OBJECT ORIENTED PROGRAMMING IN C++

- Define program
- Define header file and reserved words
- Describe the structure of C++ program
- Know the use of statement terminator
- Explain the purpose of comments and their syntax
- Explain the difference between constant and variable
- Explain the rules for specifying variable names
- Know data types used in C++
- Define constant qualifier – const
- Explain the process of declaring and initializing variables
- Use type casting
- Explain the use of cout statement
- Explain the use of cin statement
- Define getch(), gets() and puts() functions
- Define escape sequence
- Use of escape sequence in programs
- Use I/O handling functions
- Use manipulators endl and setw
- Define operators and know their use in programs
- Use unary, binary and ternary operators in programs
- Define expression
- Define and explain the order of precedence of operators
- Define and explain compound expression

C++ is a general purpose programming language that supports various computer programming models such as object-oriented programming (oops) and generic programming. It was created by Bjarne Stroustrup in early 1980s at Bell Laboratories. Its main purpose was to make writing good programs easier and more pleasant for the individual programmer.

C++ supports modern programming techniques. It is commonly used for developing high performance commercial software, games and graphics related programs.

A computer program is a set of instructions that performs a specific task when executed by a computer. It tells the computer what to do. Everything a computer does is controlled by computer program. For example, Microsoft Word is a program that allows computer users to create documents and Skype is a program used to make calls free of charge to other people who are on Skype. Generally, programs are written in English oriented high level languages such as Visual Basic, Pascal, Java, C++, etc. A computer can only understand instruction in machine language which consists of 0s and 1s. Therefore, a program written in a high level language must be translated into machine language before execution. This task is achieved by software known as **compiler**. A program written in a high language is called **source code** and its equivalent program in machine language is called **object code**.

The C++ contains many header files. Header files contain information that is required by the program in which these are used. It has *.h* extension. Some examples of header files are *iostream.h*, *conio.h* and *math.h*. These files are included in the standard library of C++ compiler.

Preprocessor directive is used to include a header file at the beginning of program. Preprocessor directive is not a normal program instruction to be executed by the CPU. It is a code for the compiler to include a header file.

The syntax of preprocessor directive to include a header file in a program is:

include <name of header file>

It begins with a # sign, followed by the word *include* and then the name of header file is written within angled brackets.

For example, to include the header file *iostream.h* in a program, the code is

include <iostream.h>

Before starting the chapter, the students could be asked few questions about the “Programming Language” to explain what they understand about it.

The effect of this line is to copy all the contents of *iostream.h* header file into the program and make them available for use. Almost all the C++ programs perform input/output operations. Therefore, generally this header file is included in the programs.

The following is another example for including *math.h* header file in a program.

```
# include <math.h>
```

The *math.h* header file contains code for performing mathematical functions such as finding the square root of a number.

Reserved Words

Reserved words are special words which are reserved by a programming language for specific purpose in program. These cannot be used as a variable names. Some examples of reserved words of C++ are *if*, *void*, *break*, *while*, *case* and *char*. All the reserved words are written in lower-case letters. There are about 80 reserved words in C++ but this may vary depending on the version being used.

The reserved words of C++ are:

and	and_eq	asm	auto	bitand
bitor	bool	break	case	catch
char	class	const	const_cast	continue
default	delete	do	double	dynamic_cast
else	enum	explicit	export	extern
false	float	for	friend	goto
if	inline	int	long	mutable
namespace	new	not	not_eq	operator
or	or_eq	private	protected	public
register	reinterpret_cast	return	short	signed
sizeof	static	static_cast	struct	switch
template	this	throw	true	try
typedef	typeid	typename	union	unsigned
using	virtual	void	volatile	wchar_t
while	xor	xor_eq		

Structure of C++ Program

A C++ program has the following structure.

preprocessor directives

Teacher should recommend a suitable C++ compiler to the students. All the parts of the compiler editor should also be explained.

```
void main()
{
    body of main() function
}
```

A C++ program starts with preprocessor directives, followed by the line ***void main()*** function and then the body of the *main()* function is written within curly brackets ({ and }). Body of main() function consists of executable statements. These statements perform a specific task when the program is executed by the CPU. There is no restriction on the number of statement that can be written in the body of main() function.

To understand the structure of program, consider the following program.

```
# include <iostream.h>
void main()
{
    cout<<''Information Technology'';
}
```

This is the first line of the program and it is a preprocessor directive. This program prints a message on the screen. Therefore, *iostream.h* header file is included which contains the code for performing input/output operations.

This line identifies the main function of the C++ program. All C++ programs must have *main()* function. If a program consists of more than one function, the location of *main()* does not matter. In C++ program the *main()* function is always executed first.

This is a C++ statement. The meaning of this statement is to print the message "Information Technology" on the screen. This statement is written within the curly brackets and is called body of the main() function.

In C++, semicolon is a statement terminator. It marks the end of a statement. All the C++ statements must end with a semicolon.

