладка программного проекта на языке С++

Цель: получить навыки создания программного проекта на языке С++, компиляции и отладки кода с использованием интегрированной среды разработки Microsoft Visual Studio и онлайн-компилятора.

Теоретический материал

I. Типичная среда программирования С++

Для выполнения программы на компьютере необходимо пройти следующие этапы: редактирование, предварительную (препроцессорную) обработку, компиляцию, компоновку, загрузку и выполнение.

Этап 1 – *Редактор* (программист): программа создается редактором и запоминается на диске.

Этап 2 – *Препроцессор:* программа предварительной обработки, подчиняется специальным командам (директивы препроцессора) для преобразования кода. Они отмечаются знаком решетка #. Например, включение других текстовых файлов в файл, подлежащий компиляции, выполнение различных текстовых замен.

Препроцессорная обработка инициируется компилятором перед преобразованием программы в машинный код.

Результат работы препроцессора – это последовательность лексем, которую называют единицей трансляции.

Существуют лексемы пяти видов: *идентификаторы, служебные слова, литералы, операции, различные разделители*.

Этап 3 – *Компилятор*: переводит программу в машинный (объектный) код и сохраняет ее на диске.

Этап 4 – *Компоновщик.* Программы на С++ содержат ссылки на функции, определенные вне самой программы, например, в стандартных библиотеках или в библиотеках проекта.

Объектный код, созданный компилятором, обычно содержит «дыры» из-за этих отсутствующих частей.

Компоновщик связывает объектный код с кодами отсутствующих функций, чтобы создать исполняемый загрузочный модуль (без пропущенных частей).

Этап 5 – *Загрузчик*: перед выполнением программа должна быть размещена в памяти. Это делается с помощью загрузчика, который забирает исполняемый загрузочный модуль с диска и перемещает его в память.

Этап 6 – компьютер выполняет поочередно в каждый момент времени по одной команде программы.

Идентификатор – это имя программного объекта из букв и цифр, причем длина его по стандарту не ограничена. При его создании в C++ следует учитывать правила:

– идентификатор может включать буквы латинского алфавита, цифры или символов подчёркивания;

– первый символ идентификатора должен начинаться с буквы, т.е. нельзя использовать первой цифру;

– возможно, но не рекомендуется первым использовать символ подчеркивания, чтобы исключить возможность совпадения с именами системных функций или переменных;

– использование совпадающих строчных и прописных букв приводит к созданию разных идентификаторов. Например, три разных идентификатора: NUMBER, number, Number;

– идентификатор должен быть отличен от ключевых слов, зарезервированных в C++ слов, а также от имен функций библиотек в C++.

Программа на языке C++ представляет собой *совокупность модулей*, (может быть один модуль), которые представляет самостоятельно транслируемые файлы, включающий одну функцию или множество функций. Функция состоит из *операторов (statement, стейтмент)*, представляющих собой законченные предложения на языке C++, которые указывают компьютеру выполнять определенные действия. Все операторы в программе на языке C++ должны заканчиваться точкой с запятой.

Основной модуль на языке C++ должен включать главную (обязательную) функцию, имеющую название `main`. В круглых скобках `main()` перечисляются аргументы или параметры функции (в данном случае аргументов нет). В фигурных скобках (`{…}`) записывается тело функции, т.е. все действия, которые она выполняет. Каждое действие в языке C++ заканчивается символом «точка с запятой» (`;`). У функции может быть результат или возвращаемое значение, в таком случае перед ее названием указывается возвращаемый тип данных (`int`, `float`, `double`, `char`, `bool`). Если функция не возвращает никакого значения, то она обозначается ключевым словом `void` [1].

С точки зрения архитектуры ЭВМ (электронная вычислительная машина – это вычислительное устройство дискретного действия, где вычисления производятся в цифровой форме [1; 2]), переменная – это символическое обозначение ячейки оперативной памяти, в которой хранятся данные для использования при выполнении программы. Доступ к значению осуществляется через имя переменной, состоящее из одной латинской буквы, из нескольких символов, целого слова или нескольких слов. Поэтому каждая переменная перед ее использованием должна быть объявлена, т. е. ей должна быть выделена область памяти [3].

В языке C++ все переменные имеют определенный тип данных, который определяет множество допустимых значений и набор операций для них.

Тип данных присваивается переменной при ее объявлении или инициализации.

II. Основные типы данных в C++ [4]

К основным типам данных в C++ относят следующие [1; 3]:

- `int` – целочисленный тип данных;
- `float` – тип данных с плавающей запятой;
- `double` – тип данных с плавающей запятой двойной точности;
- `char` – символьный тип данных;
- `bool` – логический тип данных;
- `void` – пустой.

Каждый тип данных занимает заранее известное количество байт памяти. Стандарт языка C++ *не накладывает жёстких ограничений* на размеры типов, они могут *отличаться* для разных платформ и компиляторов.

При написании программ необходимо максимально использовать готовые наборы скомпилированного кода, которые были разработаны сторонними программистами и «упакованы» для повторного использования в других программах. Такие наборы формируют *библиотеку*, которая состоит из функций, подпрограмм и других объектов. С их помощью можно не только быстрее написать программу, но и расширить ее возможности [5].

III. Объявление переменных

При объявлении переменной в C++ прежде всего необходимо указать тип данных, значение которых будет хранить в себе эта переменная, ее название и, если требуется, задать (присвоить) начальное значение. Присваивание используется для сохранения определенного значения (константы) в переменной. Присвоенные значения в процессе выполнения программы не могут быть изменены.

Оператор присваивания в языке C++ обозначается символом равно (=), не является знаком равенства и не может использоваться для сравнения значений. Например, запись вида `numberA = 1` задает переменной `numberA` значение числа `1`. В качестве оператора равенства используется двойное равно `==`.

Примеры объявления переменных:

```
int a; // объявление переменной a целого типа.
float b; // объявление переменной b типа данных с плавающей запятой.
double c = 10.2; // объявление и инициализация переменной типа double начальным значением.
char d = 'a'; // объявление и инициализация переменной типа char.
bool k = true; // объявление и инициализация логической переменной k.
```

Средства для потокового ввода и вывода данных расположены в библиотеке `iostream`, которая позволяет выводить данные на экран и обрабатывать пользовательский ввод. Для их использования следует прописать строку

```
using namespace std;
```

`cout` – это объект, который находится в библиотеке `iostream` и используется для вывода данных на экран (в консольное окно).

Оператор вывода << возвращает ссылку на объект своего первого операнда, т. е. на поток. Это позволяет использовать сцепленные операции для размещения в потоке данных. Для вывода нескольких предложений в одной строке оператор вывода << нужно использовать несколько раз.

`cin` – является противоположностью `cout`, он получает данные от пользователя с помощью оператора ввода >>. Этот оператор извлекает данные из потока, заданного ее левым операндом, и заносит их в переменную, заданную правым операндом. Операция возвращает поток, указанный как ее левый операнд. Допускаются сцепленные операции взятия из потока.

IV. Комментарии кода

Однострочные комментарии – это комментарии, которые пишутся после символов в отдельных строках и всё, что находится после этих символов комментирования, игнорируется компилятором. Например:

```
    cout << «Hello, world!» << endl; // всё, что находится справа от
двойного слеша, игнорируется компилятором
```

Как правило, однострочные комментарии используются для объяснения одной строчки кода.

Размещение комментариев таким образом может приводить к затруднению чтения кода. Следовательно, однострочные комментарии рекомендуется размещать над строками кода:

```
    // cout и endl находятся в библиотеке iostream
    cout << «Hello, world!» << endl;
```

Многострочные комментарии – это комментарии, которые пишутся между символами `/* */`. Всё, что находится между звёздочками, игнорируется компилятором:

```
    /* Многострочный комментарий.
    строки внутри игнорируются
    компилятором. */
```

V. Базовые правила написания комментариев

Во-первых, на уровне библиотек / программ / функций комментарии отвечают на вопрос *«Что?»*: «Что делают эти библиотеки / программы / функции?». Например:

```
// Эта программа вычисляет значение выражения
// Эта функция находит наибольший общий делитель чисел
// Данный фрагмент кода генерирует псевдослучайное число
```

Таким образом, комментарии позволяют понять, что делает программа, без необходимости подробного анализа кода. Это особенно важно при работе в команде, так как не каждый специалист будет знать весь код большого программного продукта.

Во-вторых, внутри библиотек / программ / функций комментарии отвечают на вопрос *«Как?»*: «Как код выполняет задание?». Например:

```
/* Для расчета средней цены товара выполняется суммирование цен
товаров по данной категории, а затем полученное значение делится
на количество товаров в данной категории */
```

С использованием комментариев можно исключить часть кода при компиляции: например, временно, при отладке программы. Чтобы закомментировать одну строку кода можно применить однострочные символы для комментирования //.

Закомментировано:

```
//    cout << 1;
```

Чтобы закомментировать блок кода, следует применить символы многострочного комментария /* */.

Закомментировано символами многострочного комментария:

```
/*
cout << 1;
cout << 2;
cout << 3;
*/
```

Пример программы:

```
#include <iostream>      // включает в программу заголовочный файл для организации ввода-вывода в языке программирования C++
using namespace std;     // сообщает компилятору, что нужно использовать группу функций, которые являются частью стандартной библиотеки std
int main()               // обязательная функция (программный блок)
{                        // начало тела функции
```

```
    cout << "Hello, world!" << endl;     // оператор вывода на экран
    return 0;                  // один из способов выхода из функции, при 0 успеш-
ный выход
    }                          // окончание тела функции
```

VI. Среды разработки программ

Среда разработки программного обеспечения Visual Studio 2022

В настоящее время среда разработки программного обеспечения Visual Studio (VS) от Microsoft широко используется на практике и является одной из самых популярных в мире. Visual Studio позволяет разрабатывать программные продукты на языках программирования C#, C++, Java, Python, JavaScript, TypeScript, Node.js и др. под операционными система Windows, MacOS, Linux.

Этапы установки и запуска Visual Studio 2022

1. Для установки Visual Studio 2022 необходимо перейти на официальный сайт Microsoft (https://visualstudio.microsoft.com/) и скачать установщик Visual Studio Community 2022 выбрав его в выпадающем меню как показано на рисунке 1.1.

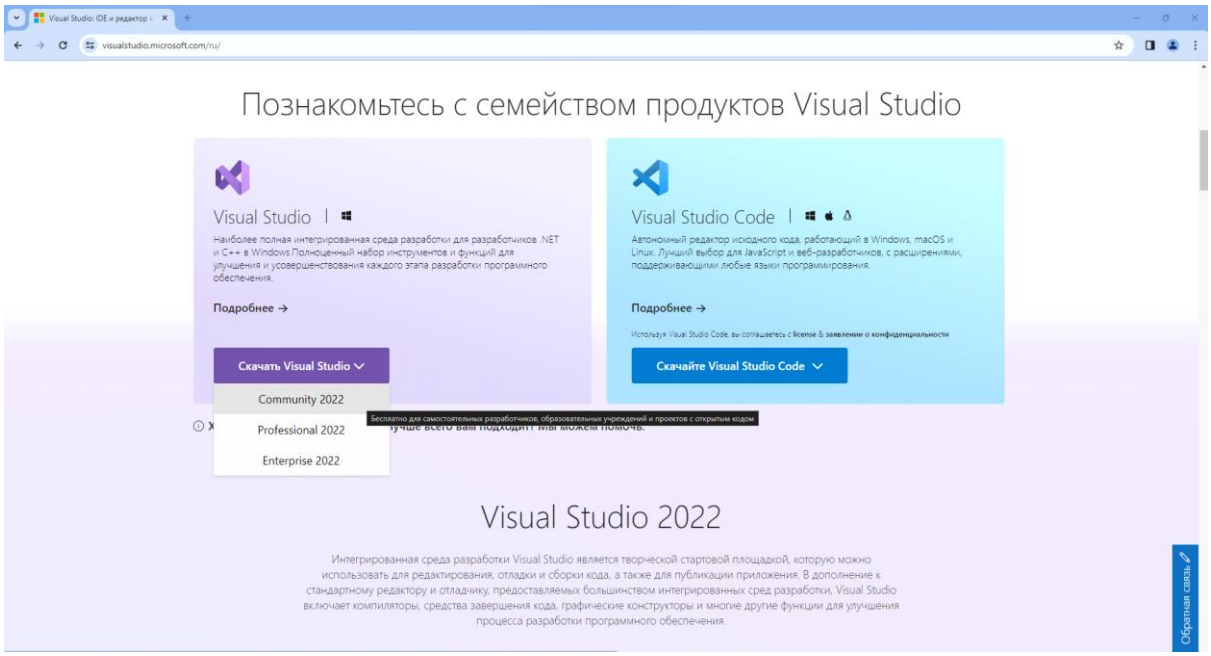


Рисунок 1.1. – Страница сайта Microsoft
для скачивания установщика Visual Studio 2022

2. После загрузки необходимо запустить скачанный файл – установщик VisualStudioSetup.exe (рисунок 1.2) – и дождаться завершения установки, как показано на рисунке 1.3.

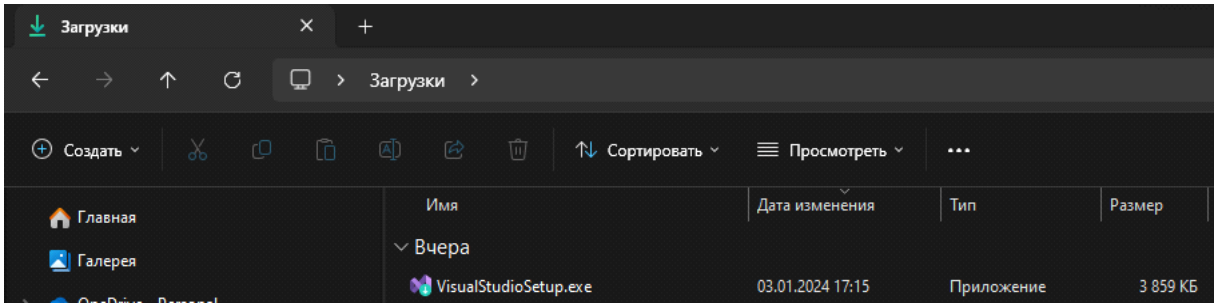


Рисунок 1.2. – Запуск установщика VS 2022

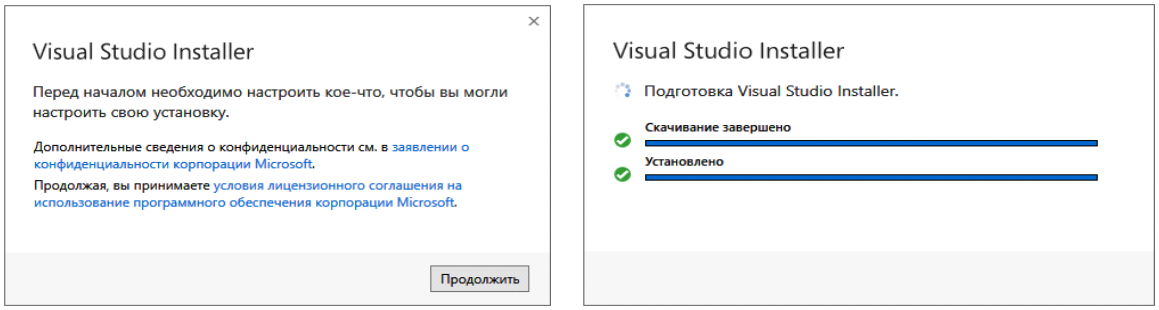


Рисунок 1.3. – Установка VS 2022

3. По завершении установки откроется окно Visual Studio Installer, в котором можно выбрать и загрузить все необходимые рабочие нагрузки, как показано на рисунке 1.4.

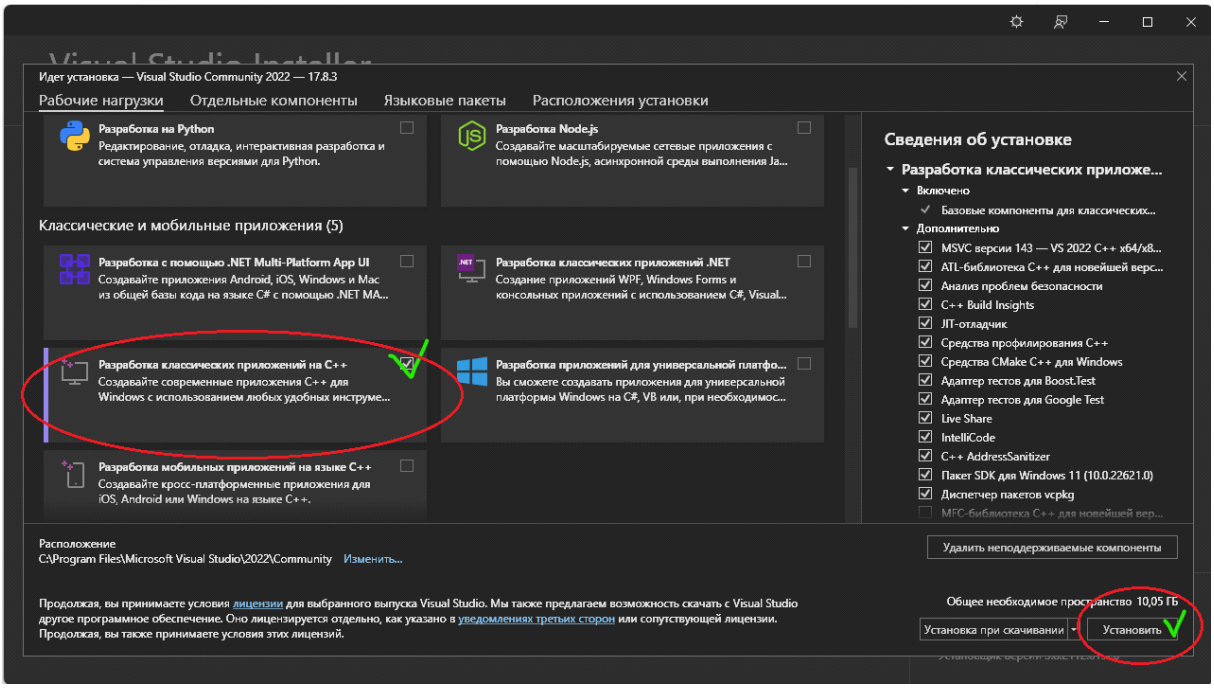


Рисунок 1.4. – Выбор рабочих нагрузок

На рисунке 1.5 показан процесс загрузки и установки выбранных нагрузок.

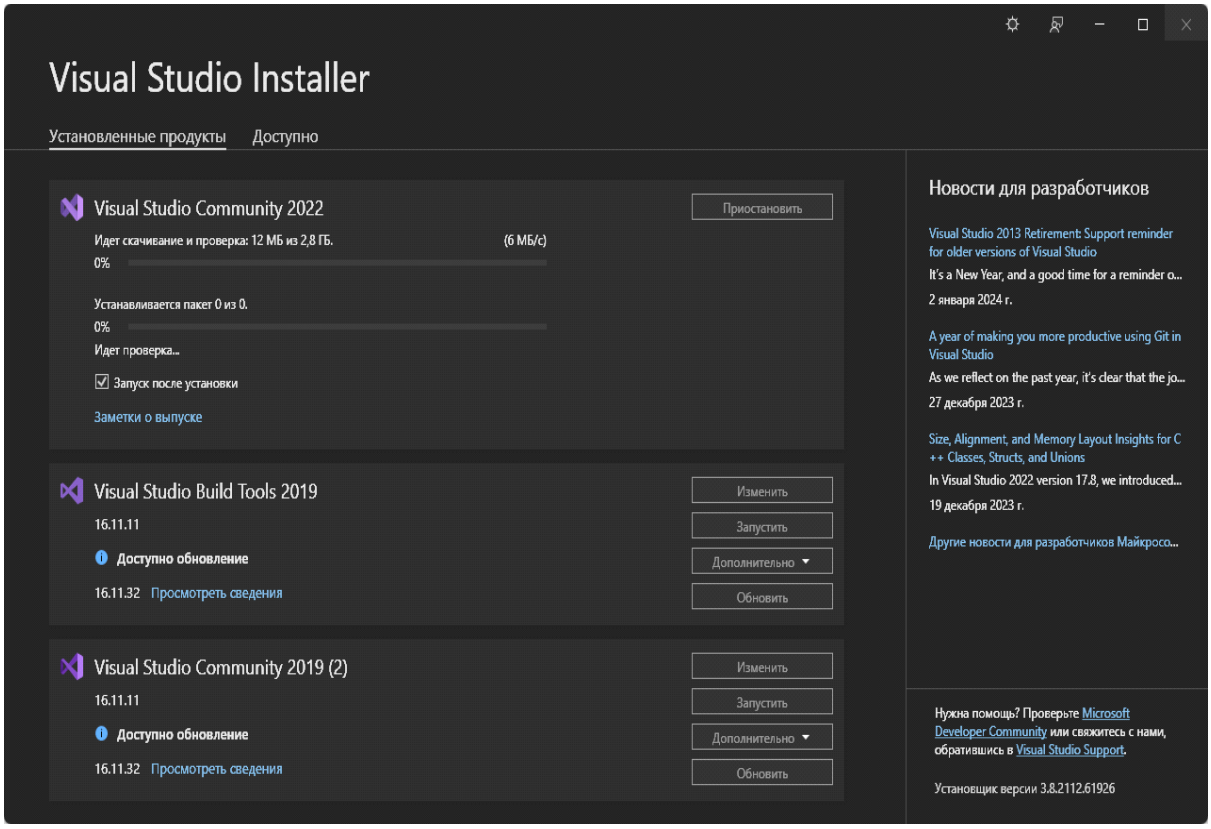


Рисунок 1.5. – Загрузка и установка рабочих нагрузок

После завершения установки всех нагрузок отобразится окно подтверждения завершения загрузки (рисунок 1.6).

