Part: 01

Introduction to C++, Installing VS Code, g++ & more

In these C++ tutorials, we are going to learn about the C++ programming language from the very basics to the industry level. This course has been designed in a way to nurture the beginners setting their feet in this discipline. So, tighten your belts, and enjoy the ride!

Before starting, let me explain to you the difference between this and any other source for learning C++:

1. This is a whole complete package in itself, covering from A to Z of the language.
2. No redundancy and monotonicity.
3. Beginner friendly, hence easy to learn.
4. No prerequisite, just a zeal for learning.

To start with, I'll be covering these things in today's tutorial:

1. What is programming and why C++?
2. Installation of Visual Studio Code
3. Installation of g++
4. Writing our first program and executing it.

What is programming, programming language, and why C++?

Programming can be understood as your instructions to the computer(machine) to solve real problems. The very basic principle of creating machines was to make life simpler and machines, on our instructions, have been able to do the same. But the distance between what we say and what the machine understands gets abridged by a programming language. This marks the importance of learning a programming language.

A programming language helps us communicate with the computer. Analogous to us humans, who need some language, be it English, Hindi, or Bangla, to talk to our people, computers too need a language to converse. Just to name a few, there is C++, C, Python, Java, etc.

Now when we start listing the names of these programming languages, the question which instinctively arises is why C++. Despite this being an 80s programming language, it never lost its sheen. C++ was an added version of C developed by Bjarne **`Stroustrup`**. It is believed to be very close to the hardware making it comparatively easy for programmers to give the instructions directly to the system without any intermediary. Another reason for C++ to be one of the most used languages is its object-orienteers. C++ is an object-oriented programming language giving it the power to create real-world software systems. You don't have to worry much about these terms, which believe me, only sound complex and ain't really! Just sit back and dive with me in this plethora of knowledge.

Installation of Visual Studio Code

Visual studio is a source code editor - free to use, provided & maintained by Microsoft. Below is the process of downloading and installing visual studio code:

Step 1: Click here and you will be redirected to the official download page of VS Code. Download the VS code according to your operating system in use. I will be downloading it for Windows 10 as shown in the below animation.

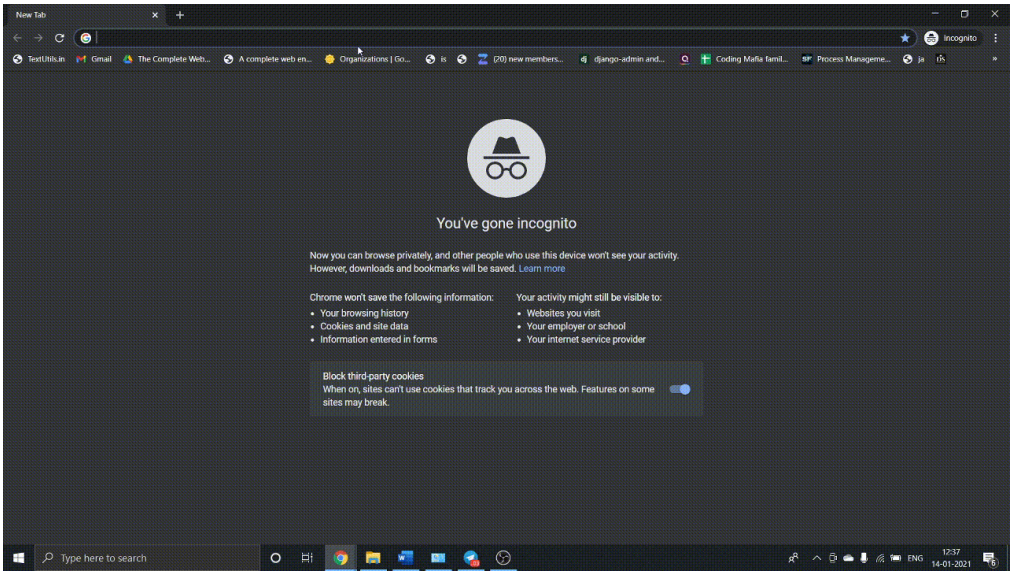


Figure 1: Visual Studio Code Website

Step 2: Once the downloading is complete, install VS code on your system like any other application.

Installation of g++

g++ is a compiler that helps us convert our source code into a .exe file. Below is the process of downloading and installing g++:

Step 1: Go to Google and search "MinGW install" and click on the MinGW link, as shown in the image below.

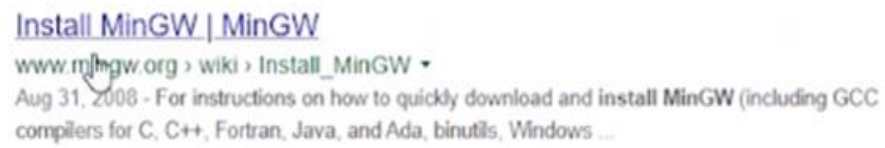


Figure 4: g++ installation from Google

Step 2: Click on the download button on the top right corner menu.

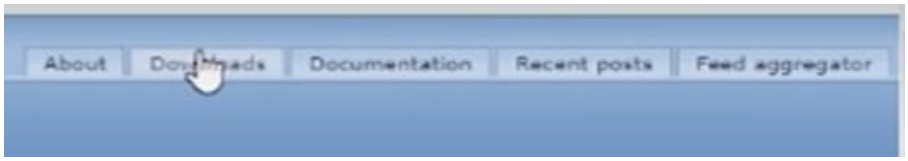


Figure 5: G++ Download Step

Step 3: After visiting the download page, click on the windows button as shown in the image below to start the downloading

Figure 6 G++ Download Step

Step 4: After the download - open the program and click "Continue" to start the installation process.

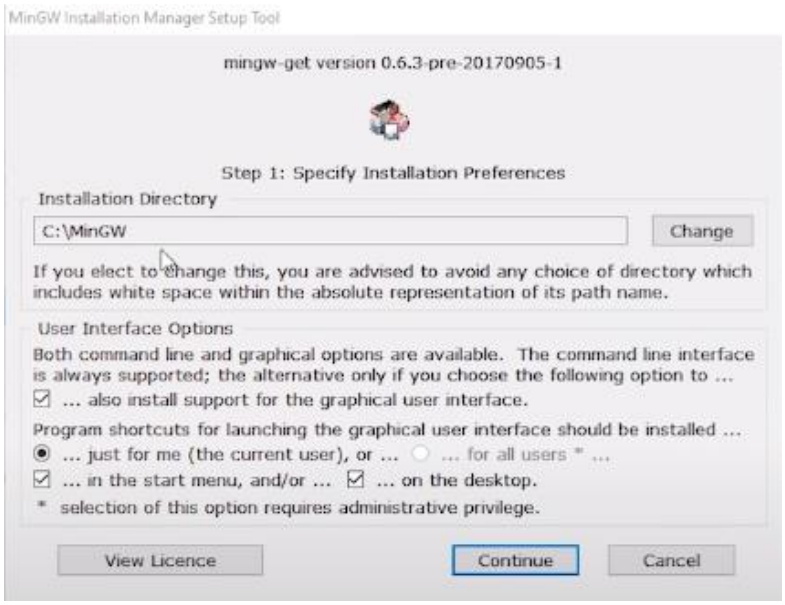


Figure 7: G++ Installation Step

Step 5: After downloading some packages, it will show you a screen, as shown in the image below. You have to mark both the boxes as in the image below, and then click on installation on the top left corner menu. Finally, click apply changes, and it will start downloading the required packages.

Figure 8: G++ Installation Step

Step 6: After finishing step 5, close the program and open C:// drive. Furthermore, locate the MinGW folder. Go to its bin directory and copy its file path, as shown in the image below.

