Unit

2

C++ PROGRAMMING BASICS

Lesson Structure

2.0 Objective

After going through this unit you will understand:

- Data Types and Manipulators in C++
- Different types of Arithmetic operators in C++
- Library functions in C++
- Structure of a C++ program
- Components of a basic C++ program

2.1 Introduction

Programs in C++ are simple and well structured. There are some basic components in every C++ program that we write. There are some special lines called Preprocessor Directives that are interpreted even before the compilation of the program begins. Programmers can give comments within their programs to make the statements understandable to novice users. The Standard Library of C++ consists of classes and functions. These classes and functions are written in the core language. They even form part of the C++ ISO Standard. Data Types in C++ are used by users to declare variables. Primary Data Types in C++ are Integer, Character, Boolean, Floating Point etc. C++ uses Datatype Modifiers to modify the length of data that a particular data type can hold. Datatype Modifiers are used with the built-in data types. In C++, data can be manipulated according to the choice of display of the programmer. To format the output, special operators called Manipulators are used.

The rest of the unit is organized as follows. Section 2.2 describes the structure of a basic C++ program. Sections 2.3 and 2.4 explain C++ Data Types and Manipulators respectively. Sections 2.5 and 2.6 explain C++ Arithmetic operators and Library functions respectively. Section 2.7 explains the preprocessor directives. Section 2.8 gives a brief overview of the unit. Section 2.9 contains some questions for the students and section 2.10 gives the references for the unit.

2.2 Basic C++ Program Construction

Let us start with a simple program that can help us to understand the fundamental components of a C++ program. The following program prints the text as shown in the Output below.

```
1.      // First C++ Program
2.      #include <iostream>
3.
4.      using namespace std;
5.      int main ()
6.      {
7.      cout << "Hello World! ";
8.      cout << " This is my first C++ Program ";
9.      return 0;
10.     }
```

OUTPUT: Hello world! This is my first C++ Program.

Let us analyze the program one statement at a time.

1^{st} Line: // First C++ Program

The first statement in the program is just a comment inserted by the programmer within the program. A comment line is indicated by two leading slash signs as shown here. Comments have no effect on the behavior of the program. This is known as single line comment. Multi line comment is shown as below:

/* This is my first program.

I will learn how to write C++ program and how to compile the same*/

2nd Line: #include <iostream>

Lines that begin with hash signs (#) are called preprocessor directives. These special lines are interpreted even before the program compilation starts. The preprocessor directive #include <iostream> informs the preprocessor to include the header iostream, which is a part of the standard C++ code. The header iostream permits the accomplishment of the standard input and output operations, for e.g., in this case it prints the output on the computer screen.

3rd Line: A blank line.

There is no effect of blank lines on the program. It just makes the program easy to read.

4th Line: using namespace std;

Suppose the code that you are writing has the function abc() in it. Similarly, there is another library that has the same function abc(). The compiler is not aware that which abc() function you are referring to in your code.

A namespace solves this problem and it is used as an extra information to distinguish between similar classes, functions, variables etc. that have the same name but are available in different libraries. Namespace basically defines the scope.

In C++, namespaces are used to define a scope. They make it possible to group global classes, objects and/or functions into a single group. Through *using namespace std;*, C++ compiler is instructed to use the standard C++ library. If this instruction is not used, then each time you use a standard C++ function or entity, you have to use std::.

Like without using namespace same program will be as below:

```
// First program in C++
#include <iostream>

int main ()
{
std::cout << "Hello World! ";
std::cout << " This is my first C++ Program ";
return 0;
}
```

5th Line: int main ()

This line initiates the declaration of a function. A function is nothing but a collection of statements that is given a name. In this case the function is given the name “main”. The declaration of a function is preceded by a type (int), followed by a name (main) and a pair of parentheses (()). Optionally parameters can be included within the parentheses.

The main function is a special function that is called automatically. The code in any other function is executed only if that function is called from the main function (directly or indirectly). For all C++ programs, the execution begins with the main function. It does not matter where the main function is located in the program.

6th and 10th Line: { and }

The body of a function is enclosed within an open brace ({) and a closing brace (}). It is defined within these braces that what actually happens when the function is called. In the above program, the body of the main function is enclosed within the open brace at line 6 and the closing brace at line 10.