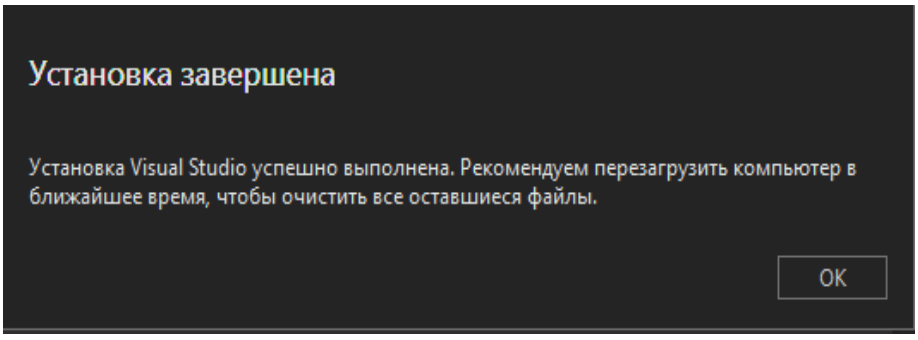


Рисунок 1.6. – Завершение установки рабочих нагрузок

4. Затем можно запускать Visual Studio 2022. На стартовом экране будет предложено войти в учётную запись, создать её или пропустить этот шаг, как показано на рисунке 1.7.

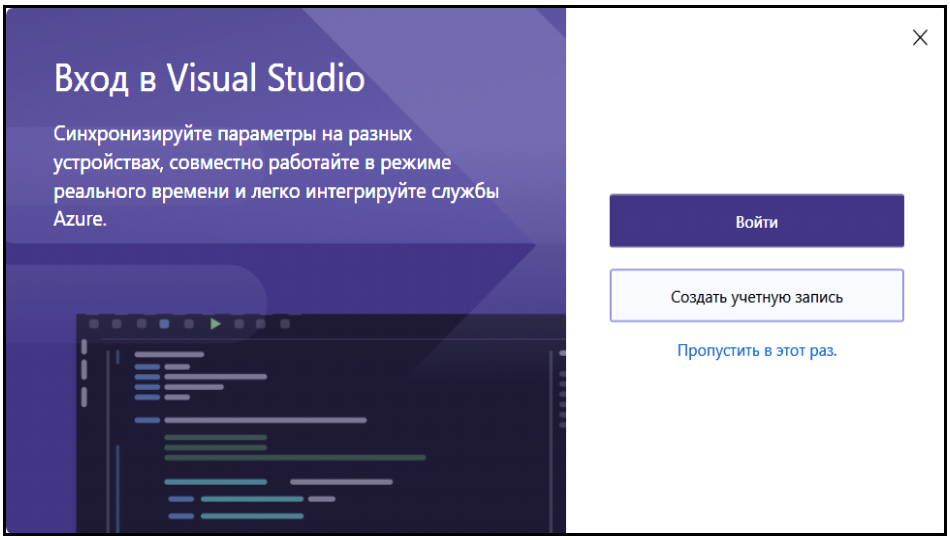


Рисунок 1.7. – Стартовый экран VS 2022

5. При первом открытии предлагается персонализировать работу с VS 2022 (рисунок 1.8).

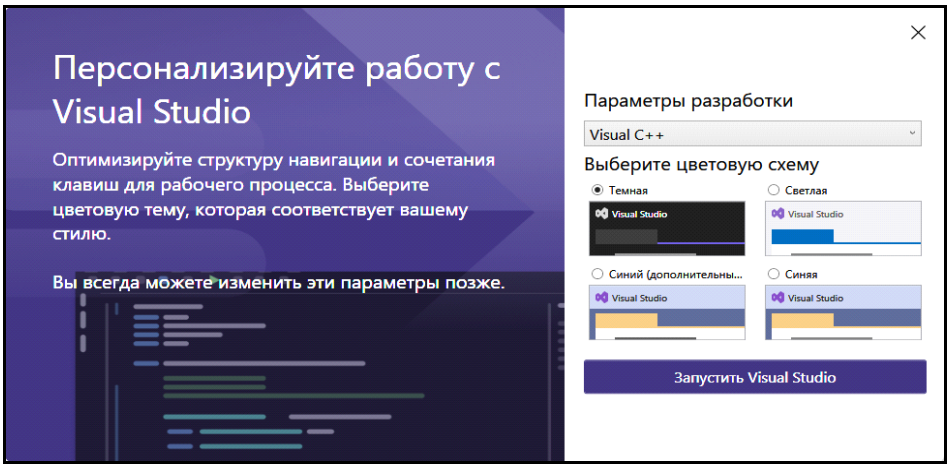


Рисунок 1.8. – Персонализация работы с VS 2022

6. После персонализации нажатием кнопки «Запустить Visual Studio» открывается начальный экран, на котором можно выбрать уже существующие проекты (клонирование репозитория[1]; открыть проект[2] или решение[3]; открыть локальную папку) либо создать новый (создание проекта), как показано на рисунке 1.9.

[1] Интернет-хранилище (например, GitHub).
[2] Содержит все файлы, которые будут скомпилированы в исполняемую программу, библиотеку или веб-сайт.
[3] Контейнер для одного или нескольких связанных проектов, а также сведения о сборке, параметры окна Visual Studio и любые другие файлы, которые не связаны с определенным проектом. Соответственно, проект находится внутри решения [6].

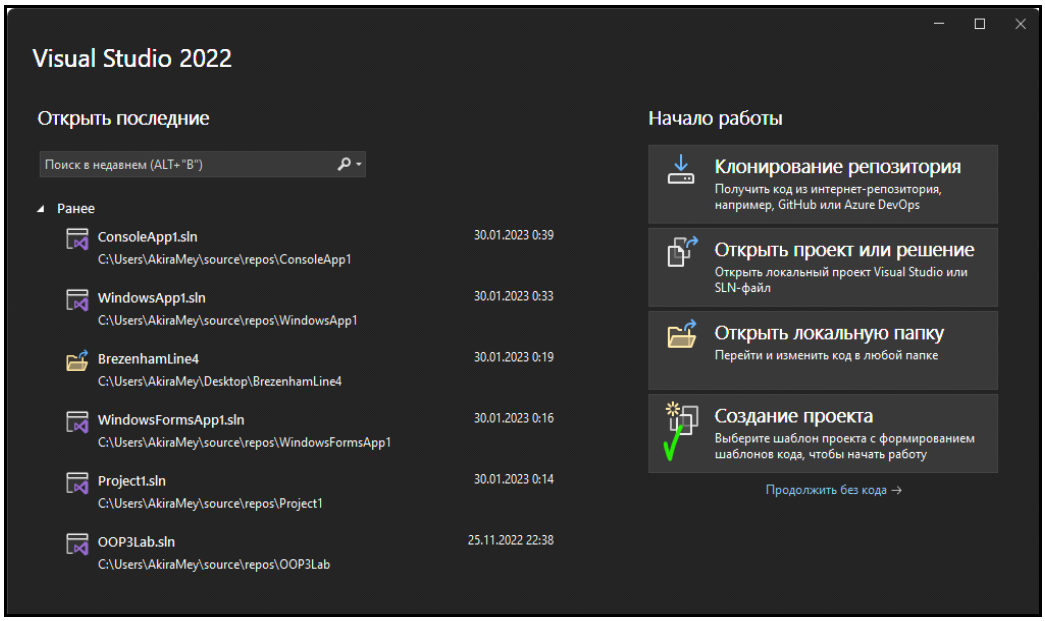


Рисунок 1.9. – Начальный экран Visual Studio 2022

7. При выборе «Создание проекта» откроется конструктор создания проекта, в котором необходимо выбрать шаблон создаваемого проекта.

В левой части конструктора создания проекта отображается список недавно выбранных шаблонов (рисунок 1.10). Они отсортированы *по времени использования*. Для быстрого и удобного выбора шаблона можно воспользоваться фильтром проектов по параметрам *Язык* (например, C# или C++), *Платформа* (например, Windows или Azure) и *Тип проекта* (например, «Классический» или «Интернет») [6].

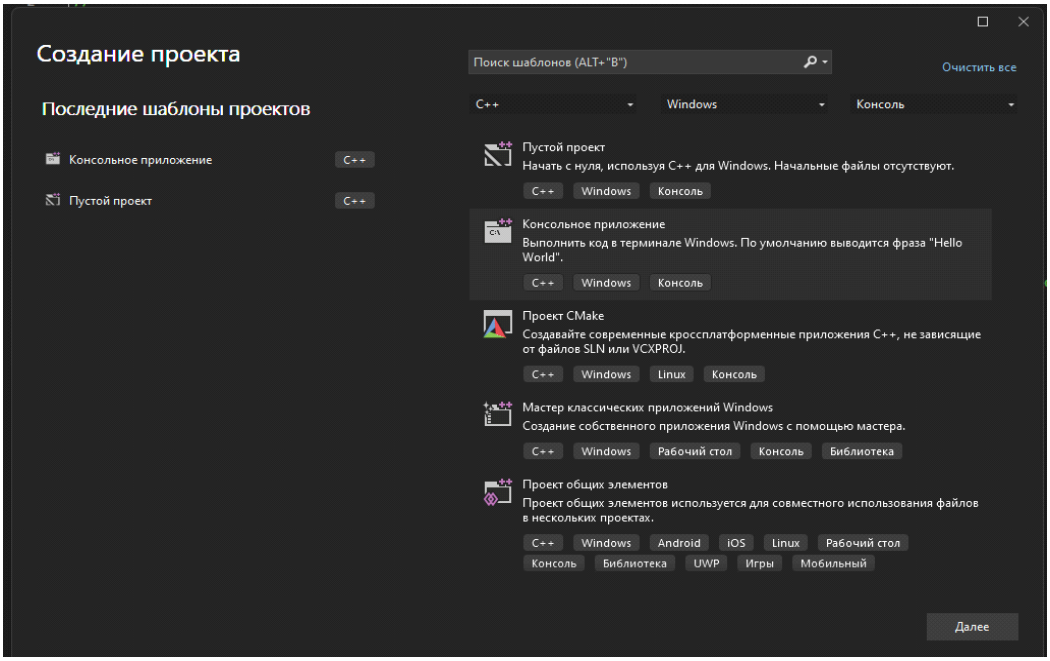


Рисунок 1.10. – Окно создания проекта

Для выполнения лабораторных работ необходимо выбрать шаблон «Консольное приложение». После выбора шаблона проекта отобразится меню настройки проекта (рисунок 1.11), в котором можно задать следующие параметры:

– *имя проекта* (генерируется автоматически, но можно изменить на любое удобное имя, соответствующее теме проекта, например, cppLab1);

– *расположение* (автоматически указывается стандартное расположение, но для удобства поиска можно поменять на любое место);

– *имя решения* (автоматически подстраивается под имя проекта, при изменении имени проекта будет также изменено и имя решения).

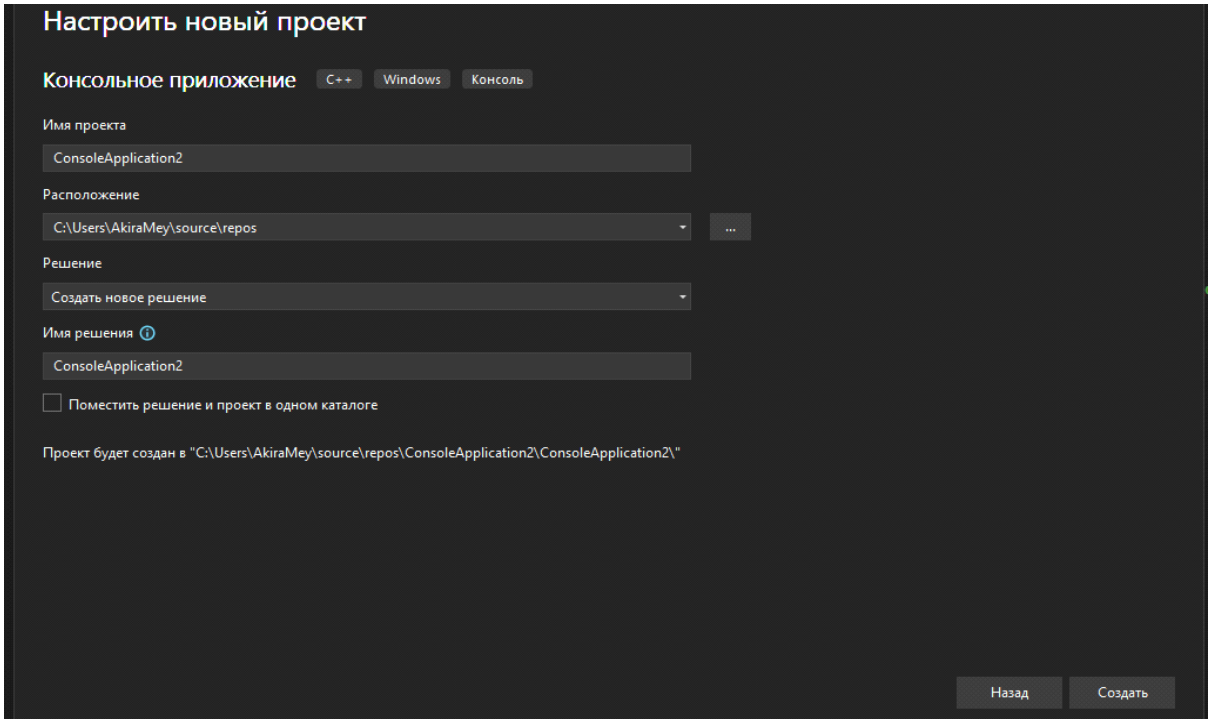


Рисунок 1.11. – Окно настройки проекта

8. По результатам создания проекта будет создано консольное приложение с одним исходным файлом (в примере это ConsoleApplication1.cpp). После завершения создания проекта с использованием шаблона «Консольное приложение» открывается рабочий экран проекта и можно приступать к написанию кода.

Для запуска выполнения программы необходимо нажать Ctrl + F5 или использовать меню *Отладка → Запуск без отладки* (рисунок 1.12). Кроме того, запустить программу на выполнение можно при помощи кнопки  панели инструментов.

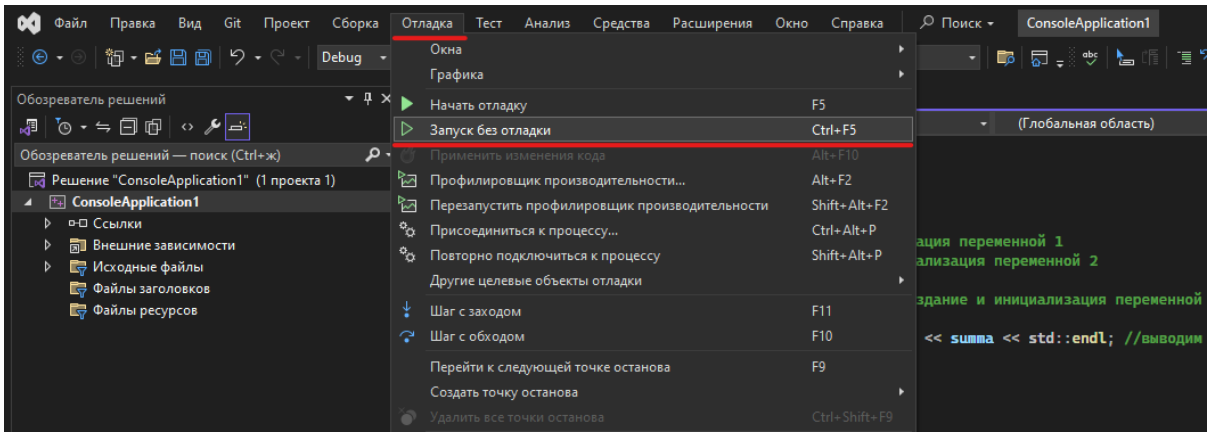


Рисунок 1.12. – Запуск проекта без отладки

В окне обозревателя решений можно добавлять файлы и управлять ими (рисунок 1.13, область окна № 1). Для добавления новых файлов кода можно использовать меню, последовательно выбирая пункты *Проект → Добавить новый элемент*. Если нужно добавить в проект существующие файлы кода, то используется последовательность *Проект → Добавить существующий элемент*. В окне «Выходные данные» можно просматривать выходные данные сборки и другие сообщения (рисунок 1.13, область окна № 2) [7].

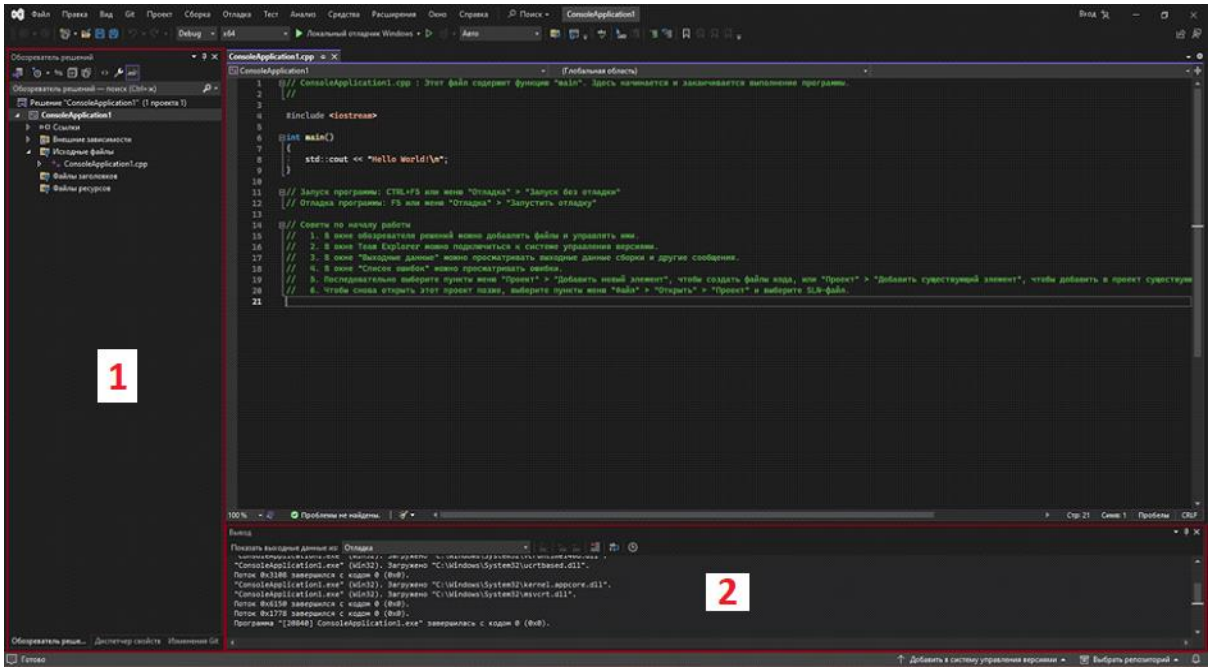


Рисунок 1.13. – Рабочий экран при написании программы в VS

В окне «Список ошибок» можно просматривать ошибки (рисунок 1.14).

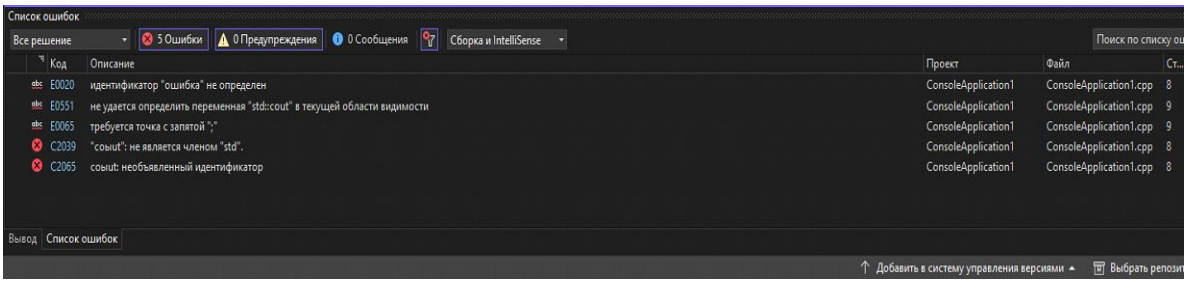


Рисунок 1.14. – Окно «Список ошибок»

На рисунке 15 показан результат выполнения программы.

