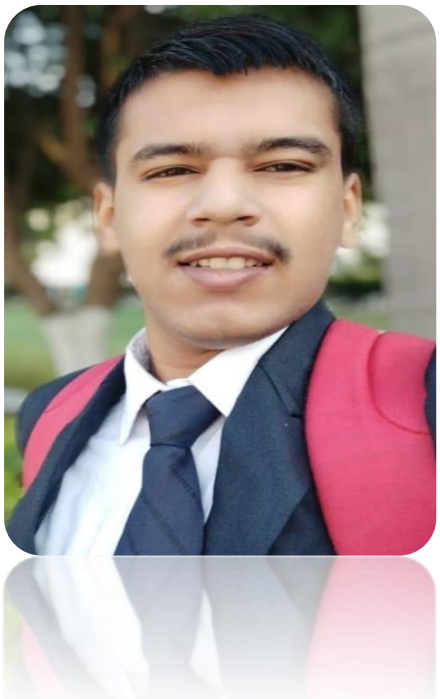
2024

C++

By: Jayant

Why Choose This book?

- Good Quality Content
- Easy to Understand
- Images and colors are used for increase performance
- highlight important points for students



Index

What is C++

(i)C++ is developed by Bjarne Stroustrup in 1979 at AT & Bell laboratory in California America.

(ii)C++ is an object-oriented programming language or it's a high-level language.

(iii)It is used for developing operating systems, games or etc.

(iv)It is basically advance version of C language because C doesn't support OOPs concept but C++ is supported OOPs concept.

(v)C++ is best language for doing DSA (Data Structure and algorithms).

Features of C++

(i)Simple to use: C++ gives easy syntax to developers for better understanding of code and easy to write.

(ii)OOPs: C++ support object-oriented programming because it gives security to protect our data.

(iii)High Level Language: C++ is a high-level language because developers easily write source code in C++ or understand it, then after C++ converted source code to machine code.

(iv)Compiler based: C++ follows by compilation process; it means compiler check whole source code; if source code is completely correct then give output otherwise give error according to mistake in source code.

Why should choose only C++?

(i)There are various programming languages or different type of languages are present in market but C++ has given high speed and performance.

(ii)It is directly access to hardware.

Difference between C and C++

C	C++
1.C is a procedural language.	1.C++ is an Object-oriented programming language.
2.Dennis Ritchie was developed by C language in 1972	2.While C++ was developed by Bjarne Stroustrup in 1979.
3.C doesn't support OOP's concept	3.C++ support OOP's concept
4.C support main function as void function	4.But C++ doesn't support void ad main function
5.In C language, use format specifier.	5.In C++, cannot use format specifier.
6.C doesn't support access specifier.	6.C++ support access specifier for data security.
7.C is less secure programming language as compared to C++.	7.C++ more secure programming language as compared to C.
8.C is nearest to hardware.	8.C++ also nearest to hardware.
9.<stdio.h> is used for basic input/output functions.	9.<iostream> is used for basic input output functions.
10.C doesn't support using namespace std.	10.C++ support using namespace std.
11.For memory allocation and deallocation use malloc, calloc, re-alloc or free.	11.For memory allocation and deallocation use new and delete operators
12.No supported Exception Handling	12.Supported Exception Handling.
13.(.C) extension is used for creating a C file.	13.(.cpp) extension is used for creating C++ file.
14.DevC++ is best ide for C language.	14.DevC++ is also best ide for C++ language.
15.Applications: Creating Operating systems, developing new languages, etc.	15.Game Development, GUI support, Compilers, etc.

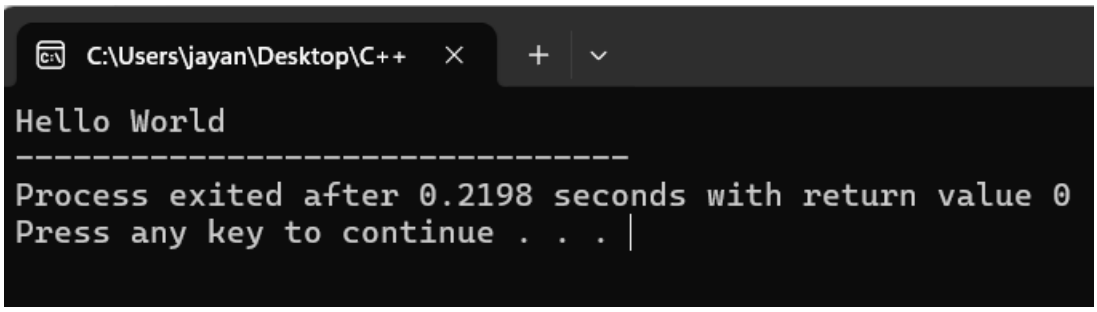
Why use (using namespace std)

(i)In C++, using namespace std is a directive that brings all the identifiers from the std namespace into the current scope.

(ii)This means you can use standard library functions, classes, and objects like cout, cin, and strings without having to prefix them with std::.

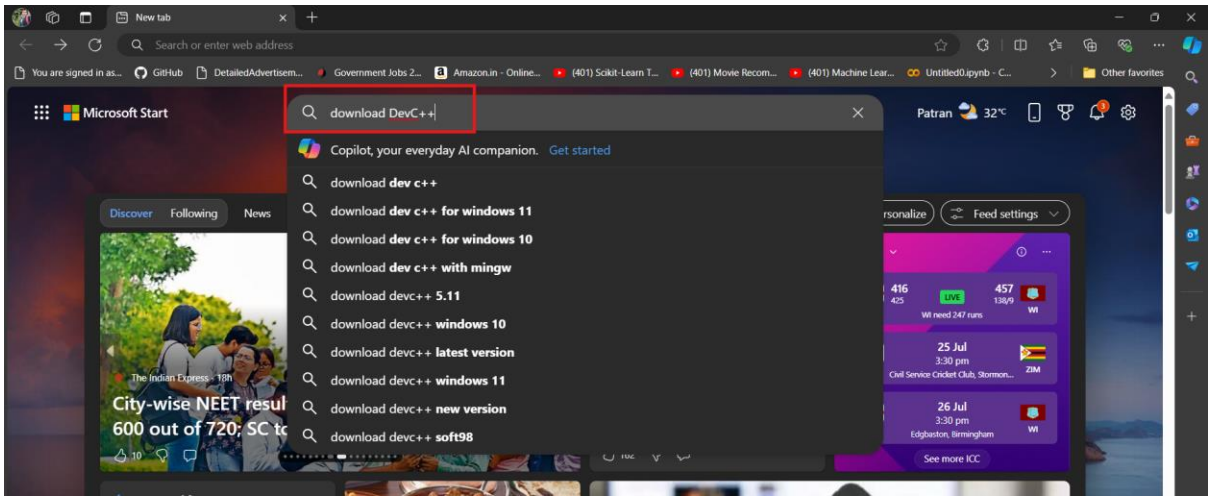
```
9
10  #include<iostream>
11  using namespace std;
12  main(){
13  cout<<"Hello World";
14  }
15
```

Output:

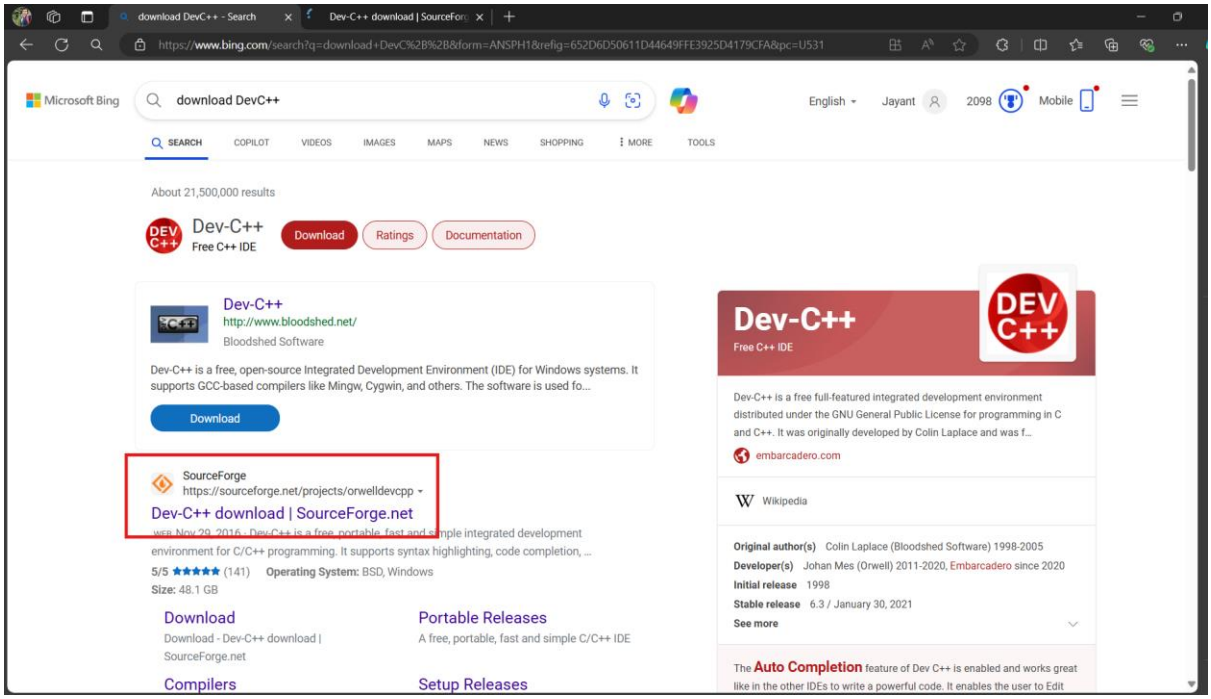


Setup C++ in VS Code

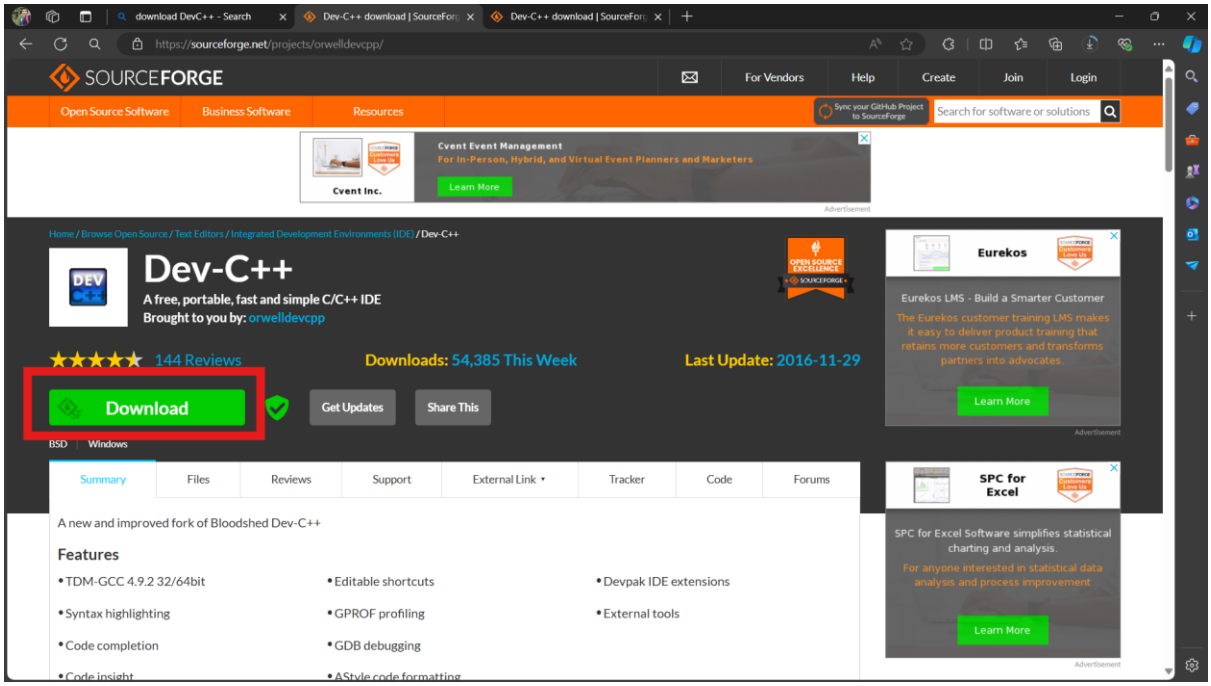
Step1: Open any browser or search download DevC++.

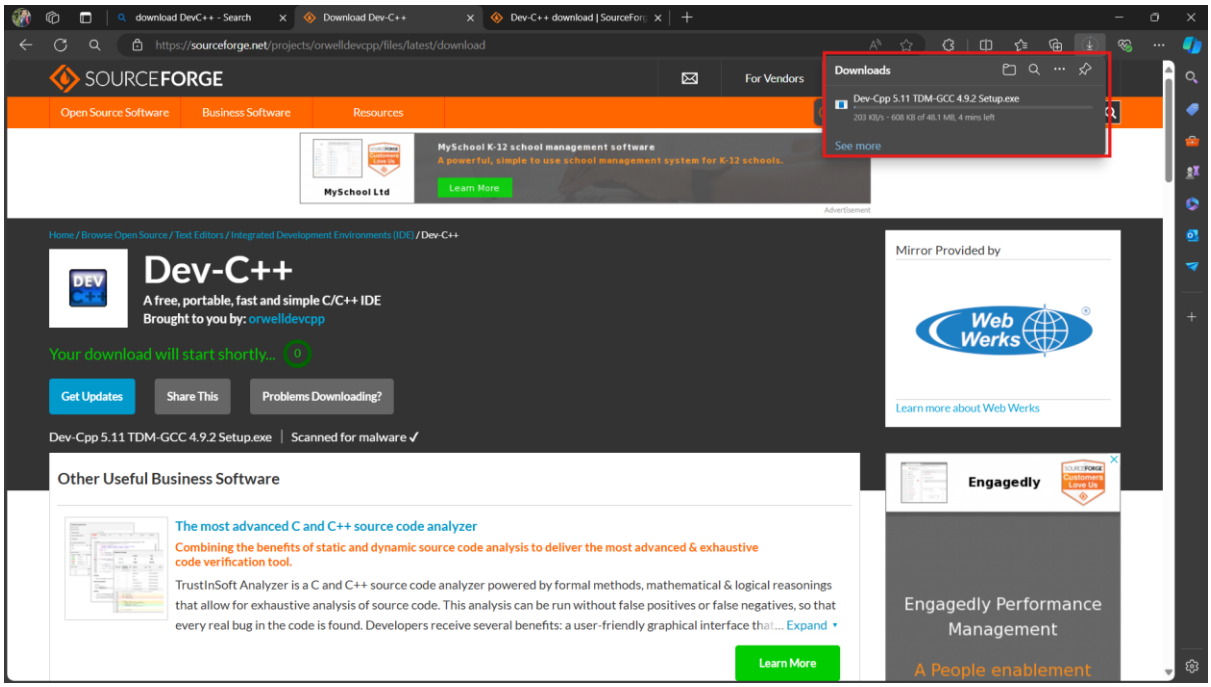


Step2: Click on Source Forge Website.

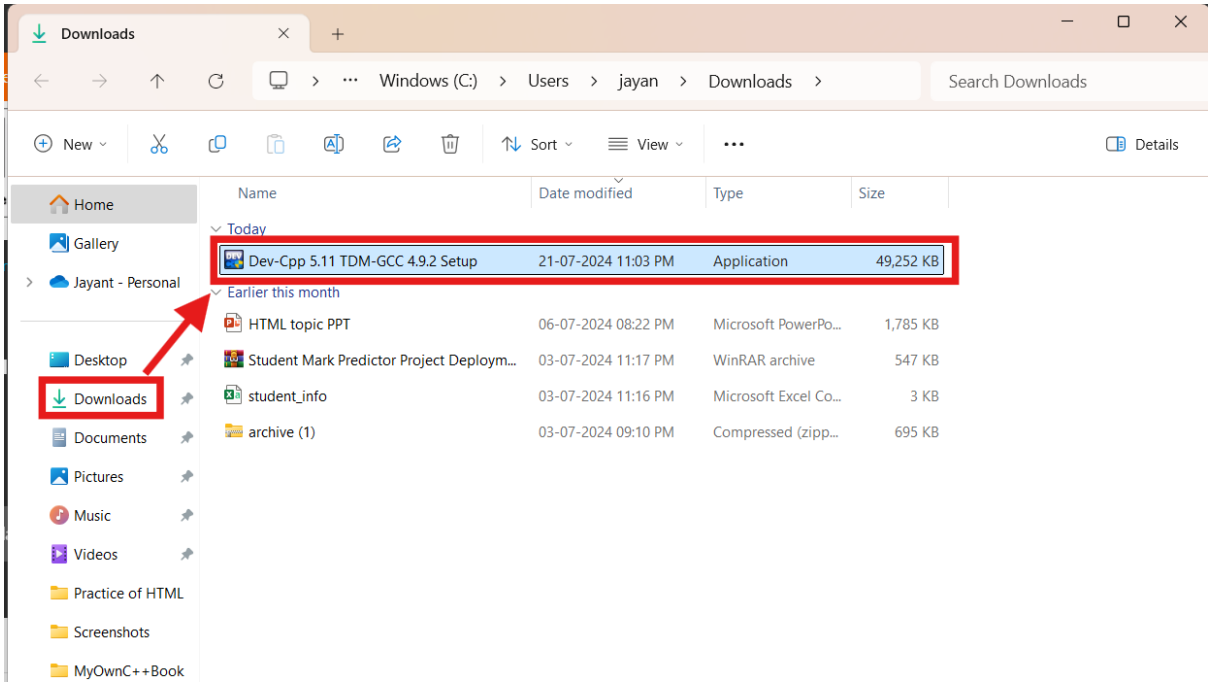


Step3: Click Download Button.



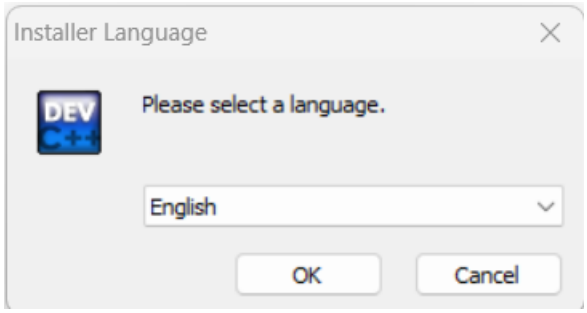


Step4: After Download, go to folder where you have downloaded DevC++ setup file or download file, then double click it.

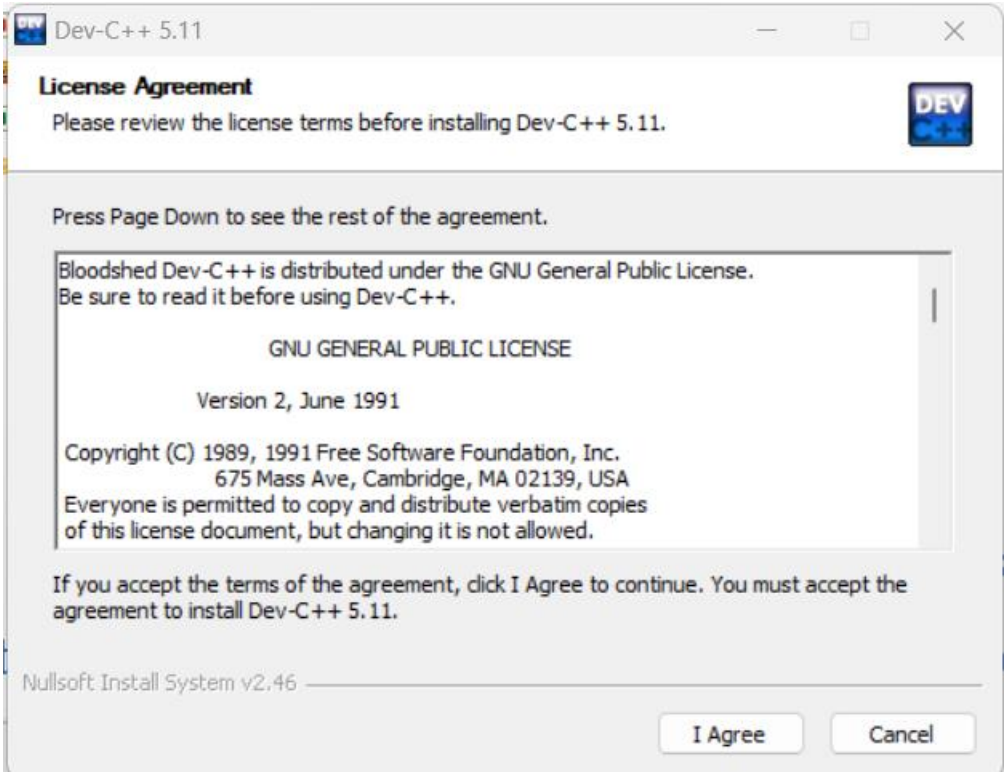


Step5: After double click, installation process will be start

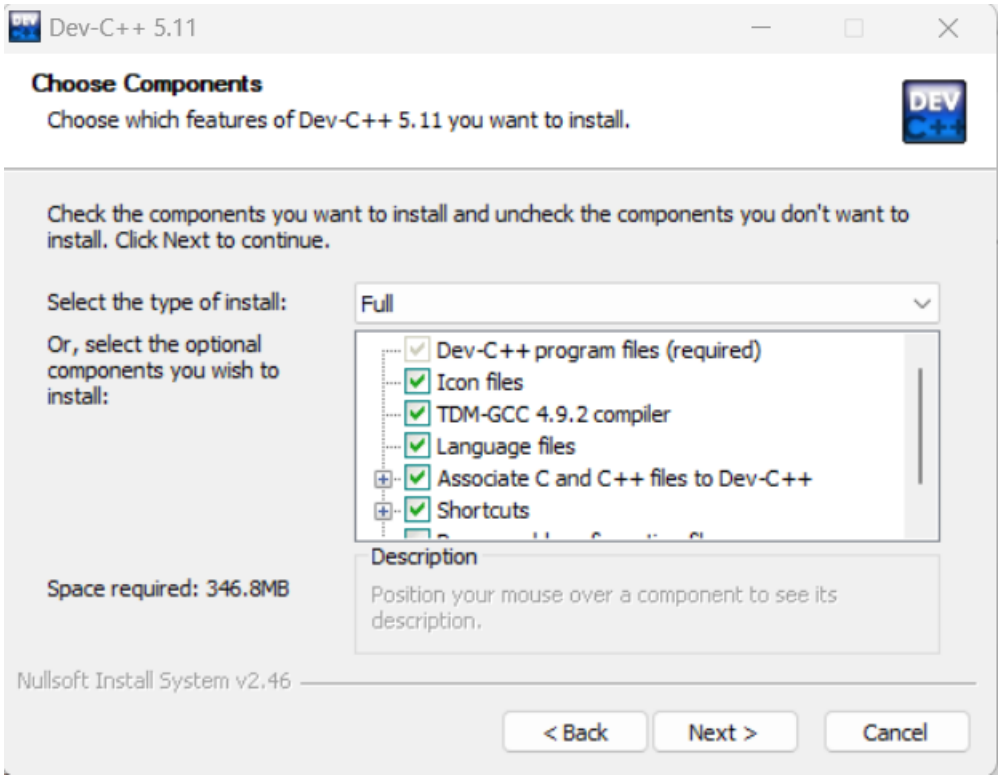
Step6: Select language, then click ok (By default selected english)



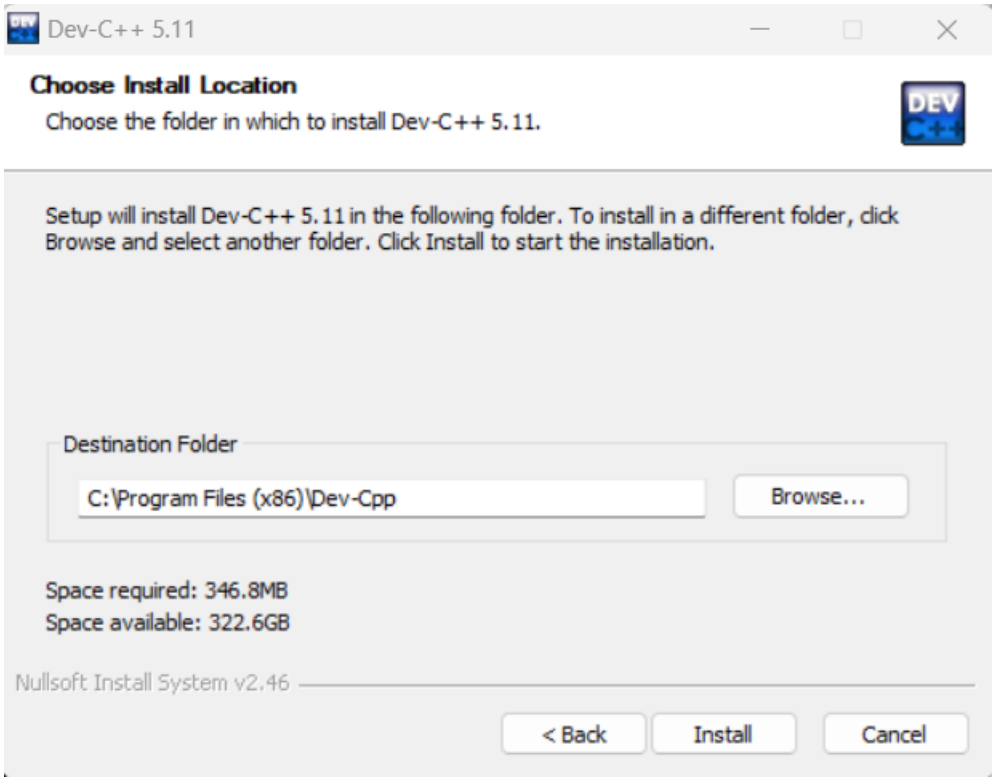
Step7: Read Agreement and click I Agree



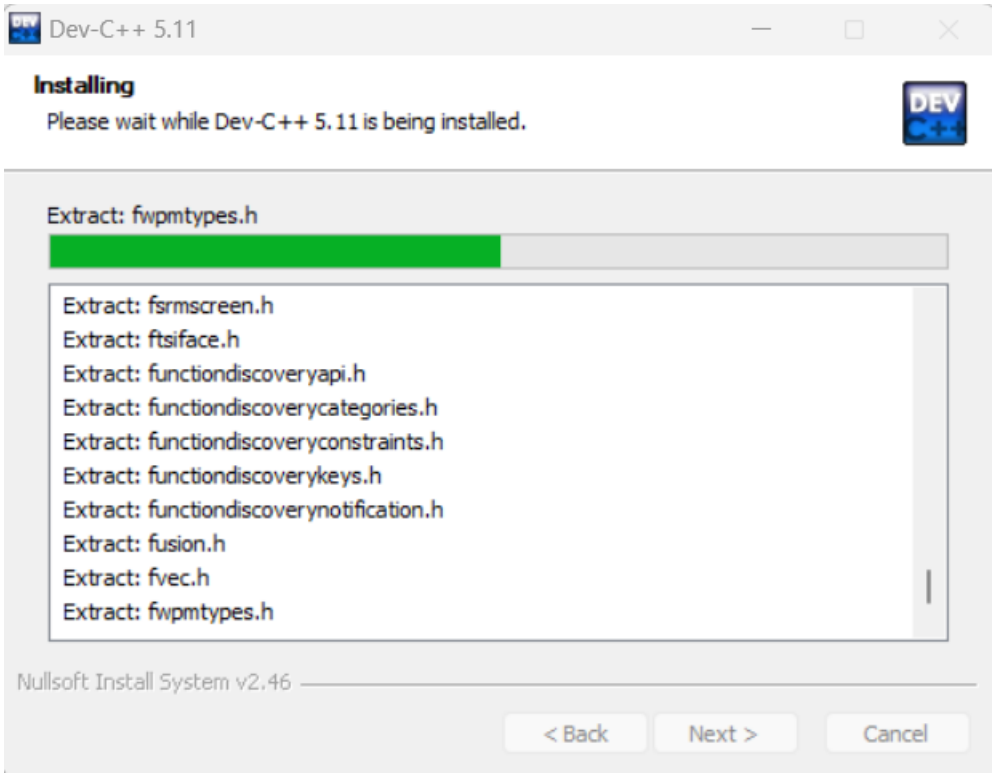
Step8: Choose Components; No need to change; Simply click on next



Step9: Select location where you want to store files of DevC++; if you want to use by default location, then click to install button

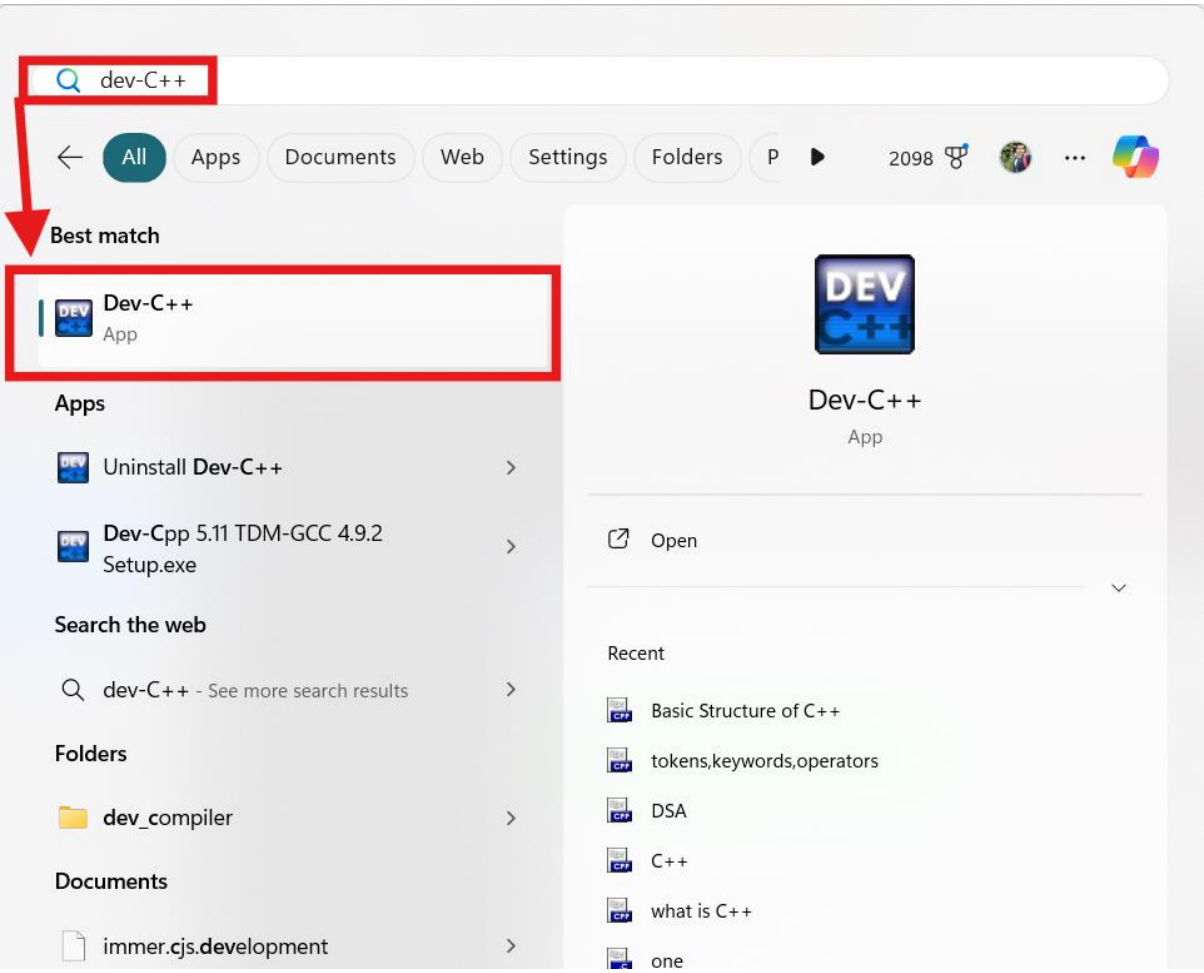


Step10: Wait until installation complete

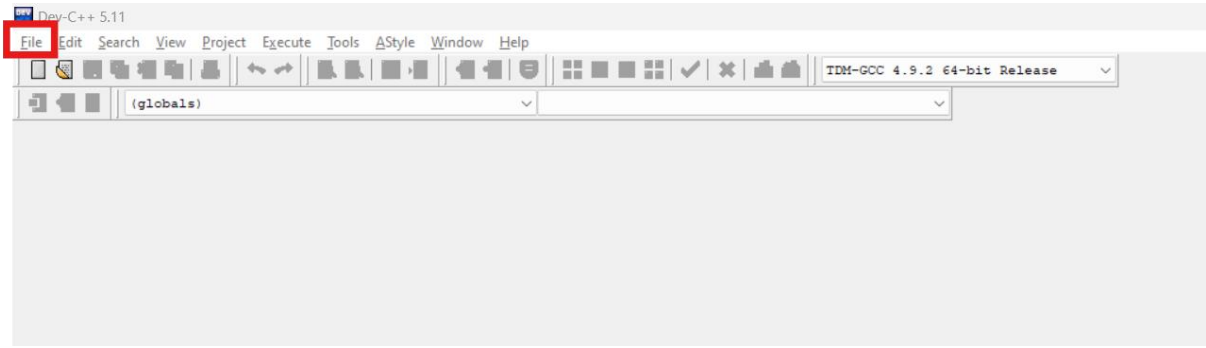


Step11: If you want to run DevC++ automatically at first time, then click on checkbox, then click finish button after installation process

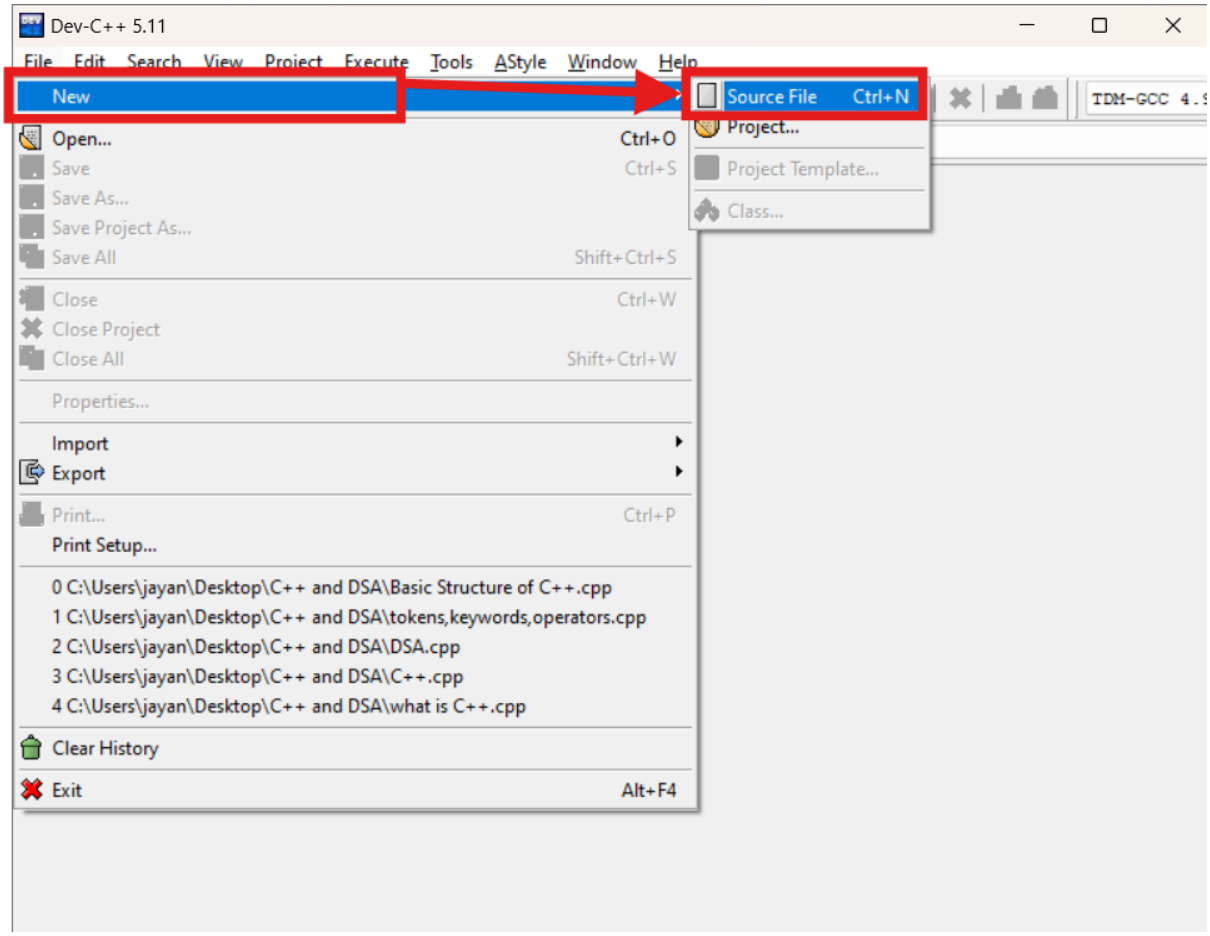
Step12: Search in windows devC++ and click it