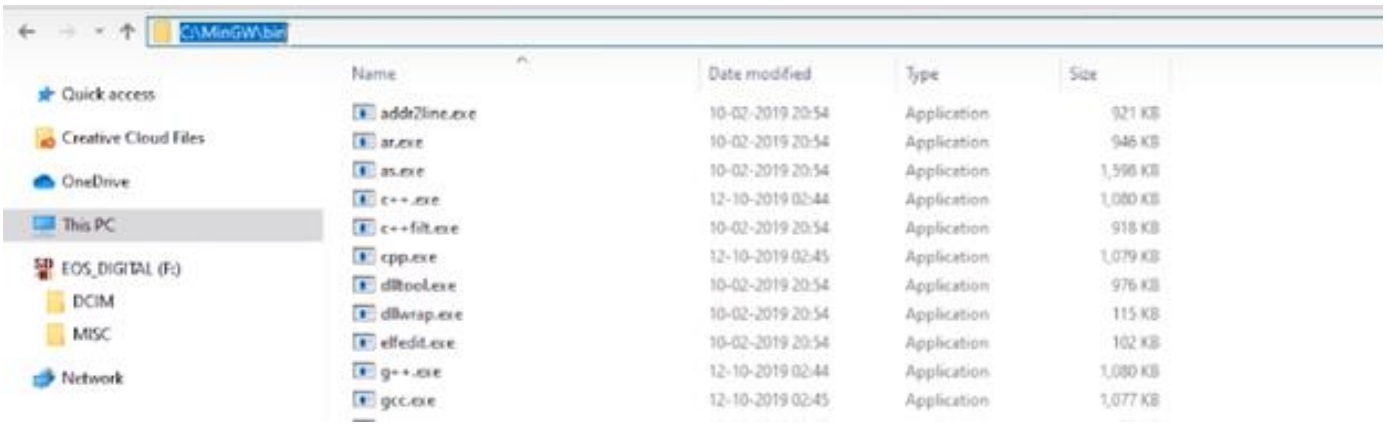


Figure 9: Copying g++ file path

Step 7: Now right-click on this pc and go to properties

Step 8: After that click on advanced system settings as shown in the image below

Figure 10: Step to Add g++ File Path

Step 9: After that click on "Environment Variables" as shown in the image below

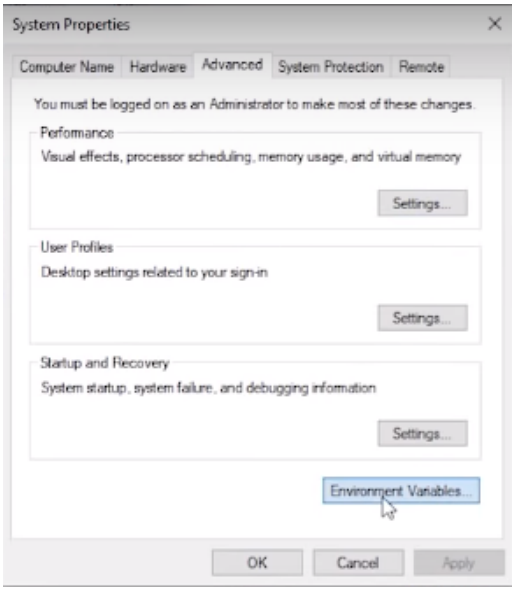


Figure 11: Step to Add G++ File Path

Step 10: After that select path and click on edit as shown in the image below

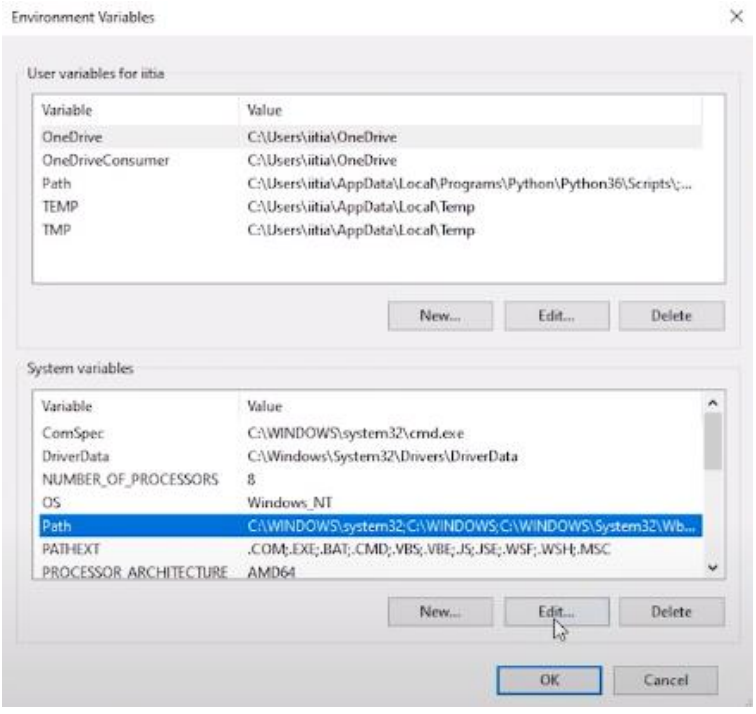


Figure 12: Step to Add g++ file path

Step 11: Then, click on new and paste the file path and click ok as shown in the image below

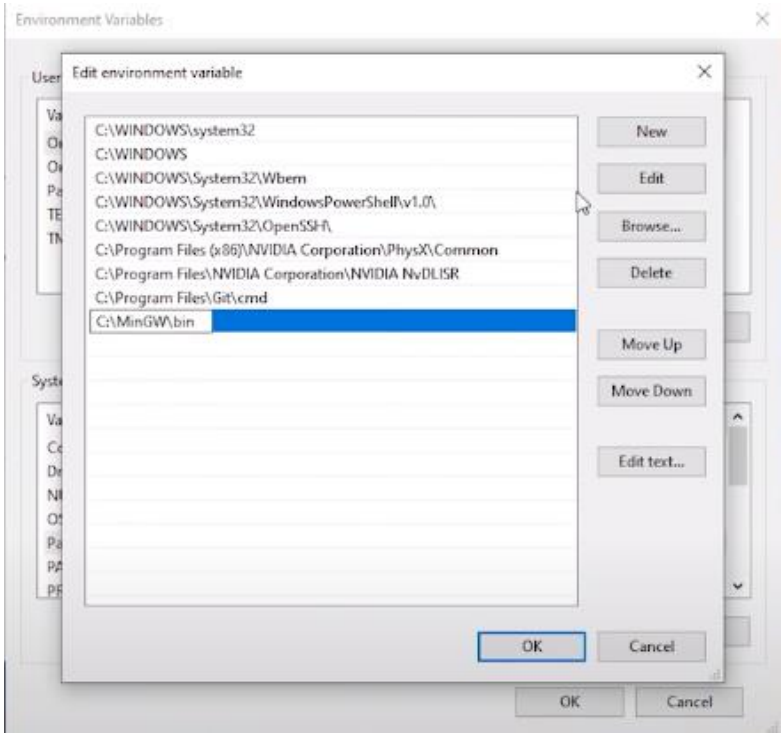


Figure 13: Adding g++ File Path

After adding the file path now, our g++ compiler is ready, and we can start coding now.

Writing Our First Program and Executing It

To write our first program, we need Visual studio code, which is a source code editor. Create a folder and then right-click inside the folder and click on "**open with code**" as shown below:

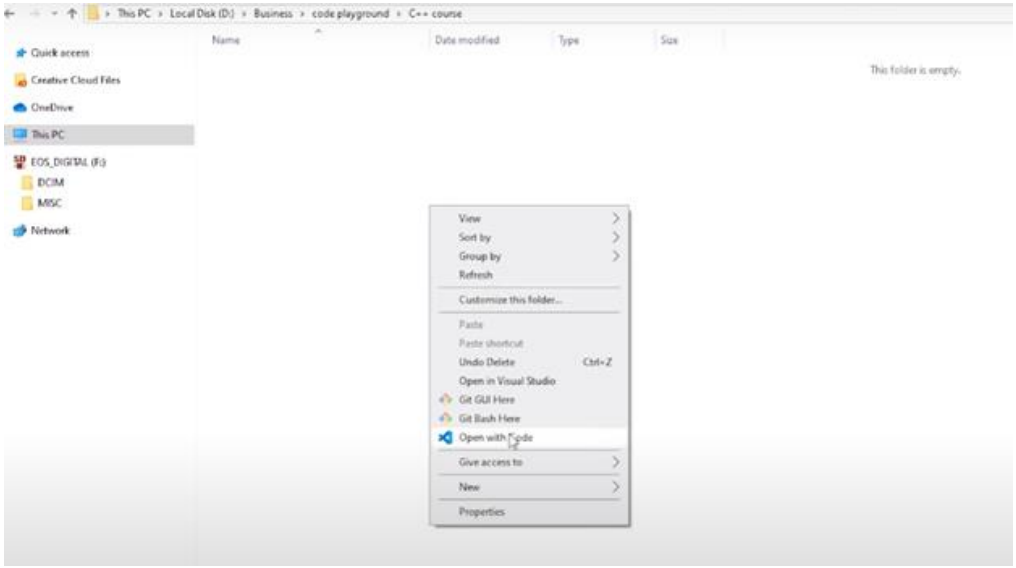


Figure 14: Opening Visual Studio Code in a folder

Following these steps will open your visual studio code with that folder as the context. After opening the VS Code, you have to install some extensions. Go to the extension menu and search "**C/C++,**" and it will show you this extension. C/C++, as well as the other extensions we add, will make our life easy while learning C++. Click on the install button, and it will start installing the extension for you, as shown in the image below.

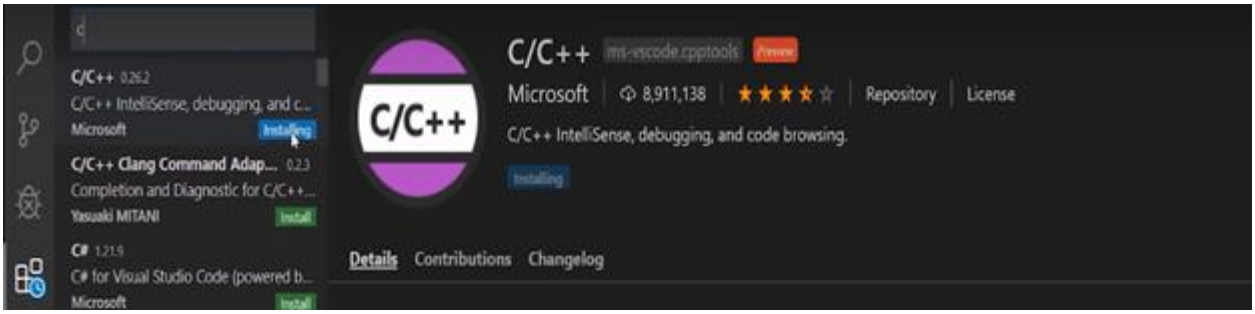


Figure 15: Installing Extension

This extension will help us in writing code through features such as auto-complete or auto-dropdown suggestions. Let us install one more extension, which will help us run our programs quickly. Go to the Extensions tab in the top left corner and search "**Code runner.**" After that, click on install.

Figure 16: Installing Extension

Now we have to create our program file and start writing our code. To create a program file, you have to go to the File menu > then click on the file button, as shown in the image below.

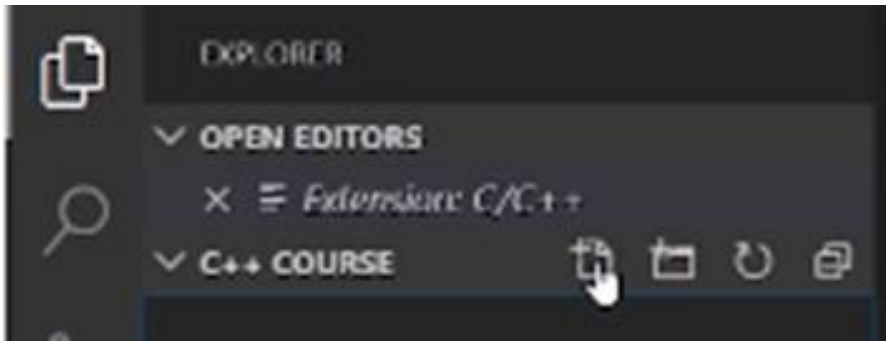


Figure 17: Creating a Program File

After clicking on the file button, it will ask you for the file name. Give the name of the file as **`"tutorial1.cpp"`** and press enter. Now the code file will be created, and you can start writing your program.

Figure 18: Running Code

In today's tutorial, we are not going to learn anything about what this code is all about. We will learn these things step by step in our upcoming lectures. Now to execute this code, press the run button, as shown in the image above, and it will give you the output as shown in the image below.

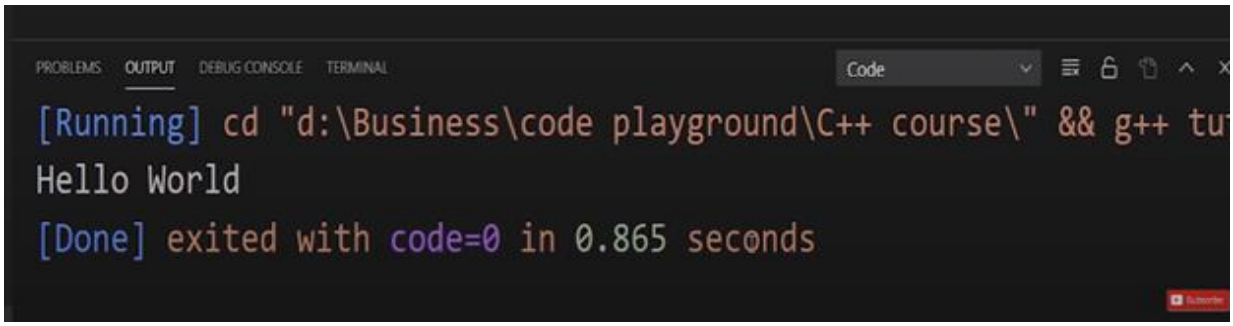


Figure 19: Program Output

Thank your friends for starting to learn C++ with me, hope you liked the tutorial. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. I hope you enjoy them.

