1- Практическое задание- Введение в C++

Table of Contents

Установка поддержки C и C++ в Visual Studio

В этой версии (Visual Studio 2022) можно легко выбрать и установить только необходимые компоненты. Поскольку она занимает меньше памяти, она быстро устанавливается и при установке меньше влияет на систему.

Примечание

Этот раздел относится к установке Visual Studio в Windows. Но, **Visual Studio Code** — это упрощенная среда кроссплатформенной разработки, работающая в системах Windows, Mac и Linux. Расширение **Microsoft C/ C++ для Visual Studio Code** поддерживает технологию IntelliSense, отладку, форматирование кода, автоматическое завершение. Visual Studio для Mac не поддерживает Microsoft C++, но поддерживает языки .NET и кроссплатформенную разработку. Инструкции по установке см. **Установка Visual Studio для Mac**.

Шаг 1. Подготовка компьютера к установке Visual Studio

Перед началом установки Visual Studio:

1. Проверьте требования к системе. Так вы узнаете, может ли ваш компьютер поддерживать Visual Studio 2022.
2. Примените актуальные обновления Windows. Эти обновления гарантируют, что на компьютере установлены последние обновления для системы безопасности и необходимые системные компоненты для Visual Studio.
3. Перезапуск. Перезагрузка гарантирует, что ожидающие установки или обновления компоненты не будут препятствовать установке Visual Studio.
4. Освободите место. Удалите ненужные файлы и приложения с системного диска. Например, запустите приложение очистки диска.

Сведения об использовании предыдущих версий Visual Studio параллельно с Visual Studio 2022 см. https://learn.microsoft.com/ru-ru/visualstudio/releases/2022/compatibility .

Шаг 2. Скачивание Visual Studio

Теперь скачайте файл начального загрузчика Visual Studio. Для этого нажмите приведенную ниже кнопку, чтобы перейти на страницу скачивания Visual Studio. Выберите выпуск **Visual Studio Community:**

https://visualstudio.microsoft.com/vs/community/

Шаг 3. Установка установщика Visual Studio

Запустите скачанный файл начального загрузчика, чтобы установить Visual Studio Installer. Новый установщик имеет меньший размер и включает все необходимое для установки и настройки Visual Studio.

1. В папке **Загрузки** дважды щелкните файл начального загрузчика, имя которого совпадает с именем одного из следующих файлов или похоже на них:
    - **vs_community.exe** для Visual Studio Community.

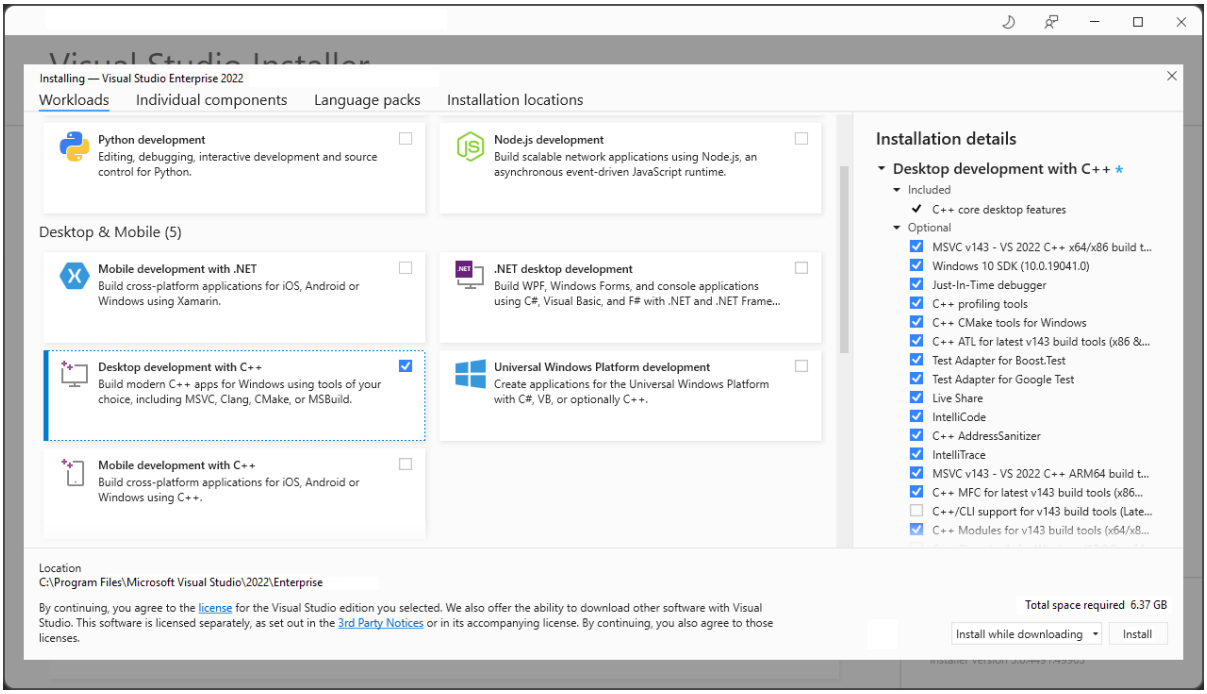
   Если появляется оповещение системы контроля учетных записей, нажмите кнопку **Да**, чтобы запустить начальный загрузчик.

2. Подтвердить условия лицензионного соглашения Майкрософт и заявление о конфиденциальности Майкрософт. Выберите **Продолжить**.

Шаг 4. Выбор рабочих нагрузок

Когда завершится установка программы установки, вы можете с ее помощью выбрать нужные *рабочие нагрузки* или наборы функций. Это делается следующим образом.

1. Найдите нужную рабочую нагрузку на экране **Установка Visual Studio**.



Для поддержки C и C++ выберите рабочую нагрузку "Разработка классических приложений на C++". В нее входит основной редактор кода по умолчанию, который предоставляет базовую поддержку редактирования кода для более чем

20 языков, возможность открывать и изменять код в любой папке без наличия проекта и интегрированное управление исходным кодом.

Другие рабочие нагрузки поддерживают дополнительные виды разработки. Например, выберите рабочую нагрузку "Разработка приложений для универсальной платформы Windows", чтобы создать приложения, использующие среду выполнения Windows для Microsoft Store. Выберите "Разработка игр на C++", чтобы создать игры, использующие DirectX, Unreal и Cocos2d. Выберите "Разработка приложений для Linux на C++" для целевых платформ Linux, включая разработку приложений Интернета-вещей.

В области **Сведения об установке** перечислены включенные и необязательные компоненты, устанавливаемые каждой рабочей нагрузкой. В этом списке можно выбрать или отменить выбор дополнительных компонентов. Например, чтобы обеспечить поддержку разработки с помощью наборов инструментов компилятора Visual Studio 2017 или 2015, выберите дополнительные компоненты MSVC v141 или MSVC v140. Вы можете добавить поддержку MFC, расширение языка экспериментальных модулей, IncrediBuild и многое другое.

2. Выбрав нужные рабочие нагрузки и дополнительные компоненты, нажмите **Установить**.

   Далее будут отображаться экраны состояния, на которых демонстрируется ход установки Visual Studio.