Step13: After opening DevC++, click on file tab

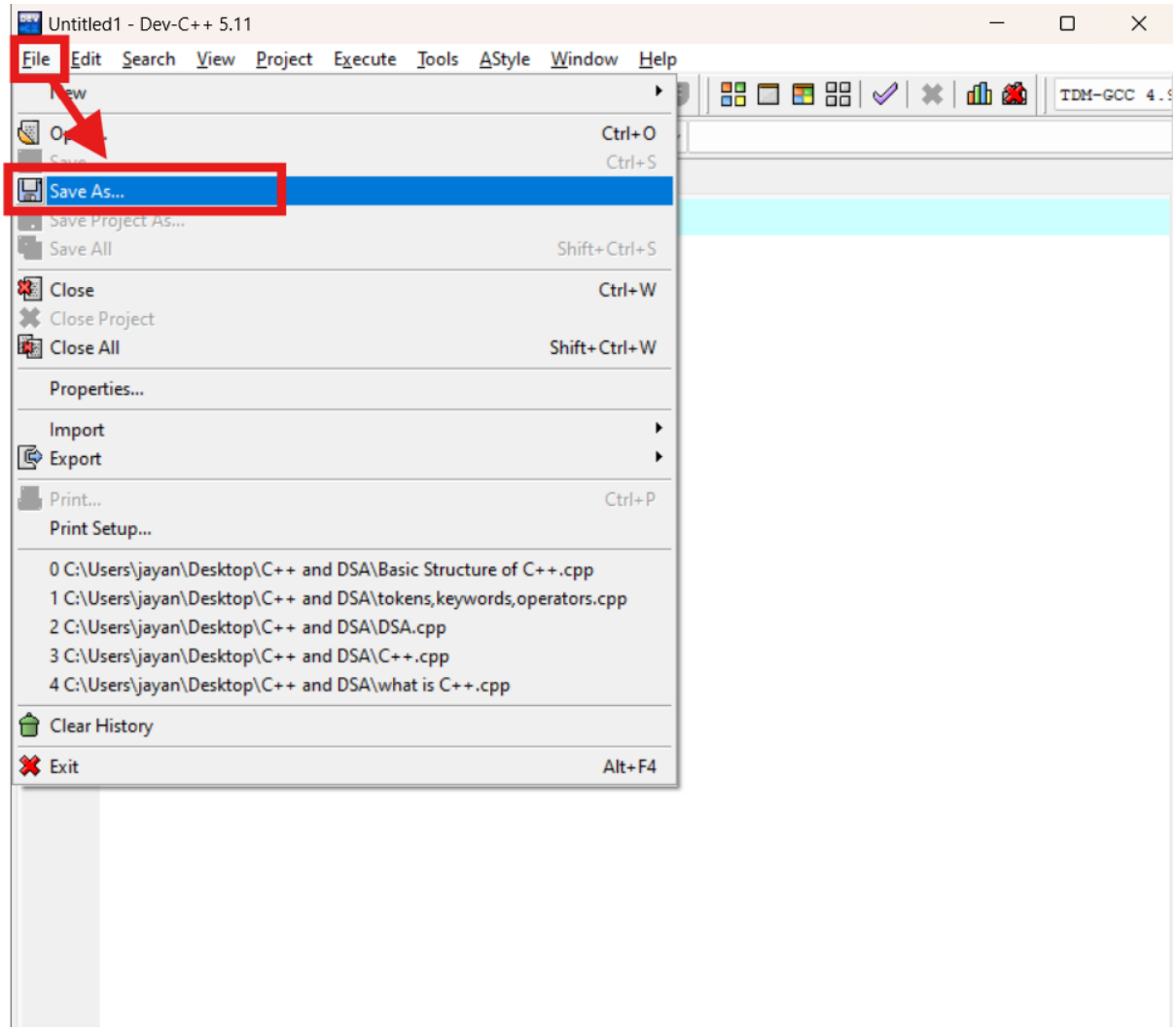


Step14: In file tab, click new option then click source file

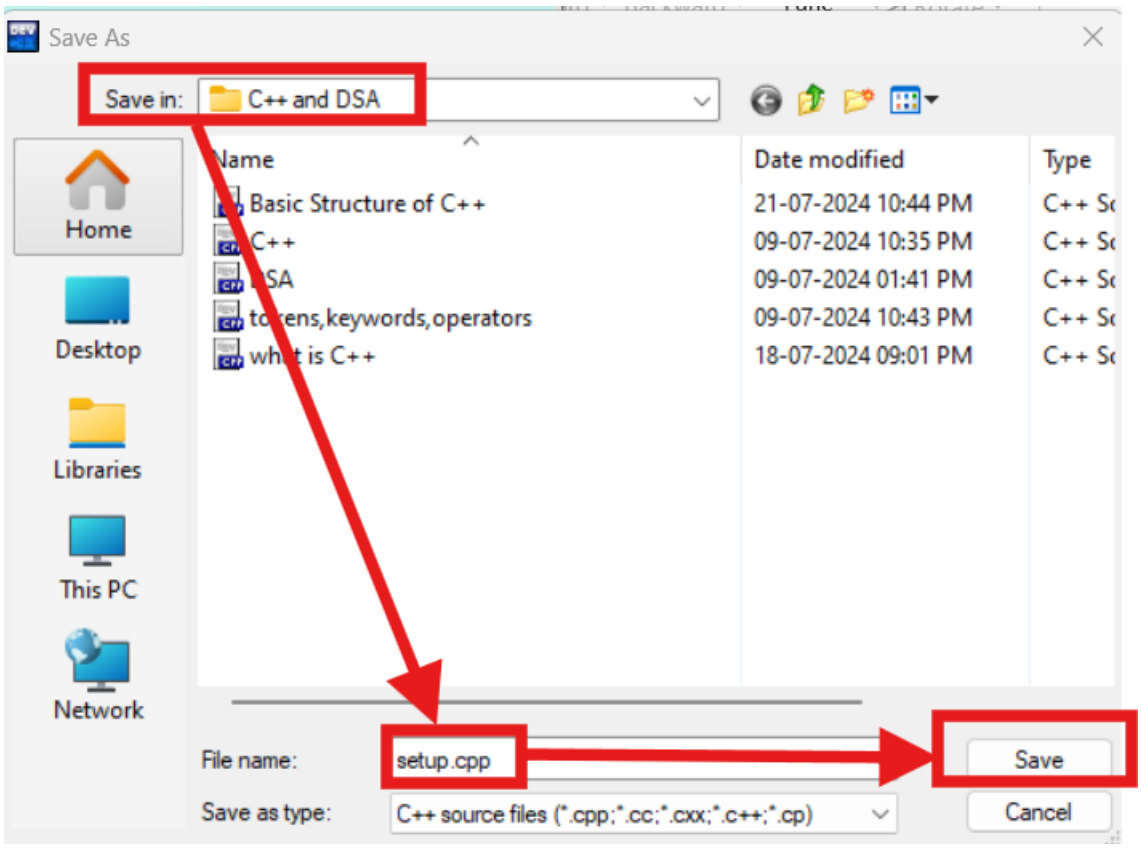


Step15: After click source file option, new file will be generated

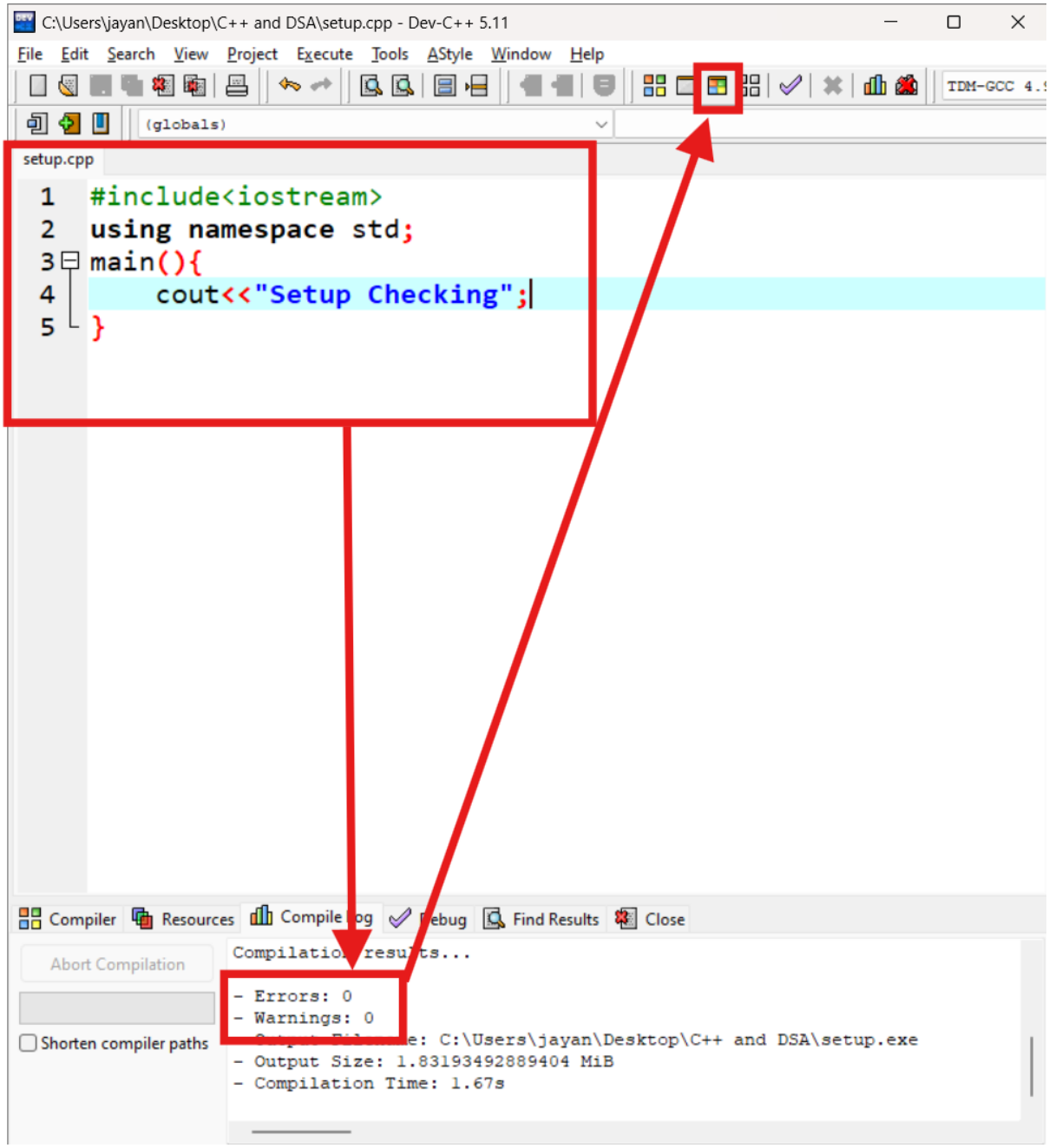
Step16: Again go to file tab and click SaveAs option for first time save the file



Step17: Select location where you want to save the file and give file name(anyname)with (.cpp) extension and click save.



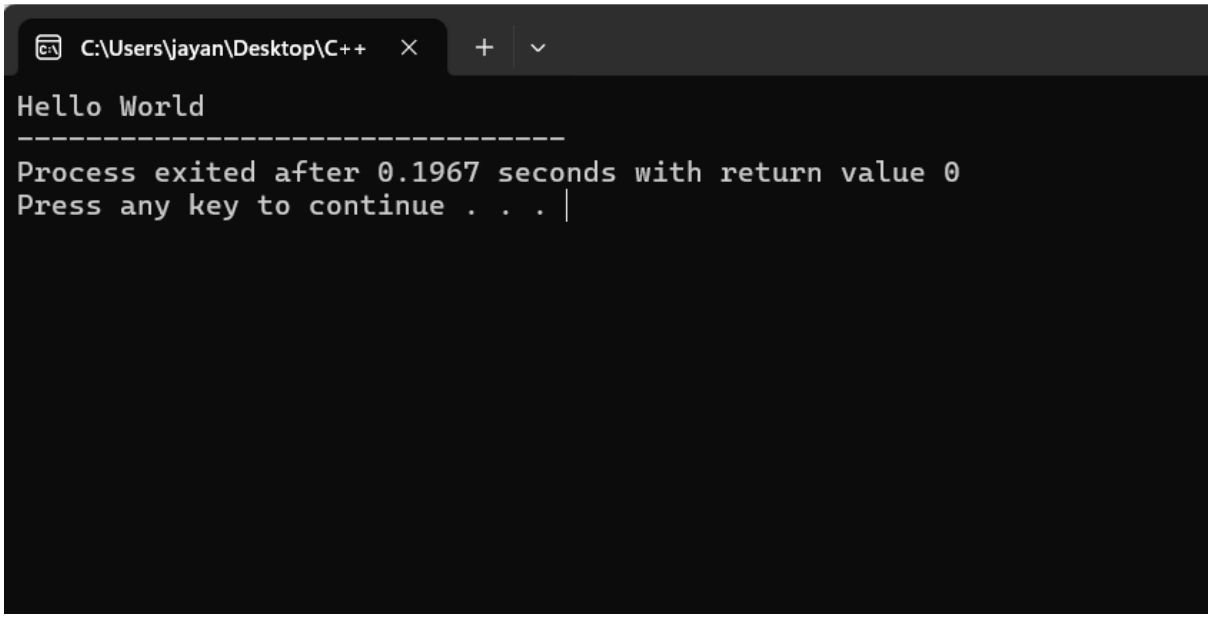
Step18: Lastly Write Source code in file and click compile and run button (file will be run).


C:\Users\jayan\Desktop\C++ and DSA\setup.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
setup.cpp
1 #include<iostream>
2 using namespace std;
3 main(){
4 cout<<"Setup Checking";
5 }
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
- Errors: 0
- Warnings: 0
- Output Size: 1.83193492889404 MiB
- Compilation Time: 1.67s

What is Console

(i)Console is an output window shown at runtime.

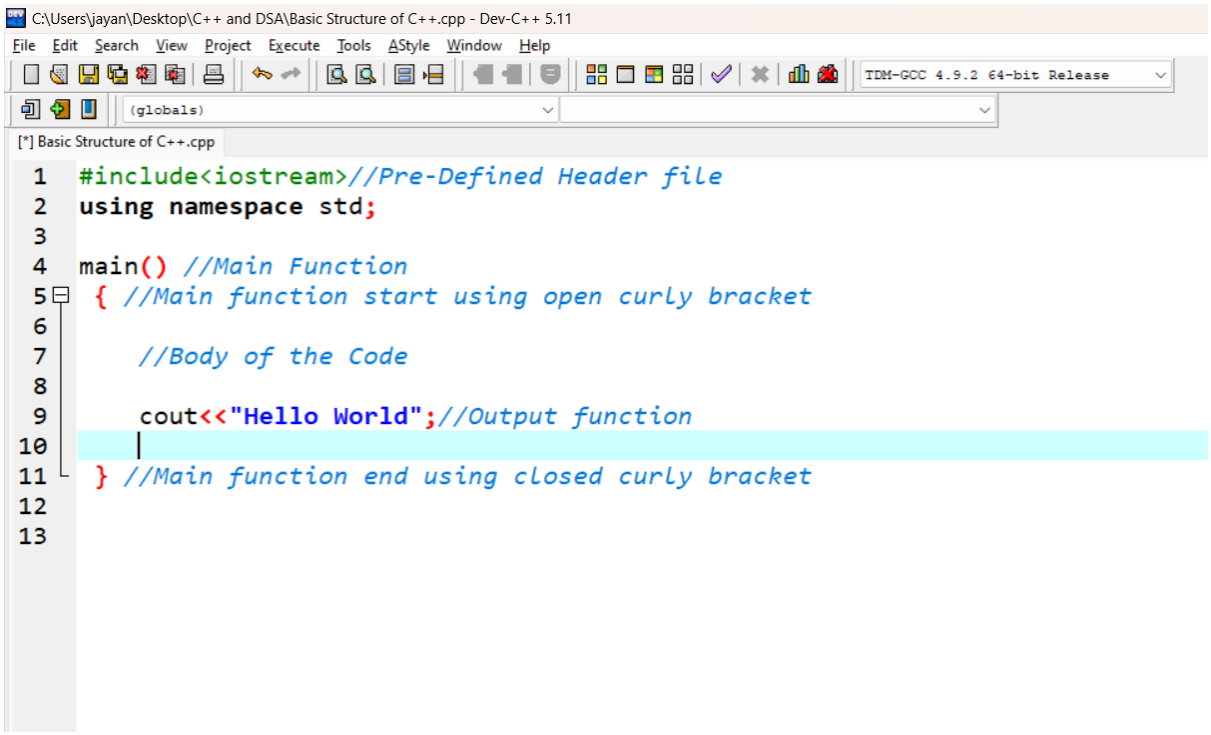
(ii)User cannot change result of the console window.



Tokens

Every Individual smallest unit is called Tokens like: keyword, identifier, literal, operator, or symbol etc.

Basic Structure of C++



Basic structure is compulsory code should write in C++ file because it is just like a protocol (rules and regulations) have necessary to write.

Steps:

Step1: #include<iostream> It is a pre-defined header file. It's contain basic input or output functions.

Step2: using namespace std using namespace std is a directive that brings all the identifiers from the std namespace into the current scope.

Step3: main() This is the main function, it's responsible for execution of source code.

Step4: { The open curly bracket is used for start the block of code.

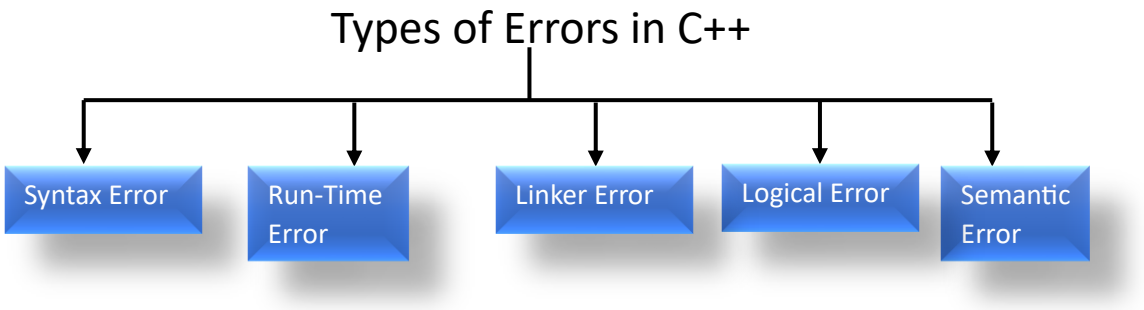
Step5: Write body or logic of the C++.

Step6: } The close curly bracket is used for end of the code block.

Errors in C++

(i)Errors are the problems or the fault that occur in the program, which makes the behavior of the program abnormal, and experienced developers can also make these faults.

(ii)Programming errors are also known as the bugs or faults, and the process of removing these bugs is known as debugging.

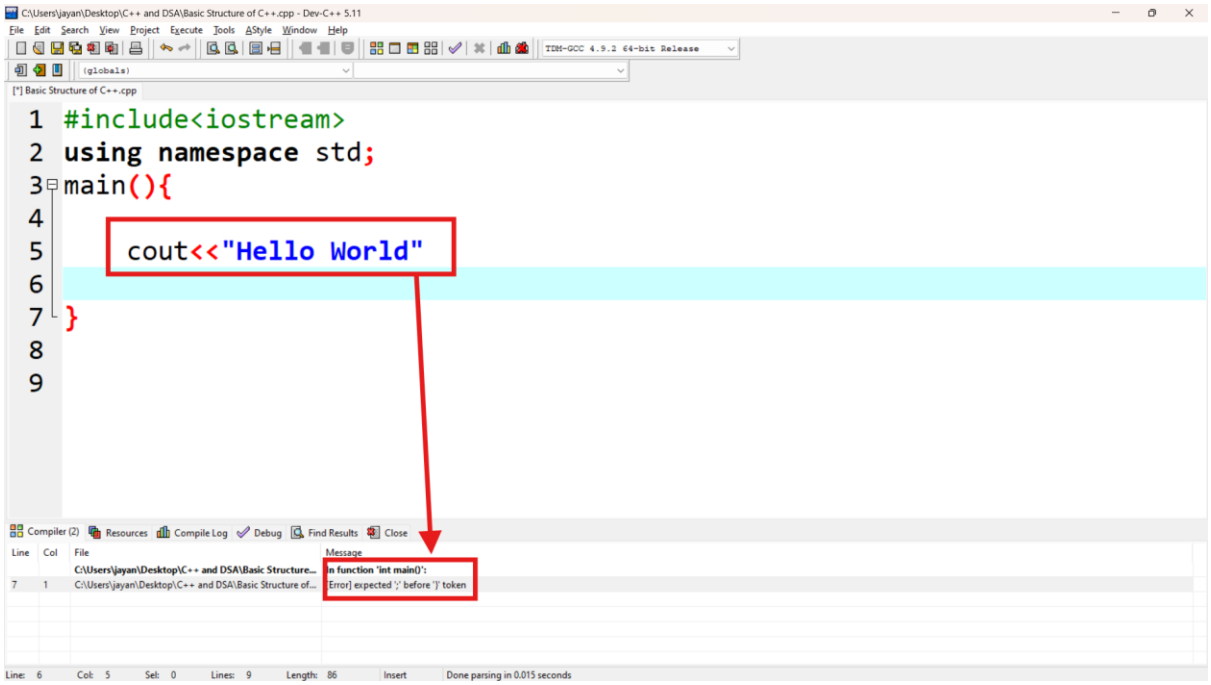


Syntax Error:(i) Syntax error are also known as the compilation errors as they occurred at the compilation time, or we can say that the syntax errors are thrown by the compilers.

(ii) These errors are mainly occurred due to the mistakes while typing or do not follow the syntax of the specified programming language.

(iii)These mistakes are generally made by beginners because they are new to the language. These errors can be easily debugged or corrected.

(iv)Example:(; { “ . . .)

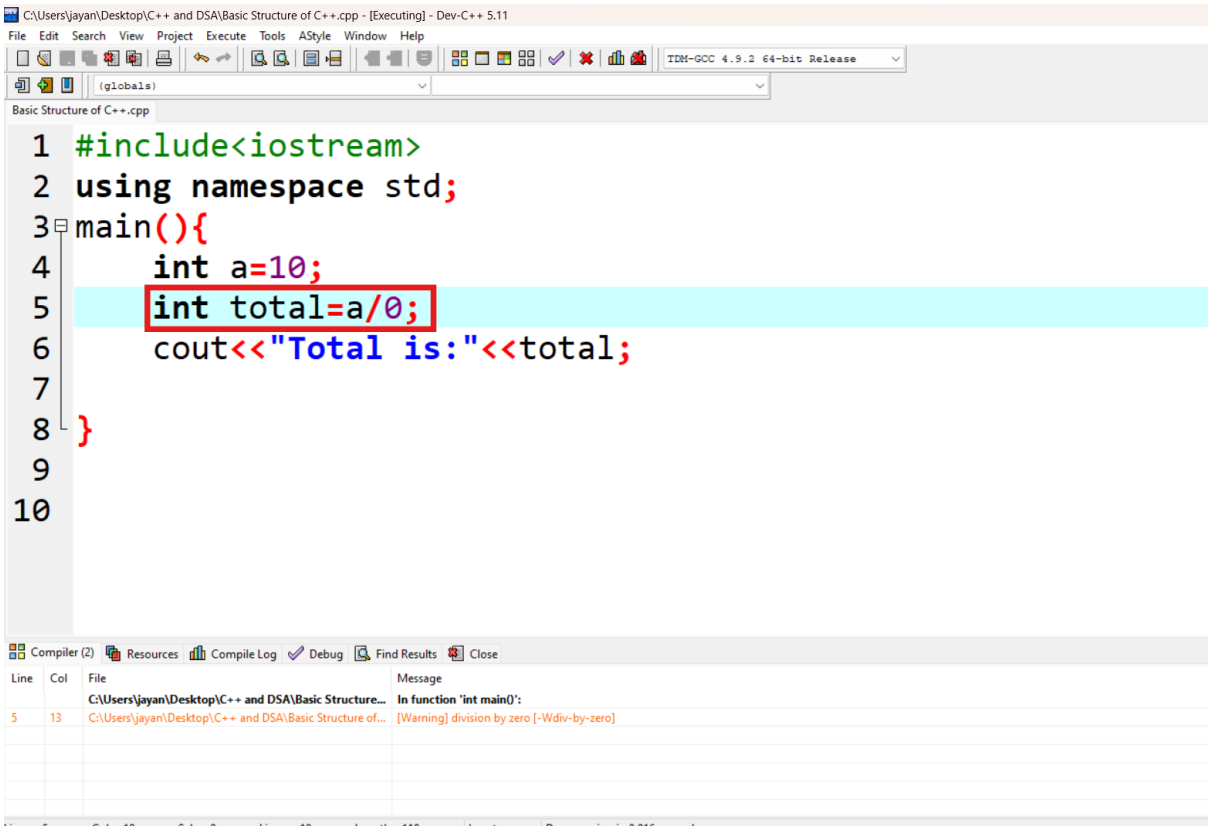


Run-Time Error:(i) Sometimes the errors exist during the execution-time even after the successful compilation known as run-time errors.

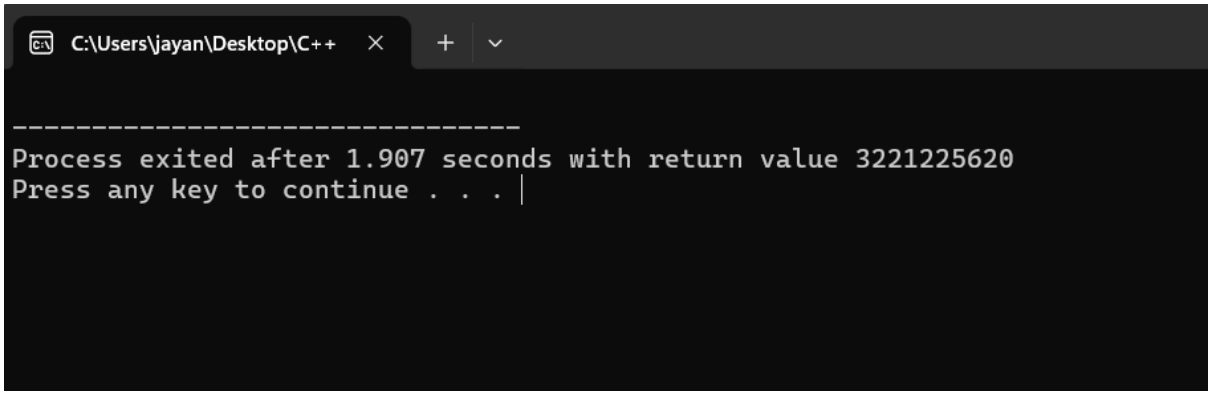
(ii)When the programming is running, and it is not able to perform the operation is the main cause of the run-time error.

(iii)The division by zero is the common example of the run-time error. This error is very difficult to find, as the compiler does not point to these errors.

(iv)Sometimes run-time error gives simplest warning but source code will be executed and it gives undesired-output.

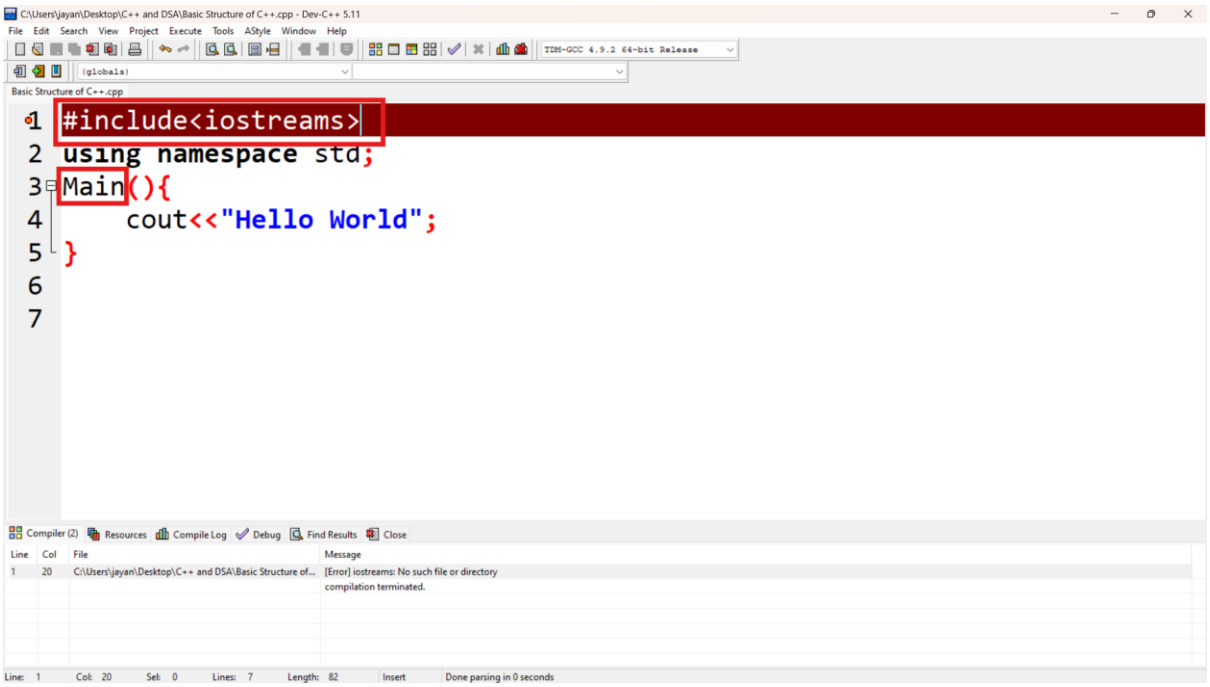


Output:



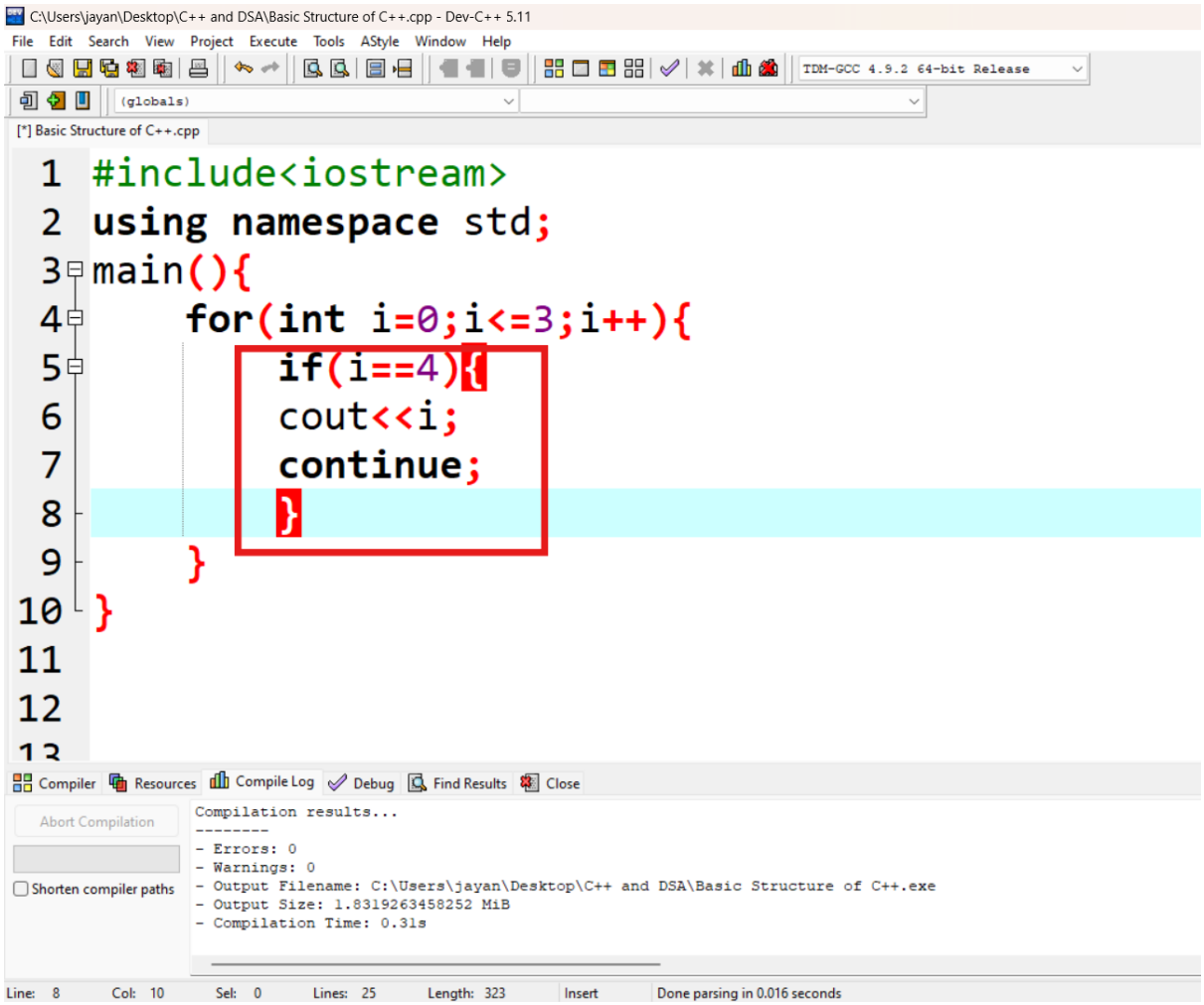
Note: Any number divided by 0 are gives not defined result in mathematics.

Linker Error: Linker errors are mainly generated when the executable file of the program is not created. This can be happened either due to the wrong function prototyping or usage of the wrong header file.

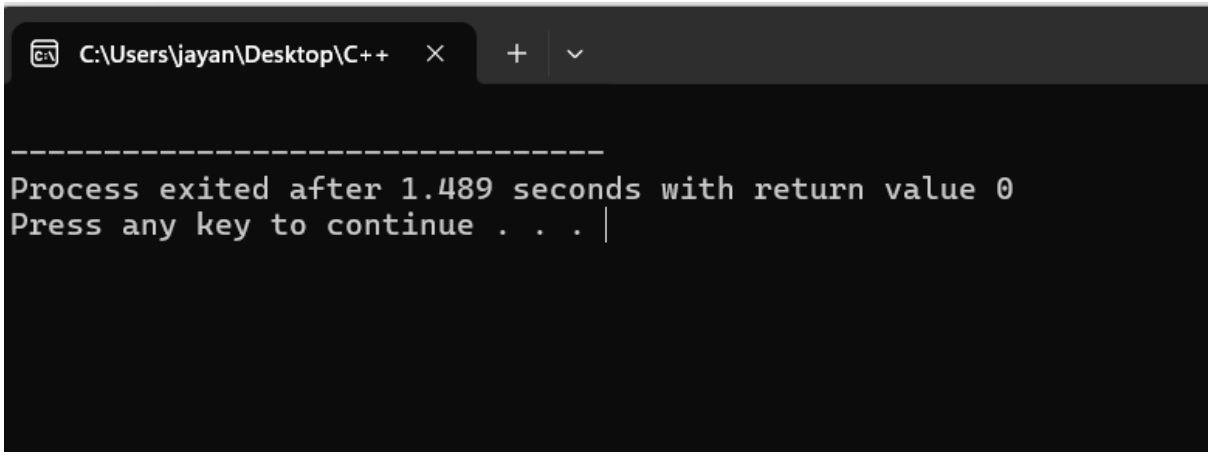


Logical Error:(i) The logical error is an error that leads to an undesired output. These errors produce the incorrect output, but they are error-free, known as logical errors.

(iii)These types of mistakes are mainly done by beginners. The occurance of these errors mainly depends upon the logical thinking of the developers.

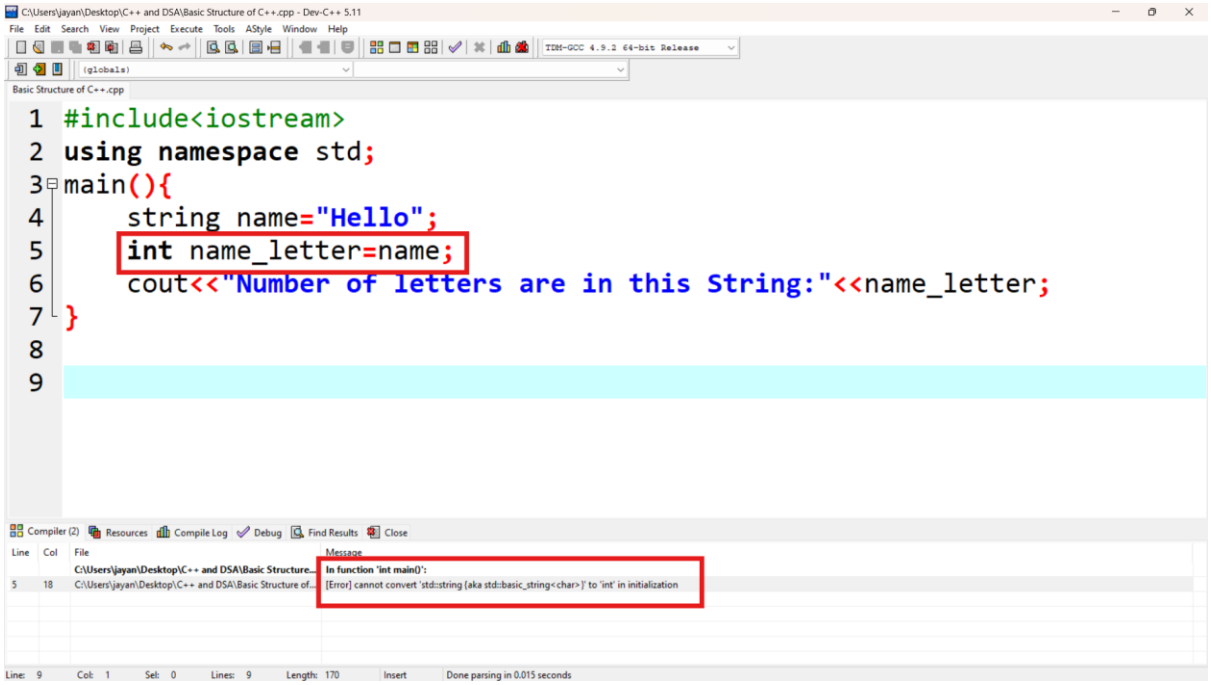

C:\Users\jayan\Desktop\C++ and DSA\Basic Structure of C++.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
[*] Basic Structure of C++.cpp
#include<iostream>
using namespace std;
main(){
for(int i=0;i<=3;i++){
if(i==4){
cout<<i;
continue;
}
}
}
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\jayan\Desktop\C++ and DSA\Basic Structure of C++.exe
- Output Size: 1.8319263458252 MiB
- Compilation Time: 0.31s
Line: 8 Col: 10 Sel: 0 Lines: 25 Length: 323 Insert Done parsing in 0.016 seconds

Output:


C:\Users\jayan\Desktop\C++

Process exited after 1.489 seconds with return value 0
Press any key to continue . . .

Semantic: Semantic error are the errors that occurred when the statements are not understandable by the compiler.



Variables

(i)Variable is a name given to a memory location where we can store different values of the same datatype during the program execution.

(ii)In other words, a variable can be defined as a storage container to hold values of the same datatype during the program execution.

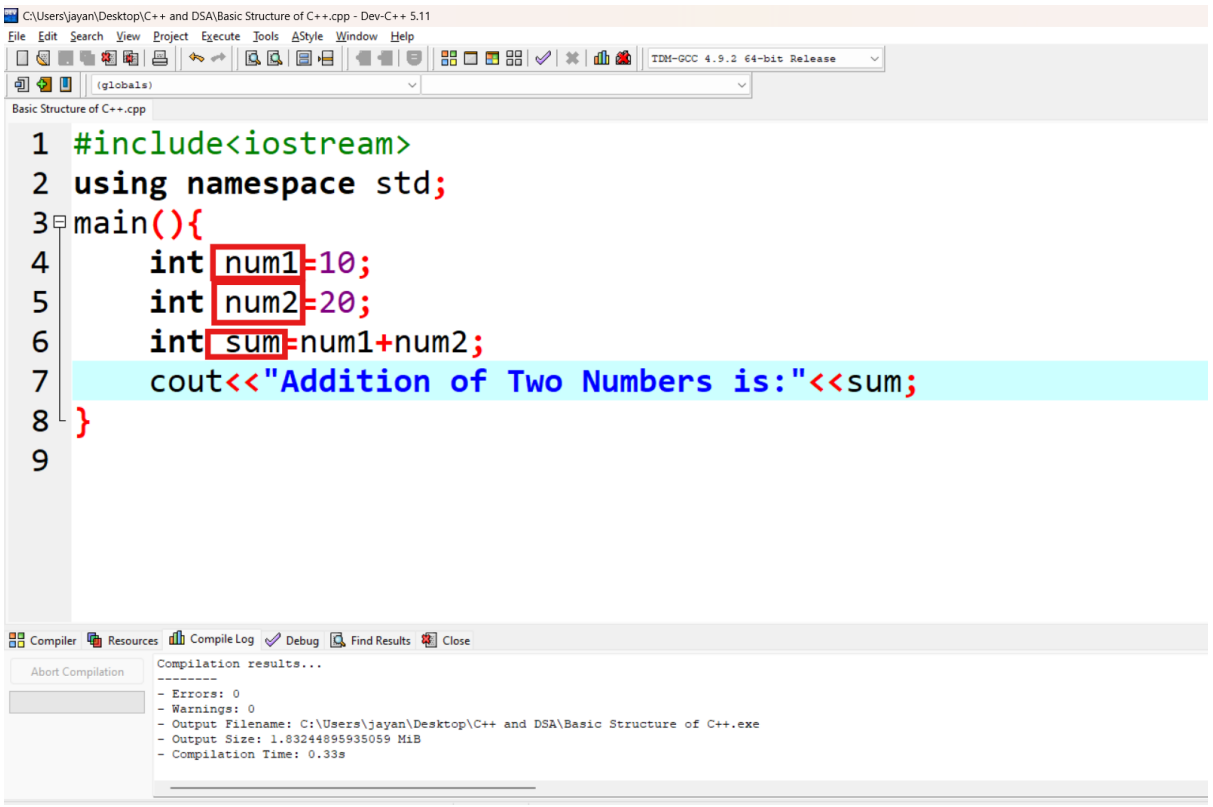
(iii)A variable name may contain letters, digits or underscore

Rules to specify a variable name:

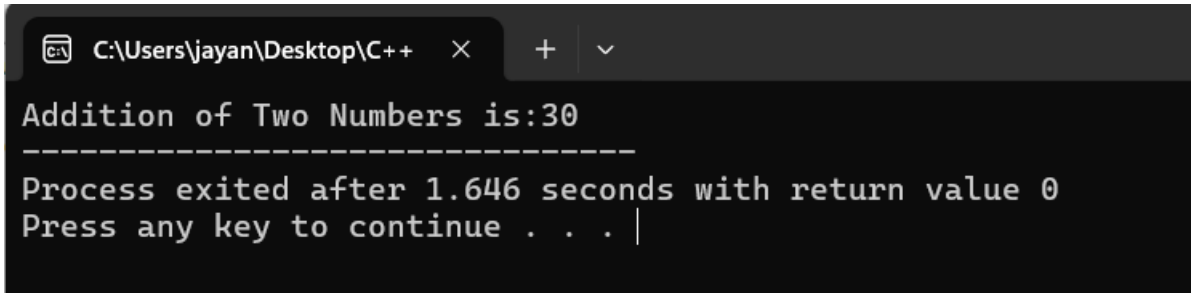
(i)Variable name should not start with a digit.