In the next tutorial, we'll be talking more about the basic structures of a C++ program, see you there, till then keep coding.

Code as described/written in the video

```
#include<iostream>

int main(){
    std::cout<<"Hello World";
    return 0;
}
```

Part: 02

Basic Structure of a C++ Program

In this series of C++ tutorials, we will visualize the basic structure of a C++ program. In our last lesson, we discussed C++, and we had also learned about how to download and install visual studio code and g++; If you haven't read the previous lecture, feel free to navigate through the course content!

In our previous lecture, we had written and executed a small C++ program. Today we are going to discuss that program in more detail.

Figure 1: C++ Code from the previous tutorial

As you can see in the image, this was the program that we had executed in our previous lecture. Now we will discuss what each line of code in the program does.

1. Let's start with the 1st line of code "**`#include<iostream>`**" - this whole text is called a **header file**.

In this line of code **`include`** is a keyword used to add libraries in our program. "**`iostream`**" is the name of a library, added to our program. The **iostream** library helps us to get input data and show output data. The iostream library also has many more uses; it is not only limited to input and output but for now, we will not go into more detail on this.

2. One more thing to consider here is that the 2nd line of code is blank; there is no code written. The thing to consider here is that it doesn't matter how many lines you have left empty in a C++ Program. The compiler will ignore those lines and only check the ones with the code.

3. Now onto the 3rd line of code "**`int main() {`**" - In this line of code, "**`int`**" is a return type which is called integer and "**`main()`**" is a function, the brackets "**`()`**" denotes that it is a function. The curly brace "**`{`**" indicates that it is an opening of a function, and the curly brace "**`}`**" indicates that it is the closing of a function. Here I will give you **an example to** better understand the functionality of "**`int main().`**" Imagine that you own a coffee shop, and you have some employees who work for you. Let's name **`(Anna, Blake, Charlie)`** as the three employees. The function of Anna is to take orders, the function of Blake is to make coffee, and Charlie's function is to deliver coffee. Now when Anna gets a coffee order, she will call Blake to make the coffee, and when the coffee is ready, Blake will call Charlie to deliver the coffee. In this scenario, we can say that **Anna is the primary function** from which all the other tasks will start, and **coffee is our return**

value (Something charlie finally gives to Blake). In this line of code, "**`main`**" is a reserved keyword, and we cannot change it to some other keyword. This was just an analogy, and you will understand this very well in upcoming tutorials.

4. Now let's talk about the 4th line of code '**`std::cout<<" hello world";`**' - In this line of code "**`std`**" is a namespace, "**`::`**" is the scope resolution operator and "**`cout<<`**" is a function which is used to output data, "**`hello world`**" is our output string and "**`;`**" is used to end a statement. The code "**`std::cout`**" tells the compiler that the "**`cout`**" identifier resides in the std namespace. Main key points here are:

- We can write as many statements as you want in a single line, but I recommend you write one statement per line to keep the code neat and clean.
- Anything which is written between double quotation " " is a string literal (More on strings later).

5. Now let's talk about the 5th line of code "**`return 0`**" - In this line of code, the return keyword will return 0 as an integer to our main function "**`int main()`**" as we have discussed before. Returning 0 as a value to the main function means successful termination of the program.

So that was the anatomy of a C++ program. I hope you understood the functions of various parts in a C++ program.

Part: 03

Variables & Comments in C++

In this tutorial, we will learn about the variables and comments in the C++ language. In our last lesson, we discussed the basic structure of a C++ program, where we understood the working of the C++ code line by line. If you haven't read the previous lecture, make sure to navigate through the course content section.

We are going to cover two important concepts of C++ language:

- **Variables in C++**
- **Comments in C++**

Before explaining the concept of variables and comments, I would like to clarify two more ideas: **low level** and **high level.** To make it easy to understand, let's consider this scenario - when we go to Google search engine and search for some queries, Google displays us some websites according to our question. Google does this for us at a very high level. We don't know what's happening at the low level until we look into Google servers (at a low level) and further to the level where the data is in the form of 0s/1s. The point I want to make here is that low level means nearest to the hardware, and a high level means farther from the hardware with a lot of layers of abstraction.

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

// This program was created by Code With Harry
/* this
is
a
multi
line
comment */
int main(){
    int sum = 6;
    cout<< "Hello world"<< sum;
    return 0;
}
```

Figure 1: C++ Sample Code

Variables in C++

Variables are containers to store our data. To make it easy to understand, I will give a scenario: to store water, we use bottles, and to store sugar, we use a box. In this scenario, the bottle and box are containers that are storing water and sugar; the same is the case with variables; they are the containers for data types. As there are many types of data types, for example, "**`int`**" is used for integers, the "**`float`**" is used for floating-point numbers, "**`char`**" is used for character, and many more data types are available, we will discuss them in upcoming lectures. The main point here is that these variables store the values of these data types. Figure 1 shows an example of a variable. "**`sum`**" is taken as an integer variable, which will store a value 6, and writing sum after the "**`cout`**" statement will show us the value of "**`sum`**" on the output window.

Comments in C++

A comment is a human-readable text in the source code, which is ignored by the compiler. There are two ways to write comments.

Single-Line Comments: 1st way is to use **"//"** before a single line of text to make it unpartable by the compiler.

Multi-Line Comments: 2nd way is to use **"/*"** as the opening and **"*/"** as the closing of the comment. We then write text in between them. If we write text without this, the compiler will try to read the text and will end up giving an error. Figure 1 shows examples of these comments.

Code as described/written in the video

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

// This program was created by Code With Harry
/* this
is
a
multi
line
comment */
int main(){
    int sum = 6;
    cout<< "Hello world"<< sum;
    return 0;
}
```

Part: 04

Variable Scope & Data Types in C++

In this series of C++ tutorials, we will visualize the variable scope and data types in the C++ language in this lecture. In our last lesson, we discussed the variable's role and comments. In this C++ tutorial, we are going to cover two important concepts of C++:

- **Variable Scope**
- **Data Types**

Before explaining the concept of variable scope, I would like to clarify about variables a little more. Variable can be defined as a container to hold data. Variables are of different types, for example:

1. **`Int`**-> Int is used to store integer data e.g (-1, 2, 5,-9, 3, 100).
2. **`Float`**-> Float is used to store decimal numbers e.g (0.5, 1.05, 3.5, 10.5)
3. **`Char`**-> Char is used to store a single character e.g. ('a', 'b',' c', 'd')
4. **`Boolean`**-> Boolean is used to store "true" or "false."
5. **`Double`**-> Double is also used to store decimal numbers but has more precision than float, e.g. (10.5895758440339...)

Here is an example to understand variables: int sum = 34; means that sum is an integer variable that holds value '34' in memory.

Syntax for Declaring Variables in C++

```
1. Data_type Variable_name = Value;
2. Ex: int a=4; char letter = ‘p’;
3. Ex: int a=4, b=6;
```

Variable Scope

The scope of a variable is the region in the program where the existence of that variable is valid. For example, consider this analogy - if one-person travels to another country illegally, we will not consider that country as its scope because he doesn't have the necessary documents to stay in that country.

Base on scope, variables can be classified into two types:

- **Local variables**
- **Global variables**

Local variables:

Local variables are declared inside the braces of any function and can be assessed only from there.

Global variables:

Global variables are declared outside any function and can be accessed from anywhere.

Data Types

Data types define the type of data that a variable can hold; for example, an integer variable can hold integer data, a character can hold character data, etc.

Data types in C++ are categorized into three groups:

- **Built-in**
- **User-defined**
- **Derived**

1. Built-in Data Types in C++:

- Int
- Float
- Char
- Double
- Boolean

2. User-Defined Data Types in C++:

- Struct
- Union
- Enum

Note: We will discuss the concepts of user-defined data types in another lecture. For now, understanding that these are user-defined data types is enough.

3. Derived Data Types in C++:

- Array
- Pointer
- Function

Note: We will discuss the concept of derived data types in another lecture. For now, understanding that these are derived data types is enough.

Practical Explanation of Initializing Variables

We have discussed a lot in theory now; we will see the actual code and its working. The code for initializing variables is shown in Figure 1.

```
5  int main(){
6      // int a = 14;
7      // int b = 15;
8      int a=14, b=15;
9      float pi=3.14;
10     char c ='d';
11     cout<<"This is tutorial 4.\nHere the value of a is "<<a<<".\nThe
       value of b is "<< b;
12     cout<<"\nThe value of pi is: "<<pi;
13     cout<<"\nThe value of c is: "<<c;
14     return 0;
15 }
```

Figure 1: initializing variables and printing their values

In this code, we have initialized different types of variables and then printed them on screen. At line no 6 and 7, we initialized two integer variables a, and b, but they are commented out for compiler to ignore them. At line no 8, we have again initialized two integer variables a, and b, but this time they both are on the same line

separated by a comma. The main thing to note here is that variables can be initialized on separate lines or in a single line. At line no 9, we have initialized a float variable pi and assigned a decimal value 3.14 to it. At line no 10, we have initialized a character variable c and assigned a character'd' to it. At line no 11, we have printed the value of a, and b. The main thing to note here is that "**/n**" is used to print a new line. At line no 12, we have printed the value of pi. Similarly, in line no 13, the value of c is printed. The output for this program is shown in figure 2.

Figure 2: Output of the Program

Practical Explanation of Variables Scope

We have discussed the variable scope in theory now; we will see its actual code and working. So the code for variables scope is shown in figure 3.

```
# include<iostream>

using namespace std;
int glo = 6;
void sum(){
    int a;
    cout<< glo;
    }

int main(){
    int glo=9;
    glo=78;
    // int a = 14;
    // int b = 15;
    int a=14, b=15;
    float pi=3.14;
    char c ='d';
    bool is_true = false;
    sum();
    cout<<glo<< is_true;
    // cout<<"This is tutorial 4.\nHere the value of a is "<<a<<".\nThe value of b is "<< b;
    // cout<<"\nThe value of pi is: "<<pi;
    // cout<<"\nThe value of c is: "<<c;
    return 0;
}
```

Figure 3: Variables Scope Code

In this piece of code, we have initialized two **"glo"** variables. 1st variable is outside the main function, and the 2nd variable is inside the main function. The value assigned to the **"glo"** variable outside the main function is 6, and the value assigned to the **"glo"** variable inside the main function is 9. One thing to note here is that in the main function, we have again assigned a value 78 to the variable **"glo"** which will update the previous value 9.

