1.1 C++ Fundamentals

➢ Character-set, Identifiers & Keywords in C++

From this Unit onwards, we will actually start **studying about various** elements of C++ programming language. Let us start

The C ++ character-set

A character denotes any alphabet, digit or special **symbol used to** represent information. The C++ character-set can be defined as a 'set of ' **permissible** under C++ language to represent information. Under C++ language, **the upper-case** letters A to Z', the lower-case letters 'a to z\ the digits '0 to 9' and **certain special symbols** are all valid characters that can be used as building blocks to **construct basic** program-elements.

Depicted below is the entire C ++ character-set:

Alphabets :	**A , B,C,…….,Y,Z** **a,b,c…….,y,z**
Digits :	**0, 1, 2, 3, 4, 5, 6, 7, 8, *9***
Special Symbols: :	**+ - * / = % # &** **! ? ^ “ ‘ ~ \ | < >** **[ ] { } : ; . , _ (blank space)**

Identifiers and Keywords

In a C++ program (i.e., a program written in C ++ language), every word is either classified as an identifier or a keyword.

Identifiers, as the name suggests, are used to identify or name various program-elements such as **variables, symbolic constants, functions** ,etc.

On the other hand, **Keywords** are a kind of **reserved words** which have standard, predefined meanings in C++.

Now, let us discuss both these concepts, viz., identifiers and keywords in detail as follow.

Identifiers

As stated before, identifiers are names that are given to various program-elements such as **variables, symbolic constants, functions, arrays, etc.** There are certain rules regarding identifier names in C++, as stated below :

➢ Identifiers must consist of **letters and digits**, in any order, except that the first character must be a letter.

➢ Both **upper-** and **lower-case** letters are permitted, though common usage .the use of lower-case letters for most types of identifiers.

Upper- and **lower**-case letters are **not interchangeable** (i.e., an uppercase letter is not equivalent to the corresponding lower-case letter). Thus, the names **rate, Rate and RATE** denote different identifiers .It is a general practice to use lower or mixed case for variable and function names, and upper-case for symbolic constants.

- The **underscore character (_)** can also be included, and is considered be a letter. An underscore is often used in the middle of an identifier. An identifier may also begin with an underscore, though this is rarely done in practice.
- An identifier can be arbitrarily long. Some implementations of C++ recognize only the first eight characters, though most implementations recognize more (typically, 31 characters). Additional characters are carried along for the programmer's convenience.
- An identifier should contain enough characters so that its meaning is readily apparent. On the other hand, an excessive number of characters should be avoided.
- There are certain reserved words called keywords that have standard predefined meanings in C++. These keywords can be used only for their intended purpose they cannot be used as programmer-defined identifiers It is required to note here that the **keywords are all lower-case**. Since uppercase and lower- case characters are not equivalent. It is possible to utilize an upper-case keyword as an identifier Normally, however, this is not done as it is considered a poor programming practice

Examples of **valid identifiers** :
z xll sum_l names area tax rate

Examples of invalid identifiers :

7th	The first character must be **a letter**.
"a"	Illegal characters **(")**.
order-no	Illegal character **(-).**
error flag	Illegal character(blank space)

Keywords

Stated before, keywords are the words whose **meaning has already** been explained to the C++ compiler (or in a broad sense to the **computer) .The** keywords cannot be used as identifier names because, if we do so, we are **trying to assign** a new meaning to the keyword, which is not allowed by the computer. The keywords are also **called 'Reserved Words'.**
Following *is* the list of keywords in C++ :

and	**and _eq**	**asm**	**auto**	***bitand***
bitor	**bool**	**break**	**case**	**catch**
char	**class**	***compl***	**const**	**const_cast**
continue	**default**	**delete**	**do**	**doable**
dynamic	**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**	**using**
union	**unsigned**	**virtual**	**void**	**volatile**
while	**wchar_t**	**dynamic_cast**		

Fig:C++ keyword

1) Explain **Identifiers** and **keywords** ?

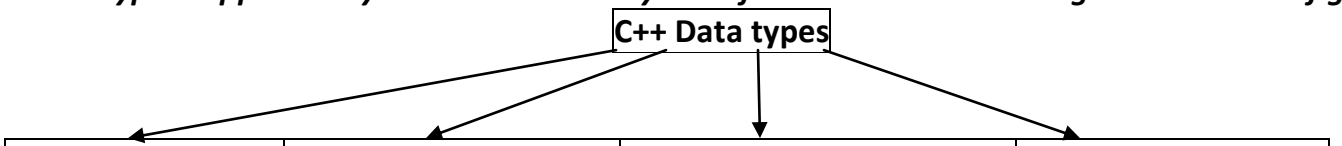
2) What is Identifiers ?what are rule governing naming of an Identifiers?

➢ Data Types in C + +

When programming, we store the variables in our computer's memory, but the computer must know what we want to store in them since storing a simple number, a letter or a large number is not going to occupy the same space in memory.

Our computer's memory is organized in bytes. A byte is the minimum amount of memory that we can manage. A byte can store a relatively small amount of data, usually an integer between 0 and 255 or one single character. But in addition, the computer can manipulate more complex data types that come from grouping several bytes, such as long numbers or numbers with decimals. Next you have a list of the existing fundamental data types in C++, as well as the range of values that can be represented with each one of them.

Data Types supported by C++ can be broadly classified under various categories as shown fig.



Build-in type	User defined type	Derived type	
int data-type. char data-type. float data-type.	enum (enumeration) data-type structure union class	Array Function Pointer, Reference (discussed in subsequence chapters 3 & 4)	**bool (boolean) type**.

✓ Build-in data type :

(1) int data-type.
(2) char data-type.
(3) float data-type.
(4) double data-type.

Now, let us discuss each one of these data-types in detail as following.

(1) int data-type

int data-type represents the whole number (i.e., integer) quantities. Integers are required not to contain a decimal point or an exponent. Range -32768 to +32767.

eg: int a=1000;

int data-type can also be further sub-divided into two broad categories, viz.,

(i) short int and long int .

(ii) signed int and unsigned int.

(i) Short int and long int :

➢ The qualifier short, when placed in front of the int declaration as C++ system that the particular variable being declared will be used fairly small integer values(the topics of declaration and variable are discuss letter in this chapter). The motivation for using short variables is primarily one of conserving the computer's memory space, which may be an issue in cases where the program needs a lot of memory and the amount of memory available is limited.