(ii)Keywords should not be used as variable name.

(iii)A variable name should not contain any special symbol except underscore (_).



Output:


C:\Users\jayan\Desktop\C++
Addition of Two Numbers is:30

Process exited after 1.646 seconds with return value 0
Press any key to continue . . .

Input/Output Function

Every C++ program perform two main functions: accept data as input, process data and produces output.

(i)Input function refers to accepting of data while output function refers to the presentation of data.

(ii)Normally input is accepted keyboard and output is displayed to screen.

(iii)C++ has two input output function: 1.cin 2. cout

Cin: cin is used for taking input from user while

Cout: cout is used for display output on console window.

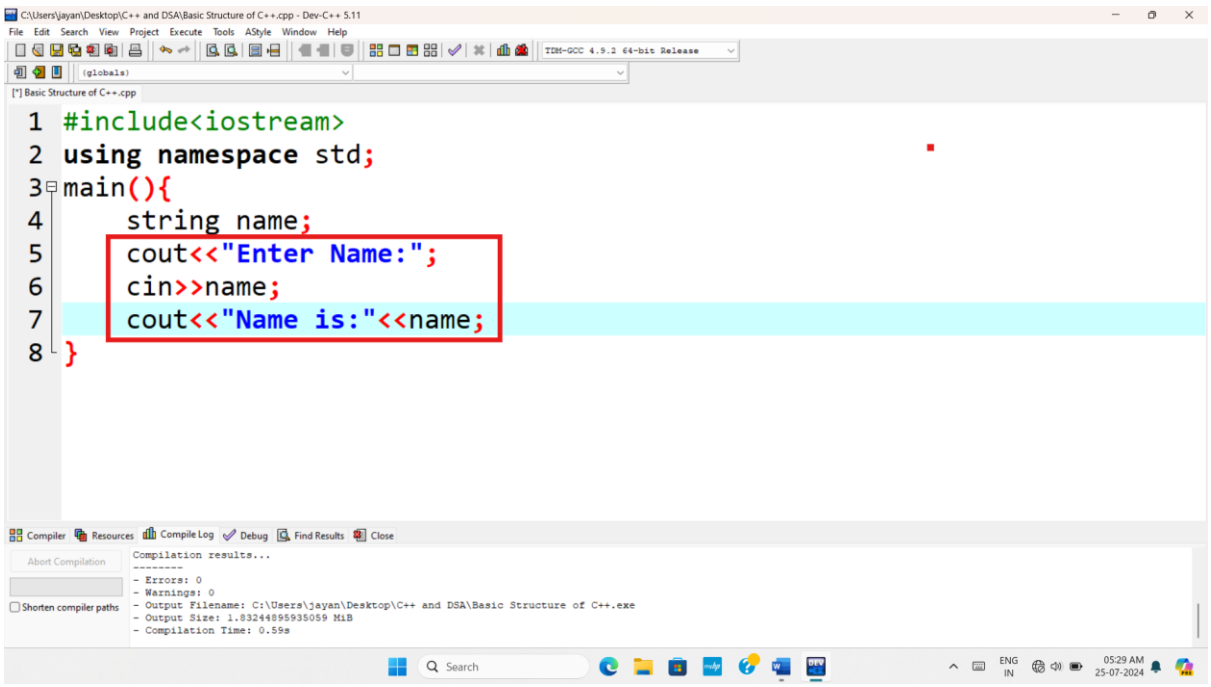
Syntax:

cin>>input_variable;

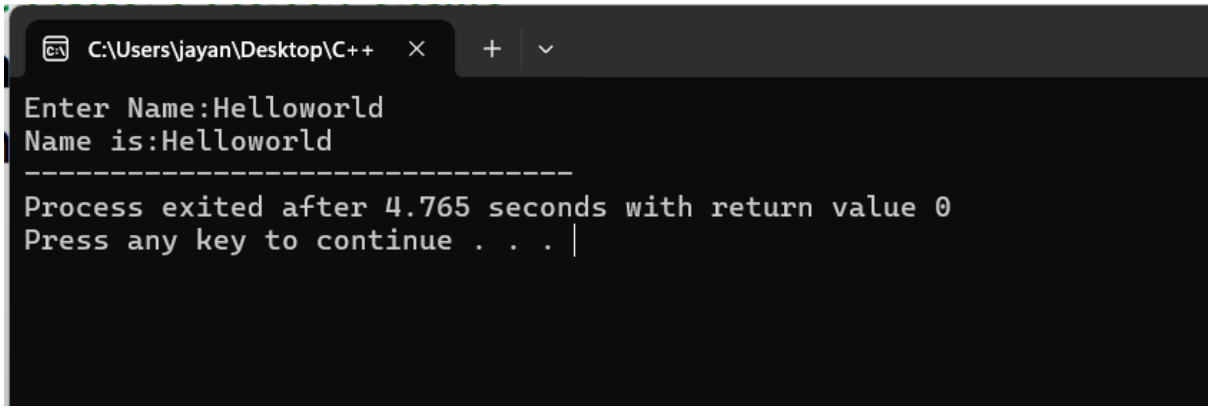
cout<<" Any type of string write in this but in inside of double column";

or

cout<<"Any type of string"<<output_variable;



Output:


C:\Users\jayan\Desktop\C++
Enter Name:Helloworld
Name is:Helloworld

Process exited after 4.765 seconds with return value 0
Press any key to continue . . .

Comments

(i)Comments is very important part of C++. Comments are used for explaining the source code and its added inside of C++ file.

(ii)It is basically designed for developers not a user.

(iii) C++ file can't give permission for execution of comments.

(iii)There are two types of comments in C++:

1.Single Line Comments

2.Multiline Comments

1.Single Line Comments:

Single line comments are used for single line.If you write some un-executable source code in single line using double slashes(//) is known as single line comments.

Syntax:

```
// This is Single line Comment
```

2.Multi-Line Comments:

Multiline Comment are used for comments in multiple lines. If you write some un-executable source code in multiple lines using (slashes and astrick) is known as multi-line comments.

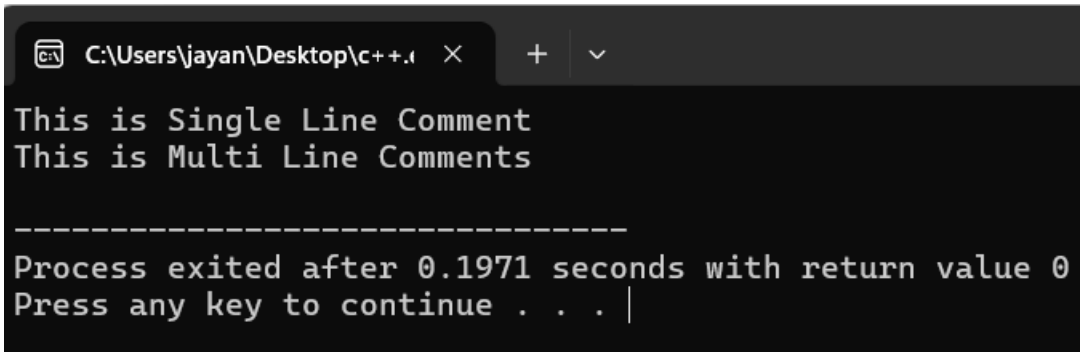
Syntax:

```
/* Multi Line Comments */
```

c++.cpp

```cpp
#include<iostream>
using namespace std;
//Main Function
main(){
    cout<<"This is Single Line Comment"<<endl;
    //Single Line Comments

    cout<<"This is Multi Line Comments"<<endl;
    /*
    Multi Line Comments
    */
}
```

Output:



Keywords

(i)In the C++ programming language, there are 95 keywords. All 95 keywords have their meaning which is already known the compiler.

(ii)Keywords are the reserved words with predefined meaning which already known to the compiler.

(iii)Whenever C++ compiler come across a keyword, automatically it understands its meaning.

Properties of keywords

(i)All the keywords in C++ programming language are defined as lowercase letters, so they must be used only in lowercase letter.

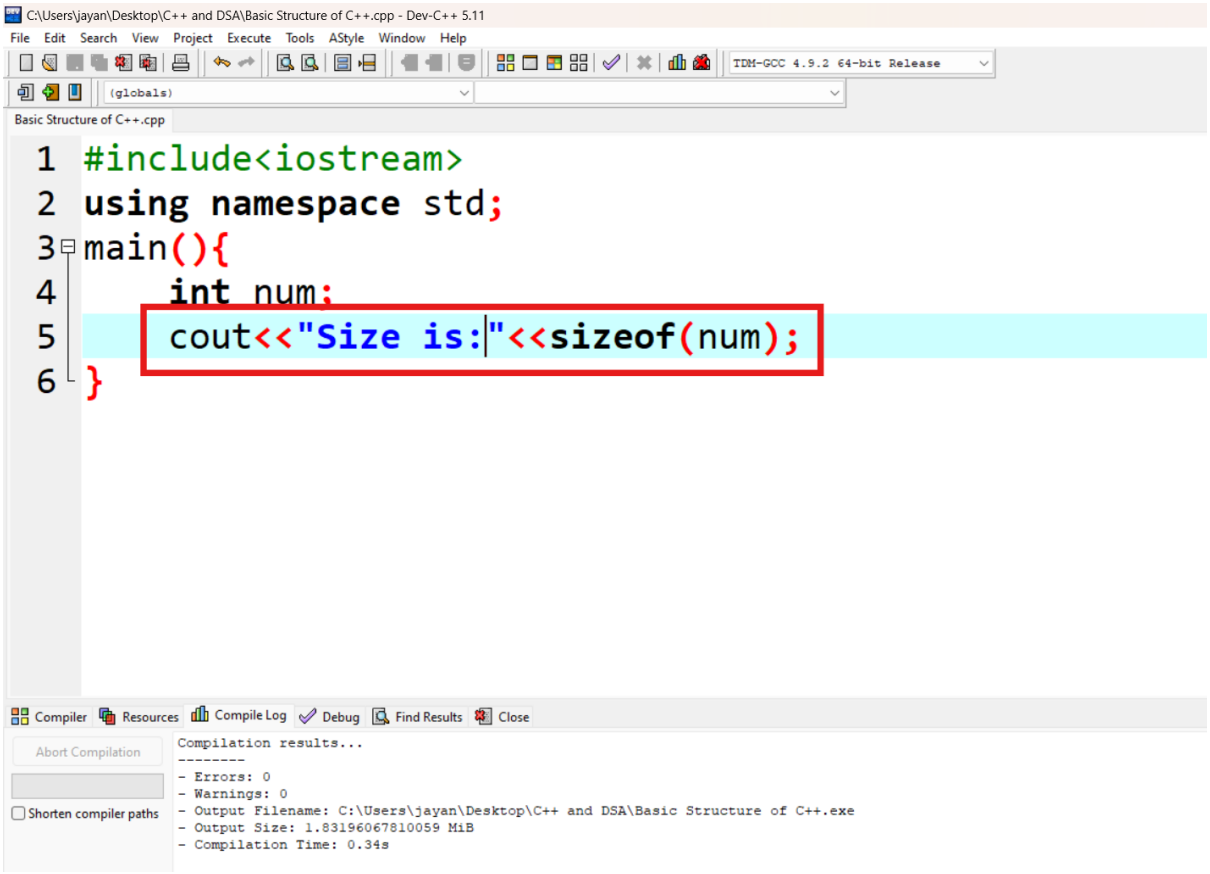
(ii)Every keywords has specific meaning, users can not change that meaning.

(iii)Keywords cannot be used as user-defined names like variable, functions, arrays, pointers,etc.

(iv)Every keywords of C++ represents something or specifies some kind of a action to be formed by the compiler.

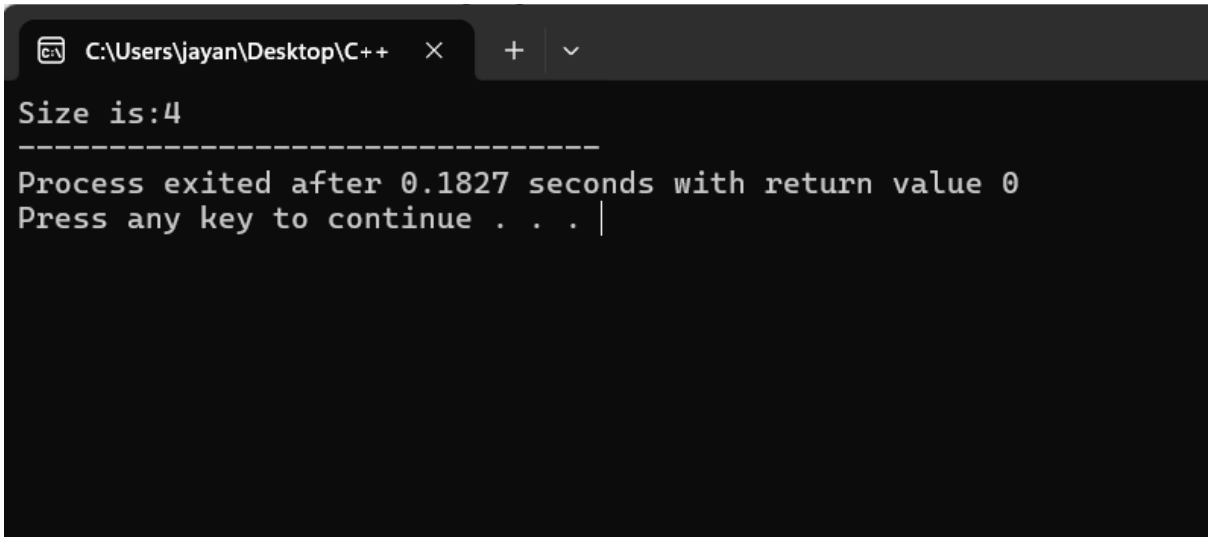
List of Keywords:

asm	double	New	switch
auto	else	operator	template
break	enum	private	this
case	extern	protected	throw
Catch	float	public	try
char	for	register	typedef
class	friend	return	union
const	goto	short	unsigned
continue	if	signed	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while


C:\Users\jayan\Desktop\C++ and DSA\Basic Structure of C++.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Basic Structure of C++.cpp
1 #include<iostream>
2 using namespace std;
3 main(){
4 int num;
5 cout<<"Size is:"<<sizeof(num);
6 }
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\jayan\Desktop\C++ and DSA\Basic Structure of C++.exe
- Output Size: 1.83196067810059 MiB
- Compilation Time: 0.34s

Output:


C:\Users\jayan\Desktop\C++
Size is:4

Process exited after 0.1827 seconds with return value 0
Press any key to continue . . .

Identifiers

An identifier is a collection of characters which acts as the name of variable, function, array, pointer, structure, etc.

Rules for creating an identifier

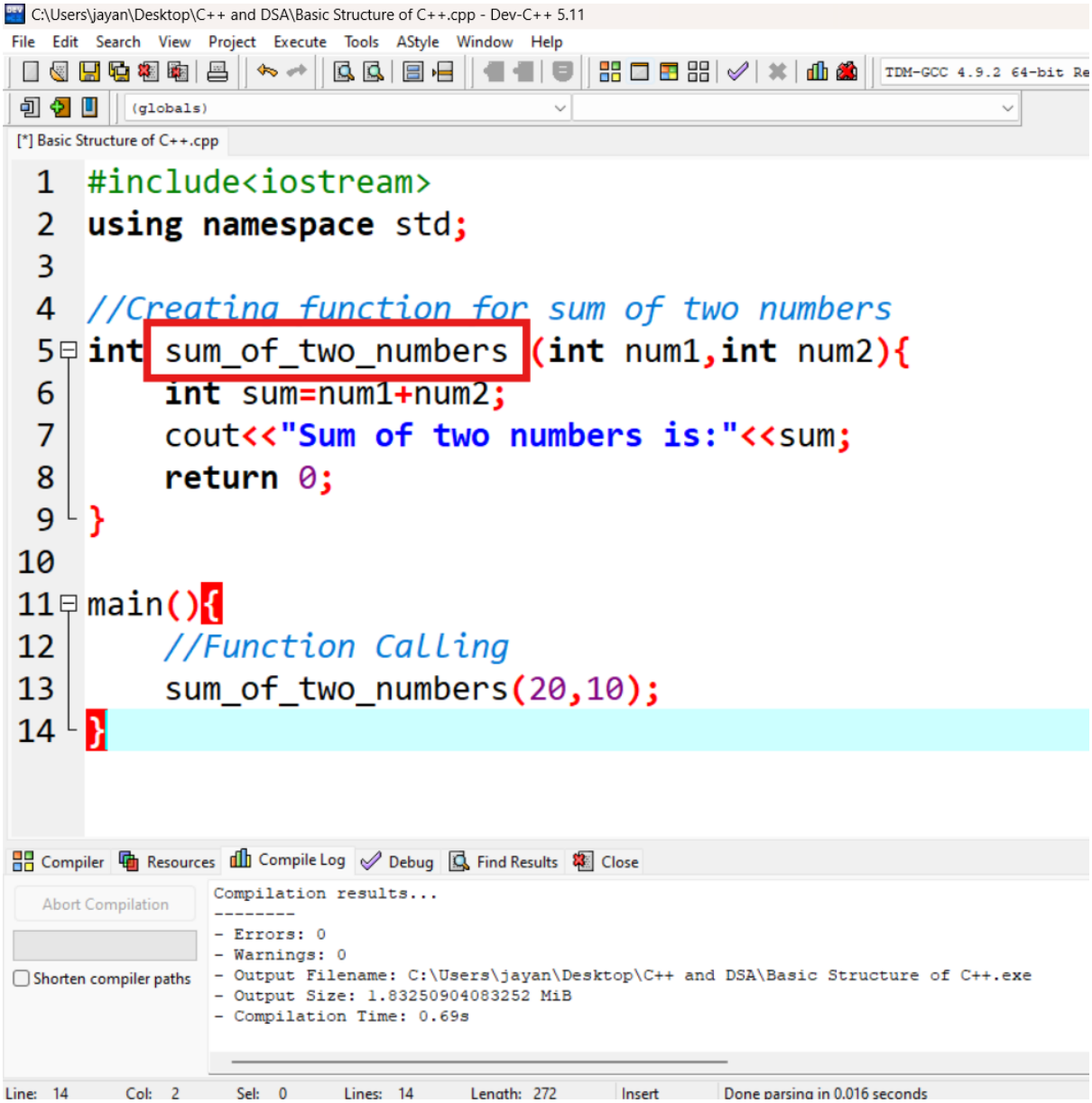
(i)An identifier can contain letters (upper and lower case), numerics &underscore symbol only.

(ii)An identifier should not start with a numerical value. It can start with a letter or an underscore.

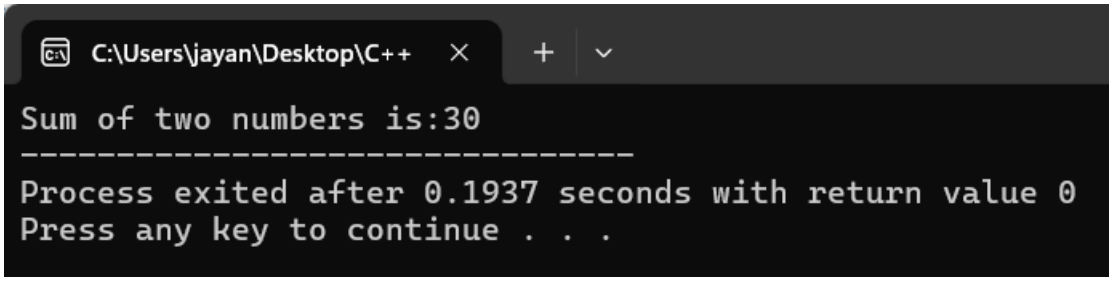
(iii)We should not use any special symbol in between the identifier even whitespace. However, the only underscore symbol is allowed.

(iv)Keywords shouldn't be used as identifier.

(v)An identifier must be unique in its scope.


C:\Users\jayan\Desktop\C++ and DSA\Basic Structure of C++.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Re
(globals)
[*] Basic Structure of C++.cpp
#include<iostream>
using namespace std;
//Creating function for sum of two numbers
int sum_of_two_numbers (int num1,int num2){
int sum=num1+num2;
cout<<"Sum of two numbers is:"<<sum;
return 0;
}
main(){
//Function Calling
sum_of_two_numbers(20,10);
}
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\jayan\Desktop\C++ and DSA\Basic Structure of C++.exe
- Output Size: 1.83250904083252 MiB
- Compilation Time: 0.69s
Line: 14 Col: 2 Sel: 0 Lines: 14 Length: 272 Insert Done parsing in 0.016 seconds

Output


C:\Users\jayan\Desktop\C++
Sum of two numbers is:30
Process exited after 0.1937 seconds with return value 0
Press any key to continue . . .

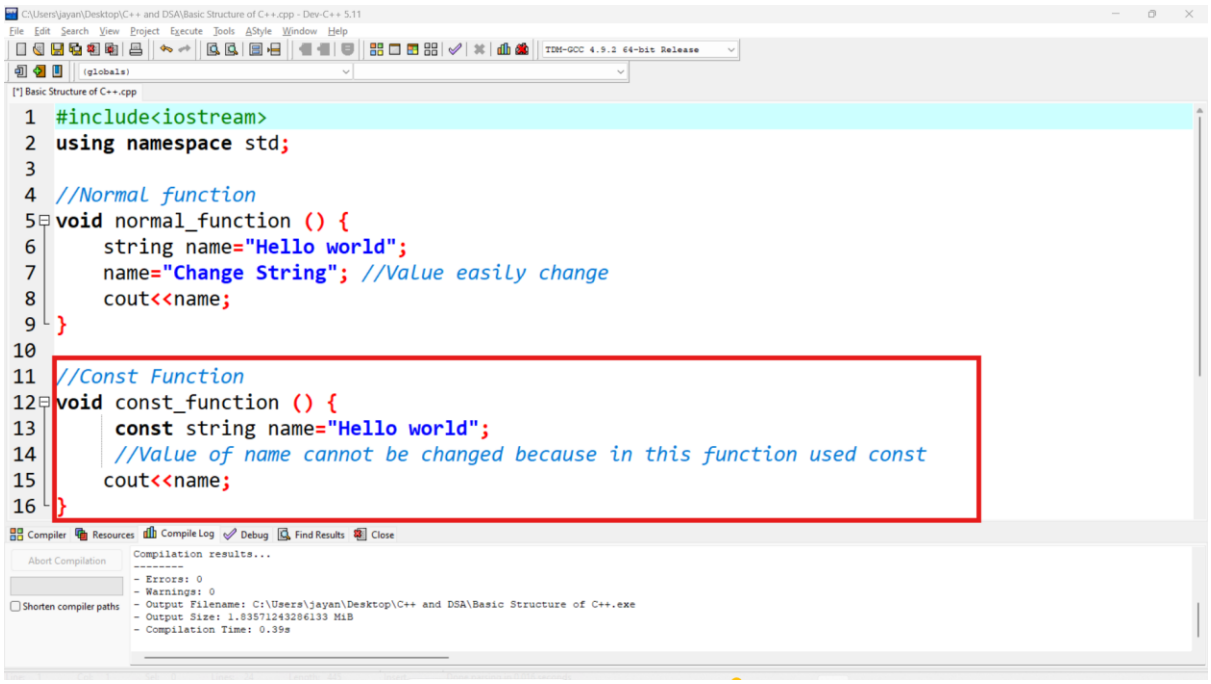
Constants

(i)A constant is a named memory location which holds only one value throughout the program execution.

(ii)In C++ programming, a constant can be of any data types like: int, float, char, string, double.

Syntax:

const datatype variable_name;



```
16 }
17
18 //main function
19 main(){
20     normal_function();
21     cout<<endl;
22     const_function();
23 }
24
```

Compiler | Resources | Compile Log | Debug | Find Results | Close

Abort Compilation

Shorten compiler paths

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\jayan\Desktop\C++ and DSA\Basic Structure of C++.exe
- Output Size: 1.83571243286133 MiB
- Compilation Time: 0.39s
```

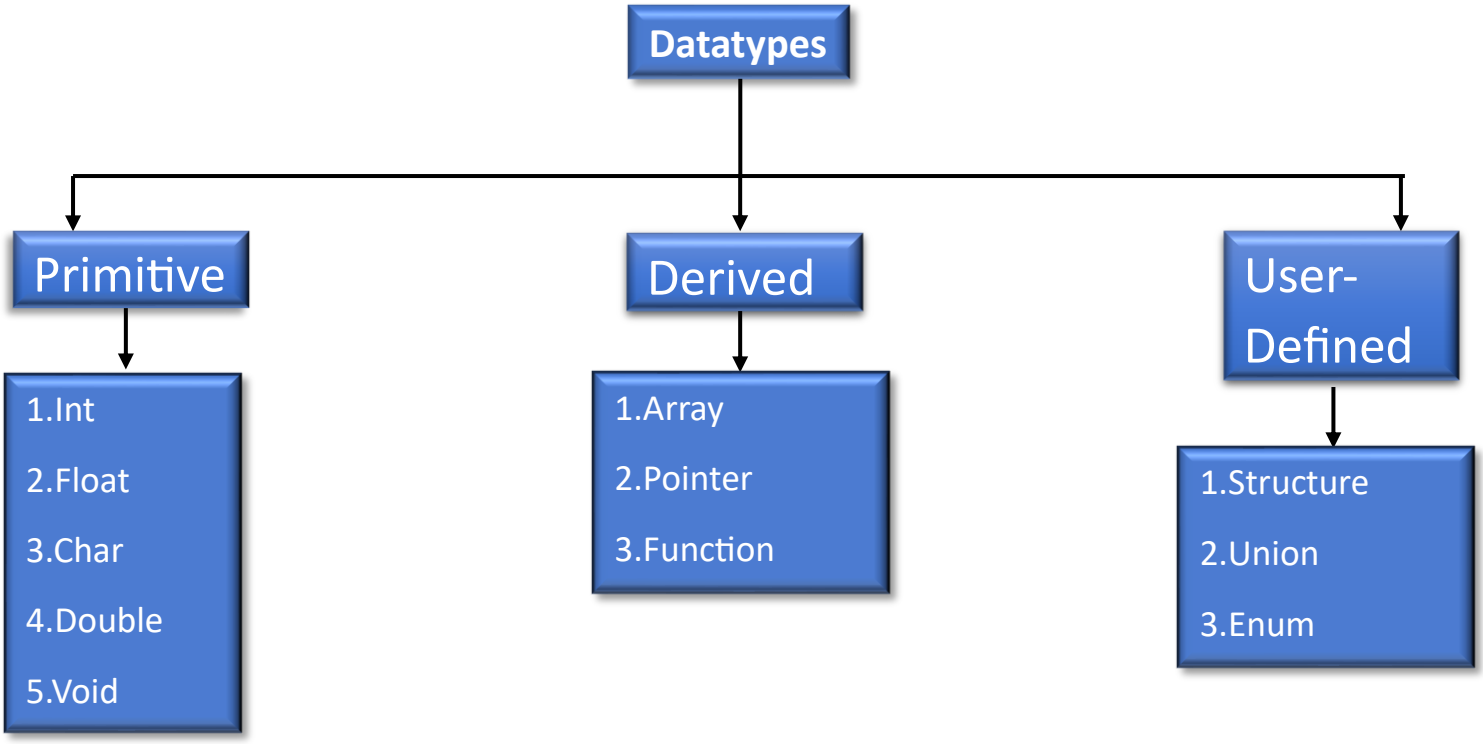
Line: 1 Col: 1 Sel: 0 Lines: 24 Length: 445 Insert Done parsing in 0.016 seconds

Output:

```
C:\Users\jayan\Desktop\C++
Change String
Hello world
--------------------------------
Process exited after 0.1885 seconds with return value 0
Press any key to continue . . .
```

Datatypes

In C++ programming, each variable or constant or array must have a data type and this type specifies how much memory is to be allocated and what type of values are to be stored in that variable, constant or array.



Integer Data Types: The integer datatype is a set of whole numbers. Every integer value does not have the decimal value. We use the keyword 'int' to represent integer data type in C. We use the keyword int to declare the variables and to specify the return type of a function.

int x;

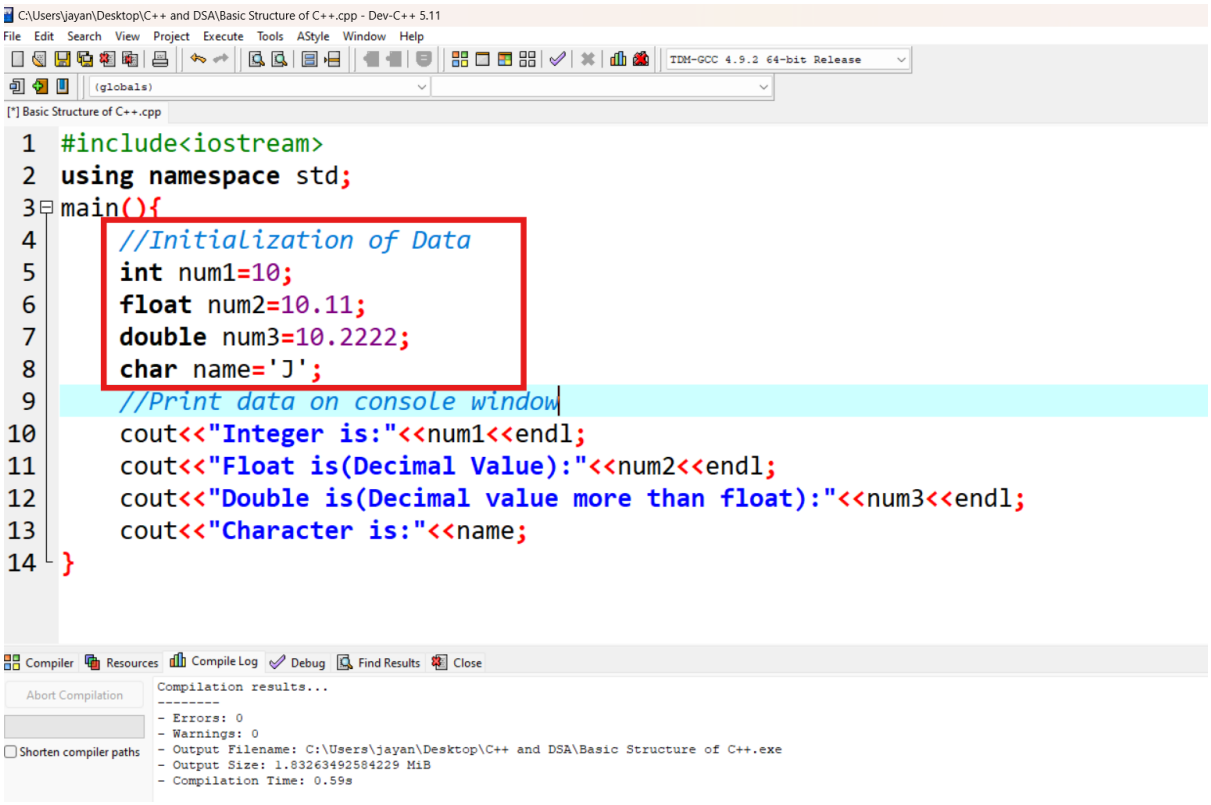
Float Data Types: Floating point data types are a set of numbers with the decimal value. Every floating-point value must contain the decimal value. The floating points data type has two variants.

(i)float

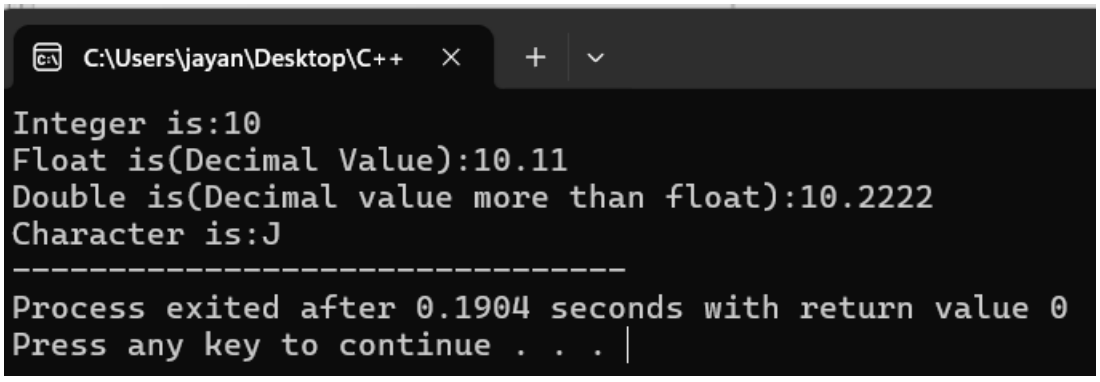
(ii)double

We can use the keyword “float” to represent floating point datatype and “double” to represent double datatype in C++.Both float and double are similar but they differ in the number of decimal places. The float value contains 6 decimal places whereas double value contains 15 decimal places.

Character Data Types: The Character data type is a set of characters enclosed in single quotes.



Output:



Formula used to calculate range of datatypes:

Signed: -2^{x-1} to $+2^{x-1}-1$

Unsigned: 0 to $2^{x}-1$

Note: Where (x) is the number of bits used to store data.

Example:2^{16}

Signed: -2^{x-1} to $+2^{x-1}-1$

$-2^{16}-1$ to $2^{16-1}-1$

-2^{15} to $2^{15}-1$

-32768 to 32768-1

-32768 to 32767

Unsigned: 0 to $2^{x}-1$

0 to $2^{16}-1$

0 to 65536-1

0 to 65535

Note: Derived and User-Defined data types are explain separately because its carry lot of functionality and huge amount of data.

Header Files and its types

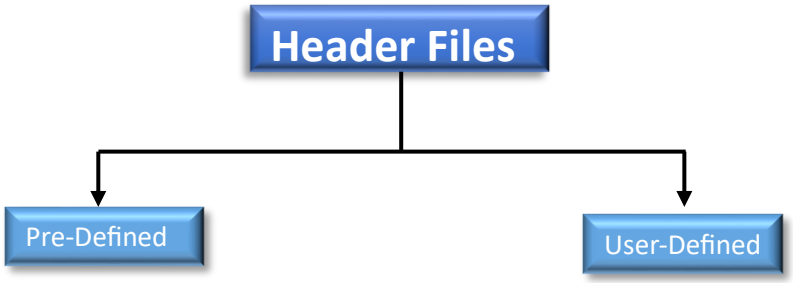
(i)C++ has numerous libraries that include pre-defined functions to make programming easier.

(ii)In C++, header files contain the set of predefined standard library functions. It's use "#include".

(iii)All the header files have a ".h" extension.

(iv)These preprocessor directives are used for instructing compiler that these files need t be processed before compilation.

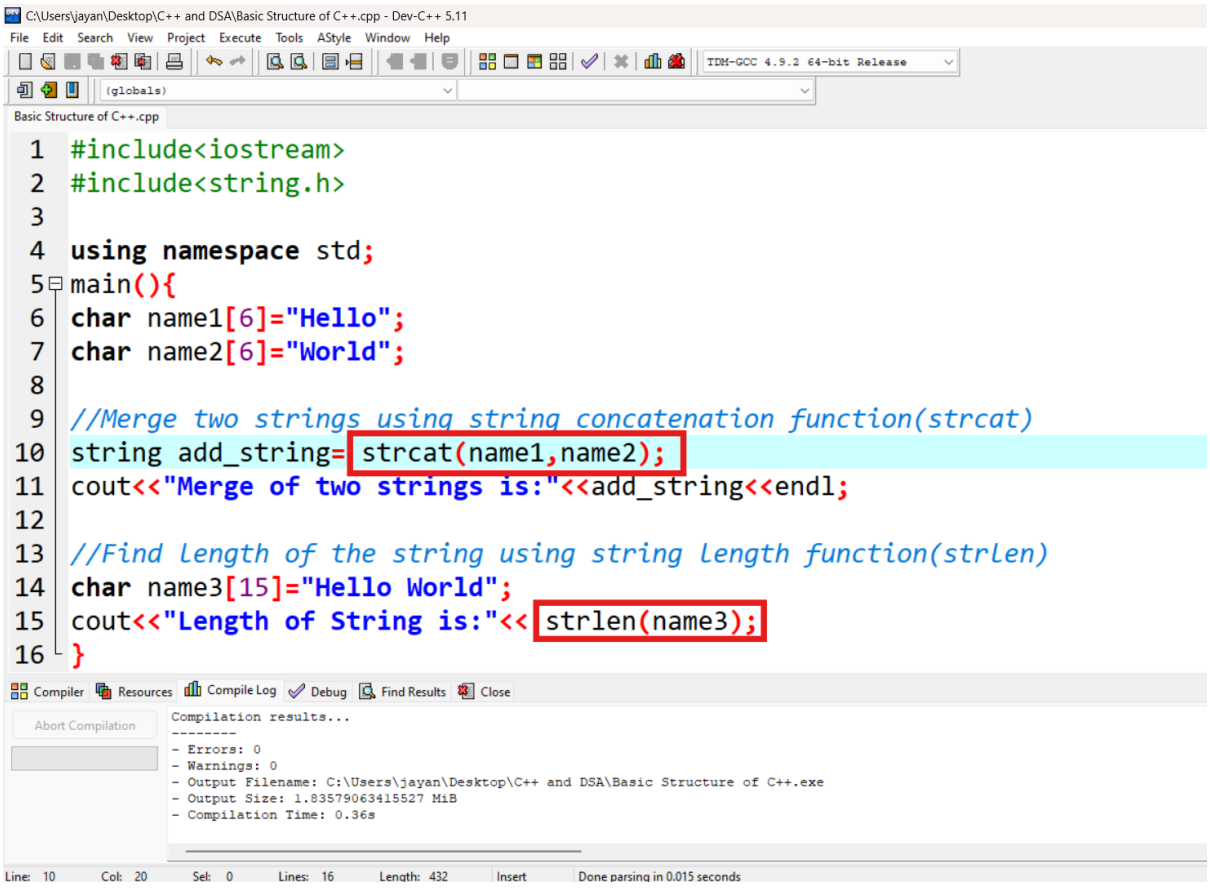
(v)In C++ source code should necessarily contain the header file which stands for standard input and output used to take input with the help of cout or cin function respectively.



Pre-Defined Header files:(i) Files which are already available in C++ compiler we just need to import them.

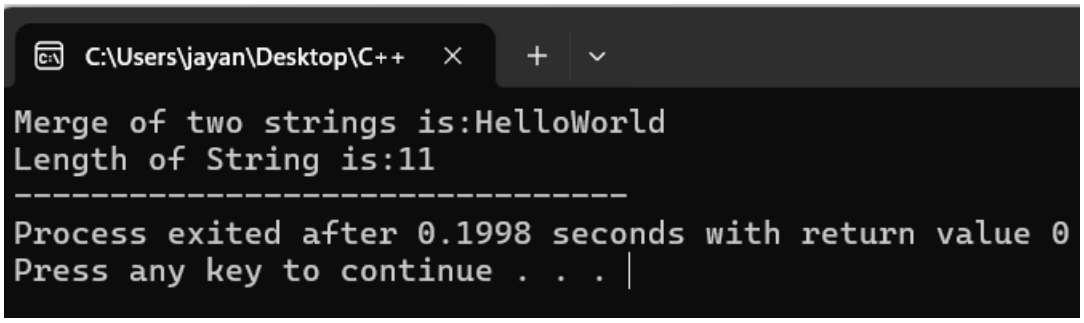
(ii)User cannot change pre-defined function because functions already are in pre-defined header files but values of functions are change easily.

(iii)There are various types of header files present in C++ like: <string.h>, <iostream.h>, <vector.h>, <conio.h>, and so on.


C:\Users\jayan\Desktop\C++ and DSA\Basic Structure of C++.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Basic Structure of C++.cpp
1 #include<iostream>
2 #include<string.h>
3
4 using namespace std;
5 main(){
6 char name1[6]="Hello";
7 char name2[6]="World";
8
9 //Merge two strings using string concatenation function(strcat)
10 string add_string= strcat(name1,name2);
11 cout<<"Merge of two strings is:"<<add_string<<endl;
12
13 //Find Length of the string using string length function(strlen)
14 char name3[15]="Hello World";
15 cout<<"Length of String is:"<< strlen(name3);
16 }
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\jayan\Desktop\C++ and DSA\Basic Structure of C++.exe
- Output Size: 1.83579063415527 MiB
- Compilation Time: 0.36s
Line: 10 Col: 20 Sel: 0 Lines: 16 Length: 432 Insert Done parsing in 0.015 seconds

Output:


C:\Users\jayan\Desktop\C++
Merge of two strings is:HelloWorld
Length of String is:11

Process exited after 0.1998 seconds with return value 0
Press any key to continue . . .

User-defined Header File:(i) These files are defined by the user and can be import using "#include".

(ii)User easily create own functions for reusability of code.

(iii)It decrease the length of code.

(iv) For create user defined header file use (.h) extension.

(v) Don't use main function for create user-defined header file.

(vi)When import the header file in main file of C++ then use double quotes (" ") for use of pre-defined header file functions.

Steps for creating user-defined header file:

User defined header file using (.h) extension

Step1: Include necessary predefined header file have need for our source code like: <iostream>

Step2: Write using namespace std;

Step3: Create functions

Step4: Save file with .h extension

Main file using (.cpp) extension

Step1: Import iostream header file or user defined header file using " " double quotes.

Step2: Write using namespace std;

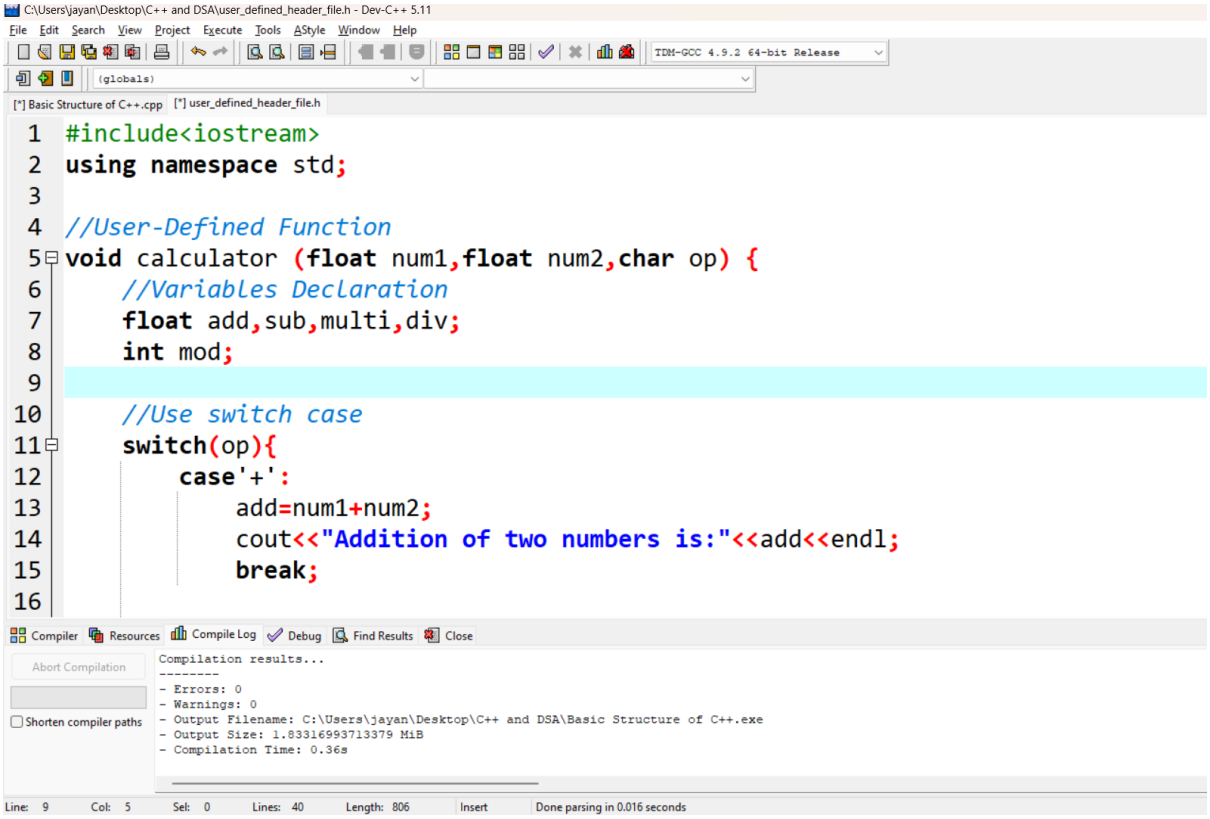
Step3: Make main function for execution of source code.

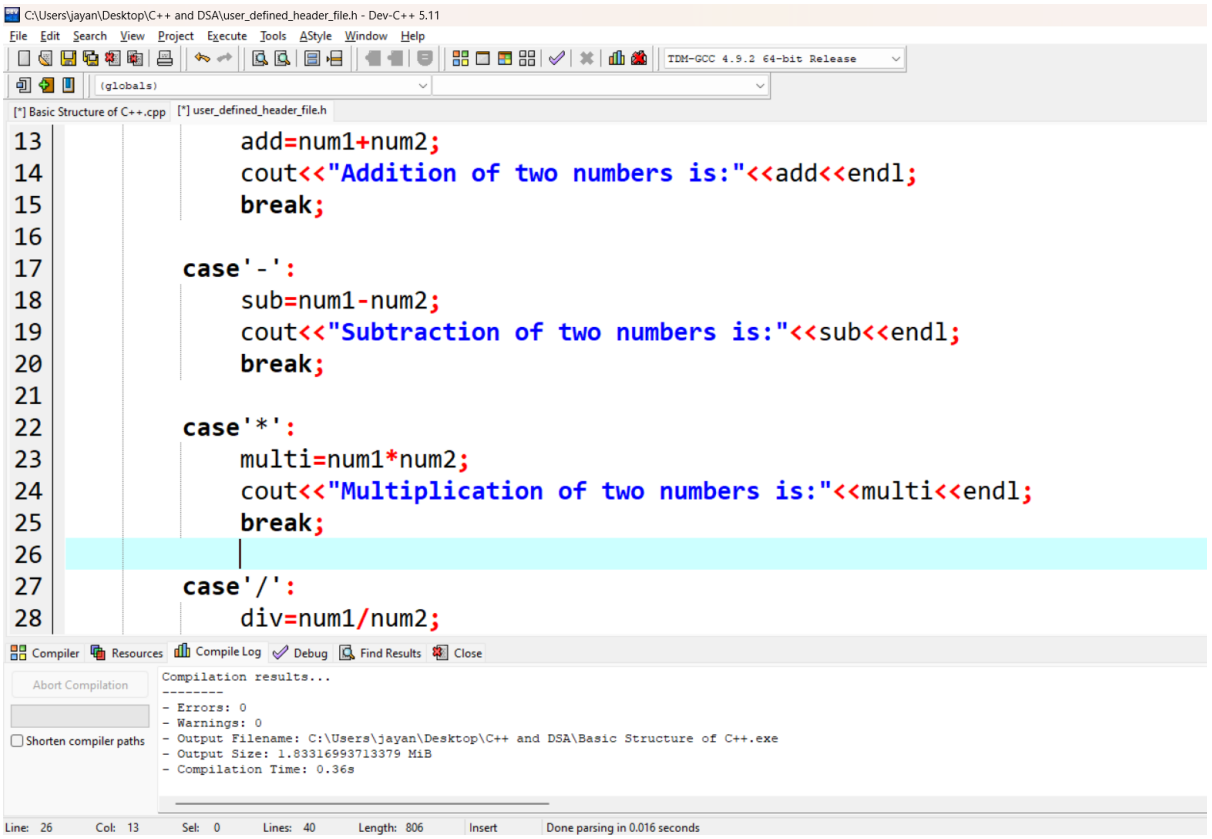
Step4: Call user-defined functions.

Step5: Save the file using (.cpp) extension

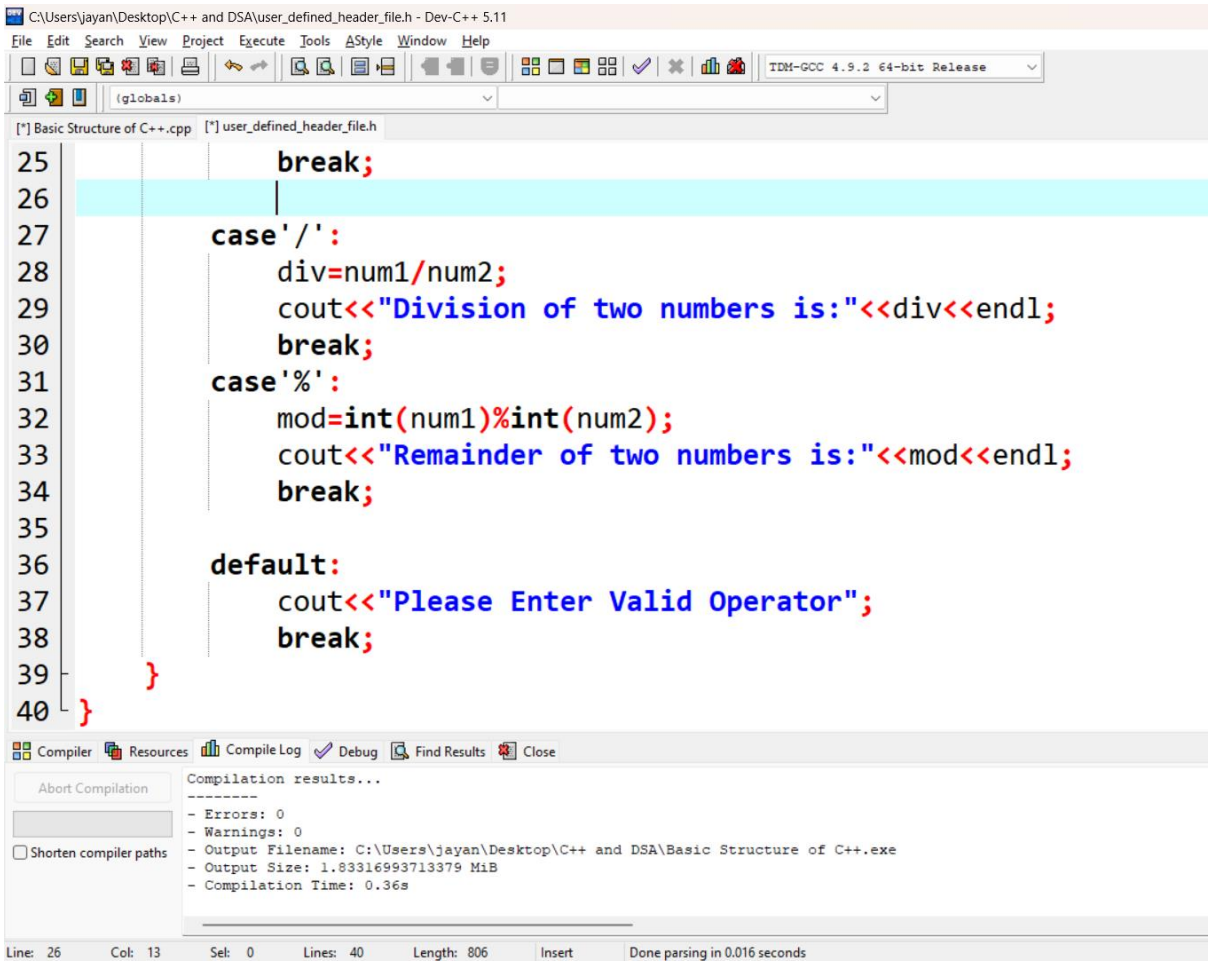
Source code of user-defined header file

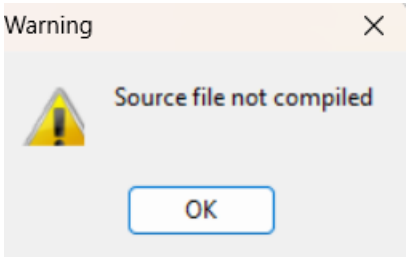
S



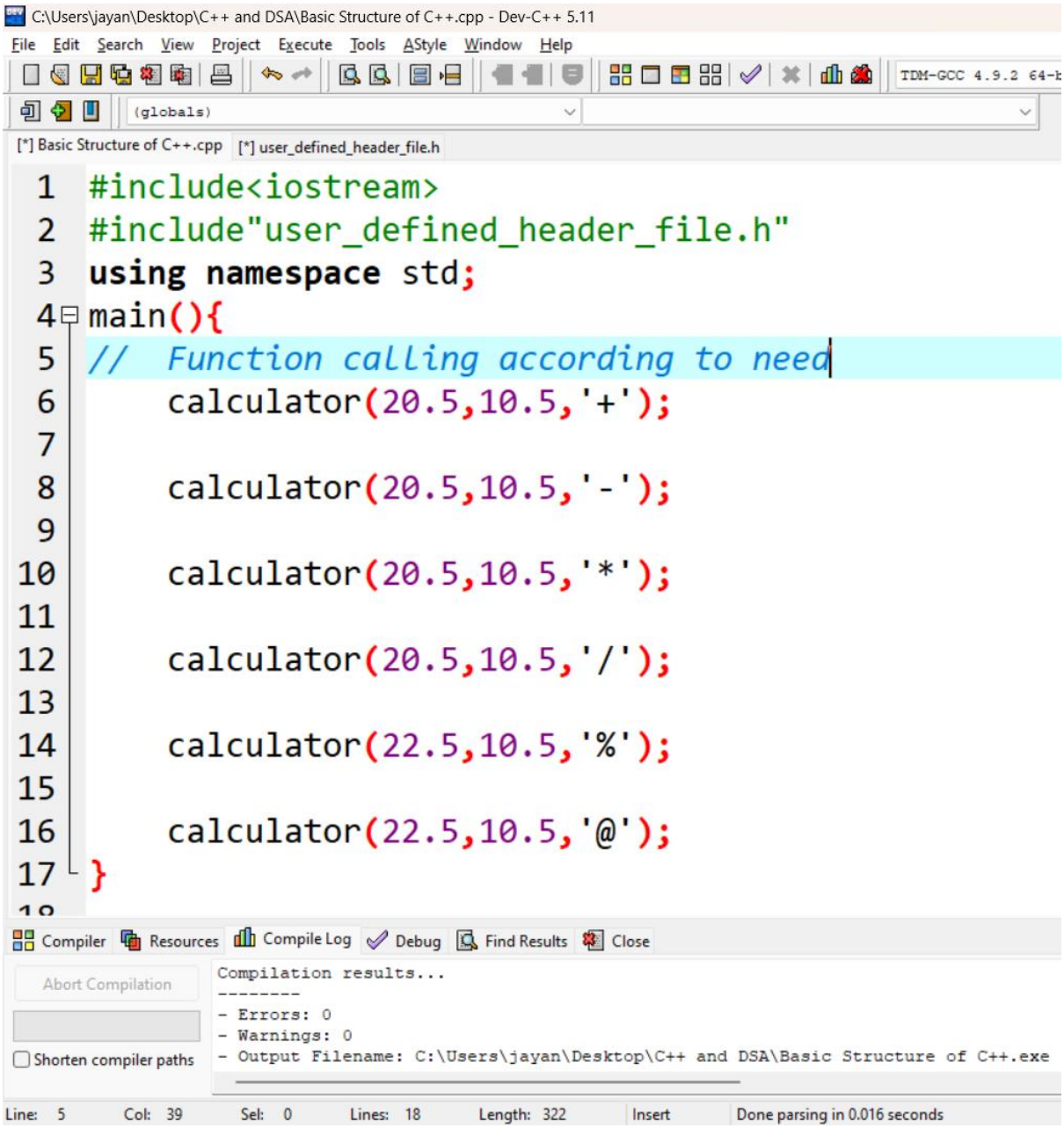

C:\Users\jayan\Desktop\C++ and DSA\user_defined_header_file.h - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
[*] Basic Structure of C++.cpp
[*] user_defined_header_file.h
13 add=num1+num2;
14 cout<<"Addition of two numbers is:"<<add<<endl;
15 break;
16
17 case'-':
18 sub=num1-num2;
19 cout<<"Subtraction of two numbers is:"<<sub<<endl;
20 break;
21
22 case'*':
23 multi=num1*num2;
24 cout<<"Multiplication of two numbers is:"<<multi<<endl;
25 break;
26
27 case'/':
28 div=num1/num2;
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\jayan\Desktop\C++ and DSA\Basic Structure of C++.exe
- Output Size: 1.83316993713379 MiB
- Compilation Time: 0.36s
Line: 26 Col: 13 Sel: 0 Lines: 40 Length: 806 Insert Done parsing in 0.016 seconds

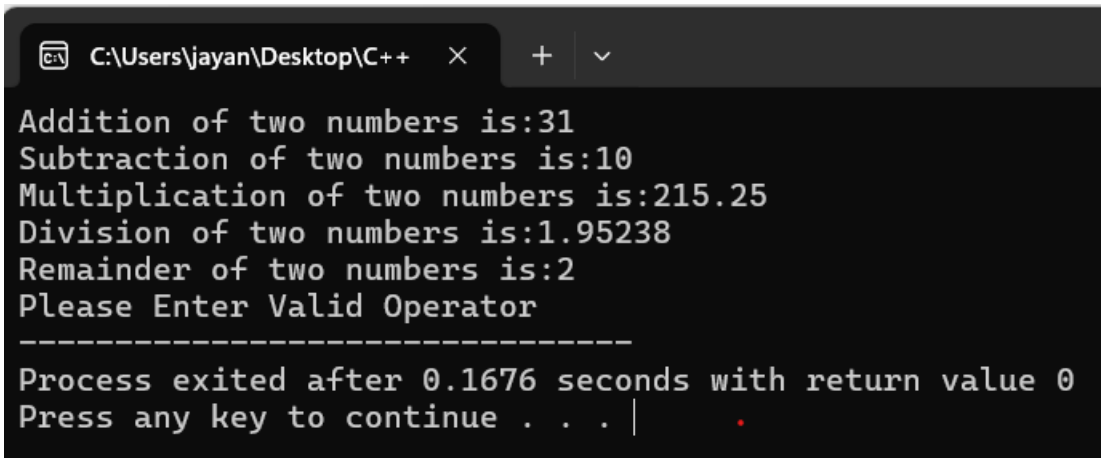

C:\Users\jayan\Desktop\C++ and DSA\user_defined_header_file.h - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
[*] Basic Structure of C++.cpp
[*] user_defined_header_file.h
25 break;
26
27 case'/':
28 div=num1/num2;
29 cout<<"Division of two numbers is:"<<div<<endl;
30 break;
31 case'%':
32 mod=int(num1)%int(num2);
33 cout<<"Remainder of two numbers is:"<<mod<<endl;
34 break;
35
36 default:
37 cout<<"Please Enter Valid Operator";
38 break;
39 }
40 }
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\jayan\Desktop\C++ and DSA\Basic Structure of C++.exe
- Output Size: 1.83316993713379 MiB
- Compilation Time: 0.36s
Line: 26 Col: 13 Sel: 0 Lines: 40 Length: 806 Insert Done parsing in 0.016 seconds


Warning
Source file not compiled
OK

Source code of Main file



Output:


C:\Users\jayan\Desktop\C++
Addition of two numbers is:31
Subtraction of two numbers is:10
Multiplication of two numbers is:215.25
Division of two numbers is:1.95238
Remainder of two numbers is:2
Please Enter Valid Operator

Process exited after 0.1676 seconds with return value 0
Press any key to continue . . .

Functions and its types

(i)A function is a group of statement that together perform a task. Every C++ program atleast one function, which is main(), and all the most trivial program can define additional functions.

(ii)You can divide up your code into separate functions. How can you divide up your code among different functions is up to you, but logically the division is much that each function perform a specific task.

(iii)A function declaration tells the compiler about a function's name, return type and parameters. A function definition provides the actual body of function.

(iv)The C++ standard library provides numerous build-in functions that your program can call.

Ex: cout<< for output, strcat() for string concatenation, etc.

(v)A function can also be referred as a method or a sub-routine or a procedure etc.

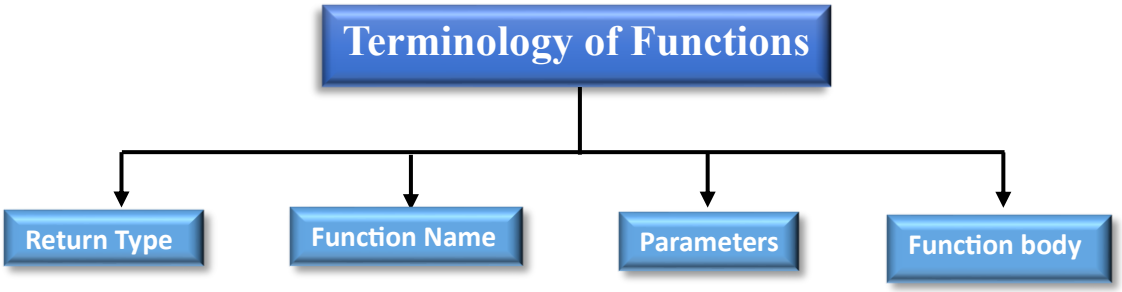
Syntax:

return-type function name(parameter list)

{ //Opening curly bracket

//body of code

} //Closing curly bracket

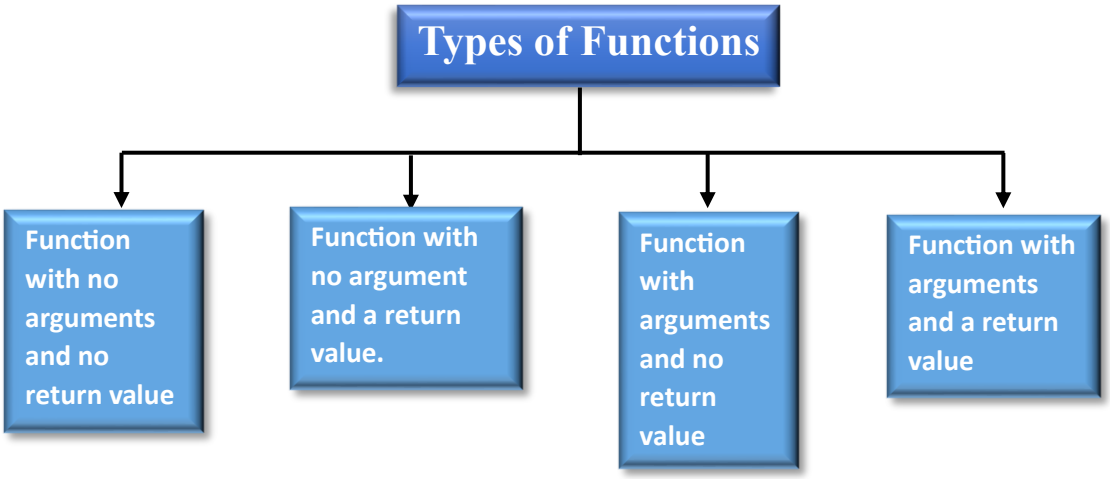


Return Type: The return type is the data type of the value the function return. Some functions perform the desired operations without returning a value. In the case, the return type is the keyword void.

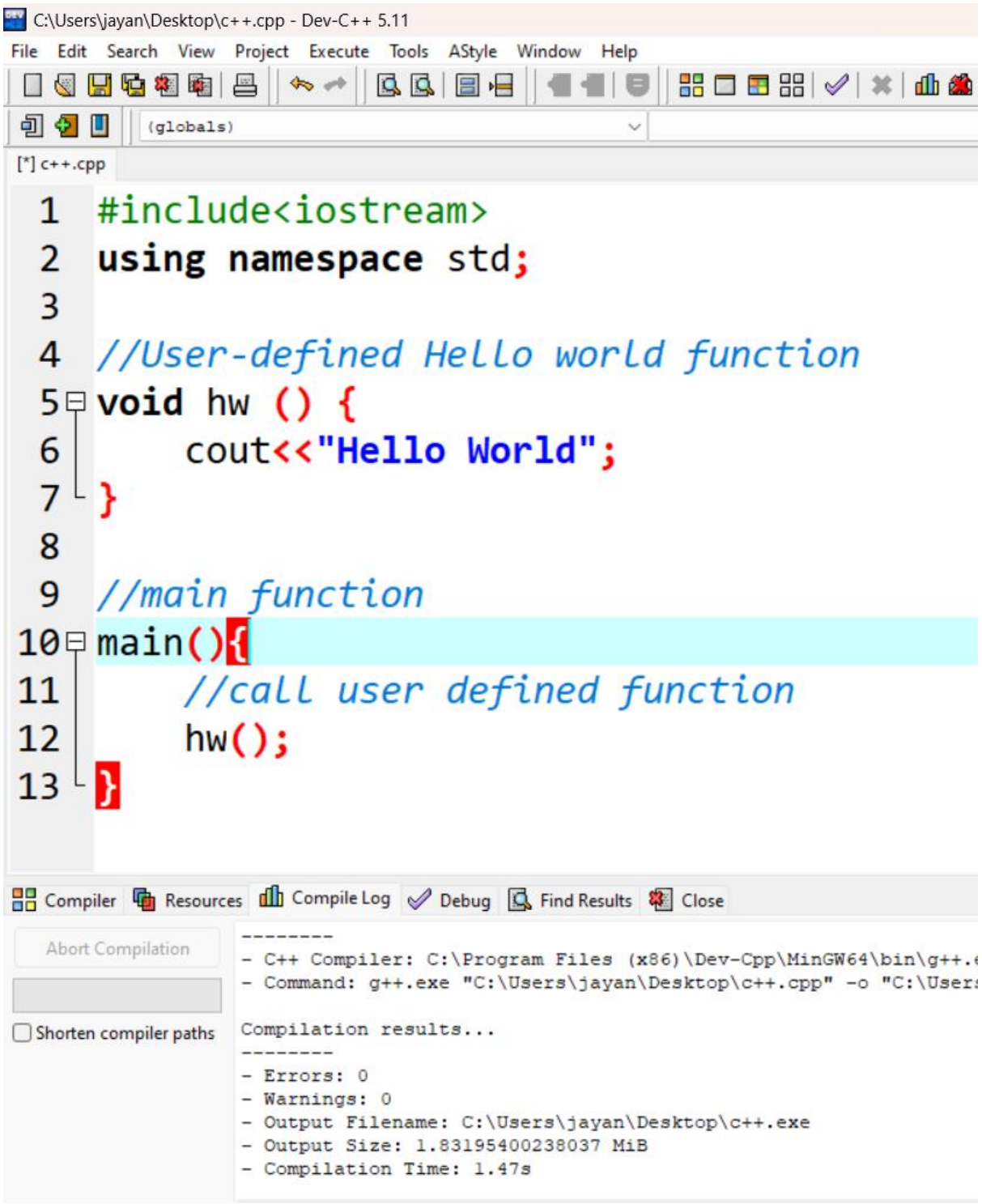
Function Name: This is the actual name of the function. The function name and the parameter list together constitute the function signature.

Parameters: A parameter is like a place holder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional that is a function may contain no parameters.

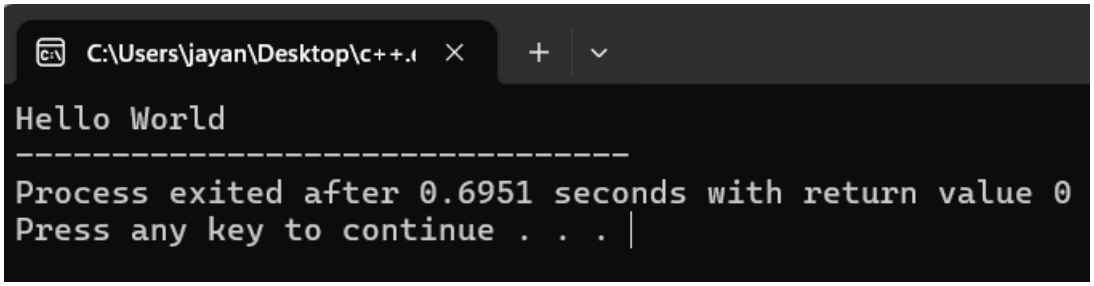
Function Body: The function body contains a collection of statements that define what the function does.



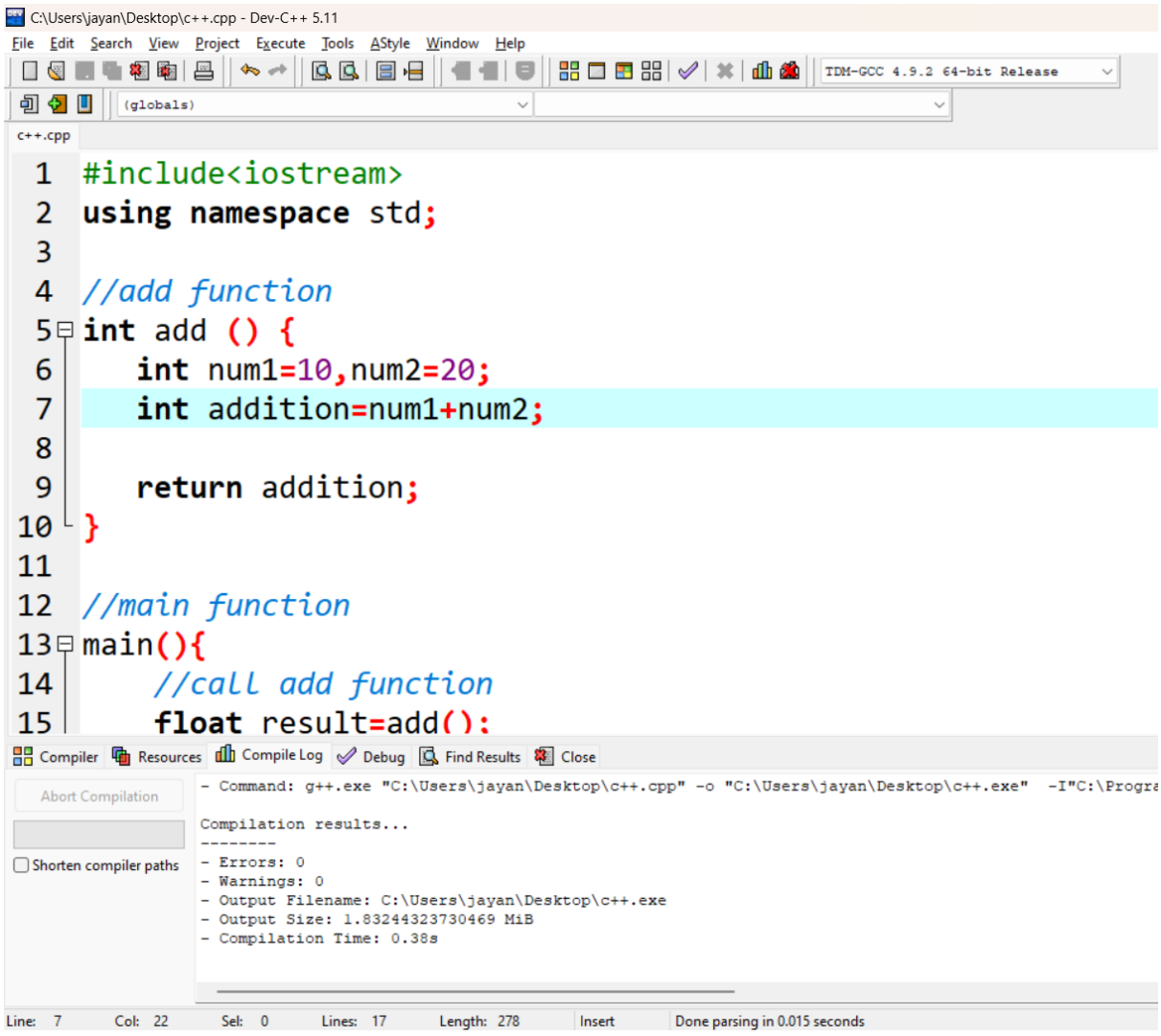
1.Function with no arguments and no return value:



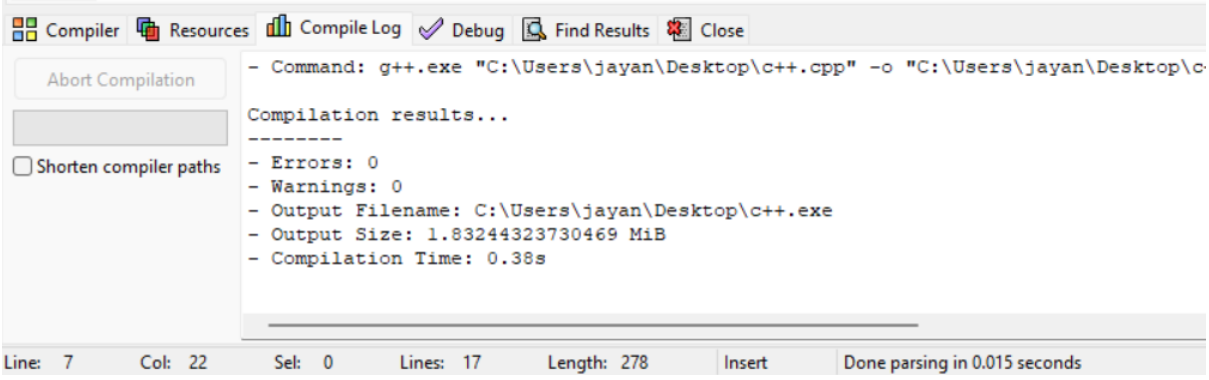
Output:



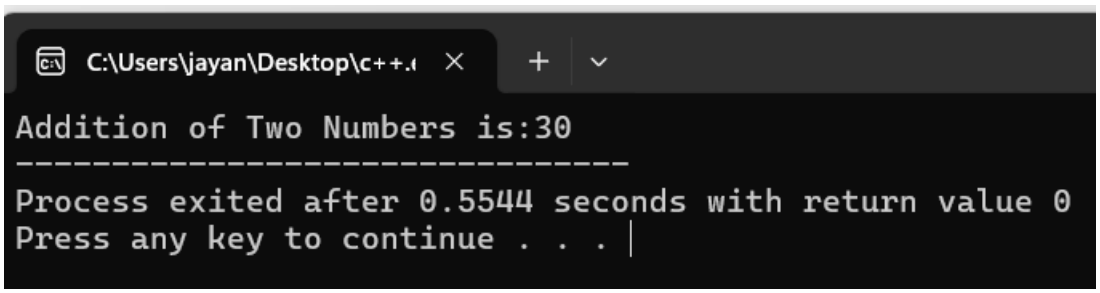
2.Function with no arguments and a return value:



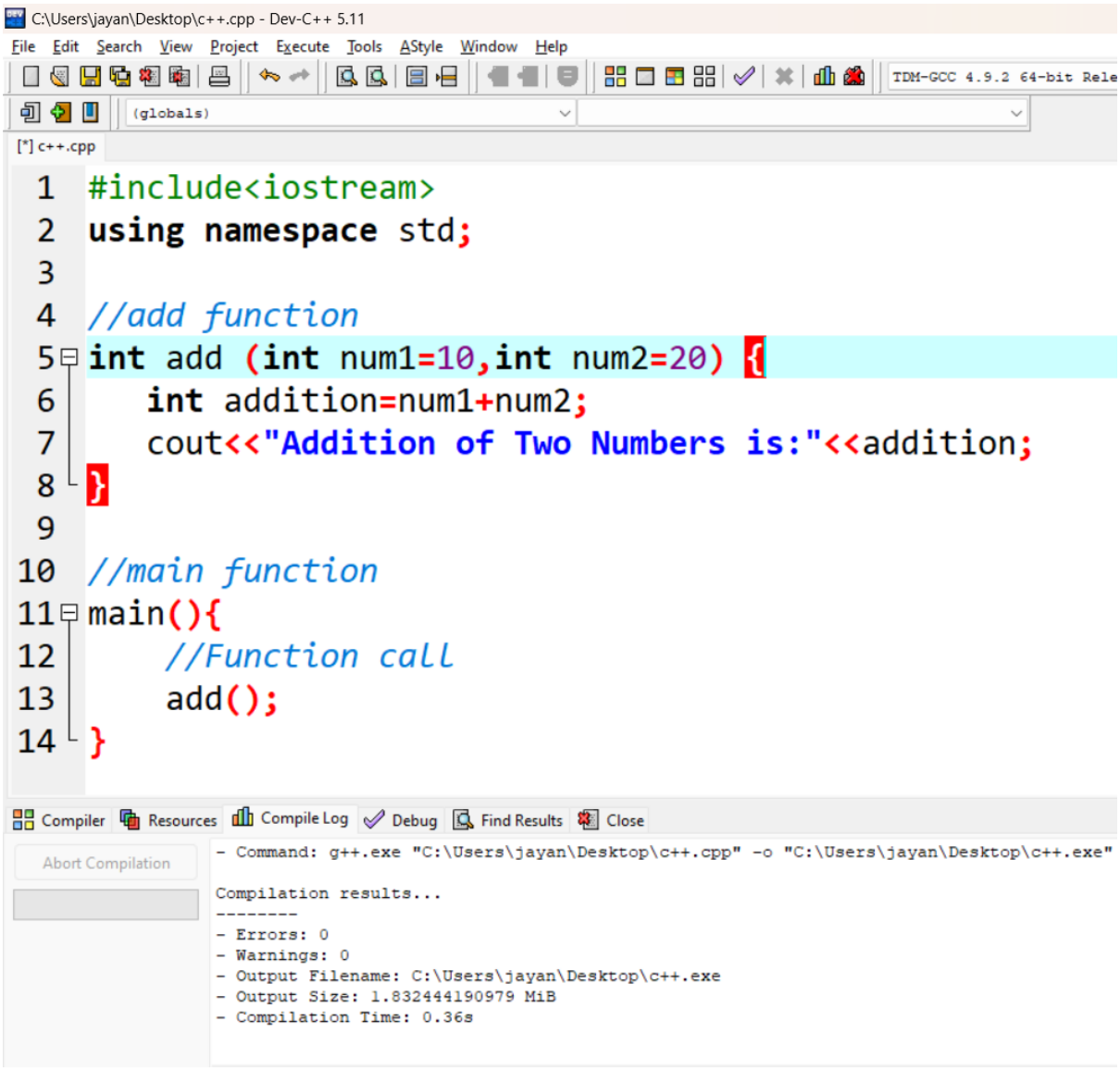
```
11
12  //main function
13  main(){
14      //call add function
15      float result=add();
16      cout<<"Addition of two numbers is:"<<result;
17  }
```



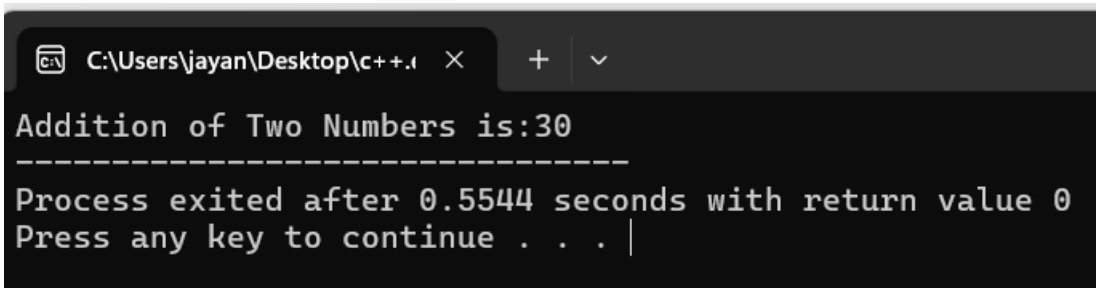
Output:



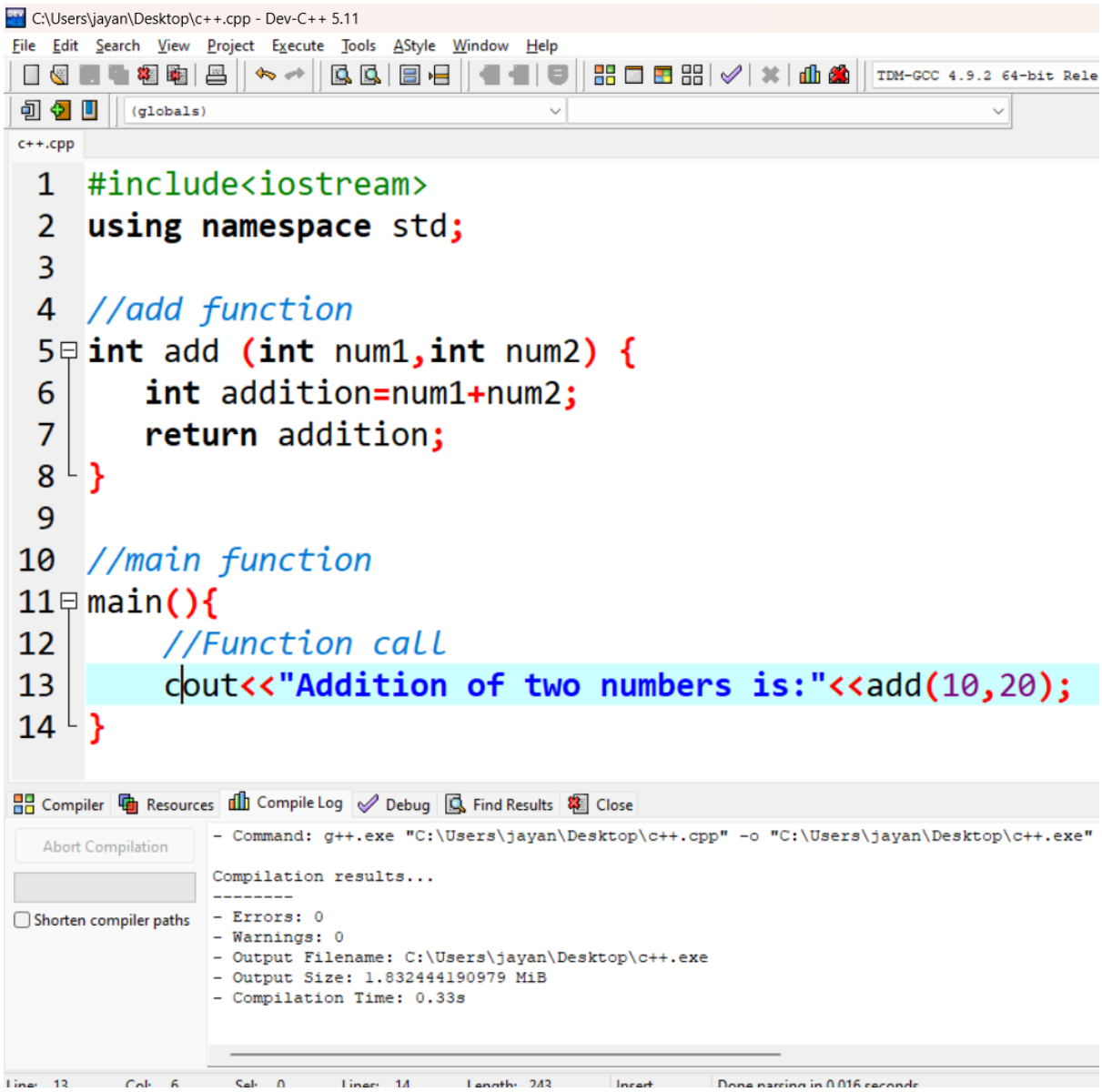
3.Function with arguments and no return value:



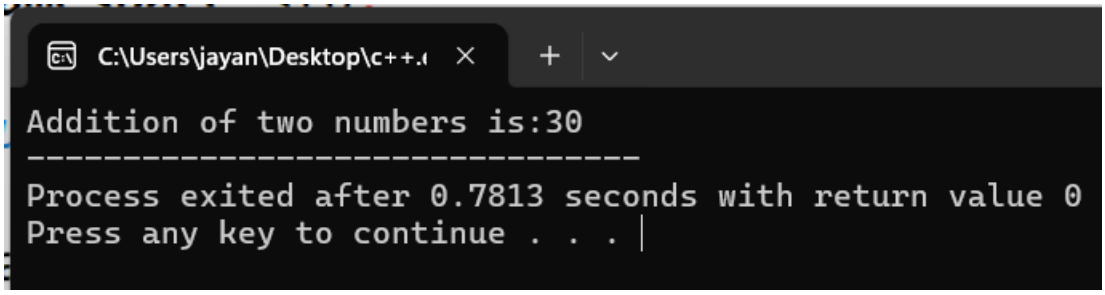
Output:



4.Function with arguments and a return value:


C:\Users\jayan\Desktop\c++.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Rele
(globals)
c++.cpp
#include<iostream>
using namespace std;
//add function
int add (int num1,int num2) {
int addition=num1+num2;
return addition;
}
//main function
main(){
//Function call
cout<<"Addition of two numbers is:"<<add(10,20);
}
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
- Command: g++.exe "C:\Users\jayan\Desktop\c++.cpp" -o "C:\Users\jayan\Desktop\c++.exe"
Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\jayan\Desktop\c++.exe
- Output Size: 1.832444190979 MiB
- Compilation Time: 0.33s

Output:

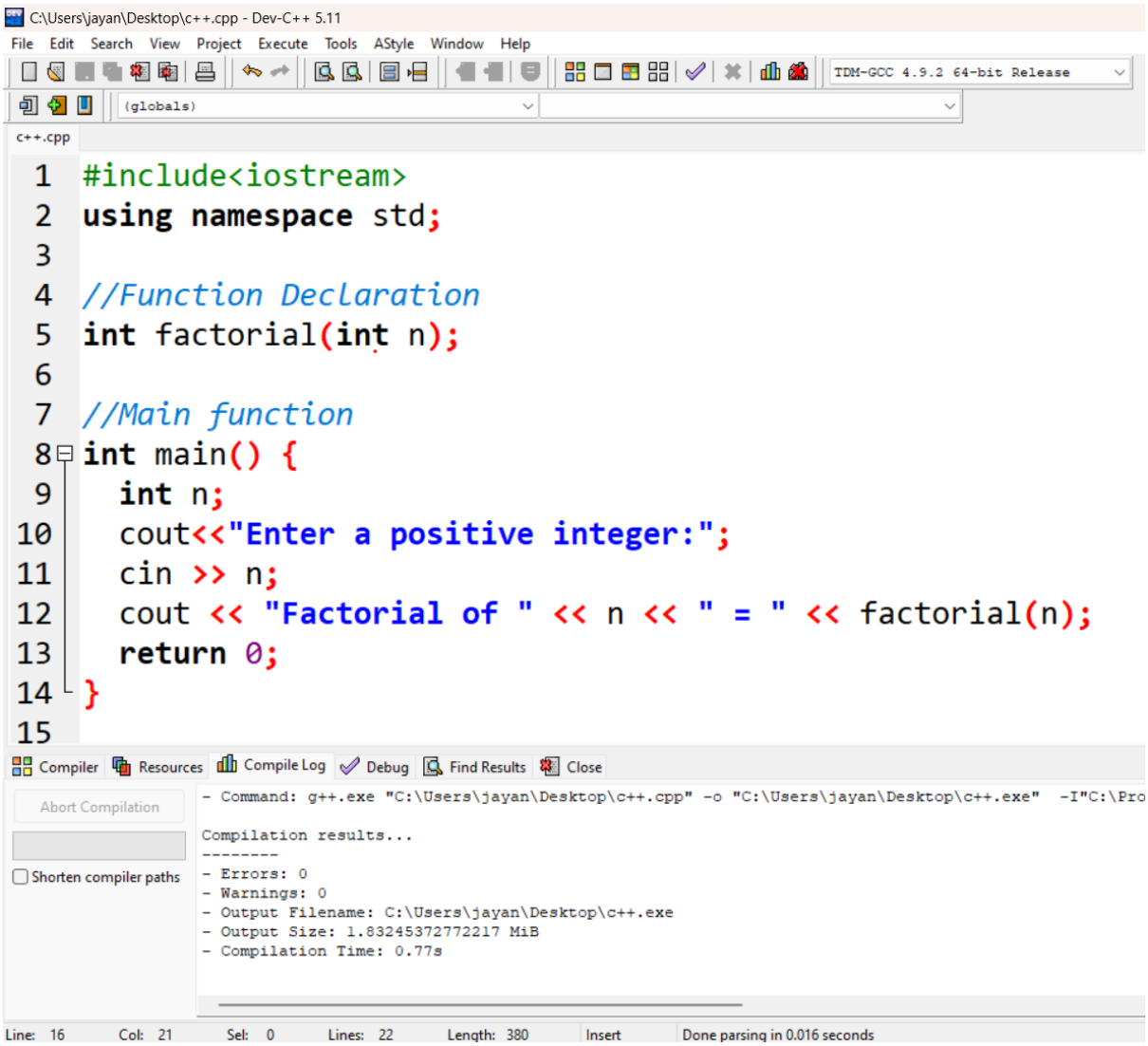

C:\Users\jayan\Desktop\c++.
Addition of two numbers is:30
Process exited after 0.7813 seconds with return value 0
Press any key to continue . . .

Recursion

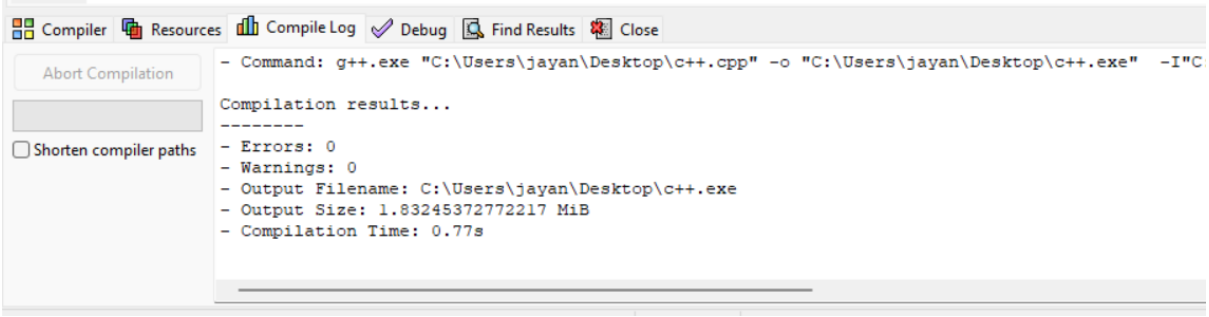
(i)Recursion is the process which comes into existence when a function call a copy of itself to work on a smaller problem. Any function which calls itself is called recursive function; it is important to impose a termination condition of recursion. Recursion code is shorter than iterative code.

(ii)Recursion cannot be applied to all the problem, but it is more useful for the tasks that can be defined in terms of similar subtasks.

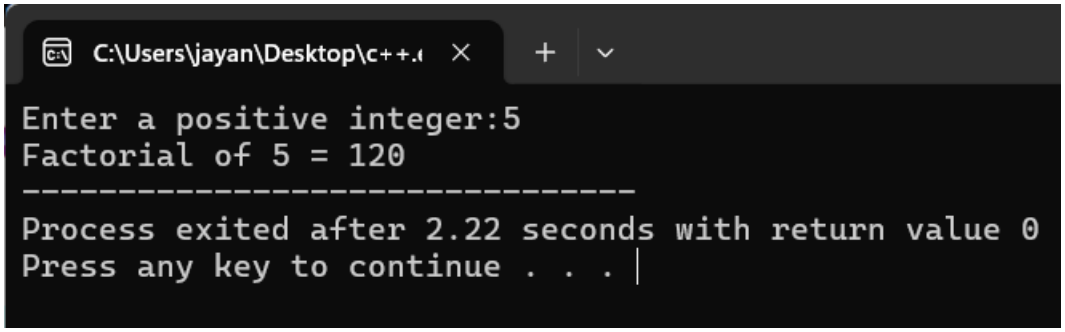
Example: Factorial, sorting or searching etc.