The following statement was used in the previous program, notice that it ends with a semicolon.

Demonstrate how a program can be written, saved, compiled and executed in C++.

```
cout<<''Information Technology'';
```

If ';' is missing the compiler will give syntax error number and also the message that ';' is missing. The error number and message may vary depending on the compiler used.

3.1.3 Comments in C++ Program

All the programming languages allow comments in programs. Comments are explanatory statements that help the reader in understanding source code. Comments can be entered at any location in the program. Comments are ignored during program execution which means they are not executable statements.

There are two types of comments in C++. These are single-line comments and multiple-line comments.

The Single-line Comments //

The single-line comments start with // (double slash) and continue until the end of the line.

The following program demonstrates the use of single-line comments.

```
// This is a very simple C++ program.
# include <iostream.h>
void main()
{
     cout<<''Information Technology'';  // It prints a message on the screen.
}
```

The Multi-line Comments /* and */

This type of comment is used for entering multiple line comments in a program. The /* is used at the beginning of comments and */ ends it.

The following program demonstrates the use of multiple-line comments.

```
/* This is my first C++ program.
   It demonstrates the
   structure of C++ program.
*/
# include <iostream.h>
void main()
{
     cout<<''Information Technology'';
}
```

The /* and */ type comments can also be used to enter comments on a single line as shown below.

```
/* This is my first C++ program. */
```

3.2 C++ CONSTANTS AND VARIABLES

Constants and variables are used in programs to perform calculations of various types. Therefore, it is important to understand how they are used in computer programs.

3.2.1 CONSTANTS AND VARIABLES

Constant

In computer programming, a constant is a value that does not change during execution of program. A constant can be a number, a character or a character string. A character string is a sequence of any number of characters.

Some examples of constants are 42, 7.25, 's' and "Computer" respectively. In C++, a single character constant is written within single quotes and a string constant within double quotes.

Variable

A variable is a name of memory location where data is stored. Variables are used in computer programs to store values of different data types. The data stored in a variable may change during program execution

3.2.2 RULES FOR SPECIFYING VARIABLE NAMES

The following are the rules for naming a variable.

i. The first character of a variable name must be alphabet or underscore.

ii. The characters allowed in a variable name are:
   - Underscore (_)
   - Digits (0 to 9)
   - Upper-case letters (A to Z)
   - Lower-case letters (a to z)

   An upper-case letter is considered different from a lower-case letter. For example, the variable *SUM* is different from *Sum* or *sum*. The underscore is generally used to improve readability. For example, the variable ***overtime*** may also be written as ***over_time***.

iii. Special symbols such as $, @, %, #, etc. are not allowed.

iv. Blank space or comma is not allowed.

v. Reserved words of C++ are not allowed to be used as a variable name.

Teacher should also explain few more related programs according to the chapter topics.

3.2.3 Data Types

Data types are declarations of variables for storing various types of data. Data types have different storage capacities. In C++ programs data type of a variable must be defined before assigning it a value.

The data types used in C++ programming are integer, floating-point, double precision and character.

Integer

It is a data type that is used to define numeric variables to store whole numbers, such as, -3, 0, 367, +2081, etc. Integers represent values that are counted, such as number of students in a class. The short form of integer is *int*. Numbers that have fractional part, such as, 3.84 cannot be stored in an integer variable.

The following table shows the integer types, the number of bytes it takes in memory to store the value and the range of numbers it can store. It is for 32-bit word size.

int	4 bytes	-2147483648 to 2147483647
unsigned int	4 bytes	0 to 4294967295
short int	2 bytes	-32768 to 32767
unsigned short int	2 bytes	0 to 65535
long int	4 bytes	-2147483648 to 2147483647
unsigned long int	4 bytes	0 to 4294967295

Floating-point

It is a data type that is used to define variables that can store numbers that have fractional part such as 3.75, -2.1, 388.80, etc. These numbers are also known as real numbers. The short form of floating-point is *float*.

The following table shows the floating-point types, the number of bytes it takes in memory to store the value and the range of real numbers it can store.

float	4 bytes	-3.4^{-38} to 3.4^{38} (with 6 digits of precision)
double	8 bytes	-1.7^{-308} to 1.7^{308} (with 15 digits of precision)

The *float* type variables might occupy different number of bytes and their range might also be different depending on the computer and the compiler being used.

Character

It is a data type that is used to define variables that can store only a single character. One byte of memory is set aside in memory to store a single character. The short form of character is *char*. A variable of type *char* can store a lower-case letter, an upper-case letter, a digit or a special character. Some examples of characters that can be stored in a variable of type *char* are ‘a’, ‘+’, ‘%’ and ‘5’. Note that characters are written within single quotation marks.

2.2.4 The const qualifier (cont'd)

In C++ programming language, *const* defines a variable whose value cannot be changed throughout the program. When the *const* qualifier is used with a variable, it no longer remains a variable because its value will not be changed. A variable defined with the *const* qualifier must be assigned some value.

