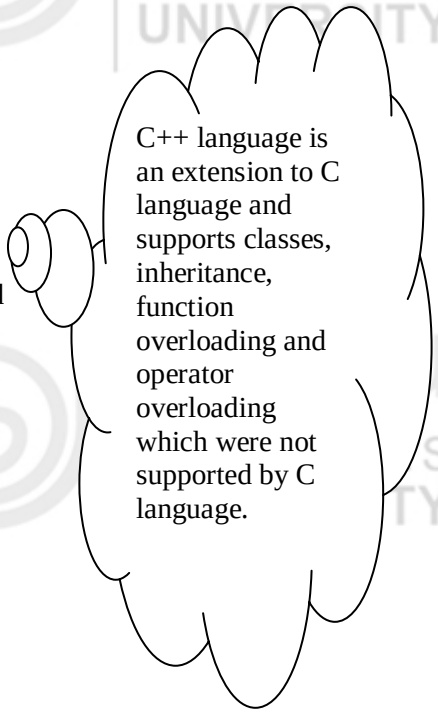
UNIT 2 INTRODUCTION TO C+ +

Structure **Page Nos.**

2.0 INTRODUCTION

In the previous unit, we have discussed concept of Objects Oriented Programming and benefit from this Object Oriented Language. In this unit we shall discuss something about Data Types, Operators and control structures used in C++. Data Type in C++ is used to define the type of data that identifiers accepts in programming and operators are used to perform a special task such as addition, multiplication, subtraction, and division etc of two or more operands during programming.

C++ is regarded as an intermediate-level language, as it comprises a combination of both high-level and low-level language features. C++ is an extension to C Programming language. C++ is one of the most popular programming languages and is used in the development of system software such as Microsoft Windows and Application Software such as device drivers, embedded software, high performance servers and client applications.



It was developed at AT&T Bell Laboratories in the early 1979s by Bjarne Stroustrup. Its initially name was C with classes, but later on in 1983 it was renamed as C++. It is a deviation from traditional procedural languages in the sense that it follows object oriented programming (OOP) approach which is quite suitable for managing large and complex programs.

An object oriented language combines the data to its function or code in such a way that access to data is allowed only through its function or code. Such combination of data and code is called an object. For example, an object called Student may contain data and function or code as shown in Figure 2.1:

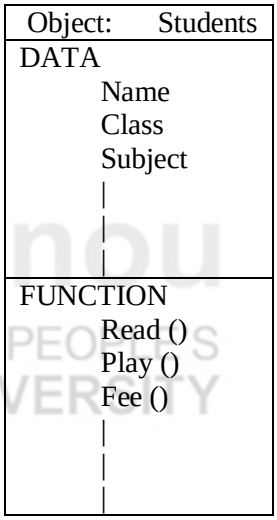


Figure 2.1: Representation of Object

The data part contains the Name, Class and Subject and function part contains three functions such as: read ( ), Play ( ) and Fee ( ). Thus, the various objects in the object-oriented language interact with each other through their respective codes or functions as shown in Figure 2.2.

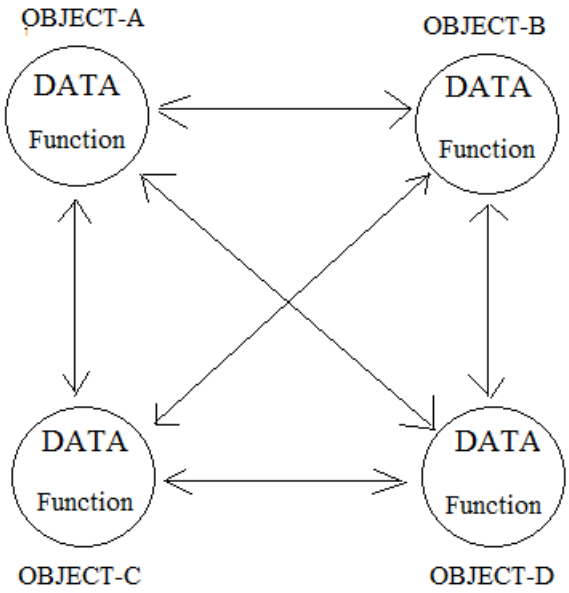


Figure 2.2: The Object-oriented approach

It may be noted here that the data of an object can be accessed only by the functions associated with that object. However, functions of one object can access the functions of other objects.

2.1 OBJECTIVES

After studying this unit, you should be able to do the following:

- explain basic concepts of Object Oriented Programming Language;
- explain operators and their syntax in C++;
- learn about C++ character set, tokens and basic data types;
- identify the difference between implicit and explicit conversions;
- explain about Input/Output streams supported by C++;
- explain the structure of a C++ program;
- write a simple program in C++; and
- understand keywords used in C++ and control structure in C++.

2.2 BASICS OF C++

C++ is an object oriented programming (OOP) language. It was developed at AT&T Bell Laboratories in the early 1979s by Bjarne Stroustrup. Its initial name was C with classes, but later on in 1983 it was renamed as C++.

It is a deviation from traditional procedural languages in the sense that it follows object oriented programming (OOP) approach which is quite suitable for managing large and complex programs. C++ language is an extension to C language and supports classes, inheritance, function overloading and operator overloading which were not supported by C language.

In any language, there are some fundamentals you need to learn before you begin to write even the most elementary programs. This chapter includes these fundamentals;

basic program constraints, variables, and Input/output formats. C++ is a superset of C language. It contains the syntax and features of C language. It contains the same control statements; the same scope and storage class rules; and even the arithmetic, logical, bitwise operators and the data types are identical. C and C++ both the languages start with main function.

The object oriented feature in C++ is helpful in developing the large programs with clarity, extensibility and easy to maintain the software after sale to customers. It is helpful to map the real-world problem properly. C++ has replaced C programming language and is the basic building block of current programming languages such as Java, C# and Dot.Net etc.

2.2.1 C++ Character Set

Character set is a set of valid characters that a language can recognise. The character set of C++ is consisting of letters, digits, and special characters. The C++ has the following character set:

Letters (Alphabets)	A------Z, a-------z
Digits	0------9
Special Characters	+, -, *, /, ^, \, ( ), [ ], { }, =, !, < >. ‘, ", $, ;, :, % , &, ?, _, #, <=, >=, @

There are 62 letters and digits character set in C++ (26 Capital Letters + 26 Small Letters + 10 Digits) as shown above. Further, C++ is a case sensitive language, i.e. the letter A and a, are distinct in C++ object oriented programming language. There are 29, punctuation and special character set in C++ and is used for various purposes during programming.

White Spaces Characters:

A character that is used to produce blank space when printed in C++ is called white space character. These are spaces, tabs, new-lines, and comments.

Tokens:

A token is a group of characters that logically combine together. The programmer can write a program by using tokens. C++ uses the following types of tokens:

- Keywords
- Identifiers
- Literals
- Punctuators
- Operators

The identifier is a sequence of characters taken from C++ character set.

2.2.2 Identifiers

A symbolic name is generally known as an identifier. Valid identifiers are a sequence of one or more letters, digits or underscore characters (_). Neither spaces nor punctuation marks or symbols can be part of an identifier. Only letters, digits and single underscore characters are valid.