Совет

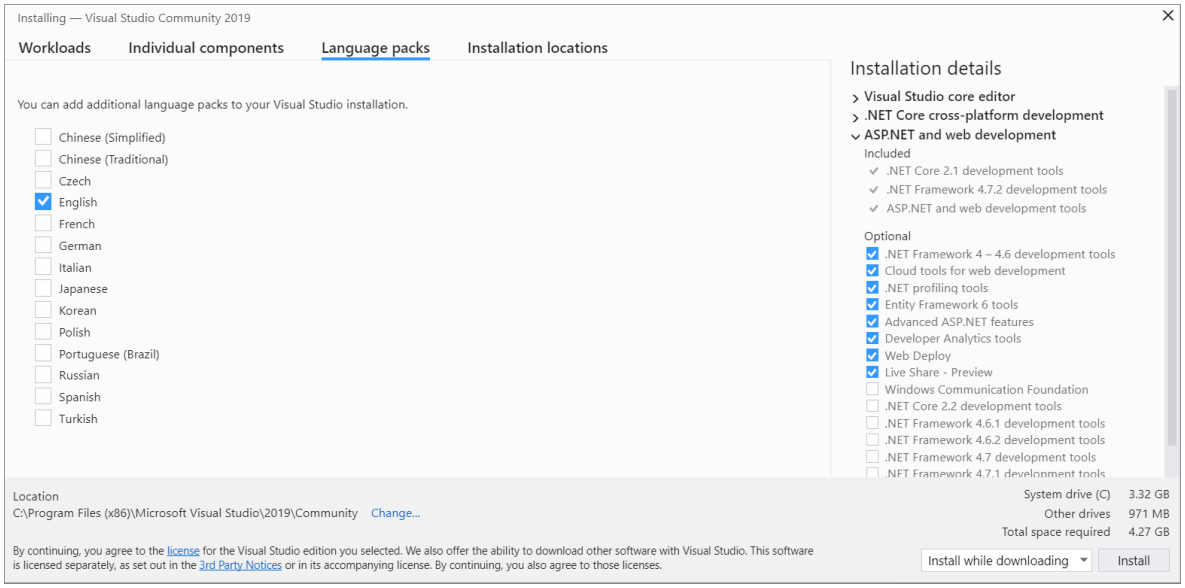
В любой момент после установки можно установить рабочие нагрузки или компоненты, которые не были установлены изначально. Если среда Visual Studio открыта, выберите пункт **Сервис**>**Получить средства и компоненты...**; откроется Visual Studio Installer. **Visual Studio Installer** можно также открыть из меню "Пуск". Здесь можно выбрать рабочие нагрузки или компоненты, которые нужно установить. Затем выберите **Изменить**.

Шаг 5. Выбор отдельных компонентов (необязательно)

Если вы не хотите использовать функцию "Рабочие нагрузки", чтобы настроить установку Visual Studio или добавить дополнительные компоненты, чем установка рабочей нагрузки, можно сделать это, установив или добавив отдельные компоненты на **вкладке "Отдельные компоненты**". Выберите нужные компоненты, а затем следуйте инструкциям.

Шаг 6. Установка языковых пакетов (необязательно)

По умолчанию при первом запуске установщик пытается использовать язык операционной системы. Чтобы установить Visual Studio на нужном языке, выберите в Visual Studio Installer вкладку **Языковые пакеты** и следуйте указаниям.



Изменение языка установщика из командной строки

Язык по умолчанию можно изменить еще одним способом — запустив установщик из командной строки. Например, можно принудительно запустить установщик на английском языке, выполнив команду `vs_installer.exe --locale en-US`. Программа установки запомнит этот параметр и использует его при следующем запуске. Установщик поддерживает следующие токены языков: zh-cn, zh-tw, cs-cz, en-us, es-es, fr-fr, de-de, it-it, ja-jp, ko-kr, pl-pl, pt-br, ru-ru и tr-tr.

Шаг 7. Изменение расположения установки (дополнительно)

Вы можете уменьшить место, занимаемое установкой Visual Studio на системном диске. Вы можете переместить кэш загрузки, общие компоненты, пакеты SDK и средства на другие диски и оставить Visual Studio на самом быстром диске.

Важно!

Вы можете выбрать другой диск только в том случае, если вы устанавливаете Visual Studio впервые. Если вы уже установили ее и хотите изменить диск, необходимо удалить Visual Studio, а затем переустановить ее.

Шаг 8. Начало разработки

1. Когда установка Visual Studio завершится, нажмите кнопку **Запустить**, чтобы приступить к разработке в Visual Studio.
2. На начальном экране выберите **Создать проект**.
3. В поле поиска введите тип приложения, которое вы хотите создать, чтобы просмотреть список доступных шаблонов. Список шаблонов зависит от рабочих нагрузок, выбранных во время установки. Чтобы просмотреть различные шаблоны, выберите разные рабочие нагрузки.

   Можно также фильтровать поиск по определенному языку программирования с помощью раскрывающегося списка **Язык**. Вы также можете выбирать фильтры из списка **Платформа** и **Тип проекта**.

4. Новый проект откроется в Visual Studio, и вы можете приступить к написанию кода!

https://learn.microsoft.com/ru-ru/cpp/build/vscpp-step-0-installation?view=msvc-170

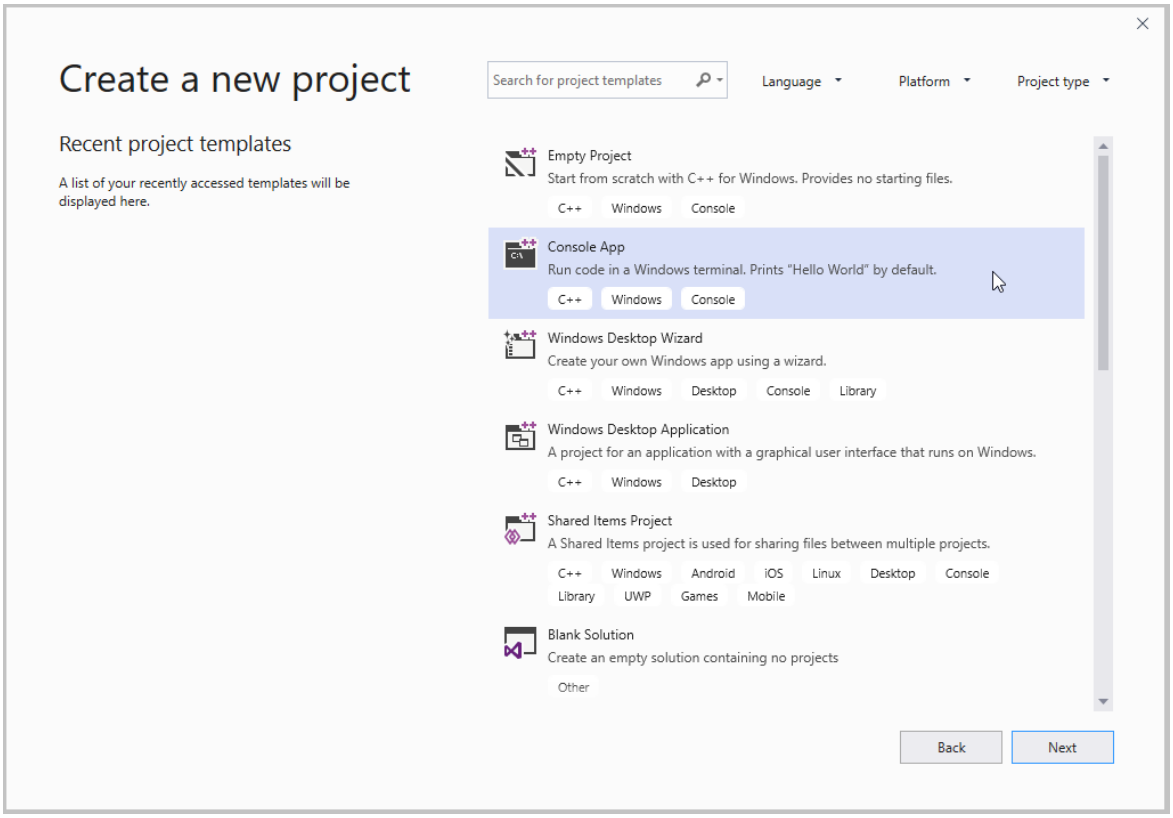
Создание и запуск проекта консольного приложения C++