It has the following syntax.

```
const data_type variable = constant;
```

For example,

```
const int AGE = 34;
const float LENGTH = 7.5;
```

The variables of type *const* are generally written in upper-case letters.

2.2.5 Variable declaration and initialization

In C++, all the variables that are going to be used in a program must be declared before use. Declaring a variable means specifying the data type of a variable. It allows the compiler to decide how many bytes should be set aside in memory for storage of value that is going to be assigned to the variable in the program.

The following are some examples of declaring variables.

```
int x,y,z;
float length,breadth,sum;
char ch;
```

Here the variables*, x, y* and *z* are declared as of type *int*, *length* and *breadth* as of type *float* and *ch* as of type *char*.

A variable may be initialized at the beginning of a program when it is declared. Initializing a variable means assigning it an initial value.

The following are some examples of initializing variables in declaration statements.

```
int x=4,y=5,z;
float length=12.5,breadth=15.25,sum=0.0;
char ch='a';
```

In the first declaration statement, the variable *x* is initialized to integer constant 4 and *y* to 5. The second statement initializes *length* to 12.5, *breadth* to 15.25 and *sum* to 0.0. The last statement initialized the character variable *ch* to 'a'.

2.2.6 Type casting

Type casting is used in C++ to convert data type from one type to another. There are two types of type casting, implicit and explicit type casting.

Implicit Type Casting

Implicit type casting automatically converts a data type to another. This is explained by the following example.

Suppose variable *q* is declared as of type float and the following calculation is to be performed.

q = 15/6;

When this integer division is performed, the result will also become an integer value 2 which will be implicitly converted to floating-point value 2.0 and assigned to the variable *q*.

If one or both of the integer constants are converted to floating-point constant (14.0 or 6.0), this will perform division using floating-point mathematics. In this case, the result produced will be 2.5 and it will be assigned to *q*.

Explicit Type Casting

In explicit type casting, a special operator is used to convert one data type into another.

The general form for conversion is

(type) expression

Here, expression can be an arithmetic expression or a variable. This is explained by the following example.

Suppose, *a* and *b* are variables of type *int* and *q* is of type *float*. The integer value 15 is stored in *a* and 6 in *b* and the following division is to be performed.

q = a/b;

When this division is performed, integer math will be used and the result produced will be 2 which will be assigned to the variable *q*.

To obtain correct result, type casting should be used to convert at least one of the operands to type *float* as shown below.

q = (float)a/b;

Now, first the value stored in *a* will be converted to type *float* and then the division will be performed. The floating-point math will be used and the correct result 2.5 will be produced which will be assigned to *q*.

Similarly, the *int* type can also be used to convert a floating-point value stored in a floating-point variable into integer type by truncating the fractional part of the number.

3.5 Input/Output in C++

In computer programming, providing data into a program from outside source is known as input and output means to display some data on screen or save in a file on a storage device.

In C++, input/output (I/O) is performed by using streams. A stream is a sequence of data that flows in and out of the program. The ***cin*** and ***cout*** are the standard statements that use the *iostream.h* header file for performing I/O operations. Therefore, it must be included at the beginning of program.

The functions *getch(), gets()* and *puts()* are also used for handling input/output operations.

3.3.1 THE *cout* STATEMENT

The *cout* statement is used to output text or values on the screen.

The following is the syntax of *cout* statement.

cout<<character string/variable;

The keyword *cout* is used with insertion operator on its right side which is two less than signs (<<), followed by a character string or a variable. The insertion operator displays the contents of the character string or the value stored in the variable on the screen. It directs the output to the standard output device which is monitor. Note that the statement ends with semicolon. The insertion operator may be used more than once in a single statement.

The following program demonstrates the use of ***cout*** statement.

```
# include <iostream.h>
void main()
{
    int a;
    a=12;
    cout<<"The value stored in a is "<<a;
}
```

The output of the program will be

The value stored in a is 12

The first output statement will print the text and the second will print the value stored in variable *a*. The output of the two *cout* statements will appear on the same line.

3.3.2 THE *cin* STATEMENT

The *cin* statement is used to input data from the keyboard and assign it to one or more variables.

The following is the syntax of *cin* statement.

cin>>variable;

The keyword *cin* is used with the extraction operator on the right side which is two greater than signs (>>), followed by a variable. When this input statement is executed, it causes the program to wait for the user to input data. The data entered by the user is assigned to the variable.

The following program demonstrates the use of *cin* statement.

```
#include <iostream.h>
void main()
{
    int n;
```

```
    cout<<"Enter an integer:";
    cin>>n;
    cout<<The number you typed is <<n:
}
```

When this program is executed, the first *cout* statement prompts the user to enter an integer. The *cin* statement causes the typed number to be assigned to variable *n*. The last statement prints the number stored in *n* on the screen.

The execution of program is shown below.

```
Enter an integer:7
The number you typed is 7
```

More than one extraction operator can be used in a single *cin* statement to input more than one data values as shown below.

```
cin>>a>>b;
```

The values for variables *a* and *b* should be input with space between them.