In addition, variable identifiers always have to begin with a letter. In no case can they begin with a digit. Another rule for declaring identifiers is that they cannot match any keyword of the C++ programming language. The rules for the formation of identifiers can be summarised as:

An identifier may include of alphabets, digits and/or underscores.
It must not start with a digit.

C++ is case sensitive, i.e., upper case and lower case letters are considered different form each other. It may be noted that TOTAL and total are two different identifier names.

It should not be a reserved word.

A member function with the same name as its class is called constructor and it is used to initialize the objects of that class type with an initial value. Objects generally need to initialize variables or assign dynamic memory during their process of creation to become operative and to avoid returning unexpected values during their execution. For example, to avoid unexpected results in the example given below we have initialized the value of rollno as 0 and marks as 0.0.

2.2.3 Keywords

There are some reserved words in C++ which have predefined meaning to complier called keywords. These are also known as reserved words and are always written or typed in lower cases. There are following keywords in C++ object oriented language:

List of Keywords:

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

2.3 A SIMPLE C++ PROGRAM

The best way to start learning a programming language is by writing a program. A simple C++ program has four sections and these are shown in following C++ program

Simple C++ Program:

```
#include <iostream.h>                    // Section: 1- The include Directive
using namespace std;                  // Section :2 - Class declaration and member
functions
int main ()                           // Section: 3 - Main function definition
{                                     // Section: 4 - Declaration of an object
  cout << "Hello World!";
  return 0;
}
```

Output:

Hello World!

This is one of the simplest programs that can be written in C+ programming language. It contains all the fundamental components which every C++ program can have. Line by line explanation of the codes of this program and its sections is given below:

Section: 1 – The include Directive

#include <iostream.h>

Lines beginning with a hash sign (#) are directives for the pre-processor. They are not regular code lines with expressions but indications for the compiler's pre-processor. In this case the directive #include <iostream> tells the pre-processor to include the iostream standard file. This specific file (iostream) includes the declarations of the basic standard input-output library in C++, and it is included because its functionality is going to be used later in the program.

Section: 2 – Class declaration and member functions

using namespace std;

All the elements of the standard ANSI C++ library are declared within namespace std;. The syntax of this command is: using namespace std;. In order to access its functionality we declare all the entities inside namespace std;. This line is very often used in C++ programs that use the standard library and defines a scope for the identifiers that are used in a program.

Section: 3 - Main function definition

int main ()

This line corresponds to the beginning of the definition of the main function. The main function is the point by where all C++ programs start their execution, independently of its location within the source code. It does not matter whether there are other functions with other names defined before or after it - the instructions contained within this function's definition will always be the first ones to be executed in any C++ program. For that same reason, it is essential that all C++ programs have a main function.

The word main is followed in the code by a pair of parentheses (()). That is because it is a function declaration: In C++, what differentiates a function declaration from other types of expressions is these parentheses that follow its name. Optionally, these parentheses may enclose a list of parameters within-them.

Section: 4 - Declaration of an object

Right after these parentheses we can find the body of the main function enclosed in braces ({}). What is contained within these braces is what the function does when it is executed.

cout << "Hello World!";

This line is a C++ statement. A statement is a simple or compound expression that can actually produce some effect. In fact, this statement is used to display output on the

screen of the computer. cout is the name of the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters.
cout is declared in the iostream standard file within the std namespace, so that's why we needed to include that specific file and to declare that we were going to use this specific namespace earlier in our code.

Notice that the statement ends with a semicolon character (;). This character is used to mark the end of the statement and in fact it must be included at the end of all expression statements in all C++ programs.

return 0;

The return statement causes the main function to finish.

☞ Check Your Progress 1

Fill in the appropriate words from following:

a) An int data type requires _____
   2 bytes
   4 bytes
   1 bytes
   8 bytes

b) iostream.h _____
   is a header file
   pre-processor directives
   user-defined function
   both a and b

c) cout in C++ is a _____
   object
   class
   function
   command

d) The standard C++ comment is _____
   /
   //
   /* and */
   None of the above

Short Answer type questions:

1) What do you mean by keyword in C++?

   ……

   ……

   ……

2) What do you mean by identifier in C++?

   ……

   ……

   ……

3) What do you mean by data types in C++?

……

……

……

4) What is the difference between variable and constant in C++ programming language?

……

……

……

2.4 SOME SIMPLE C+ + PROGRAMS

In this section, some basic C++ program are given. You practice it and may write some more program like these.

Program: 1

```
// Printing a message
#include <iostream.h>
int main(void)
{
  cout << "Hello, this is my first C++ program" << endl;
  return 0;
}
```

Output:

Hello, this is my first C++ program

Program: 2

```
// Printing name
#include <iostream.h>
 # include<conio.h>
 main()
{
char  name [15];
clrscr();
cout << "Enter your name:";
cin >> name;
cout<<"Your name is: " <<name;
return0;
}
```

Output:

Enter your name: Ram
Your name is: Ram

Program: 3

```
// operating with variables
#include <iostream.h>
using namespace std;
int main ()
{
  // declaring variables:
  int a, b;
  int result;
  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;
  // print out the result:
  cout << result;
  // terminate the program:
  return 0;
}
```

Output:

Result = 4

Program: 4

```
// initialization of variables
#include <iostream.h>
using namespace std;
int main ()
{
  int a=5;               // initial value = 5
  int b(2);              // initial value = 2
  int result;            // initial value undetermined
  a = a + 3;
  result = a - b;
  cout << result;
  return 0;
}
```

Output:

Result = 6

Program: 5

```
// my first string
#include <iostream.h>
#include <string>
using namespace std;
int main ()
{
  string mystring;
  mystring = "This is the initial string content";
  cout << mystring << endl;
  mystring = "This is a different string content";
  cout << mystring << endl;
  return 0;
}
```

Output:

This is the initial string content
This is a different string content

Program: 6

```
// defined constants: calculate circumference
#include <iostream.h>
using namespace std;
#define PI 3.14159
#define NEWLINE '\n'
int main ()
{
  double r = 5.0;               // radius
  double circle;
  circle = 2 * PI * r;
  cout << circle;
  cout << NEWLINE;
  return 0;
}
```

Output:

Circle = 31.4258714

Program: 7

```
#include <iostream.h>
 # include<conio.h>
 main()
{
int num, num1;
clrscr();
cout << "Enter two numbers:”;
cin >> name>>num1;
cout<<”Entered numbers are : “ ;
cout <<num<<”\t”<<num1;
return0;
}
```

Output:

```
Enter two numbers: 9, 15
Entered numbers are 9, 15
```

2.5 DIFFERENCE BETWEEN C AND C++

Following are some differences between C and C ++ :

- C++ is regarded as an intermediate-level language. It comprises a combination of both high-level and low-level language features. C++ is an extension to C Programming language. The difference between the two languages can be summarised as follows:
- The variable declaration in C, must occur at the top of the function block and it must be declared before any executable statement. In C++ variables can be declared anywhere in the program.
- In C++ we can change the scope of a variable by using scope resolution operator. There is no such facility in C language.
- C Language follows the top-down approach while C++ follows both top-down and bottom-up design approach.
- C is a procedure language and C++ is an object oriented language.
- C allows a maximum of 32 characters in an identifier name whereas C++ allows no limit on identifier length.
- C++ is an extension to C language and allows declaration of class, while C language does not allow this feature.
- C++ allows inheritance and polymorphism while C language does not.