Обычно программист C++ начинает с создания приложения "Hello, world!", которое запускается из командной строки. Это то, что вы создаете в Visual Studio на этом шаге.

Создание проекта приложения

Visual Studio использует *проекты*, чтобы упорядочить код для приложения, и *решения*, чтобы упорядочить проекты. Проект содержит все параметры, конфигурации и правила, используемые для сборки приложения. Он управляет связью между всеми файлами проекта и любыми внешними файлами. Чтобы создать приложение, сначала создайте проект и решение.

1. В Visual Studio откройте **меню "Файл"** и выберите **"Создать > проект** ", чтобы открыть **диалоговое окно "Создать проект** ". Выберите шаблон **Консольное приложение** с тегами **C++**, **Windows** и **Консоль**, а затем нажмите кнопку **Далее**.

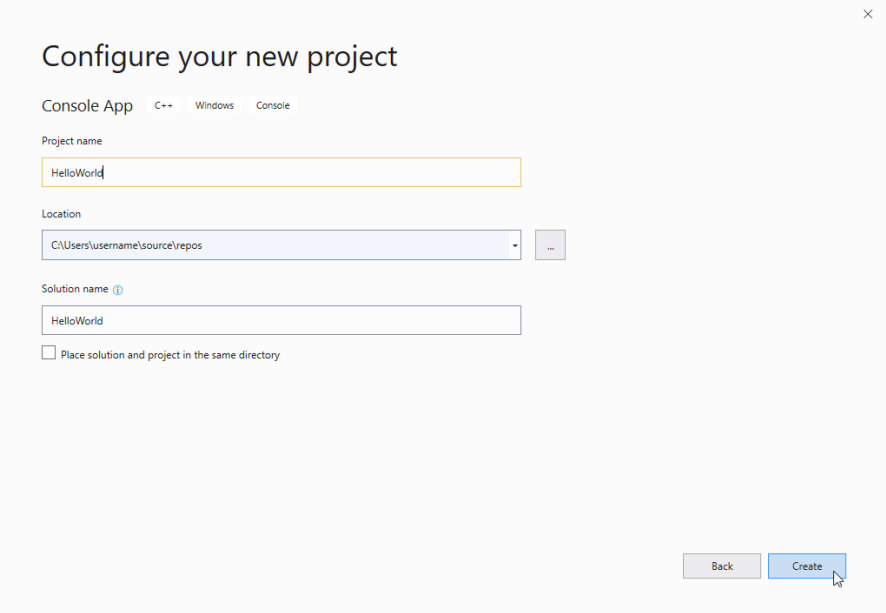


Диалоговое окно создания проекта с выбранным шаблоном консольного приложения. Этот шаблон говорит: запуск кода в терминале Windows. По умолчанию выводит hello world. Содержит теги c++, Windows и консоль.

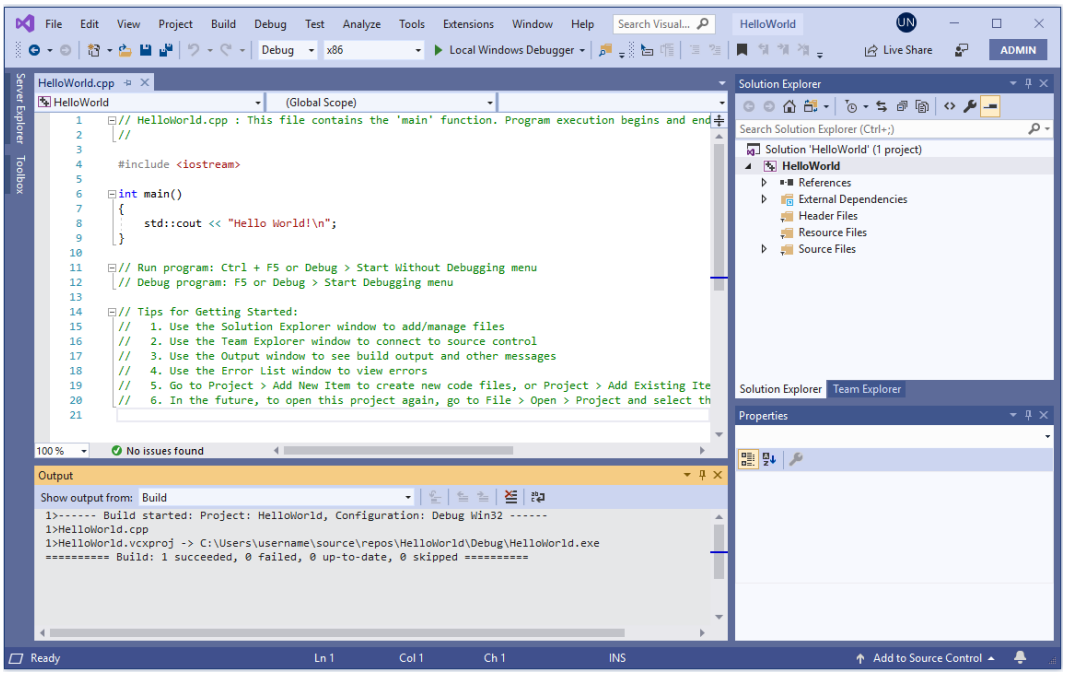
2. В диалоговом окне **Настроить новый проект** в поле **Имя проекта** введите *HelloWorld*. Выберите **Создать**, чтобы создать проект.

   Диалоговое окно "Настройка нового проекта" с помощью HelloWorld, введенного в поле "Имя проекта".

   Visual Studio создаст проект. Вы можете приступать к добавлению и изменению исходного кода. По умолчанию шаблон консольного приложения предоставляет исходный код для приложения Hello World, как показано ниже.



Отображает новый проект. Файл HelloWorld.cpp открыт, отображая код по умолчанию, включенный в этот шаблон. Этот код состоит из #include iostream и функции main(), содержащей строку: std::cout << кавычки hello world!\n quote;



Когда код в редакторе будет выглядеть таким образом, вы можете перейти к следующему шагу и начать разработку приложения.