These three functions are also used for handling I/O operations.

In some situations in programming, it is required to read a single character the instant it is typed without waiting for Enter key to be pressed. For example, in a game we might want an object to move each time we press one of the arrow keys. It would be awkward to press the Enter key each time we pressed an arrow key.

The *getch()* function is used for this purpose. The *get* means it gets something from an input device and *ch* means it gets a character. This function uses the *conio.h* header file. Therefore, it must be included in the program.

The following program demonstrates the use of *getch()* function to read a character from the keyboard and displays it on the screen.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    char ch;
    cout<<"Enter a character:";
    ch=getch;
    cout<<The character you typed is <<ch;
}
```

The execution of the program is shown below with the input character *a*.

```
Enter a character:
```

The character you typed is a

Note that, when this function is used, the input character *a* is read and stored in variable *ch* but it is not displayed on the screen.

If the user wants the typed character to be displayed on the screen then another similar function *getche()* is to be used. In this function the letter *e* is added which means to echo or display the input character on the screen.

When executing a program, some C++ compilers display the output for a very short time and immediately returns to the editing window. The user is not able to see the program output. Therefore, very often, getch() function is used at the end of the program so that the user is able to see the program output and has to enter any character to return to the editing window. Since the character entered is not used in program, there is no need to store it in a variable. This is shown below.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    char ch;
    cout<<"Enter a character:";
    ch=getche();
    cout<<The character you typed is <<ch;
}
```

The execution of the program is shown below with the input character 'h'.

Enter a character:

The character you typed is h

Note that, when this function is used, the input character 'h' is read and stored in variable *ch* and displayed on the screen.

These functions are used to handle input/output of character strings. The gets() function is used to input a string from the keyboard and the puts() is used to display it on the screen. In C++ strings are handled as arrays. These functions will be explained in Chapter 5 in which arrays are discussed.

Escape sequences are special characters used to control the output look on output devices. These characters are usually not printed. These are used inside the output statement.

An escape sequence begins with a backslash (\) followed by a code character. These are called escape sequences because the backslash causes an 'escape' from the normal way characters are interpreted in C++ programming language. Escape sequences are used for

special purposes in programming such as to begin printing on the next line, to issue a tab, to print special characters, etc.

3.5.2 COMMONLY USED ESCAPE SEQUENCES

The commonly used escape sequences are *\a, \b, \n, \r, \t,* \\, *\'* and *\".*

Suppose we want to print the following message on the screen.

```
cout<< "There are many versions of";
cout<< "Windows operating system.";
```

When the above two statements are executed, the output will appear on a single line as shown below.

There are many versions of Windows operating system.

If it is desired to display the output in two lines then *\n* escape sequence can be used in various ways to move cursor to the beginning of next line.

```
cout<< "\nThere are many versions of";
cout<< "\nWindows operating system.";
```

The same output can also be obtained by the following two statements.

```
cout<< " \nThere are many versions of\n";
cout<< "Windows operating system.";
```

A single output statement can also be used.

```
cout<< "\nThere are many versions of\nWindows operating system.";
```

Similarly, the \t escape sequence can also be used to tab over eight characters as shown in the following statement.

```
cout<< "C++\tis\ta\thigh\tlevel\tlanguage";
```

The output of this statement will be

C++ is a high level language

A list of commonly used escape sequences is given in the following table with their meanings.

\a	Produces alert (beep) sound
\b	Moves cursor backward by one position
\n	Moves cursor to the beginning of next line
\r	Moves cursor to the beginning of current line
\t	Moves cursor to the next horizontal tabular position
****	Produces a backslash
\'	Produces a single quote
\"	Produces a double quote

The following statement uses the \" escape sequence to print the word "Pakistan" in double quotation marks.

cout<<"This will print the word \"Pakistan\" in double quotation marks.";

The escape sequences \' and \\ are also used in the same way to print a single quote or a backslash.

Program 1: The following program reads three integers and prints their sum and product.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int x,y,z,sum,prod;
    cout<<"\nEnter first number:";
    cin>>x;
    cout<<"\nEnter second number:";
    cin>>y;
    cout<<"\nEnter third number:";
    cin>>z;
    sum=x+y+z;
     prod=x*y*z;
    cout<<"\nSum="<<sum;
    cout<<"\nProduct="<<prod;
     getch();
}
```

The execution of the program is shown below.