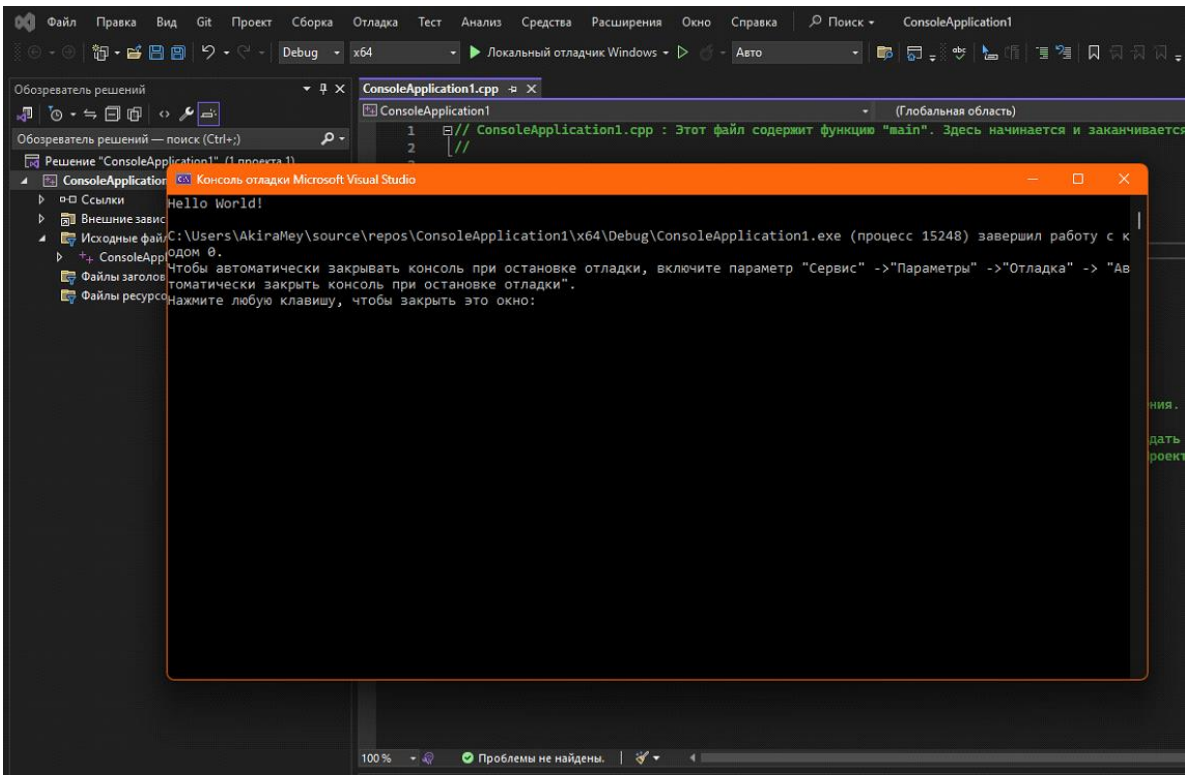
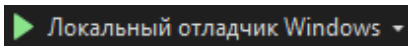


Рисунок 1.15. – Результат выполнения программы

Режим отладки

Для демонстрации использования режима отладки при разработке программного обеспечения используем код, показанный рисунке 1.16.

Для отладки программы необходимо нажать F5 или использовать меню *Отладка → Запустить отладку* (рисунок 1.17). Кроме того, запустить процесс отладки можно при помощи кнопки 

панели инструментов.

При нажатии клавиши F5 происходит запуск приложения с присоединенным отладчиком. Приложение будет загружено и выдаст выходные данные консоли.

Для отладки необходимо задать точку прерывания (останова) программы, которая указывает, где VS следует приостановить выполнение кода для проверки значения переменных, состояния памяти, либо выполнения ветви кода [7]. Точки прерывания эффективны, если известны строка или раздел кода, которые необходимо анализировать. Например, в строке 11 функции `main` можно указать точку останова, щелкнув левое поле строки кода с номером (рисунок 1.18). В этом месте появится красная точка.

```
#include <iostream>

int main()
{
    int variable1 = 1; // создание и инициализация переменной 1
    float variable2 = 2.2; // создание и инициализация переменной 2

    float summa = variable1 + variable2; // создание и инициализация переменной суммы

    std::cout << " Variable 1 +  Variable 2: " << summa << std::endl; //выводим полученную сумму в консоль


}
```

Рисунок 1.16. – Исходный код

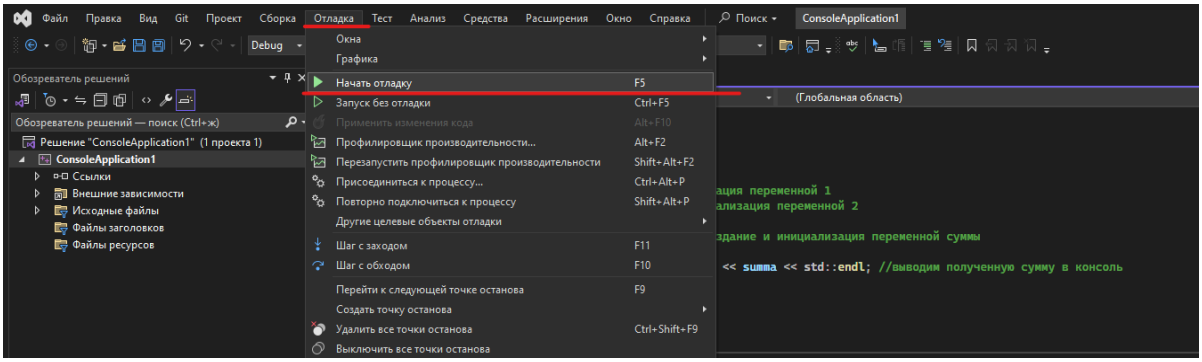


Рисунок 1.17. – Запуск с отладкой

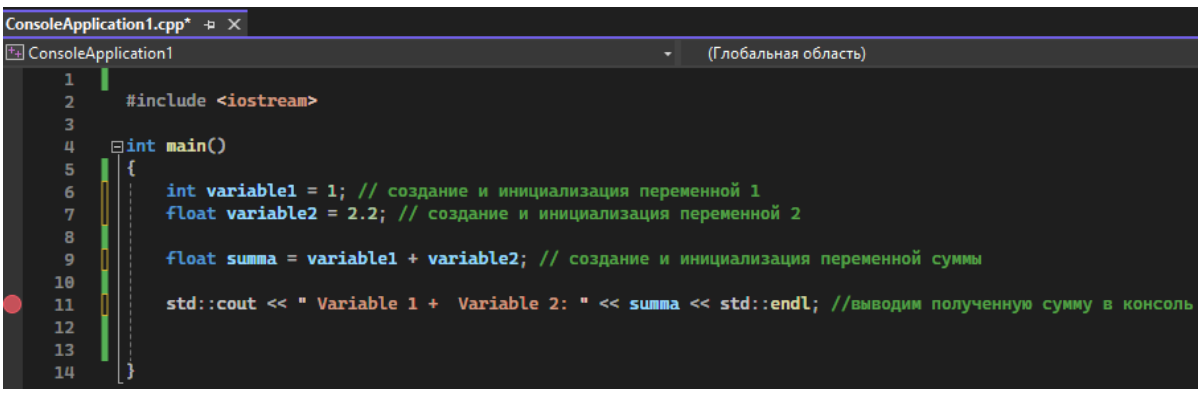


Рисунок 1.18. – Установка точки прерывания

Далее следует снова нажать клавишу F5 или кнопку «Начать отладку». Приложение запустится и отладчик перейдет к строке кода, где задана точка останова, в консоль выведутся данные по строке с точкой остановы (рисунок 1.19):

1 – название переменных, их значение и тип данных;

2 – местонахождение точки остановы.

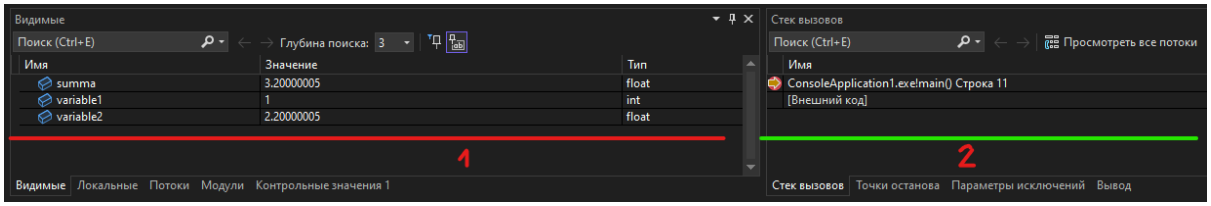


Рисунок 1.19. – Данные по строке с точкой останова

Желтая стрелка указывает на оператор, на котором приостановлен отладчик (рисунок 1.20).

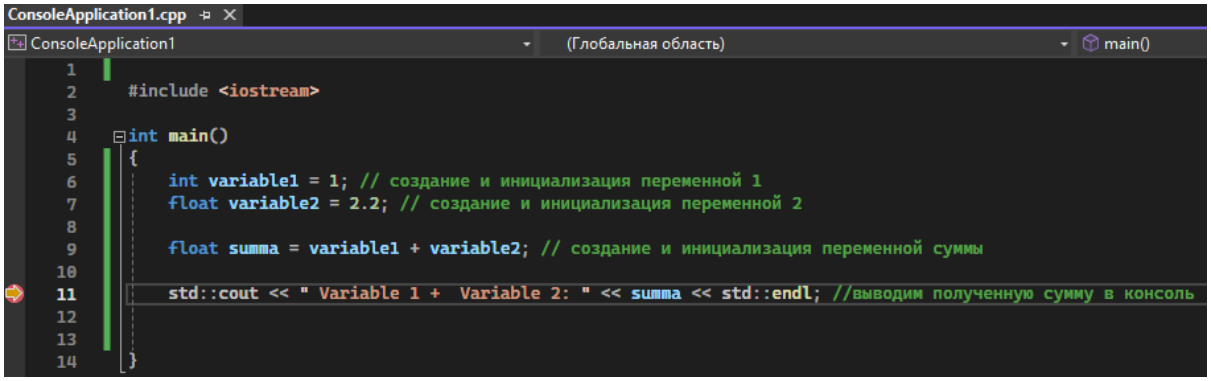


Рисунок 1.20. – Точка приостановки отладчика

В этой же точке приостанавливается выполнение приложения, т. е. этот оператор пока не выполнен. Если приложение еще не запущено, клавиша F5 запускает отладчик и останавливается в первой точке останова. В противном случае F5 продолжает выполнение приложения до следующей точки останова. Нажимая F11, можно построчно отслеживать выполнение кода. Функции, позволяющие проверять переменные, являются полезными возможностями отладчика. Чтобы посмотреть значение переменной на момент приостановки приложения, нужно навести мышь на переменную (рисунок 1.21).

В окне «Видимые» (см. рисунок 1.19) отображаются все переменные, используемые в текущей или предыдущей строке. В окне «Локальные» показаны переменные, которые находятся в текущей области, т. е. текущем контексте выполнения.

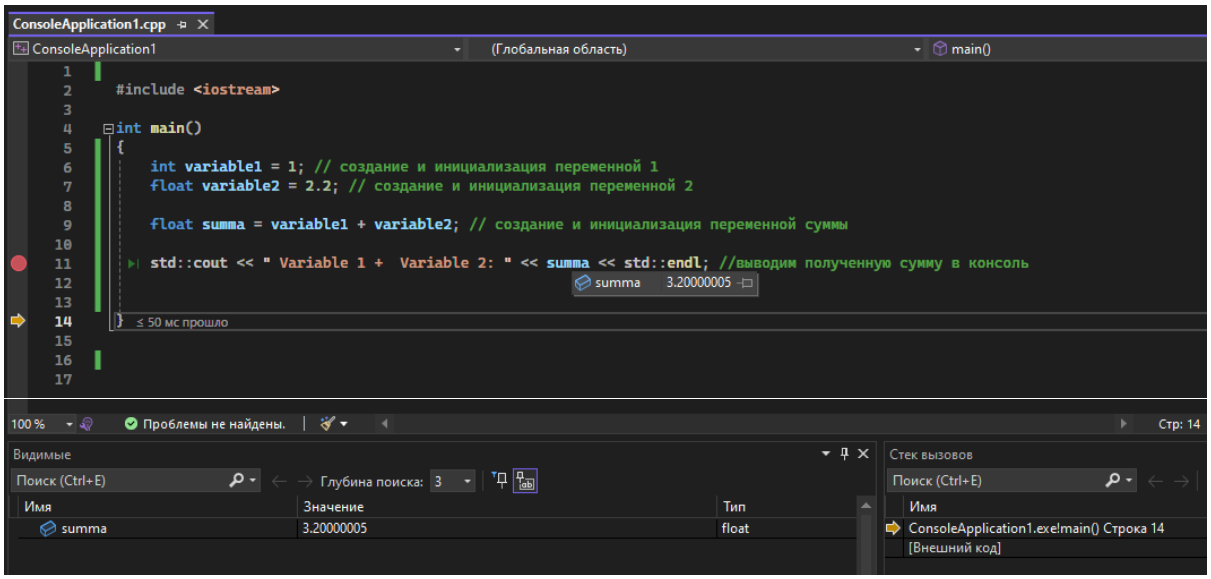


Рисунок 1.21. – Пример отображения значения переменной в приостановленном приложении

Для перезапуска отладчика можно использовать кнопку «Перезапустить» на панели инструментов отладки или комбинацию клавиш Ctrl + Shift + F5. На рисунке 1.22 показан результат выполнения кода приложения.

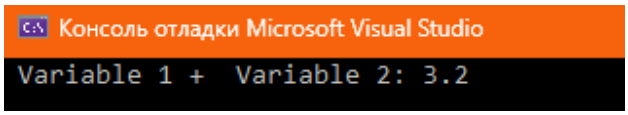


Рисунок 1.22. – Результат выполнения кода приложения

Более подробная информация по работе с Visual Studio представлена на ресурсе https://visualstudio.microsoft.com/ru/vs/getting-started/.

Онлайн-компилятор OnlineGDB

OnlineGDB – это онлайн-инструмент для компиляции и отладки кода, написанного на различных языках программирования, в том числе и на C++ (доступен по ссылке https://www.onlinegdb.com/online_c++_compiler).

Преимуществом OnlineGDB является простота в использовании и интуитивно понятный интерфейс (рисунок 1.23). На рисунке 1.23 выделено 6 основных областей:

1 – основное меню;

2 – строка меню проекта;

3 – файл исходного кода;

4 – код программы;

5 – выпадающий список с выбором языка программирования;

6 – консоль приложения.

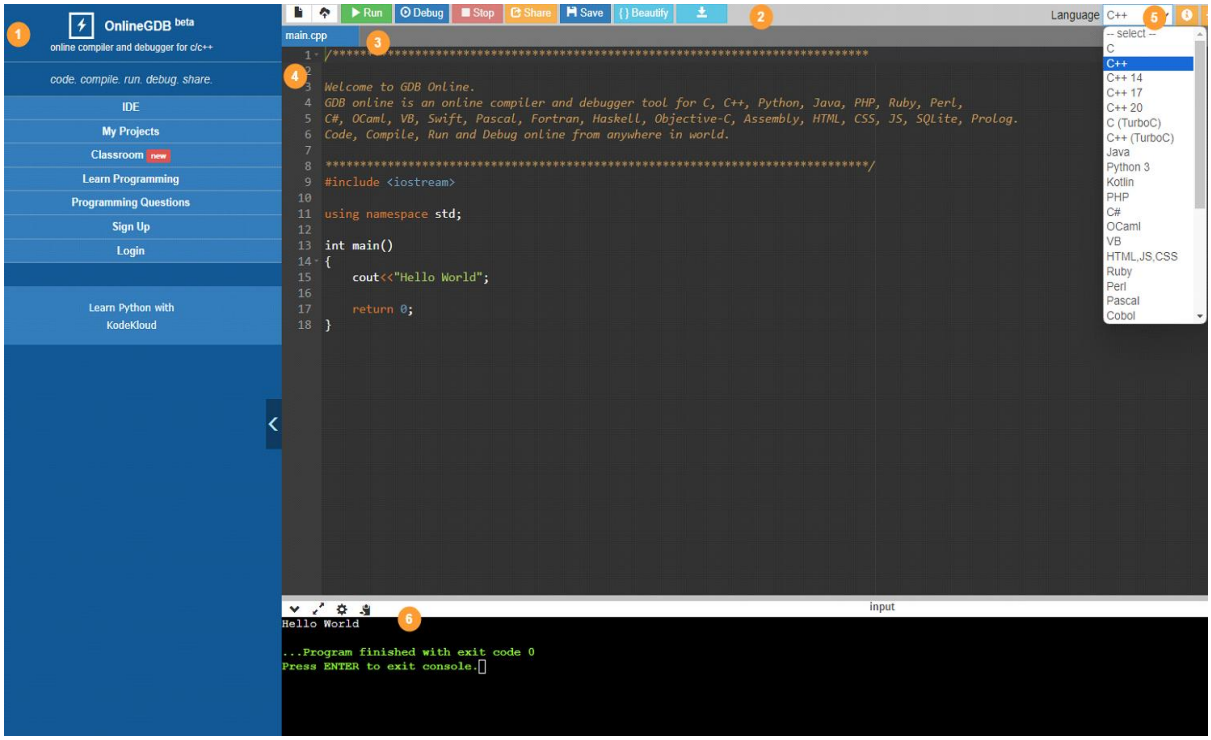


Рисунок 1.23. – Рабочий экран OnlineGDB

Онлайн-компилятор OnlineGDB предоставляет возможность регистрации на сайте, а после авторизации появляется возможность создавать новые проекты, при их наличии открывать список уже имеющихся (рисунок 1.24).

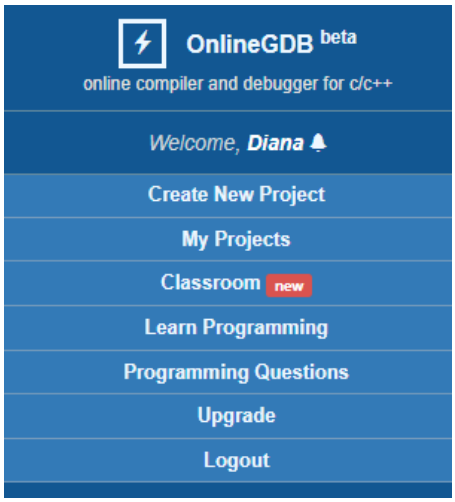


Рисунок 1.24. – Меню сайта

На рисунке 1.25 показано окно с уроками по программированию, а на рисунке 1.26 – окно форума.