Понимание структуры базовой программы на C++

Базовая программа на C++ делится на следующие три части:

- Раздел стандартных библиотек
- Основная функция
- Раздел тела

Рассмотрим на примере реализацию программы Hello World:

```
#include <iostream>

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

Или можно было бы записать так:

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

int main()
{
    cout << "Hello World!\n";
}
```

1. *Раздел Стандартных Библиотек*

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

Программа часто содержит множество программных конструкций, включая встроенные функции, классы, ключевые слова, константы, операторы и многое другое, предварительно определенное в стандартной библиотеке C++. Для использования таких предопределенных компонентов в приложении должен быть предоставлен подходящий заголовок. Кроме того, стандартные заголовки предоставляют такие сведения, как тип данных констант, прототип, определение и возвращаемый тип библиотечных функций, среди прочего. Специальная инструкция препроцессора, известная как **#include**, копирует и вставляет полный текст файла, заключенный в угловые скобки, в исходный код.

Потоки ввода-вывода обозначаются аббревиатурой «**iostream**» и представляют собой стандартный файл, который должен быть включен в

компилятор C++. В этой команде содержатся коды пользовательского ввода и отображения.

Комитет по стандартам C++ внес несколько улучшений в C++ с момента создания языка. Аналогичной новой функцией этого языка является **Namespace**. Он позволяет объединять несколько вещей под одним именем, включая классы, объекты, функции и другие токены C++. Отдельные пространства имен могут формироваться разными пользователями. Они могут использовать имена для сущностей, которые в результате похожи. Делая это, можно избежать ошибки времени компиляции, вызванной конфликтами идентичных имен. Объекты стандартной библиотеки были реорганизованы Комитетом по стандартам C++ под пространством имен std. Для всех имен в определенном наборе пространство имен является используемым префиксом. В этом приложении в файле **iostream** определены— **cout**. А в C++ символ **::** называется оператором разрешения области видимости. Он дает понять, к какому пространству имен (**Namespace**) или классу принадлежит символ

2. *Основная Функция*

```
int main()
```

Функция запуска, называемая **main()**, инициирует выполнение программы C++. Функция main служит основой любой программы на C++. Каждый оператор C++, который необходимо выполнить, записывается в функции **main ( )**. Все инструкции, заключенные в открывающие и закрывающие фигурные скобки, окружающие основную часть кода, выполняются компилятором **( )**. Программа завершается, и значение возвращается в операционную систему, как только все инструкции в main() были выполнены. В C++ **main()** обычно дает операционной системе значение int. Следовательно, оператор return 0 должен стоять в конце **main()**. Возвращаемые значения 0 и ненулевые указывают на успех и неудачу соответственно.

```
{       указывает на начало блока кода,
}       обозначает его конец.
```

Когда ваше программное обеспечение запускается компьютером, операционная система вызывает эту функцию.

В VS C++ 2022 необязательно вводить " return 0; " в конце функции main, и компилятор включает его автоматически

3. *Раздел Тела*

```
cout << "Hello World!\n";
```

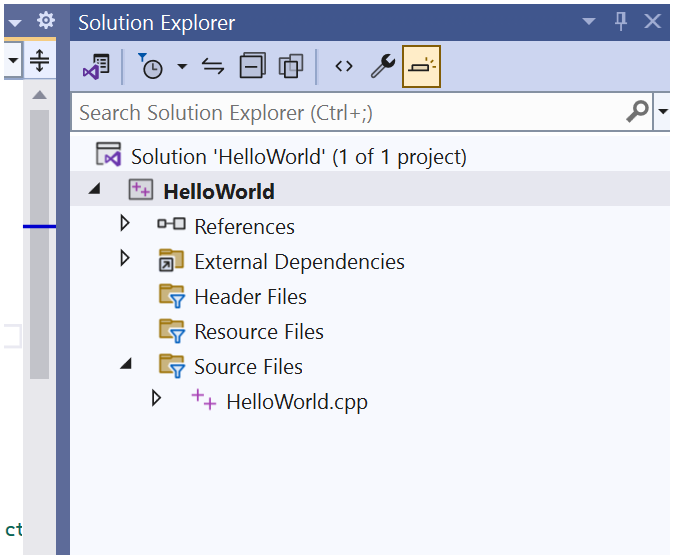
Вывод символов обозначается аббревиатурой **cout**, которая показывает все, что заключено в скобки **<<**.

Данная строка выводит на консоль строку `Hello World!` и переход к следующей строке с помощью `\n`, и данная строка является инструкцией и поэтому завершается точкой с запятой.

Но, следующие две строки кода выводит на консоль строку `Hello World!` и переход к следующей строке с помощью `\n`, и в новой строке выводит на консоль строку `Hello World!` и переход к следующей строке с помощью `\n` и они являются инструкциями и поэтому заканчиваются точкой с запятой:

```
cout << "Hello World!\n";
cout << "Bey World!\n";
```

Понимание структуры проекта



Скриншот, показывает обозреватель решений в Visual Studio 2022, особенно для проекта на C++. Вот краткое описание содержимого и структуры, которую вы видите:

1- **Solution 'HelloWorld' (1 из 1 проекта)**: это контейнер решения, который может содержать один или несколько проектов. В данном случае он содержит один проект под названием "HelloWorld".
2- **HelloWorld**: это проект внутри решения, где организован ваш код на C++ и связанные с ним файлы. Проект настроен с различными папками, которые помогают организовать различные типы файлов:
   - **References**: эта папка могла бы содержать сборки или объекты, на которые ссылается ваш проект. В контексте C++, это может включать ссылки на другие проекты или библиотеки, используемые в рамках этого проекта.
   - **External Dependencies**: это файлы, не входящие в проект, от которых зависит компиляция и выполнение вашего проекта. Это может включать библиотеки, фреймворки или другие инструменты. В C++ это часто включает заголовочные файлы и библиотеки, которые не являются частью проекта, но требуются для компиляции.
   - **Header Files**: Обычно содержат файлы .h или .hpp . Эти файлы объявляют интерфейсы ваших классов, функций и констант.
   - **Resource Files**: Эта папка может содержать файлы, не являющиеся кодом, но необходимые вашей программе, такие как изображения, значки или другие двоичные или текстовые ресурсы.
   - **Source Files**: Здесь находятся ваши файлы .cpp. Реализация классов и функций вашей программы находится в этих файлах. На скриншоте есть файл с именем HelloWorld.cpp, который, вероятно, содержит главную функцию и точку входа приложения.

Каждая из этих папок помогает организовать проект таким образом, чтобы вы могли легко управлять и перемещаться по различным частям вашей кодовой базы. Структура предназначена для обеспечения чёткого разделения между разными типами файлов, что особенно полезно в крупных проектах.