➢ On the other hand, the qualifier long, when placed in **front of** the int declaration, tells the

C++ system that the particular variable being **declared** will be used to store fairly large integer values, long integers cause the **program to** run a bit slower, but the range of values that we can use is expanded tremendously. (range is -2147483648 to +2147483648)

(ii) signed int and unsigned int:

- In case of **signed int** (or even in case of ordinary int, short int, long int as by default even int is signed int)the left most bit (of computer-memory) is reserved for the sign. However,

In case of **unsigned int**, there is no such kind of reservation and so all of the bits are used to represent the numerical value. Thus, an unsigned int can be approximately twice as large as an ordinary int.

- One important thing to be noted here is that unsigned int is used only when it is known in **advance** that the value stored in a given integer-variable will always be positive.

(2) char data type

char **data-type** is used to represent single characters. These character value is enclosed in pair of single quotes.

```
char a='A' ;
char b='a';
```

Hence, the char type generally requires only one byte of memory that is 8 bit number formats, they will have range specified as -128 to +127 in signed format signed char & unsigned char ,both occupying one byte each ,but different ranges. signed char range is -128 to +127 & unsigned char is 0 to 255.

(3) float data-type

- float data-type (also called floating point) represents values containing **decimal** places. A floating point value is distinguished by the presence of a decimal point. It is permissible to omit digits before the decimal point, or digits after the decimal point, but obviously not permissible to omit both. The values **3., 125.8 and -.0001** are all valid examples of floating point values.
- Floating point values can also be expressed in so-called scientific notation. The value **1. 7e4** is a floating point value expressed in this notation, and represents the value **1.7×10^4**. The value before the letter **e** is known as the **mantissa** while the value that follows the letter e is called the **exponent.** This exponent, which may be preceded by an optional plus or minus sign represents the **power of 10** that the mantissa *is to be* multiplied by. So, in the **value 2 *.25e-3*** *the 2.25 is (he* value of the mantissa and -3 is the value of the exponent. This value represents **$2.25 * 10^{-3}$** or **0.00225**. Incidentally, the letter **e** which separates the mantissa from the exponent can be written in either lower- or upper-case.
- A float occupied 4 byte in memory and can range from -3.4e38 to +3.4e38.

(4) double data-type :

double data-type is very similar to type float. It is used whenever the accuracy provided by a float variable is not sufficient. Variables **declared be** of type. double can store nearly twice as many significant digits as the float data type can double data type occupies 8 byte in memory & has range from -1.7e308 to+1.7e 308 a variable of double type can be declared as

double i,j;

if the situation demands usage of real number which lie even beyond the range offered by double data type then there exists a long double which can range from -1.7e 4932 to +1.7e4932 .A long double occupies 10 bytes in memory.

✓ User defined data type :

Structure ,union,class(discussed in chapter 4) & enumeration are User defined data type.

enum (enumeration) data-type

In addition to the predefined data types such as int and char, C++ allows you to define your own special data types. An enumeration type is an integral type that is defined by the user and has the **syntax** as :

enum typename {enumerator-list};

enum :C + + keyword
typename :It stands for an identifier that names the type being defined
enumerator-list :It stands for a list of names for integer constants.

For example, the following defines the enumeration type Semester, specifying the three possible values that a variable of that type can have :

enum Semester {WINTER,MONSOON,SUMMER};

We can **then** declare variables of this above type as :

Semester s1, s2;

& we can use those variables and those type values as we would with predefined types as :

```
/* Program to illustrate Enumerated Data Type */
#include<iostream.h>
enum Semester{WINTER, MONSOON, SUMMER};
    int main( )
    {
        Semester s1, s2 ;   // Declaration of variables of type Semester
        s1 = MONSOON ;
        s2 = WINTER ;
        if (s1 == s2)
        cout <<"same semester"<<endl;
        return 0 ;      }
```

OUTPUT

Above program will not anything because s1 is not equal to s2

✓ bool (boolean) data-type :

This is a new addition to C++. The data type bool gets its **name from** George Boole, a 19th century English mathematician who invented the concept of **rang** logical operators with true or false values. These values are often called **boolean** values.

This data type can take only two values true or false .it is most commonly used to hold the results of comparisons. For example :

```
bool a, b;
int x = 10, y = - 20, z = 30;
a = x < y;
b = y >= z;
```

Here, a gets a value true, whereas, b gets a value f al
By definition, true has a value 1 (when converted to an integer**) and** false has a value 0 (when converted to an integer). Conversely, integer is **can be** implicitly converted to bool values : non-zero integers. convert to true AND ZERO integers convert to false.

programming Example (along with its Output) :

```
#include <iostream.h>
#include<conio.h>
 int main()
{/ /  prints the value of a boolean variable
bool flag = false;
cout << "flag =" << flag << endl;
flag = true;
cout << "flag =  << flag << endl;
return 0; }
```

```
/**********Output ***************/
flag =  0
flag =  1
```

here that the value false is printed as the integer 0 & the value true is printed the integer 1.
NOTE:cout<< and endl used in this program are discussed later in this chapter. At It sent, just for understanding, consider cout<< .

✓ TYPE CONVERSION :

C++ converts means conversion of one data type to other data-type(e.g. integral types into floating point types).two type of Type conversion

(1) Automatically Converted :

In c++ ,if we have mixed mode expression(i.e different type of data in one expression) then the lower data-type variable is automatically converted to the data type of the higher data-type variable .

/* Program for illustration of Type Conversions*/

```
#include<iostream.h>
   int main( )
  {
    int n = 22;
   float x = 3.14159;
   x= x+ n; // the value 22 is automatically converted to 22.0
   cout << x - 2;// the value 2 is automatically converted to 2.0
   return 0;  }
```

```
23.14159
```

(2) Type Casting :

Converting from integer to float like this is what one would expect and is usually taken for granted. But converting from a floating point type to an integer type is not automatic.

Syntax: **(cast-type) expression;**

Or

cast-type(expression);

cast-type :refer to the data-type to which we desire to convert the expression to and,
expression :is any valid c++ expression or it may be a single variable.

➢ **Program Simple Type Casting**

```
#include<iostream.h>
  int main( )
   {
       // casts a double value as an int
       double v = 1234.56789;
        int n = int (v);
        cout << " v = " << v <<", n = " << n ;
        return 0; }
```

v=1234.57 ,n=1234

The double value 1234.56789 is converted to the int value 1234.

At a Glance……..