```
 Enter first number:3
 Enter second number:4
 Enter third number:5
Sum=12
Product=60
```

When this program is executed, it prompts the user to enter three numbers. The user enters the numbers, 3,4 and 5 separated by space and presses the Enter key. The program calculates the sum and product and displays on the screen.

Program 2: The following program reads the base and height of a triangle and prints area using floating-point values.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    float base,height,area;
    cout<< "\nEnter the base:";
    cin>>base;
    cout<< "\nEnter the height:";
    cin>>height;
    area=(base*height)/2;
    cout<< "\nThe area of triangle is "<<area;
    getch();
}
```

In this program, two separate input statements are used for reading the values for variables *base* and *height*. The execution of program is shown is below.

```
Enter the base:4.5;
Enter the height 6.2;
The area of triangle is 13.95
```

A manipulator is a command in C++ that is used for formatted output. It modifies the output in various ways. There are many manipulators available in C++. The most commonly used manipulators are ***endl*** and ***setw***. The *iomanip.h* header file should be included when manipulators are used in a program. Only the *endl* manipulator can be used without using *iomanip.h* file.

The *endl* manipulator has the same function as the *\n* escape sequence. It causes a linefeed in the *cout* statement so that the subsequent text is displayed on the next line.

The following program demonstrates the use of *endl* manipulator.

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
void main()
{
    cout<< "\nI am a student."<<endl;
    cout<< "I was born in 2001.";
```

```
    getch();
}
```

5

The output of the program will be:

```
I am a student.
I was born in 2001.
```

A single *cout* statement can also be used as shown below to obtain the same output.

```
cout<< "I am a student."<<endl<< "I was born in 2001.";
```

The *setw* manipulator is used in output statement to set the minimum field width.

It has the general form:

setw(n)

Here, *n* is an integer value that causes the number or text that follows to be printed within a field width of *n* characters. The number or text is right justified within the set field width. It is commonly used in programs to align numbers or text on output.

The following program demonstrates the use of *setw* manipulator in a program.

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
void main()
{
    int price1=8540,price2=325,price3=27800;
    cout<< "Product        "<<setw(10)<< "Price"<<endl;
    cout<< "Hard disk      "<<setw(10)<<price1<<endl;
    cout<< "Mouse           "<<setw(10)<<price2<<endl;
    cout<< "Computer       "<<setw(10)<<price3<<endl;
    getch();
}
```

This program will print the values following the *setw* manipulator within a field width of 10 characters and they will be right justified.

The output of the program is given below.

```
Product          Price
Hard disk        8540
Mouse             325
Computer        27800
```

The same program can also be written using a single *cout* statement as shown in the next program.

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
void main()
{
    int price1=8540,price2=325,price3=27800;
    cout<< "Product          "<<setw(10)<<"Price"<<endl
        << "Hard disk        "<<setw(10)<<price1<<endl
        << "Mouse            "<<setw(10)<<price2<<endl
     << "Computer       "<<setw(10)<<price3<<endl;
    getch();
}
```

If *setw* manipulator is not used, the output will be:

```
Product      Price
Hard disk    8540
Mouse        325
Computer     27800
```

The commonly used header files are listed in the following table with their purpose.

iostream.h	Provides basic input/ouput operations such as cin and cout.
conio.h	Stands for console input/output. It manages input/output on console based applications. Console applications take input from keyboard and display output on monitor.
math.h	Provides basic mathematical operations such as sqrt(), pow(), etc.
string.h	Provides several functions to manipulate strings such as strcmp (compare string), strcpy (copy string), etc.
iomanip.h	Provides manipulator function such as setw(), endl, setprecision(), etc.
time.h	Provides date and time operations

An operator is a symbol that tells the computer to perform a specific operation. For example the '+' operator is used to add two numbers.

The following types of operators are commonly used in C++.

- Assignment operators
- Arithmetic operators
- Arithmetic assignment operators
- Increment/Decrement operators
- Relational operators
- Logical operators
- Ternary operator

Assignment operator is equal sign (=). It is used to assign value of an expression to a variable. It has the general form

variable = expression;

where expression may be a constant, another variable to which a value has previously been assigned or a formula to be evaluated. For example:

z = x + y;

When this statement is executed, the values stored in variables *x* and *y* will be added and the resulting answer will be stored in variable *z*.

Arithmetic operators are used to perform arithmetic operations that include addition, subtraction, multiplication, division and to find the remainder of integer division.

The types of arithmetic operators used in C++ programming are described with their operations in the table given below.

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operator

The modulo operator (%) gives the remainder after division of one number by another. For example,

a = 20 % 6;

will give the result 2 which will be assigned to the variable *a* because 6 will divide 20 by 3 with a remainder of 2.

Some more examples of modulo operator are given below.

13 % 4	will give the result 1
15 % 4	will give the result 3
10 % 5	will give the result 0
4 % 5	will give the result 4

In addition to equal (=) assignment operator, there are a number of assignment operators unique to C++ which are known as arithmetic assignment operators. These include +=, - =, *=, /= and %=.

Suppose *op* represents an arithmetic operator. Then, the arithmetic assignment operator has the following general form to assign value of an expression to a variable.

variable op = expression;

For example:

a += b;

It is equivalent to:

a = a + b;

The effect is exactly the same but the expression is more compact. The arithmetic assignment operators are described in the following table.

+=	Adds the right side operand to the left side operand and assigns the result to the left side operand	a+=b It is equivalent to a=a+b
-=	Subtracts the right side operand from the left side operand and assigns the result to the left side operand	k-=n It is equivalent to k=k-n
***=**	Multiplies the right side operand with the left side operand and assigns the result to the left side operand	prod*=n It is equivalent to prod=prod*n
/=	Divides the left side operand by the right side operand and assigns the result to the left side operand	n/=m It is equivalent to n=n/m
%=	Takes modulus and assigns the result to the left side operand	x%=y It is equivalent to x=x%y

The increment operator is ++ and it is used to add one to the value stored in a variable. The decrement operator is -- and it subtracts one from the value stored in a variable. The purpose of using these operators is simply to shorten the expression.

Examples:

++x and x++ are both equivalent to x = x + 1

- - x and x- - are both equivalent to x = x - 1

When increment or decrement operator is written before the variable, it is known as *prefix* and when it is written after the variable, it is known as *postfix*.

In certain situations, ++*x* and *x*++ have different effect. This is because ++*x* increments *x* before using its value whereas *x*++ increments *x* after its value is used.

As an example, suppose *x* has the value 3. The statement

y = ++x;

will first increment *x* and then assigns the value 4 to *y*. But the statement

y = x++;

will first assign the value 3 to *y* and then increments *x* to 4. In both cases *x* is assigned the value 4 but *y* is assigned different value.

The same rule applies to - -*x* and *x*- - as well.

Example of prefix increment:

```
#include<iostream.h>
#include<conio.h>
void main()
{
 int x,y;
 y=10;
 x=++y;
 cout<<"x: "<<x;
 cout<<"y: "<<y;
 getch();
}
```

Output:

```
 x: 11
 y: 11
```

Example of postfix increment:

```
#include<iostream.h>
#include<conio.h>
 void main()
```

```
{
 int x,y;
 y=10;
 x=y++;
 cout<<"x: "<<x;
 cout<<"y: "<<y;
 getch();
}
```

Output:

```
x: 10
y: 11
```

Relational Operators

Relational operators are used to compare two values of the same type. These operators are very helpful in computer programming when a flow of program is based on a condition. After evaluation of a relational expression, the result produced is **True** or **False**.

Six types of relational operators are available in C++ language. These are described in the following table.

= =	equal to	a==b
!=	not equal to	n!=m
<	less than	a<b+c
>	greater than	z>y
<=	less than or equal to	z<=(x+y)/2
>=	greater than or equal to	f>=a+12

The following are some examples of relational operators.

```
x > y
x = = y
z > x + y
z <= x + y
x!=10
```

If *x* has the value 12 and *y* has the value 7, then the condition in first line will become true since 12 is greater than 7.

The condition of second expression will become false since 12 is not equal to 7.

In the third expression, if *z* has the value 15, then the condition will become false since 15 is not greater than the sum of *x* and *y* which is 19.

In the fourth expression, 15 is less than 19. Therefore, the condition will become true.

In the last expression if *x* is any number other than 10 then the expression will be true. It will only be false when *x* is equal to 10.

Note: In C++ language true is represented by the integer 1 and false is represented by the integer 0.

Logical Operators

Logical operators are used in programming when it is required to take some action based on more than one condition. When two or more conditions are combined, it is called compound condition.

There are three types of logical operators. These are described below.

&&	AND
||	OR
!	NOT

- **Logical AND (&&) Operator**

It is used to form compound condition in which two relational expressions are evaluated. One relational expression is to the left and the other to the right of the operator. If both of the relational expressions (conditions) are true then the compound condition is considered true otherwise it is false.

The following is a compound condition that uses the && operator.

(x>=1) && (y<=10)

When this compound condition is evaluated, it will produce the result true if *x* is greater than or equal to 1 and y is less than or equal to 10. In other words, the result will be true if both conditions are true that is *x* is in between 1 and 10 otherwise it will be false.

The following compound condition will check whether the character stored in character variable *ch* is a lowercase letter or not.

(ch>='a') && (ch<='z')

The following truth table shows all the possible results of AND (&&) logical operator for two expressions.

False	False	False
False	True	False
True	False	False
True	True	True

- **Logical OR (||) Operator**

Logical OR operator is also used to form a compound condition. Just like the logical AND operator, one relational expression is to the left and the other to the right of the OR operator. The compound condition is true if either of the conditions is true or both are true. It is considered false only if both of the conditions are false.

The following is an example of compound condition that uses || operator.

(x>y) || (z==8)

This compound condition will produce the result true, if one of the conditions is true, that is, if *x* is greater than *y* or *z* is equal to 8. The result will only be false when both of the conditions are false, that is, *x* is not greater than *y* and *z* is not equal to 8.

The following truth table shows all the possible results of OR (||) logical operator for two expressions.

False	False	False
False	True	True
True	False	True
True	True	True

Logical NOT (!) Operator

The logical NOT operator is used with a single expression (condition) and evaluates to true if the expression is false and vice versa. In other words, it reverses the result of a single expression.

For example, the expression

!(z<10)

will be true if *z* is not less than 10. In other words, the condition will be true if *z* is greater than or equal to 10. Some programmers may prefer to write the above expression as given below which is easy to understand and has the same effect.

(z>=10)

The following truth table shows all the possible results of NOT (!) logical operator for one expression.

False	True
True	False

Ternary Operator ? : or Conditional Operator

The ? : operator is known as ternary or conditional operator. It returns one of two values depending on the result of a condition. Therefore, it is also known as conditional operator. It is very useful in situation where the programmer needs to choose one of two options depending on a single condition.

The general form of ternary operator is

Condition? Expression1 : Expression2;

The condition is evaluated. If it is true then Expression1 is evaluated otherwise Expression2 is evaluated.

The following is an example of using ternary operator.

(x>y)? x+y: x-y

When this code is executed, the condition *x>y* is evaluated. If the result is true then the value *x+y* is evaluated otherwise the value *x-y* is evaluated. It allows to execute different code based on the result of condition *x>y*.

The result of ternary operator can be assigned to a variable as shown below.

k = (x>y)? x+y: x-y;

This is demonstrated in the following program.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int x,y,k;
    x=15; y=10;
    k=(x>y)? x + y: x – y;
    cout<< “The value of k is ”<<k<<endl;
    getch();
}
```