7th Line: cout << "Hello World!";

8th Line: cout << " This is my first C++ Program ";

Lines 7 and 8 are C++ statements. Statements are executed in the exact order in which they appear inside the body of a function.

The statements consist of three parts: The first part, i.e. cout, recognizes the standard character output device (generally the computer screen). The second part, i.e. insertion operator (<<), specifies that anything that follows the insertion operator is inserted into cout. The third part, i.e. a sentence within quotes (for e.g. “Hello World!”) is the content that is to be inserted into the standard output.

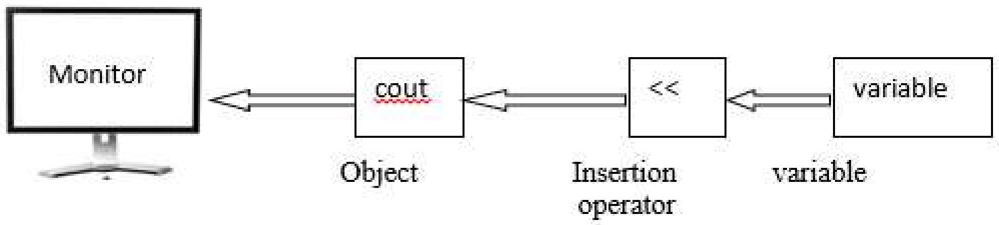


Figure 2.1: Usage of cout

The reader should notice that the statement ends with a semicolon (;). In fact, all the statements in C++ should end with a semicolon character. A common mistake by programmers is leaving the semicolon at the end of statements.

9th Line: return 0;

The task of the return statement is to send a status report to the OS about the program execution. It informs the OS whether the program has executed properly or not.

Program: print the given lines on computer screen.

I am 18 years old.

I am student of Polytechnic.

Solution:

```
#include <iostream>
```

```
using namespace std;
int main(int argc, char *argv[])
{
        int age;
        age=18;
        cout<<" I am "<<age<<" years old.\n";
        cout<<" I am student of Polytechnic.\n";
        return 0;
}
```

Program: write first program using class

```
#include <iostream>
using namespace std;
class Test      //class declaration start with keyword class and then name of the class
{
        int i;                  //data variable
        public:
        void show()             //Member Function
        {
                cout<<"Inside Member Function";
        }
}; // end of Class
int main()
{
        Test obj;               // Creating object of Abc class
        obj.show();             // Use of class object to call member function
}
```

Output:

```
Inside Member Function
```

How to define a class in C++?

To define a class in C++, we use the keyword class, which is followed by the class name. The class body is enclosed within the opening and closing curly brackets. The closing curly bracket is followed by a semicolon.

```
class ClassName
  {
  // data
  // functions
  };
```

How to create object in C++?

Syntax to define the object in main function.

```
className cn; //cn is the object
```

This is very similar to declaration of variable in C / C++.

How to access data member and member function in C++?

The dot operator (.) is used to access the data members and member functions. For e.g.,

```
obj2.func1();
```

The above statement will call the function func1() inside the Test class for object obj2.

To access the data member, the following statement is used:

```
obj1.data1 = 7.3;
```

The reader should note that if he wants to access private members, then it can be done from within the class only.

Program: usage of classes and objects in C++

```
#include <iostream>
using namespace std;
class Sample
{
      private:
         int data_1;
         float data_2;
      public:
         void insertIntegerNumber(int d_1)
         {
                  data_1 = d_1;
                  cout << "Number: " << data_1;
         }
         float insertFloatingNumber()
         {
                  cout << "\nEnter data: ";
                  cin >> data_2;
                  return data_2;
```

```
		}
	};
	int main()
	{
		Sample obj1, obj2;
		float f;
		obj1.insertIntegerNumber(15);
		f = obj2.insertFloatingNumber();
		cout << "You entered " << f;
		return 0;
	}
```

Output:

```
	Number: 15
	Enter data: 23.3
	You entered 23.3
```

2.3 C++ Data Types

There are two types of C++ Data Types:

Primitive Data Types: Primitive Data Types are predefined or built-in data types that can be used by the user directly for variable declaration. In C++, the following primitive data types are there:

- Character
- Floating Point
- Integer
- Valueless or Void
- Boolean
- Double Floating Point
- Wide Character

Abstract or user defined data type: Abstract data types are defined by the user himself.