After initializing the **"glo"** variables, we had output the **"glo"** variables at two places in our program the 1st place is inside the main function, and the 2nd place is inside the sum function.

The main thing to note here is that:

- When the **"cout"** will run inside the sum function, it will check for **"glo"** variable value inside the sum function. As we can see that there is no **"log"** variable initialized inside the sum function, it will check for the **"glo"** variable outside of the sum function scope, which we call a global scope. As we can see, that "glo" function is initialized in the global scope with the value 6, so the sum function will take that value.
- When the **"cout"** will run inside the main function, it will check for **"glo"** variable value inside the main function first, and as we can see that there is a **"glo"** variable initialized inside the main function scope which is a local scope, it will use that value.

The output of this program is shown in figure 4.

Figure 4: Variable Scope Output

As we can see that the output is **"6781"**. The output 6 is from the cout of **"glo"** in sum function, the output 78 is from the cout of **"glo"** in the main function, and the output 1 is from the cout of **"is_true"** Boolean variable in the main function.

Rules to Declare Variables in C++

- Variable names in C++ language can range from 1 to 255
- Variable names must start with a letter of the alphabet or an underscore
- After the first letter, variable names can contain letters and numbers
- Variable names are case sensitive
- No spaces and special characters are allowed
- We cannot use reserved keywords as a variable name

Code as described/written in the video

```
# include<iostream>

using namespace std;
int glo = 6;
void sum(){
    int a;
    cout<< glo;
    }

int main(){
    int glo=9;
    glo=78;
    // int a = 14;
    // int b = 15;
    int a=14, b=15;
    float pi=3.14;
    char c ='d';
    bool is_true = false;
    sum();
    cout<<glo<< is_true;
    // cout<<"This is tutorial 4.\nHere the value of a is "<<a<<".\nThe value of b
is "<< b;
    // cout<<"\nThe value of pi is: "<<pi;
    // cout<<"\nThe value of c is: "<<c;
    return 0;
}
```

Part: 05

C++ Basic Input/ Output & More

In this tutorial, we will visualize basic input and output in the C++ language. In our last lesson, we discussed the variable's scope and data types. In this C++ tutorial, we are going to cover basic input and output:

Basic Input and Output in C++

C++ language comes with different libraries, which helps us in performing input/output operations. In C++ sequence of bytes corresponding to input and output are commonly known as streams. There are two types of streams:

Input stream

In the input stream, the direction of the flow of bytes occurs from the input device (for ex keyboard) to the main memory.

Output stream

In output stream, the direction of flow of bytes occurs from main memory to the output device (for ex-display)

Practical Explanation of Input/ Output

We will see the actual code for input/output, and it's working. Consider the code below:

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

int main()
{
    int num1, num2;
    cout<<"Enter the value of num1:\n"; /* '<<' is called Insertion operator */
    cin>>num1; /* '>>' is called Extraction operator */

    cout<<"Enter the value of num2:\n"; /* '<<' is called Insertion operator */
    cin>>num2; /* '>>' is called Extraction operator */

    cout<<"The sum is "<< num1+num2;