The value of x is greater than y. Therefore, the condition (x>y) is true and k will be assigned the sum of x and y.

The output of the program will be

The value of k is 25

The ternary operator also allows to output text as shown below.

(x>y)? cout<< “x is greater than y”: cout<< “x is smaller than y”;

This is demonstrated in the following program.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int x,y,k;
    x=3; y=10;
    (x>y)? cout<< “x is greater than y”<<endl:
```

```
        cout<< "x is smaller than y"<<endl;
    getch();
}
```

In this program the value of x is smaller than y. Therefore, the condition (x>y) is false and the second output statement will be executed.

The output of the program will be

x is smaller than y

2.4.2 UNARY, BINARY AND TERNARY OPERATORS

There are three types of operators in C++ which are unary, binary and ternary operators.

The operator that works on a single operand is known as unary operators. Unary operators are -, ++, -- and the logical operator !.

Some examples of unary operators are

```
a= -b;
k++;
- -x;
```

The operators that work on two operands are known as binary operators. Binary operators are - , +, *,/, % and logical operators && and ||.

Some examples of binary operators are

```
a=b+c;
z=x*y;
k=d%e;
```

The conditional operator (? :) is known as ternary operator since it has three parts. These three parts are a condition and two expressions. The condition is evaluated and based on its result one of the two expressions is executed.

2.4.3 ORDER OF PRECEDENCE OF OPERATORS

Order of precedence of operators describes the rules according to which operations are to be performed in an expression.

In the following table, the operator that has the highest precedence is written at the top and the one with the lowest precedence is written at the bottom.

1.	*, /, %	Multiplication, Division and Remainder
2.	+, -	Addition and Subtraction
3.	<, <=, >, >=	Relational Operators

4.	= =, !=	Equal to and Not Equal to
5.	!	Logical NOT
6.	&&	Logical AND
7.	||	Logical OR
8.	=, *=, /=, %=, +=, -=	Assignment Operators

For example, in the expression

7 + 3 * 2

multiplication operator has higher precedence than the addition operator and thus will be evaluated first.

Parentheses are used in expression to change the order of evaluation of operators specified by operator precedence.

For example, in the above example, if it is required to perform addition before multiplication then parentheses can be used as shown below.

(7 + 3) * 2

When two operators of same precedence occur in an expression then they are evaluated from left to right as they occur.

When an expression contains arithmetic, relational and logical operators, the arithmetic operators are evaluated first, relational operators next and logical are evaluated last. Assignment operators are always applied at the end.

An expression is a combination of constants, variables and operators. Constants and variables are operands and operators tell the computer what types of action to perform on the operands.

There are three types of expressions in C++. These are arithmetic, relational and compound or logical expression.

An expression that contains constants, variables and arithmetic operators is called arithmetic expression.

For example, in the expression

a + 2 * b

a, *b* and 2 are operands and + and * are arithmetic operators. When this expression is evaluated, the values stored in variables *a* and *b* are substituted and the result produces another value.

The following are some more examples of arithmetic expressions.

5*(a – b)

Teacher should give some home assignments to the students at the end of the chapter.

(a + b)/(a – b) + a*b

3*a + 8 – b + (b + c)/2

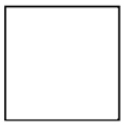
An expression that contains a relational operator to compare values of same type is called relational expression. Relational expressions are used in programming to create conditions based on which computer takes different path during program execution.

An expression that combines two or more conditions using logical operators, && or || is called compound expression.

The following is an example of compound expression.

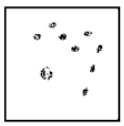
((ch>='a')&&(ch<='z'))||((ch>='A')&&(ch<='Z'))

The first compound condition ((ch>='a')&&(ch<='z')) checks whether the character stored in variable ch is a lower-case letter or not and the seconds compound condition ((ch>='A')&&(ch<='Z')) checks whether it is an upper-case letter or not. If one of the two compound conditions is true, the compound expression will be true because the two compound conditions are combined with an || (OR) operator.



- A computer program is a set of instructions to perform a specific task.
- Header files contain information that is required by the program in which they are used. It has *.h* extension.
- Reserved words are special words which are reserved by a programming language for specific purpose in program.