Name	Bytes*	Description	Range*
char	1	character or integer 8 bits length.	**signed:** -128 to 127 **unsigned:** 0 to 255
int	*	Its length traditionally depends on the length of the system's Word **type**, thus in MSDOS it is 16 bits long, whereas in 32 bit systems (like Windows 9x/2000/NT and systems that work under protected mode in x86 systems) it is 32 bits long (4 bytes).	-32768 to 32767
Short int	2	integer 16 bits length.	**signed:** -32768 to 32767 **unsigned:** 0 to 65535
Long int	4	integer 32 bits length.	**signed:**-2147483648 to 2147483647 **unsigned:** 0 to 4294967295
float	4	floating point number.	-3.4e38 to+ 3.4e38(7 digits)
double	8	double precision floating point number.	-1.7e308 to+1.7e308(15digits)
long double	10	long double precision floating point number.	-1.2e4932 to+1.2e4932 (19 digits)
bool	1	Boolean value. It can take one of two values: true or false NOTE: this is a type recently added by the ANSI-C++ standard. Not all compilers support it. Consult section **bool type** for compatibility information.	**true** or **false**

* Values of columns Bytes and Range may vary depending on your system. The values included here are the most commonly accepted and used by almost all compilers.In addition to these fundamental data types there also exist the pointers and the **void** parameter type specification

QUESTIONS

1) Explain Various **data types** of c++ with example.

2) Enumeration data-type **(may 09)**

➢ Constants, Variables & Declarations in C++

The alphabets, numbers and special symbols when properly combined from constants ,variable and keyword.

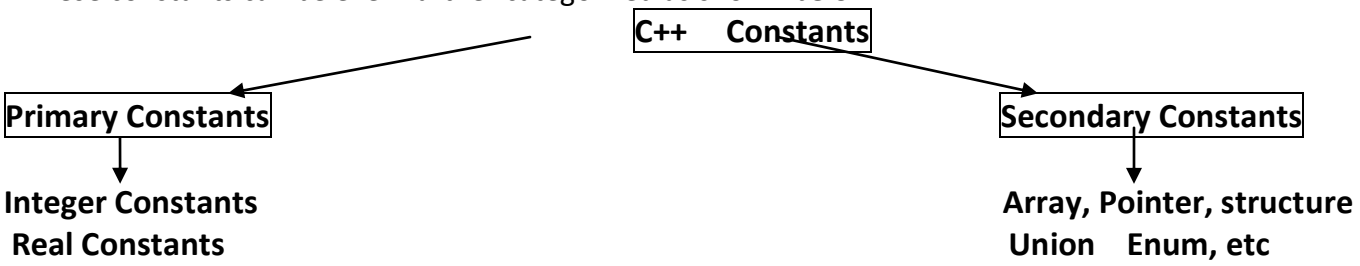
✓ Constants

A constant can be defined as a **quantity that doesn't change during the execution of program.** Now, let us discuss these concepts of Constants in C ++ can be divided into two major categories.

(a) Primary Constants.

(b) Secondary Constants.

These constants can be even further categorized as shown **below:**



In this unit we would restrict our explanation to only Primary Constants namely, Integer, Real and Character Constants. Detailed below are each of these constants :

(I) Integer Constants :

Integer Constants are whole numbers without any fractional part. The following rules **apply** for constructing integer constants in C++ :

(a) An integer constant must have **at least one digit.**

(b) It must not have a **decimal point.**

(c) It could be either **positive or negative.**

(d) If no sign precedes an integer constant it is assumed to be **positive.**

(e) **No commas or blanks** are allowed within an integer constant.

(f) The **allowable range** for integer constants is **-32768 to +32767.**

Examples of valid integer constants :

0 1 743 5280 32767 9999

Examples of invalid integer constants :

12,24 5 - Illegal character (,).

36.0 - Illegal character (.).

10 20 - Illegal character (blank space).

123-45-6789 - Illegal character (-)

0900 - The first digit cannot be a zero.

(II) Real Constants :

Real constants are often called Floating Point Constants. Real Constants could **be written** in two forms*, Fractional* Form and *Exponential* Form.

• **Fractional-Form of Real Constants** :

Following rules must be observed while constructing real constants expressed in fractional form

(a) A real constant must have **at least one** digit.

(b) It must have a **decimal point.**

(c) It could be either **positive** or **negative.**

(d) **If no sign precedes** the constant then it is assumed to be **positive.**

(e) **No commas or blanks** are allowed within a real constant.

• **Exponential-Form of Real Constants** :

The exponential form of representation of real constants is usually used if the value of the constant is either **too small or too large**. It however doesn't restrict us in any way from using exponential form of representation for other real constants.

In exponential form of representation, the real constant is represented in two parts : The part appearing **before 'e'** is called **mantissa** whereas the part following e* is called **exponent.**

Follow. rules must be observed while constructing real constants expressed in exponential form

(**a**) The **mantissa** part and the **exponential** part should be **separated by** a **letter e.**

(**b**) The **mantissa part** may have a **positive** or **negative** sign.

(c) Default sign of mantissa part is **positive.**

(d) The **exponent** must have **at least one digit** which must be a **positive or negative integer.** Default sign is positive.

(e) Range of real constants expressed in exponential form is **-3-4e38 to +3.4e38.**

➢ Examples of **valid** real constants :

0.	**1.**	**0.2**	**827.602**
50000.	**0.000743**	**12.3**	**315.0066**
2E-8	**0.006e-3**	**1.6667E+8**	**.12121212e12**

➢ Examples of **invalid** real constants :

1	-Either a decimal point or an exponent most be present.
1,000.0	-Illegal character (,).
2E+10.2	- The exponent must be an integer quantity (it cannot contain **a** decimal point.)
3E 10	-Illegal character (blank space) in **the exponent.**

(III) Character Constants :

A character constant is a single character, enclosed in apostrophes (i.e., single quotation marks). The following rules apply for constructing character constants in C++ :

(a) A character constant is either a **single alphabet,** a **single digit** or a **single special symbol** enclosed within **single inverted commas.**

(b) The **maximum length** of a character constant can be **one** character.

Examples of valid character constants :

'a' 'F' '7' '='

✓ Variables in C++

A variable can be defined as "**a quantity that varies during program execution".** A variable is a symbol that represents a **storage location in the computer's memory**. The information that is stored in that location is called the value of the variable. One common way for a variable to obtain a value is by an assignment. This has the syntax :

variable = expression;

First, the **expression is** evaluated and then the resulting value is assigned to the variable. **The equal "=" is the assignment operator in C+ +** .

Rules apply in variable:

- **Length of** variable name could be 1 to 8 characters some compilers allow variable name whose length could be up-to 40 character.
- Variable name could be combination of alphabets, digits & special symbol **underscore(_).**
- 1st character in variable name must be an **alphabet**.
- **No commas** or **blank space** are allowed within variable name.
- No special symbol other than an **underscore(_)** can be used in a variable name .

si_int m_hra pop_e_89

- Example of variable: int print;
  char name;
  float name;

✓ Declarations In C++

- **A declaration** associates a group of variables with a specific data-type. All variables **must** be declared before they can appear in executable statements.
- **A declaration** consists of a data-type, followed by one or more variable names, **ending with a** semicolon .**Example :**

int a.b,c;
float rootl, root2;

Here **a,b,** and **c** are declared to be integer variables, **rootl** and **root2** are floating-point **variables** and **flag** is a char-type variable.