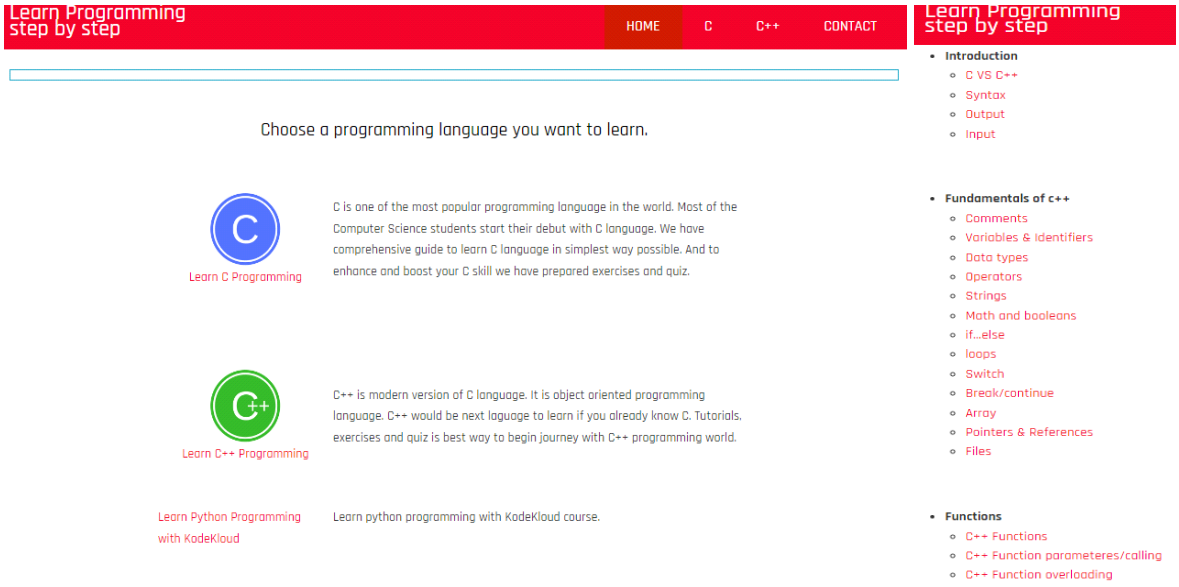


Рисунок 1.25. – Уроки по программированию

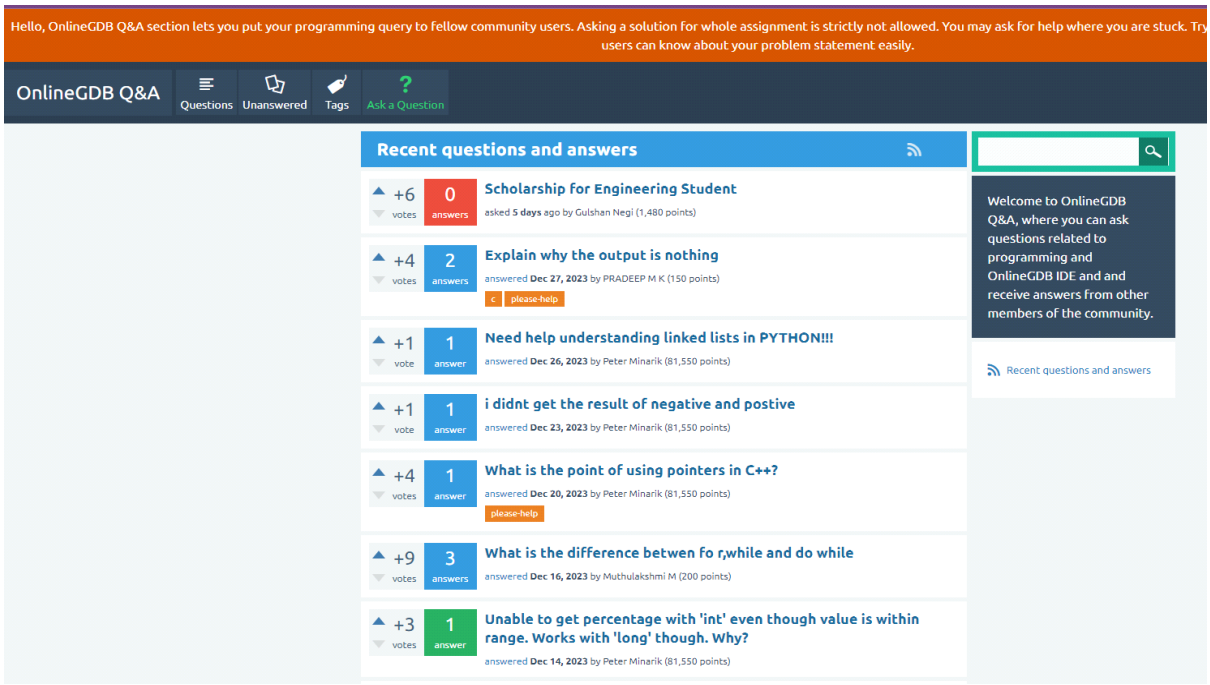


Рисунок 1.26. – Форум вопросов по программированию

При работе с проектом удобно пользоваться строкой меню с кнопками (рисунок 1.27):

1 – создать новый файл;

2 – загрузить файл;

3 – выполнить компиляцию проекта;

4 – отладка проекта;

5 – завершить выполнение программы;
6 – поделиться файлом с исходным кодом;
7 – сохранение файла с кодом;
8 – табуляция кода, т. е. выделение отдельных блоков кода отступами;
9 – скачать файл с исходным кодом.

Рисунок 1.27. – Строка меню проекта

Написание и компиляция кода в OnlineGDB

Для создания нового проекта рекомендуется (но не обязательно) авторизоваться на сайте https://www.onlinegdb.com и выбрать пункт меню *Create New Project* (рисунок 1.28).

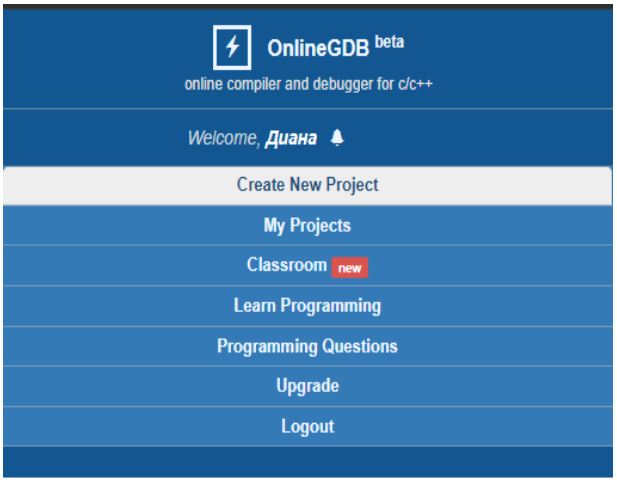


Рисунок 1.28. – Создание нового проекта

В появившемся окне присутствует базовый код компилятора (см. рисунок 1.20), который при необходимости можно удалить и начать написание программы с нуля.

Для примера рассмотрим выполнение следующего кода:

```
#include <iostream>

int main()
{
  int variable1 = 1; // создание и инициализация переменной 1
  float variable2 = 2.2; // создание и инициализация переменной 2
  // создание и инициализация переменной суммы
  float summa = variable1 + variable2;

  //выводим полученную сумму в консоль
  std::cout << " Variable 1 + Variable 2: " << summa << std::endl;
}
```

Для запуска кода без отладки необходимо нажать кнопку ▶ Run в строке меню проекта. При успешной компиляции кода проекта результат выполнения будет отображен в консоли (рисунок 1.29).

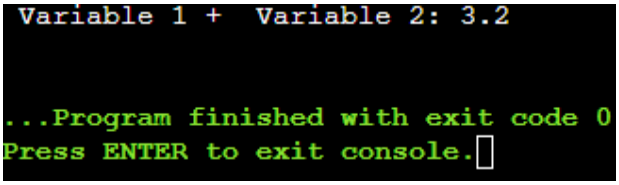


Рисунок 1.29. – Компиляция кода без отладки

При возникновении ошибки во время компиляции с отладкой и без нее в консоль выведется сообщение с указанной причиной и местом ошибки, например, как показано на рисунке 1.30.

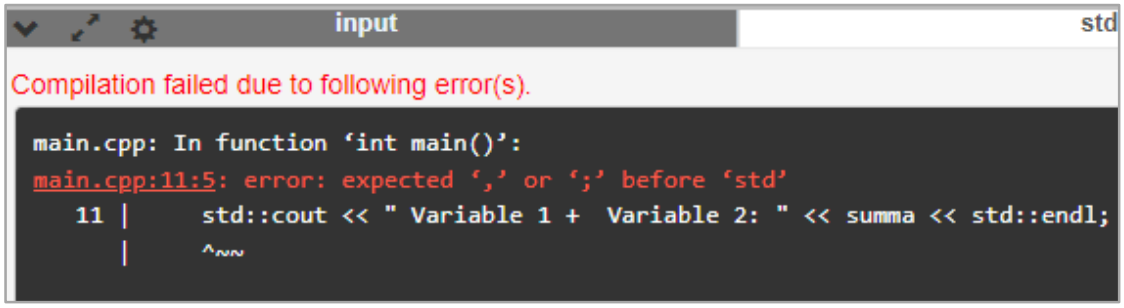


Рисунок 1.30. – Результат отображения ошибки при компиляции без отладки

Для выполнения кода с отладкой необходимо нажать Debug в строке меню проекта. Результат успешной компиляции кода с отладкой в консоли показан на рисунке 1.31.

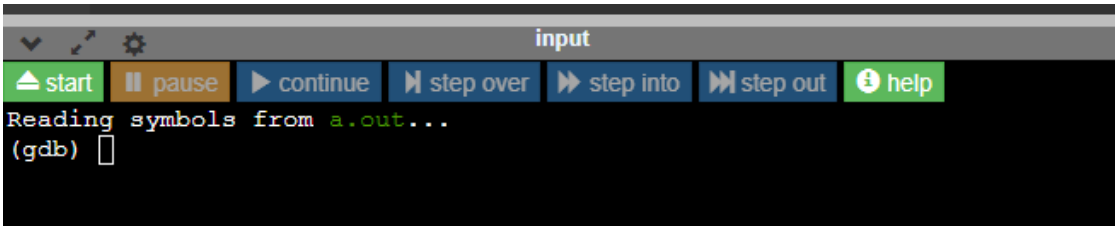


Рисунок 1.31. – Результат отображения успешной компиляции кода с отладкой

В теле функции `main` установите точку останова, щелкнув левое поле строки кода с номером 11 `std::cout << " Variable 1 + Variable 2: " << summa << std::endl;` (рисунок 1.32). В месте установки точки останова появится красная точка.

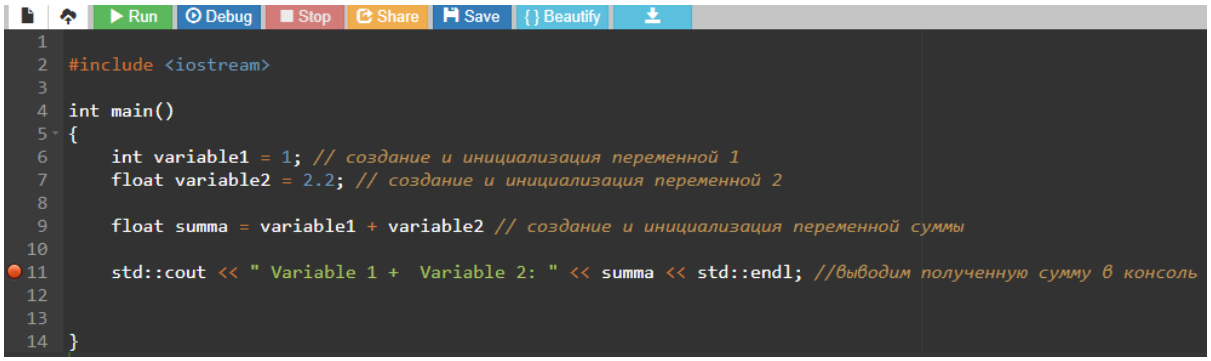


Рисунок 1.32. – Отображение точка прерывания

После запуска программы с отладчиком и установленной точкой останова выполнение программы приостановится на первой точке останова. В консоль выведутся данные по строке с точкой останова: чтение из потока вывода программы (Reading symbols from a.out); информация об остановке (info break); номер точки останова (Num 1); тип – точка останова (Type breakpoint); содержимое и адрес (Disp End Adress keep y 0x0000000000000011e5); функция, в которой находится строка с точкой останова (what in main()); файл, в котором находится эта функция (at main.cpp); номер строки с точкой останова (8) (рисунок 1.33).

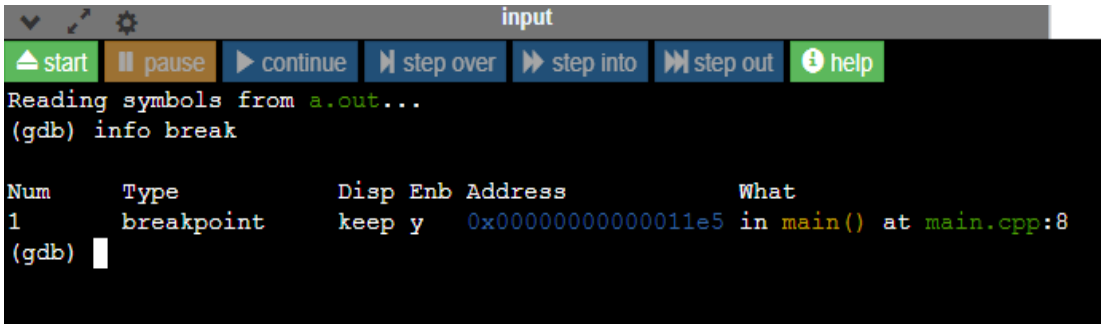


Рисунок 1.33. – Компиляции кода с отладкой и точкой останова на 8-й строке кода

Ход работы

Ознакомиться с теоретическим материалом и закрепить знания с подготовкой ответов на контрольные вопросы.

Создание консольного приложения в MS VS

1. Выбрать меню *Файл* → *Создать* → *Проект*, ввести название проекта (lab1_фамилия).

2. Открыть двойным нажатием на панели «Обозреватель решений» файл lab1_фамилия.cpp.

3. Записать в файл следующие строки:

```
#include <iostream>
using namespace std;
int main()
{
  int variableX = 10, variableY = 20;
  int variableZ = variableX + variableY;
  cout<< "z = " << variableZ <<endl;
  return 0;
}
```

4. Записать в отчет содержимое файла (Листинг 1).

5. Скомпилировать полученную программу (*Отладка → Запуск без отладки*).

6. Сделать скриншот результата выполнения программы и прикрепить его в отчет (Рисунок 1 – Результат выполнения программы).

7. Убедиться, что компиляция прошла успешно. При возникновении ошибки остановить выполнение программы, исправить ошибку и скомпилировать снова.

8. Установить точку прерывания на строке «`int variableX = 10, variableY = 20;`», нажав левой кнопкой мыши на серой границе слева от строки или установив курсор на строке и нажав кнопку F9: должна появиться красная точка.

9. Запустить программу на выполнение в отладчике (*Отладка → Начать отладку*):

- выполнение программы остановится на строке с точкой прерывания;
- далее необходимо выполнить одну строку программы (*Отладка → Шаг с обходом*);
- наведите курсор мыши на переменные `variableX` и `variableY` и посмотрите, какие значения они приняли, запишите их в отчет;
- правой кнопкой мыши нажмите на переменную `variableZ` и в появившемся меню выберите пункт *Добавить контрольное значение*;
- выполните одну строку программы (*Отладка → Шаг с обходом*);
- значение переменной `variableZ` изменилось (подсвечено красным цветом на панели «Контрольные значения»), запишите в отчет;
- выберите *Отладка → Остановить отладку*, чтобы закончить выполнение программы.

10. Запустите программу без отладчика (*Отладка → Запуск без отладки*).

11. Запишите в отчет результаты работы программы, появившиеся на экране.

Создание проекта в онлайн-компиляторе OnlineGDB

1. Записать в файл следующие строки:

```
#include <iostream>
using namespace std;
int main()
{
  int variableX = 10, variableY = 20;
  int variableZ = variableX + variableY;
  cout<< "z = " << variableZ <<endl;
  return 0;
}
```

2. Скомпилировать полученную программу (*Run*).

3. Сделать скриншот результата выполнения программы и прикрепить его в отчет.

4. Убедиться, что компиляция прошла успешно. При возникновении ошибки остановить выполнение программы, исправить ошибку и скомпилировать снова.

5. Установить точку прерывания на строке «`int variableX = 10, variableY = 20;`», нажав левой кнопкой мыши на серой границе слева от строки.

6. Запустить программу на выполнение в отладчике (*Debug*):
   - выполнение программы остановится на строке с точкой прерывания;
   - далее необходимо выполнить одну строку программы (*Step over*);
   - в меню справа посмотрите, какие значения имеют переменные `variableX` и `variableY`, запишите их в отчет;
   - выберите *Continue*, чтобы закончить выполнение программы.

7. Запустите программу без отладчика (*Start* в верхней панели инструментов).

8. Запишите в отчет результаты работы программы, появившиеся на экране.

9. Оформить отчет.

10. Загрузить отчет на образовательную платформу.

11. Защитить работу у преподавателя. Для этого требуется:
    - знать ответы на теоретические вопросы;
    - уметь создать проект в MS VS и онлайн-компиляторе;
    - уметь компилировать программу, выполнять отладку с использованием точек прерывания и пошагового выполнения.

Контрольные вопросы

1. Какие этапы необходимо пройти для создания и выполнения программы на компьютере в типичной среде программирования C++? Поясните каждый из них.

2. Что понимают под идентификатором в программировании? Какие правила нужно соблюдать в C++ при задании идентификаторов?

3. Сформулируйте основные требования к представлению компьютерной программы на C++.

4. Что такое переменная? Назовите и дайте характеристику основным типам данных в C++. Приведите примеры объявления переменных, поясните их.

5. Какие средства потокового ввода и вывода данных используются в C++? Приведите примеры операторов с пояснением.

6. Сформулируйте основные принципы написания комментариев для кода. Как закомментировать строку и фрагмент текста или кода?

7. Поясните программные термины: проект, решение, репозиторий.

8. Что такое компиляция программы? Как ее можно выполнить с использованием Microsoft Visual Studio (OnlineGDB)?

9. Создайте новый проект с использованием Microsoft Visual Studio (OnlineGDB).

10. Поясните, для чего используется режим отладки при разработке программы?

ЛАБОРАТОРНАЯ РАБОТА № 2
Арифметические операции над числами

Цель: получить навыки программной реализации арифметических операций на языке программирования C++ с консольным вводом и выводом данных.

Теоретический материал

При написании программы на C++ следует учитывать, что это язык программирования со статической типизацией. *Статическая типизация* – приём, при котором переменная, параметр подпрограммы, возвращаемое значение функции связывается с типом в момент объявления и тип не может быть изменён позже.

Минимальное и максимальное значение для числового типа можно получить с использованием стандартной библиотеки общего назначения `limits`, которая включает определение характеристик общих типов переменных:

```
#include <iostream>
#include <limits>
using namespace std;
int main()
{
       // для типа float:
  cout << "minimum value: " << numeric_limits<float>::min() << "\n"
  << "maximum value: " << numeric_limits<float>::max() << "\n";
  return 0;
}
```

Размер переменной или типа на этапе компиляции можно определить с помощью оператора `sizeof`:

```
#include <iostream>
#include <limits>
using namespace std;
int main()
{
  cout << "int: " << sizeof(int) << "\n";
  cout << "long int: " << sizeof(long int) << "\n";
  return 0;
}
int: 4
long int: 8
```

I. Арифметические операции

Арифметические операции производятся над числами, а значения, которые участвуют в операции, называются *операндами*. В языке программирования C++ арифметические операции могут быть *бинарными* (производятся над двумя операндами) и *унарными* (выполняются над одним операндом). К бинарным операциям относят следующие:

1. «+» – операция сложения возвращает сумму двух чисел:

```
int number1 {10};
int number2 {7};
int number3 {number1 + number2}; // 17
int number4 {number2 + 2}; // 9
```

В этом примере результат операций применяется для инициализации переменных, но можно использовать операцию присвоения для установки значения переменных:

```
int number1 {10};
int number2 {7};
int number3 = number1 + number2; // 17
int number4 = number2 + 2; // 9
```

2. «–» – операция вычитания возвращает разность двух чисел:

```
int number1 {10};
int number2 {7};
int number3 {number1 - number2}; // 3
int number4 {number2 - 2}; // 5
```

3. «*» – Операция умножения возвращает произведение двух чисел:

```
int number1 {10};
int number2 {7};
int number3 {number1 * number2}; // 70
int number4 {number2 * 2}; // 14
```

4. «/» – Операция деления возвращает частное двух чисел:

```
int number1 {24};
int number2 {4};
int number3 {number1 / number2}; // 6
int number4 {number2 / 4}; // 1
```

При реализации деления с применением C++ следует учитывать, что если в операции участвуют два целых числа, то в случае наличия дробной части

в результате, она будет отбрасываться, даже если результату присваивается переменная `float` или `double`:

```
#include <iostream>
int main()
{
 int number1 {24};
 int number2 {4};

 float number3 {number1 / number2}; // 6
 double number4 {5 / number2};   // 1

 std::cout << "number3 = " << number3 << std::endl;
 std::cout << "number4 = " << number4 << std::endl;
}
```

Для того, чтобы результат мог быть представлен дробным числом (числом с плавающей точкой или точкой), один из операндов также должен представлять собой число с плавающей точкой:

```
#include <iostream>
int main()
{
 float number1 {26};
 int number2 {5};
 float number3 {number1 / number2}; // 5.2
 double number4 {4.0 / number2}; // 0.8
 std::cout << "numbe3 = " << number3 << std::endl;
 std::cout << "number4 = " << number4 << std::endl;
}
```

5. «%» – Операция получения остатка от целочисленного деления:

```
int number1 {26};
int number2 {5};
int number3 {number1 % number2}; // 26 % 5 = 26 - 5 * 5 = 1
int number4 {4 % number2}; // 4 % 5 = 4
```

Правила и примеры записи арифметических операций на C++ приведены в таблице 2.1.

Таблица 2.1. – Правила записи арифметических операций на C++

Операция C++	Арифметическая операция	Алгебраическое выражение	Выражение на C++
Сложение	+	f + 7	f + 7
Вычитание	–	p – c	p – c
Умножение	*	bm	b * m
Деление	/	x / y, x ÷ y	x / y
Вычисление остатка	%	r mod s	r % s

II. Представление чисел в ЭВМ и особенности арифметических операций

Вещественные числа в компьютере хранятся в формате с плавающей точкой (запятой). Число с плавающей запятой состоит из набора отдельных двоичных разрядов:

- *знак (sign),*
- *порядок (exponent)*
- *мантисса (mantis).*

В формате с плавающей точкой вещественное число *X* представляется в виде произведения *мантиссы M* и основания системы счисления *B* в целой степени *E*, называемой *порядком*.