The definitions of the primitive data types have been give below:

- ***Character***: The data type *Character* stores characters. The keyword that is used for this data type is *char*. Each Character generally occupies a memory space of 1 byte. The range is from -128 to 127 or 0 to 255.
- ***Floating Point***: The data type Floating Point stores decimal values or single precision floating point values. The keyword that is used for this data type is *float*. Each Float variable generally occupies a memory space of 4 bytes.
- ***Integer***: The keyword that is used for this data type is *int*. Each Integer generally occupies a memory space of 4 bytes. The range is from -2147483648 to 2147483647.

- ***Void***: This data type means that there is no value. This data type is used for the functions that do not return any value.
- ***Boolean***: The data type Boolean stores logical values or boolean. The two values that can be stored in a boolean variable is either true or false. The Keyword that is used for data type boolean is *bool*.
- ***Double Floating Point***: The data type Double Floating Point stores decimal values or double precision floating point values. The keyword that is used for this data type is *double*. Each Double variable generally occupies a memory space of 8 bytes.
- ***Wide Character***: The data type Wide character is also a character data type. But it generally occupies 2 or 4 bytes of memory space. The keyword that is used for this data type is *wchar_t*.

Datatype Modifiers: In C++, Datatype Modifiers are used along with Primitive data types to alter the data length that can be held by a specific data type. Following are the Datatype modifiers in C++:

- Short
- Long
- Signed
- Unsigned

Table 1.1 shows the altered size and range of built-in data types when used in combination with datatype modifiers.

Table 1.1: Size and Range of Data Types

Data Type	Size (In Bytes)	Range
Int	4	-2,147,483,648 to 2,147,483,647
unsigned short int	2	0 to 65,535
short int	2	-32,768 to 32,767
unsigned int	4	0 to 4,294,967,295
unsigned long int	4	0 to 4,294,967,295
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long long int	8	0 to 18,446,744,073,709,551,615

long long int	8	-(2^63) to (2^63)-1
Float	4	
double	8	
long double	12	
unsigned char	1	0 to 255
signed char	1	-128 to 127
wchar_t	2 or 4	1 wide character

2.4 C++ Manipulators

C++ offers several input/output manipulators for formatting, commonly used manipulators are given below:

endl

endl manipulator is used to Terminate a line and flushes the buffer.

To produce a newline while writing the output in C++, you can either use ‘\n’ or ‘std::endl’. But each one has a varying outcome.

- std::endl inserts a new line and flushes the output buffer.
- '\n' only inserts a new line but does not flush the output buffer.

When writing debugging messages that need to be seen instantly, you should not use ‘\n’ but should use std::endl so that the flush is forced to happen instantly.

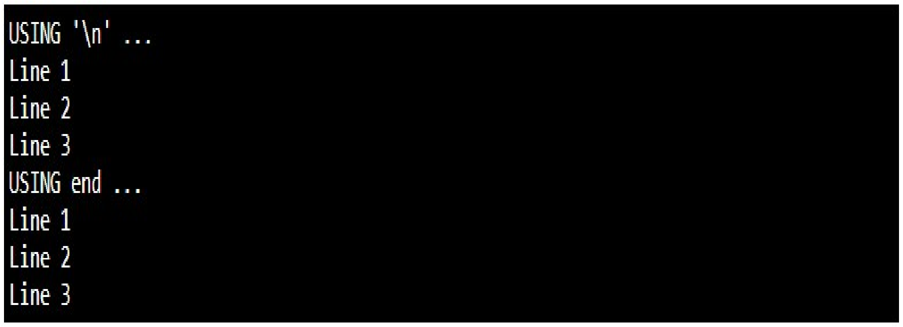
The following example shows how both the versions are used. However, the flushing is not visible in this example.

Program: manipulators in C++

```
#include <iostream.h>

int main()

{

        cout<<"USING '\\n' ...\n";

        cout<<"Line 1 \nLine 2 \nLine 3 \n";
```

```
            cout<<"USING end ..."<< endl;
            cout<< "Line 1" << endl << "Line 2" << endl << "Line 3" << endl;
            return 0;
    }
```

Output:



```
USING '\n' ...
Line 1
Line 2
Line 3
USING end ...
Line 1
Line 2
Line 3
```

setw() and setfill()

setw manipulator sets the width of the filed assigned for the output.