➢ The first program in c++ :

```
#include<iostream.h>
#include<conio.h>
int main( )
{
cout<<"the 1st c++ program";
return 0;
}
```

```
/*****output*******/
the 1st c++ program
```

- **Output Operator : (Cout)**

cout<<"the 1st c++ program;

This statement causes the string in Quotation marks to be displayed on the screen. the standard output stream in c++.hear the standard output stream represent the screen. <<is insertion or put operator. **(**discussed in subsequence chapters 2).

- **Standard Input :(Cin)**

cin>>no1;
is an input statement and cause the program to wait for the user to type in number. the operator of extraction (>>) or get from operator. it extracts(or takes) the values from the keyboard and assign it to the variable on right .

- **A special mention of header file required for input and output in c++:**

#include<iostream.h> : This header file contains declarations that are needed by count & cin objects and << & >> operators. Without these declarations, the compiler won't recognize cout & cin .

OPERATORS

Operators:

An operator is a symbol that operates certain data type. The set of values ,on which the operations are called as the operands. The operator thus operates on an operand. Operators could be classified as "unary", "binary" or "ternary" depending on the number of operands i.e. one ,two or three respectively.

Basic Operators in C++ :

1) **Arithmetic operators .**
2) **Unary operators.**
3) **Increment and Decrement operators.**
4) **Relational operators.**
5) **Logical operators.**
6) **Assignment operators.**
7) **Ternary/Conditional Operator.**
8) **Bitwise operators.**

1) Arithmetic operators: (+, -, *, /, %)

The five arithmetical operations:

Operator	Description
+	Addition
-	subtraction
*	multiplication
/	division
%	remainder after integer division

- Binary arithmetic operators are +, -, *,/ & modulus operator %. Integer division truncates any fractional part.
- Modulus operator(%) : it is actually performs the operation of division on the two operands,but result that it gives is not the quotient of the division operation but it is reminder of the division operation.hence it is also called as remainder.
- The operators * / & % all have the same priority, which is higher than priority of binary addition (+) & subtraction (-). In case of an expression containing the operators having the **same precedence** it gets evaluated from **left to right**. This default precedence can be overridden by using a set of parentheses. If there is more than one set of parentheses, the innermost parentheses will be performed first, followed by the operations with-in the second innermost pair and so on.

- **Arithmetic operator Example:**

  34 + 5 = 39
  12 – 7 = 5
  15 * 5 = 75
  14 / 8 = 1
  17 % 6 = 5

➢ ***Program to illustrate Arithmetic Operators***

```
 # include<iostream.h>
 #include<conio.h>
 void main( )
{  int a =54, b = 20 ;  // Dynamic initialization
clrscr();
cout <<" \n a =  "<<a<<"  and b = "<<b ;
cout<<" \n a + b  =  " << a+b ;          // 54 + 20
cout<<" \n a - b  =  " << a - b ;        // 54 - 20
cout<<" \n a * b  =  " << a * b ;        // 54 * 20
cout<<" \n a / b  =  " << a / b ;        // 54 / 20
cout<<" \n a % b  =  " << a % b ;      // 54 % 20
getch();
 }
```

********** **Output** **********

a = 54 and b = 20
a + b = 74
a - b = 34
a * b = 1080
a / b = 2
a % b = 14

The last operator used in program is modulus operator. the modulus operator results in remainder from the division thus ,54%20=14 is the remainder after 54 is divided by 20.

- **Precedence and associativity of Arithmetic operators**

➢ Many times it happens that operators with equal precedence occur in the same expression . Here comes the concept of Associativity. The Associativity of the arithmetic operations is given following table .

Operator	Operations	Associativity
* / %	Multiplication Division Modulus	Left to Right
+ -	Addition Subtraction	Left to Right

➢ The Assciativity of the arithmetic operation is left to right . (*, / ,%) are same equal precedence and (+,-) are same equal precedence .

➢ ***Program to illustrate Associative law for Arithmetic Operators***

```
 # include <iostream.h>
 #include<conio.h>
  int main( )
 {     int a, b, c, d, e, f;
        clrscr();
        a = 5, b = 2 ;
        c = 3,  d = 7;
        e = 13;
        f = a * b - d % c + e ;
        cout<<"\n value  =  "<<f ;
        getch();
 }
```

********** **Output** **********

value=22

2) Increment and Decrement operators

➢ This operators also fall under the broad category of **unary operators** but are quite distinct than unary minus

➢ The **increment** and **decrement** operators are very useful in c++ language They are

extensively used in for and while loops.

- The SYNTAX_of these operators is given below:

++<variable name >
<variable name>++
--<variable name>
<variable name>--

The value of integral objects can be incremented with the ++ & --operators ,respectively

Operator	Operation
+ +	increment the value of the variable by 1
--	decrements the value variable by 1

- The operator ++ adds 1 to the operand and
- - - subtracts 1 from the operand.

- **Increment operator Example:**

++m and m++

- ***Program to illustrate Increment Operator***

```
# include <iostream.h>
#include<conio.h>
int main( )
{
int a = 0;
clrscr( );
cout << "\n"<<a;
a++;
cout << "\n"<<a;
a++;
cout << "\n"<<a;
a++;
cout << "\n"<<a;
a++;
cout << "\n"<<a;
a++;
cout << "\n"<<a;
getch( );
return 0;
}
```

```
********** Output **********
0
1
2
3
4
5
```

- ***Program to illustrate Increment Operator***

```
# include <iostream.h>
#include<conio.h>
int main( )
{
    int a = 0;
   clrscr();
    cout <<"\n" <<a++ ;
    cout <<"\n" <<a++ ;
    cout <<"\n" <<a++;
    cout <<"\n" <<a++;
    cout <<"\n" <<a++;
```

```
**********Output**********
0
1
2
3
4
5
```

```
        cout <<"\n" <<a ;
        getch();
        return 0;
  }
```

- **Decrement Operator Example:**

  --m and m--

➢ ***Program to illustrate Decrement Operator***

```
# include <iostream.h>
#include<conio.h>
int main( )
{
      int a = 5;
      clrscr();
      cout <<"\n" <<a-- ;
      cout <<"\n" <<a-- ;
      cout <<"\n" <<a-- ;
      cout <<"\n" <<a-- ;
      cout <<"\n" <<a ;
      getch();
      return 0;
}
```

```
********** output*********
5
4
3
2
1
```

- **Pre/Post Increment Decrement Operator**

Each of these operators has two version : a 'pre' and 'post' version.