```
12    cout << "Factorial of " << n << " = " << factorial(n);
13    return 0;
14  }
15
16  //Factorial function
17  int factorial(int n) {
18    if(n > 1)
19      return n * factorial(n - 1);
20    else
21      return 1;
22  }
```



Output:



Function invoking

Functions can be invoked into two types (i)Call by Value (ii) Call by Reference. These two ways are generally differentiate by the type of values passed to them as parameters.

Actual Parameters: The parameters passed to function are called Actual parameters.

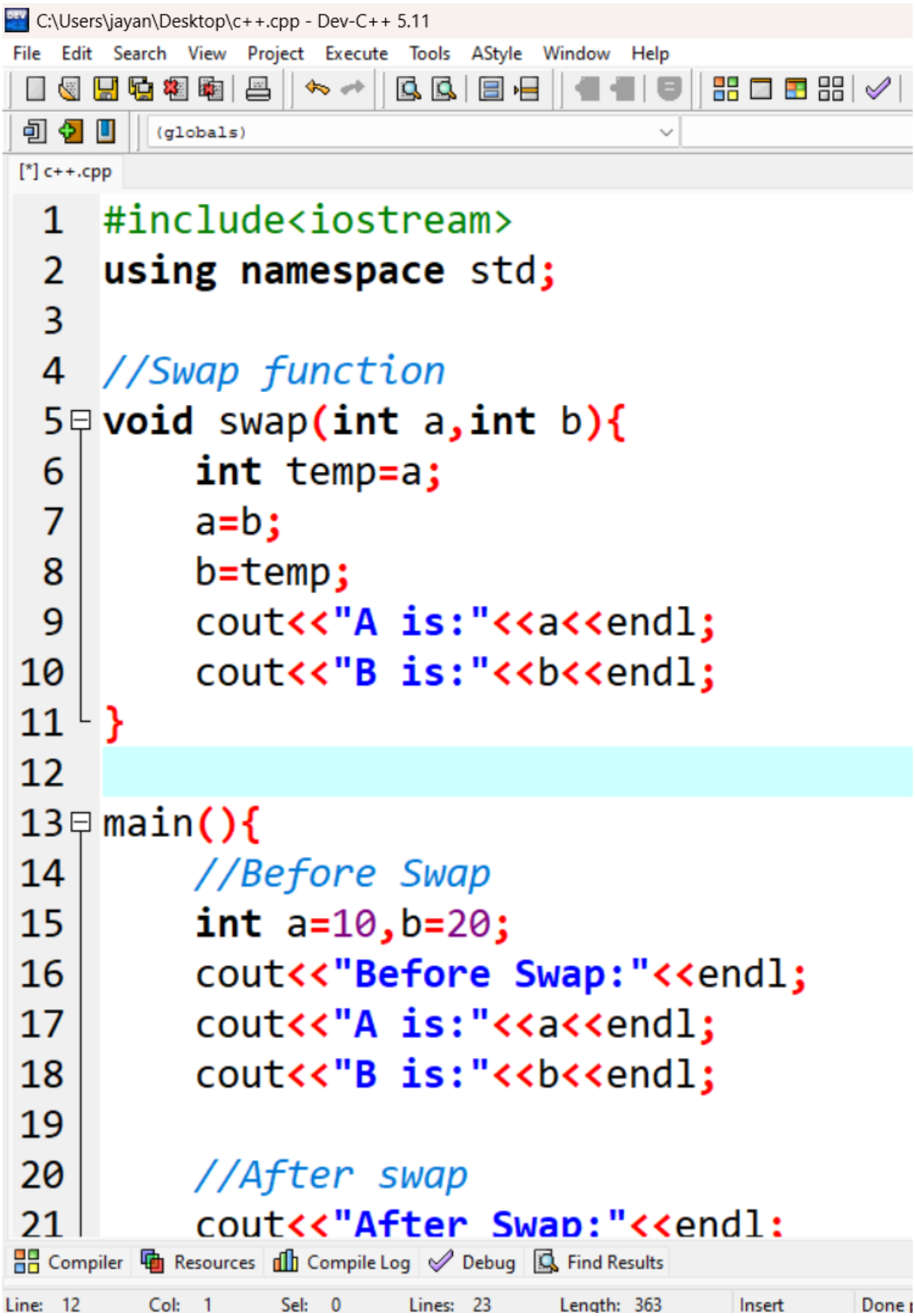
Formal Parameters: The parameters received by function are called formal parameters.

Call by Value:(i) In call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that value of the variable is used in the function call in the call by value method.

(ii)In call by method, we can not modify the value of the actual parameter by the formal parameter.

(iii)Different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.

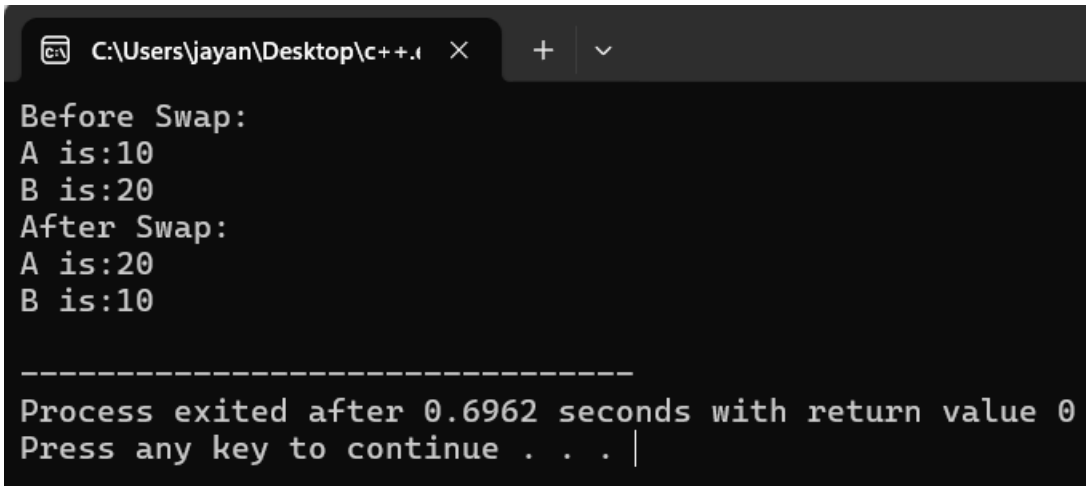
(iv)The actual parameter is the argument which is used in the function call where as formal parameter is the argument which is used in the function definition.


C:\Users\jayan\Desktop\c++.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
[*] c++.cpp
#include<iostream>
using namespace std;
//Swap function
void swap(int a,int b){
int temp=a;
a=b;
b=temp;
cout<<"A is:"<<a<<endl;
cout<<"B is:"<<b<<endl;
}
main(){
//Before Swap
int a=10,b=20;
cout<<"Before Swap:"<<endl;
cout<<"A is:"<<a<<endl;
cout<<"B is:"<<b<<endl;
//After swap
cout<<"After Swap:"<<endl;
Compiler Resources Compile Log Debug Find Results
Line: 12 Col: 1 Sel: 0 Lines: 23 Length: 363 Insert

```
19
20      //After swap
21      cout<<"After Swap:"<<endl;
22      swap(10,20);
23  }
```

1/ 9 void swap (int *a, int *b)

Output:

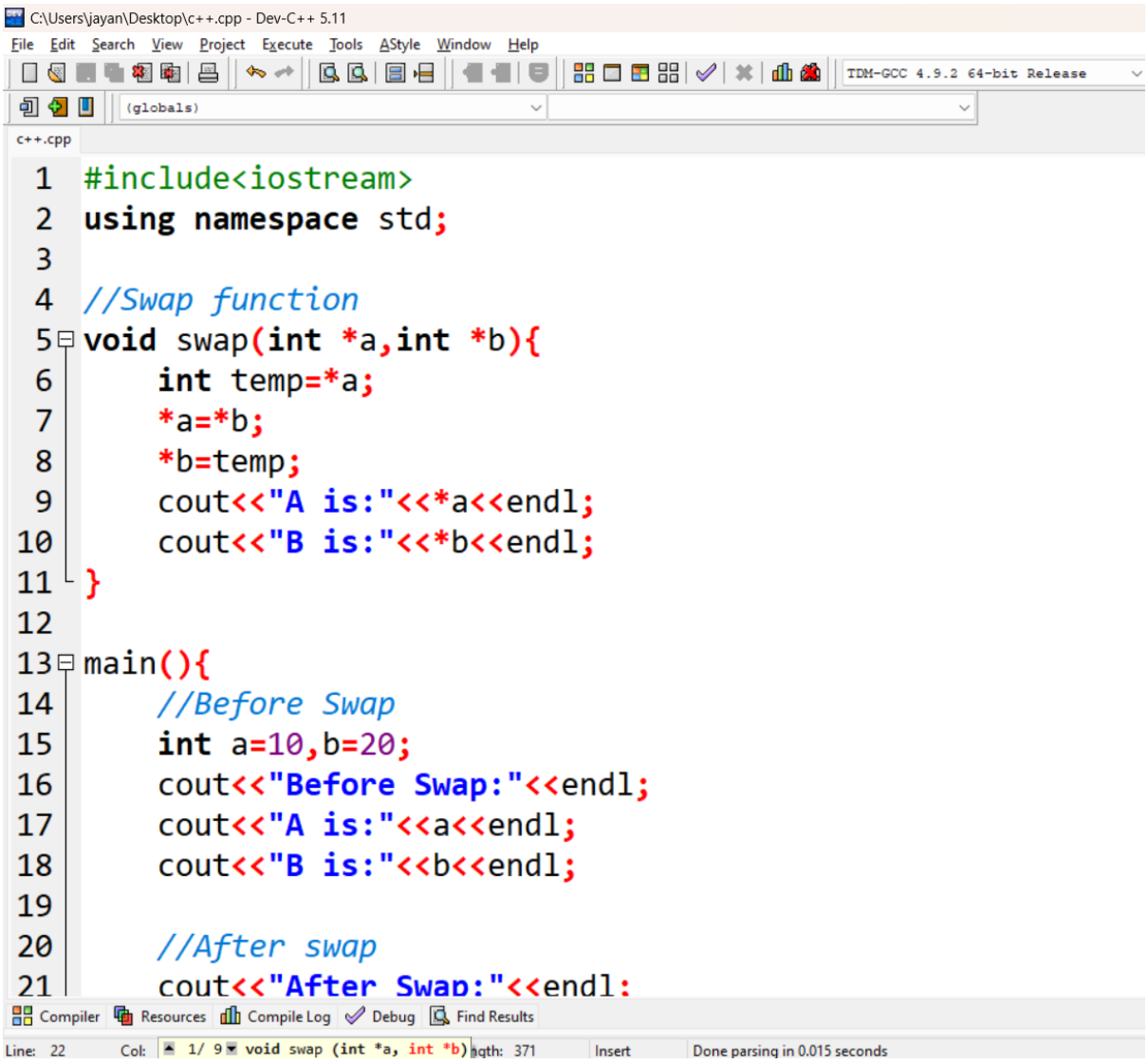


Call by Reference:(i) In call by reference, the address of the variable is passed into the function call as the actual parameter.

(ii)The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.

(iii)In call by reference, the memory allocation is similar for both the formal parameters and actual parameters.

All operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.



```
18      cout<<"B is:"<<b<<endl;
19
20      //After swap
21      cout<<"After Swap:"<<endl;
22      swap(&a,&b);
23  }
```

1/ 9 void swap (int *a, int *b)

Compiler | Resources | Compile Log | Debug | Find Results

Output:

```
C:\Users\jayan\Desktop\c++.(
Before Swap:
A is:10
B is:20
After Swap:
A is:20
B is:10

--------------------------------
Process exited after 0.6962 seconds with return value 0
Press any key to continue . . .
```

Special Symbols

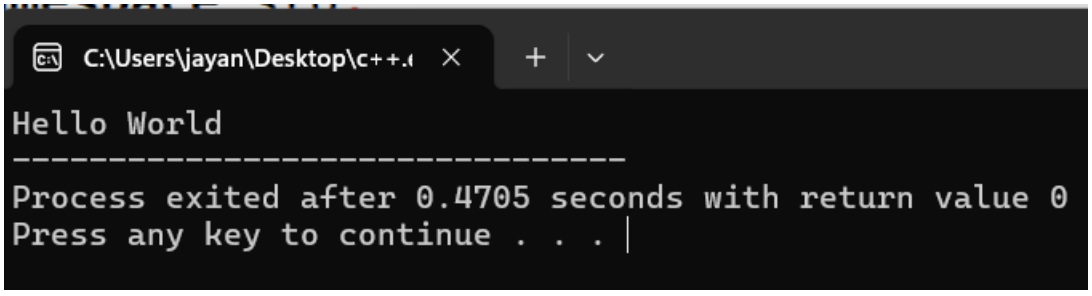
C++ support a rich set of special symbols that include symbols to perform mathematical operations, to check condition, whitespaces, backspaces, or other special symbols.

Ex: #, @, %, ( ), { }, etc

[*] c++.cpp

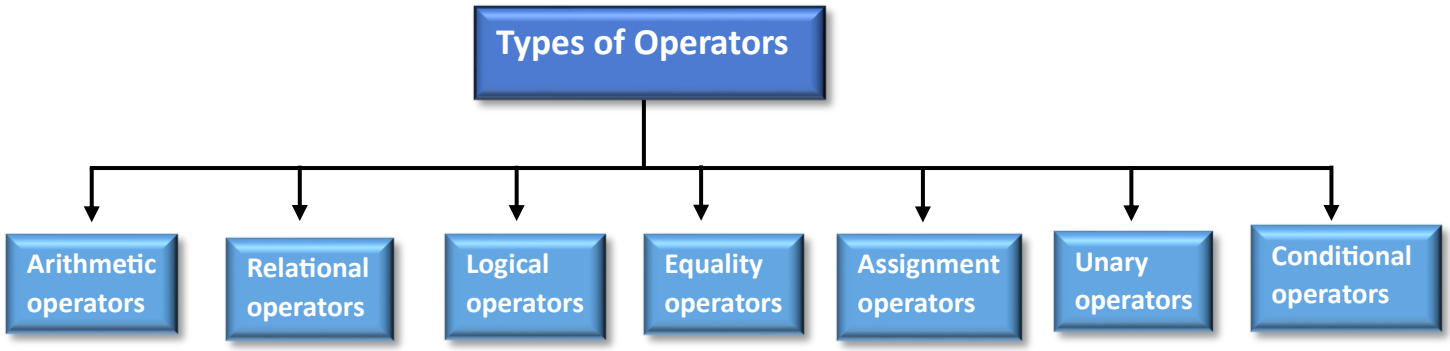
```cpp
#include<iostream>
using namespace std;
//main function
//Special symbols:(),{},[],!,?,~,...
main() {
    cout<<"Hello World";
}
```

Output:



Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C++ language is rich in build in operators.



Arithmetic Operators:

An arithmetic operators are performs mathematical operations such as addition, subtraction, multiplication or division etc.

Operators	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo(Remainder)

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

//main function
main() {
    //Arithmetic operators
    int num1=10,num2=20;//Initialise the values
    //Perform Addition,Subtraction,Multiplication,Division or Modulo(Remainder)
    int add=num1+num2;
    int sub=num1+num2;
    int multi=num1*num2;
    float div=float(num1)/float(num2);
    //Output Functions
    cout<<"Addition is:"<<add<<endl;
    cout<<"Subtraction is:"<<sub<<endl;
    cout<<"Multiplication is:"<<multi<<endl;
    cout<<"Division is"<<div;
}
```

Output:

```
C:\Users\jayan\Desktop\c++.
Addition is:30
Subtraction is:30
Multiplication is:200
Division is0.5
--------------------------------
Process exited after 1.368 seconds with return value 0
Press any key to continue . . .
```

Relational Operator:

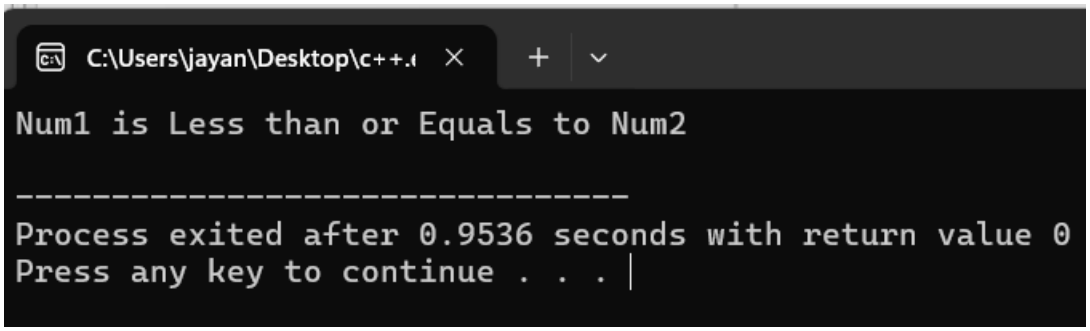
A relational operator checks the relationship between two operands. If the relation is true, it return 1; if the relation is false, it return 0.

Relational operator are used for decision making and loops.

Operators	Description
>	Greater Than
<	Less Than
>=	Greater than equals to
<=	Less Than equals to
==	Equals to
!=	Not Equals to

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

//main function
main() {
    //Relational operators
    int num1=10,num2=20;//Initialise the values
    //Conditions
    if(num1 > num2){
        cout<<"Num1 is Greater than Num2"<<endl;
    }
    if(num1 <= num2){
        cout<<"Num1 is Less than or Equals to Num2"<<endl;
    }
}
```

Output:


C:\Users\jayan\Desktop\c++.
Num1 is Less than or Equals to Num2

Process exited after 0.9536 seconds with return value 0
Press any key to continue . . .

Logical operator:

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operator are commonly used in decision making in C++ programming.

Operators	Description
&&	Logical And
||	Logical OR
!	Logical Not

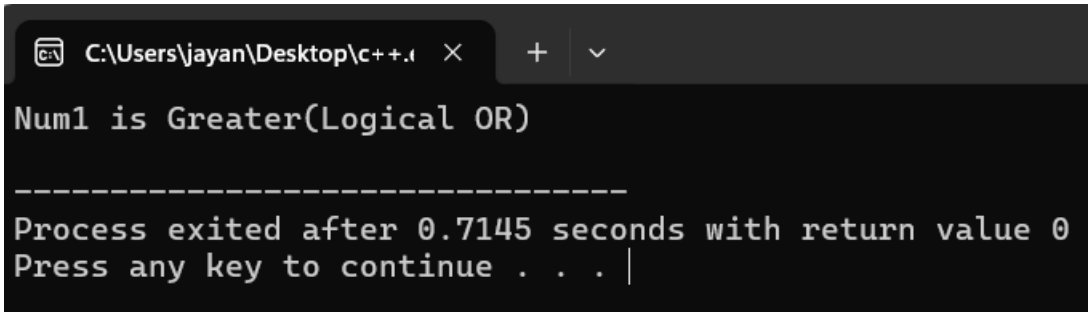
Note: (i) !(!=5):If Not Equals to is in brackets and not operator are also in outside the brackets, so situation will be opposite(Not operator cannot be work).

(!=5) : 0

!(!=5) : 1

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

//main function
main() {
    //Logical operators
    int num1=10,num2=20,num3=5;//Initialise the values
    //Conditions
    if(num1>num2 && num1>num3){
        cout<<"Num1 is Greater(Logical AND)"<<endl;
    }
    if(num1>num2 || num1>num3){
        cout<<"Num1 is Greater(Logical OR)"<<endl;
    }
}
```

Output:


C:\Users\jayan\Desktop\c++.
Num1 is Greater(Logical OR)

Process exited after 0.7145 seconds with return value 0
Press any key to continue . . .

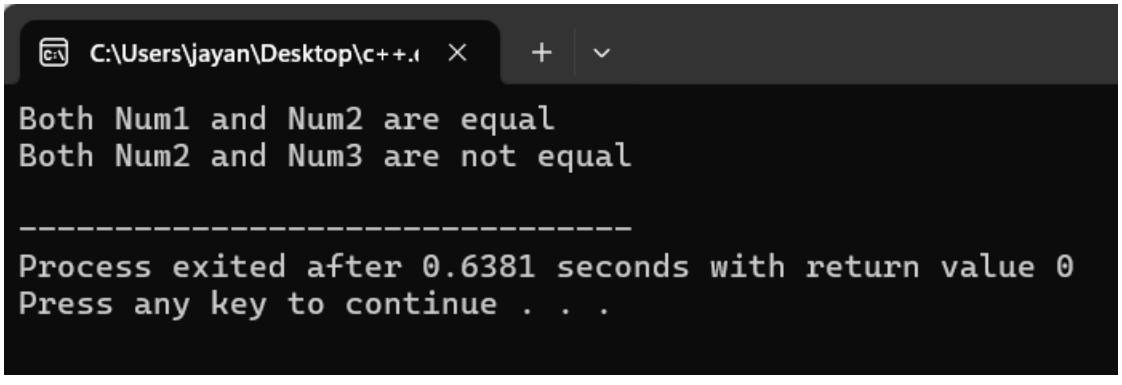
Equality operators:

(i)(==):Equals To is used to compare two values or expressions. The result is true if the expression are equal or otherwise false.

(ii)(!=):Not Equals To returns true if the operands doesn't have the same value; otherwise, it returns false.

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

//main function
main() {
    //Logical operators
    int num1=10,num2=10,num3=5;//Initialise the values
    //Conditions
    if(num1==num2){
        cout<<"Both Num1 and Num2 are equal"<<endl;
    }
    if(num2!=num3){
        cout<<"Both Num2 and Num3 are not equal"<<endl;
    }
}
```

Output:



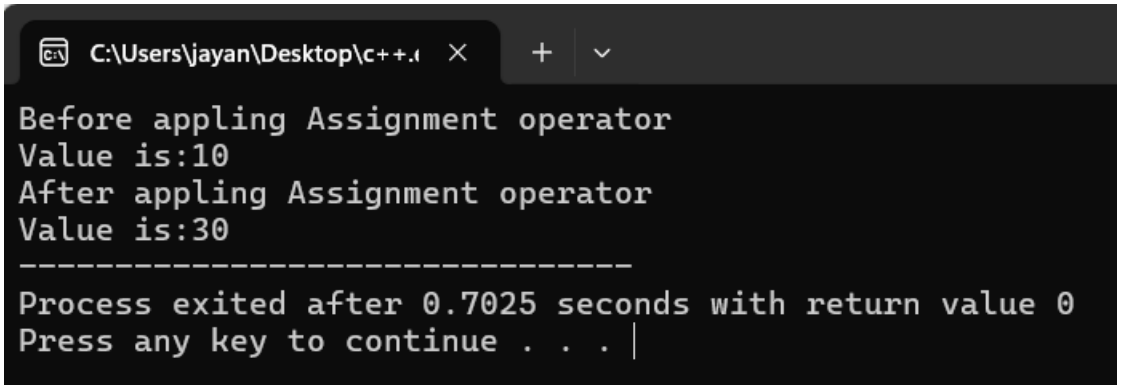
Assignment operators:

An assignment operator is used for assigning a value to a variable. The most common operator is (=).

Operators	Description
+=	Addition or Equal To
-=	Subtraction or Equal To
*=	Multiplication or Equal To
/=	Division or Equal To
%=	Modulo or Equal To

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

//main function
main() {
    //Assignment operators
    int num1=10;//Initialise the values
    //Before appling Assignment operator
    cout<<"Before appling Assignment operator"<<endl;
    cout<<"Value is:"<<num1<<endl;
    //After appling Assignment operator
    num1 += 20;
    cout<<"After appling Assignment operator"<<endl;
    cout<<"Value is:"<<num1;

}
```

Output:



Unary Minus (--):

(i)The minus operator changes the sign of its argument. A positive number becomes negative and a negative number becomes positive.

(ii)Unary minus is different from subtraction operator, as subtraction requires two operands.

(iii) (--) = --a or a--

(iv).It is used to decrement the value of the variable one.

(i)Prefix decrement:

In this method, the operator proceeds the operand (eg: --a). This value of operator will be altered before it is used.

(ii)Postfix decrement:

In this method, the operator follows the operand (eg: a--). The value of operator will altered after it is used.

Increment (++):

It is used the value of the variable by 1.

(i)Prefix increment: In this method, the operator proceed the operand (++a). The value of operand will be altered before, it is used.

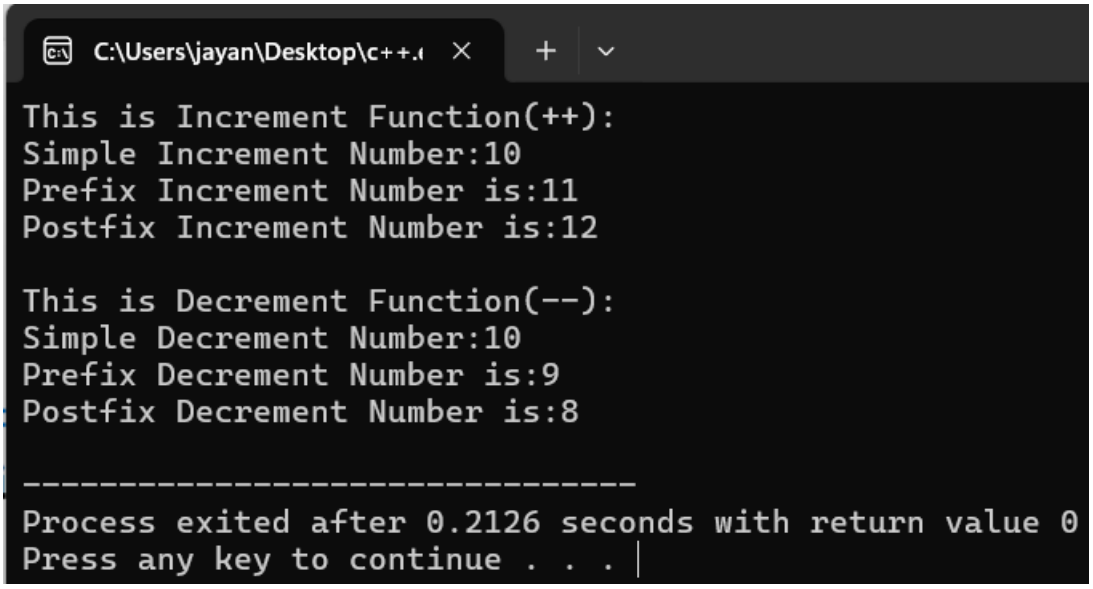
(ii)Postfix increment: In this method, the operator follows the operands(eg. a++). The value operand will be altered after it is used.

c++.cpp

```cpp
#include<iostream>
using namespace std;
//Function for(++)
void increment(){
    int num1=10;
    cout<<endl<<"Simple Increment Number:"<<num1<<endl;
    ++num1;//num1=value+1
    cout<<"Prefix Increment Number is:"<<num1<<endl;
    num1++;//num1=value+1
    cout<<"Postfix Increment Number is:"<<num1<<endl;
}
//Function for(--)
void decrement(){
    int num2=10;
    cout<<endl<<"Simple Decrement Number:"<<num2<<endl;
    --num2;//num2=value-1
    cout<<"Prefix Decrement Number is:"<<num2<<endl;
    num2--;//num2=value-1
    cout<<"Postfix Decrement Number is:"<<num2<<endl;
}
//Main Function
main(){
    cout<<"This is Increment Function(++):";
    increment();
```

```
21  //Main Function
22  main(){
23      cout<<"This is Increment Function(++):";
24      increment();
25      cout<<endl;
26      cout<<"This is Decrement Function(--):";
27      decrement();
28  }
```

Output:



Conditional operator

(i)Conditional Operator is known as ternary operator

(ii)It is shortest form of writing conditional statements. It can be used as an inline conditional statement in place of if-else to execute some conditional code.

(iii)(? or :) these symbols are used for conditional operator.