    return 0;
}
```

Figure 1: Basic input/output program

In this piece of code, we have declared two integer variables "**num1**" and "**num2**". Firstly we used "**cout**" to print "**Enter the value of num1:**" as it is on the screen, and then we used "**cin**" to take the input in "**num1**" at run time from the user.

Secondly, we used "**cout**" to print "**Enter the value of num2:**" as it is on the screen, and then we used "**cin**" to take the input in "**num2**" at run time from the user.

In the end, we used "**cout**" to print "**The sum is**" as it is on the screen and also gave the literal "**num1+num2**" which will add the values of both variables and print it on the screen.

The output of the following program is shown in figure 2.

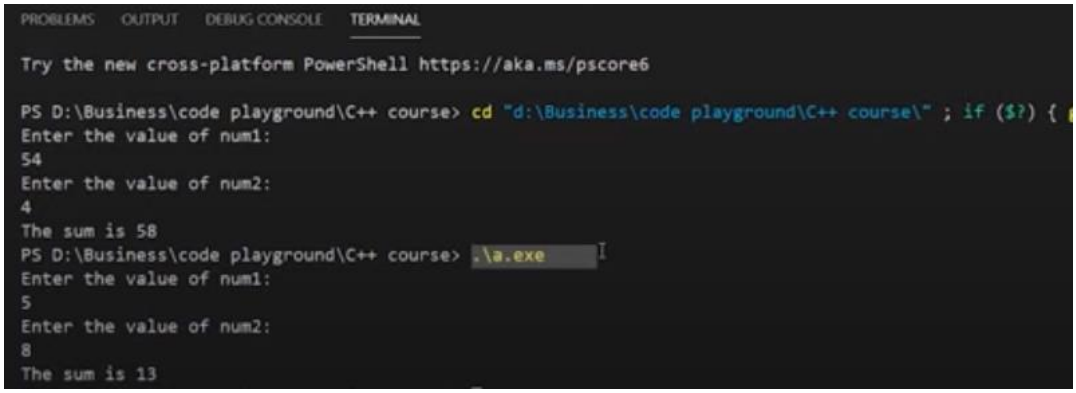


Figure 2: Output of the Program

We have executed our program two times, which can be seen in figure 2. In our 1st execution, we had input the value **"54"** for the variable **"num1"** and value **"4"** for the variable **"num2"**. This gives us the sum of both numbers as **"58"**.

In our 2nd execution, we had input the value **"5"** for the variable **"num1"** and value **"8"** for the variable **"num2"**. This gives us the sum of both numbers as **"13"**.

Important Points

1. The sign **"<<"** is called insertion operator
2. The sign **">>"** is called extraction operator
3. **"cout"** keyword is used to print
4. **"cin"** keyword is used to take input at run time.

Reserved keywords in C++

Reserved keywords are those keywords that are used by the language itself, which is why these keywords are not available for re-definition or overloading. In short, you cannot create variables with these names. A list of reserved keywords is shown in figure 3.

This is a list of reserved keywords in C++. Since they are used by the language, these keywords are not available for re-definition or overloading.

A – C	D – P	R – Z
alignas (since C++11)	**decltype** (since C++11)	**reflexpr** (reflection TS)
alignof (since C++11)	**default** (1)	**register** (2)
and	**delete** (1)	**reinterpret_cast**
and_eq	**do**	**requires** (since C++20)
asm	**double**	**return**
atomic_cancel (TM TS)	**dynamic_cast**	**short**
atomic_commit (TM TS)	**else**	**signed**
atomic_noexcept (TM TS)	**enum**	**sizeof** (1)
auto (1)	**explicit**	**static**
bitand	**export** (1) (3)	**static_assert** (since C++11)
bitor	**extern** (1)	**static_cast**
bool	**false**	**struct** (1)
	float	

`break`
`case`
`catch`
`char`
`char8_t` (since C++20)
`char16_t` (since C++11)
`char32_t` (since C++11)
`class` (1)
`compl`
`concept` (since C++20)
`const`
`consteval` (since C++20)
`constexpr` (since C++11)
`constinit` (since C++20)
`const_cast`
`continue`
`co_await` (since C++20)
`co_return` (since C++20)
`co_yield` (since C++20)
	`for`
	`friend`
	`goto`
	`if`
	`inline` (1)
	`int`
	`long`
	`mutable` (1)
	`namespace`
	`new`
	`noexcept` (since C++11)
	`not`
	`not_eq`
	`nullptr` (since C++11)
	`operator`
	`or`
	`or_eq`
	`private`
	`protected`
	`public`
		`switch`
		`synchronized` (TM TS)
		`template`
		`this`
		`thread_local` (since C++11)
		`throw`
		`true`
		`try`
		`typedef`
		`typeid`
		`typename`
		`union`
		`unsigned`
		`using` (1)
		`virtual`
		`void`
		`volatile`
		`wchar_t`
		`while`
		`xor`
		`xor_eq`

- (1) — meaning changed or new meaning added in C++11.
- (2) — meaning changed in C++17.
- (3) — meaning changed in C++20.

Note that **`and`**, **`bitor`**, **`or`**, **`xor`**, **`compl`**, **`bitand`**, **`and_eq`**, **`or_eq`**, **`xor_eq`**, **`not`**, and **`not_eq`** (along with the digraphs **`<%`**, **`%>`**, **`<:`**, **`:>`**, **`%:`**, **and** **`%:%:`**) provide an alternative way to represent standard tokens.

In addition to keywords, there are identifiers with special meaning, which may be used as names of objects or functions, but have special meaning in certain contexts.

`final` (C++11)
`override` (C++11)
`transaction_safe` (TM TS)
`transaction_safe_dynamic` (TM TS)
`import` (C++20)
`module` (C++20)

Also, all identifiers that contain a double underscore __ in any position and each identifier that begins with an underscore followed by an uppercase letter is always reserved and all identifiers that begin with an underscore are reserved for use as names in the global namespace. See identifiers for more details.

The namespace std is used to place names of the standard C++ library. See Extending namespace std for the rules about adding names to it.

The name **posix** is reserved for a future top-level namespace. The behaviour is undefined if a program declares or defines anything in that namespace.	(since C++11)

The following tokens are recognized by the preprocessor when in context of a preprocessor directive:

if	**ifdef**	**include**	**defined**	**export** (C++20)
elif	**ifndef**	**line**	**__has_include** (since C++17)	**import** (C++20)
else	**define**	**error**	**__has_cpp_attribute** (since C++20)	**module** (C++20)
endif	**undef**	**pragma**		

ASCII Table

Introducton: I adapted this information from a web site and I have made it available locally. ASCII stands for American Standard Code for Information Interchange. Below is the ASCII character table, including descriptions of the first 32 characters. ASCII was originally designed for use with teletypes, and so the descriptions are somewhat obscure and their use is frequently not as intended.

Java actually uses Unicode, which includes ASCII and other characters from languages around the world.

ASCII Table

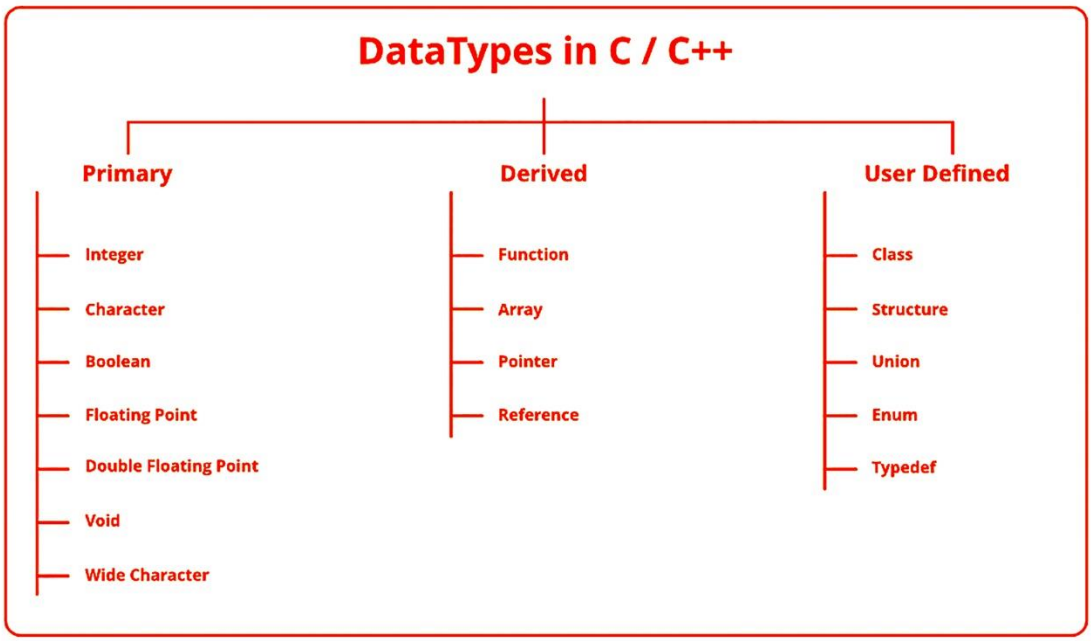
- Dec = Decimal Value
- Char = Character
- '5' has the int value 53
- if we write '5'-'0' it evaluates to 53-48, or the int 5
- if we write char c = 'B'+32; then c stores 'b'

Dec → Char	Dec → Char	Dec → Char	Dec → Char
0 → **NUL (null)**	32 → **SPACE**	64 → **@**	96 → **`**
1 → **SOH (start of heading)**	33 → **!**	65 → **A**	97 → **a**
2 → **STX (start of text)**	34 → **"**	66 → **B**	98 → **b**
3 → **ETX (end of text)**	35 → **#**	67 → **C**	99 → **c**
4 → **EOT (end of transmission)**	36 → **$**	68 → **D**	100 → **d**
5 → **ENQ (enquiry)**	37 → **%**	69 → **E**	101 → **e**
6 → **ACK (acknowledge)**	38 → **&**	70 → **F**	102 → **f**
7 → **BEL (bell)**	39 → **'**	71 → **G**	103 → **g**
8 → **BS (backspace)**	40 → **(**	72 → **H**	104 → **h**
9 → **TAB (horizontal tab)**	41 → **)**	73 → **I**	105 → **i**
10 → **LF (NL line feed, new line)**	42 → *****	74 → **J**	106 → **j**
11 → **VT (vertical tab)**	43 → **+**	75 → **K**	107 → **k**
12 → **FF (NP form feed, new page)**	44 → **,**	76 → **L**	108 → **l**
13 → **CR (carriage return)**	45 → **-**	77 → **M**	109 → **m**
	46 → **.**	78 → **N**	110 → **n**

14 → SO (shift out)	47 → /	79 → O	111 → o
15 → SI (shift in)	48 → 0	80 → P	112 → p
16 → DLE (data link escape)	49 → 1	81 → Q	113 → q
17 → DC1 (device control 1)	50 → 2	82 → R	114 → r
18 → DC2 (device control 2)	51 → 3	83 → S	115 → s
19 → DC3 (device control 3)	52 → 4	84 → T	116 → t
20 → DC4 (device control 4)	53 → 5	85 → U	117 → u
21 → NAK (negative acknowledge)	54 → 6	86 → V	118 → v
22 → SYN (synchronous idle)	55 → 7	87 → W	119 → w
23 → ETB (end of trans. block)	56 → 8	88 → X	120 → x
24 → CAN (cancel)	57 → 9	89 → Y	121 → y
25 → EM (end of medium)	58 → :	90 → Z	122 → z
26 → SUB (substitute)	59 → ;	91 → [	123 → {
27 → ESC (escape)	60 → <	92 → \	124 → |
28 → FS (file separator)	61 → =	93 →]	125 → }
29 → GS (group separator)	62 → >	94 → ^	126 → ~
30 → RS (record separator)	63 → ?	95 → _	127 → DEL
31 → US (unit separator)			

C++ Data Types

All variables use data-type during declaration to restrict the type of data to be stored. Therefore, we can say that data types are used to tell the variables the type of data it can store. Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on the data-type with which it is declared. Every data type requires a different amount of memory.



Attention reader! All those who say programming isn't for kids, just haven't met the right mentors yet. Join the Demo Class for First Step to Coding Course, specifically designed for students of class 8 to 12.
The students will get to learn more about the world of programming in these free classes which will definitely help them in making a wise career choice in the future.

Data types in C++ is mainly divided into three types:

1. **Primitive Data Types**: These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char, float, bool etc. Primitive data types available in C++ are:
    - Integer
    - Character
    - Boolean
    - Floating Point
    - Double Floating Point
    - Valueless or Void
    - Wide Character
2. **Derived Data Types:** The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:
    - Function
    - Array
    - Pointer
    - Reference
3. **Abstract or User-Defined Data Types**: These data types are defined by user itself. Like, defining a class in C++ or a structure. C++ provides the following user-defined datatypes:
    - Class
    - Structure
    - Union
    - Enumeration
    - Typedef defined DataType

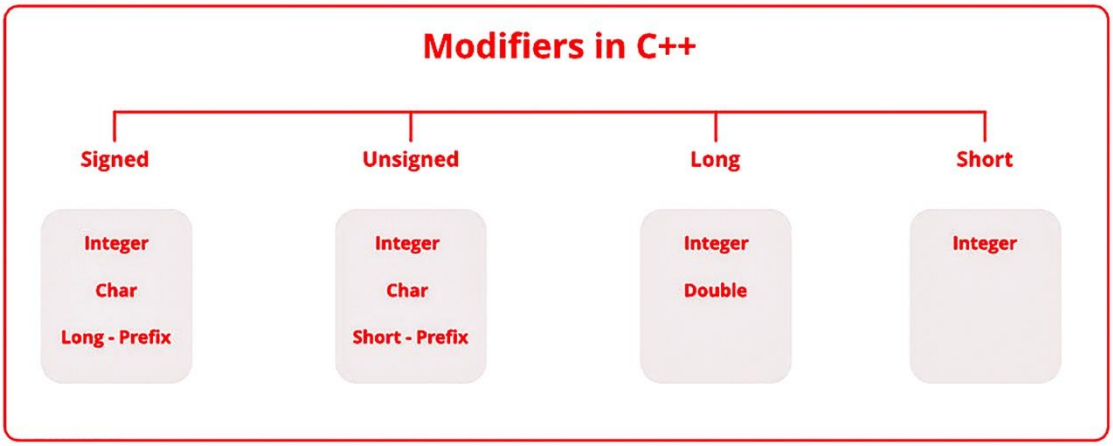
This article discusses primitive data types **available in C++.**

- **Integer**: Keyword used for integer data types is **int**. Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.
- **Character**: Character data type is used for storing characters. Keyword used for character data type is **char**. Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.
- **Boolean**: Boolean data type is used for storing **`boolean`** or logical values. A **`boolean`** variable can store either *true* or *false*. Keyword used for **`boolean`** data type is **bool**.

- **Floating Point**: Floating Point data type is used for storing single precision floating point values or decimal values. Keyword used for floating point data type is **float**. Float variables typically requires 4 byte of memory space.
- **Double Floating Point**: Double Floating-Point data type is used for storing double precision floating point values or decimal values. Keyword used for double floating-point data type is **double**. Double variables typically require 8 byte of memory space.
- **void**: Void means without any value. void datatype represents a valueless entity. Void data type is used for those function which does not returns a value.
- **Wide Character**: Wide character data type is also a character data type but this data type has size greater than the normal 8-bit datatype. Represented by **`wchar_t`**. It is generally 2 or 4 bytes long.

Datatype Modifiers

As the name implies, datatype modifiers are used with the built-in data types to modify the length of data that a particular data type can hold.



Data type modifiers available in C++ are:

- Signed
- Unsigned
- Short
- Long

Below table summarizes the modified size and range of built-in datatypes when combined with the type modifiers:

Data Type	Size (in bytes)	Range
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	8	0 to 4,294,967,295
long long int	8	-(2^63) to (2^63)-1

unsigned long long int	8	0 to 18,446,744,073,709,551,615
signed char	1	-128 to 127
unsigned char	1	0 to 255
float	4	
double	8	
long double	12	
wchar_t	2 or 4	1 wide character

Note: Above values may vary from compiler to compiler. In the above example, we have considered GCC 32 bit.

We can display the size of all the data types by using the sizeof() operator and passing the keyword of the datatype as argument to this function as shown below:

C++ program to sizes of data types

```
#include<iostream>
using namespace std;
int main()
{
    cout << "Size of char: " << sizeof(char)
      << " byte" << endl;
    cout << "Size of int: " << sizeof(int)
      << " bytes" << endl;
    cout << "Size of short int: " << sizeof(short int)
      << " bytes" << endl;
    cout << "Size of long int: " << sizeof(long int)
       << " bytes" << endl;
    cout << "Size of signed long int: " << sizeof(signed long int)
       << " bytes" << endl;
    cout << "Size of unsigned long int: " << sizeof(unsigned long int)
       << " bytes" << endl;
    cout << "Size of float: " << sizeof(float)
       << " bytes" <<endl;
    cout << "Size of double: " << sizeof(double)
       << " bytes" << endl;
    cout << "Size of wchar_t: " << sizeof(wchar_t)
       << " bytes" <<endl;
    return 0;
}
```

Output:

```
Size of char: 1 byte
Size of int: 4 bytes
Size of short int: 2 bytes
Size of long int: 8 bytes
Size of signed long int: 8 bytes
Size of unsigned long int: 8 bytes
Size of float: 4 bytes
Size of double: 8 bytes
Size of wchar_t: 4 bytes
```

Figure 3: Reserved keywords in C++

Code as described/written in the video

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

int main()
{
    int num1, num2;
    cout<<"Enter the value of num:\n"; /* '<<' is called Insertion operator */
    cin>>num1; /* '>>' is called Extraction operator */

    cout<<"Enter the value of num2:\n"; /* '<<' is called Insertion operator */
    cin>>num2; /* '>>' is called Extraction operator */

    cout<<"The sum is "<< num1+num2;

    return 0;
}
```

Part: 06

C++ Header files & Operators

In this C++ tutorial, we will talk about header files and operators. In our last lesson, we discussed the basic input and output. Let's now cover header files and operators in C++ language:

Header Files in C++

"**#include**" is used in C++ to import header files in our C++ program. The reason to introduce the "**<iostream>**" header file into our program is that functions like "**cout**" and "**cin**" are defined in that header file. There are two types of header files:

System Header Files

System header files ships with the compiler. For example, "**#include <iostream>**". To see the references for header files.