В соответствии со стандартом IEEE754 (Institute of Electrical and Electronics Engineers, Институт инженеров электротехники и электроники) число с плавающей запятой представляется в двоичной системе счисления, т. е. в виде набора битов. Некоторая часть из них кодирует мантиссу числа (*M*), а оставшиеся биты кодируют показатель степени (*E*), кроме этого один бит используется для кодирования знака числа (0 – если число положительное, 1 – если число отрицательное):

$$X = (-1)^S \times M \times B^E,$$

где *B* – основание системы счисления;

S – указатель знака.

Порядок записывается как целое число в коде со сдвигом и указывает, на какое количество позиций и в каком направлении должна сместиться в мантиссе точка (запятая), отделяющая дробную часть от целой.

В ЭВМ используют нормализованное представление числа в форме с плавающей точкой. Мантисса в нормализованном представлении для десятичного числа принимает значения от 1 (включительно) до 10 (не включая). Соответственно, мантисса должна представлять собой *дробное число* на данном интервале. При использовании такой формы любое число записывается единственным образом, исключая 0. Соответственно, в информатике для числа 0 используется специальный признак (бит). Нормализованная запись нуля в десятичной системе:

$$0 = 0{,}0 \times 10^0.$$

Примеры нормализованных записей в двоичной системе счисления:

$1011.11 = 0.101111 \times 2^{100}$ – сдвиг вправо на четыре разряда $(100)_2 = (4)_{10}$;

$1{,}55624 \times 10^2 = 1{,}0011011100 \times 2^{111}$ – мантисса $M = 1{,}0011011100$ и порядок + 111.

В памяти компьютера мантисса представляется как целое число, содержащее только значащие цифры (0 целых и запятая не хранится).

Порядок *n*-разрядного нормализованного числа задается в смещенном представлении: если для порядка выделено *k* разрядов, то к истинному значению порядка прибавляют смещение, равное 2^{k-1}. Таким образом, порядок, принимающий значения в диапазоне от – 128 до + 127, представляется смещенным порядком, значения которого меняются от 0 до 255.

На рисунке 2.1 представлен формат нормализованного числа одинарной точности (32 бита для представления числа), а на рисунке 2.2 показан формат числа двойной точности (64 бита для представления числа).

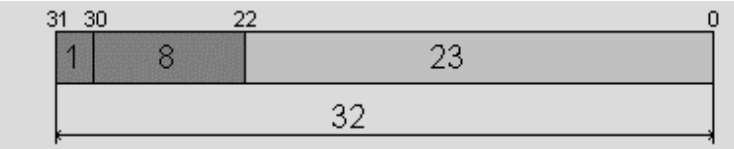


Рисунок 2.1. – Формат нормализированного числа одинарной точности

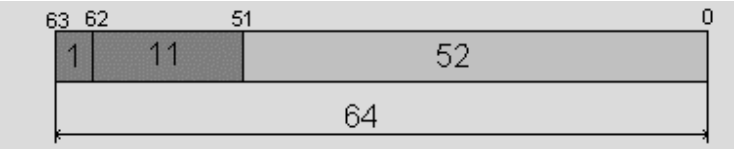


Рисунок 2.2. – Формат нормализированного числа двойной точности

При сложении или вычитании чисел с плавающей точкой, которые значительно отличны между собой по значению, следует учитывать допустимую разрядность типа. Например, при суммировании чисел 1.23E-4 (0.000123) и 3.65E+6 (3650000) сумма должна быть равна 3650000,000123. Но при преобразовании в число с плавающей запятой с точностью до семи цифр результат будет следующим:

$$3.650000E+06 + 1.230000E-04 = 3.650000E+06$$

При реализации на C++:

```
#include <iostream>

int main()
{
 float number1{ 1.23E-4 }; // 0.000123
 float number2{ 3.65E+6 }; // 3650000
 float sum {number1 + number2}; // sum =3.65e+06
 std::cout << "sum =" << sum << "\n";
}

sum = 3.65e+06
```

Таким образом число 3.65E+6 не изменилось, поскольку для хранения точности отводится только 7 цифр.

Также стоит отметить, что стандарт IEEE, который реализуется компиляторами C++, определяет специальные значения для чисел с плавающей точкой, в которых мантисса на бинарном уровне состоит только из нулей, а экспонента, которая состоит из одних единиц, в зависимости от знака представляет значения `+infinity` (плюс бесконечность +∞) и `-infinity` (минус бесконечность –∞). При делении положительного числа на ноль результатом будет `+infinity`, а при делении отрицательного числа на ноль, соответственно, `-infinity`.

Другое специальное значение с плавающей точкой, определенное этим стандартом, представляет значение `NaN` (Not-a-Number, не число). Это значение представляет результат операции, который не определяется математически, например, когда ноль делится на ноль или бесконечность на бесконечность. Результатом любой операции, в которой один или оба операнда являются `NaN`, также является `NaN`.

Рассмотрим следующую программу:

```
#include <iostream>

int main()
{
 double number1{ 1.5 }, number2{}, number3{}, number4{-1.5};
 double result { number1 / number2 };

 std::cout << number1 << "/" << number2 << " = " << result << std::endl;
 result = number4 / number3;

 std::cout << number4 << "/" << number3 << " = " << result << std::endl;
 result = number2 / number3;

 std::cout << number2 << "/" << number3 << " = " << result << std::endl;

 std::cout << result << " + " << number1 << " = " << result + number1 << std::endl;
}
```

В выражении `number1/number2` число 1.5 делится на 0, поэтому результатом будет плюс бесконечность.

Аналогично в выражении `number2/number3` число –1.5 делится на 0, поэтому результатом будет минус бесконечность.

В выражении `number2/number3` число 0 делится на 0, поэтому результатом будет `NaN`.

Соответственно, последнее выражение `result + number1` будет аналогично `NaN + 0`, соответственно, результат – `NaN`.

Консольный вывод программы показан на рисунке 2.3.

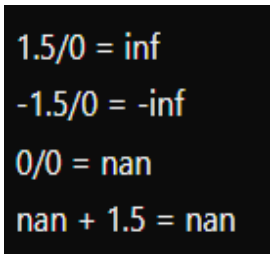


Рисунок 2.3. – Результат выполнения операций с нулем в C++

III. Инкремент и декремент

Две унарные арифметические операции, которые производятся над одним числом, называются ++ (*инкремент*) и -- (*декремент*). Каждая из операций имеет две разновидности: *префиксная* и *постфиксная*. Примеры выражений для операций присваивания приведены в таблице 2.2.

1. Префиксный инкремент

`++x` – увеличивает значение переменной на единицу и полученный результат используется как значение выражения.

```
#include <iostream>

int main()
{
 int number1 {9};
 int number2 {++number1};

 std::cout << "number1 = " << number1 << std::endl; // 10
 std::cout << "number2 = " << number2 << std::endl; // 10
}
```

2. Постфиксный инкремент

`x++` – увеличивает значение переменной на единицу, но значением выражения будет то, которое было до увеличения на единицу.

```
#include <iostream>

int main()
{
 int number1 {9};
 int number2 {number1++};

 std::cout << "number1 = " << number1 << std::endl; // 10
 std::cout << "number2 = " << number2 << std::endl; // 9
}
```

3. Префиксный декремент

`--x` – уменьшает значение переменной на единицу и полученное значение используется как значение выражения.

```
#include <iostream>

int main()
{
 int number1 {9};
 int number2 {--number1};

 std::cout << "number1 = " << number1 << std::endl; // 8
 std::cout << "number2 = " << number2 << std::endl; // 8
}
```

4. Постфиксный декремент

`x--` – уменьшает значение переменной на единицу, но значением выражения будет то, которое было до уменьшения на единицу.

```
#include <iostream>
int main()
{
 int number1 {9};
 int number2 {number1--};
 std::cout << "number1 = " << number1 << std::endl; // 8
 std::cout << "number2 = " << number2 << std::endl; // 9
}
```

Таблица 2.2. – Примеры выражений для операций присваивания

Операции присваивания	Пример	Пояснение	Результат присваивания
Для int c =3, d=5, e=4, f=6, g=12			
+=	c+=7	c = c + 7	c = 10
-=	d-=4	d = d – 4	d = 1
*=	e*=5	e = e * 5	e = 20
/=	f/=3	f = f / 3	f = 2
%=	g%=7	g = g % 7	g = 5

IV. Приоритет операторов

Операторы могут быть *левоассоциативными* (выполняются слева направо) и *правоассоциативными* (выполняются справа налево). Подавляющее большинство операторов левоассоциативны (например, бинарные арифметические операции), поэтому большинство выражений оценивается слева направо. Правоассоциативными операторами являются все унарные операторы, различные операторы присваивания и условный оператор.

Следует учитывать, что одни операции имеют больший приоритет, чем другие, и поэтому выполняются вначале. Операции в порядке уменьшения приоритета:

++ (инкремент), -- (декремент)

* (умножение), / (деление), % (остаток от деления)

+ (сложение), - (вычитание)

Приоритет операций следует учитывать при выполнении набора арифметических выражений:

```
int number1 = 9;
int number2 = 6;
int number3 = number1 + 5 * ++number2; // 39
```

Хотя операции выполняются слева направо, но вначале будет выполняться операция инкремента `++number2`, которая увеличит значение переменной `number2` и возвратит его в качестве результата, так как эта операция имеет больший приоритет. Затем выполняется умножение `5*++number2`, и только в последнюю очередь выполняется сложение `number1 + 5 * ++b`.

Следует учитывать, что если в одной инструкции для одной переменной сразу несколько раз вызываются операции инкремента и декремента, то результат может быть неопределенным и многое зависит от конкретного компилятора. Например:

```
int count {1};
int result = ++count * 3 + count++ * 5;
```

Так, и `g++`, и `clang++` скомпилируют данный код, и результат переменной `result` будет таким, как, в принципе, и ожидается – `16`, но компилятор `clang++` также сгенерирует предупреждение.

V. Переопределение порядка операций

Скобки позволяют переопределить порядок вычислений. Например:

```
#include <iostream>

int main()
{
 int number1 {5};
 int number2 {6};
 int number3 {(number1 + 9) * ++number2}; // 98
 std::cout << "number3 = " << number3 << std::endl;
}
```

Несмотря на то что операция сложения имеет меньший приоритет, вначале будет выполняться именно сложение, а не умножение, так как операция сложения заключена в скобки.

Ход работы

1. Ознакомиться с теоретическим материалом и закрепить знания подготовкой ответов на контрольные вопросы.

2. Написать программу вычислений в соответствии с заданным вариантом (числовые параметры задаются самостоятельно вводом с клавиатуры).

3. Значение переменной вводить с консоли, при этом на экран консоли вывести наименование вводимой переменной. Результаты вычислений выводить на консоль, сопровождая их наименованиями выводимых значений.

4. Объяснить полученные результаты вычислений.

5. Оформить отчет.

6. Загрузить отчет на образовательную платформу.

7. Защитить работу у преподавателя. Для этого требуется знать ответы на теоретические вопросы, уметь написать код для аналогичной задачи без использования сторонних источников.

Таблица 2.3. – Варианты заданий

№ варианта	Задание 1	Задание 2
1	*Сложить два значения:* float number1{ 4.568E–2 }; float number2{ 7.3E+7 }; *Перемножить два значения:* float number1{ 4.568E–2 }; float number2{ 7.3E+7 }; *Вычесть number2 – number1:* float number1{ 14.5E–3 }; float number2{ 7.3E+6 }; *Поделить number2 на number1:* float number1{ 14.5E–3 }; float number2{ 7.3E+6 };	*Сложить два значения:* double number1{ 4.568E–2 }; double number2{ 7.3E+7 }; *Перемножить два значения:* double number1{ 4.568E–2 }; double number2{ 7.3E+7 }; *Вычесть number2 – number1:* double number1{ 14.5E–3 }; double number2{ 7.3E+6 }; *Поделить number2 на number1:* double number1{ 14.5E–3 }; double number2{ 7.3E+6 };
2	*Сложить два значения:* float number1{ 7.65E–4 }; float number2{ 5.67E+5 }; *Вычесть number2 – number1:* float number1{ 27.5E–3 }; float number2{ 9.3E+5 }; *Перемножить два значения:* float number1{ 7.65E–4 }; float number2{ 5.67E+5 }; *Поделить number2 на number1:* float number1{ 27.5E–3 }; float number2{ 9.3E+5 };	*Сложить два значения:* double number1{ 7.65E–4 }; double number2{ 5.67E+5 }; *Вычесть number2 – number1:* double number1{ 27.5E–3 }; double number2{ 9.3E+5 }; *Перемножить два значения:* double number1{ 7.65E–4 }; double number2{ 5.67E+5 }; *Поделить number2 на number1:* double number1{ 27.5E–3 }; double number2{ 9.3E+5 };
3	*Сложить два значения:* float number1{ 76.5E–4 }; float number2{ 7.3E+5 }; *Вычесть number2 – number1:* float number1{ 33.3E–3 };	*Сложить два значения:* double number1{ 76.5–4 }; double number2{ 7.3E+5 }; *Вычесть number2 – number1:* double number1{ 33.3E–3 };

№ варианта	Задание 1	Задание 2
	float number2{ 10.9E+4 }; *Перемножить два значения:* float number1{ 76.5E–4 }; float number2{ 7.3E+5 }; *Поделить number2 на number1:* float number1{ 33.3E–3 }; float number2{ 10.9E+4 };	double number2{ 10.9E+4 }; *Перемножить два значения:* double number1{ 76.5–4 }; double number2{ 7.3E+5 }; *Поделить number2 на number1:* double number1{ 33.3E–3 }; double number2{ 10.9E+4 };
4	*Сложить два значения:* float number1{ 118.8E–2 }; float number2{ 1.0003E+5 }; *Вычесть number2 – number1:* float number1{ 52.5E–6 }; float number2{ 76.38 }; *Перемножить два значения:* float number1{ 118.8E–2 }; float number2{ 1.0003E+5 }; *Поделить number2 на number1:* float number1{ 52.5E–6 }; float number2{ 76.38 };	*Сложить два значения:* double number1{ 118.8E–2 }; double number2{ 1.0003E+5 }; *Вычесть number2 – number1:* double number1{ 52.5E–6 }; double number2{ 76.38 }; *Перемножить два значения:* double number1{ 118.8E–2 }; double number2{ 1.0003E+5 }; *Поделить number2 на number1:* double number1{ 52.5E–6 }; double number2{ 76.38 };
5	*Сложить два значения:* float number1{ 32.568E–1 }; float number2{ 7.0001E+6 }; *Вычесть number2 – number1:* float number1{ 11.02E–3 }; float number2{ 45.9E+3 }; *Перемножить два значения:* float number1{ 32.568E–1 }; float number2{ 7.0001E+6 }; *Поделить number2 на number1:* float number1{ 11.02E–3 }; float number2{ 45.9E+3 };	*Сложить два значения:* double number1{ 32.568E–1 }; double number2{ 7.0001E+6 }; *Вычесть number2 – number1:* double number1{ 11.02E–3 }; double number2{ 45.9E+3 }; *Перемножить два значения:* double number1{ 32.568E–1 }; double number2{ 7.0001E+6 }; *Поделить number2 на number1:* double number1{ 11.02E–3 }; double number2{ 45.9E+3 };
6	*Сложить два значения:* float number1{ 21.568E–4 }; float number2{ 4.206E+5 }; *Вычесть number2 – number1:* float number1{ 62.0507E–3 }; float number2{ 11.2008E+4 }; *Перемножить два значения:* float number1{ 21.568E–4 }; float number2{ 4.206E+5 }; Поделить number2 на number1: float number1{ 62.0507E–3 }; float number2{ 11.2008E+4 };	*Сложить два значения:* double number1{ 21.568E–4 }; double number2{ 4.206E+5 }; *Вычесть number2 – number1:* double number1{ 62.0507E–3 }; double number2{ 11.2008E+4 }; *Перемножить два значения:* double number1{ 21.568E–4 }; double number2{ 4.206E+5 }; Поделить number2 на number1: double number1{ 62.0507E–3 }; double number2{ 11.2008E+4 };

№ варианта	Задание 1	Задание 2
7	Сложить два значения: float number1{ 34.111 E–3 }; float number2{ 9.045E+5 }; Вычесть number2 – number1: float number1{ 27.99E–5 }; float number2{ 66.7034+3 }; Перемножить два значения: float number1{ 34.111 E–3 }; float number2{ 9.045E+5 }; Поделить number2 на number1: float number1{ 27.99E–5 }; float number2{ 66.7034+3 };	Сложить два значения: double number1{ 34.111E–3 }; double number2{ 9.045E+5 }; Вычесть number2 – number1: double number1{ 27.99E–5 }; double number2{ 66.7034E+3 }; Перемножить два значения: double number1{ 34.111E–3 }; double number2{ 9.045E+5 }; Поделить number2 на number1: double number1{ 27.99E–5 }; double number2{ 66.7034E+3 };
8	Сложить два значения: float number1{ 41.00208E–2 }; float number2{ 39.008E+5 }; Вычесть number2 – number1: float number1{ 104.007E–2 }; float number2{ 82.06E+5 }; Перемножить два значения: float number1{ 41.00208E–2 }; float number2{ 39.008E+5 }; Поделить number2 на number1: float number1{ 104.007E–2 }; float number2{ 82.06E+5 };	Сложить два значения: double number1{ 41.00208E–2 }; double number2{ 39.008E+5 }; Вычесть number2 – number1: double number1{ 104.007E–2 }; double number2{ 82.06E+5 }; Перемножить два значения: double number1{ 41.00208E–2 }; double number2{ 39.008E+5 }; Поделить number2 на number1: double number1{ 104.007E–2 }; double number2{ 82.06E+5 };
9	Сложить два значения: float number1{ 79.99E–3 }; float number2{ 9.73E+5 }; Вычесть number2 – number1: float number1{ 39.505E–3 }; float number2{ 0.7389E+8 }; Перемножить два значения: float number1{ 79.99E–3 }; float number2{ 9.73E+5 }; Поделить number2 на number1: float number1{ 39.505E–3 }; float number2{ 0.7389E+8 };	Сложить два значения: double number1{ 79.99E–3 }; double number2{ 9.73E+5 }; Вычесть number2 – number1: double number1{ 39.505E–3 }; double number2{ 0.7389E+8 }; Перемножить два значения: double number1{ 79.99E–3 }; double number2{ 9.73E+5 }; Поделить number2 на number1: double number1{ 39.505E–3 }; double number2{ 0.7389E+8 };
10	Сложить два значения: float number1{ 28.017 }; float number2{ 1.82E+9 }; Вычесть number2 – number1: float number1{ 14.5E–4 }; float number2{ 902.783E+2 }; Перемножить два значения:	Сложить два значения: double number1{ 28.017 }; double number2{ 1.82E+9 }; Вычесть number2 – number1: double number1{ 14.5E–4 }; double number2{ 902.783E+2 }; Перемножить два значения:

№ варианта	Задание 1	Задание 2
	float number1{ 28.017 }; float number2{ 1.82E+9 }; Поделить number2 на number1: float number1{ 14.5E–4 }; float number2{ 902.783E+2 };	double number1{ 28.017 }; double number2{ 1.82E+9 }; Поделить number2 на number1: double number1{ 14.5E–4 }; double number2{ 902.783E+2 };

Контрольные вопросы

1. Какие арифметические операции выполняются над числами в языке C++? Поясните особенности разных типов, приведите примеры.

2. Назовите вид типизации языка C++. Дайте пояснение с примером.

3. Сформулируйте правило представления вещественных чисел в компьютере. Приведите пример.

4. Представьте и поясните форматы одинарной и двойной точности нормализированного числа.

5. Какие особенности следует учитывать при работе с числами с плавающей точкой? Приведите примеры вычислений для нормализованных чисел.

6. В каких случаях могут быть получены значения: `+infinity`, `–infinity`, `Not-a-Number`? Что они обозначают? Приведите примеры.

7. Какие унарные арифметические операции выполняются над числами в языке C++? Дайте характеристику их разновидностям и приведите примеры.

8. Каков приоритет выполнения арифметических операций на C++?

9. Как можно переопределить порядок операций? Приведите пример.

10. Напишите код без сторонних источников для выполнения арифметических операций с вводом и выводом данных через консоль.

ЛАБОРАТОРНАЯ РАБОТА № 3
Реализация линейных алгоритмов

Цель: получить навыки составления и программной реализации простых линейных алгоритмов на языке программирования C++ с использованием стандартной математической библиотеки.

Теоретический материал

Понятие алгоритма является одним из ключевых в вычислительной математике и программировании. *Алгоритм* – это понятная и конечная последовательность точных действий над некоторыми объектами для достижения конкретной цели либо для решения конкретной задачи или группы задач за конечное число шагов [7].

Применительно к созданию программного обеспечения формальный исполнитель алгоритма – это компьютер. Поэтому системой команд является совокупность допустимых команд используемого языка программирования.

Процесс разработки алгоритма называется алгоритмизацией поставленной задачи. Следует учитывать следующие свойства алгоритма:

1. *Понятность* – каждая команда должна быть понятна исполнителю, т. е. включена в его систему команд.

2. *Дискретность* – последовательность действий представляется в виде конечного количества этапов.

3. *Определенность* – каждое правило алгоритма должно быть четким, однозначным. Также однозначно должен быть определен порядок выполнения шагов.