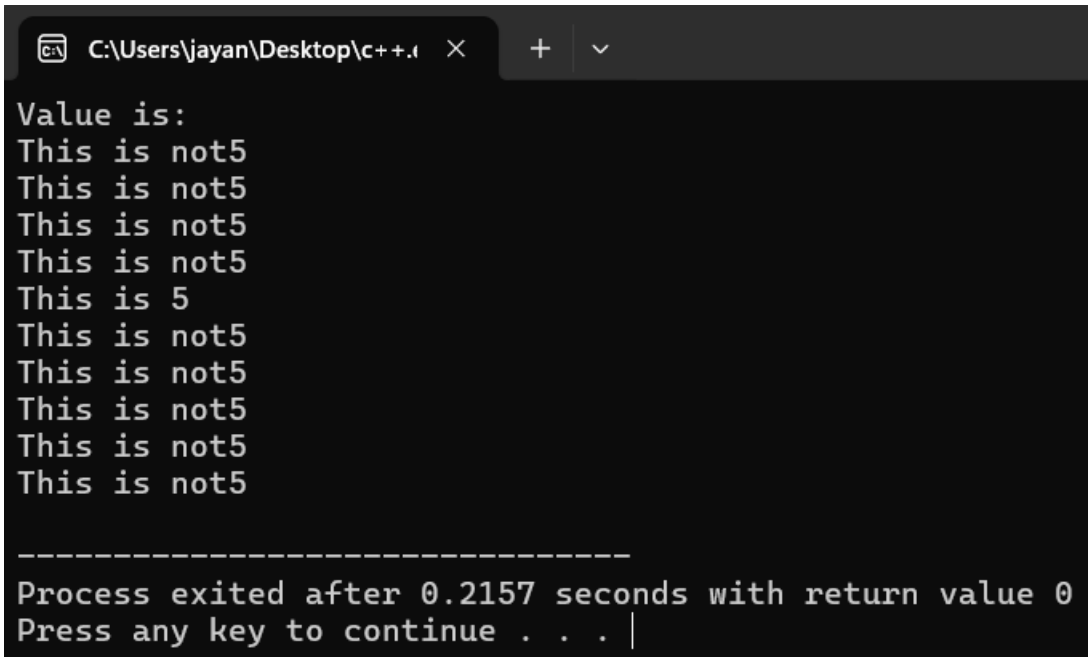
Syntax:

Datatype variableName=Expression ? statement1:statement2;

c++.cpp

```cpp
#include<iostream>
using namespace std;
//Function for Conditional Operator
void conditional_operator(){
    cout<<"Value is:"<<endl;
    for(int num=1;num<=10;num++){
        //Conditional operator
        string output=num==5?"This is 5":"This is not5";
        cout<<output<<endl;
    }
}
main(){
    conditional_operator();
}
```

Output:



Bitwise Operator:

To perform bit level operation in C++ programming, bitwise operator are used.

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise Complement
<<	Bitwise left shift
>>	Bitwise right shift

Strings

(i)The string can be defined as the one-dimensional array of characters terminated by a null (“\0”).

(ii)The character array or the string is used to manipulate text such as word or sentences.

(iii)Each character in the array occupies one byte of memory, and the last character must always be 0.

(iv)The termination character (“\0”) is important in a string since it is only way to identify where the string ends. When we defined a string as char s[10],the character s[s] is implicitly initialized with the null in the memory.

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

//main function
main() {

char s[20];
cout<<"Enter String:";
cin>>s;
cout<<"String is:"<<s;

}
```

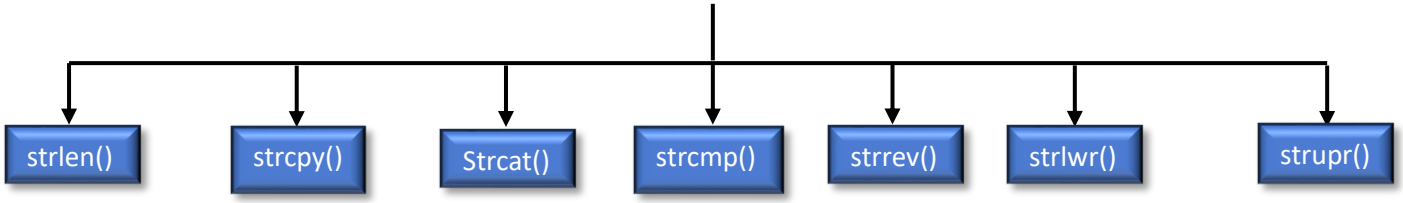
Output:

```
C:\Users\jayan\Desktop\c++.
Enter String:Hello
String is:Hello
--------------------------------
Process exited after 4.589 seconds with return value 0
Press any key to continue . . .
```

String Functions

In C++ we will see how to compare two strings, Concatenate, strings, copy one string to another & perform various string manipulation operations. We can perform such operations using the pre-defined functions of "string.h" header file. In order to use these string functions, you must include string.h file in our c++ program.

Types of String Functions



strlen(): The strlen() function return the length of the given string. It doesn't count null character '\0'.

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

//main function
main() {
char alphabet[10]="Hello";
int len=strlen(alphabet);
cout<<"Length of string is:"<<len;
}
```

Output:

```
C:\Users\jayan\Desktop\c++.
Length of string is:5
--------------------------------
Process exited after 0.76 seconds with return value 0
Press any key to continue . . .
```

strcpy(): Copies the contents of source string to destination string.

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

//main function
main() {
char alphabets1[10]="Hello";
char alphabets2[10];
cout<<"The value of Second string before apply string function:"<<alphabets2<<endl;
strcpy(alphabets2,alphabets1);
cout<<"The value of Second string after apply string function:"<<alphabets2;
}
```

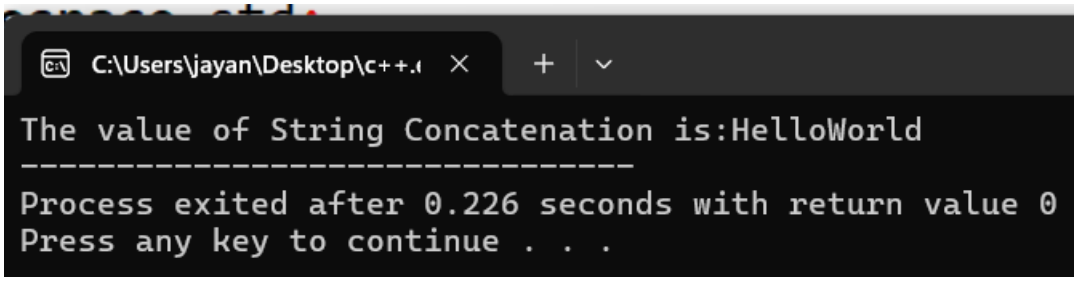
Output:

```
The value of Second string before apply string function:
The value of Second string after apply string function:Hello
--------------------------------
Process exited after 0.1725 seconds with return value 0
Press any key to continue . . .
```

strcat(): Concatenation or join first string with second string. The result of the string is stored in first string.

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

//main function
main() {
char alphabets1[10]="Hello";
char alphabets2[10]="World";
cout<<"The value of String Concatenation is:"<<strcat(alphabets1,alphabets2);
}
```

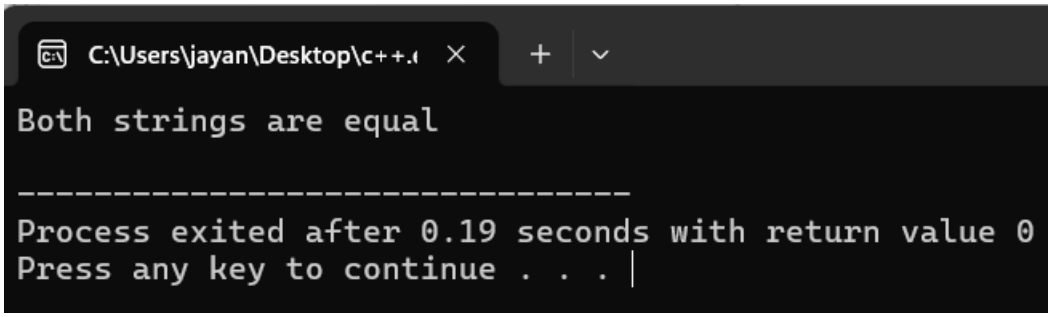
Output:



strcmp(): Compare the first string with second string.If both strings are same, it returns 0.

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

//main function
main() {
char alphabets1[10]="Hello";
char alphabets2[10]="Hello";
if(strcmp(alphabets1,alphabets2)==0)
cout<<"Both strings are equal"<<endl;
else
cout<<"Both strings are not equal";
}
```

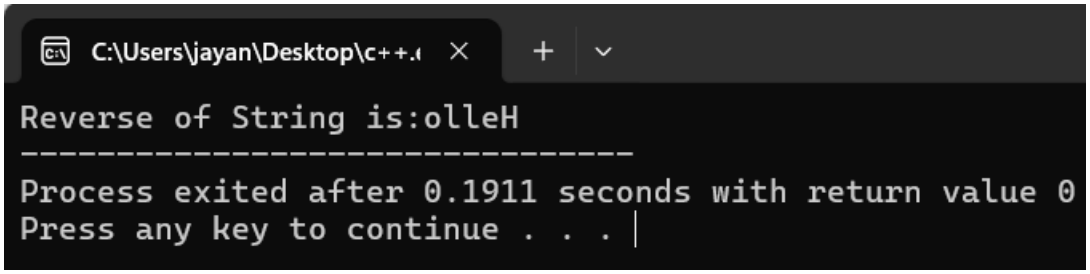
Output:



strrev(): It returns reverse of the string.

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

//main function
main() {
char alphabets1[10]="Hello";
cout<<"Reverse of String is:"<<strrev(alphabets1);
}
```

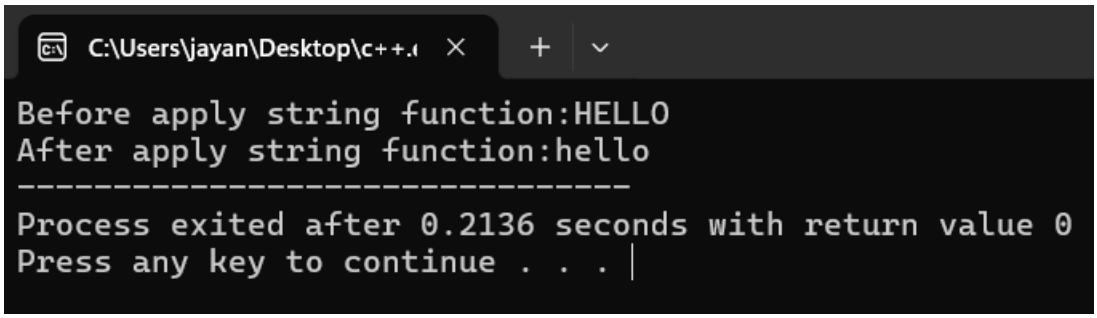
Output:



strlwr(): It returns string characters in lowercase.

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

//main function
main() {
char alphabets1[10]="HELLO";
cout<<"Before apply string function:"<<alphabets1<<endl;
cout<<"After apply string function:"<<strlwr(alphabets1);
}
```

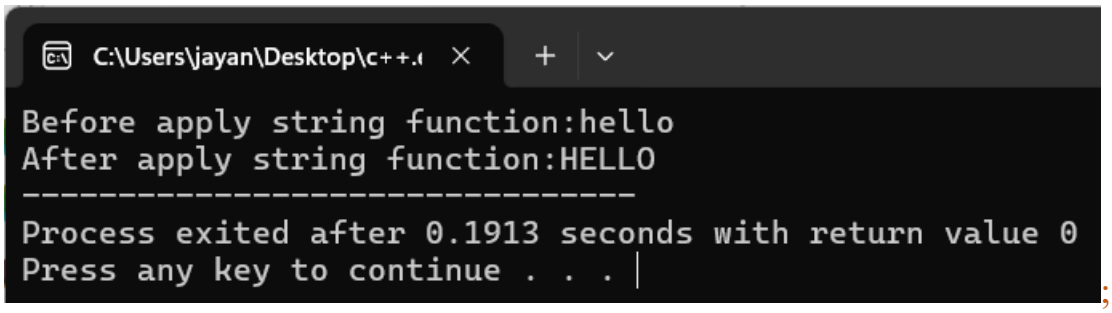
Output:



strupr(): It returns string characters in uppercase.

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

//main function
main() {
char alphabets1[10]="hello";
cout<<"Before apply string function:"<<alphabets1<<endl;
cout<<"After apply string function:"<<strupr(alphabets1);
}
```

Output:

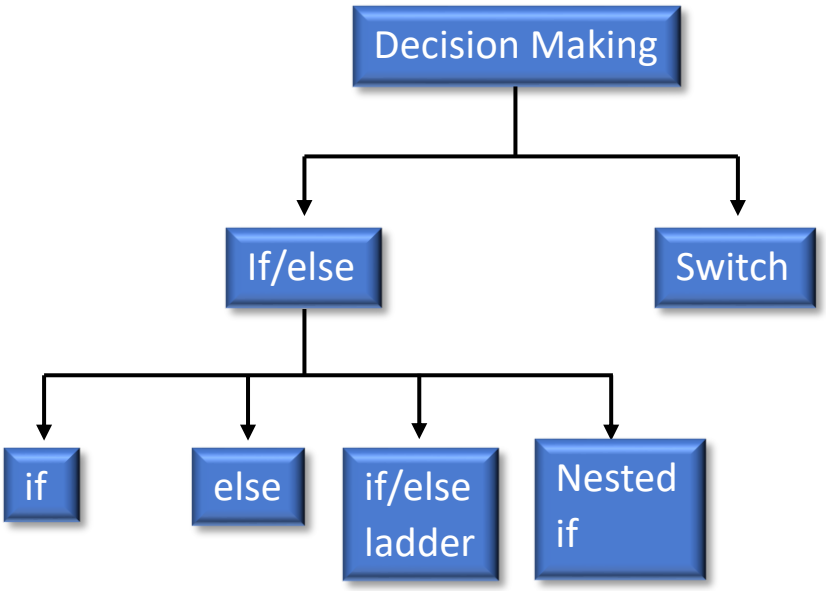


Conditional Statement (if/else or switch)

(i)There come situation in real life when we need to make some decision and based on these decisions, we decide that should we do next. Similar situation arise in programming also where we need to make some decision and based on these decision, we will execute the next block of code.

(ii) Decision making statements in programming languages or C++ decides the direction of flow of program execution.

(ii) It is also known as Decision Making statement.



If statement:

If statement is the most simple decision making statement.It is used to decide whether a certain statement or block of statement will be executed or not.

Syntax

If(condition){

Body of if statement

}

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

//main function
main() {
int num1=10,num2=10;
if(num1==num2)
cout<<"Num1 and Num2 are equal"<<endl;
}
```

Output:

```
C:\Users\jayan\Desktop\c++.
Num1 and Num2 are equal

--------------------------------
Process exited after 0.203 seconds with return value 0
Press any key to continue . . .
```

If/else statement:

The if statement alone tells us that if a condition is true it will be execute a block of statement and if the condition is false it won't. But if we want to do something else if the condition is false. Here comes the else statement. We can use the else statement with if statement to execute a block of code when the condition is false.

Syntax:

if(condition){

//body of if statement

}

else{

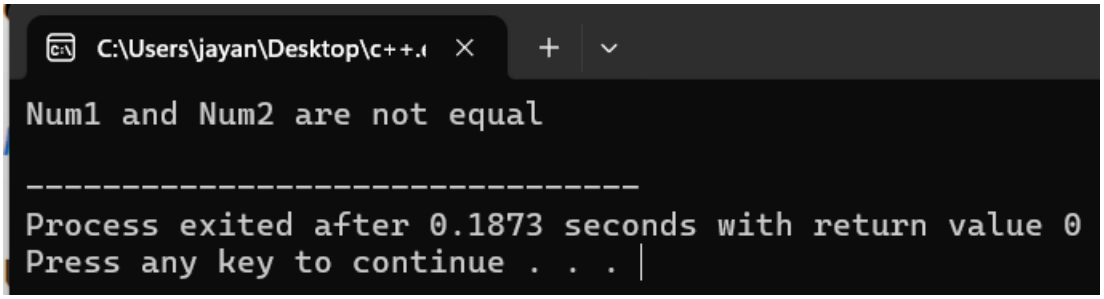
Body of else statement

}

c++.cpp

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

//main function
main() {
    int num1=10,num2=20;
    if(num1==num2){
        cout<<"Num1 and Num2 are equal"<<endl;
    }
    else{
        cout<<"Num1 and Num2 are not equal"<<endl;
    }
}
```

Output:



If/else ladder:

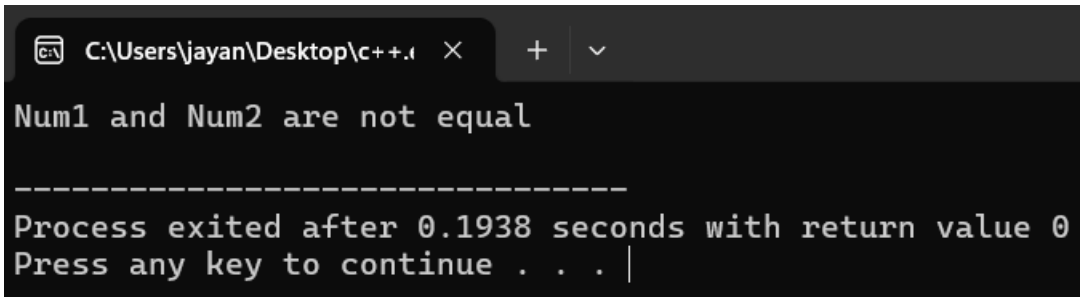
It is used to performed multiple cases for different conditions. In this statement there is on if condition and multiple else if conditions and one else block.

Syntax:

```
if(condition){
//body of if statement
}
else if(condition){
//body of else if statement
}
else if(condition){
//body of else if statement
}
else{
//body of else statement
}
```

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

//main function
main() {
    int num1=10,num2=20;
    if(num1==num2){
        cout<<"Num1 and Num2 are equal"<<endl;
    }
    else if(num1!=num2){
        cout<<"Num1 and Num2 are not equal"<<endl;
    }
    else{
        cout<<"This number is not valid"<<endl;
    }
}
```

Output:



Nested if:

Whenever if statement contain itself another if statement is called Nested-if statement.

Syntax:

```
if(condition){
        if(condition){
                //body of 2nd if statement
        }
        else{
                //body of else statement
        }
//body of 1st statement
}
else{
//body of else statement
}
```

```cpp
#include<iostream>
#include<string.h>
using namespace std;
main() {
    int num1=10,num2=20,num3=5;
    if(num1>num2){
        if(num1>num3){
            cout<<"Num1 is Greater:"<<num1<<endl;
        }
        else{
            cout<<"Num3 is Greater:"<<num3<<endl;
        }
    }
    else{
        if(num2>num3){
            cout<<"Num2 is Greater:"<<num2<<endl;
        }
        else{
            cout<<"Num3 is Greater:"<<num3<<endl;
        }
    }
}
```

Output:

```
C:\Users\jayan\Desktop\c++.
Num2 is Greater:20

--------------------------------
Process exited after 0.1972 seconds with return value 0
Press any key to continue . . .
```

Switch Statement

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

Rules:

(i)You can have any number of case statements within a switch. Each case is followed by the value of compared to and a colon.

(ii)The constant-expression of a case must be the same datatype as the variable in the switch, and it must be a constant or a literal.

(iii)When the variable being switch on is equal to case, the statements following that case will execute until a break statement is reached.

(iv)When a break statement is reached, the switch terminates, and the flow if control jump to the next line following the switch statement.

(v)Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.

(vi)A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the case is true. No break is needed in the default case.

Syntax:

```
switch(expression){
        case constant_expression:
                statements
                break;
        case constant_expression:
                statements
                break;
        default:
                statements
```

```
        break;
}
```

```
#include<iostream>
using namespace std;
main() {
    char grade='A';
    switch(grade){
        case'A':
            cout<<"Excellent"<<endl;
            break;
        case'B':
            cout<<"Very Good"<<endl;
            break;
        case'C':
            cout<<"Good"<<endl;
            break;
        default:
            cout<<"Please Enter Valid Grade"<<endl;
    }
}
```

Output:

```
C:\Users\jayan\Desktop\c++.
Excellent

--------------------------------
Process exited after 0.1803 seconds with return value 0
Press any key to continue . . .
```

Loops (Control Statement)

A loop executes the sequence of statements many times until the sated condition becomes false. A loop consists two parts, a body of a loop and a control statement. The control statement is a combination of some conditions that direct the body of the loop execute until the specified condition becomes false. The purpose of the loop is to repeat the same code a number of times.

Entry Controlled loop:

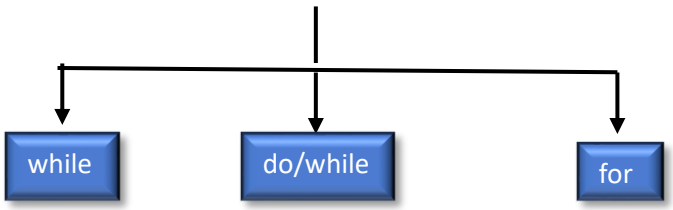
In an Entry controlled loop, a condition is checked before executing the body of a loop. It is also called as pre-checking loop.

Exit Controlled loop:

In an Exit Controlled loop, a condition is checked after executing the body of a loop. It is also called a post-checking loop.

The control conditions must be well defined and specified otherwise the loop will execute an infinite number of times. The loop doesn't stop executing and process the statements number of times is called as an infinite loop. An infinite loop is also called as an "Endless loop".

Types of Loops



While:

A while loop is the most straight forward looping structure.

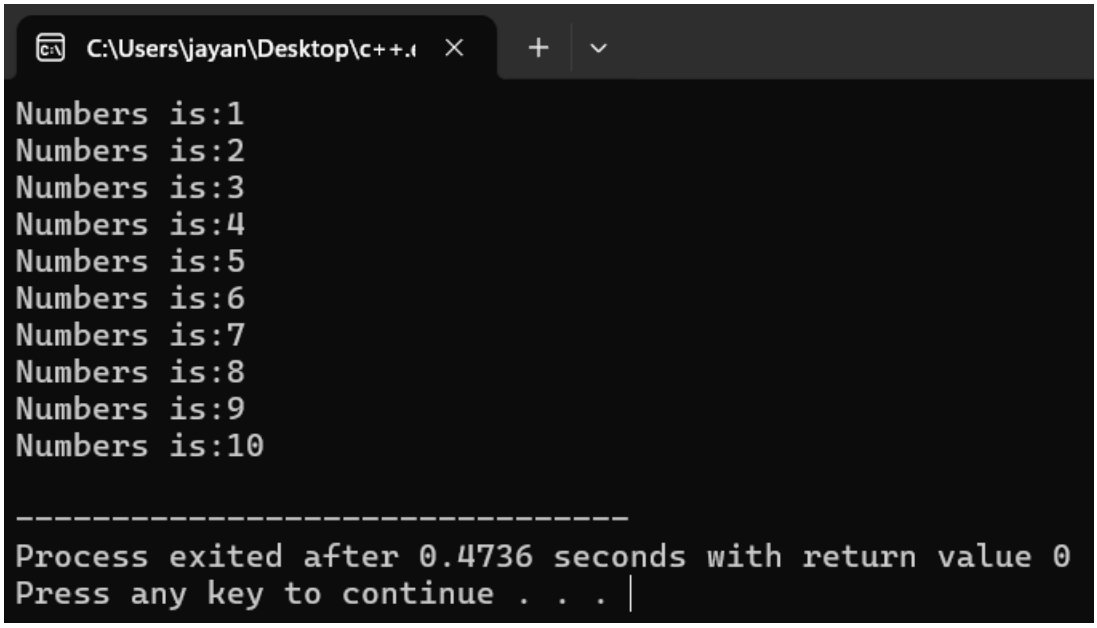
Syntax:

while(condition){

//body of while }

```
#include<iostream>
using namespace std;
main() {
    int a=1;
    while(a<=10){
        cout<<"Numbers is:"<<a<<endl;
        a++;
    }
}
```

Output:



Do-While:

A do while loop in C++ is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit controlled loop.

Syntax:

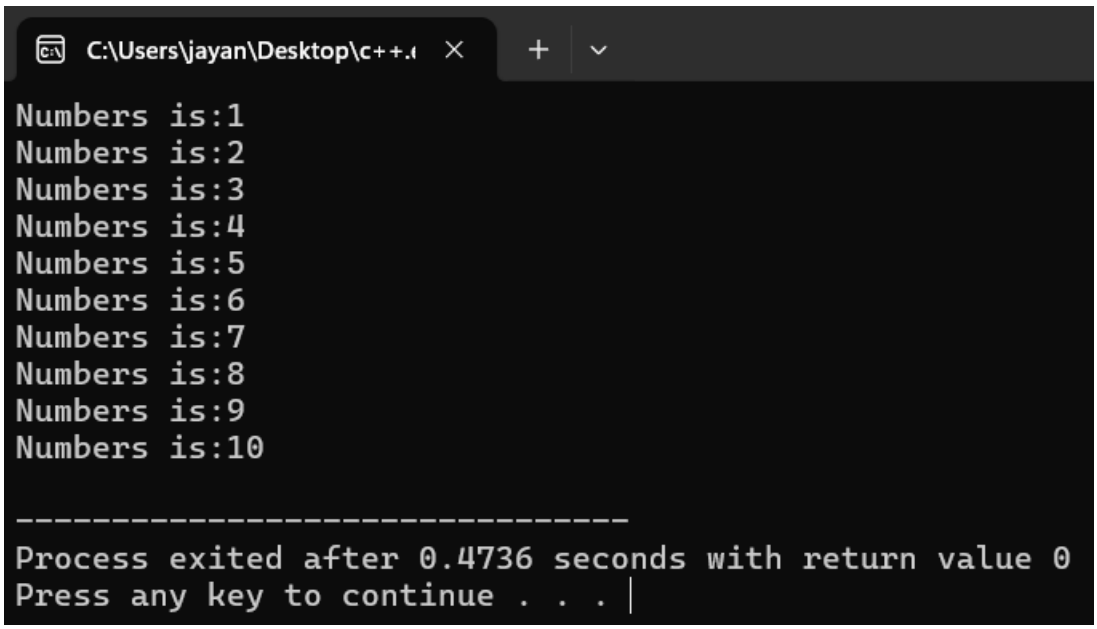
do{

//body of do

}while(condition);

```cpp
#include<iostream>
using namespace std;
main() {
    int a=1;
    do{
        cout<<"Numbers is:"<<a<<endl;
        a++;
    }while(a<=10);
}
```

Output:



For:

A for loop is a more efficient loop structure in C++ programming. The general structure is offer loop syntax:

Syntax

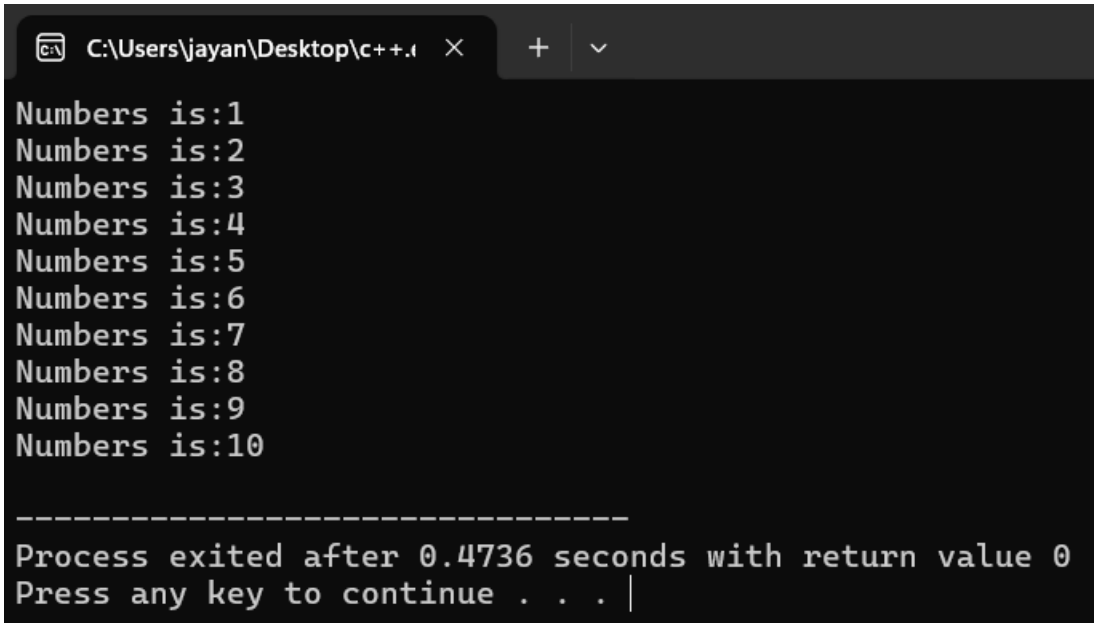
for(initial value;condition;operator(++,--)){

//body of for

}

```
#include<iostream>
using namespace std;
main() {
    for(int num=1;num<=10;num++){
        cout<<"Numbers is:"<<num<<endl;
    }
}
```

Output:



Nested for loop:

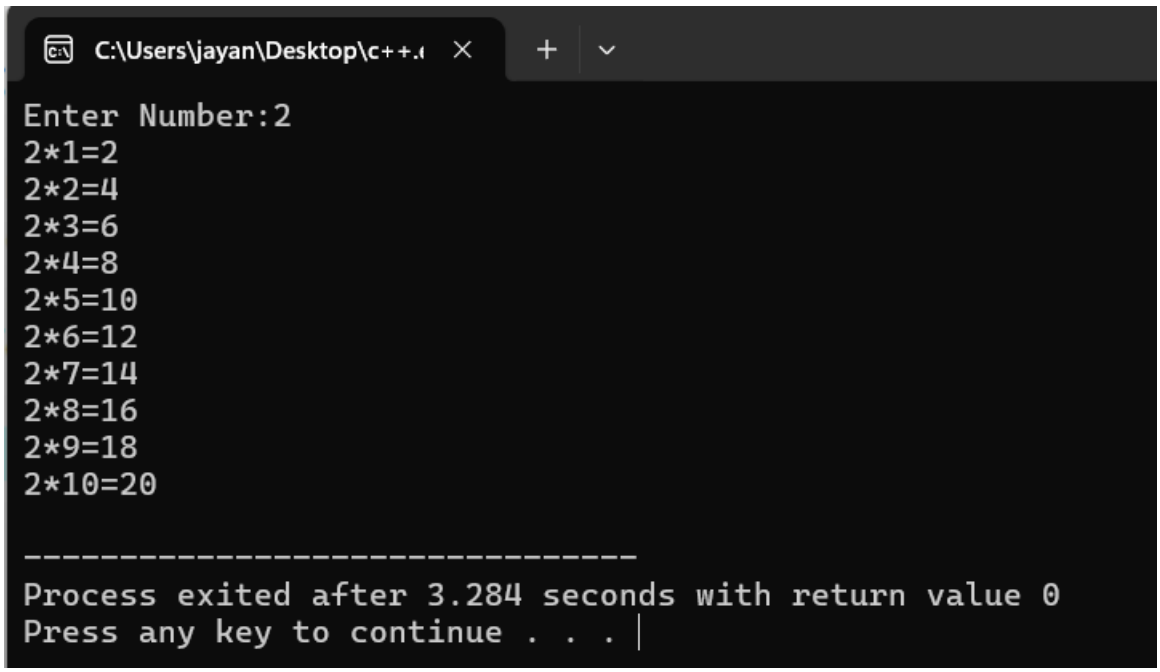
For loops can also be nested where there is and outer loop and an inner loops. For each iteration if the outer loop, the inner loop repeats its entire cycle.

Syntax

```
for(initial value; condition; operator(++,--){
        For (initial value; condition; operator(++,--){
                //body of second loop
        }
        //body of first loop
}
```

```
#include<iostream>
using namespace std;
main() {
    int table_no;
    cout<<"Enter Number:";
    cin>>table_no;
    for(int num1=table_no;num1<=table_no;num1++){
        for(int num2=1;num2<=10;num2++){
            cout<<num1<<"*"<<num2<<"="<<num1*num2<<endl;
        }

    }
}
```

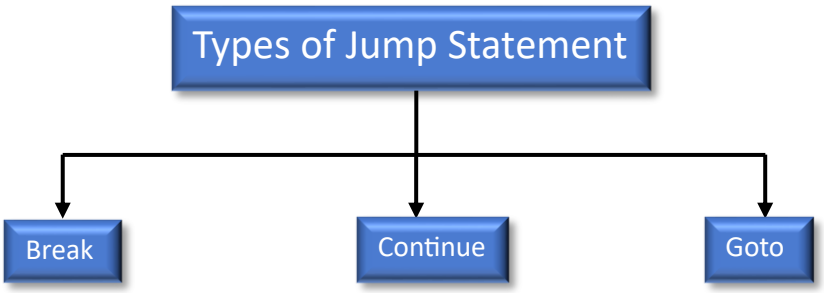
Output:


C:\Users\jayan\Desktop\c++.
Enter Number:2
2*1=2
2*2=4
2*3=6
2*4=8
2*5=10
2*6=12
2*7=14
2*8=16
2*9=18
2*10=20

Process exited after 3.284 seconds with return value 0
Press any key to continue . . .

Jumping Statement

The statement are used in C++ for unconditional flow of control throughout the function in a program. They support three types of jump statement.



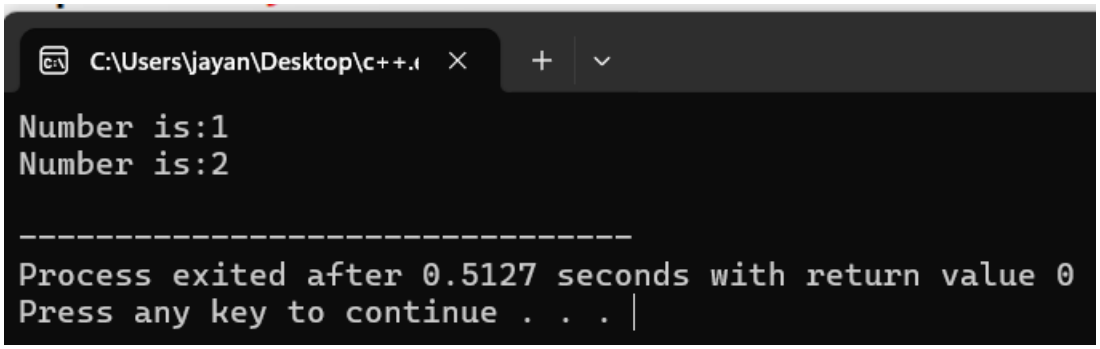
Break:

(i)The loop control statement is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stops there and control returns from the loop immediately to the first statement after the loop.

(ii)Basically break statements are used in the situations when we are not sure about the actual number of iterations for the loop or we want to terminate the loop based on some condition.

```
#include<iostream>
using namespace std;
main() {
    for(int i=1;i<=5;i++){
        if(i==3){
            break;
        }
        cout<<"Number is:"<<i<<endl;
    }
}
```

Output:

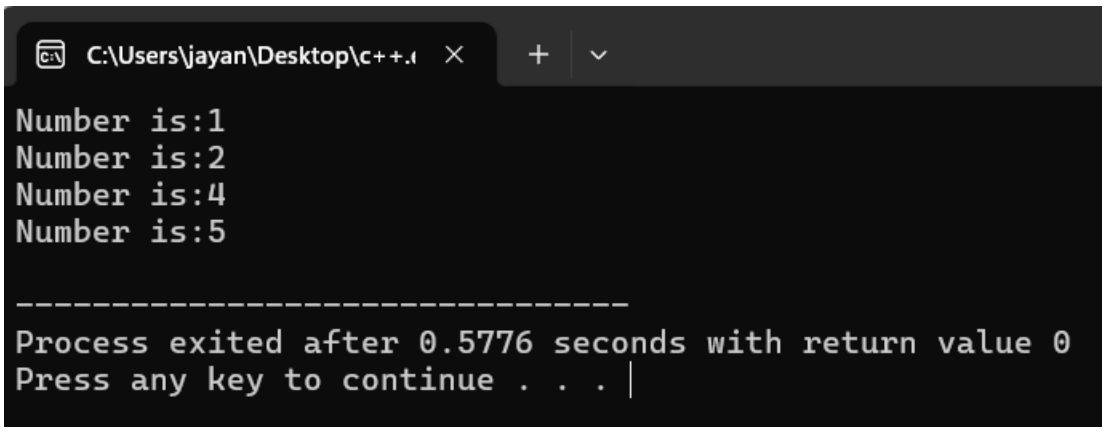


Continue:

The continue statement is opposite to that break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.

```
#include<iostream>
using namespace std;
main() {
    for(int i=1;i<=5;i++){
        if(i==3){
            continue;
        }
        cout<<"Number is:"<<i<<endl;
    }
}
```

Output:

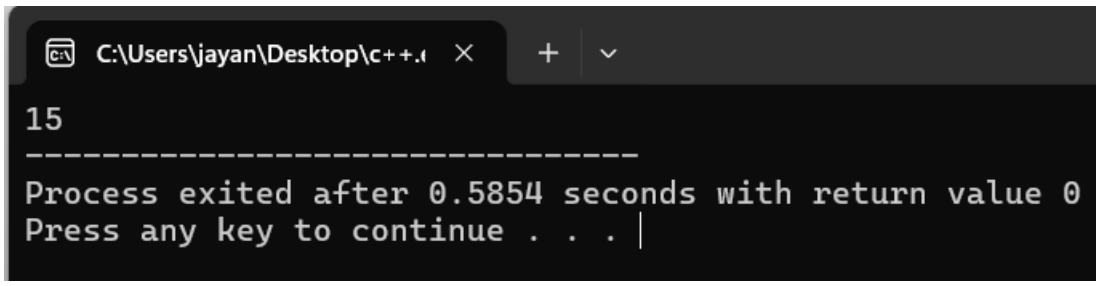


Goto:

The goto statement in C++ also referred to as unconditional jump statement can be used to jump from one point to another point within a function.

```
#include<iostream>
using namespace std;
main() {
    int i,sum=0;
    for(i=0;i<=10;i++){
        sum+=i;
        if(i==5){
            goto addition;
        }
    }
    addition:
        cout<<sum;
}
```

Output:


C:\Users\jayan\Desktop\c++.
15

Process exited after 0.5854 seconds with return value 0
Press any key to continue . . .

Pointers

The pointer in C++, is a variable that stores address of another variable. A pointer can also be used to refer to another pointer function. A pointer can be increment and decrement, point to the next/previous memory location. The purpose of pointer is to save memory space and achieve faster execution time.

Note: & : Address , * : Declaration

Declaring a pointer:

(i)Pointers declared before they can be used in your program. Pointers can be name anything you want as long as the obey C++ naming rules.

Syntax:

data_type *pointer_variable_name;

(ii)The asterisk (*) which is indirection operator, declares a pointer.

Initialize a pointer:

(i)If pointers in C++ programming are not uninitialized and used in the program, the result are unpredictable and potentially disastrous.

(ii)We use the ampersand (&) operator, placed before the name of a variable whose address we need. Pointer initialization is done with the following syntax.

pointer=&variable;

Advantages:

(i)Pointers are useful for accessing memory location.

(ii)Pointers provide an efficient way for accessing the elements of an array structure.

(iii)Pointers are used for dynamic memory allocation as well as deallocation.

(iv)Pointers are used to form complex data structures such as linked list, graph, tree etc.

Disadvantages:

(i)Pointers are a little complex to understand.

(ii)Pointers can lead to various errors such as segmentation faults or can access a memory location which is not required at all.

(iii)If an incorrect value is provided to a pointer, it may cause memory corruption.

(iv)Programmers find it very difficult to work with the pointers; therefore it is programmer's responsibility to manipulate a pointer carefully.