2.6 DATA TYPES IN C++

In C++ programming, we store the variables in our computer's memory, but the computer has to know what kind of data we want to store in them. The amount of memory required to store a single number is not the same as required by a single letter or a large number. Further, interpretation of different data is different inside computers memory.

The memory in computer system is organized in bits and bytes. A byte is the minimum amount of memory that we can manage in C++. A byte can store a relatively small amount of data: one single character or a small integer. In addition, the computer can manipulate more complex data types that come from grouping several bytes, such as long numbers or non-integer numbers.

Data Type in C++ is used to define the type of data that identifiers accepts in programming and operators are used to perform a special task such as addition, multiplication, subtraction, and division etc of two or more operands during programming.

C++ supports a large number of data types. The built in or basic data types supported by C++ are integer, floating point and character type. A brief discussion on these types is shown in Figure 2.3 which are shown below:

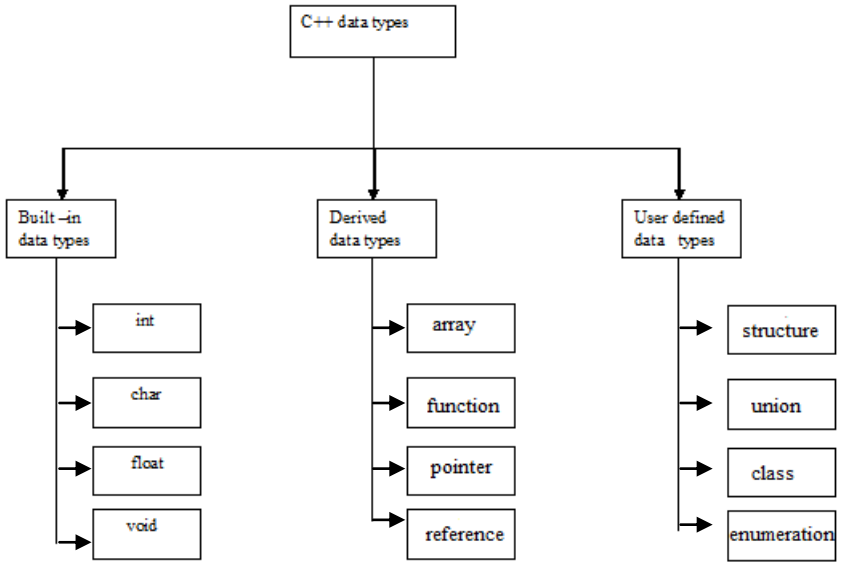


Figure 2.3 Hierarchy of C++ Data types

2.6.1 Built-in Data Types

There are four types of built-in data types as shown in the fig: 2. Let us discuss each of these and the range of values accepted by them one by one.

Integer Data type (int)

An integer is an integral whole number without a decimal point. These numbers are used for counting. For example 26, 373, -1729 are valid integers. Normally an integer can hold numbers from -32768 to 32767.

The int data type can be further categorized into following:

- Short
- Long
- Unsigned

The short int data type is used to store integer with a range of – 32768 to 32767, However, if the need be, a long integer (long int) can also be used to hold integers from -2, 147, 483, 648 to 2, 147, 483, 648. The unsigned int can have only positive integers and its range lies up to 65536.

Floating point data type (float)

A floating point number has a decimal point. Even if it has an integral value, it must include a decimal point at the end. These numbers are used for measuring quantities. Examples of valid floating point numbers are: 27.4, -92.7, and 40.03.

A float type data can be used to hold numbers from 3.4*10-38 to 3.4*10+38 with six or seven digits of precision. However, for more precision a double precision type (double) can be used to hold numbers from 1.7*10-308 to 1.7*10+308 with about 15 digits of precision.

Summary of Basic fundamental data types as well as the range of values accepted by each data type is shown in the following table.

Void data type

It is used for following purposes:

- It specifies the return type of a function when the function is not returning any value.
- It indicates an empty parameter list on a function when no arguments are passed.
- A void pointer can be assigned a pointer value of any basic data type.

Char data type

It is used to store character values in the identifier. Its size and range of values is given in Table 2.1.

Table: 1 Basic Fundamental Data Type

Name	Description	Size*	Range
Char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
Int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
Bool	Boolean value. It can take one of two values: true or false.	1byte	true or false
Float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
Double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	Wide character.	2 or 4 bytes	1 wide character

Note: The values of the column Size and Range given in the table above, depends on the computer system on which the program is compiled. The values shown above are those found on 32-bit computer systems. But for other systems, the general specification is that int has the natural size suggested by the system architecture (one "word") and the four integer type's char, short, int and long must each one be at least as large as the one preceding it, with char being always one byte in size. The same applies to the floating point types float, double and long double, where each one must provide at least as much precision as the preceding one.

2.6.2 Derived Data types

C++ also permits four types of derived data types. As the name suggests, derived data types are basically derived from the built-in data types. There are four derived data types. These are:

- Array
- Function
- Pointer, and
- Reference

We will discuss these data types subsequently in this unit.

2.6.3 User Defined Data Types

C++ also permits four types of user defined data types. As the name suggests, user defined data types are defined by the programmers during the coding of software development. There are four user defined data types. These are:

- Structure
- Union
- Class, and
- Enumerator

We will discuss these data types in the later units of this course.

2.7 TYPE CONVERSION

In C++ object oriented language smaller memory data type variable can be converted to large data type by the compiler. It is required to make the language robust. When a variable of int type is multiplied by a variable of float type then the output is saved inside the computer system memory as double data type. Thus C++ permits mixed expressions. Type conversion can be done by following two ways:

a) Automatic

When an expression consists of more than one type of data elements in an expression, the C++ compiler converts the smaller data type element in larger data type element. This process is known as implicit or automatic conversion.

b) Typecasting

This statement allows the programmer to convert one data type into another data type by writing the following syntax:

aCharVar = static_cast<char>(an IntVar);

Here in the above syntax char variable will be converted into int Variable after execution of the syntax in the C++ program.

2.8 VARIABLES

A variable is the most fundamental aspect of any computer language. It is a location in the computer memory which can store data and is given a symbolic name for easy reference. The variables can be used to hold different values at different values at different times during the execution of a program.

To understand more clearly, let us take following example:

Total = 20.00 (i)
Net = Total - 12.00 (ii)

In equation (i), a value 20.00 has been stored in a memory location Total. The variable Total is used in statement (ii) for the calculation of another variable Net. The point worth noting is that the variable Total is used in statement (ii) by its name not by its value. Before a variable is used in a program, it has to be defined. This activity enables the compiler to make available the appropriate type of location in the memory. The definition of a variable consists of the type name followed by the name of the variable.

Declaration of variables:

In order to use a variable in C++, we must first declare it specifying which data type we want it to be. The syntax to declare a new variable is to write the specifier of the desired data type (like int, bool, float, etc.) followed by a valid variable identifier. For example:

```
int a;
float mynumber;
```

These are two valid declarations of variables. The first one declares a variable of type int with the identifier a. The second one declares a variable of type float with the identifier mynumber. Once declared, the variables a and mynumber can be used within the rest of their scope in the program.