Создание проекта приложения: Проблемы

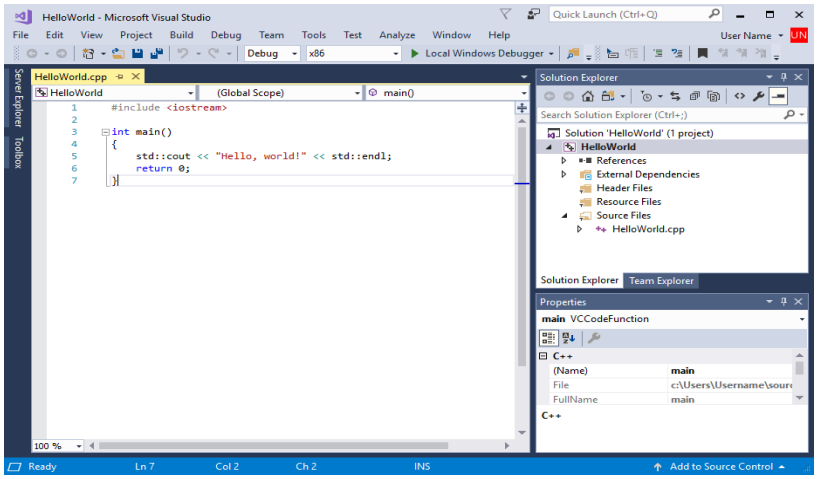
В диалоговом окне **Новый проект** должен быть шаблон **Консольное приложение** с тегами **C++**, **Windows** и **Консоль**. Если его нет, возможны две причины. Он может быть отфильтрован из списка или не установлен. Сначала проверьте раскрывающиеся списки фильтров в верхней части списка шаблонов. Выберите фильтры **C++**, **Windows** и **Консоль**. Должен появиться шаблон **консольного приложения** C++. Если этого не произошло, значит, рабочая нагрузка **Разработка классических приложений на C++** не установлена. Чтобы установить рабочую нагрузку **Разработка классических приложений на C++**, можно запустить установщик прямо из диалогового окна **Новый проект**. Чтобы запустить установщик, щелкните ссылку **Установка других средств и компонентов** внизу списка

шаблонов. Если в диалоговом окне **Контроль учетных записей пользователей** запрашиваются разрешения, выберите **Да**. В установщике должна быть выбрана рабочая нагрузка **Разработка классических приложений на C++**.
Выберите **Изменить**, чтобы обновить установку Visual Studio. Если проект с таким именем уже существует, выберите другое имя для проекта. Можно также удалить существующий проект и повторить попытку. Чтобы удалить существующий проект, удалите папку решения (папку, содержащую `helloworld.sln` файл) в проводник.

https://learn.microsoft.com/ru-ru/cpp/build/vscpp-step-1-create?view=msvc-170

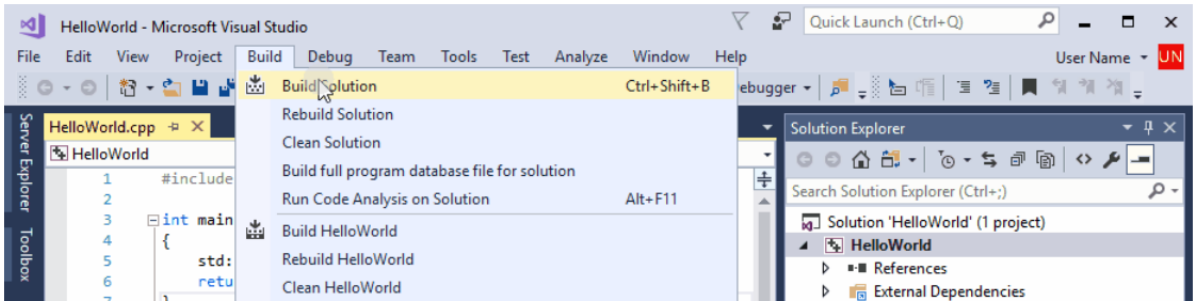
Запуск проекта приложения

Если VS выглядит следующим образом, можно приступать к сборке и запуску приложения:

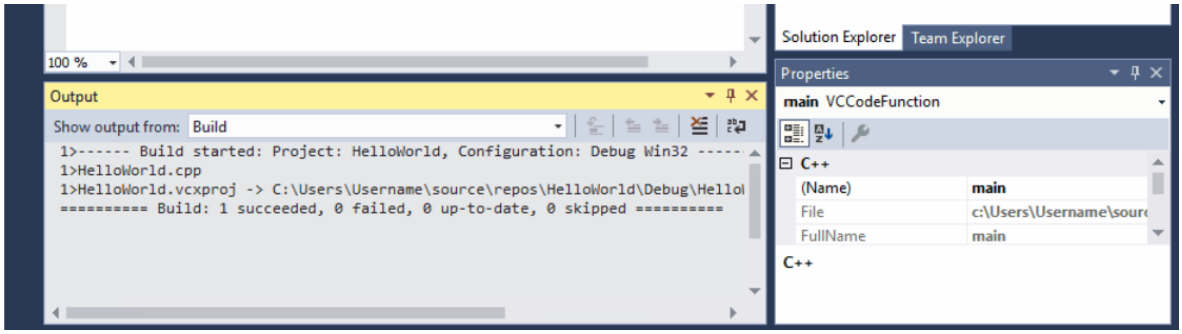


Сборка и запуск кода в Visual Studio

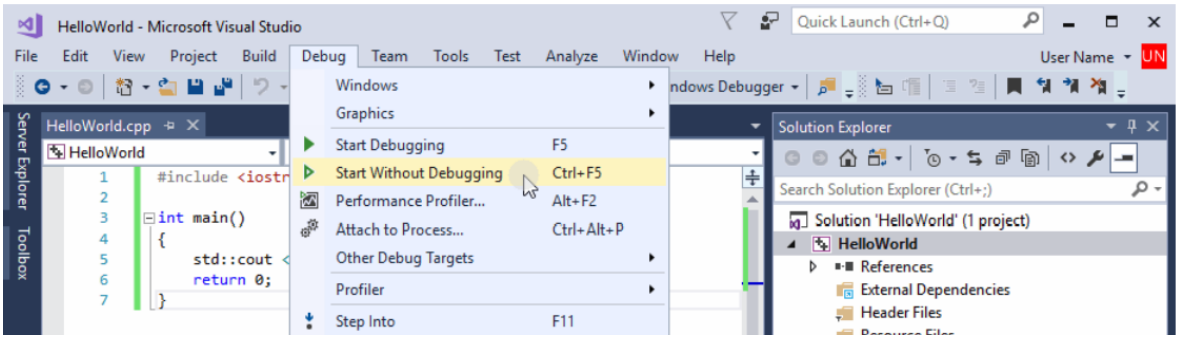
1. Для сборки проекта выберите в меню **Сборка** пункт **Собрать решение**.



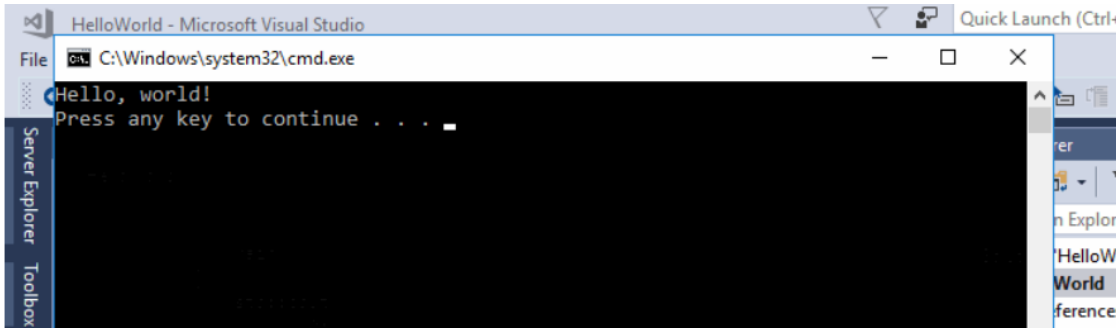
Окно **Вывод** отображает результаты процесса сборки.