```
#include<iostream>
using namespace std;
main() {
int i,a[5]={1,2,3,4,5};
int *p=a;
for(i=0;i<5;i++){
    cout<<*p<<endl;
    p++;
    }
}
```

Output:

```
C:\Users\jayan\Desktop\c++.
1
2
3
4
5

--------------------------------
Process exited after 1.128 seconds with return value 0
Press any key to continue . . .
```

Structure

(i)In C++, there are cases where we need to store multiple attributes of an entity. It is not necessary that an entity has all the information of one type only. It can have different attributes of different data types.

(ii) Construct individual arrays for storing names, roll numbers, and marks.

(iii)Use a special data structure to store the collection of different data types.

(iv)Structure in C++ is a user-defined data type that enables us to store the collection of different datatypes.

(v) struct keyword are used for create structure

Syntax:

```
struct structure_name{
datatype member1;
datatype member2;
datatype member n;
};
```

Declaration of Structure:

(i)We can declare a variable for the structure so that we can access the member of the structure easily. There are two ways to declare struct variable.

(ii)By struct keyword within main() function.

(iii)By declaring a variable at the time of defining the structure.

(iv)The variable e1 and e2 can be used to access the values stored in the structure.

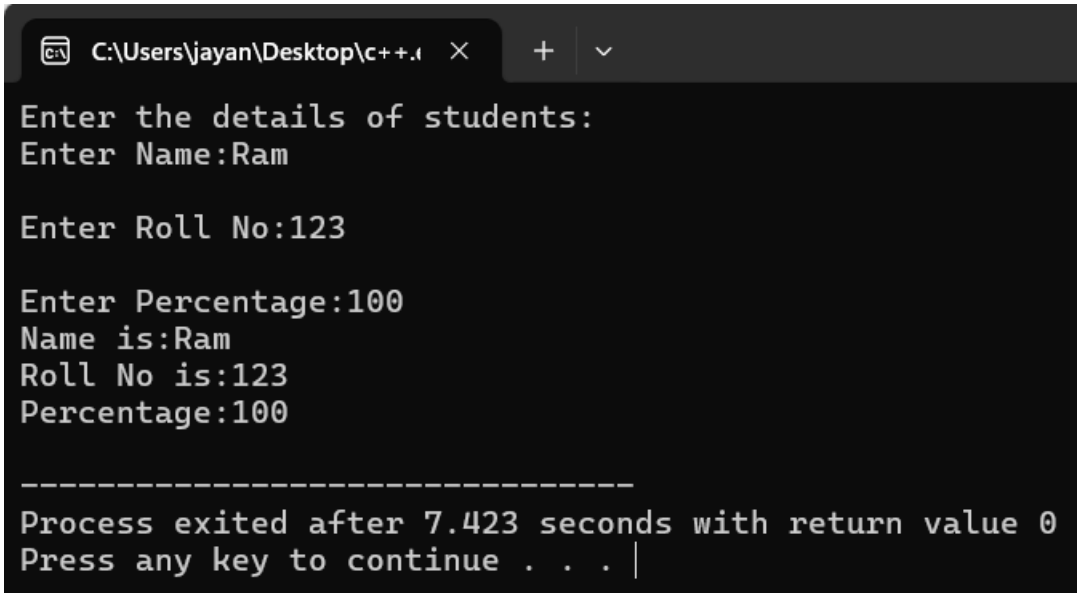
(v) Another way to declare variable at the time of defining the structure:

```
struct employee{
int id;
char name[50];
float salary;
```

}e1,e2;

```
#include<iostream>
using namespace std;
//Structure
struct student{
    int roll_no;
    char name[10];
    float percentage;
}s1;
//Main function
main() {
cout<<"Enter the details of students:"<<endl;
cout<<"Enter Name:";
cin>>s1.name;
cout<<endl<<"Enter Roll No:";
cin>>s1.roll_no;
cout<<endl<<"Enter Percentage:";
cin>>s1.percentage;
cout<<"Name is:"<<s1.name<<endl;
cout<<"Roll No is:"<<s1.roll_no<<endl;
cout<<"Percentage:"<<s1.percentage<<endl;
}
```

Output:


C:\Users\jayan\Desktop\c++.
Enter the details of students:
Enter Name:Ram
Enter Roll No:123
Enter Percentage:100
Name is:Ram
Roll No is:123
Percentage:100

Process exited after 7.423 seconds with return value 0
Press any key to continue . . .

Union

(i)A union is a special datatype available in C++ that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple purpose.

(ii)The union statement defines a new datatype with more than one member for your program:

Syntax:

Union union_name{

Member definition;

}variables;

(iii)A variable of datatype can store an integer, a floating point numbers, or a string of characters. It means a single variable. Same memory location, can be used to store multiple types of data. You can use any built-in or user defined datatypes inside a union based on your requirements.

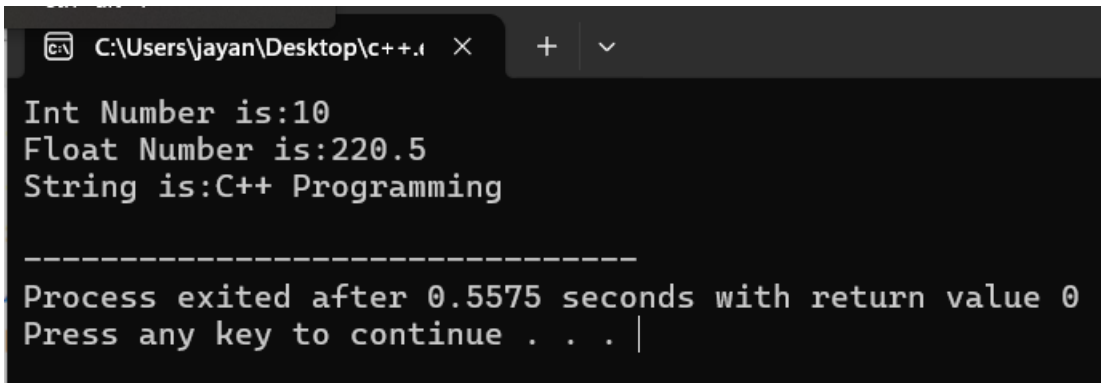
(iv)The memory occupied by a union will be large enough to hold the largest member of the union.

Accessing Union Members

To access any member of a union, we use the member access operator(.). The member access operator is coded as a period between the union variable name and the union member that we with wish to access. You would use the keyword union to define variables of union type.

```cpp
#include<iostream>
#include<string.h>
using namespace std;
//Structure
union Data{
    int i;
    float f;
    char str[20];
};
//Main function
main() {
union Data data;
data.i=10;
cout<<"Int Number is:"<<data.i<<endl;

data.f=220.5;
cout<<"Float Number is:"<<data.f<<endl;

strcpy(data.str,"C++ Programming");
cout<<"String is:"<<data.str<<endl;
}
```

Output:



Storage Classes

Storage classes are used to determine the lifetime, visibility, memory location, and initial value of a variable. They are four types of storage classes in C++.

Storage Classes	Storage Place	Default Value	Scope	Lifetime
Auto	Ram	Garbage value	Local	Within Function
Extern	Ram	Zero	Global	Till the end of the program maybe decrease anywhere in the program.
Static	Ram	Zero	Local	Till end of the main program, retains value between multiple function call.
Register	Register	Garbage Value	Local	Within the function

Automatic(auto):

(i)Automatic variable are allocated memory automatically at runtime.

(ii)Visibility of the auto variables is limited to the block in which they are defined.

(iii)Scope of the auto variable is limited to the block in which they are defined.

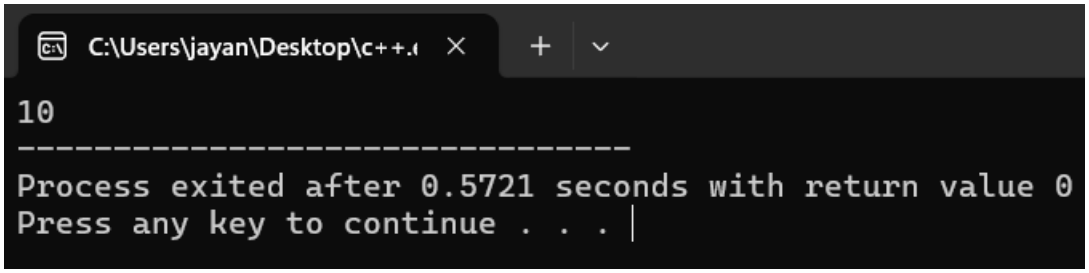
(iv)It is initialized to the garbage value by default.

(v)The memory assigned to automatic variables gets freed upon exiting from the block.

(vi)Every local variable is automatic in C++ by default.

```
#include<iostream>
#include<string.h>
using namespace std;
main(){
    auto int a=10;
    cout<<a;
}
```

Output:



Static:

(i)The variables defined as static specifier can hold their values between the multiple function calls.

(ii)Static local variable are visible only to the function or the block in which they are defined.

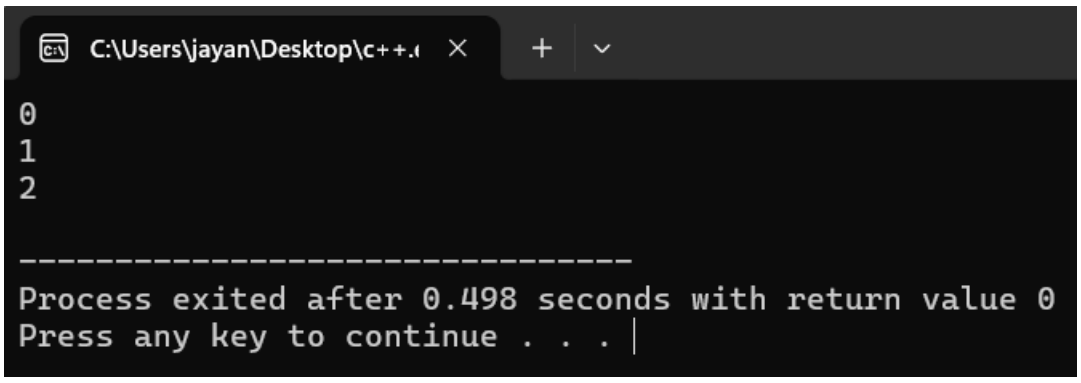
(iii)A same static variable can be declared many times but can be assigned at only one time.

(iv)Default initial value of the static integral variable is 0 otherwise null.

(v)The visibility of the static global variable is limited to the life in which in which it has declared.

```
#include<iostream>
#include<string.h>
using namespace std;
int fun_static(){
    static int num=0;
    cout<<num<<endl;
    num++;
    return 0;
}
main(){
    for(int i=1;i<4;i++){
        fun_static();
    }
}
```

Output:



Register:

(i)The variable defines as the register is allocated the memory into the CPU, registers depending upon size of the memory remaining in the CPU.

(ii)We can not use & operator for the register variable.

(iii) The access time of the register variable is faster than the automatic variables.

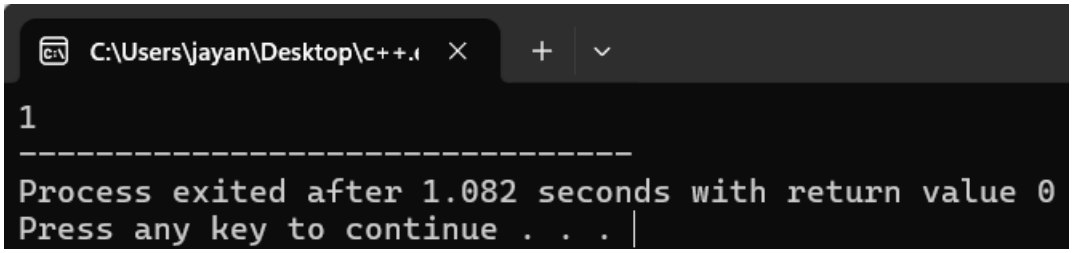
(iv)The initial default value of the register local variable is 0.

(v) We can store pointers into the register, register can store the address of a variable.

(vi)Static variables cannot stored into the register since we can not use more than one storage specifier for the same variable.

```
#include<iostream>
#include<string.h>
using namespace std;
main(){
register int a;
cout<<a;
}
```

Output:



External:

(i)The external storage class is used to tell the compiler that the variable that the variable defined as extern is declared with an external linkage else where in the program.

(ii)The variable declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in the program.

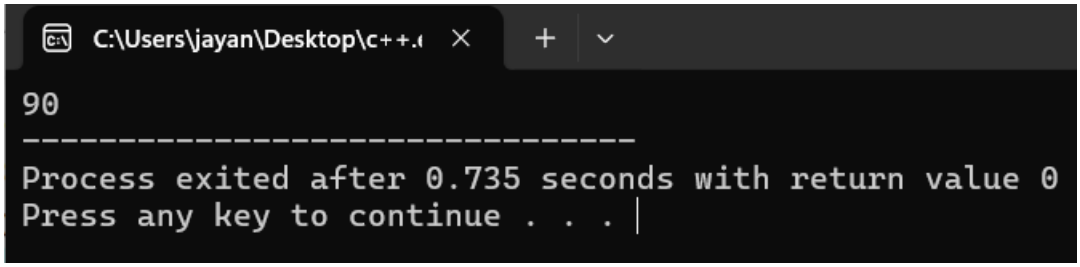
(iii)The default value of external integral type of 0 otherwise null.

(iv)We cannot initialize the external variable within any block or method.

(v) An external variable can be declared many times but can be initialized at once only.

```
#include<iostream>
using namespace std;
extern int a=90;
main(){
cout<<a;
}
```

Output:


C:\Users\jayan\Desktop\c++.
90

Process exited after 0.735 seconds with return value 0
Press any key to continue . . .

Arrays and its types

(i)Array is a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array us used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same.

Declaration

To declare an array in C++, a programmer specifies the type of the elements and number of elements required by an array.

Datatype array_name[array_size];

This is called single dimensional array.

The array size must be an integer constant greater than zero and type can be any valid C++ datatype.

Double balance[10];

Array initialization:

The simplest way to initialize an array is by using the index of each element. We can initialize each element of the array by using the index.

Marks[0]=80;

Marks[1]=60;

(i)You can initialize an array in C++ either one by one or using a single statement.

(ii)double balance[5]={1000.0,2.3,5.9,6.3,200.0};

(iii)The number of values between braces{} cannot be larger than the number of elements that we declare for the array between square brackets[].

(iv)If you omit the size of the array, an array just big enough to hold the initialization is created.

(v)Assign a single element of the array

Balance[4]=50.0

(vi)The above statement assigns the 5th element in the array with a value of 50.0.All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be total size of the array.

Accessing an array element:

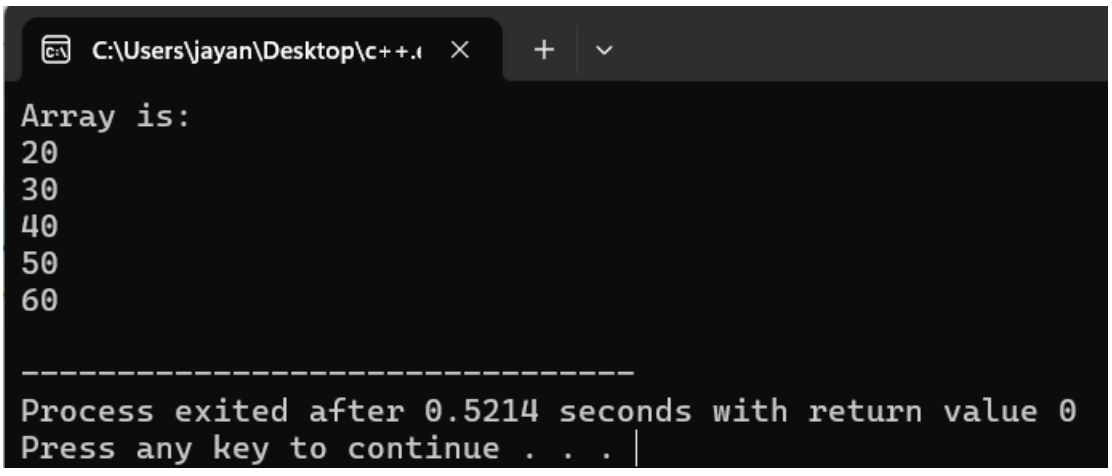
(i)An element is accessed by indexing the array name.This is done by placing the index of the element within the square brackets after the name of the array.

Double salary=balance[9];

(ii)The above statement will take the 10th element from the array and assign the value to salary variable.

```
#include<iostream>
using namespace std;
//main function
main(){
    int i=0,marks[5]={20,30,40,50,60};
    cout<<"Array is:"<<endl;
    for(i=0;i<5;i++){
        cout<<marks[i]<<endl;
    }
}
```

Output:



Two Dimensional Arrays:

(i)The simplest form of multidimensional array is the two dimensional arrays. A two dimensional arrays is, in essence a list of one-dimensional arrays. To declare a two dimensional integer array of size[x][y], you would write something.

(ii) datatype arrayName[x][y];

(iii)Where type can be any valid C++ datatype and arrayName will be a valid C++ identifier. A two dimensional array can be considered as a table which will have x number of rows and y number of columns.

	Column1	Column1	Column2	Column3
Row0	**a[0][0]**	**a[0][1]**	**a[0][2]**	**a[0][3]**
Row1	**a[1][0]**	**a[1][1]**	**a[1][2]**	**a[1][3]**
Row2	**a[2][0]**	**a[2][1]**	**a[2][2]**	**a[2][3]**

Initializing 2D Array

Multidimensional arrays many be initialized by specifying bracket values for each row.Following is an array with 3 rows and each rows has 4 columns.

Int a[3][4]={{0,1,2,3},{4,5,6,7},{8,9,10,11}};

Accessing a 2D Array

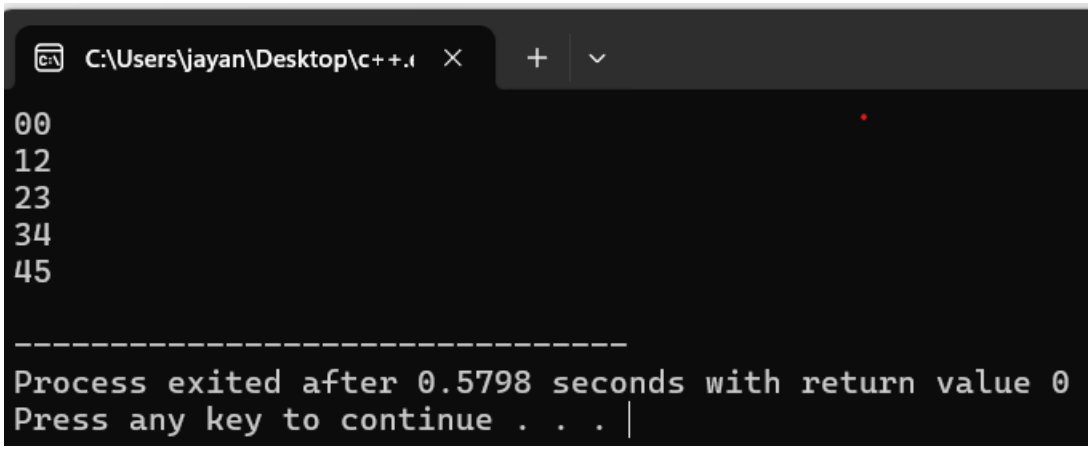
An element in a two dimensional array is accessed by using the subscript, row index and column index of the array.

int val=a[2][3];

The above statement will be take the 4^{th} element from the 3^{rd} row of the array.

```cpp
#include<iostream>
using namespace std;
//main function
main(){
    int a[5][2]={{0,0},{1,2},{2,3},{3,4},{4,5}};
    for(int i=0;i<5;i++){
        for(int j=0;j<2;j++){
            cout<<a[i][j];
        }
        cout<<endl;
    }
}
```

Output:



3D Arrays

Total numbers of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensional.

Declare 3D Array:

(i)Specify datatype, arrayName, block size, row size and column size.

(ii)Each subscript can be written within its own separate pair of brackets.

```
int arr[2][2][2];
```

Accessing 3D array:

Accessing elements in three dimensional arrays is also similar to that of two dimensional arrays. The difference is we have to use three loops instead of two loops for one additional dimension in three dimensional arrays.

```
#include<iostream>
using namespace std;
//main function
main(){
    int arr[3][3][3]={
    {
        {11,12,13},
        {14,15,16},
        {16,17,18},
    },
    {
        {1,1,1},
        {1,1,1},
        {1,1,1},
    },
    {
        {1,2,3},
        {1,2,3},
        {1,2,3},
    }
};

for(int i=0;i<3;i++){
    for(int j=0;j<3;j++){
        for(int k=0;k<3;k++){
```

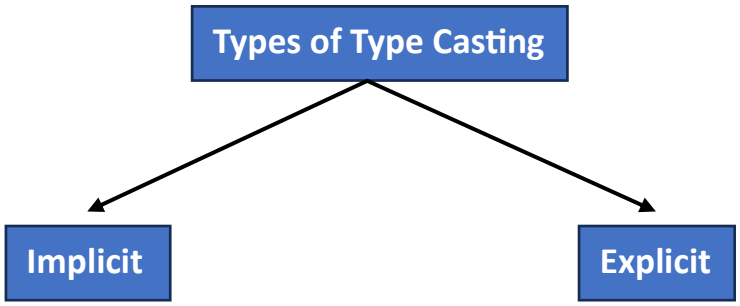
```
23 for(int i=0;i<3;i++){
24     for(int j=0;j<3;j++){
25         for(int k=0;k<3;k++){
26             cout<<arr[i][j][k]<<" ";
27         }
28         cout<<endl;
29     }
30     cout<<endl;
31 }
32 }
```

Output:

```
C:\Users\jayan\Desktop\c++.

11 12 13
14 15 16
16 17 18

1 1 1
1 1 1
1 1 1

1 2 3
1 2 3
1 2 3


--------------------------------
Process exited after 0.9506 seconds with return value 0
Press any key to continue . . .
```

Type Casting

Type casting is converting one data type into another one data type. It is also known as data conversion or type conversion.

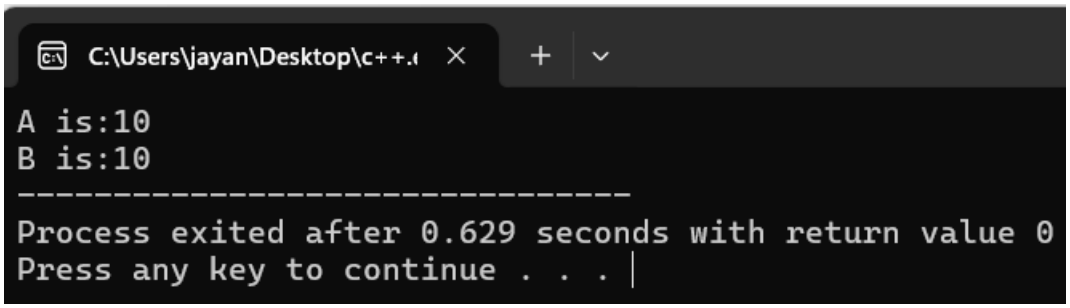


Implicit:

Implicit type casting means conversion of datatypes without losing its original meaning. This type of typecasting is essential when you want to change datatype without changing the significance of the value stored inside the variable.

```
#include<iostream>
using namespace std;
//main function
main(){
    short a=10;
    int b;
    b=a;
    cout<<"A is:"<<a<<endl;
    cout<<"B is:"<<b;
}
```

Output:

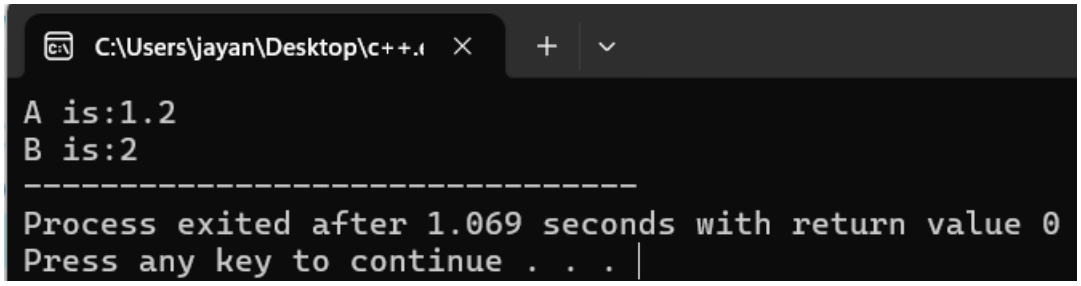


Explicit:

In Explicit type casting, the data type converted automatically. There are some scenarios in which we may have to force type conversion.

```
#include<iostream>
using namespace std;
//main function
main(){
    float a=1.2;
    int b=int(a)+1;
    cout<<"A is:"<<a<<endl;
    cout<<"B is:"<<b;
}
```

Output:

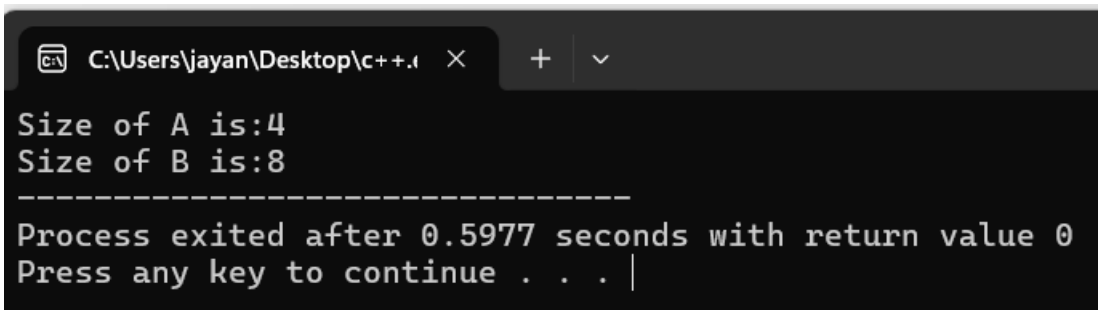


Size-of operator

The operator returns the size of its operand, in bytes. The size of operator always proceeds its operand. The operand is an expression, or it may be cast.

```
#include<iostream>
using namespace std;
//main function
main(){
   int a;
   double b;
   cout<<"Size of A is:"<<sizeof(a)<<endl;
   cout<<"Size of B is:"<<sizeof(b);
}
```

Output:



Pre-Processor Directives

The preprocessor in C++ is used for processing the code before it is compiled by the compiler. It does many tasks such as including files, conditional compilation, using macros, etc. The preprocessor also allows the developers to select which portions of code should be included or excluded.

In C++, the preprocessor directives are special commands that are used to instruct the preprocessor. It begins with a '#' symbol and tells the preprocessor to the modify source code before compilation.

Directives	Purpose
#include	Links a header file in the source code.
#define	Creates a symbolic or macro constant.
#undef	Deletes a macro that has already been defined.
#ifdef/#ifndef	Compilation that is conditional on the presence or absence of a macro.
#if/#else/#endif	Compilation that is conditional based on some expression.
#error	Halts the compilation process and produces and error notice.
#warning	During compiler, a warning notice is shown.
#pragma	Provide the compiler specific instruction.

#include:

The #include preprocessor directive is used to include the contents of one file into the current one, use the #include directive.

Syntax:

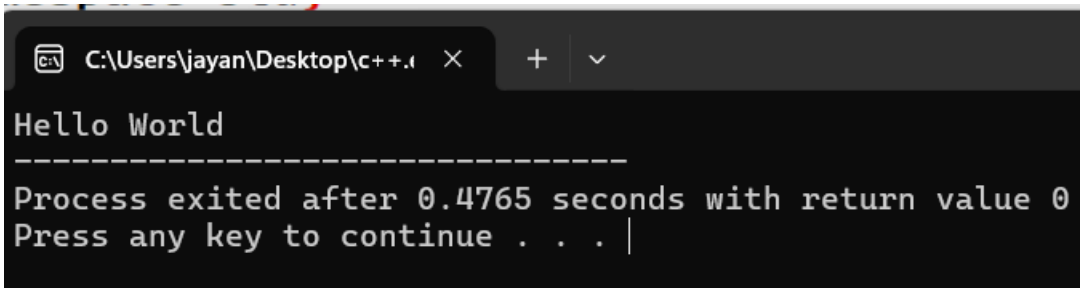
#include<headerFileName>

or

#include"headerFileName"

```cpp
#include<iostream>
using namespace std;
//Main Function
int main(){
    cout<<"Hello World";
    return 0;
}
```

Output:



#define:

(i)The #define preprocessor directive is used to define macros. Macro names are symbolic and may be used to represent constant values or short bits of code.

(ii)Using #define preprocessor makes our code more readable and easily maintainable as we can replace numbers and code snippets with a meaningful name.

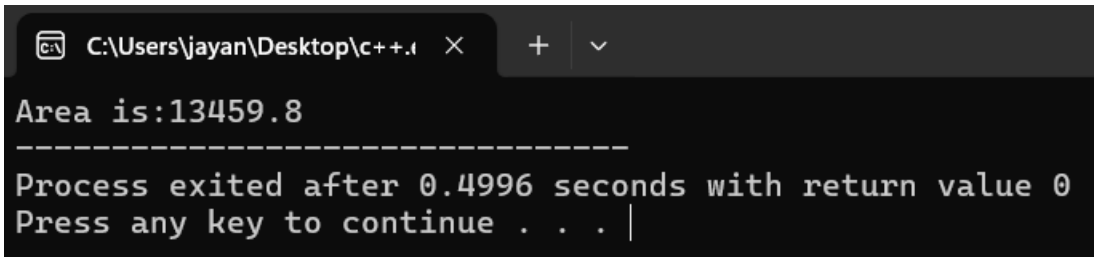
Syntax:

#define macroName value

c++.cpp

```cpp
#include<iostream>
#define PI 3.14
using namespace std;
//Main Function
void area(float radius,float height){
    float total=PI*(radius*radius)*height;
    cout<<"Area is:"<<total;
}
int main(){
    area(20.5,10.2);
    return 0;
}
```

Output:



OOPs (Object Oriented Programming)

Classes and Objects

Classes:

A class in C++ is the building blocks, that leads to object oriented programming. It is a user-defined data type, which holds its own data members and members functions, which can be accessed by creating an instance of that class.

Syntax:

```
class className{
//Access Specifier:
//Data Members;
//Member Functions()
};
```

Note: By default access specifier is private in C++.

Object:

An object is an instance of class. When a class is defined no memory is allocated but when it is instantiated (i.e an object is created) memory is allocated.

Declaring Objects:

When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Accessing data members and member functions

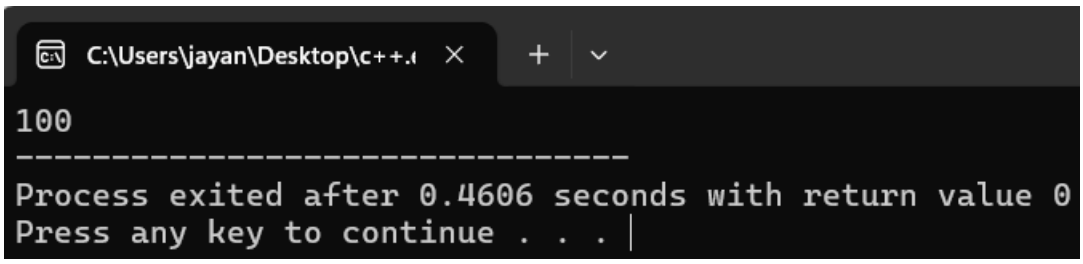
The data members and members functions of class can be accessed using the dot(.) operator with the object.

Syntax:

```
ClassName objectName;
```

```cpp
#include<iostream>
using namespace std;
//Class
class Calculation{
    public:
    int add(){
        int num1=10,num2=90;
        int total=num1+num2;
        cout<<total;
    }
};
//main function
main(){
  //Create object
  Calculation obj;
  //function call
  obj.add();
}
```

Output:



Encapsulation

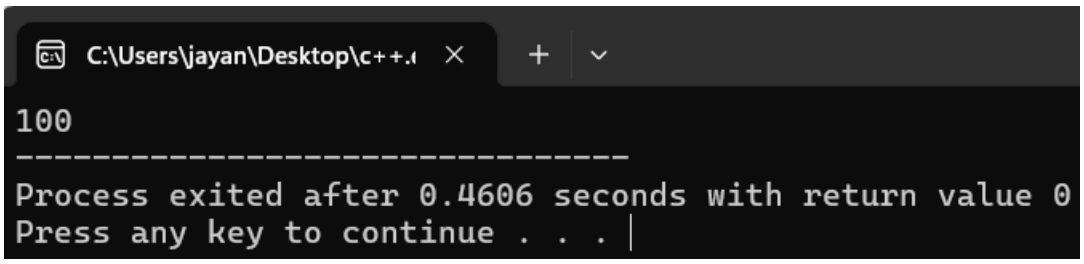
(i)The wrapping up of data and function into a single unit (called class) is known as Encapsulation.

(ii)Data and Encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions which wrapped in the class can access it.

(iii)These functions provide the interface between the objects data and the program. This insulation of the data from direct access by the program is called hiding or information hiding.

```
#include<iostream>
using namespace std;
//Class
class Calculation{
    public:
    int add(){
        int num1=10,num2=90;
        int total=num1+num2;
        cout<<total;
    }
};
//main function
main(){
  //Create object
  Calculation obj;
  //function call
  obj.add();
}
```

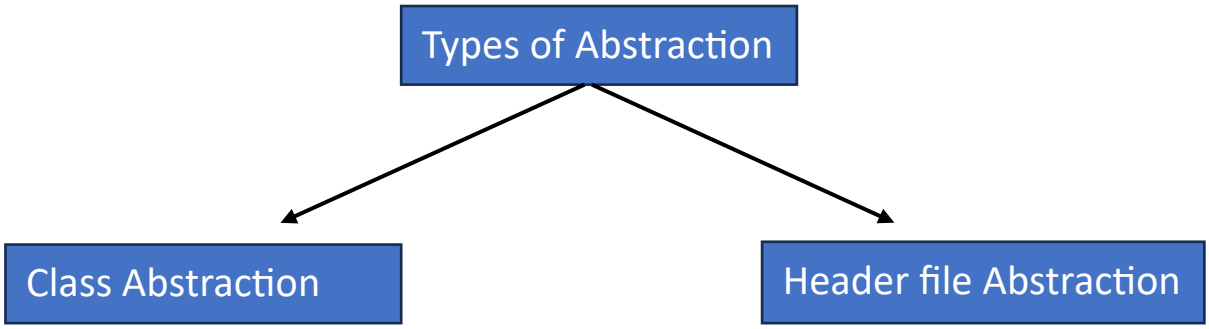
Output:


C:\Users\jayan\Desktop\c++.
100

Process exited after 0.4606 seconds with return value 0
Press any key to continue . . .

Abstraction

Data Abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.



Class Abstraction:

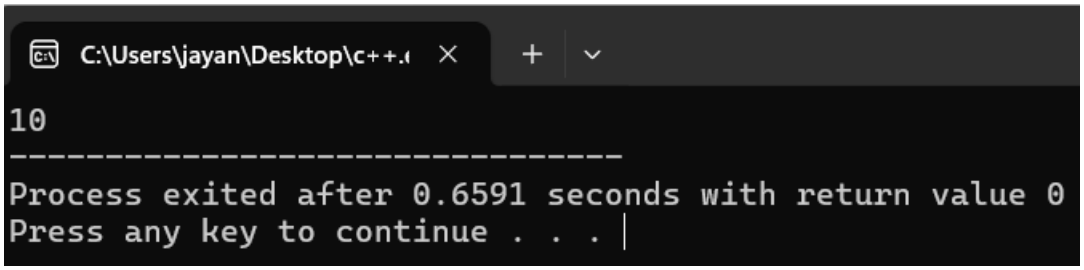
We can implement abstraction in C++ using classes. Class helps us to group data members and members functions using available access specifiers.

Header File Abstraction:

It refers to that header file when functions are hided in already in the header file without using header files functions cannot access.