If you are going to declare more than one variable of the same type, you can declare all of them in a single statement by separating their identifiers with commas. For example:

```
int a,b,c;
```

This declares three variables (a, b and c), all of them of type int, and has exactly the same meaning as:

```
int a;
int b;
int c;
```

Similarly, a variable Total of type float can be declared as shown below:

```
float Total;
```

Similarly the variable Net can also be defined as shown below:

```
float Net;
```

Examples of some valid variable declarations are:

(i) int count;
(ii) int i, j, k;
(iii) char ch, first;
(iv) float total, Net;
(v) long int sal;

2.9 LITERALS OR CONSTANTS

A number which does not charge its value during execution of a program is known as a constant or literals. Any attempt to change the value of a constant will result in an error message. A keyword const is added to the declaration of an identifier to make that identifier constant. A constant in C++ can be of any of the basic data types. Let us consider the following C++ expression:

Once an identifier is declared as constant at the time of declaration, its value can't be changed during the execution of the program.

const float Pi = 3.1215;

The above declaration means that Pi is a constant of float types having a value: 3.1415.

Examples of some valid constant declarations are:

const int rate = 50;
const float Pi = 3.1415;
const char ch = 'A';
Scope of variables:

Let us now discuss scope of variables in C++ programming. A variable can be either of global or local scope. A global variable is a variable declared in the main body of the C++ source code, outside all the functions. Global variables can be called from anywhere in the code, even inside functions, whenever it is after its declaration.

The local variable is one declared within the body of a function or a block. To illustrate the scope of global variable and local variable, let us look at the figure 4. The scope of local variables is limited to the block enclosed in braces ({}) where they are declared. For example, if they are declared at the beginning of the body of a function (like in function main), their scope is between its declaration point and the end of that function.

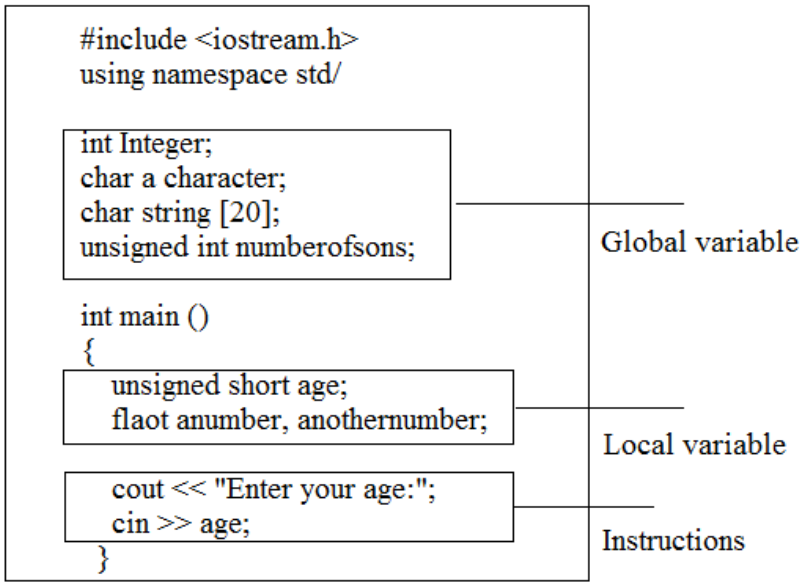


Figure 2.4: Scope of variables in C++ program

In the example above, this means that if another function existed in addition to main, the local variables, declared in main could not be accessed from the other function and *vice versa*.

☞ Check Your Progress 2

1) What do you mean by global variable and local variable in C++?

……

……

……

2) What do you mean string literals in C++ programming language?

……

……

……

3) Explain scope of a variable?

……

……

……

4) Explain the difference between C and C++?

……

……

……

2.10 OPERATORS IN C++

C++ has a rich set of operators. Operators are the term used to describe the action to be taken between two data operands. Expressions are made by combining operators between operands. C++ supports six types of operators:

- Arithmetical operators
- Relational operators
- Logical operators
- Bitwise operators
- Precedence of operators
- Special operators
- Escape sequence

2.10.1 Arithmetical operators

An operator that performs an arithmetic (numeric) operation such as +, -, *, /, or % is called arithmetic operator. Arithmetic operation requires two or more operands. Therefore these operators are called binary operators. The Table 2.2 shows the arithmetic operators:

Table 2.2:Operators Meaning with Example

Operator	Meaning	Example	Answer
+	addition	8+5	13
-	subtraction	8-5	3
*	multiplication	8*5	40
/	division	10/2	5
%	modulo	5%2	1

2.10.2 Relational operators

The relational operators shown in Table 2.3 are used to test the relation between two values. All relational operators are binary operators and therefore require two operands. A relational expression returns zero when the relation is false and a non-zero when it is true.

Table 2.3: Relational operators with Example

Operator	Meaning	Example
==	Equal to	5==5
!=	Not equal to	5!=7
>	Greater than	7>5
<	Less than	8<9
>=	Greater than or equal to	8>=8
<=	Less than or equal to	9<=9

2.10.3 Logical operators

The (!) operator is the C++ operator to perform the Boolean operation NOT. It has only one operand, located at its right, and the only thing that it does is to inverse the value of it, producing false if its operand is true and true if its operand is false. Basically, it returns the opposite Boolean value of evaluating its operand. Logical operators of C++ are given in Table 2.4.

Table 2.4: Logical operators

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

To understand the use of these operators in C++, let us take following example:

Example:

```
!(5 == 5)    // evaluates to false because the expression at its right (5 == 5) is true.
!(6 <= 4)    // evaluates to true because (6 <= 4) would be false.
!true        // evaluates to false
!false       // evaluates to true.
```

The logical operators && and || are used when evaluating two expressions to obtain a single relational result. The operator && corresponds with Boolean logical operation AND. This operation results true if both its two operands are true and false otherwise. The Table 2.5 shows the result of operator && by evaluating the expression a && b:

Table 2.5: Use of && Operator

Operand (a)	Operand (b)	Result
true	True	True
true	False	False
false	True	False
false	False	False

The operator || corresponds with Boolean logical operation OR. This operation results true if either one of its two operands is true, thus being false only when both operands are false themselves. To understand the use || OR operator, let us take the possible results of a || b in Table 2.6.

Table 2.6: Use of || Operator

Operand (a)	Operand (b)	Result (a || b)
True	True	True
True	False	True
False	True	True
False	False	False

Example:

((5 == 5) && (3 > 6)) // evaluates to false (true && false).
((5 == 5) || (3 > 6)) // evaluates to true (true || false).

2.10.4 Bitwise Operators

In C++ programming language, bitwise operators are used to modify the bits of the binary pattern of the variables. Table 2.7 gives use of some bitwise operators:

Table 2.7: Use of Bitwise Operator

operator	Asm equivalent	Description
&	AND	Bitwise AND
|	OR	Bitwise Inclusive OR
^	XOR	Bitwise Exclusive OR
~	NOT	Unary complement (bit inversion)
<<	SHL	Shift Left
>>	SHR	Shift Right

2.10.5 Precedence of Operators

In case of several operators in an expression, we may have some doubt about which operand is evaluated first and which later. For example, let us take following expression:

a = 5 + 7 % 2
Here we may doubt if it really means:
A = 5 + (7 % 2) // with a result of 6, or
a = (5 + 7) % 2 // with a result of 0

The correct answer is the first of the two expressions, with a result of 6. Precedence order of some operators in C++ programming language is given in the Table 2.8.