2. Чтобы запустить этот код, в строке меню выберите **Отладка** и **Запуск без отладки**.



Открывается окно консоли, и запускается ваше приложение. При запуске консольного приложения в Visual Studio система выполняет код, а затем выводит сообщение "Нажмите любую клавишу, чтобы продолжить. . .", чтобы вы могли просмотреть выходные данные.



Поздравляем! Вы создали свое первое консольное приложение "Hello, world!" в Visual Studio! Нажмите любую клавишу, чтобы закрыть окно консоли и вернуться в редактор Visual Studio.

Сборка и запуск кода в Visual Studio: проблемы

Если в редакторе исходного кода отображаются красные волнистые линии, то сборка может содержать ошибки или предупреждения. Убедитесь, что код соответствует примеру в написании, пунктуации и регистре

Добавление комментариев

1. **Однострочный комментарий в C++:** чтобы сделать одну строку кода комментарием, используйте два слэша (//) в начале строки. Все, что следует за //, будет рассматриваться как комментарий и не будет выполняться компилятором.

Пример:

```
// Это однострочный комментарий
```

2. **Многострочный комментарий в C++**: чтобы закомментировать несколько строк сразу, используйте /* в начале комментируемого блока и */ в конце. Все, что находится между /* и */, будет считаться комментарием.

Пример:

```
/* Это многострочный
   комментарий, охватывающий
   несколько строк */
```

Основной ввод/вывод

cin >> для ввода в C++

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

int main()
{
    int number;
    string word;
    cout << "Enter a number: ";
    cin >> number;
    cout << number<<endl;
    cout << "Enter a word: ";
    cin >> word;
    cout << word<< endl;
}
```

Можно использовать только **cin >>** для ввода в C++, но важно понимать ограничения. **cin >>** используется для форматированного ввода. Он считывает данные из стандартного потока ввода (**cin**) и сохраняет их в заданную переменную. Однако **cin >>** прекращает чтение ввода, когда встречает пробел (например, пробел, табуляцию или символ новой строки). Это поведение делает его менее подходящим для чтения целых строк текста, содержащих пробелы. Если вы попытаетесь использовать cin >> для чтения полной строки текста с пробелами, он прочитает только до первого пробела. Еще одна проблема с использованием cin >> заключается в его поведении с символами новой строки. cin >> оставляет символ новой строки в буфере ввода.

getline для ввода в C++

Для того чтобы изменить вашу программу для использования функции getline вместо cin >>, необходимо сделать несколько изменений. Функция getline используется для чтения строки из ввода, включая пробелы, до встречи символа новой строки. Поскольку вы также читаете целое число, вам нужно будет использовать cin для целого числа, а затем cin.ignore(), чтобы очистить буфер от символа новой строки перед использованием getline. Вот как будет выглядеть ваша измененная программа:

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

int main()
{
    int number;
    string word;

    cout << "Enter a number: ";
    cin >> number;
    cout << number << endl;

    // Clear the newline character from the buffer
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    cout << "Enter a word: ";
    getline(cin, word);
    cout << word << endl;

    return 0;
}
```

В этой программе **cin.ignore(numeric_limits<streamsize>::max(), '\n')** используется для игнорирования символов до следующего символа новой строки. Это необходимо, потому что после чтения целого числа символ новой строки, созданный нажатием Enter, все еще находится в буфере ввода, и getline рассматривал бы это как окончание своего ввода, если бы он не был очищен. Рассмотрим несколько примеров, чтобы проиллюстрировать различия:

```cpp
#include <iostream>
#include <string>

using namespace std;

int main() {
    string userInput;

    // Using getline to read a line of text
    cout << "Please enter your name and father's name, (In English), for example Ivan Ivanov: ";
    getline(cin, userInput);
    cout << "You entered using getline: \"" << userInput << "\"" << endl << endl;


    // Using cin >> to read a line of text
    cout << "Again, please enter enter your name and father's name, (In English) for example Ivan Ivanov:";
    cin >> userInput;  // This will only read up to the first whitespace
    cout << "You entered using cin >>: \"" << userInput << "\"" << endl;

    return 0;
}
```

Консольное приложение на русском языке и c++

1- Кодировка консоли: наиболее распространенная проблема заключается в том, что кодировка консоли или терминала может не поддерживать кириллические символы. Вы можете установить кодировку консоли на набор символов, совместимых с кириллицей, например, UTF-8, который поддерживает русские символы.

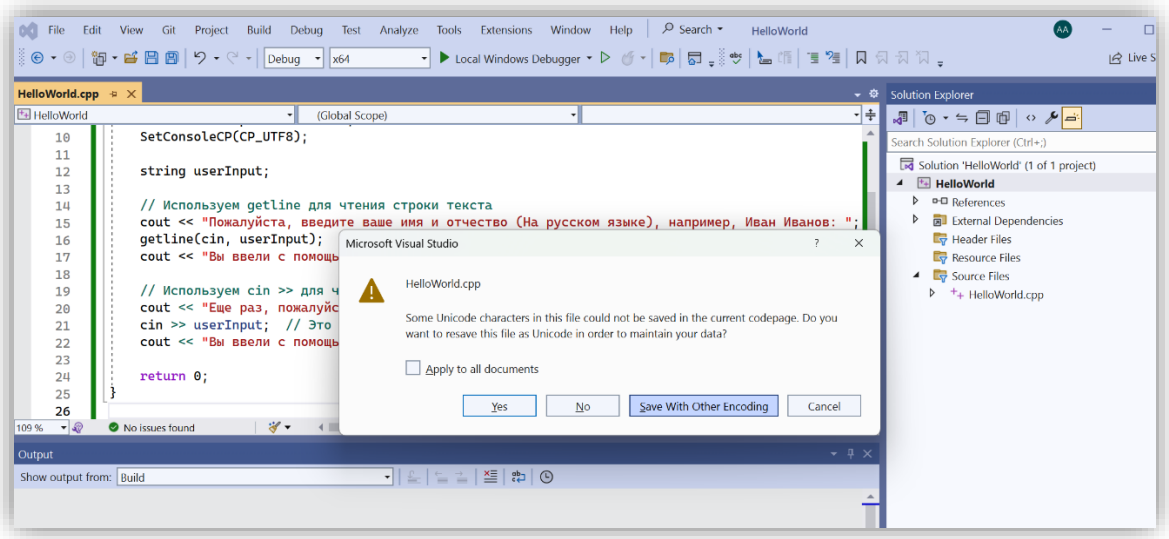
- На Windows: Вы можете изменить кодировку консоли на UTF-8, добавив SetConsoleOutputCP(CP_UTF8); в начало вашей основной функции. Вам также потребуется подключить <windows.h> для использования этой функции.
- На Linux или MacOS: Большинство терминалов в Linux и MacOS изначально поддерживают кодировку UTF-8, но вы можете убедиться в этом, запустив locale в терминале, чтобы проверить текущие настройки кодировки.