The field width specifies what should be the minimum number of characters that can be written in the output representation. If the field width is more than the standard width of the representation, then *setfill()* is used to pad the representation with fill characters.

Syntax:

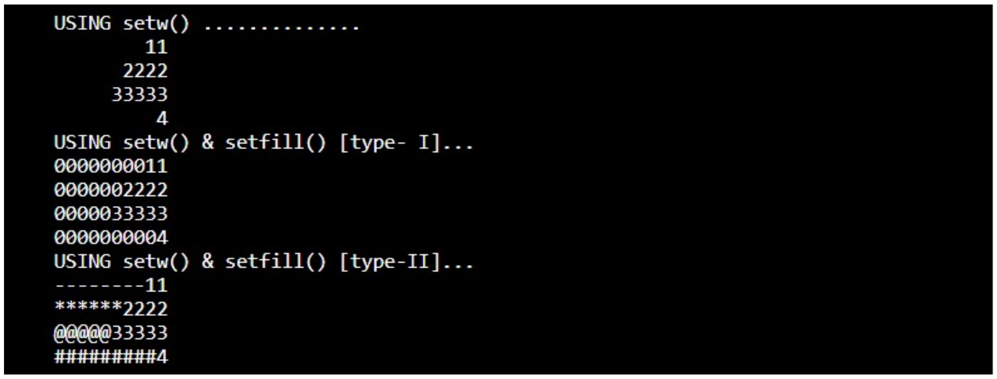
```
setw([number_of_characters]);
setfill([character]);
```

Program: usage of setw() and setfill()

```
#include <iostream.h>
#include <iomanip.h>
int main()
{
   cout<<"USING setw() ..............\n";
   cout<< setw(10) <<11<<"\n";
   cout<< setw(10) <<2222<<"\n";
   cout<< setw(10) <<33333<<"\n";
   cout<< setw(10) <<4<<"\n";

   cout<<"USING setw() & setfill() [type- I]...\n";
   cout<< setfill('0');
```

```
        cout<< setw(10) <<11<<"\n";
        cout<< setw(10) <<2222<<"\n";
        cout<< setw(10) <<33333<<"\n";
        cout<< setw(10) <<4<<"\n";

        cout<<"USING setw() & setfill() [type-II]...\n";
        cout<< setfill('-')<< setw(10) <<11<<"\n";
        cout<< setfill('*')<< setw(10) <<2222<<"\n";
        cout<< setfill('@')<< setw(10) <<33333<<"\n";
        cout<< setfill('#')<< setw(10) <<4<<"\n";
        return 0;
    }
```

Output:



```
USING setw() ..............
        11
      2222
     33333
         4
USING setw() & setfill() [type- I]...
0000000011
0000002222
0000033333
0000000004
USING setw() & setfill() [type-II]...
--------11
******2222
@@@@@33333
#########4
```

setf() and setprecision()

setprecision manipulator sets the total number of digits to be displayed, when floating point numbers are printed.

Syntax:

```
setprecision([number_of_digits]);
cout<<setprecision(5)<<1234.537;
// output will be: 1234.5
```

On the default floating point notation, the precision field specifies the total number of digits that should be displayed as output, including the digits before the decimal point and the digits after the decimal point.

In scientific and fixed notations, the number of digits before the decimal point is of no importance. In this case, the precision field specifies the number of digits after the decimal point. If needed, zeros can be added at the trail.