Table 2.8: Precedence of operators in descending order

Level of Precedence	Operator	Description	Grouping
1	::	scope	Left-to-right
2	() [] . -> ++ -- dynamic_cast static_cast reinterpret_cast const_cast typeid	postfix	Left-to-right
3	++ -- ~ ! sizeof new delete	unary (prefix)	Right-to-left
	* &	indirection and reference (pointers)	
	+ -	unary sign operator	
4	(type)	type casting	Right-to-left
5	.* ->*	pointer-to-member	Left-to-right
6	* / %	multiplicative	Left-to-right
7	+ -	additive	Left-to-right
8	<< >>	shift	Left-to-right
9	< > <= >=	relational	Left-to-right
10	== !=	equality	Left-to-right
11	&	bitwise AND	Left-to-right
12	^	bitwise XOR	Left-to-right
13	|	bitwise OR	Left-to-right
14	&&	logical AND	Left-to-right
15	||	logical OR	Left-to-right
16	?:	conditional	Right-to-left
17	= *= /= %= += -= >>= <<= &= ^= |=	assignment	Right-to-left
18	,	comma	Left-to-right

Grouping defines the precedence order in which operators are evaluated in the case that there are several operators of the same level in an expression. Thus if you want to write complicated expressions and you are not completely sure of the precedence levels, always include parentheses. It will also make your code easier to read.

2.10.6 Special Operators

Apart from the above operators that we have discussed above so far, C++ programming language supports some special operators. Some of them are: increment and decrement operator; size of operator; comma operator etc.

Increment and Decrement Operator

In C++ programming language, Increment and decrement operators can be used in two ways: they may either precede or follow the operand. The prefix version before the operand and postfix version comes after the operand. The two versions have the same effect on the operand, but they differ when they are applied in an expression. The prefix increment operator follows "change then use" rule and post fix operator follows "use then change" rule.

The size of operator

We know that different types of variables, constant, etc. require different amount of memory to store them. The sizeof operator can be used to find how many bytes are required for an object to store in memory.

Example:

sizeof (char) returns 1
sizeof (int) returns 2
sizeof (float) returns 4
if k is integer variable, the sizeof (k) returns 2.

The sizeof operator determines the amount of memory required for an object at compile time rather than at run time.

The comma operator

The comma operator gives left to right evaluation of expressions. It enables to put more than one expression separated by comma on a single line.

Example:

int i = 20, j = 25;

In the above statements, comma is used as a separator between the two statements.

2.10.7 Escape Sequence

There are some characters which can't be typed by keyboard in C++ programming language. These are called non-graphic characters. An escape sequence is represented by backslash (\) followed by one or more characters. The Table 2.9 gives a listing of common escape sequences.

Table 2.9: Escape Sequence

Sequence	Task
\a	Bell (beep)
\b	Backspace
\f	Formatted
\n	Newline or line feed
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\?	Question mark
\ \	Backslash
\ '	Single quote
\ "	Double quote
\ xhh	Hexadecimal number (hh represents the number in hexadecimal)
\ 000	Octal number (00 represents the number in octal)
\0	Null

Punctuators

In C++ programming language, following characters are used as punctuators for enhancing the readability and maintainability of programs.

Brackets [ ]	opening and closing brackets indicate single and multidimensional array subscript.
Parentheses ( )	opening and closing brackets indicate functions calls, function parameters for grouping expressions etc.
Braces { }	opening and closing braces indicate the start and end of a compound statement.
Comma ,	it is used as a separator in a function argument list.
Semicolon ;	it is used as a statement terminator.
Colon :	it indicates a labelled statement or conditional operator symbol.
Asterisk *	it is used in pointer declaration or as multiplication operator
Equal sign =	it is used as an assignment operator.
Pound sign #	it is used as pre-processor directive

2.11 CONTROL STRUCTURE IN C++

C++ program is usually not limited to a linear sequence of instructions but it may bifurcate, repeat code or may have to take decisions during the process of coding. For that purpose, C++ provides control structures which are used to control the flow of program.

Before we discuss control structures, let us first discuss a new concept: the compound-statement or block, which is very much needed to understand well the flow of control in a program.

A block is a group of statements which are separated by semicolons (;) like all C++ statements, but grouped together in a block enclosed in braces: { }: for example:

```
{
statement1;
statement2;
statement3;
...
}
```

represents a compound statement or block.

In C++ object oriented programming, the control structure can be classified into following three categories:

Selection or conditional statement;
Iterating or looping statement;
Breaking statement;

Let us discuss the above control statement and their types in the following section.

2.11.1 Selection or conditional statement

In this type of statement, the execution of a block depends on the next condition. If the condition evaluates to true, then one set of statement is executed, otherwise another set of statements is executed. C++ provides following types of selection statements:

If;
If-else;
Nested if;
Switch
conditional

a) if statement:

The syntax of if statement is

```
If (expression)
        {
(Body of if)
Statements;
}
```

Where, expression is the condition that is being evaluated. If this condition is true, statement is executed. If it is false, statement is ignored (not executed) and the program continues right after this conditional structure.

Program: 1

```
# include <iostream.h>
main()
{       int a, b;
a=10;
b=20;
if (a<b)
cout <<"a is less than b";
}
```

Output:

a is less than b.

Since here in the program value of a is less than the value of b, so the output of the program 1 is "a is less than b"

b) if-else statement:

The syntax of if –else statement is

```
If (expression)
        {
          (Body of if)
         Statements 1;
         }
        else
        {
          (Body of else}
           Statement 2
        }
```

Where, expression is the condition that is being evaluated. If this condition is true, statement - 1 is executed. If it is false, then if statement is skipped and the body of else statement is executed.

Program: 2

```
# include <iostream.h>
main()
{        int a, b;
a=10;
b=20;
if (a<b)
cout <<"a is less than b";
}
else
{
         cout<< "b is less than a"
}
```

Output:

a is less than b.

Since here in the program value of a is less than the value of b so the output of the program 1 is "a is less than b"

c) Switch statement:

Switch statement is used for multiple branch selection. The syntax of switch statement is

```
switch (expression)
        {
 case exp 1:
First case body;
Break;
case exp 2:
Second case body;
Break;
case exp 3:
Third case body;
Break;
        default:
                default case body;
}
```

Here, expression is the condition that is being evaluated. If the case 1 condition is true, First case body is executed, otherwise case exp 2 is checked and so on....If none of case expressions is true then the value of default case body is executed.

Program: 3

```
# include <iostream.h>
# include <conio.h>
 int main()
{
        clrscr();
                int d_o_w;
cout <<"Enter number of week's day (1-7)";
cin>>d_o_w;
        switch(d_o_w)
{

case 1: cout<<"/n Sunday";
break;
case 2: cout<<"/n Monday";
break;
case 3: cout<<"/n Tuesday";
break;
case 4: cout<<"/n Wednesday";
break;
case 5: cout<<"/n Thursday";
break;
case 6: cout<<"/n Friday";
break;
case 7: cout<<"/n Saturday";
break;
default: cout<<"/n Wrong number of day";
}
return 0;
}
```

Output:

Enter number of week's dat (1-7): 4

Wednesday

d) Nested if statement:

A nested if statement is a statement that has another if in its if's body or in its else's body. The syntax of switch statement is

```
if (expression 1)
                statement 1;
else if (expression 2)
statement 2;
else if (expression 3)
statement 3;
.
.
.
else
statement;
```

Here, expression is the condition that is being evaluated. If the case 1 condition is true, First case body is executed, otherwise it is skipped and next else expression is evaluated and so on.....