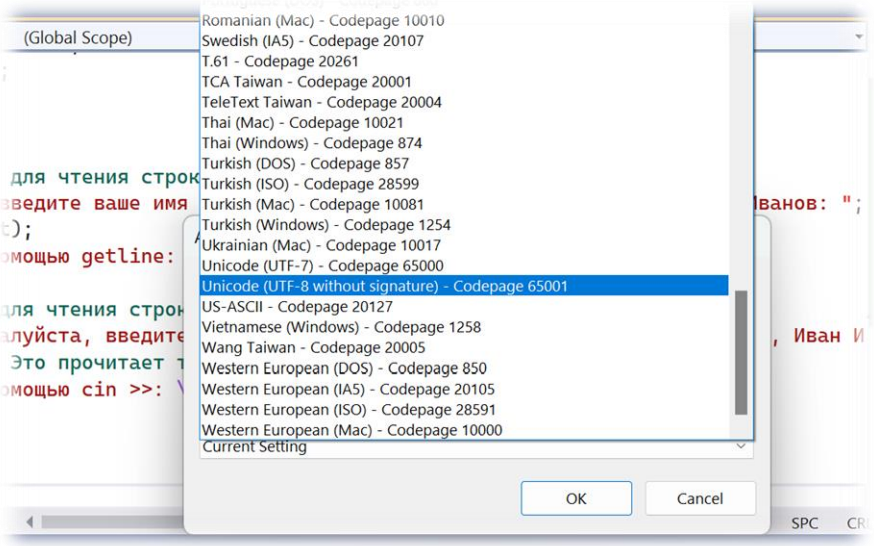
2- Кодировка исходного кода: убедитесь, что ваш файл исходного кода сохранен с кодировкой UTF-8. Обычно это настройка в вашем текстовом редакторе или IDE.
3- Поддержка шрифтов: Шрифт, используемый вашей консолью или терминалом, должен поддерживать кириллические символы. Некоторые базовые шрифты могут не включать кириллические символы, поэтому вам может потребоваться переключиться на другой шрифт в настройках консоли.
4- Системная локаль: если вы используете систему, где язык по умолчанию не установлен на язык, использующий кириллическое письмо, вы можете столкнуться с проблемами отображения русских символов. Вы можете проверить и изменить настройки системной локали в панели управления (на Windows) или в системных настройках (на Linux и MacOS).

Вот пример того, как вы можете изменить свой код C++ для Windows:

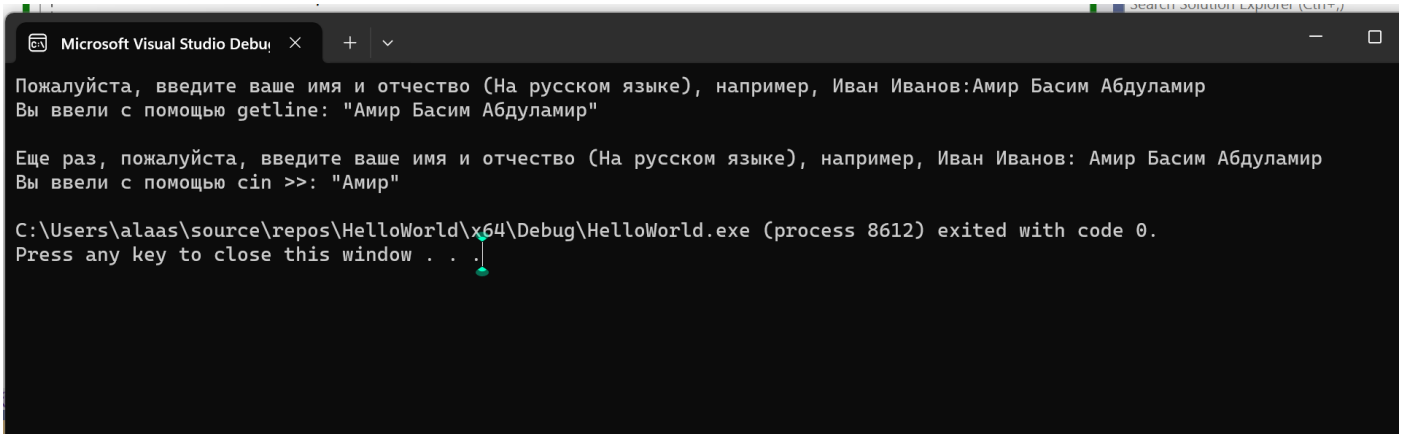
```cpp
#include <iostream>
#include <string>
#include <windows.h>

using namespace std;

int main() {
    // Set console to UTF-8 to support Cyrillic characters
    SetConsoleOutputCP(CP_UTF8);
    SetConsoleCP(CP_UTF8);

    string userInput;

    // Используем getline для чтения строки текста
    cout << "Пожалуйста, введите ваше имя и отчество (На русском языке), например, Иван Иванов: ";
    getline(cin, userInput);
    cout << "Вы ввели с помощью getline: \"" << userInput << "\"" << endl << endl;

    // Используем cin >> для чтения строки текста
    cout << "Еще раз, пожалуйста, введите ваше имя и отчество (На русском языке), например, Иван Иванов: ";
    cin >> userInput;  // Это прочитает только до первого пробела
    cout << "Вы ввели с помощью cin >>: \"" << userInput << "\"" << endl;

    return 0;
}
```

После этого при попытке сохранить файл вам должен быть показан диалог. Пожалуйста, следуйте инструкциям:





После этого при запуске программы проблем с русским языком не должно быть:



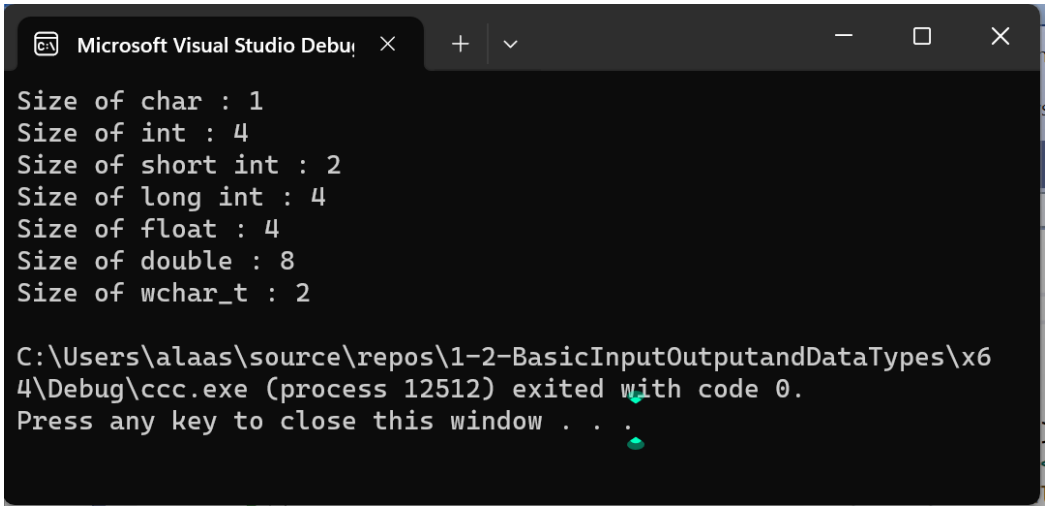
Примитивные встроенные типы (Primitive Built-in Types)