➢ **'pre' version:** The 'pre' version performs the operation(***either adding 1 or subtracting 1***)On the object before the resulting value in its surrounding context.

➢ **'post' version:** The 'post' version performs the operation after the objects current value has been used .

So consider a variable say i, then see following table.

Operator	Representation
Pre-increment	+ + i
Post increment	i + +
Pre decrement	-- i
Post decrement	i - -

➢ ***Program to illustrate Pre/Post Increment Operator***

```
# include <iostream.h>
#include<conio.h>
int main( )
{       int a = 3;
clrscr();
cout <<"\n" <<a ;
cout <<"\n" <<a++ ;
cout <<"\n" <<++a ;
getch();
```

```
******* Output **********
3
3
5
```

```
return 0;
}
```

➢ ***Program to illustrate Pre/Post Decrement Operator***

```
    # include <iostream.h>
    #include<conio.h>
    int main( )
    {   int a = 7;
clrscr();
cout <<"\n"<<a ;
cout <<"\n"<<--a ;
cout <<"\n"<<a-- ;
cout <<"\n"<<--a ;
getch();
return 0;
}
```

```
**********output*******
7
6
6
4
```

- **Precedence Associativity of Increment/Decrement Operators Associativity (Asked In Exam :May 2008)**

Operator	Operations	Associativity
++	Increment	Right to Left
- -	Decrement	Right to Left

Let the consider small program based on this.

➢ ***Program to illustrate Associativity of Increment/Decrement Operator***

```
  # include <iostream.h>
  #include<conio.h>
  int main( )
  {  int x = 4, y = 9;
          int z;
          z = (x++) + (--y) + y ;
            cout << "\n Value  =  " <<z;
           z =(--x) + x + (y--) ;
          cout << "\n Value  =  " <<z;
           return 0;}
```

```
**********output*********
 Value=20
 Value=16
```

Precedence & Associativity w.r.t. Operators

Till now we have seen 3 types of operators , namely

- Arithmetic operators
- Assignment operator
- Increment /Decrement operators

The Precedence between these 3 types of operators is given in following table,

Operations	Operator
Increment/Decrements	++ , - -
Arithmetic operators	*, / ,%
	+ , -

Assignment	=

Precedence Level

The Associativity of these operators is listed below,

Operator	Associativity
Increment/Decrements	Right to Left
Arithmetic	Left to right
Assignment	Right to Left

Fig:Associativity

3) Unary Operator

- C++ includes a class of operator that act upon a single operand to produce a new value. such operator are known as Unary operators .
- Unary operators usually precede their single operands, though some Unary operators are written after their operands.
- Example :

-743 -0x7fff -0.2 E-8
-root 1 -(x + y) -3*(x + y)

- **Precedence and associativity of Unary operator**

Operator	operations	Associativity
-	Unary minus	Right to Left
++ --	Increment Decrement	Right to Left
* / %	Multiplication Division Modulus	Left to Right
+ -	Addition Subtraction	Left to Right
=	Assignment	Right to Left

- **Program Unary Operator:**

- ***Program to illustrate Unary Operator***

```
# include <iostream.h>
#include<conio.h>
   int main( )
   {
int a = 8;
clrscr();
cout << "value =  " << - a ;
a = - a;
cout << "value =  " << - a ;
```

****** **Output** ******
Value = - 8
Value= 8

```
getch();
return 0;
}
```

➢ ***Program to illustrate precedence level of Unary Operator***

```
# include <iostream.h>
#include<conio.h>
   int main( )
   {
int a, b, c ;
clrscr();
a = 2 ;
b = 5 ;
c = 10 ;
cout << "value =  " << a + b * -c ;
cout << "value =  " << - c / b * c - a ;
cout << "value =  " << - a  + ++b % a ;
getch();
return 0;
}
```

```
**********output******
Value = -48
Value = -22
Value = -2
```

4) Relational operators:

- Relational operator are binary operators. they require 2 operands to perform their operations on. They establish a certain relation between the two operands for this they can also called as comparison operators.
- There are six relational operators in c++ .

Operator	Operation
>	Greater than
>=	Greater than or equals to
<	Less than
<=	Less than or equals to
= =	equals to
!=	not equals to

- **Precedence and associativity of Relational operators**

Operator	Operation	Associativity
>	Greater than	Left to Right
>=	Greater than or equals to	
<	Less than	
<=	Less than or equals to	

➢ These operators all fall within the same precedence group, which is lower than the arithmetic and unary operators.

➢ Closely associated with the above-mentioned relational operators are the following two equality operators:

Operator	Operation	Associativity

= = !=	equals to not equals to	 Left to Right

- The equality operators fall precedence group beneath the relational operators. These operators also have a left to right associativity .
- The value of the relational expression is of integer type and is 1, if the result of comparison is true and 0 if it is false.

- **Relational operator Example:**

14 > 8 has the value 1, as it is true
34 <= 19 has the value 0, as it is false

- ***Program to illustrate Relational Operator***

```
# include<iostream.h>
#include<conio.h>
int main()
{
int a,b,c;
a = 2;
b = 7;
clrscr();
cout <<"a = "<< a ;
cout<<"\n b = "<< b ;
cout<<"\n"<< (a<b) ;
cout<<"\n"<< (a>b) ;
cout<<"\n"<< (a<=b);
cout<<"\n"<< (a>=b);
cout<<"\n"<< (a==b);
cout<<"\n"<< (a!=b);
a = 12 ;
b = 5 ;
cout<<"a="<< a;
cout<<"\n b= "<< b;
cout<<"\n"<< (a<b);
cout<<"\n"<< (a>b);
cout<<"\n"<< (a<=b);
cout<<"\n" << (a>=b);
cout<<"\n"<< (a==b);
cout<<"\n"<< (a!=b);
getch();
return 0;
 }
```

```
********** Output *********
   a =2
   b = 7
   1
   0
   1
   0
   0
   1
   a=12
   b= 5
   0
   1
   0
   1
   0
   1
```

- ***Program to illustrate Precedence Level of Relational Operators***

```
# include <iostream.h>
#include<conio.h>
int main( )
{
int a =1 , b = 3 , c = 7 ;
cout <<" \n "<< a < b + c ;
cout <<" \n"<< c - b != a ;
cout <<" \n"<< a ++>= c ;
```

```
***********output***********
1
1
0
1
0
```

```
cout <<" \n"<< ++a = = b ;
cout <<" \n"<< c%b > a ;
cout <<" \n"<< a*b <= -c ;
return 0;
 }
```