Program Segment: 4

```
# include <iostream.h>
# include <conio.h>
void main(void)
{
   float a,b,c,d:
         cout<< “Enter any four numbers \n”;
cin >>a >>b >>c >>d;
if (a > b) {
         if (a > c) {
                  if (a > d)
  cout << "largest = “ << a << end1;
  else
  cout << “largest = “ << d << end1;
   }
else
      {
         if (c > d)
                cout << " largest = “ << c << end1;
            else
                cout << “largest = “ << d << end1;
     }
  } // end of outer if part
else
  if (b > c) {
               if (b > d)
        cout << "largest = “ << b << end1;
else
        cout << largest = “ << d << end1;
              }
else {
          if (c > d)
            cout<< “largest = “ << c << end1;
        else
            cout << “largest = “ << d << end1;
       }
} // end of main program
```

Output:

Enter any four numbers
10 20 30 35

largest = 35

2.11.2 Iterative or looping statement

In C++ , programming language looping statement is used to repeat a set of instructions until certain condition is fulfilled. The iteration statements are also called loops or looping statement. C++ allows following four kinds of iterative loops:

- for loop
- while loop
- do-while loop and
- nested loops

a) for loop

This loop is easiest amongst all loops in C++ programming. The syntax of this loop is:

```
for (initialization; condition; increase)
{
        (body of the loop) statements;
}
```

This loop is specially designed to perform a repetitive action with a counter which is initialized and increased on each iteration

It works in the following way:

initialization is executed. Generally, it is an initial value setting for a counter variable. This is executed only once.
condition is checked. If it is true the loop continues, otherwise the loop ends and statement is skipped (not executed).
statement is executed. As usual, it can be either a single statement or a block enclosed in braces { }.
finally, whatever is specified in the increase field is executed and the loop gets back to step 2.

Program: 5

```
// Program by using a for loop
#include <iostream.h>
using namespace std;
int main ()
{
  for (int n=10; n>0; n--)
          {
          cout << n << ", ";
          }
  cout << "FIRE!\n";
  return 0;
}
```

Output:

10, 9, 8, 7, 6, 5, 4, 3, 2, 1, FIRE!

b) while loop

If we do not know the number of iterations before starting the loop then while loop is used. Its syntax is as follows:

The functionality of this loop is simply to repeat statement while the condition set in expression is true.

```
initialization;
while (expression)
{
statement;
increment;
}
```

Program: 6

```
// Program by using while loop
#include <iostream.h>
using namespace std;
int main ()
{
  int n;
  cout << "Enter the starting number : ";
  cin >> n;
            while (n>0)
            {
            cout << n << ", ";
            -n;
            }
            cout << "FIRE!\n";
  return 0;
}
```

Output:

Enter the starting number: 8
8, 7, 6, 5, 4, 3, 2, 1, FIRE!

When execution of program starts, the user is prompted to insert a starting number for the countdown. Then the while loop begins, if the value entered by the user fulfils the condition n>0 (that n is greater than zero) the block that follows the condition will be executed and repeated while the condition (n>0) remains being true.

c) The do-while loop

Unlike for and while loops, the do-while is an exit-controlled loop i.e., it evaluates its test – expression at the bottom of the loop after executing its loop-body statement. This means that a do-while loop always executes at least once, even when the test – expression evaluates to false initially.

Its syntax is given as:

```
do
{
statement
}
while (test condition);
```

Program: 7

```
// number echoer
#include <iostream.h>
using namespace std;
int main ()
{
          unsigned long n;
        do
        {
                cout << "Enter number (0 to end): ";
                cin >> n;
                cout << "You entered: " << n << "\n";
        }
        while (n != 0);
 return 0;
}
```

Output:

```
Enter number (0 to end): 12345
You entered: 12345
Enter number (0 to end): 160277
You entered: 160277
Enter number (0 to end): 0
You entered: 0
```

The do-while loop is usually used when the condition that has to determine the end of the loop is determined within the loop statement itself, like in the previous case, where the user input within the block is what is used to determine if the loop has to end.

d) Nested for loop

If a loop is placed inside the same loop then it is called nested for loop in C++ programming language. To understand, it let us take the following program:

Program: 8

```
// Program to print the pyramid of numbers by using a nested for loop
#include <iostream.h>
#include <conio.h>
#include <math.h>
void main ()
{
clrscr();
int n, i, j, k;
cout << "Enter the number of rows in the pyramid:";
cin>> n;
for (i=1; i<=n-i; i++)
{
        for (j=1; j<=n-i; j++)
        {
        cout <<  " ";
```

```
}
        for (k=1; k<+i; k++)
        {
        cout<< k;
        }
        cout << end;
}
getch();
}
```

Output:

```
Enter the number of rows in the pyramid: 5
     1
    12
   123
  1234
 12345
```

2.11.3 Breaking statement

Using break, we can leave a loop even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end. In this section, we will discuss following breaking statements:

- break statement
- continue statement
- goto statement and
- exit statement

a) break statement

The break statement is used to terminate the execution of the loop program. It terminates the loop in which it is written and transfers the control to the immediate next statement outside the loop. The break statement is normally used in the switch conditional statement. To understand, it let us take the following C++ program:

Program: 9

```
// break loop example
#include <iostream.h>
using namespace std;
int main ()
{
  int n;
  for (n=10; n>0; n--)
        {
        cout << n << ", ";
        if (n==3)
                {
```

```
                cout << "countdown aborted!";
                break;
                }
        }
 return 0;
}
```

Output:

10, 9, 8, 7, 6, 5, 4, 3, countdown aborted!

b) continue statement

The continue statement causes the program to skip the rest of the loop in the current iteration as if the end of the statement block had been reached, causing it to jump to the start of the following iteration. For example, we are going to skip the number 5 in our countdown:

Program: 10

```
// continue loop example
#include <iostream.h>
using namespace std;

int main ()
{
  for (int n=10; n>0; n--) {
    if (n==5) continue;
    cout << n << ", ";
  }
  cout << "FIRE!\n";
  return 0;
}
```

Output:

10, 9, 8, 7, 6, 4, 3, 2, 1, FIRE!

c) goto statement

The goto statement is used to transfer control to some other parts of the program. It is used to alter the execution sequence of the program. To illustrate goto statement, let us take the following C++ program:

Program: 11

```
// goto loop example
#include <iostream.h>
using namespace std;
int main ()
{
  int n=10;
  loop:
```

```
cout << n << ", ";
  n--;
  if (n>0) goto loop;
  cout << "FIRE!\n";
  return 0;
}
```

Output:

10, 9, 8, 7, 6, 5, 4, 3, 2, 1, FIRE!

d) exit () statement

The exit statement is used to terminate the execution of the program. It is used when we want to stop the execution of the program depending on some condition. When we use exit () statement, we have to include other library functions such as process.h, or stdio.h file. To illustrate exit () statement, let us take the following C++ program:

Program: 12

```
// Program for exit statement
#include <iostream.h>
#include <conio.h>
void main ()
{

clrscr();
int i, number;
i = 1;
while(i<5)
{
 cout <<"Enter the number:";
cin>>number;
if number>5
{
cout <<"The number is greater than five or equal to" <<end1;
exit();
 }
 cout << "The number is: "<<number<<end1;
i++
}
getch();
}
```

Output:

Enter the number: 2
The number is: 2
Enter the number: 3
The number is: 3
Enter the number: 9
The number is greater than or equal to 9

2.12 I/O FORMATTING

In this section, we will discuss those functions which are used to format the Input and output of a C++ program. These functions are helpful in managing the I/O operations in C++ programming.