Syntax:

```
setf([flag_value],[field bitmask]);
```

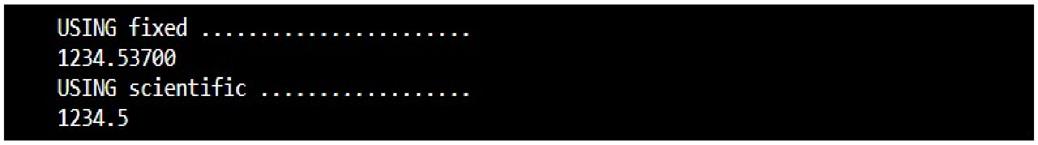
Table 1.2: Field bitmask and Flag values

field bitmask	flag values
adjustfield	left, right or internal
basefield	dec, oct or hex
floatfield	scientific or fixed

Program: usage of setf() and setprecision()

```
#include <iostream.h>
#include <iomanip.h>
int main()
{
        cout<<"USING fixed .......................\n";
        cout.setf(ios::floatfield,ios::fixed);
        cout<< setprecision(5)<<1234.537<< endl;

        cout<<"USING scientific ..................\n";
        cout.setf(ios::floatfield,ios::scientific);
        cout<< setprecision(5)<<1234.537<< endl;
        return 0;
}
```

Output:



```
USING fixed .......................
1234.53700
USING scientific ..................
1234.5
```

2.5 C++ Arithmetic Operators

Table 1.3 shows the arithmetic operators that are used in C++. Suppose that A variable holds the value 10 and B variable holds the value 20.

Table 1.3: Arithmetic Operators in C++

Operator	Description	Example
/	Dividing numerator by the denominator	B / A gives 2
++	Increment operator- increments the value of the integer by one	A++ gives 11
*	Multiplication of both the operands	A * B gives 200
+	Addition of two operands	A + B gives 30
-	Subtracting second operand from first operand	A - B gives -10
%	Modulus operator - remainder of integer division	B % A gives 0
--	Decrement operator- decrements the value of the integer by one	A-- gives 9

2.6 C++ Library Functions

The Standard Library of C++ is composed of two parts.

- **The Standard Function Library** – This library comprises of the basic stand-alone functions that do not form the part of any class. The function library has been inherited from C language.
- **The Object Oriented Class Library** − This library consists of classes and their associated functions.

The Standard Library of C++ also includes all the standard libraries of C, with some modifications and additions to support type safety.

The Standard Function Library

The standard function library can be categorized into the following types:

- I/O - functions for standard I/O
- Mathematical
- Time, date, and localization
- Wide-character functions
- String and character handling
- Dynamic allocation
- Miscellaneous

The Object Oriented Class Library

The Object Oriented Class Library of C++ specifies a set of classes that allow a number of activities like numeric processing and including strings, I/O. Following are the constituents of the Library:

- Exception Handling Classes
- The STL Container Classes
- The Localization library
- The STL Algorithms
- The Standard C++ I/O Classes
- The STL Function Objects
- The String Class
- The STL Iterators
- The Numeric Classes
- The STL Allocators
- Miscellaneous Support Library

2.7 Preprocessor Directives in C++

Preprocessors are programs that process our source code before compilation. In between writing a program and executing a program, a good number of steps are involved. Before understanding Preprocessors, we should look at the following steps first.

.

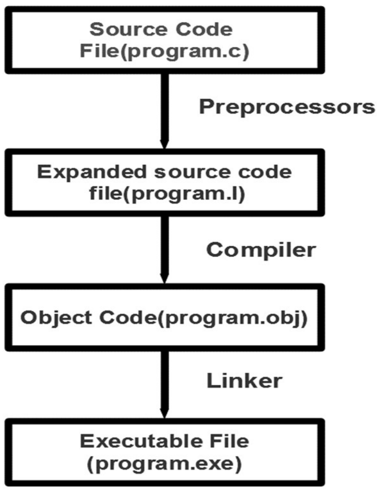

Source Code File(program.c)
Preprocessors
Expanded source code file(program.I)
Compiler
Object Code(program.obj)
Linker
Executable File (program.exe)

Figure 2.2: Usage of Preprocessors

The reader can see the steps involved in Figure 2.2 above. Initially, the programmer writes the source code which is then stored in the file program.c. Next, the file is processed by Preprocessors which leads to the generation of an expanded source code file named program. The compiler then compiles the expanded file leading to the generation of an object code file named program.obj. In the end, the object code file is linked to the object code of the library functions by the linker. This again leads to the generation of an executable file program.exe.

Preprocessors provide preprocessor directives that instruct the compiler to preprocess the source code even before the compilation starts. All preprocessor directives in C++ start with the hash symbol (#). If there is a '#' symbol before a statement in C++, then it means that the statement is a preprocessor directive. Preprocessor directives can be placed at any location in the program. Some examples of preprocessor directives are #define, #include, etc.