5) Assignment Operators

- To assign a value means
- There are several different assignment operators in c++ . all of them are used to form assignment expression ,which assign the value of an expression to an identifier
- Most commonly assignment operator are = .
- **precedence** of assignment operator (**Right to left)**
- **syntax:**

  variable = expression

  example:

  a =3

  x=y

- c++ contains the following five **additional assignment operator(+ =,- =,* =, % =)**

Syntax:

expression1+= expression

is equivalent to:

expression1=expression1+expression2

similarly, the assignment expression

expression 1 -= expression2

is equivalent to:

expression1=expression1-expression2

- **Assignment operator example:**

Operator	Expression	Result
i + = 3	i = i + 3	i = 18
i - = 2	i = i – 2	i = 13
i * = 4	i = i * 4	i = 60
i / = 3	i = i / 3	i = 5
i % = 4	i = i % 4	i = 3

Table : Examples for Compound Assignment operators

- ***Program to illustrate the use of Compound Assignment Operator***

```
# include <iostream.h>
#include<conio.h>
int main( )
        {      int n  = 22;
        clrscr();
        cout <<"n = "<<n<<endl;
        n += 9 ;                    //adds 9 to n
        cout << "After n  + =  9 , n = " << n << endl ;
        n -= 5 ;                    // subtracts 5 from n
        cout << "After n  - =  5 , n = " << n << endl ;
        n *= 2 ;                    // multiplies n by 2
        cout << "After n  * =  2 , n = " << n << endl ;
```

```
********output*****
n = 22
After n  + =  9 ,n = 31
After n  - =  5 , n = 26
After n  * =  2 , n =52
After n  / =  3 , n =17
After n  % =  7 , n = 3
```

```
n /= 3 ;                    // divides n by 3
cout << "After n  / =  3 , n = " << n << endl ;
n %= 7 ;//reduces n to the remainder from dividing by 7
cout << "After n  % =  7 , n = " << n << endl ;
getch();
return 0;
}
```

6) Logical operators

- The logical operators && (AND), || (OR) allow two or more expressions to be combined to form a single expression.
- The expressions involving these operators are evaluated left to right, and evaluation stops as soon as the truth or the falsehood of the result is known.

Operator	Usage
&&	Logical AND. Returns 1 if both the expressions are non-zero.
||	Logical OR. Returns 1 if either of the expression is non-zero.
!	Unary negation. It converts a non-zero operand into 0 and a zero operand into 1.

- **Example of Logical && operator :**

a > b && x== 10

The expression on left is a > b and that on the right is x= =10. the above-stated ***whole expression evaluates to true(1) only if both***

- **Example of Logical || operator :**

a < m || a < n

The expression is true if one of the expression(either a<m or a<n**) is true**. or both of them are true. That is if the value of a is less thane that of m or n then whole expression evaluates to true. needless to say , it evaluates to true in case a less than both m and n.

- **Example of Logical != operator :**

the not operator takes single expression and evaluates to true(1) if the expression is false (0), and evaluates to false(0), and evaluates to false (0) if the expression is true(1). in other words it just reverses the value of the expression. for example

!(x >= y)

Note: All the expressions, which are part of a compound expression, may not be evaluated, when they are connected by && or || operators.

Expr1	Expr2	Expr1 && Expr2	Expr1 || Expr2
0	0	0	0
0	non-zero	0	1
non-zero	0	0	1
non-zero	non-zero	1	1

Table : Operation of logical && and || operators

7) Ternary/Conditional Operator :

- The conditional expressions written with the ternary operator "**? :**" provides an alternate way to write the if conditional construct. This operator takes three arguments.
- The syntax is:

condition ? expression1 : expression2

(true) (false)

- If condition is true(i.e.**Value is non-zero**), then the value returned would be expression1.
- Otherwise condition is false(value is zero), then value returned would be expression2.

- ***Program to find minimum Number***

```
# include <iostream.h>
#include<conio.h>
 int main( )
{  int a =3 , b = 7 , c = 0 ;
   clrscr();
   int temp, result;
   temp = a < b ? a : b ;
   result = temp < c ? temp : c ;
   cout<<"The minimum number in"<<a<<","<<b<<","<<c<<" is "<< result;
   getch();
   return 0;
}
```

**********output**********

```
The minimum number in 3,7,0 is 0
```

- **Program to find minimum Number**

```
# include <iostream.h>
#include<conio.h>
int main( )
{
  int a =3 , b = 7 , c = 0 ;
  clrscr();
  int result;
  result = ( a < b ? a : b ) < c ? ( a < b ? a : b ) : c ;
  cout<<"The minimum number in"<<a<<","<<b<<","<<c<<" is "<< result;
  getch();
  return 0;
  }
```

**********output**********

```
The minimum number in 3,7,0 is 0
```

9) Bitwise Operators

supports a full complement of bitwise operators. Since C was designed to take the place of assembly language for most programming tasks, it needed to be able to support many operations that can be done in assembler, including operations on bits. *Bitwise operation* refers to testing, setting, or shifting the actual bits in a byte or word, which correspond to the

char and **int** data types and variants. You cannot use bitwise operations on **float**, **double**, **long double**,**void**, **bool**,

C ++ provides six operators for bit manipulation; these may only be applied to integral operands,that is, char, short, int, and long, whether signed or unsigned.

There are 6 bit manipulation operators:

Operator	meaning
&	BITWISE AND
|	BITWISE OR
^	BITWISE XOR(EXCLUSIVE OR)
~	One's complement
>>	Right SHIFT
<<	LEFT SHIFT

Operator	Symbol	Form	Operation
bitwise AND	&	x & y	each bit in x AND each bit in y
bitwise OR	|	x | y	each bit in x OR each bit in y
bitwise XOR	^	x ^ y	each bit in x XOR each bit in y
bitwise NOT	~	~x	all bits in x flipped
left shift	<<	x << y	all bits in x shifted left y bits
right shift	>>	x >> y	all bits in x shifted right y bits

1) BITWISE AND Operator :

Think of the bitwise AND as a way to clear a bit. That is, any bit that is 0 in either operand causes the corresponding bit in the outcome to be set to 0. For example, the following function reads a character from the modem port and resets the parity bit to 0:

TRUE TABLE OF BITWISE AND Operator

X	Y	OUTPUT(X&Y)
0	0	0
0	1	0
1	0	0
1	1	1

- ***Program : Bit-wise AND Operator***

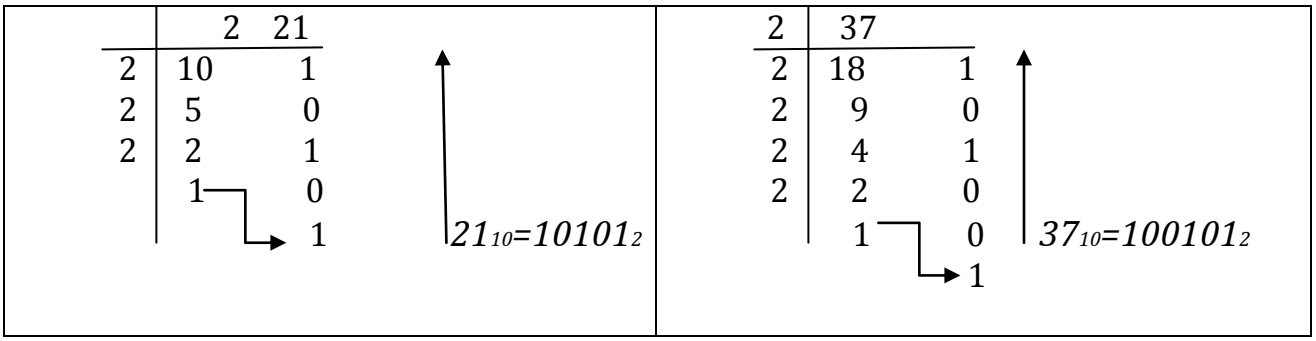
```
#include<iostream.h>
# include <conio.h>
void   main ()
{
      int a , b , c ;
      clrscr();
      a = 21 ;
```

```
**********OUTPUT**********
operand1 = 21
operand 2 =37
operand1 AND operand2=5
```

```
        b = 37 ;
        c = a & b ;
        cout<<"operand1 = "<<a<<endl ;
        cout<<"operand2 = "<<b<<endl;
        cout<<"operand1 AND
        operand2"<<c;
        getch( );
}
```

Explanation:

- ***The bitwise operator –'&' is used which will operator on two operands a and b. To have the bitwise AND operation performed, first the operands must be represented in the corresponding binary number system.***
-
- ***Decimal values gives are 21 and 37 respectively.***



These 2 number are 21_{10} and 37_{10} will be 16 bit format as,

$21_{10}=0000000000010101_2$ $37_{10}=0000000000100101_2$

The next statement is, c=a|b;

Here the bitwise OR operator is operating on the two operands. it will perform the ORing operation starting from the LSB and moving left towards the MSB. See fig

$21_{10}=0000000000010101_2$
| | | | | | | | | | | | | | | |
$37_{10}=0000000000100101_2$

0000000000110101_2

Fig : Bit-wise OR operator

The result is 110101 ignoring all the pervious 0's its equivalent decimal number can be calculated as,

110101_2
$=(1*210101_2$
$=32+16+4+1$
$=53_{10}$

- ***Program : Masking using bit-wise AND Operator***

```
# include<iostream.h>
```

**********OUTPUT**********

```
#include<conio.h>
void main ()
{
   int a , b ;
   clrscr();
   a = 33 ;
   b = a & 3 ;
   cout<<"operand1 = "<<a<<endl ;
   cout<<"Masking value = 3\n";
   cout <<"Masked Value ="<<b ;
   getch( );
}
```

```
operand1 = 33
Masking value = 3;
Masked Value=1
```

2) Bit-wise OR OPERATOR :

The next bit-wise operator is Bit-wise OR OPERATOR. Its fu

Ction or operation in similar to the logical or operator. But logical OR operates works on

TRUE TABLE OF BITWISE OR Operator

X	Y	OUTPUT(X | Y)
0	1	0
0	1	1
1	0	1
1	1	1

- **Program 1.31 : Bit-wise OR Operator**

```
# include <iostream.h>
# include <conio.h>
void main ()
{
   int a , b , c ;
   a = 21 ;
   b = 37 ;
   c = a | b ;
   cout<<"operand1 = " <<a<<endl;
   cout <<"operand 2 = "<<b<<endl;
   cout <<" operand1 OR operand2 ="<<c;
   getch();
 }
```

```
******Output**********
operand1 = 21
operand 2 =37
 operand1 OR operand2=53
```

3)Bitwise EXOR Operator :

TRUE TABLE OF BITWISE EX-OR Operator

X	Y	OUTPUT(X^Y)
0	1	0
0	1	0
1	0	0
1	1	1

- **Program : Bit-wise XOR Operator**

```
# include <iostream.h>
# include <conio.h>
void main ()
{
        int a , b , c ;
        a = 21 ;
        b = 37 ;
        c = a ^ b ;
        cout <<"operand1 = "<<a<<endl;
        cout<<"operand 2 = "<<b<<endl;
        cout <<"operand 1 XOR operand 2 = "<<c;
        getch( );
}
```

```
******OUTPUT*******
operand1 = 21
operand 2 = 37
operand 1 XOR operand 2 =48
```

- **Program : Toggling using bit-wise XOR Operator**

```
# include <iostream.h>
# include <conio.h>
void main ()
{
        int a , b ;
        a = 9 ;
        b = 11 ;
        cout<<"original value="<<a<<endl;
        cout<<"Toggled value=" <<(a^15);
        cout<<"\n original value="<<b<<endl;
        cout <<"\n Toggled value="<<(b^15);
}
```

```
*********OUTPUT**********
Original value=9
Toggled value=6
original value=11
Toggled value=4
```

➢ **Program : Toggling using bit-wise XOR Operator**

```
# include <iostream.h>
# include <conio.h>
void main ()
{
int a , b ;
clrscr() ;
a = 9 ;
b = 11 ;
cout<<"original Values= " <<a<<" " <<b ;
cout <<"\n Toggled Value (a^b) = "<<(a^b);
cout<<"\n Re-Toggled Value (a^b^b) = "<<(a^b^b);
a = 6 ;
b = 4 ;
cout<<"original Values   = "<<a<<"" <<b;
cout<<"\n Toggled Value (a^b)" <<(a^b);
cout <<"\n Re-Toggled Value (a^b^b) = "<<(a^b^b);
getch();
}
```

```
*********OUTPUT**********
original Values= 9 11
Toggled Value (a^b) = 2
Re-Toggled Value (a^b^b) = 9
original Values =  6  4
Toggled Value (a^b)" <<(a^b);
Re-Toggled Value (a^b^b) = 6
```

4) *One's complement*

TRUE TABLE *One's complement* Operator

X	**OUTPUT(~X)**
0	**1**
1	**0**

➢ ***Program : Binary Subtraction***

```
#include<iostream.h>
#include<conio.h>
void main ()
{
        int a = 17 , b = 26 ;
        int result ;
         a = ~ a ;
        a = a + 1 ;
        result = b + a ;
        cout <<b<<" -" <<a<<"=" <<result;
        getch();
```

```
    }
```