C + + supports input/output statements which can be used to feed new data into the computer or obtain output on an output device such as: VDU, printer etc. It provides both formatted and unformatted stream I/O statements. The following C + + streams can be used for the input/output purpose. In this section, we will discuss following I/O formatting functions:

Comments in C++
Unformatted Console I/O Functions
Setw I/O Formatting in C++
Inline Functions

Input/Output

Output is accomplished by using cout, which opens a "stream" to the standard output device (the screen). Data is inserted into the output stream using the << (insertion) operator. Input is accomplished by using cin, which opens a "stream" from the standard input device (keyboard). Data is retrieved from the stream by using the >> (extraction) operator.
Note: Every cin should be prefaced by a cout to prompt the user.

(i) Include <iostream.h>

The lines in the above program that start with symbol '#' are called directives and are instructions to the compiler. The word include with '#' tells the compiler to include the file iostream.h into the file of the above program. File iostream.h is a header file needed for input/output requirements of the program. Therefore, this file has been included at the top of the program.

(ii) void main ( )

The word main is a function name. The brackets ( ) with main tells that main ( ) is a function. The word void before main ( ) indicates that no value is being returned by the function main ( ). Every C++ program consists of one or more functions. However, when program is loaded in the memory, the control is handed over to function main ( ) and it is the first function to be executed.

(iii) The curly brackets and body of the function main ( )

Each, C ++ program starts with function called main ( ). The body of the function is enclosed between curly braces. These braces are equivalent to Pascal's BEGIN and END keywords. The program statements are written within the brackets. Each statement must end by a semicolon, without which an error message is generated.

2.12.1 Comments in C++

A comment is a statement in the program body to enhance the reading and understanding of the program. Comments are included in a program to make it more readable. If a comment is short and can be accommodated in a single line, then it is started with double slash sequence in the first line of the program. The syntax of short and one line comment is:

// Comment line...

However, if there are multiple lines in a comment, it is enclosed between the two symbols /* and */
Everything between /* and */ is ignored by the complier. The syntax of multiple line comment is

/* Start of multiple line comment
.....
.....
.....End of multiple line comment */

2.12.2 Unformatted Console I/O Functions

The I/O functions such as getch(), putchar(), get(), and put() etc. are called unformatted console I/O functions. The header file for these functions is <stdio.h> and should be included in the beginning of the program. The meaning and use of these functions can be illustrated as follows:

getch() This function is used to accept the input character which is typed by the keyboard during the execution of C++ program.

putchar() It displays the character on the screen at the current location of cursor.

Get (), and put () are the string functions in C++ programming language.

2.12.3 Setw - I/O Formatting in C++

In C++ programming language, stew () function is used to set the number of characters to be used as the field width for the next insertion operation. The field width determines the minimum number of characters to be written in some output representations.

This manipulator is declared in header <iomanip.h>, along with the other parameterized manipulators. This header file declares the implementation-specific requirement for setw (). To understand the use of sew () function in C++ programming; let us look at the following programs:

Program: 1

A program to insert a tab character between two variables by using setw ()

```
// setw example
#include <iostream.h>
#include <iomanip.h>
void main (void)
{
  int x, y, z;
  x = 400;
  y = 500;
  z = 600;
cout << x <<  '\t' << y << '\t' << z << end1;
}
```

Output:

400 500 600

Here in the above program 1 setw function has been used to insert a tab between three variables while displaying the content on the screen of computer.

2.12.4 Inline Functions

An inline function is written in one line when they are invoked. These functions are very short, and contain one or two statements. Inline functions are functions where the call is made to inline functions. The actual code then gets placed in the calling program.

Normally, a function call transfers the control from the calling program to the function and after the execution of the program returns the control back to the calling program after the function call. These concepts of function save program space and memory space and are used because the function is stored only in one place and is only executed when it is called. This execution may be time consuming since the registers and other processes must be saved before the function gets called.

The extra time needed and the process of saving is valid for larger functions. If the function is short, the programmer may wish to place the code of the function in the calling program in order for it to be executed. This type of function is best handled by the inline function.

The inline function takes the format as a normal function but when it is compiled it is compiled as inline code. The function is placed separately as inline function, thus adding readability to the source program. When the program is compiled, the code present in function body is replaced in the place of function call.

General Format of inline Function:

The general format of inline function is as follows:

```
inline datatype function_name(arguments)
```

The keyword inline specified in the above example, designates the function as inline function. For example, if a programmer wishes to have a function named exforsys with return value as integer and with no arguments as inline it is written as follows:
inline int exforsys ( )

Program 2

```
#include <iostream.h>
using namespace std;
int exforsys (into);
void main ( )
{
      int x;
      cout << "n Enter the Input Value: ";
      cin>>x;
      cout << "n The Output is: " << exforsys(x);
}

inline int exforsys(int x1)
{
      return 5*x1;
}
```

Output:

Enter the input value: 10
The output is> 50
Press any key to continue

2.12.5 Setprecision ()- I/O Formatting in C++

This function is used to control the number of digits of an output stream to be displayed on the screen in floating point value. This function is included in <iomanip.h> the header file and is included in the beginning of any C++ program. The syntax of this function is given as follows:

setprecision (int p);

To understand the use of this function, let us take the following C++ program:

Program 3

```
# include <iostream.h>
# include <iomanip.h>
void main (void)
{
float x,y,z;
x = 11;
y = 7;
z = x/y;
cout << setprecision(1) << z << end1;
cout << setprecision(2) << z << end1;
cout << setprecision(3) << z << end1;
cout << setprecision(4) << z << end1;
cout << setprecision(5) << z << end1;
}
```

Output:

1.6
1.57
1.571
1.5714
1.57142

2.12.6 Showpoint bit format flag- I/O Formatting in C++

This flag is used to show the decimal point for all floating point values. By default, it takes six decimal point values in C++ programming. The syntax of this flag is given as follows:

```
cout.setf(ios::shpowpoint);
```

To understand the use of this flag, let us take the following C++ program:

Program 4

```
# include <iostream.h>
void main ()
{
float w,x,y,z;
w = 2.34567845612
x = 11.34567653433;
y = 7.2345458765432;
z =  - 2345.5677225844;
cout.setf (ios::showpoint);
cout << "w = "  << w << "\n";
cout << "x = "  << x << "\n";
cout << "y = "  << y << "\n";
cout << "z = "  << z << "\n";
}
```

Output:

w = 2.345678
x= 11.345677
y = 7.234546
z = -2345.567723

2.12.7 Input and output stream flags - I/O Formatting in C++

To use many of the (I/O) manipulators, I/O streams have a flag field that specifies the current setting of decimal places and upper and lower case of alphabets in the output of a C++ program.