User-Defined Header Files

The programmer writes User-defined header files himself. To include your header file in the program, you first need to make a header file in the current directory, and then you can add it.

Operators in C++

Operators are used for producing output by performing various types of calculations on two or more inputs. In this lecture, we will see the operators in C++.

Arithmetic Operators

Arithmetic operators are used for performing mathematical operations like (+, -, *). The arithmetic operators are shown in Figure 1.

```
int a=4, b=5;
cout<<"Operators in C++:"<<endl;
cout<<"Following are the types of operators in C++"<<endl;
// Arithmetic operators
cout<<"The value of a + b is "<<a+b<<endl;
cout<<"The value of a - b is "<<a-b<<endl;
cout<<"The value of a * b is "<<a*b<<endl;
cout<<"The value of a / b is "<<a/b<<endl;
cout<<"The value of a % b is "<<a%b<<endl;
cout<<"The value of a++ is "<<a++<<endl;
cout<<"The value of a-- is "<<a--<<endl;
cout<<"The value of ++a is "<<++a<<endl;
cout<<"The value of --a is "<<--a<<endl;
cout<<endl;
```

Figure 1: Arithmetic Operators

1. The function "**a+b**", will add a and b values and print them.
2. The function "**a-b**", will subtract a and b values and print them.
3. The function "**a*b**" will multiply a and b values and print them.
4. The function "**a/b**", will divide a and b values and print them.
5. The function "**a%b**", will take the modulus of a and b values and print them.

6. The function "**a++**" will first print the value of a and then increment it by 1.
7. The function "**a--**", will first print the value of a and then decrement it by 1.
8. The function "**++a**", will first increment it by one and then print its value.
9. The function "**--a**", will first decrement it by one and then print its value.

The output of these arithmetic operators is shown in figure 2.

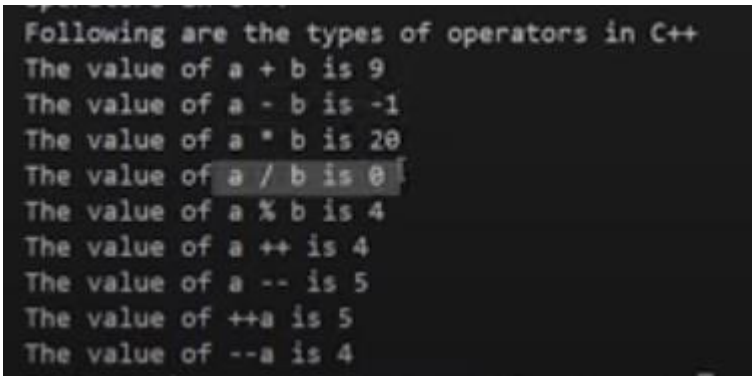


Figure 2: Arithmetic Operators Output

- **Assignment Operators**

Assignment operators are used for assigning values to variables. For example: **`int a = 10, b = 5;`**

- **Comparison Operators**

Comparison operators are used for comparing two values. Examples of comparison operators are shown in figure 3.

```
// Comparison operators
cout<<"Following are the comparison operators in C++"<<endl;
cout<<"The value of a == b is "<<(a==b)<<endl;
cout<<"The value of a != b is "<<(a!=b)<<endl;
cout<<"The value of a >= b is "<<(a>=b)<<endl;
cout<<"The value of a <= b is "<<(a<=b)<<endl;
cout<<"The value of a > b is "<<(a>b)<<endl;
cout<<"The value of a < b is "<<(a<b)<<endl;
```

Figure 3: Comparison Operators

1. The function "**(a==b)**", will compare a and b values and check if they are equal. The output will be one if equal, and 0 if not.
2. The function "**(a!=b)**", will compare a and b values and check if "**a**" is not equal to "**b**". The output will be one if not equal and 0 if equal.
3. The function "**(a>=b)**", will compare a and b values and check if "**a**" is greater than or equal to "**b**". The output will be one if greater or equal, and 0 if not.
4. The function "**(a<=b)**", will compare a and b values and check if "**b**" is less than or equal to "**a**". The output will be one if less or equal, and 0 if not.
5. The function "**(a>b)**", will compare a and b values and check if "**a**" is greater than "**b**". The output will be one if greater and 0 if not.
6. The function "**(a<b)**", will compare a and b values and check if "**b**" is less than "**a**". The output will be one if less and 0 if not.

The output of these comparison operators is shown in figure 4.

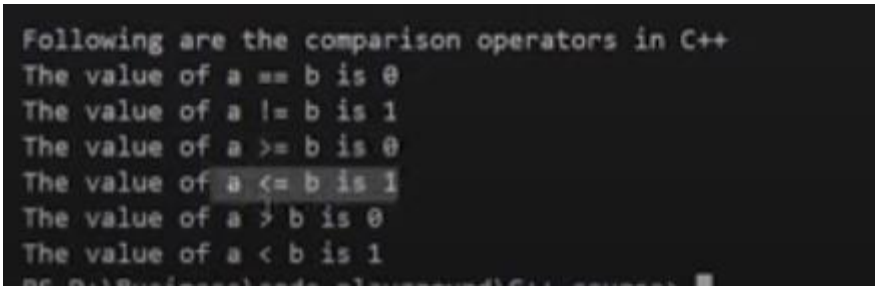


Figure 4: Comparison Operators Output

- Logical Operators

Logical operators are used for comparing two expressions. For example **`((a==b) && (a>b))`**. More examples of logical operators are shown in figure 5.

```
// Logical operators
cout<<"Following are the logical operators in C++"<<endl;
cout<<"The value of this logical and operator ((a==b) && (a<b)) is:"<<((a==b) && (a<b))<<endl;
cout<<"The value of this logical or operator ((a==b) || (a<b)) is:"<<((a==b) || (a<b))<<endl;
cout<<"The value of this logical not operator (!(a==b)) is:"<<(!(a==b))<<endl;
```

Figure 5: Logical Operators

1. The function "**`((a==b)&& (a<b))`**" will first compare a and b values and check if they are equal or not; if they are equal, the next expression will check whether "**`a`**" is smaller than "**`b`**". The output will be one if both expressions are true and 0 if not.
2. The function "**`((a==b) || (a<b))`**", will first compare a and b values and check if they are equal or not, even if they are not equal it will still check the next expression ie whether "**`a`**" is smaller than "**`b`**" or not. The output will be one if any one of the expressions is true and 0 if both are false.
3. The function "**`(!(a==b))`**", will first compare a and b values and check if they are equal or not. The output will be inversed ie if "**`a`**" and "**`b`**" are equal; the output will be 0 and 1 otherwise.

The output of these logical operators is shown in figure 6.

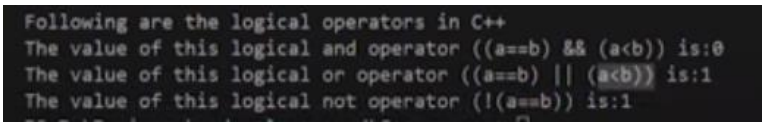


Figure 6: Logical Operators Output

That's it for this tutorial. In this lecture, we have covered some important operators in C++ language, but there are still some operators left, which we will cover in upcoming tutorials.

Code as described/written in the video

```
// There are two types of header files:
// 1. System header files: It comes with the compiler
#include<iostream>
// 2. User defined header files: It is written by the programmer
// #include "this.h" //--> This will produce an error if this.h is no present in the
current directory

using namespace std;

int main(){
    int a=4, b=5;
    cout<<"Operators in C++:"<<endl;
    cout<<"Following are the types of operators in C++"<<endl;
    // Arithmetic operators
```

```
    cout<<"The value of a + b is "<<a+b<<endl;
    cout<<"The value of a - b is "<<a-b<<endl;
    cout<<"The value of a * b is "<<a*b<<endl;
    cout<<"The value of a / b is "<<a/b<<endl;
    cout<<"The value of a % b is "<<a%b<<endl;
    cout<<"The value of a++ is "<<a++<<endl;
    cout<<"The value of a-- is "<<a--<<endl;
    cout<<"The value of ++a is "<<++a<<endl;
    cout<<"The value of --a is "<<--a<<endl;
    cout<<endl;

    // Assignment Operators --> used to assign values to variables
    // int a =3, b=9;
    // char d='d';

    // Comparison operators
    cout<<"Following are the comparison operators in C++"<<endl;
    cout<<"The value of a == b is "<<(a==b)<<endl;
    cout<<"The value of a != b is "<<(a!=b)<<endl;
    cout<<"The value of a >= b is "<<(a>=b)<<endl;
    cout<<"The value of a <= b is "<<(a<=b)<<endl;
    cout<<"The value of a > b is "<<(a>b)<<endl;
    cout<<"The value of a < b is "<<(a<b)<<endl;

    // Logical operators
    cout<<"Following are the logical operators in C++"<<endl;
    cout<<"The value of this logical and operator ((a==b) && (a<b)) is:"<<((a==b) &&
(a<b))<<endl;
    cout<<"The value of this logical or operator ((a==b) || (a<b)) is:"<<((a==b) ||
(a<b))<<endl;
    cout<<"The value of this logical not operator (!(a==b)) is:"<<(!(a==b))<<endl;


    return 0;
}
```

Part: 7

C++ Reference Variables & Typecasting

In this C++ tutorial, we will discuss the reference variables and typecasting. In our last lesson, we discussed the header files and operators in C++. These are the topics which we are going to cover in this tutorial:

- **Built-in Data Types**
- **Float, Double and Long Double Literals**
- **Reference Variables**
- **Typecasting**

Built-in Data Types

As discussed in our previous lectures, built-in data types are pre-defined by the language and can be used directly. An example program for built-in data types is shown in figure 1.

```
5  int c = 45;
6
7  int main(){
8      int a, b, c;
9      cout<<"Enter the value of a:"<<endl;
10     cin>>a;
11     cout<<"Enter the value of b:"<<endl;
12     cin>>b;
13     c = a + b;
14     cout<<"The sum is "<<c<<endl;
15     cout<<"The global c is "<<::c;
16
```

Figure 1: Built-in Data Types

The code of built-in data types can be seen in figure 1 where we have declared three variables "**a, b and c**" inside the main function and one variable "**c**" outside the main function which is a global variable. To access the value of the global variable "**c,**" we use **scope resolution operator** "**::**" with the "**c**" variable. The output of the following program is shown in figure 2.

```
Enter the value of a:
5
Enter the value of b:
9
The sum is 14
The global c is 45
```

Figure 2: Built-in Data Types Output

As we have entered the value of the variable "**a**" as five and "**b**" as 6, it gives us the sum 14, but for the global variable, it has given us the value 45.