- In C++, semicolon is a statement terminator. It marks the end of a statement. All the C++ statements must end with a semicolon.
- Comments are explanatory statements that help the reader in understanding source code.
- A constant is a value that does not change during execution of program.
- A variable is a name of memory location where data is stored. Variables are used in computer programs to store values of different data types.
- Escape sequences are special characters used to control printing on the output device. These characters are not printed. These are used inside the output statement.
- Order of precedence of operators describes the rules according to which operations are to be performed in an expression.
- An operator is a symbol that tells the computer to perform a specific computing task.
- Assignment operator is equal sign (=). It is used to assign value of an expression to a variable.
- Arithmetic operators are used to perform arithmetic operations that include addition, subtraction, multiplication, division and to find the remainder of integer division.

- Relational operators are used to compare two values of the same type.
- Logical operators are used in programming when it is required to take some action based on more than one condition.
- An expression is a combination of constants, variables and operators. Constants and variables are operands and operators tell the computer what types of action to perform on the operands.
- An expression that combines two or more conditions using logical operators, && or || is called compound expression.



Q2. Select the best answer for the following MCQs:

i. Which of the following is ignored during program execution?
   a) Reserved word b) Constant
   c) Comment d) const qualifier
ii. What is the range of unsigned short integer?
   a) -2147483648 to 2147483647 b) 0 to 4294967295
   c) -32768 to 32767 d) 0 to 65535
iii. In C++ the expression sum=sum+n can also be written as:
   a) sum+=n b). sum=n++ c) sum=+n d) n+=sum
iv. Which of the following is equal to operator?
   a) += b) == c) = d) =+
v. Which of the following is an arithmetic operator?
   a) && b) % c) <= d) ++
vi. Which of the following operators is used to form compound condition?
   a) Arithmetic operator b) Assignment operator
   c) Relational operator d) Logical operator
vii. The number of bytes reserved for a variable of data type ‘float’ is:
   a) 2 b) 4 c) 6 d) 8
viii. How cursor is moved to the next tabular position for printing data?
   a) By using reserved word b) By using manipulator
) By using escape sequence d) By using header file

Q3. Write short answers to the following questions:

i. Define reserved words and give three examples.

ii. What is the purpose of using header file in program?

iii. State whether the following variable names are valid or invalid. State the reason for invalid variable names.

a) a123	b) _abcd	c) 5hml	d) tot.al
e) f3ss1	f) c$avg	g) net_weight	h) cout

iv. Why escape sequence is used? Give three examples with explanation.

v. Differentiate between relational and logical operators.

vi. What will be the output of the following statements?

a) cout<< “17/3 is equal to ”<<17/3;

b) cout<< “10.0/4 is equal to ”<<10.0/4;

c) cout<< “40/5%3*7 is equal to ”<<(40/5%3*7);

vii. Evaluate the following integer expressions.

a) 3+4*5	b) 4*5/10+8	c) 3*(2+7*4)
d) 20-2/6+3	e) (20-2)/(6+3)	f) 25%7

viii. Evaluate the following expressions that have integer and floating-point data types.

a) 10.0+15/2+4.3 b) 10.0+15.0/2+4.3 c) 4/6*3.0 +6

ix. What will be the output of the following program?

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int num1,num2,total;
    num1=13;
    num2=20;
    total=num1+num2;
    cout<< “The total of ”<<num1<< “ and ”
        <<num2<< “ is ”<<total<<endl;
    getch();
}
```

x. What will be the output of the following program?

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int n;
```

O 9

```
        n=10;
        cout<< “The initial value of n is ”<<n<<endl;
        n++;
        cout<< “The value of n is now ”<<n<<endl;
        n++; n++;
        cout<< “The value of n is now ”<<n<<endl;
        n--;
        cout<< “The value of n is now ”<<n<<endl;
        getch();
    }
```

i. Define variable and write the rules for specifying variable names.
ii. Why type casting is used? Explain the types of type casting with an example of each type.
iii. Define *endl* and *setw* manipulators and give an example of each.
iv. What is meant by precedence of operators? Write the operators with the highest precedence at the top and the lowest at the bottom.

i. Practice all the Example programs given in the chapter.
ii. Write a program that reads four integers and prints their sum, product and average.
iii. Write a program that reads length and breadth of a rectangle and prints its area.
iv. Write a program that reads temperature in Fahrenheit and prints its equivalent temperature in Celsius using the following formula.

$$c = 5/9(f - 32)$$