4. *Результативность* – при точном исполнении алгоритма процесс должен прекратиться за конечное число шагов и с определённым результатом. Причем вывод о том, что решения не существует, также является результатом.

5. *Массовость* – алгоритм должен решать любую задачу из того класса задач, для решения которых он разработан, или с различным набором исходных данных.

Линейным называется алгоритм, в котором результат получается путем однократного выполнения заданной последовательности действий при любых значениях исходных данных. Операторы программы выполняются последовательно, один за другим, в соответствии с их расположением в программе [9].

Для выполнения программы на C++, предполагающей вычисление разных математических функций, можно использовать библиотеки `cmath` или `math.h` путем включения их в программу: `#include <cmath>` или `#include <math.h>`.

Следует отметить, что данные библиотеки в общем характеризуются достаточно схожим набором функций, однако все же есть некоторые отличия между

ними. Библиотека cmath включена в стандартную библиотеку C++, а math.h является заголовочным файлом стандартной библиотеки языка программирования C, а соответственно, и C++. Одним из основных преимуществ cmath является возможность работы с комплексными числами.

Некоторые функции стандартной математической библиотеки cmath:

- fabs(x) – вычисление модуля значения x;
- ceil(x) – округление x до наибольшего целого значения;
- floor(x) – округление до наименьшего целого значения;
- round(x) – округление до ближайшего целого значения;
- fmod(x) – остаток от деления числителя на знаменатель;
- pow(x,y) – возведение числа x в степень y;
- pow(x,1/y) – вычисление корня заданной степени y из числа x. Например, pow(8,1.0/3), позволит вычислить корень третьей степени из числа 8. Следует учитывать правило записи выражения для деления чисел в C++, рассмотренное в предыдущей работе: чтобы результат деления чисел был числом с плавающей точкой необходимо, чтобы хотя бы одно из чисел было записано числом с плавающей точкой, для данного примера как 1.0 или 3.0;
- sqrt(x) – вычисление квадратного корня числа x;
- exp(x) – вычисление экспоненты, т. е. e^x;
- frexp(x) – вычисляет мантиссу M (типа double, большее или равное 0.5 и меньшее 1.0) и целое n, такое, что $x = M*2^n$. Функция frexp сохраняет n в целой переменной, на которую указывает exponent.
- log(x) – натуральный логарифм от x;
- log10(x) – десятичный логарифм от x;
- modf(x) – разделение вещественного значения на дробную и целую части;
- cosh(x) – вычисление гиперболического косинуса от x;
- sinh(x) – вычисление гиперболического синуса от x;
- tanh(x) – вычисление гиперболического тангенса от x;
- cos(x) – вычисление косинуса угла (x в радианах);
- sin(x) – вычисление синуса угла (x в радианах);
- tan(x) – вычисление тангенса угла (x в радианах);
- acos(x) – вычисление арккосинуса (результат в радианах);
- asin(x) – вычисление арксинуса (результат в радианах);
- atan(x) – вычисление арктангенса (результат в радианах);
- atan2(x) – вычисление арктангенса и квадранта по координатам x и y (результат в радианах).

Пример использования математических функций библиотеки C++:

```
#include <iostream>
#include <cmath>  // подключение библиотеки

using namespace std;

 int main()
{
// вычисление модуля значения
  cout << "Модуль числа (-121) = " << fabs(-121)<< "\n";

// возведение числа (-2) в степень (3)
  cout << "Возведение в степень (-2, 3) = " << pow(-2, 3)<< "\n";

//округление до наименьшего целого значения
  cout << " Наименьшее целое значение (4.3) = " << floor(4.3)<< "\n";

// округление до наибольшего целого значения
  cout << " Наибольшее целое значение (4.3) = " << ceil(4.3)<< "\n";

//округление до ближайшего целого значения
  cout << " Ближайшее целое значение (-4.3) = " << round(-4.3)<< "\n";

//округление до наименьшего целого значения
  cout << " Наименьшее целое значение (-4.3) = " << floor(-4.3)<< "\n";

//округление до ближайшего целого значения
  cout << " Наибольшее целое значение (-4.3) = " << ceil(-4.3)<< "\n";

  return 0;
}
```

Пример. Составить алгоритм и написать код для вычисления значения функции

$$y = -(2^5 \times \sqrt[3]{a \times 5} + \log_{10} b)$$

с выводом в консоль результата под корнем, значения функции и его модуля. Учитывать, что переменные `a` и `b` могут быть только целыми числами.

Шаги алгоритма:

1) задание значения `a`;
2) задание пользователем значения `b` с учетом того, что оно не должно быть равно нулю;
3) вычисление значения под корнем;
4) вычисление значения функции;
5) вычисление модуля от величины, полученной на шаге 4;
6) вывод полученных результатов.

С учетом описанного алгоритма и изученных правил написания программ на C++ код будет иметь следующий вид:

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

using namespace std;

int main( )
{
int a, b;
float y, z, yabs;

cout<<"Введите значение a "<<'\n';
cin>>a; // ввод значения переменной a
cout<<"Введите значение b, которое должно быть отлично от нуля "<<'\n';
cin>>b; // ввод значения переменной b

z=pow(a*5,1.0/3); // вычисление значения под корнем
y=-(pow(2,5) * z + log10(b)); //вычисление значения y
yabs=fabs(y);

cout<<"Значение под корнем = "<<z<<'\n'; //вывод значения y
cout<<"Вычисленное значение функции y = "<<y<<'\n'; //вывод значения y
cout<<"Модуль значения y = "<<yabs<<'\n'; //вывод модуля

return 0;
}
```

Результат выполнения приведенного выше кода при заданных a = 2 и b = 100 показан на рисунке 3.1.

```
Введите значение a
2
Введите значение b, которое должно быть отлично от нуля
100
Значение под корнем  = 2.15443
Вычисленное значение функции y = -70.9419
Модуль значения y = 70.9419


...Program finished with exit code 0
Press ENTER to exit console.
```

Рисунок 3.1. – Результат выполнения программы для приведенного выше примера кода программы

Ход работы

1. Ознакомиться с теоретическим материалом и закрепить знания подготовкой ответов на контрольные вопросы.
2. Получить у преподавателя вариант индивидуального задания (таблица 3.1).
3. Разработать и записать алгоритм решения задачи.
4. Составить текст программы.
5. Значение переменной вводить с консоли, при этом на экран консоли вывести наименование вводимой переменной. Результаты вычислений выводить на консоль, сопровождая их наименованиями выводимых значений.
6. Проанализировать результаты работы выполнения программы. Убедиться в правильности решения задачи, обосновать свой вывод о правильности работы программы, результат привести в отчете.
7. Оформить отчет.
8. Загрузить отчет на образовательную платформу.
9. Защитить работу у преподавателя. Для этого требуется знать ответы на теоретические вопросы, уметь написать код для аналогичной задачи без использования сторонних источников.

Задание 1

В соответствии с вариантом составить алгоритм и написать код для вычисления значения функции.

Таблица 3.1. – Варианты к заданию 1

№ варианта	Функция	Дополнительное задание
1	$y = \sqrt{7^8 \times \sqrt[3]{a \times 3} + \cos b}$	Вычислить модуль полученного значения функции
2	$y = -(8^3 \times \sqrt[7]{b \times 9} + \sin a)$	Округлить полученное значение функции до наибольшего целого значения
3	$y = \frac{2^5}{\sqrt[3]{b \times 12} + \log_{10} a}$	Округлить полученное значение функции до наименьшего целого значения
4	$y = \frac{8}{5} \times 2^7 \times \sqrt[3]{a \times 5} + \log_{10} b$	Округлить полученное значение функции до ближайшего целого значения
5	$y = \left(\frac{\sqrt[6]{\frac{a}{b}} + \cos a}{3^5} \right)^3$	Вычислить остаток от деления результата функции

№ варианта	Функция	Дополнительное задание
6	$y = \left(\frac{\sqrt[6]{\frac{a}{5}} + \cos b}{2^6 \times \sin a} \right)^9$	Разделить результат функции на дробную и целую части
7	$y = -\left(5^2 \times \sqrt[3]{b \times \frac{5}{6}} + \log_{10} a + \cos a \right)$	Вычислить модуль полученного значения функции
8	$y = \left(a^2 \times \sqrt[4]{b \times \frac{a}{8}} + \log_{10} b + \cos b \right)^3$	Округлить полученное значение функции до наибольшего целого значения
9	$y = \sin a \times \cos b - (2^8 \times \sqrt[3]{a \times 5})$	Округлить полученное значение функции до наименьшего целого значения
10	$y = \tan a^b \times (3^8 - \sqrt[3]{a \times \log_{10} b} - 9)$	Округлить полученное значение функции до ближайшего целого значения

Задание 2

Таблица 3.2. – Варианты к заданию 2

№ варианта	Задание
1	Вычислить периметр прямоугольника
2	Вычислить площадь прямоугольника
3	Вычислить площадь треугольника
4	Вычислить высоту параллелепипеда
5	Вычислить площадь круга
6	Вычислить периметр треугольника
7	Вычислить длину окружности
8	Вычислить высоту параллелепипеда
9	Вычислить гипотенузу в прямом треугольнике
10	Вычислить высоту прямоугольника

Контрольные вопросы

1. Дайте определение понятию «алгоритм». Каково основное назначение алгоритма?
2. Сформулируйте свойства алгоритма и поясните каждое из них.
3. Какой алгоритм будет линейным?
4. Каким образом можно ускорить написание кода программы при математических вычислениях?

5. Какую функцию можно использовать для вычисления квадратного корня в C++? Это единственный вариант или нет?

6. Перечислите функции стандартной математической библиотеки `cmath`.

7. Назовите отличия функций `cmath` и `math.h`.

8. Запишите строку кода для вычисления корня пятой степени числа `x` с использованием стандартной библиотеки.

9. Запишите примеры использования библиотеки C++ для вычисления тригонометрических функций.

10. Запишите пример кода программы для вычисления гипотенузы прямоугольного треугольника с вводом данных с консоли и выводом результата в консоль.

ЛАБОРАТОРНАЯ РАБОТА № 4
Условные операторы и операторы сравнения

Цель: получение практических навыков при программировании разветвляющихся алгоритмов с использованием условных операторов и операторов сравнения.

Теоретический материал

I. Операторы сравнения

Это специальные символы или комбинации символов, которые позволяют сравнивать значения переменных и получать результат в виде логического значения в зависимости от выполнения условия сравнения.

В C++ общий вид операции сравнения может быть представлен так:

```
<выражение 1> <оператор> <выражение 2>
```

Пример реализации:

```
int numberA = 3, numberB = 11;
if (numberA < numberB) // использование оператора сравнения «<»
{
//// Код, который выполнится, если a меньше b
     std::cout << "a меньше b";
}
else
{
//// Код, который выполнится, если a больше b
     std::cout << "a не меньше b";
}
```

В качестве выражений могут выступать любые базовые типы. Значения выражений перед сравнением преобразуются к одному типу. Результатом операции сравнения является либо значение 1 (`true`), если отношение истинно, либо значение 0 (`false`), если отношение ложно. Операция сравнения может использоваться в любых арифметических выражениях.

Таблица 4.1. – Виды операторов сравнения

Оператор	Описание	Пример
==	равенство	(a == b)
!=	неравенство	(a != b)
>=	больше или равно	(a >= b)
<=	меньше или равно	(a <= b)
>	больше	(a > b)
<	меньше	(a < b)

В C++, операторы сравнения имеют разные приоритеты, определяющие порядок их выполнения в выражениях. Приоритет выполнения операторов сравнения определяется следующим образом:

– сначала выполняются операторы "!=" (не равно) и "==" (равно);

– затем, операторы "<" (меньше), "<=" (меньше или равно), ">" (больше) и ">=" (больше или равно).

При сравнении выражений с использованием нескольких операторов сравнения в одном выражении операторы будут выполняться согласно их приоритету. Если приоритет неясен из–за сложности выражения, то можно использовать скобки для явного указания порядка выполнения операций.

Таким образом, при написании выражений с операторами сравнения важно помнить об их приоритетах, чтобы избежать ошибок и получить корректный результат сравнения.

II. Условные операторы

Это операторы, позволяющие выполнять определенный блок кода в зависимости от условия. Они позволяют программе принимать решения на основе сравнения значений переменных или выражений. Основные условные операторы:

– `if` – используется для выполнения блока кода, если указанное условие истинно;

– `else` – используется в паре с `if`, чтобы выполнить блок кода, если условие оператора `if` ложно.

– `else if` – позволяет проверить дополнительные условия, если предыдущие условия были ложными, и выполнить соответствующий блок кода.

Условный оператор `if` (если) позволяет разветвлять выполнение программы в зависимости от логических величин, т. е. результатов работы операторов сравнения и логических переменных.

```
if (логическая величина)
{
// выполняется, если лог. величина – true
}
```

Оператор `else` (иначе) работает в паре с оператором `if` и позволяет предусмотреть действие на случай невыполнения `if`:

```
if (логическая величина)
{
// выполняется, если логическая величина – true
}

else
```

```
{
// выполняется, если логическая величина – false
}
```

Также существует третья конструкция `else if`, позволяющая ещё больше разветвить код:

```
if (логическая величина 1)
{
// выполняется, если логическая величина 1 – true
}

else if (логическая величина 2)
{
// выполняется, если логическая величина 2 – true
}

else
{
// выполняется иначе
}
```

Пример:

```
// при выполнении одного действия
// внутри условия {} не обязательны
// если number1 больше number2, то number3 = 15
if (number1 > number2) number3 = 15;

else number3 = 30;    // если нет, то number3 = 30

// при выполнении двух и более действий
// внутри условия {} обязательны!
if (number1 > number2)
{
 number3 = 10;
 number2 = number3;
}

else
{
 number3 = 20;
 number2 = number1;
}
```

Код, расположенный внутри блока `{}`, называется телом оператора `if`. Если условие является истинным, то код внутри тела оператора `if` будет выполнен. Если условие является ложным, то код внутри тела оператора `if` будет пропущен, и выполнение программы перейдет к следующим строкам кода.

Рассмотрим пример сочетания операторов `if`, `else if` и `else`, который определяет является введенное пользователем число положительным, или отрицательным:

```
#include <iostream>

int main()
{
 int число;
 std::cout << "Введите число: ";
 std::cin >> число;

 if (число > 0)
 {
   std::cout << "Число положительное" << std::endl;
 }

 else if (число < 0)
 {
   std::cout << "Число отрицательное" << std::endl;
 }

 else
 {
   std::cout << "Число равно нулю" << std::endl;
 }

 return 0;
}
```

В этом примере, если число больше нуля, выводится сообщение "Число положительное". Если число меньше нуля, выводится сообщение "Число отрицательное". В противном случае, если число равно нулю, выводится сообщение "Число равно нулю".

Таким образом, оператор `if` позволяет управлять программой и создавать разветвлённые действия в зависимости от разных условий.

Ход работы

1. Ознакомиться с теоретическим материалом и закрепить знания подготовкой ответов на контрольные вопросы

2. Написать программу вычислений в соответствии с заданным вариантом (числовые параметры задаются самостоятельно вводом с клавиатуры).

3. Значение переменной вводить с консоли, при этом на экран консоли вывести наименование вводимой переменной. Результаты вычислений выводить на консоль, сопровождая их наименованиями выводимых значений.

4. Объяснить полученные результаты вычислений.
5. Оформить отчет.
6. Загрузить отчет на образовательную платформу.
7. Защитить работу у преподавателя. Для этого требуется знать ответы на теоретические вопросы, уметь написать код для аналогичной задачи без использования сторонних источников.

Задание

При помощи операторов сравнения и оператора `if` выполните задания в соответствии с вариантом из таблицы 4.2.

Таблица 4.2. – Варианты заданий

№ варианта	Задания
1	1. Напишите программу, которая определяет, является ли введенное число четным, нечетным или равным нулю. 2. Напишите программу, которая проверяет, является ли треугольник, заданный тремя сторонами, равнобедренным
2	1. Напишите программу, которая проверяет, является ли треугольник, заданный тремя сторонами, прямоугольным. 2. Напишите программу, которая определяет, является ли введенный пользователем год годом столетия
3	1. Напишите программу, которая возвращает минимальное из трех чисел. 2. Напишите программу, которая определяет, в какой четверти координатной плоскости находится точка с заданными координатами
4	1. Напишите программу, которая возвращает наибольшее из четырех чисел. 2. Напишите программу, которая определяет, является ли треугольник, заданный тремя сторонами, равносторонним
5	1. Напишите программу, которая запрашивает у пользователя день недели и выводит сообщение о том, будний это день или выходной. 2. Напишите программу, которая определяет, является ли введенное шестизначное число палиндромом (число одинаково читающееся в обоих направлениях, например, 123321)
6	1. Напишите программу, которая определяет, является ли введенное число положительным, отрицательным или равным нулю. 2. Напишите программу, которая проверяет, является ли введенный номер дня и месяца корректным (от 1 до 12 для месяца, и от 1 до 31 для дня)
7	1. Напишите программу, которая определяет, является ли введенная буква гласной или согласной. 2. Напишите программу, которая определяет, какое время года наступило, по введенному пользователем месяцу
8	1. Напишите программу, которая определяет, является ли введенный год високосным. 2. Напишите программу, которая сравнивает два числа, введенных пользователем, и сообщает, какое из них больше и на сколько

№ варианта	Задания
9	1. Напишите программу, которая определяет, входит ли число в заданный пользователем диапазон. 2. Напишите программу, которая сортирует в порядке возрастания пять введенных чисел
10	1. Напишите программу, которая запрашивает у пользователя его возраст и выводит сообщение о том, можно ли ему получить права. Если нет, то сколько лет осталось ждать. 2. Напишите программу, которая проверяет, является ли введенное число четным или нечетным, положительным или отрицательным

Контрольные вопросы

1. В каких случаях необходимо использовать условные операторы и операторы сравнения? Приведите примеры.
2. Дайте определение операторам сравнения.
3. Запишите общий вид операции сравнения в C++.
4. Дайте определение термину «условные операторы».
5. Перечислите операторы сравнения, охарактеризуйте их и приведите примеры.
6. Поясните, как определяется приоритет выполнения операторов сравнения?
7. Какие условные операторы используются в C++ и для каких задач?
8. Как работает оператор `if`? Приведите пример.
9. Как работает оператор `else`? Приведите пример.
10. Как работает оператор `else if`? Приведите пример.

ЛАБОРАТОРНАЯ РАБОТА № 5
Реализация разветвляющихся алгоритмов при помощи оператора `switch`

Цель: получение практических навыков при программировании разветвляющихся алгоритмов с использованием условного оператора `switch`.

Теоретический материал

Разветвляющиеся алгоритмы – это алгоритмы, которые в зависимости от определенных условий выбирают тот или иной путь выполнения. Они позволяют программе принимать решения на основе входящих данных и выполнять различные действия в зависимости от этих данных. Разветвляющиеся алгоритмы могут быть реализованы с помощью условных операторов, таких как `if-else` или `switch`.

Условный оператор `switch` – это оператор, который используется для выбора одного из нескольких вариантов выполнения кода в зависимости от значения выражения. Он позволяет сравнить выражение со списком значений и выполнить определенные инструкции для соответствующего варианта.

Можно выделить основные отличия оператора `switch` от других условных операторов: значение выражения `switch` сравнивается только с константами, в отличие от оператора `if`, который допускает разные сравнения или любые логические выражения; две константы не могут иметь одинаковых значений в различных разделах `case`, только в случае когда один оператор `switch` вложен в другой; используемые символьные константы в `switch` автоматически преобразовываются в целочисленные.

Общий синтаксис оператора `switch` выглядит следующим образом:

```
switch (выражение)
{
 case значение1:
  // код, который будет выполнен, если выражение равно значению1
  break;
 case значение2:
  // код, который будет выполнен, если выражение равно значению2
  break;
 case значение3:
  // код, который будет выполнен, если выражение равно значению3
  break;
 // и так далее
 default:
```

```
      // код, который будет выполнен, если выражение не равно ни од-
ному из значений
      break;
    }
```

В блоке `case` указывается значение, с которым будет сравниваться выражение. Если выражение равно значению, выполняется код, находящийся под этим блоком `case`. Оператор `break` используется для выхода из оператора `switch` и завершения его выполнения.

Блок `default` указывает код, который будет выполнен, если выражение не равно ни одному из значений, указанных в блоках `case`.

Оператор `switch` это удобный способ заменить множество вложенных конструкций `if-else`, если необходимо выполнить различные действия в зависимости от значения одной переменной. Однако необходимо помнить, что в операторе `switch` можно использовать только константные выражения, а действия в `case` должны завершаться оператором `break`, иначе управление будет передано следующему блоку кода.

Пример программы на C++, использующей оператор `switch`:

```
#include <iostream>

int main()
{
 int оценка;
 std::cout << "Введите оценку (от 1 до 5): ";
 std::cin >> оценка;

 switch (оценка)
 {
  case 1:
   std::cout << "Очень плохо" << std::endl;
   break;
  case 2:
   std::cout << "Плохо" << std::endl;
   break;
  case 3:
   std::cout << "Удовлетворительно" << std::endl;
   break;
  case 4:
   std::cout << "Хорошо" << std::endl;
   break;
  case 5:
   std::cout << "Отлично" << std::endl;
   break;
  default:
```