```cpp
#include<iostream>
using namespace std;
//Class
class Calculation{
    //data hiding
    private:
        int a;
    public:
    int b(){
    a=10;
    cout<<a;
    return 0;
    }
};
//main function
main(){
  //Create object
  Calculation obj;
  //function call
  obj.b();
}
```

Output:



Constructor

(i)Constructor in C++ is a special method that is invoked automatically at the time an object of a class is created.

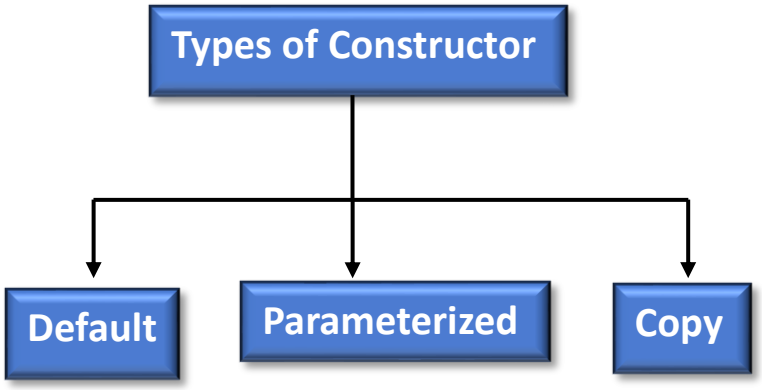
(ii)It is used to initialize the data members of new objects generally.

(iii)The constructor in C++ has the same name as the class. It constructs the values i.e. provides data for the object which is why it is known as a constructor.

Syntax:

```
class className{
//access specifier
//Make Constructor
    className(){
        //Constructor Definition
        }
};
//Main Function
int main(){
//Create object
className obj;
return 0;
}
```

Note: No need to call constructor through object because after creation of object, constructor will automatically executed.



Default Constructor:

Default Constructor is the constructor which doesn't take any argument or parameter. It has no parameters. It's also called zero-parameter constructor.

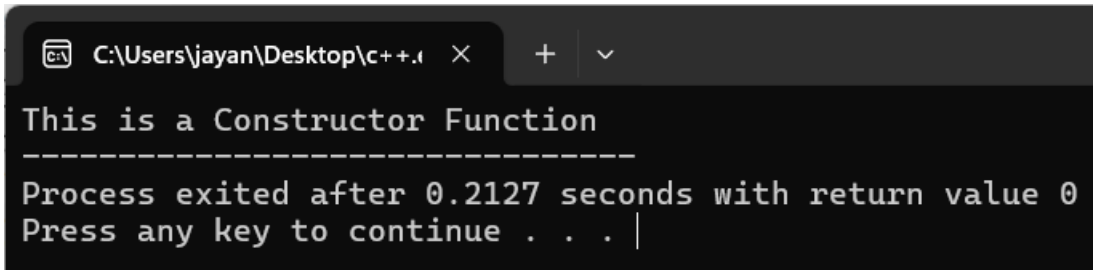
Syntax:

```
class className{
//access specifier
//Make Constructor
    className(){
        //Constructor Definition
        }
};
//Main Function
int main(){
//Create object
className obj;
return 0;
}
```

c++.cpp

```cpp
#include<iostream>
using namespace std;
//Make class
class A {
    public:
        //Make Constructor
        A(){
            cout<<"This is a Constructor Function";
        }
};
//Main Function
int main(){
    //Object Creation or constructor calling
    A obj;
    return 0;
}
```

Output:



Parameterized Constructor:

The constructor which take arguments or parameters is called parameterized constructor.

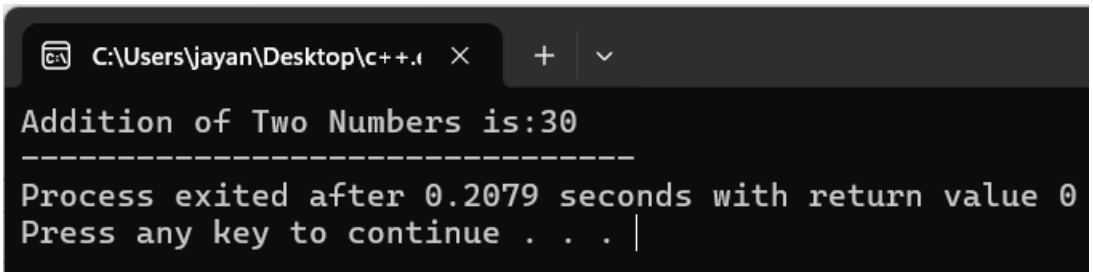
Syntax:

```
class className{
//access specifier
//Make Constructor
    className(Number of Parameters){
        //Constructor Definition
        }
};
//Main Function
int main(){
//Create object
className obj(Values of Parameters);
return 0;
}
```

c++.cpp

```cpp
#include<iostream>
using namespace std;
//Make class
class Add {
    public:
        //Make Constructor
        Add(int num1,int num2){
            int total=num1+num2;
            cout<<"Addition of Two Numbers is:"<<total;
        }
};
//Main Function
int main(){
    //Object Creation or constructor calling
    Add obj(20,10);
    return 0;
}
```

Output:



Copy Constructor:

(i)Copy Constructor is a type of constructor which is used to create a copy of an already existing object of a class type.

(ii) A copy constructor is a member function that initializes an object using another object of the same class.

(iii)The Compiler provides a default copy constructor to all the classes.

Syntax:

```
class className{
//access specifier
//Make Constructor
    className(className &obj){
        //Constructor Definition
        }
};
```

c++.cpp

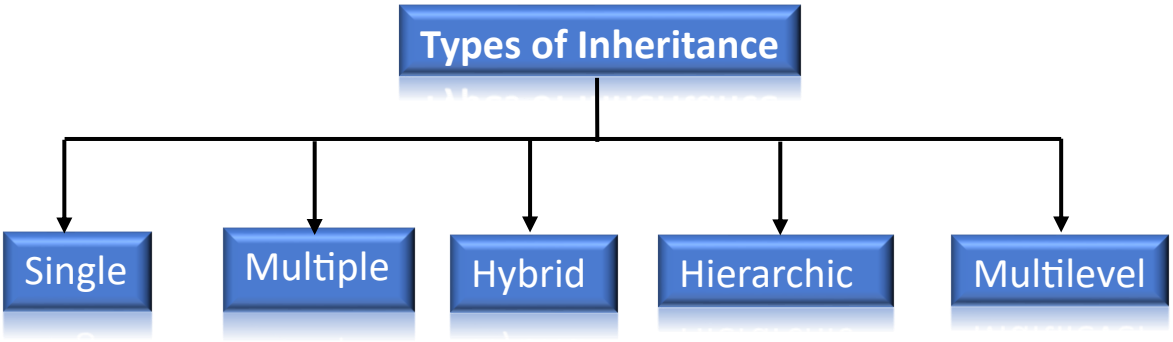
```cpp
#include<iostream>
using namespace std;
//Make class
class A {
    public:
        int x;
        //Paramterized Constructor
        A(int a){
            x=a;
        }
        //Copy Constructor
        A(A &i){
            x=i.x;
        }
};
//Main Function
int main(){
    A a1(20);//Paramterized Constructor
    A a2(a1);//Copy Constructor
    cout<<a2.x;
    return 0;
}
```

Output:

```
C:\Users\jayan\Desktop\c++.
20
--------------------------------
Process exited after 0.188 seconds with return value 0
Press any key to continue . . .
```

Inheritance

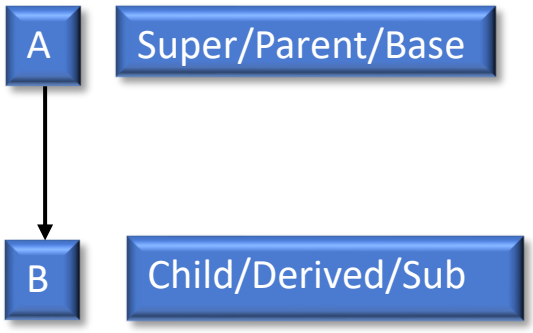
In C++, inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviors which are defined in other class.



Single Inheritance:

Single inheritance enables a derived class to inherit properties and behavior from a single parent class. It allows a derived class to inherit the properties or behavior of a base class, thus enabling code reusability as well as adding new features to the existing code.

Block Diagram:



Syntax:

Base Class:

```
class className{
//Access Specifier
//Data Members
//Member Functions
};
```

Derived Class:

```
class className : public BaseClassName{
//Access Specifier
//Function Definition of Base Class
Or
//Functions of Derived class
};
```

Main Function

```
main(){
derivedClassName objectName;
objName.functionName();
}
```

```
#include<iostream>
using namespace std;
//Parent Class
class Parent{
    public:
        int num1,num2;
        int show();
};
//Child Class
class Child : public Parent{
    public:
        int show(){
            cout<<"Enter First Number:";
            cin>>num1;
            cout<<"Enter Second Number:";
            cin>>num2;
            //print the numbers on screen
            cout<<"First Number is:"<<num1<<endl;
            cout<<"Second Number is:"<<num2;
        }
};
//Main Class
main(){
    Child obj;
    obj.show();
}
```

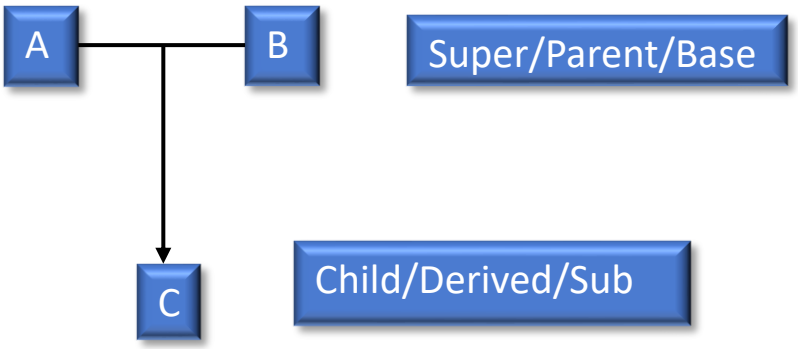
```
23 main(){
24     Child obj;
25     obj.show();
26 }
27
```

Output:

```
C:\Users\jayan\Desktop\c++.
Enter First Number:20
Enter Second Number:20
First Number is:20
Second Number is:20
--------------------------------
Process exited after 3.995 seconds with return value 0
Press any key to continue . . .
```

Multiple Inheritance:

Multiple Inheritance is a type of inheritance where a class can inherit form more than one classes.



```cpp
#include<iostream>
using namespace std;
//Parent Class 1
class Father{
    public:
        int num1,num2;
        int show();
};
//Parent Class 2
class Mother{
    public:
        int num3,num4;
        int get_show();
};
//Child Class
class Child : public Father,public Mother{
    public:
        int show(){
            cout<<"Enter First Number:";
            cin>>num1;
            cout<<"Enter Second Number:";
            cin>>num2;
            //print the numbers on screen
            cout<<"First Number is:"<<num1<<endl;
            cout<<"Second Number is:"<<num2<<endl;
```

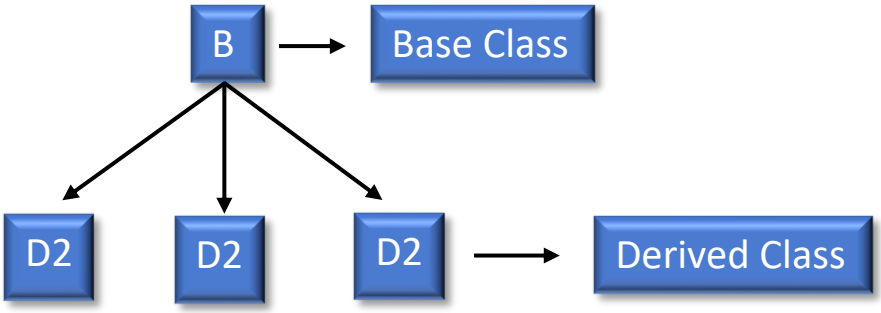
```
23              //print the numbers on screen
24              cout<<"First Number is:"<<num1<<endl;
25              cout<<"Second Number is:"<<num2<<endl;
26              return 0;
27          }
28          int get_show(){
29              cout<<"Enter Third Number:";
30              cin>>num3;
31              cout<<"Enter Fourth Number:";
32              cin>>num4;
33              //print the numbers on screen
34              cout<<"Third Number is:"<<num3<<endl;
35              cout<<"Fourth Number is:"<<num4<<endl;
36              return 0;
37          }
38  };
39  //Main Class
40  main(){
41      Child obj;
42      obj.show();
43      obj.get_show();
44  }
45
```

Output:

```
C:\Users\jayan\Desktop\c++.(
Enter First Number:10
Enter Second Number:20
First Number is:10
Second Number is:20
Enter Third Number:30
Enter Fourth Number:40
Third Number is:30
Fourth Number is:40

--------------------------------
Process exited after 7.827 seconds with return value 0
Press any key to continue . . .
```

Hierarchical Inheritance

Hierarchical inheritance is a kind of inheritance where more than one class is inherited from a single parent class.




```
#include<iostream>
using namespace std;
//Base Class
class Base{
    public:
        int add(int num1,int num2);
        int sub(int num3,int num4);
};
//Derived Class 1
class Derived1 : public Base{
    public:
        int add(int num1,int num2){
            int total1=num1+num2;
            cout<<"Addition of Two Numbers is:"<<total1<<endl;
            return 0;
        }
};
//Derived Class 2
class Derived2 : public Base{
    public:
        int sub(int num3,int num4){
            int total2=num3-num4;
            cout<<"Subtraction of Two Numbers is:"<<total2<<endl;
            return 0;
        }
```

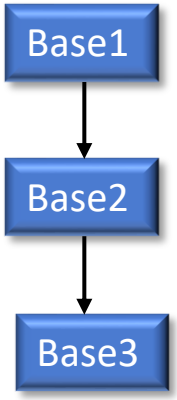
```
        int sub(int num3,int num4){
            int total2=num3-num4;
            cout<<"Subtraction of Two Numbers is:"<<total2<<endl;
            return 0;
        }
};
//Main Function
int main(){
    //Create Object
    Derived1 obj1;
    Derived2 obj2;
    //Call the functions through objects
    obj1.add(10,20);
    obj2.sub(20,10);
    return 0;
}
```

Output:

```
C:\Users\jayan\Desktop\c++.
Addition of Two Numbers is:30
Subtraction of Two Numbers is:10

--------------------------------
Process exited after 0.4532 seconds with return value 0
Press any key to continue . . .
```

Multilevel Inheritance:

In this heritance, user can make different classes as (parent) without create Derived classes.



c++.cpp

```cpp
#include<iostream>
using namespace std;
//Base Class 1
class Base1{
    public:
    int num1=10;
};
//Base Class 2
class Base2{
    public:
    int num2=20;
};
//Base Class 3
class Base3{
    public:
    int num3=30;
};
//Main Function
int main(){
    Base1 obj1;
    Base2 obj2;
    Base3 obj3;
    cout<<"Number of Base Class 1 is:"<<obj1.num1<<endl;
    cout<<"Number of Base Class 2 is:"<<obj2.num2<<endl;
    cout<<"Number of Base Class 3 is:"<<obj3.num3<<endl;

    return 0;
}
```

Compiler (3) | Resources | Compile Log | Debug | Find Results | Close

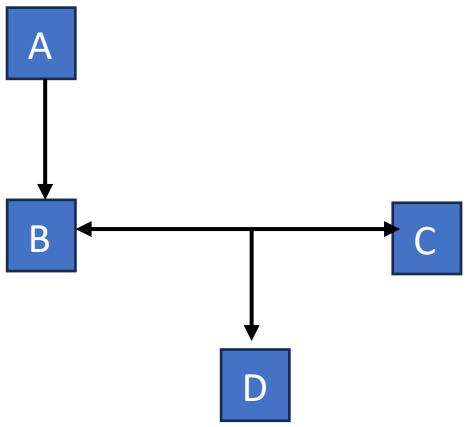
Line	Col	File	Message
6	11	C:\Users\jayan\Desktop\c++.cpp	[Warning] non-static data member initializers only available with -std=c++11 or -std=gnu++11
11	11	C:\Users\jayan\Desktop\c++.cpp	[Warning] non-static data member initializers only available with -std=c++11 or -std=gnu++11
16	11	C:\Users\jayan\Desktop\c++.cpp	[Warning] non-static data member initializers only available with -std=c++11 or -std=gnu++11

Output:

```
C:\Users\jayan\Desktop\c++.(
Number of Base Class 1 is:10
Number of Base Class 2 is:20
Number of Base Class 3 is:30

--------------------------------
Process exited after 0.6266 seconds with return value 0
Press any key to continue . . .
```

Hybrid Inheritance:

In Hybrid Inheritance, multiple types of inheritance are combined within a single class hierarchy, enabling a varied and flexible structure of classes. In hybrid inheritance, within the same class, we can have elements of single inheritance, multiple inheritance, multilevel inheritance, and hierarchical inheritance



```cpp
#include<iostream>
using namespace std;
//Base Class 1
class Base{
    public:
    int base1();
};
//Child Class 1 or Base 2 class
class Child1 : public Base{
    public:
        int num1=10;
        int base1(){
            cout<<"This is First Base Class"<<endl
        }
};
//Base 3 class
class Base3{
    public:
        int num2=20;
};
//Child class 2
class Child2 : public Child1,public Base3{
    public:
        void showData(){
            cout<<"First Number is:"<<num1<<endl;
```

```
21  //Child class 2
22  class Child2 : public Child1,public Base3{
23      public:
24          void showData(){
25              cout<<"First Number is:"<<num1<<endl;
26              cout<<"Second Number is:"<<num2<<endl;
27          }
28  };
29  //Main class
30  main(){
31      Child1 obj;
32      Child2 obj2;
33      obj.base1();
34      obj2.showData();
35  }
```

Compiler (2) | Resources | Compile Log | Debug | Find Results | Close

Line	Col	File	Message
11	12	C:\Users\jayan\Desktop\c++.cpp	[Warning] non-static data member initializers only available with -std=c++11 or -std=gnu++11
19	12	C:\Users\jayan\Desktop\c++.cpp	[Warning] non-static data member initializers only available with -std=c++11 or -std=gnu++11

Output:

```
C:\Users\jayan\Desktop\c++.
This is First Base Class
First Number is:10
Second Number is:20

--------------------------------
Process exited after 0.5858 seconds with return value 0
Press any key to continue . . .
```

Scope resolution operator

The scope resolution operator is used to reference the global variable or member function that is out of scope. Therefore, we use the scope resolution operator to access the hidden variable or function of a program. The operator is represented as the double colon (::) symbol.

Uses of Scope Resolution Operator

(i)It is used to access the hidden variables or member functions of a program.

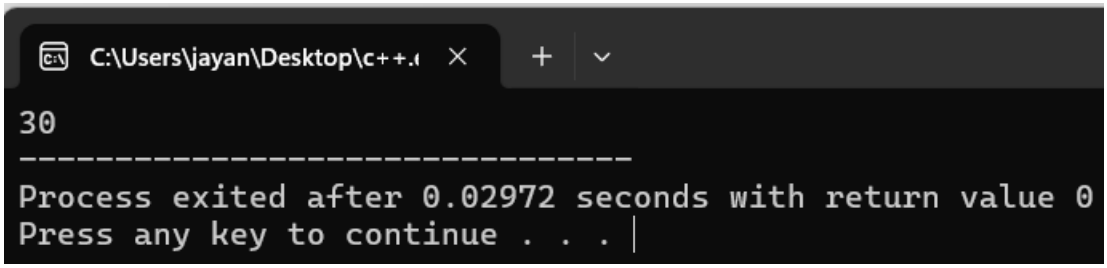
(ii)It defines the member function outside of the class using the scope resolution.

(iii)It is used to access the static variable and static function of a class.

(iv)The scope resolution operator is used to override function in the Inheritance.

```
#include<iostream>
using namespace std;
class Scope_Resolution{
    public:
        int add(int num1,int num2);
};
int Scope_Resolution::add(int num1,int num2){
    int total=num1+num2;
    cout<<total;
}
main(){
    Scope_Resolution obj;
    obj.add(20,10);
}
```

Output:


C:\Users\jayan\Desktop\c++.
30

Process exited after 0.02972 seconds with return value 0
Press any key to continue . . .

Access Specifiers

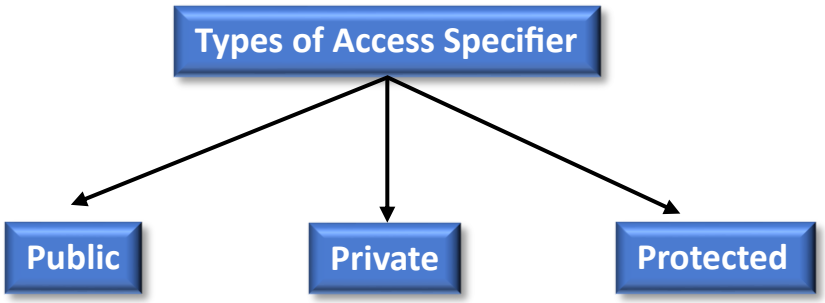
(i)Access Specifier is used for data hiding and it's also known as Access Modifier.

(ii)Data hiding refers to restricting access to data members of a class. This is to prevent other functions and classes from tampering with the class data.

(iii)However, it is also important to make some member functions and member data accessible so that the hidden data can be manipulated indirectly.

(iv)The access modifiers of C++ allow us to determine which class members are accessible to other classes and functions, and which are not.

(v)By default private access specifier will be executed in class.



Public:

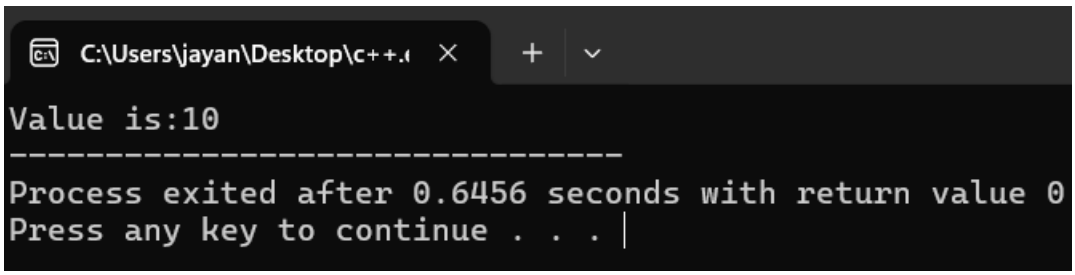
(i)The public keyword is used to create public members (data and functions).

(ii)The public members are accessible from any part of the program.

c++.cpp

```cpp
#include<iostream>
using namespace std;
//Class
class A{
    public:
        int a=10;
};
//Main function;
main(){
    A obj;
    cout<<"Value is:"<<obj.a;
}
```

Output:



Private:

(i)The private keyword is used to create private members (data and functions).

(ii)The private members can only be accessed from within the class.

(iii)However, friend classes and friend functions can access private members.

c++.cpp

```
2  using namespace std;
3  //Class
4  class A{
5      private:
6          int a=10;
7      public:
8          void value(){
9              cout<<"Value of A is:"<<a;
10         }
11
12 };
13 //Main function;
14 main(){
15     A obj;
16     obj.value();
17 }
```

Output:

```
C:\Users\jayan\Desktop\c++.(
Value of A is:10
--------------------------------
Process exited after 0.5246 seconds with return value 0
Press any key to continue . . .
```

Protected:

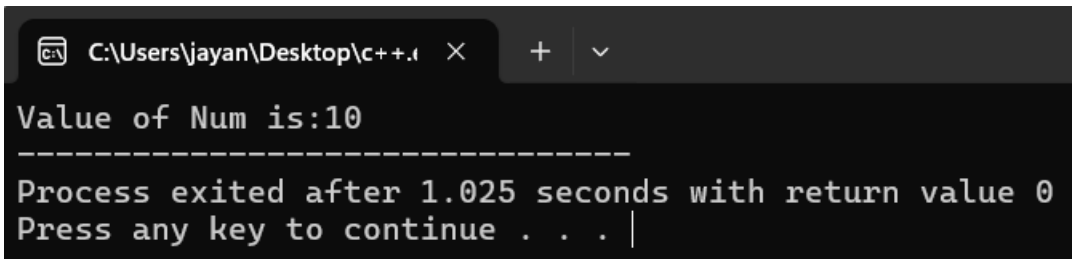
(i)The protected keyword is used to create protected members (data and function).

(ii)The protected members can be accessed within the class and from the derived class.

[*] c++.cpp

```cpp
#include<iostream>
using namespace std;
//Class
class A{
    protected:
        int num=10;
};
class B : protected A{
    public:
        void showData(){
            cout<<"Value of Num is:"<<num;
        }
};
//Main function;
main(){
    B obj;
    obj.showData();
}
```

Output:

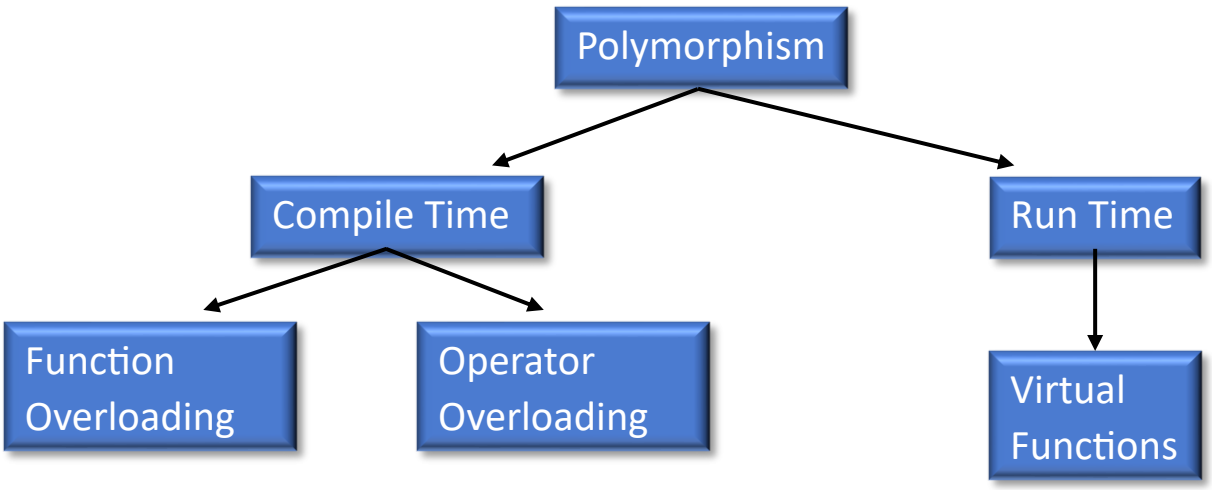

C:\Users\jayan\Desktop\c++.
Value of Num is:10

Process exited after 1.025 seconds with return value 0
Press any key to continue . . .

Polymorphism

(i)Poly means many or morphism means forms.

(ii)Polymorphism refers to the ability to call different functions by using only one type of function call.



Function Overloading:

(i)When there are multiple functions with the same name but different parameters, then the functions are said to be overloaded, hence this is known as Function Overloading.

(ii)Functions can be overloaded by changing the number of arguments or changing the type of arguments.

(iii) In simple terms, it is a feature of object-oriented programming providing many functions that have the same name but distinct parameters when numerous tasks are listed under one function name.

Rules of Function Overloading:

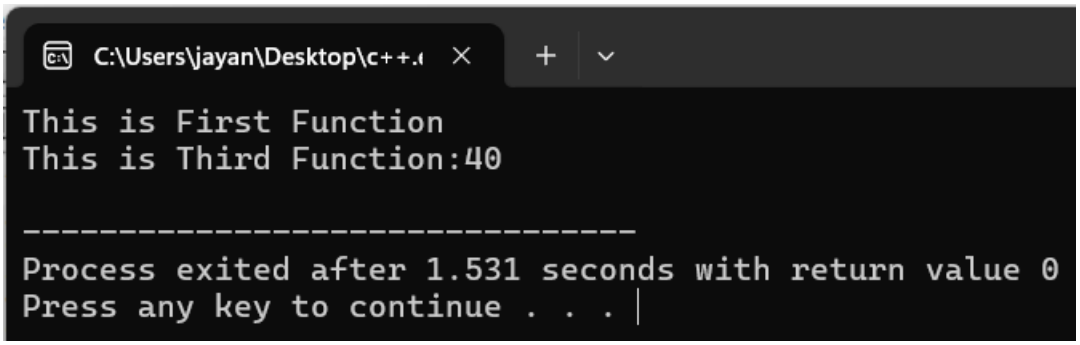
(i)Function name must be same.

(ii)Number of parameters or data types of parameters can be changeable.

c++.cpp

```cpp
#include<iostream>
using namespace std;
//Create Functions
void fun(){
    cout<<"This is First Function"<<endl;
}

void fun(int a){
    cout<<"This is Second Function:"<<a<<endl;
}

void fun(int a,int b){
    cout<<"This is Third Function:"<<a+b<<endl;
}
//Main Function
main(){
    fun();
    fun(30,10);
}
```

Output:



Operator Overloading:

C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. Operator overloading is a compile-time polymorphism.

For example: we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +. Other example classes where arithmetic operators may be overloaded are Complex Numbers, Fractional Numbers, Big integers, etc.

Rules for operator overloading:

(i)Existing operators can only be overloaded, but the new operators cannot be overloaded.

(ii)The overloaded operator contains at least one operand of the user-defined data type.

(iii)We cannot use friend function to overload certain operators. However, the member function can be used to overload those operators.

(iv)When unary operators are overloaded through a member function take no explicit arguments, but, if they are overloaded by a friend function, takes one argument.

(iv)When binary operators are overloaded through a member function takes one explicit argument, and if they are overloaded through a friend function takes two explicit arguments.

[*] c++.cpp

```cpp
#include <iostream>
using namespace std;
class A
{
    int x;
    public:
      A(){}
    A(int i)
    {
       x=i;
    }
    void operator+(A);
    void display();
};
void A :: operator+(A a)
{
    int m = x+a.x;
    cout<<"The result of the addition of two objects is : "<<m;
}
int main()
{   A a1(5);
    A a2(4);
    a1+a2;
    return 0;
}
```

Output:

```
C:\Users\jayan\Desktop\c++.(
The result of the addition of two objects is : 9
--------------------------------
Process exited after 0.8991 seconds with return value 0
Press any key to continue . . .
```

Virtual Functions:

(i)Virtual functions are public member functions, used for special purpose. These are defined in base class and again are re-defined in the derived class.

(ii)They are mainly used to achieve runtime polymorphism.

(iii)Virtual functions are declared with a virtual keyword in a base class.

Rules of virtual functions:

(i)Virtual functions cannot be static.

(ii)A virtual function can be a friend function of another class.

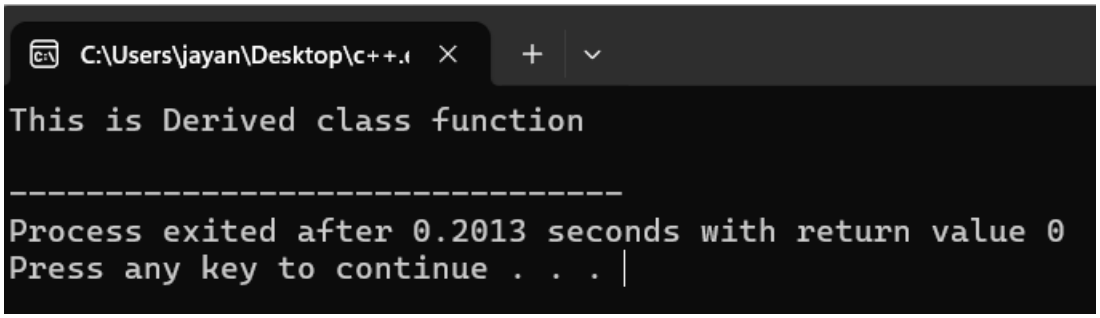
(iii)Virtual functions should be accessed using a pointer or reference of base class type to achieve runtime polymorphism.

(iv)A class many have a virtual destructor but it cannot have a virtual constructor.

c++.cpp

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

class Base{
    public:
        virtual void print(){
            cout<<"This is base class function"<<endl;
        }
};
class Derived : public Base{
    public:
        void print(){
            cout<<"This is Derived class function"<<endl;
        }
};
//Main class
int main(){
    Base* print;
    Derived obj;
    print=&obj;
    print->print();//function call
    return 0;
}
```

Output:


C:\Users\jayan\Desktop\c++.
This is Derived class function

Process exited after 0.2013 seconds with return value 0
Press any key to continue . . .

This pointer

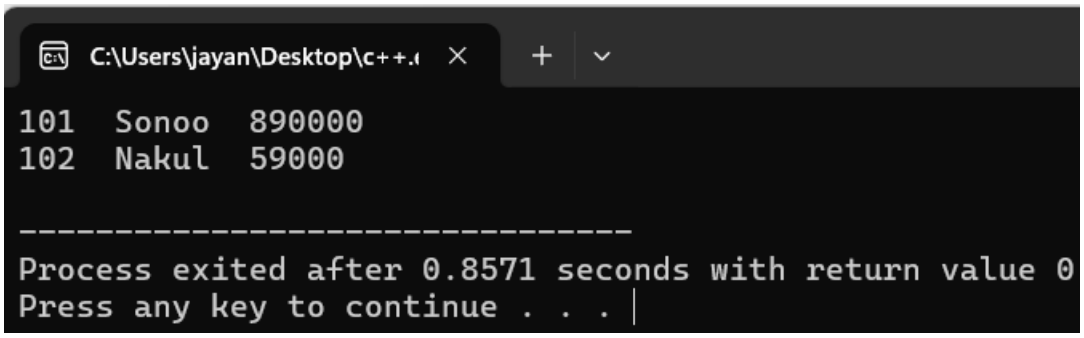
Every object in C++ has access to its own address through an important pointer called this pointer. The (this) pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.

Friend functions do not have a (this) pointer, because friends are not members of a class. Only member functions have a (this) pointer.

c++.cpp

```cpp
#include <iostream>
using namespace std;
class Employee {
   public:
       int id; //data member (also instance variable)
       string name; //data member(also instance variable)
       float salary;
       Employee(int id, string name, float salary)
        {
            this->id = id;
           this->name = name;
           this->salary = salary;
        }
       void display()
        {
           cout<<id<<"  "<<name<<"  "<<salary<<endl;
        }
};
int main(void) {
    Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee
    Employee e2=Employee(102, "Nakul", 59000); //creating an object of Employee
    e1.display();
    e2.display();
    return 0;
}
```

Output:


C:\Users\jayan\Desktop\c++.
101 Sonoo 890000
102 Nakul 59000

Process exited after 0.8571 seconds with return value 0
Press any key to continue . . .

New and Delete Operator

(i)In C++ programming language, the new and delete operators are mainly utilized for dynamic memory allocation and deallocation. They enable us to dynamically allocate and free the memory, which means that we can create objects whose size and lifetime may be defined at runtime. However, there are some main distinctions between the two operators that we should be aware of.

(ii) The new operator is mainly utilized to allocate memory for an object or an array of objects. It returns a pointer to the allocated memory. On the other hand, the delete operator is mainly utilized to deallocate memory that was previously allocated with new. It frees the memory pointed to by the pointer and sets the pointer to null.

Syntax (New Operator)

pointer_variable = new data_type;

Syntax (Delete Operator)

delete pointer_variable;

Initialization

The new operator initializes the memory allocated for an object with the default constructor of the class. If there is no default constructor, we may utilize the placement new operator to call the appropriate constructor. The delete operator does not perform any initialization.

Allocation

The new operator may be mainly utilized to allocate memory for an array of objects. The syntax for array allocation is:

1. pointer_variable = new data_type[array_size];

On the other hand, the delete operator may be mainly utilized to deallocate an array of objects allocated with the new operator. The syntax for array deallocation is:

delete[] pointer_variable;

New Operator

c++.cpp

```cpp
#include <iostream>
using namespace std;
int main () {
   int *ptr1 = NULL;
   ptr1 = new int;
   float *ptr2 = new float(223.324);
   int *ptr3 = new int[28];
   *ptr1 = 28;
   cout << "Value of pointer variable 1 : " << *ptr1 << endl;
   cout << "Value of pointer variable 2 : " << *ptr2 << endl;
   if (!ptr3)
   cout << "Allocation of memory failed\n";
   else {
      for (int i = 10; i < 15; i++)
      ptr3[i] = i+1;
      cout << "Value of store in block of memory: ";
      for (int i = 10; i < 15; i++)
      cout << ptr3[i] << " ";
   }
   return 0;
}
```

Output:

```
C:\Users\jayan\Desktop\c++.
Value of pointer variable 1 : 28
Value of pointer variable 2 : 223.324
Value of store in block of memory: 11 12 13 14 15
--------------------------------
Process exited after 0.6354 seconds with return value 0
Press any key to continue . . .
```

Delete Operator:

c++.cpp

```cpp
#include <iostream>
using namespace std;
int main () {
   int *ptr1 = NULL;
   ptr1 = new int;
   float *ptr2 = new float(299.121);
   int *ptr3 = new int[28];
   *ptr1 = 28;
   cout << "Value of pointer variable 1 : " << *ptr1 << endl;
   cout << "Value of pointer variable 2 : " << *ptr2 << endl;
   if (!ptr3)
   cout << "Allocation of memory failed\n";
   else {
      for (int i = 10; i < 15; i++)
      ptr3[i] = i+1;
      cout << "Value of store in block of memory: ";
      for (int i = 10; i < 15; i++)
      cout << ptr3[i] << " ";
   }
   delete ptr1;
   delete ptr2;
   delete[] ptr3;
   return 0;
}
```

Output:

```
C:\Users\jayan\Desktop\c++.
Value of pointer variable 1 : 28
Value of pointer variable 2 : 299.121
Value of store in block of memory: 11 12 13 14 15
--------------------------------
Process exited after 1.476 seconds with return value 0
Press any key to continue . . .
```

Files Handling

(i)File handling is used to store data permanently in a computer. Using file handling we can store our data in secondary memory (Hard disk).

(ii)In C++ programming we are using the iostream standard library, it provides cin and cout methods for reading from input and writing to output respectively.

(iii)To read and write from a file we are using the standard C++ library called fstream

(iv)File handling in C++ is a mechanism to create and perform read/write operations on a file.

Operations of File Handling(fstream):

Sr.No	Data Type & Description
1	ofstream This data type represents the output file stream and is used to create files and to write information to files.
2	ifstream This data type represents the input file stream and is used to read information from files.
3	fstream This data type represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files.

Steps of File Handling:

Step1: Naming a file
Step2: Opening a file
Step3: Writing data into the file
Step4: Reading data from the file
Step5: Closing a file.

Flags:

ios::in	Open a file for reading.
ios::out	Open a file for writing.
ios::binary	Open a file in binary instead of text.
ios::ate	File pointer points at the end of file after opening.
ios::app	Appends the new content at the end of file
ios::trunc	If files exists, all of its content is discarded.

Functions of File Handling:

No.	Function	Description
1	fopen()	opens new or existing file
2	fprintf()	write data into the file
3	fscanf()	reads data from the file
4	fputc()	writes a character into the file
5	fgetc()	reads a character from file
6	fclose()	closes the file
7	fseek()	sets the file pointer to given position

Modes of File Handling

Mode	Description
r	opens a text file in read mode
w	opens a text file in write mode
a	opens a text file in append mode
r+	opens a text file in read and write mode

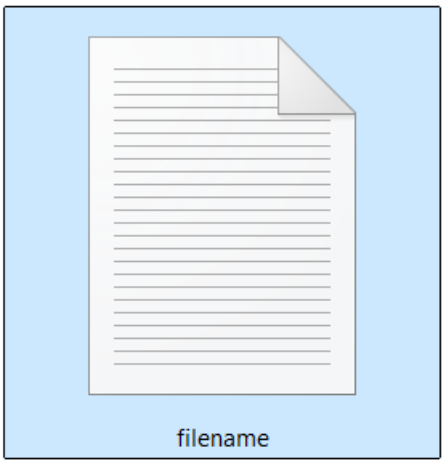
w+	opens a text file in read and write mode
a+	opens a text file in read and write mode
rb	opens a binary file in read mode
wb	opens a binary file in write mode
ab	opens a binary file in append mode
rb+	opens a binary file in read and write mode
wb+	opens a binary file in read and write mode
ab+	opens a binary file in read and write mode

Create and Write a file

c++.cpp

```cpp
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  // Create and open a text file
  ofstream MyFile("C:/Users/jayan/Desktop/filename.txt");

  // Write to the file
  MyFile << "Files can be tricky, but it is fun enough!";

  // Close the file
  MyFile.close();
}
```

Output:



Note: Go to your file location where you saved, then open this file for show content.

Reading Content from file

```cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main () {
  // Create a text file
  ofstream MyWriteFile("C:/Users/jayan/Desktop/filename.txt");
  // Write to the file
  MyWriteFile << "Files can be tricky, but it is fun enough!";
  // Close the file
  MyWriteFile.close();
  // Create a text string, which is used to output the text file
  string myText;
  // Read from the text file
  ifstream MyReadFile("filename.txt");
  // Use a while loop together with the getline() function to read the file line by line
  while (getline (MyReadFile, myText)) {
    // Output the text from the file
    cout << myText;
  }
  // Close the file
  MyReadFile.close();
}
```

Output:

```
Files can be tricky, but it is fun enough!
--------------------------------
Process exited after 1.012 seconds with return value 0
Press any key to continue . . .
```

Exception Handling

(i)An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

(ii)Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch, and throw.

Try	A **try** block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.
Catch	A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The **catch** keyword indicates the catching of an exception.
Throw	A program throws an exception when a problem shows up. This is done using a **throw** keyword.

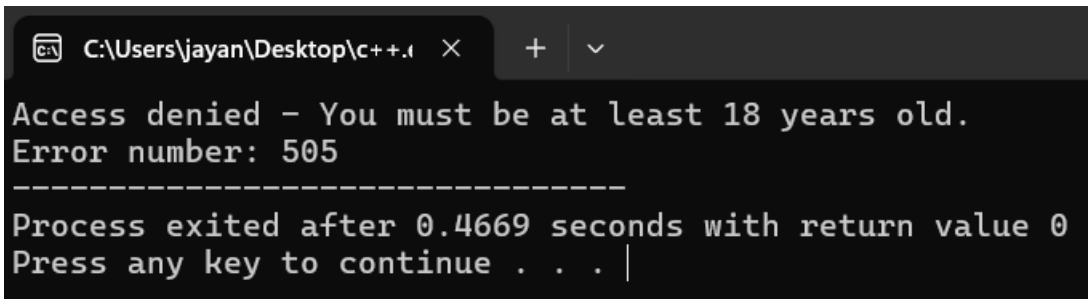
Syntax:

```
try {
   // protected code
} catch( ExceptionName e1 ) {
   // catch block
} catch( ExceptionName e2 ) {
   // catch block
} catch( ExceptionName eN ) {
   // catch block
}
```

c++.cpp

```cpp
#include <iostream>
using namespace std;
//Main Function
int main() {
  try {
    int age = 15;
    if (age >= 18) {
      cout << "Access granted - you are old enough.";
    } else {
      throw 505;
    }
  }
  catch (int myNum) {
    cout << "Access denied - You must be at least 18 years old.\n";
    cout << "Error number: " << myNum;
  }
  return 0;
}

```

Output



Friend Function

A friend class can access private and protected members of other classes in which it is declared as a friend. It is sometimes useful to allow a particular class to access private and protected members of other classes.

Syntax:

```
class class_name
{
    friend data_type function_name(argument/s);
}
```

Characteristics of Friend Function:

(i)The function is not in the scope of the class to which it has been declared as a friend.

(ii)It cannot be called using the object as it is not in the scope of that class.

(iii)It can be invoked like a normal function without using the object.

(iv)It cannot access the member names directly and has to use an object name and dot membership operator with the member name.

(v)It can be declared either in the private or the public part.

```cpp
#include <iostream>
using namespace std;
class Distance {
    private:
        int meter;
        // friend function
        friend int addFive(Distance);
    public:
        Distance() : meter(0) {}

};
// friend function definition
int addFive(Distance d) {
    //accessing private members from the friend function
    d.meter += 5;
    return d.meter;
}
int main() {
    Distance D;
    cout << "Distance: " << addFive(D);
    return 0;
}
```

Output:

```
C:\Users\jayan\Desktop\c++.
Distance: 5
--------------------------------
Process exited after 0.5577 seconds with return value 0
Press any key to continue . . .
```