Basic types of preprocessor directives are:

1. File Inclusion
2. Conditional Compilation
3. Macros
4. Other directives

1. **File Inclusion**: File Inclusion instructs the compiler to include a file in the program. The user can include two types of files in his program.

a. **Header File or Standard files**: The Header Files consist of the definition of pre-defined functions such as scanf() and printf() etc. For these functions to work, the header files must be included in the program. Different header files contain the declarations for different functions.

   Syntax:

```
#include< file-name >
```

where *file-name* is the name of the file that is to be included. The opening bracket (<) and the closing bracket (>) instruct the compiler to search the file in the standard directory.

b. **User defined files**: It is a good practice to divide a large program into smaller files and include them as and when required. Such files are user defined files and can be included in the following way.

Syntax:

```
#include"filename"
```

2. **Conditional Compilation**: Conditional Compilation allows us to compile a particular part of the program or skip compilation of a particular part of the program on the basis of some conditions. Two preprocessing commands i.e. '**ifdef**' and '**endif**' can help in accomplishing this.

Syntax:

```
ifdef macro-name
        line1;
        line2;
        line3;
        .
        .
        .
        lineN;
endif
```

If the macro named '*macro-name*' is defined, then the lines are normally executed. Otherwise, the compiler skips these lines.

3. **Macros**: A Macro is a piece of code in a program that is given a name. When the compiler sees this name in the program, it replaces the name with the actual piece of code. In C++, to define a macro, the directive '#define' is used. We now look at the following program to understand macro definition.

Program: usage of macro

```
#include<iostream>

// definition of the macro

#define LIMIT 6

int main()
```

```
{
        for(int k=0; k < LIMIT; k++)
        {
                std::cout<<k<<"\n";
        }
        return 0;
}
```

Output:

```
0
1
2
3
4
5
```

4. **Other directives**: Other than the above discussed directives we have two more directives that are used, but not so frequently.

a. **#pragma Directive**: This is a directive that is used for special purpose, i.e. to turn some features on or off. Such directives vary from one compiler to another i.e. they are specific to the compiler being used.

b. **#undef Directive**: This directive undefines an existing macro.

   Syntax:

   ```
   #undef LIMIT
   ```

   This statement undefines LIMIT which is an existing macro. After the execution of the statement "#ifdef LIMIT" will evaluate to false.

2.8 Summary

All C++ programs consist of well-defined components. Comments which are inserted in between statements do not affect the behavior of the program. However, they can help in understanding the meaning of the statements. Preprocessor directives are special lines that are interpreted even before the compilation of the program begins. The Standard Library of C++ consists of classes and functions. These classes and functions are written in the core language. They even form part of the C++ ISO Standard. The main function is a special function that is called automatically. The code in any other function is executed only if that function is called from the main function (directly or indirectly). For all C++ programs, the execution begins with the main function. It does not matter where the main function is located in the program. To define a class in C++, we use the keyword class, which is followed by the class name. Data members and member functions can be accessed by using a dot operator (.). Data Types in C++ can be divided into two types: Primitive Data Types and Abstract Data Types. Primitive Data Types are predefined data types or built-in data types used directly by the users for variable declaration. Abstract Data Types are defined by the user himself. In C++, Manipulators are used to format the output. The output can be designed to match the user's choice of display

2.9 Questions

1. Write a simple C++ program that takes the user's name and age as input and prints it on the screen.
2. What do you mean by Preprocessor Directives?
3. What is the purpose of C++ Standard Library?
4. Why is the main function called the Starting function?
5. Give an example to show how data members and member functions are accessed in C++.
6. Briefly explain the Primary Data Types in C++.
7. Explain the following C++ Manipulators - setfill() and setprecision().
8. Describe the different types of Arithmetic Operators in C++.
9. What are Datatype Modifiers? Explain their usage.
10. Give an example to show how Objects are created in C++.

2.10 Suggested Readings

1. Lafore, Robert. Object-oriented programming in C++. Pearson Education, 1997.
2. https://www.tutorialspoint.com/cplusplus/cpp_standard_library.htm
3. https://www.geeksforgeeks.org/c-data-types/