Primitive Built-in Types	Typical Bit Width	Typical Range
char	1 byte	-128 to 127 or 0 to 255 (implementation-defined)
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127

int	4 bytes (commonly)	-2147483648 to 2147483647
unsigned int	4 bytes (commonly)	0 to 4294967295
signed int	Same as int	Same as int
short int	2 bytes	-32768 to 32767
unsigned short int	2 bytes	0 to 65535
signed short int	Same as short int	Same as short int
long int	4 bytes on 32-bit, 8 bytes on 64-bit	-2147483648 to 2147483647 (32-bit), -9223372036854775808 to 9223372036854775807 (64-bit)
unsigned long int	4 bytes on 32-bit, 8 bytes on 64-bit	0 to 4294967295 (32-bit), 0 to 18446744073709551615 (64-bit)
long long int	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long long int	8 bytes	0 to 18446744073709551615
float	4 bytes	Implementation-specific (IEEE 754)
double	8 bytes	Implementation-specific (IEEE 754)
long double	12 or 16 bytes (varies)	Implementation-specific
wchar_t	2 or 4 bytes (varies)	1 wide character
bool	1 byte (commonly)	**true** or **false**

Ниже приведен пример, который позволит получить правильный размер различных типов данных на вашем компьютере:

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

int main() {
          cout << "Size of char : " << sizeof(char) << endl;
          cout << "Size of int : " << sizeof(int) << endl;
          cout << "Size of short int : " << sizeof(short int) << endl;
          cout << "Size of long int : " << sizeof(long int) << endl;
          cout << "Size of float : " << sizeof(float) << endl;
          cout << "Size of double : " << sizeof(double) << endl;
          cout << "Size of wchar_t : " << sizeof(wchar_t) << endl;

          return 0;
}
```



Базовые Условные Операторы (Basic Conditional Statements)

Инструкция if (Условный оператор if)

Инструкция if используется для проверки истинности выражения.

Если условие оценивается как истинное, то код внутри блока выполняется; в противном случае он будет пропущен. Пример:

```
if (a == 10) {

  // Code goes here

}
```

Клоз else (Ветка else)

К инструкции if можно добавить ветку else:

Если условие оценивается как истинное, выполняется код в части if.

Если условие оценивается как ложное, выполняется код в части else.

Пример:

```
if (year == 1991) {
  // This runs if it is true
}
else {
  // This runs if it is false
}
```

Инструкция Else If (Условный оператор Else If)

Можно добавить одну или несколько инструкций else if между if и else для проверки дополнительных условий.

Пример:

```
if (apple > 8) {
  // Some code here
}
else if (apple > 6) {
  // Some code here
}
else {
  // Some code here


}
```

Оператор Switch (Конструкция Switch)

Оператор switch предоставляет способ проверки выражения на соответствие различным случаям. Если найдено совпадение, начинает выполняться код внутри соответствующего блока. Ключевое слово break может использоваться для завершения выполнения данного случая.

```
switch (grade) {
  case 9:
    std::cout << "Freshman\n";
    break;
  case 10:
    std::cout << "Sophomore\n";
    break;
  case 11:
    std::cout << "Junior\n";
    break;
  case 12:
    std::cout << "Senior\n";
    break;
  default:
    std::cout << "Invalid\n";
    break;
}
```

Полный пример: калькулятор с использованием if-else-if

```
#include <iostream>

int main() {
    char operation;
    double num1, num2;

    std::cout << "Enter an operation (+, -, *, /): ";
    std::cin >> operation;

    std::cout << "Enter two numbers: ";
    std::cin >> num1 >> num2;

    if (operation == '+') {
        std::cout << "Result: " << num1 + num2 << std::endl;
    }
    else if (operation == '-') {
        std::cout << "Result: " << num1 - num2 << std::endl;
    }
    else if (operation == '*') {
        std::cout << "Result: " << num1 * num2 << std::endl;
    }
    else if (operation == '/') {
        if (num2 != 0) {
            std::cout << "Result: " << num1 / num2 << std::endl;
        }
        else {
            std::cout << "Error: Division by zero" << std::endl;
        }
    }
```

```
    else {
        std::cout << "Error: Invalid operation" << std::endl;
    }

    return 0;
}
```

Задача самостоятельной реализации 1

Перепишите приведенный выше пример калькулятора, чтобы заменить цепочку (if-else-if) на (switch).

Цикл

Эти циклы **for**, **while** и **do-while** являются основными конструкциями для повторения определенных действий в программировании на C++. Выбор между ними зависит от конкретных задач и условий выполнения кода.

1- **Цикл for** - Этот цикл используется, когда количество итераций известно заранее. Он состоит из трех частей: инициализации, условия продолжения и инкремента (или декремента).

Пример:

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

int main() {
    SetConsoleOutputCP(CP_UTF8);
    SetConsoleCP(CP_UTF8);

    int n, sum = 0;
    std::cout << "Введите число: ";
    std::cin >> n;

    for (int i = 1; i <= n; i++) {
        sum += i;
    }

    std::cout << "Сумма чисел от 1 до " << n << " равна " << sum << std::endl;
    return 0;
}
```

2- **Цикл while** - Этот цикл выполняется, пока условие истинно. Условие проверяется перед каждой итерацией.

Пример:

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

int main() {
    SetConsoleOutputCP(CP_UTF8);
    SetConsoleCP(CP_UTF8);

    int n, sum = 0, i = 1;
    std::cout << "Введите число: ";
    std::cin >> n;

    while (i <= n) {
        sum += i;
        i++;
    }

    std::cout << "Сумма чисел от 1 до " << n << " равна " << sum << std::endl;
    return 0;
}
```

3- **Цикл do-while** - Этот цикл схож с while, но условие проверяется после выполнения тела цикла, поэтому тело цикла выполнится хотя бы один раз, даже если условие изначально ложно.

Пример:

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

int main() {
    SetConsoleOutputCP(CP_UTF8);
    SetConsoleCP(CP_UTF8);

    int n, sum = 0, i = 1;
    std::cout << "Введите число: ";
    std::cin >> n;

    do {
        sum += i;
        i++;
    } while (i <= n);

    std::cout << "Сумма чисел от 1 до " << n << " равна " << sum << std::endl;
    return 0;
}
```

Задача самостоятельной реализации 2

С помощью (switch) напишите программу будет спрашивать пользователя, какой тип цикла он хочет использовать для выполнения задачи.

```
    std::cout << "Выберите тип цикла (1 - for, 2 - while, 3 - do-while): ";
    std::cin >> choice;
```

Вложенные циклы

Когда внутри тела цикла размещается другой цикл, это называется вложенным циклом. Внешний цикл выполняется один раз, и каждый раз, когда он выполняется, внутренний цикл выполняется полностью. Допустим, вы хотите напечатать таблицу умножения для чисел от 1 до 5. В этом случае можно использовать вложенные циклы for:

```
#include <iostream>

int main() {
    for (int i = 1; i <= 5; i++) {          // Внешний цикл
        for (int j = 1; j <= 5; j++) {      // Внутренний цикл
            std::cout << i * j << " ";      // Выполняется для каждой пары i и j
        }
        std::cout << std::endl;             // Переход на новую строку после завершения внутреннего цикла
    }
    return 0;
}
```