Flag Name	Meaning
skipws	skip white space during input
right	left justification of output
internal	pad after sign or base indicator
dec	decimal base
oct	octal base
hex	hexa decimal base
showbase	show base for octal and hexadecimal numbers
showpoint	show the decimal points for all floating numbers
uppercase	show upper case hex numbers
showpos	show '+' to positive numbers
scientific	use e for floating notations
fixed	use floating notations
unitbuf	flush all streams after insertions
stdio	flush out, stderr after insertion

Some of the I/O flag name and their meaning are given in the table above.

☞Check Your Progress 3

1) Explain the function of operators in C++?

……

……

……

2) Explain the term control structure in C++?

……

……

……

3) What do you mean by I/O formatting in C++ programming language?

……

……

……

4) Write a program in C++ to demonstrate the use of switch statement?

……

……

……

2.13 SUMMARY

In this unit you have learnt the features of object oriented programming, particularly that of C++ language. We have explained the C+ character set, tokens which include variables, constants and operators and data types used. In C++, the interchanging of data takes place automatically or by the programmer. The concept of input/output statement has been explained. The concept of comment statement which makes the program more readable is also given. Finally, we have also discussed control structure in C++ which is used to control execution sequence during the compilation and execution of program. At the end, you should be able to write a C++ program which will take input from the user, manipulate and print it on the screen by reading this unit.

2.14 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

Answers to fill in the blanks type questions

a) - 1, b) - 1 c) - 2 d) – 3

Answers to short answer type questions

1) There are some reserved words in C++ which have predefined meaning to complier called keywords. These are also known as reserved words and are always written or typed in lower cases.
2) A symbolic name is generally known as an identifier. The identifier is a sequence of characters taken from C++ character set. Valid identifiers are a sequence of one or more letters, digits or underscore characters (_). Neither spaces nor punctuation marks or symbols can be part of an identifier.

3) Data Type in C++ is used to define the type of data that identifiers accepts in programming and operators are used to perform a special task such as addition, multiplication, subtraction, and division etc of two or more operands during programming.

4) A variable is the most fundamental aspect of any computer language. It is a location in the computer memory which can store data and is given a symbolic name for easy reference. The variables can be used to hold different values at different values at different times during the execution of a program.

   A number which does not *change* its value during execution of a program is known as a constant or literals. Any attempt to change the value of a constant will result in an error message. A keyword const is added to the declaration of an identifier to make that identifier constant. A constant in C++ can be of any of the basic data types.

Check Your Progress 2

1) A global variable is a variable declared in the main body of the C++ source code, outside all the functions. Global variables can be called from anywhere in the code, even inside functions, whenever it is after its declaration.

   The local variable is one declared within the body of a function or a block.

2) A string literal consists of zero or more characters from the source character set surrounded by double quotation marks ("). A string literal represents a sequence of characters that, taken together, form a null-terminated string.
   String literals may contain any graphic character from the source character set except the double quotation mark ("), backslash (\), or newline character. They may contain the same escape sequences supported by C++ language. C++ strings have these types:

   Array of char[n], where n is the length of the string (in characters) plus 1 for the terminating '\0' that marks the end of the string.
   Array of wchar_t, for wide-character strings.

3) Scope of a variable in C++ can be defined as: a variable can be either of global or local scope. A global variable is a variable declared in the main body of the C++ source code, outside all the functions. Global variables can be called from anywhere in the code, even inside functions, whenever it is after its declaration.

4) The difference between the two languages can be summarised as follows:

   The variable declaration in C must occur at the top of the function block and it must be declared before any executable statement. In C++ variables can be declared anywhere in the program.

   In C++ we can change the scope of a variable by using scope resolution operator. There is no such facility in C language.

C Language follows the top-down approach while C++ follows both top-down and bottom-up design approach.

C is a procedure language and C++ is an object oriented language.

C allows a maximum of 32 characters in an identifier name whereas C++ allows no limit on identifier length.

C++ is an extension to C language and allows declaration of class, while C language does not allow this feature.

Check Your Progress 3

1) C++ has a rich set of operators. Operators is the term used to describe the action to be taken between two data operands. Expressions are made by combining operators between operands. C++ supports six types of operators:

- Arithmetical operators
- Relational operators
- Logical operators
- Bitwise operators
- Precedence of operators
- Special operators

2) C++ program is usually not limited to a linear sequence of instructions but it may bifurcate, repeat code or may have to take decisions during the process of coding. For that purpose, C++ provides control structures which are used to control the flow of program.

In C++ object oriented programming, the control structure can be classified into following three categories:

- Selection or conditional statement;
- Iterating or looping statement;
- Breaking statement;

3) I/O functions are those functions which are used to format the Input and output of a C++ program. These functions are helpful in managing the I/O operations in C++ programming.

C + + supports input/output statements which can be used to feed new data into the computer or obtain output on an output device such as: VDU, printer etc. It provides both formatted and unformatted stream I/O statements. The following C + + streams can be used for the input/output purpose. C++ supports following I/O formatting functions:

Comments in C++
Unformatted Console I/O Functions
Setw I/O Formatting in C++
Inline Functions

```
1)  # include <iostream.h>
    # include <conio.h>
     int main()
    {
         clrscr();
                  int d_o_w;
    cout <<"Enter number of week's day (1-7)";
    cin>>d_o_w;
         switch(d_o_w)
    {
    case 1: cout<<"/n Sunday";
```

```
break;
case 2: cout<<"/n Monday";
break;
case 3: cout<<"/n Tuesday";
break;
case 4: cout<<"/n Wednesday";
break;
case 5: cout<<"/n Thursday";
break;
case 6: cout<<"/n Friday";
break;
case 7: cout<<"/n Saturday";
break;
default: cout<<"/n Wrong number of day";
}
return 0;
}
```

Output:

Enter number of week's dat (1-7) : 4

Wednesday

2.15 FURTHER READINGS AND REFERENCES

1) E Balagurusamy, *Object Oriented Programming with C++*, Tata McGraw-Hill Publishing Company Ltd, New Delhi , 2001.
2) Er V. K. Jain, *Object Oriented Programming with C++*, Cyber Tech Publication, Daryaganj N Delhi-110002
3) Robert Lafore, *Object Oriented Programming in C++*, Galgotia Publications Pvt. Ltd. Daryaganj N Delhi-11002
4) Rajesh K Shukla, *Object Oriented Programming in C++*, Wiley India Publishing Pvt. Ltd. Daryaganj, N delhi-110002
5) Bjarne AT&T Labs Murray Hill, New Jersey Stroustrup, Basics of C++ Programming, Special Edition, Publisher: Addison-Wesley Professional.
6) D Ravichandran, Programming with C++, Tata McGraw-Hill Publishing Company Ltd, New Delhi - 110008

Reference Websites:

(1) www.sciencedirect.com
(2) www.ieee.org
(3) www.webpedia.com
(4) www.microsoft.com
(5) www.freetechbooks.com
(6) www.computer basics.com
(7) www.youtube.com