Float, Double and Long Double Literals

The main reason to discuss these literals was to tell you an important concept about them. The float, double and long double literals program is shown in figure 3.

```
float d=34.4F;
long double e = 34.4L;
cout<<"The size of 34.4 is "<<sizeof(34.4)<<endl;
cout<<"The size of 34.4f is "<<sizeof(34.4f)<<endl;
cout<<"The size of 34.4F is "<<sizeof(34.4F)<<endl;
cout<<"The size of 34.4l is "<<sizeof(34.4l)<<endl;
cout<<"The size of 34.4L is "<<sizeof(34.4L)<<endl;
```

Figure 3: Float, Double & Long Double Literals

So the crucial concept which I am talking about is that in C++ language, double is the default type given to a decimal literal (34.4 is double by default and not float), so to use it as float, you have to specify it like "**34.4F,**" as shown in figure 3. To display the size of float, double, and long double literals, we have used a "**sizeof**" operator. The output of this program is shown in figure 4.

```
The size of 34.4 is 8
The size of 34.4f is 4
The size of 34.4F is 4
The size of 34.4l is 12
The size of 34.4L is 12
The value of d is 34.4
The value of e is 34.4
```

Figure 4: Float, Double, Long Double Literal Output

Reference Variable

Reference variables can be defined as another name for an already existing variable. These are also called an alias. For example, let us say we have a variable with the name of "**sum**", but we also want to use the same variable with the name of "**add**", to do that we will make a reference variable with the name of "**add**". The example code for the reference variable is shown in figure 5.

```
float x = 455;
float & y = x;
cout<<x<<endl;
cout<<y<<endl;
```

Figure 5: Reference Variable Code

As shown in figure 5, we initialized a variable "**x**" with the value "**455**". Then we assigned the value of "**x**" to a reference variable "**y**". The ampersand "**&**" symbol is used with the "**y**" variable to make it reference variable. Now the variable "**y**" will start referring to the value of the variable "**x**". The output for variable "**x**" and "**y**" is shown in figure 6.

```
PS D:\Business\code playground\C++ course> c
455
455
PS D:\Business\code playground\C++ course>
```

Figure 6: Reference Variable Code Output

Typecasting

Typecasting can be defined as converting one data type into another. Example code for type casting is shown in figure 7.

```
// *************Typecasting***************
int a = 45;
float b = 45.46;
cout<<"The value of a is "<<(float)a<<endl;
cout<<"The value of a is "<<float(a)<<endl;

cout<<"The value of b is "<<(int)b<<endl;
cout<<"The value of b is "<<int(b)<<endl;
int c = int(b);

cout<<"The expression is "<<a + b<<endl;
cout<<"The expression is "<<a + int(b)<<endl;
cout<<"The expression is "<<a + (int)b<<endl;
```

Figure 7: Typecasting Example Code

As shown in figure 7, we have initialized two variables, integer "**a**" and float "**b**". After that, we converted an integer variable "**a**" into a float variable and float variable "**b**" into an integer variable. In C++, there are two ways to typecast a variable, either using "**(float)a**" or using "**float(a)**". The output for the above program is shown in figure 8.

```
PS D:\Business\code playground\C++ course>
The value of a is 45
The value of a is 45
The value of b is 45
The value of b is 45
The expression is 90.46
The expression is 90
The expression is 90
```

Figure 8: Typecasting Program Output

Code as described/written in the video

```
#include<iostream>

using namespace std;

int c = 45;

int main(){

    // *************Build in Data types*****************
    // int a, b, c;
    // cout<<"Enter the value of a:"<<endl;
    // cin>>a;
    // cout<<"Enter the value of b:"<<endl;
    // cin>>b;
    // c = a + b;
    // cout<<"The sum is "<<c<<endl;
    // cout<<"The global c is "<<::c;

    // ************* Float, double and long double Literals****************
    // float d=34.4F;
    // long double e = 34.4L;
    // cout<<"The size of 34.4 is "<<sizeof(34.4)<<endl;
```

```
    // cout<<"The size of 34.4f is "<<sizeof(34.4f)<<endl;
    // cout<<"The size of 34.4F is "<<sizeof(34.4F)<<endl;
    // cout<<"The size of 34.4l is "<<sizeof(34.4l)<<endl;
    // cout<<"The size of 34.4L is "<<sizeof(34.4L)<<endl;
    // cout<<"The value of d is "<<d<<endl<<"The value of e is "<<e;

    // *************Reference Variables****************
    // Rohan Das----> Monty -----> Rohu ------> Dangerous Coder
    // float x = 455;
    // float & y = x;
    // cout<<x<<endl;
    // cout<<y<<endl;

    // *************Typecasting***************
    int a = 45;
    float b = 45.46;
    cout<<"The value of a is "<<(float)a<<endl;
    cout<<"The value of a is "<<float(a)<<endl;

    cout<<"The value of b is "<<(int)b<<endl;
    cout<<"The value of b is "<<int(b)<<endl;
    int c = int(b);

    cout<<"The expression is "<<a + b<<endl;
    cout<<"The expression is "<<a + int(b)<<endl;
    cout<<"The expression is "<<a + (int)b<<endl;


    return 0;
}
```

Part: 8

Constants, Manipulators & Operator Precedence

In this series of our C++ tutorials, we will visualize the constants, manipulator, and operator precedence in C++ language in this lecture. In our last lesson, we discussed the reference variable and typecasting in C++. If you haven't read the previous tutorial.

In this C++ tutorial, the topics which we are going to cover today are given below:

- **Constants in C++**
- **Manipulator in C++**
- **Operator Precedence in C++**

Constants in C++

Constants are unchangeable; when a constant variable is initialized in a program, its value cannot be changed afterwards. An example program for constants is shown in figure 1.

```
9     // Constants in C++
10    const float a = 3.11;
11    cout<<"The value of a was: "<<a<<endl;
12    a = 45.6;
13    cout<<"The value of a is: "<<a<<endl;
14    return 0;
15 }
```

Figure 1: Constants in C++

As shown in figure 2, a constant float variable **"a"** is initialized with a value **"3.11"** but when we tried to update the value of **"a"** with a value of **"45.6"** the compiler throw us an error that the constant variable is being reassigned a value. An error message can be seen in figure 2.

```
tut8.cpp: In function 'int main()':
tut8.cpp:12:9: error: assignment of read-only variable 'a'
     a = 45.6;
         ^~~~
```

Figure 2: Constant Program Error

Manipulator

In C++ programming, language manipulators are used in the formatting of output. The two most commonly used manipulators are: **"endl"** and **"setw"**.

- **"endl"** is used for the next line.
- **"setw"** is used to specify the width of the output.

An example program to show the working of a manipulator is shown in figure 3.

```
int a =3, b=78, c=1233;
cout<<"The value of a without setw is: "<<a<<endl;
cout<<"The value of b without setw is: "<<b<<endl;
cout<<"The value of c without setw is: "<<c<<endl;

cout<<"The value of a is: "<<setw(4)<<a<<endl;
cout<<"The value of b is: "<<setw(4)<<b<<endl;
cout<<"The value of c is: "<<setw(4)<<c<<endl;
return 0;
```

Figure 3: Manipulators in C++

As shown in figure 3, we have initialized three integer variables "**a, b, c**". First, we printed all the three variables and used "**endl**" to print each variable in a new line. After that, we again printed the three variables and used "**setw(4),**" which will set there width to "**4**". The output for the following program is shown in figure 4.

```
The value of a without setw is: 3
The value of b without setw is: 78
The value of c without setw is: 1233
The value of a is:    3
The value of b is:   78
The value of c is: 1233
```

Figure 4: Manipulators Program Output

Operator Precedence & Operator Associativity

Operator precedence helps us to solve an expression. For example, in an expression "**int c = a*b+c**" the multiplication operator's precedence is higher than the precedence of addition operator, so the multiplication between "**a & b**" first and then addition will be performed.

Operator associativity helps us to solve an expression; when two or more operators have the same precedence, the operator associativity helps us to decide that we should solve the expression from "**left-to-right**" or from "**right-to-left**".

Operator precedence and operator associativity can be seen from here. An example program for operator precedence and operator associativity is shown in figure 5.

```
// Operator Precedence
int a =3, b=4;
// int c = (a*5)+b;
int c = ((((a*5)+b)-45)+87);
cout<<c;
return 0;
```

Figure 5: Operator Precedence & Associativity Example program

As shown in figure 5, we initialized two integer variables and then wrote an expression "**int c = a*5+b;**" on which we have already discussed. Then we have written another expression "**int c = ((((a*5)+b)-45)+87);**". The precedence of multiply is higher than addition so the multiplication will be done first, but

the precedence of addition and subtraction is same, so here we will check the associativity which is "**left-to-right**" so the addition is performed first and then subtraction is performed.