```
        std::cout << "Ошибка: введена некорректная оценка" <<
std::endl;
        break;
      }
     return 0;
     }
```

В этом примере пользователю предлагается ввести оценку от 1 до 5. Затем оператор `switch` сравнивает введенное значение с каждым блоком `case` и выполняет соответствующий код. Если введенное значение не соответствует ни одному из блоков `case`, то будет выполнен код в блоке `default` и будет выведено сообщение об ошибке.

Ход работы

1. Ознакомиться с теоретическим материалом и закрепить знания подготовкой ответов на контрольные вопросы.
2. Написать программу вычислений в соответствии с заданным вариантом (числовые параметры задаются самостоятельно вводом с клавиатуры).
3. Значение переменной вводить с консоли, при этом на экран консоли вывести наименование вводимой переменной. Результаты вычислений выводить на консоль, сопровождая их наименованиями выводимых значений.
4. Объяснить полученные результаты вычислений.
5. Оформить отчет.
6. Загрузить отчет на образовательную платформу.
7. Защитить работу у преподавателя. Для этого требуется знать ответы на теоретические вопросы, уметь написать код для аналогичной задачи без использования сторонних источников.

Задание

При помощи оператора `switch` выполните задания в соответствии с вариантом из таблицы 5.1.

Таблица 5.1. – Варианты заданий

№ варианта	Задание
1	Напишите программу «калькулятор», которая считывает код операции (1 – добавление, 2 – вычитание, 3 – умножение, 4 – деление) и два операнда, а затем выполняет выбранную операцию
2	Напишите программу, которая запрашивает у пользователя один из трех операторов (+, –, *) и два числа, а затем выполняет выбранную операцию

№ варианта	Задание
3	Напишите программу, которая запрашивает у пользователя фигуру (круг, квадрат, треугольника) и вычисляет ее площадь
4	Напишите программу, которая запрашивает у пользователя номер месяца и выводит соответствующее ему время года
5	Напишите программу, которая запрашивает у пользователя первую букву месяца и выводит название этого месяца
6	Напишите программу, которая запрашивает у пользователя номер месяца и год, а затем выводит количество дней в этом месяце
7	Напишите программу, которая запрашивает у пользователя номер дня недели и номер месяца, а затем выводит дату в формате "День недели, число месяца"(Пример: день недели – 2, номер месяца – 3. Вывод: Вторник, март)
8	Напишите программу, которая запрашивает у пользователя длины трех сторон треугольника и определяет, является ли он равносторонним, равнобедренным или разносторонним
9	Напишите программу, которая запрашивает у пользователя фигуру (круг, квадрат, треугольник) и рассчитывает периметр выбранной фигуры
10	Напишите программу, которая запрашивает у пользователя шкалу температуры (Цельсия, Фаренгейта, Кельвина) и конвертирует введенное значение в выбранную шкалу

Контрольные вопросы

1. Какие алгоритмы называют разветвляющимися?
2. С помощью каких операторов могут быть реализованы разветвляющиеся алгоритмы?
3. Сформулируйте и поясните основные особенности оператора `switch`.
4. Запишите и поясните общий синтаксис (формат записи) оператора `switch`.
5. Для чего используется оператор `break`?
6. Для чего предназначен блок `default`?
7. В каких случаях следует использовать оператор `switch`?
8. Запишите код, использующей оператор `switch`, для решения поставленной задачи. Объясните код.

ЛАБОРАТОРНАЯ РАБОТА № 6
Логические операции и булевы функции в программировании

Цель: изучить основные логические операции и научиться применять булевы функции при написании программ на языке C++.

Теоретические сведения

I. Логические операции

Логические операции – это операции, позволяющие создавать сложные логические выражения и принимать решения на основе их истинности или ложности. Они широко используются в условных операторах, циклах и других конструкциях программы.

Для объявления и инициализации булевых переменных в C++ используется следующий синтаксис:

```
    bool isTrue = true;
    bool isFalse = false;
```

В приведенном примере переменная `isTrue` инициализируется значением `true`, а переменная `isFalse` – значением `false`.

Основными логическими операциями являются следующие три:

– `И (AND)` – возвращает `true`, если оба операнда истинны, и `false` в противном случае. В C++ оператор для логического `И` – это двойной амперсанд (&&). Например:

```
    bool a = true;
    bool b = false;
    bool result = a && b; // result будет равен false, так как один
из операндов (b) является ложным
```

– `ИЛИ (OR)` – это бинарная операция, которая возвращает `true`, если хотя бы один из операндов является истинным (`true`). Если оба операнда являются ложными (`false`), то результат будет `false`. В C++ оператор для логического `ИЛИ` – это две вертикальные линии (||). Например:

```
    bool a = true;
    bool b = false;
    bool result = a || b; // result будет равен true, так как один
из операндов (a) является истинным
```

– НЕ (NOT) – это унарная операция, которая инвертирует значение операнда. Если операнд равен true, то результат будет false, и наоборот. В C++ оператор для логического НЕ – это восклицательный знак (!). Например:

```
    bool a = true;
    bool result = !a; // result будет равен false, так как операнд a
равен true
```

В алгебре логики (алгебра логики – раздел математической логики, в котором изучаются логические операции над высказываниями) используются *таблицы истинности*, которые показывают все возможные значения булевых переменных и результаты логических операций для этих значений. Таблица истинности позволяет анализировать и понимать логические выражения и функции. Она состоит из заголовка и тела таблицы. Заголовок содержит названия всех булевых переменных логического выражения. Тело таблицы содержит все возможные комбинации значений булевых переменных и результаты вычисления логического выражения для каждой комбинации. Количество строк в теле таблицы равно 2^n, где n – количество булевых переменных [10].

Таблица 6.1. – Таблица истинности логической операции И (AND) для двух переменных x_1 и x_2, выходное значение – y.

x_1	x_2	y
1(true)	1(true)	1(true)
0(false)	1(true)	0(false)
1(true)	0(false)	0(false)
0(false)	0(false)	0(false)

Таблица 6.2. – Таблица истинности логической операции НЕ (NOT) для двух переменных x_1 и x_2, выходное значение – y.

x_1	x_2	y
1(true)	1(true)	1(true)
0(false)	1(true)	1(true)
1(true)	0(false)	1(true)
0(false)	0(false)	0(false)

Таблица 6.3. – Таблица истинности логической операции ИЛИ (OR) для переменной x, выходное значение – y.

x	y
1(true)	0(false)
0(false)	1(true)

Логическая функция – это функция, у которой значения переменных и значение функции выражают логическую истинность. Она принимает одно или несколько

булевых значений в качестве аргументов и возвращает булев результат. Логические функции могут быть представлены с помощью логических выражений или таблиц истинности. Логические функции одной переменной представлены в таблице 6.4. Анализ таблицы свидетельствует о правилах: y_0 – константа равна 0; y_1 – соответствует значению входной переменной; y_2 – инверсное значение входной переменной (операция НЕ); y_3 – константа равна 1.

Таблица 6.4. – Табличное задание логических функций одной переменной

x	y_0	y_1	y_2	y_3
0	0	0	1	1
1	0	1	0	1

В таблице 6.5 представлен пример задания функций для двух переменных.

Таблица 6.5. – Табличное задание логических функций для двух переменных

№	x_1	x_2	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
2	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
3	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Наиболее распространенной в алгебре логики является *функционально полная система логических функций*, которая в качестве базовых логических функций использует функцию одной переменной НЕ и две функции двух переменных – И и ИЛИ.

Логическое выражение содержит одну или несколько логических операций и возвращает булево значение (`true` или `false`).

В языке программирования С++ логические функции могут быть реализованы с использованием операторов и выражений.

Пример реализации логической функции И (AND) *в С++:*

```
bool logicalAnd(bool a, bool b)
{
 return a && b;
}
```

Пример реализации логической функции ИЛИ (OR) *в С++:*

```
bool logicalOr(bool a, bool b)
{
 return a || b;
}
```

Пример реализации логической функции НЕ (NOT) *в C++:*

```
bool logicalNot(bool a)
{
 return !a;
}
```

Пример реализации логической функции Исключающее ИЛИ (XOR) *в C++:*

```
bool logicalXor(bool a, bool b)
{
 return (a && !b) || (!a && b);
}
```

В отличие от приведенных примеров в программных проектах логические функции могут быть более сложными и содержать большее количество аргументов.

Логические функции позволяют создавать условия и принимать решения на основе логических значений. Они являются важным инструментом в программировании и позволяют создавать более гибкие и управляемые программы и часто используются в условных операторах (`if-else`), циклах (`for`, `while`) и других конструкциях программы [11].

Условный оператор `if` позволяет выполнить определенный блок кода, если заданное логическое условие истинно:

```
if (условие)
{
 // выполняемый код, если условие истинно
}
```

Пример:

```
int x = 7;
if (x > 2)
{
 // выполняемый код, если x больше 2
}
```

Также с помощью алгебры логики можно комбинировать несколько условий с помощью логических операторов, таких как И (AND), ИЛИ (OR) и НЕ (NOT), для создания более сложных условий:

```
if (условие1 && условие2)
{
 // выполняемый код, если оба условия истинны
}
if (условие1 || условие2)
```

```
{
 // выполняемый код, если хотя бы одно из условий истинно
}
if (!условие)
{
 // выполняемый код, если условие ложно
}
```

Пример:

```
int x = 7;
int y = 14;
if (x > 0 && y > 0)
{
 // выполняемый код, если и x, и y больше нуля
}
if (x > 0 || y > 0)
{
 // выполняемый код, если хотя бы одно из условий истинно
}
if (!(x > 0))
{
 // выполняемый код, если условие ложно
}
```

Пример использования булевых переменных в конструкция `if` *и* `while`*:*

```
bool isTrue = true;
bool isFalse = false;

if (isTrue && !isFalse)
{
  // выполнить код, если isTrue равно true и isFalse равно false
}

while (isTrue || isFalse)
{
  // выполнить код, пока isTrue равно true или isFalse равно false
}
```

В приведенном примере условный оператор `if` проверяет, что переменная `isTrue` равна `true` и переменная `isFalse` равна `false`. Если это условие выполняется, то выполняется код внутри блока `if`. Цикл `while` выполняется, пока переменная `isTrue` равна `true` или переменная `isFalse` равна `false`.

II. Булевы переменные и выражения

Являются важным инструментом в программировании, так как позволяют программе принимать решения на основе логических условий. В C++ порядок условий играет очень большую роль. Логические выражения и переменные проверяются *слева направо*, и если результат всего выражения в скобках будет однозначно

определён после проверки первого выражения – остальные выражения проверяться не будут. Например, если в выражении `if (a && b && c)` хотя бы `a` имеет значение `false`, проверка остальных выражений (`b` и `c`) уже не выполняется, потому что всё выражение заведомо будет `false`.

Для другого случая, если в выражении `if (a || b || c)` хотя бы `a` будет `true` – всё выражение также будет `true`, причем `b` и `c` не будут проверяться.

Знание таких особенностей может помочь в оптимизации кода. Например, есть какой-то флаг и выражение, которое вычисляется прямо в условии и сразу проверяется. Если флаг опущен, программа не будет тратить время на лишние вычисления и сразу покинет условие. Например:

```
if (flag && analogRead(0) > 500)
{
 // делать что-то
}
```

Алгебра логики также используется для создания логических выражений, которые позволяют вычислять логические значения на основе заданных условий.

Например, в языке программирования C++ можно использовать логические выражения для проверки равенства или неравенства двух значений:

```
int number1 = 10;
int number2 = 20;
bool equal = (number1 == number2); // false
bool notEqual = (number1 != number2); // true
```

Также с помощью алгебры логики можно создавать более сложные логические выражения, комбинируя их с помощью логических операторов:

```
int number1 = 10;
int number2 = 20;
int number3 = 25;
// true, если number1 < number2 < number3
bool inRange = (number1 < number2 && number2 < number3);
// true, если number1 < number2 или number2 > number3
bool notInRange = (number1 < number2 || number2 > number3);
```

В C++ используется тернарный оператор (знак вопроса `?`), который является более коротким аналогом для записи конструкции `if-else`. Действие с оператором `?` имеет следующий вид:

```
условие ? выражение 1: выражение 2
```

При этом вычисляется условие, если оно истинно, то всё действие возвращает значение выражения 1, а если оно ложно, то всё действие возвращает значение выражения 2.

Пример:

```
byte number1, number2;
number1 = 15;
// если number1 > 20, number2 получает значение 30
// иначе number2 получает значение 40
number2 = (number1 > 20) ? 30 : 40;
```

Аналогичная конструкция с применением `if-else`

```
number1 = 15;
if (number1 > 20) number2 = 30;
else number2 = 40;
```

Другой вариант с вычислением:

```
byte number1 = 15, number2 = 5;
//прибавим к number2 результат выражения number1*10 если number1 > 10
//иначе прибавим number1+10
number2 += (number1 > 9) ? (number1 * 10) : (number1 + 10);
```

Алгебра логики также позволяет создавать и использовать *логические функции*, которые принимают одно или несколько логических значений в качестве аргументов и возвращают логическое значение в результате своей работы.

Логические функции позволяют абстрагироваться от конкретных условий и создавать более общие и переиспользуемые блоки кода. Они могут быть использованы для проверки определенных условий, выполнения сложных логических операций и принятия решений в программе [10].

Примеры логических функций:

```
bool isEven(int number)
{
 return (number % 2 == 0);
}
bool isPositive(int number)
{
 return (number > 0);
}
```

Приведенные функции могут быть использованы в программе для проверки четности или определения положительного числа:

```
int x = 5;
if (isEven(x))
{
 // выполняемый код, если число четное
}
```

```
if (isPositive(x))
{
 // выполняемый код, если число положительное
}
```

Таким образом, алгебра логики играет важную роль в программировании на языке C++, позволяя работать с логическими значениями, создавать условия и принимать решения на основе логических выражений. Она помогает создавать более гибкие и управляемые программы, а также повышает эффективность и надежность программного кода.

Ход работы

1. Ознакомиться с теоретическим материалом и закрепить знания подготовкой ответов на контрольные вопросы.
2. Получить у преподавателя вариант индивидуального задания (таблица 3.1).
3. Разработать и записать алгоритм решения задачи.
4. Составить текст программы.
5. Значение переменной вводить с консоли, при этом на экран консоли вывести наименование вводимой переменной. Результаты вычислений выводить на консоль, сопровождая их наименованиями выводимых значений.
6. Проанализировать результаты выполнения программы. Убедиться в правильности решения задачи, обосновать свой вывод о правильности работы программы, результат привести в отчете.
7. Оформить отчет.
8. Загрузить отчет на образовательную платформу.
9. Защитить работу у преподавателя. Для этого требуется знать ответы на теоретические вопросы, уметь написать код для аналогичной задачи без использования сторонних источников.

Задание 1

Написать код на C++:

1) для логической функции `AND`, которая принимает три аргумента и возвращает их логическое `И`.

2) для логической функции `OR`, которая, принимает три аргумента и, не применяя `if`, возвращает их логическое `ИЛИ`.

3) для логической функции `NOT`, которая принимает один аргумент и возвращает его логическое отрицание.

Задание 2

Написать программу с использованием логических функций (все значения вводятся с клавиатуры).

Таблица 6.6. – Варианты заданий

Вариант	Задание
1	Определить, являются ли числа `num1` и `num2` четными
2	Определить, являются ли числа `num1` и `num2` положительным
3	Определить, являются ли числа `num1` и `num2` отрицательным
4	Определить, являются ли числа `num1` и `num2` нечетным
5	Определить, является ли число `num1` квадратом числа `num2`
6	Определить, является ли число `num1` корнем числа `num2`
7	Определить, является ли результат выражения `num1−num2` положительным
8	Определить, является ли результат выражения `num1−num2` отрицательным
9	Определить, является ли результат выражения `num1−num2` четным
10	Определить, является ли результат выражения `num1−num2` нечетным

Контрольные вопросы

1. Что изучает алгебра логики? Сформулируйте используемые ею основные понятия.

2. Назовите основные логические операции и запишите для них таблицы истинности.

3. Приведите правила записи логических операций на C++.

4. Сформулируйте определение для логической функции. Приведите пример описания логическим выражением или таблицей истинности для функции одной переменной.

5. Приведите примеры кода с реализацией логических функций.

6. Приведите пример кода с комбинированием логических операций, отличный от представленного в теоретических сведениях.

7. Какие особенности логических выражений в C++ могут позволить оптимизировать код. Поясните и приведите пример.

8. Сформулируйте назначение тернарного оператора в C++. Поясните принцип его работы и приведите пример реализации.

9. Приведите пример на C++ для проверки равенства (неравенства) двух значений с использованием логических выражений.

10. Запишите код программы на C++, которая принимает два аргумента и возвращает их логическое `И`.

ЛАБОРАТОРНАЯ РАБОТА № 7
Реализация циклических алгоритмов

Цель: получение практических навыков при программировании циклических алгоритмов с использованием операторов циклов с предусловием, постусловием и со счетчиком.

Теоретический материал

Многие программы требуют циклического (итерационного) повторения выполнения группы операторов. Для их реализации в составе современных языков программирования высокого уровня предусмотрены специальные языковые конструкции. Граф-схема цикла с предусловием приведена на рисунок 7.1.

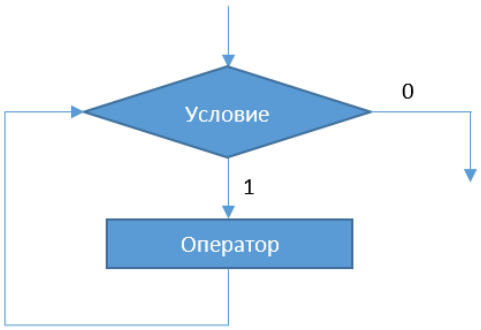


Рисунок 7.1. – Граф-схема цикла с предусловием

На языке программирования C++ такой цикл записывается следующим образом:

```
while (Условие)
Оператор;
```

В качестве условия выступает выражение булевского типа. Тело цикла выполняется до тех пор, пока значение условия истинно. Как только условие становится ложным, управление передается следующему за циклом оператору. Значение условия вычисляется на каждой итерации цикла. Тело цикла может не выполниться ни разу, если при первом проходе условие ложно. При необходимости записи нескольких операторов в теле цикла необходимо использовать операторные скобки.

В качестве примера рассмотрим нахождение суммы квадратов целых чисел от `1` до `N`:

$$S = \sum_{i=1}^{N} i2.$$

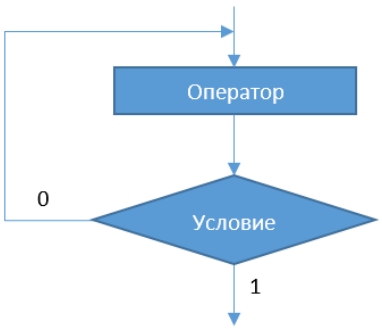
Для использования данного цикла в задаче нахождения суммы необходимо определиться с инициализирующими действиями, телом цикла и условием

завершения. При вычислении необходимо использовать вспомогательную переменную `i`, в которой хранится текущее число. Изначально оно равно `1`, на каждом шаге производится переход к следующему слагаемому путем увеличения его значения на единицу. Значение суммы изначально равно нулю, на каждом шаге оно увеличивается на выражение `i*i`. В теле цикла два оператора, поэтому необходимо использовать операторные скобки. Цикл продолжается до тех пор, пока значение переменной `i` не больше `N`. Соответствующая программа с использованием цикла `while` представлена ниже:

```
void main()
{
 int N = 15;
 int S = 0;
 int i = 6;

 while (i <= N)
  {
   S += i * i;
   i++;
  }

 cout << S << endl;

 getchar();
}
```

Еще одним видом цикла является цикл с постусловием. В нем условие располагается после тела цикла.

Рисунок 7.2. – Граф-схема цикла с постусловием



В языке программирования C++ такой цикл записывается следующим образом:

```
do
{
 Оператор1;
 Оператор2;
 ...
```

```
 ОператорN;
} while (Условие);
```

В качестве условия также выступает выражение булевского типа, ложное значение которого производит переход на новую итерацию цикла, а истинное является завершением цикла (наоборот по отношению к циклу `while`). Фактически цикл выполняется до тех пор, пока не станет истинным условие. В языке C++ цикл с постусловием имеет противоположное условие завершения.

Значение условия вычисляется на каждой итерации цикла. Тело цикла всегда выполняется хотя бы один раз, т. к. проверка условия завершения производится после тела цикла.

Для нахождения рассмотренной выше суммы квадратов чисел используются те же самые действия инициализации и операторы тела цикла. Программа с использованием цикла с постусловием выглядит следующим образом:

```
void main()
{
 int N = 15;
 int S = 0;
 int i = 4;

 do
 {
  S += i * i;
  i++;
 }

 while (i <= N);
 cout << S << endl;
 getchar();
}
```

Еще одним типом цикла является цикл со счетчиком. Данная форма цикла является достаточно удобной при записи цикла в случае, когда заранее известны пределы изменения переменной-счетчика (фактически количество итераций цикла) и шаг изменения счетчика равен, например, +1 или –1.

В языке C++ цикл со счетчиком записывается в следующем виде:

```
for (Инициализация; Условие_завершения;
Действие_между_итерациями) Оператор;
```

Действие «Инициализация» выполняется однократно перед выполнением тела цикла. Цикл выполняется до тех пор, пока выражение «`Условие_завершения`» истинно. При переходе от итерации к итерации выполняются операторы, записанные в секции «`Действие_между_итерациями`».

Для нахождения рассмотренной выше суммы квадратов с использованием цикла со счетчиком можно использовать следующую последовательность действий:

```
void main()
{
 int N = 15;
 int S = 0;
 int i = 4;

 for (int i = 1; i <= N; i++)
 S += i * i;

 cout << S << endl;
 getchar();
}
```

Следует отметить, что в цикле `for` его разделы можно пропускать.

Пример:

```
for (x=0; x!=29;)
scanf("%d", &x);
```

В данном примере отсутствует приращение счетчика и при такой записи кода на каждой итерации значение переменной `x` только сравнивается с числом. Программа будет выполняться, пока не будет введено значение `x = 29`.

Иногда в процессе выполнения цикла возникает необходимость в его прерывании либо досрочном переходе к следующей итерации. Для прерывания цикла используется процедура `break`, для досрочного перехода к следующей итерации – `continue`. Поддержка данных процедур интегрирована в компилятор, что делает их похожими на операторы. Процедурами `break` и `continue` можно пользоваться только в пределах тела цикла, в противном случае компилятор выдаст сообщение об ошибке. Если имеют место вложенные циклы, то процедуры `break` и `continue` оказывают влияние на изменение выполнения наиболее вложенного цикла, в котором они расположены.

Достаточно часто в программах используются циклы, условие выхода из которых никогда не выполняется (умышленно). Подобные циклы называются бесконечными и записываются следующим образом:

```
while (1)
 Оператор;
 do
{
 Оператор;
}
while (1);
```

Если программа попадает в такой цикл, то она перестает реагировать на действия пользователя – «повисает», что является крайне нежелательным. Поэтому обычно бесконечные циклы используются в совокупности с процедурой `break`, которая выполняется по какому-либо условию в теле цикла и тем самым прерывает его.

Ход работы

1. Ознакомиться с теоретическим материалом и закрепить знания подготовкой ответов на контрольные вопросы
2. Написать программу вычислений в соответствии с заданным вариантом (числовые параметры задаются самостоятельно вводом с клавиатуры).
3. Значение переменной вводить с консоли, при этом на экран консоли вывести наименование вводимой переменной. Результаты вычислений выводить на консоль, сопровождая их наименованиями выводимых значений.
4. Объяснить полученные результаты вычислений.
5. Оформить отчет.
6. Загрузить отчет на образовательную платформу.
7. Защитить работу у преподавателя. Для этого требуется знать ответы на теоретические вопросы, уметь написать код для аналогичной задачи без использования сторонних источников.

Таблица 7.1. – Варианты заданий

№ варианта	Задания
1	1. Напишите программу, которая выводит на экран числа от 1 до N. 2. Напишите программу, которая будет запрашивать у пользователя ввод чисел до тех пор, пока он не введет число 0. После этого программа должна вывести сумму всех введенных чисел
2	1. Вывести на экран все простые числа в диапазоне от 1 до N. Число называется простым, если оно делится без остатка только само на себя и на 1. 2. Напишите программу, которая запрашивает у пользователя целые числа до тех пор, пока он не введет число, кратное 7. После этого программа должна вывести на экран сообщение «Спасибо!»
3	1. Создайте программу, которая выводит на экран таблицу умножения для заданного числа N. 2. Написать программу, которая будет запрашивать у пользователя ввод чисел и выводить их квадраты, пока пользователь не введет отрицательное число. После этого программа должна вывести сумму всех введенных чисел и их квадратов

№ варианта	Задания
4	1. Напишите программу, которая будет запрашивать у пользователя ввод чисел и выводить их среднее арифметическое, пока пользователь не введет 0. После этого программа должна вывести сумму всех введенных чисел и их среднее арифметическое. 2. Создайте программу, которая находит сумму цифр введенного числа
5	1. Напишите программу, которая будет запрашивать у пользователя ввод чисел и выводить их сумму, пока пользователь не введет отрицательное число. После этого программа должна вывести сумму всех введенных положительных чисел и их количество. 2. Напишите программу, которая выводит на экран числа от 1 до 100, пропуская каждое второе число
6	1. Напишите программу, которая находит все делители введенного числа N. 2. Реализуйте программу, которая находит сумму цифр числа до тех пор, пока не останется одна цифра
7	1. Напишите программу, которая находит количество чисел, оканчивающихся на цифру 5, в диапазоне от 1 до 1000. 2. Напишите программу, которая постоянно печатает степени целого числа 2, соответственно 2, 4, 8, 16, 32, 64 и т. д., пока число не превысит N

Контрольные вопросы

1. Для каких типов задач требуется применение циклического повторения одних и те же шагов? Приведите примеры.
2. Какие типы циклов существуют? Как они записываются в конструкциях языков высокого уровня?
3. Представьте графическую схему цикла с предусловием и поясните ее.
4. Представьте графическую схему цикла с постусловием и поясните ее.
5. Что является условием завершения цикла каждого конкретного типа?
6. Какое минимальное и максимальное количество раз может быть выполнен цикл каждого из типов?
7. Приведите два примера цикла `for`, в которых пропущены разные разделы.
8. Что такое бесконечные циклы?
9. Какие средства языка применяются для досрочного прерывания тела цикла? Приведите пример задачи и кода.
10. Какие средства языка применяются для досрочного перехода к следующей итерации цикла? Приведите пример задачи и кода.

ЛАБОРАТОРНАЯ РАБОТА № 8
Реализация смешанных алгоритмов

Цель: получение практических навыков при программировании смешанных алгоритмов.

Теоретический материал

I. Смешанные алгоритмы

Это методы обработки данных или решения задач, которые комбинируют в себе несколько различных подходов или методов,что позволяет использовать преимущества каждого из методов и минимизировать их недостатки. Смешанные алгоритмы широко применяются в различных областях, включая машинное обучение, криптографию, оптимизацию, компьютерное зрение и др.

В криптографии смешанные алгоритмы могут использоваться для обеспечения безопасности передачи данных путем комбинирования различных методов шифрования или аутентификации [12].

Таким образом, смешанные алгоритмы позволяют создавать более эффективные и универсальные решения для сложных задач, иногда превосходящие возможности отдельных методов.

Для работы с смешанными алгоритмами в C++ нужно иметь хорошее понимание стандартной библиотеки и умение комбинировать различные алгоритмы правильным образом.

Пример смешанного алгоритма в C++:

```
#include <iostream>

int mixed(int a, int b)
{
  if (a % b == 0)
  {
    return 1; // второе число кратно первому
  }
  else
  {
    return 0; // второе число не кратно первому
  }
}

int main()
{
  int n;
  std::cout << "Введите количество пар целых чисел: ";
```

```
    std::cin >> n;

    for (int i = 0; i < n; ++i)
    {
      int num1, num2;
      std::cout << "Введите два целых числа через пробел: ";
      std::cin >> num1 >> num2;

      if (multiple(num2, num1))
      {
        std::cout << "Второе число кратно первому." << std::endl;
       }
       else
       {
        std::cout << "Второе число не кратно первому." << std::endl;
       }
    }

    return 0;
  }
```

Этот код включает функцию `mixed`, которая определяет, кратно ли второе число первому. Если второе число кратно первому, функция возвращает `1`, иначе `0`. В функции `main` программа запрашивает количество пар целых чисел, затем последовательно вводит пары чисел и выводит результат работы функции `mixed` для каждой пары.

Еще одним вариантом смешанного алгоритма может быть функция рандома.

Пример. Вычислим интервал времени в секундах между двумя моментами, находящимися внутри двенадцатичасового цикла, при помощи функции рандома:

```
  #include <iostream> // Подключение библиотеки для ввода/вывода
  #include <cstdlib> // Подключение библиотеки для работы
с функцией rand
  #include <ctime> // Подключение библиотеки для работы
с функцией time

  int main()
  {
    // Используем текущее время как начальное значение для генератора
    srand(static_cast<unsigned int>(time(0)));

    // Генерация случайного числа от 1 до 10
    int randomNumber = rand() % 10 + 1;

    // Вывод сгенерированного случайного числа
    std::cout << "Случайное число: " << randomNumber << std::endl;

    return 0;
  }
```

II. Итеративный подход смешанных алгоритмов

Это метод решения задачи, основанный на использовании циклов. В этом подходе используются циклы (например, `for` или `while`) для последовательного выполнения действий до достижения желаемого результата.

Преимущества итеративного подхода:

– *простота:* итеративный подход обычно более прямолинеен и интуитивно понятен. Он состоит из последовательного выполнения действий с использованием циклов, что делает его легче понять и реализовать;

– *эффективность:* итеративные алгоритмы обычно имеют небольшие накладные расходы, связанные с вызовом функций, и могут быть оптимизированы для более быстрой работы;

– *управление памятью:* в отличие от рекурсии, итеративные алгоритмы не требуют большого объема памяти для хранения промежуточных результатов.

Недостатки итеративного подхода:

– *читабельность кода:* в итеративном подходе может быть необходимо использовать большое количество структур управления потоком, таких как циклы и условные инструкции, что может сделать код менее читабельным, особенно для более сложных алгоритмов;

– *ограниченность:* некоторые алгоритмы могут быть сложными для реализации с использованием итераций. Например, алгоритмы, связанные с деревьями или графами, могут быть более естественно выражены с помощью рекурсии.

Пример итеративного подхода для вычисления n-го числа Фибоначчи:

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

int fibonacci_iterative(int n)
{
 if (n <= 1)
 {
   return n;
 }

 int a = 0, b = 1, c;
 for (int i = 2; i <= n; ++i)
 {
  c = a + b;
  a = b;
  b = c;
 }
```

```
    return b;
}

int main()
{
    int n = 10;
    cout << "Итеративный подход: Fibonacci(" << n << ") = " << fibo-
nacci_iterative(n) << endl;
    return 0;
}
```

В данном примере описана функция `fibonacci_iterative`, которая принимает на вход число `n`, для которого нужно вычислить `n`-е число Фибоначчи. В самом начале проверяется базовый случай, когда `n` меньше или равно `1`, в таком случае просто возвращается само число `n`, в остальных случаях используется цикл `for`, в котором вычисляются следующие числа Фибоначчи по формуле `c = a + b`, где `a` и `b` – два предыдущих числа. После прохождения цикла возвращается последнее найденное число `b`.

Ход работы

1. Ознакомиться с теоретическим материалом и закрепить знания подготовкой ответов на контрольные вопросы.
2. Написать программу вычислений в соответствии с заданным вариантом (числовые параметры задаются самостоятельно вводом с клавиатуры).
3. Значение переменной вводить с консоли, при этом на экран консоли вывести наименование вводимой переменной. Результаты вычислений выводить на консоль, сопровождая их наименованиями выводимых значений.
4. Объяснить полученные результаты вычислений.
5. Оформить отчет.
6. Загрузить отчет на образовательную платформу.
7. Защитить работу у преподавателя. Для этого требуется знать ответы на теоретические вопросы, уметь написать код для аналогичной задачи без использования сторонних источников.

Таблица 8.1. – Варианты заданий

№ варианта	Задания
1	Разработайте следующие целые функции: а) функцию `celsius`, которая возвращает температуру по Цельсию, эквивалентную температуре по Фаренгейту; б) используйте эту функцию для написания программы, которая печатает таблицу, показывающую эквивалент по Фаренгейту всех температур по Цельсию от 0 до 100 градусов

№ варианта	Задания
2	Разработайте следующие целые функции: а) функцию `fahrenheit`, которая возвращает температуру по Фаренгейту, эквивалентную температуре по Цельсию; б) используйте эту функцию для написания программы, которая печатает таблицу, показывающую эквивалент по Цельсию всех температур по Фаренгейту от 32 до 212 градусов
3	Напишите программу, которая поможет учащимся начальной школы изучить умножение. Используйте `rand` для выработки двух положительных одноразрядных целых чисел. Программа должна печатать вопрос типа «Сколько будет `6 * 7`?». Затем учащийся печатает ответ. Ваша программа проверяет этот ответ. Если он правильный, напечатайте «Очень хорошо!» и затем задайте следующий вопрос на умножение. Если ответ неправильный, напечатайте «Нет. Повторите, пожалуйста, снова.» и затем задавайте тот же самый вопрос повторно до получения правильного ответа
4	Напишите программу, которая поможет учащимся начальной школы изучить деление. Используйте `rand` для выработки двух положительных одноразрядных целых чисел. Программа должна печатать вопрос типа «Сколько будет `16 / 7`?». Затем учащийся печатает ответ. Ваша программа проверяет этот ответ. Если он правильный, напечатайте «Очень хорошо!» и затем задайте следующий вопрос на деление. Если ответ неправильный, напечатайте «Нет. Повторите, пожалуйста, снова.» и затем задавайте тот же самый вопрос повторно до получения правильного ответа
5	Напишите программу, которая поможет учащимся начальной школы изучить сложение. Используйте `rand` для выработки двух положительных одноразрядных целых чисел. Программа должна печатать вопрос типа «Сколько будет `16 + 7`?». Затем учащийся печатает ответ. Ваша программа проверяет этот ответ. Если он правильный, напечатайте «Очень хорошо!» и затем задайте следующий вопрос на деление. Если ответ неправильный, напечатайте «Нет. Повторите, пожалуйста, снова.» и затем задавайте тот же самый вопрос повторно до получения правильного ответа
6	Напишите программу, моделирующую бросание монеты. Для каждого броска монеты программа должна печатать `Орел` или `Решка`. Промоделируйте с помощью этой программы бросание 100 раз и подсчитайте, сколько раз появилась каждая сторона монеты. Напечатайте результаты. Программа должна вызывать отдельную функцию `flip`, которая не принимает никаких аргументов и возвращает `0` для `Орла` и `1` для `Решки`. Замечание: если программа действительно моделирует бросание монеты, каждая сторона монеты должна появляться примерно в половине случаев

<table>
<tr><th>№ варианта</th><th>Задания</th></tr>
<tr><td>7</td><td>Напишите программу, которая бы выполняла следующее:
а) вычисляет целую часть частного от деления целого числа <code>a</code> на целое число <code>b</code>;
б) вычисляет целый остаток от деления целого числа <code>a</code> на целое число <code>b</code>;
в) использовать фрагменты программ, созданные в пунктах а) и б), для написания функции, которая вводит целое число из диапазона от 1 до 1000 и печатает его как последовательность цифр, каждая из которых отделена от соседней двумя пробелами. Например, целое число <code>456</code> должно быть напечатано в виде
<code>4 5 6</code></td></tr>
<tr><td>8</td><td>За стоянку до трех часов парковочный гараж запрашивает плату минимум $1.50. В случае стоянки более трех часов гараж дополнительно запрашивает $0.50 за каждый полный или неполный час сверх трех часов.
Напишите программу расчета и печати оплаты за парковку для каждого из трех клиентов, которые парковали свои автомобили вчера в этом гараже. Вы должны вводить длительность парковки для каждого клиента вручную. Программа должна печатать результаты вчерашнего дохода в аккуратном табулированном формате. Используйте функцию <code>calculateCharges</code>, чтобы определять плату для каждого клиента.
Результаты работы должны представляться в следующем формате:
<table>
<tr><th>Автомобиль</th><th>Часы</th><th>Плата</th></tr>
<tr><td>1</td><td>1,5</td><td>2,00</td></tr>
<tr><td>2</td><td>4,0</td><td>2,50</td></tr>
<tr><td>3</td><td>24,0</td><td>10,00</td></tr>
<tr><td>Итого</td><td>29,5</td><td>14,50</td></tr>
</table></td></tr>
<tr><td>9</td><td>За стоянку до пяти часов парковочный гараж запрашивает плату минимум $5.00. В случае стоянки более пяти часов гараж дополнительно запрашивает $1.50 за каждый полный или неполный час сверх пяти часов.
Напишите программу расчета и печати оплаты за парковку для каждого из трех клиентов, которые парковали свои автомобили вчера в этом гараже. Вы должны вводить длительность парковки для каждого клиента вручную. Программа должна печатать результаты вчерашнего дохода в аккуратном табулированном формате. Используйте функцию <code>calculateCharges</code>, чтобы определять плату для каждого клиента.
Результаты работы должны представляться в следующем формате:
<table>
<tr><th>Автомобиль</th><th>Часы</th><th>Плата</th></tr>
<tr><td>1</td><td>1,5</td><td>2,00</td></tr>
<tr><td>2</td><td>4,0</td><td>2,50</td></tr>
<tr><td>3</td><td>24,0</td><td>10,00</td></tr>
<tr><td>Итого</td><td>29,5</td><td>14,50</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>№ варианта</th><th>Задания</th></tr>
<tr><td>10</td><td>За стоянку до двух часов парковочный гараж запрашивает плату минимум $2.00. В случае стоянки более двух часов гараж дополнительно запрашивает $0.75 за каждый полный или неполный час сверх двух часов.
Напишите программу расчета и печати оплаты за парковку для каждого из трех клиентов, которые парковали свои автомобили вчера в этом гараже. Вы должны вводить длительность парковки для каждого клиента вручную. Программа должна печатать результаты вчерашнего дохода в аккуратном табулированном формате. Используйте функцию <code>calculateCharges</code>, чтобы определять плату для каждого клиента.
Результаты работы должны представляться в следующем формате:
<table>
<tr><th>Автомобиль</th><th>Часы</th><th>Плата</th></tr>
<tr><td>1</td><td>1,5</td><td>2,00</td></tr>
<tr><td>2</td><td>4,0</td><td>3,50</td></tr>
<tr><td>3</td><td>24,0</td><td>18,50</td></tr>
<tr><td>Итого</td><td>29,5</td><td>22,25</td></tr>
</table>
</td></tr>
</table>

Контрольные вопросы

1. Что такое смешанные алгоритмы?
2. Для каких задач следует использовать смешанные алгоритмы и в чем их преимущество?
3. Приведите пример смешанного алгоритма и объясните его.
4. Охарактеризуйте итеративный подход смешанных алгоритмов.
5. Какие циклы используются в итеративном подходе?
6. Сформулируйте преимущества и недостатки итеративного подхода.

СПИСОК ЛИТЕРАТУРЫ

1. Дейтел Х. М., Дейтел П. Дж. Как программировать на C++: Пятое издание. Пер. с англ. – М.:ООО «Бином-Пресс», 2008. – 1456 с.

2. Состояние и тенденции развития ЭВМ[Электронный ресурс]. – URL: https://phys.bspu.by/static/lib/inf/posob/stu_m/glaves/glava5/gl_5_1.html (дата обращения: 09.05.2024).

3. Система типов C++ [Электронный ресурс]. – URL: https://learn.microsoft.com/ru–ru/cpp/cpp/cpp–type–system–modern–cpp?view=msvc–170 (дата обращения: 09.05.2024).

4. Основы C++ [Электронный ресурс]. – URL: https://education.yandex.ru/handbook/cpp/article/data–types (дата обращения: 09.05.2024).

5. Microsoft.com // Создание проекта в Visual Studio [Электронный ресурс]. – URL: https://learn.microsoft.com/ru–ru/visualstudio/ide/create–new–project?view=vs–2022 (дата обращения: 09.05.2024).

6. Microsoft.com [Электронный ресурс] // Создание проекта консольного приложения C++. – URL: https://learn.microsoft.com/ru–ru/cpp/build/vscpp–step–1–create?view=msvc–170 (дата обращения: 09.05.2024).

7. Бекман Игорь Н. Компьютерные науки. Курс лекций. Лекция 7. Алгоритмы [Электронный ресурс]. – URL: https://profbeckman.narod.ru/Komp.files/Lec7.pdf (дата обращения: 09.05.2024).

8. Ермилов С. В., Жилевич М. И., Филипова Л. Г. Информатика [Электронный ресурс]. – Минск: БНТУ, 2024. – URL: https://rep.bntu.by/bitstream/handle/data/141802/Informatika.pdf?sequence=1&isAllowed=y (дата обращения: 09.05.2024).

9. Алгебра логики. Основные логические операции и их таблицы истинности. Основные законы алгебры логики. // iMath Wiki. – URL: https://wiki.livid.pp.ru/students/cs/lectures/6.html (дата обращения: 09.05.2024).

10. Логические функции и таблицы их истинности [Электронный ресурс]. – Пенза: ПГУ, 2019 – URL: https://dep_ivs.pnzgu.ru/files/dep_ivs.pnzgu.ru/kurs_lekciy_discipliny_aks.doc (дата обращения: 09.05.2024).

11. Шаньгин В. Ф. Защита компьютерной информации. Эффективные методы и средства. – М: ДМК Пресс, 2010. – 546 с.