Code as described/written in the video

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

using namespace std;

int main(){
    // int a = 34;
    // cout<<"The value of a was: "<<a;
    // a = 45;
    // cout<<"The value of a is: "<<a;
    // Constants in C++
    // const int a = 3;
    // cout<<"The value of a was: "<<a<<endl;
    // a = 45; // You will get an error because a is a constant
    // cout<<"The value of a is: "<<a<<endl;

    // Manipulators in C++
    // int a =3, b=78, c=1233;
    // cout<<"The value of a without setw is: "<<a<<endl;
    // cout<<"The value of b without setw is: "<<b<<endl;
    // cout<<"The value of c without setw is: "<<c<<endl;

    // cout<<"The value of a is: "<<setw(4)<<a<<endl;
    // cout<<"The value of b is: "<<setw(4)<<b<<endl;
    // cout<<"The value of c is: "<<setw(4)<<c<<endl;


    // Operator Precedence
    int a =3, b=4;
    // int c = (a*5)+b;
    int c = ((((a*5)+b)-45)+87);
    cout<<c;
    return 0;
}
```

Part: 09

C++ Control Structures, If Else and Switch-Case Statement

In this series of our C++ tutorials, we will visualize the control structure, if-else, and switch statements in the C++ language in this lecture. In our last lesson, we discussed the constant, manipulators and operator precedence in C++.

In this C++ tutorial, the topics which we are going to cover today are given below:

- **Control Structures in C++**
- **IF Else in C++**
- **Switch Statement in C++**

Control Structures in C++

The work of control structures is to give flow and logic to a program. There are three types of basic control structures in C++.

1. **Sequence Structure**

Sequence structure refers to the sequence in which program execute instructions one after another. An example diagram for the sequence structure is shown in figure 1.

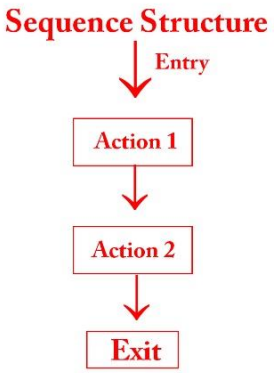


Figure 1: Sequence Structure

2. **Selection Structure**

Selection structure refers to the execution of instruction according to the selected condition, which can be either true or false. There are two ways to implement selection structures, by **"`if-else statements`"** or by **"`switch case statements`"**. An example diagram for selection structure is shown in figure 2.

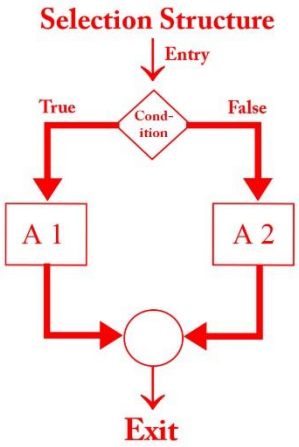


Figure 2: Selection Structure

3. Loop Structure

Loop structure refers to the execution of an instruction in a loop until the condition gets false. An example diagram for loop structure is shown in figure 3.

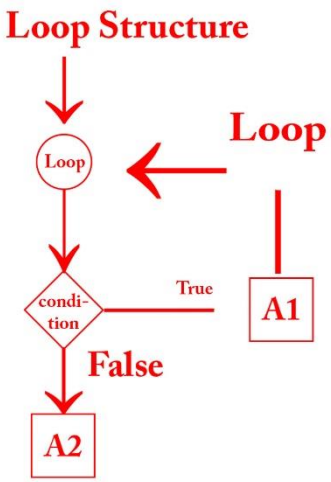


Figure 3: Loop Structure

If Else Statements in C++

As we have discussed the concepts of the different control structure, If else statements are used to implement a selection structure. An example program for if-else is shown in figure 4.

```
7   int age;
8   cout<< "Tell me your age"<<endl;
9   cin>>age;
10  if(age<18){
11      cout<<"You can not come to my party"<<endl;
12  }
13  else if(age==18){
14      cout<<"You are a kid and you will get a kid pass to the party"<<endl;
15  }
16  else{
17      cout<<"You can come to the party"<<endl;
18  }
```

Figure 4: If-Else program in C++

As shown in figure 4, we declared a variable **“age”** and used **“cin"** function to gets its value from the user at run time. At line 10 we have used **"if”** statement and give a condition **“(age<18)”** which means that if the age entered by the user is smaller than **"18”** the output will be **“you cannot come to my party”** but if the age is not smaller than **“18”** the compiler will move to the next condition.

At line 13 we have used **“else if”** statement and given another condition **“age==18"** which means that if the age entered by the user is equal to **"18”** the output will be **“you are a kid and you will get a kid pass to the party”** but if the age is not equal to the **“18”** the compiler will move to the next condition.

At line 16 we have used **“else"** condition which means that if none of the above condition is **"true"** the output will be **"you can come to the party”**.

The output for the following program is shown in figure 5.

```
PS D:\Business\code playgrounc
Tell me your age
81
You can come to the party
```

Figure 5: If-Else Program Output

As can be seen in figure 5, that when we entered the age "**81**" which was greater than 18, so it gives us the output "**you can come to the party**". The main thing to note here is that we can use as many "**else if**" statements as we want.

Switch Case Statements in C++

In switch-case statements, the value of the variable is tested with all the cases. An example program for the switch case statement is shown in figure 6.

```
26    switch (age)
27    {
28    case 18:
29        cout<<"You are 18"<<endl;
30        break;
31    case 22:
32        cout<<"You are 22"<<endl;
33        break;
34    case 2:
35        cout<<"You are 2"<<endl;
36        break;
37
38    default:
39    cout<<"No special cases"<<endl;
40        break;
41    }
42
43    cout<<"Done with switch case";
```

Figure 6: Switch Case Statement Program

As shown in figure 4, we passed a variable "**age**" to the switch statement. The switch statement will compare the value of variable "**age**" with all cases. For example, if the entered value for variable "**age**" is "**18**", the case with value "**18**" will be executed and prints "**you are 18**". The keyword "**break**" will let the compiler skips all other cases and goes out of the switch case statement. An output of the following program is shown in figure 6.

```
Tell me your age
18
You are 18
Done with switch case
```

Figure 7: Switch Case Statement Program Output

As shown in figure 7, we entered the value "**18**" for the variable "**age**", and it gives us an output "**you are 18**" and "**Done with switch case**". The main thing to note here is that after running the "**case 18**" is skips all the other cases due to the "**break**" statement and printed "**Done with switch case**" which was outside of the switch case statement.

Code as described/written in the video

```
#include<iostream>

using namespace std;

int main(){
    // cout<<"This is tutorial 9";
    int age;
    cout<< "Tell me your age"<<endl;
    cin>>age;

    // 1. Selection control structure: If else-if else ladder
    // if((age<18) && (age>0)){
    //     cout<<"You can not come to my party"<<endl;
    // }
    // else if(age==18){
    //     cout<<"You are a kid and you will get a kid pass to the party"<<endl;
    // }
    // else if(age<1){
    //     cout<<"You are not yet born"<<endl;
    // }
    // else{
    //     cout<<"You can come to the party"<<endl;
    // }

    // 2. Selection control structure: Switch Case statements
    switch (age)
    {
    case 18:
        cout<<"You are 18"<<endl;
        break;
    case 22:
        cout<<"You are 22"<<endl;
        break;
    case 2:
        cout<<"You are 2"<<endl;
        break;

    default:
    cout<<"No special cases"<<endl;
        break;
    }

    cout<<"Done with switch case";
    return 0;
}
```

Part: 10

For, While and do-while loops in C++

For, While and Do-While Loops in C++

In this series of our C++ tutorials, we will visualize for loop, while loop, and do-while loop in C++ language in this lecture. In our last lesson, we discussed the control structures, If-else statements, and switch statements in C++.

Loops in C++

Loops are block statements, which keeps on repeatedly executing until a specified condition is met. There are three types of loops in C++

- **For loop in C++**
- **While loop in C++**
- **Do While in C++**

For Loop in C++

For loop help us to run some specific code repeatedly until the specified condition is met. An example program **for the loop** is shown in figure 1.

```
for (int i = 0; i < 4; i++)
{
    /* code */
    cout<<i<<endl;
}
```

Figure 1: For Loop Program

As shown in figure 1, we created **for loop,** and inside its condition, there are three statements separated by a semicolon. The 1st statement is called **"`initialization`"**, the 2nd statement is called **"`condition`"**, and the 3rd statement is called **"`updation`"**. After that, there is a loop body in which code is written, which needs to be repeated. Here is how our for loop will be executed:

- Initialize integer variable **"`i`"** with value **"`0`"**
- Check the condition if the value of the variable **"`i`"** is smaller than **"`4`"**
- If the condition is true go into loop body and execute the code
- Update the value of **"`i`"** by one
- Keep repeating this step until the condition gets false

The output for the following program is shown in figure 2.

```
0
1
2
3
```

Figure 2: For Loop Program Output

While Loop in C++

While loop helps us to run some specific code repeatedly until the specified condition is met. An example program of **while loop** is shown in figure 3.

Figure 3: While Loop Program

As shown in figure 3, we created **a while loop,** and inside its condition, there is one statement. The statement is called "**condition**". Here is how our while loop will be executed:

- Initialize integer variable **"i"** with value **"1"**
- Check the condition if the value of the variable **"i"** is smaller or equal to **"40."**
- If the condition is true to go into loop body and execute the code
- Update the value of **"i"** by one
- Keep repeating this step until the condition gets false.

Do-While Loop in C++

The do-while loop helps us to run some specific code repeatedly until a specified condition is met. An example program of the **do-while loop** is shown in figure 1.

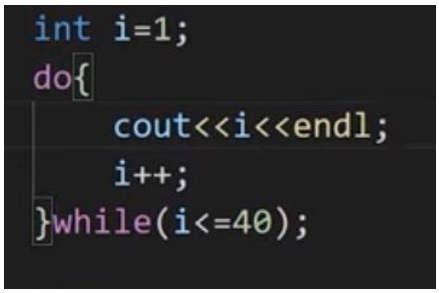


Figure 4: Do-While Loop Program

As shown in figure 4, we created a **do-while loop,** and the syntax of the do-while loop is like write body with **"do"** keyword and at the end of body write **"while"** keyword with the condition. Here is how our do-while loop will be executed:

- Initialize integer variable **"i"** with value **"1"**
- Go into loop body and execute the code
- Check the condition if the value of the variable **"i"** is smaller or equal to **"40"**
- If the condition is true - go into loop body and execute the code
- Keep repeating this step until the condition gets false

Code as described/written in the video

```
#include <iostream>

using namespace std;
int main()
{
    /*Loops in C++:
    There are three types of loops
in C++:
        1. For loop
        2. While Loop
        3. do-While Loop
    */

    /*For loop in C++*/
    // int i=1;
    // cout<<i;
    // i++;

    // Syntax for for loop
    // for(initialization;
condition; updation)
    // {
    //     loop body(C++ code);
    // }

    // for (int i = 1; i <= 40;
i++)
    // {
    //     /* code */
    //     cout<<i<<endl;
    // }

    // Example of infinite for loop
    // for (int i = 1; 34 <= 40;
i++)
    // {
    //     /* code */
    //     cout<<i<<endl;
    // }

    /*While loop in C++*/
    // Syntax:
    // while(condition)
    // {
    //     C++ statements;
    // }

    //  Printing 1 to 40 using
while loop
    // int i=1;
    // while(i<=40){
    //     cout<<i<<endl;
    //     i++;
    // }

    // Example of infinite while
loop
    // int i = 1;
    // while (true)
    // {
    //     cout << i << endl;
    //     i++;
    // }

    /* do While loop in C++*/
        // Syntax:
        // do
        // {
        //     C++ statements;
        // }while(condition);

    //  Printing 1 to 40 using
while loop
    // int i=1;
    // do{
    //     cout<<i<<endl;
    //     i++;
    // }while(false);

    return 0;
}
```