**********OUTPUT**********

```
26-17=9
```

5) Bitwise Right SHIFT OPERATOR :

- ***Program : Bit-wise Right Shift Operator***

```
#include<iostream.h>
void main ()
{
        int a = 63 ;
        cout<<"a="<<a<<endl;
        a = a >> 4 ;
        cout<< "Value of a,right-shifted 4 times = "<< a;
}
```

**********OUTPUT**********

```
a=63
Value of a,right-shifted 4 times=3
```

6) Bitwise LEFT SHIFT OPERATOR :

- ***Program: Bit-wise Left Shift Operator */***

```
# include <iostream.h>
void main ()
{
int a = 10 , b = -10 ;
cout<<"a = "<<a<<endl;
a = a << 3 ;
cout <<"Value of a,left-shifted 3 times = "<<a<<endl;
cout <<"b = " <<b<<endl;
b = b << 2 ;
cout <<"Value of b,left-shifted 2 times = "<<b<<endl;
}
```

**********OUTPUT**********

```
a = 10
Value of a,left-shifted 3
times=80
b = -10
Value of a,left-shifted 2 times=-
40
```

- ***Program: Multiplication & Division using Left Shift & Right Shift***

```
#include <iostream.h>
void main ()
{
        int a = 4 , b = 7 ;
        int x ;
        x = a << 1 ;
        cout<<"Number="<<a<<endl ;
        cout<< "Single Left Shift = "<<x<<endl ;
        x = a >> 1 ;
        cout <<"Single Right Shift = "<<x<<endl ;
```

*******OUTPUT*******

```
Number=4
Single Left Shift =8
Single Right Shift = 2
Number =7
Single Left Shift =14
Single Right Shift=3
```

```
        x = b << 1 ;
        cout<<"Number ="<<x<<endl;
        cout<<"Single Left Shift = "<<x<<endl ;
        x = b >> 1 ;
        cout<<"Single Right Shift = "<<x<<endl ;
}
```

- **Precedence and associativity of Relational operators**

Operator	Operation	Associativity
~	One's complement	Left to Right
<<	LEFT SHIFT	
>>	Right SHIFT	
&	BITWISE AND	
|	BITWISE OR	
^	BITWISE XOR(EXCLUSIVE OR)	

7) **The size of operator*:***

The sizeof operator returns the number of bytes the operand occupies in memory. The operand
may be a variable, a constant or a data type qualifier.

➢ ***Program to illustrate the use of sizeof operator***

```
# include <iostream.h>
int main( )
{
// Prints the storage sizes of the fundamental types :
cout << "Number of Bytes used :\n";
cout <<"\t char : "<< sizeof(char)<<endl;
cout <<"\t short: "<< sizeof(short)<<endl;
cout <<"\t long : "<< sizeof(long)<<endl;
cout <<"\t unsigned char :"<< sizeof(unsigned char)<<endl;
cout <<"\t unsigned short :"<< sizeof(unsigned
short)<<endl;
cout <<"\t unsigned int  : "<< sizeof(unsigned int )<<endl;
cout <<"\t unsigned long  : "<< sizeof(unsigned long )<<endl;
cout <<"\t signed char : "<< sizeof(signed char)<<endl;
cout <<"\t float : "<< sizeof(float)<<endl;
cout <<"\t double : "<< sizeof(double)<<endl;
cout <<"\t long double : "<< sizeof(long double)<<endl;
```

```
******OUTPUT*****
Number of Bytes used :
char :1
short:2
int: 4
long: 4
unsigned short  : 2
unsigned int  : 4
unsigned long  :4
signed char : 1
float : 4
double : 8
long double :10
```

```
return 0;
}
```

TYPE CAST OPERATOR:

C++ permits explicit type conversion of variable or expression using the Type cast operator
Syntax:

Type-name(expression)

Example:

```
Avg=sum / float(i);
```

➢ **Program Simple Type Casting */**

```
#include<iostream.h>
int main( )
 {
    // casts a double value as an int

     double v = 1234.56789;
      int n = int (v);
      cout << " v = " << v <<", n = " << n ;
      return 0;
   }
```

**********OUTPUT**********

```
V=123.57,  n=1234
```

a) What is the o/p of the following expression? If a=1,b=2,c=3 give the contents of a,b,c at the end of every expression.

1) a = a++ + b—

Ans: Above expression evaluated in this way,

a= (1) + (2)
a = 3
Hence a=3, b=1 , c= No changes.

2) b = b + c-- * --a

Ans: Above expression evaluated in this way
b = (2) + (3) * (0)
b = 2 + 0
b = 2
Hence a = 0 ,b = 2 and c = 2

(DEC 2007)

(1) what is the output of the following program

```
#include<iostream.h>
        int main( ){
        int b=3,a=2,ab=4;
        int I;
        int in'2'*'2';
        char ch='c';
        cout<<ch<<" '<<(++ch<<endl;
cout<<a<<" "<<(++a)<<endl;
 cout<<b<<" "<<(++b)<<endl;
 cout<<ab<<" "<<ab<<endl;
 cout<<a<<" "<<!!a;
 return 0:  }
```

(May 2007)

(2) What will be the ouput of the following code.

```
#include<iostream.h>
int main( )
{int a=3;
cout<<" "<<a<<endl;
cout<<" "<<(a++)<<endl;
cout<<" "<<(a++)<<endl;
return 0;              }
```

(DEC 2008)

(1) Explain logical operators. Write a C++ program to demonstrate the same.

(May 2009)

Data Input and Output:

➢ Data Input and Output in C:

in last Unit, we studied about library functions. Now, one interesting thing, which is indirectly linked with the library functions, needs to be noted here with regards to C language. As a programming language, C does not provide any input-output (I/O) statements. C simply has no provision for receiving data from any of the input devices (like say keyboard, floppy, etc.), nor for sending data to the Output devices (like say VDU, floppy, etc.). However, though C has no provision for input/output, it of course has to be dealt with at some point or the other. For this reason, a set of standard library functions provided by the operating system like UNIX for input and Output are borrowed and used by C.

Now, regarding the access of these functions. An input/output function can be accessed from anywhere within a program simply by writing the function name, followed by a list of arguments enclosed in parentheses. The arguments represent data items that are sent to the function. Some input/output functions do not *require* arguments, though the empty *parentheses must still* appear.

Most versions of C include a collection of header files that provide necessary information (e.g., symbolic constants) in support of the various library functions. Each file generally contains information in support of a group of related library functions. These files are entered into the program via an #include statement at the beginning of the program.As a rule, the header file required by the standard input/output library functions is called stdio.h. Now, let us study in detail the respective input/output functions as follows :

(1) printf ( ) :

printf() is a library function used to Output the data onto the standard Output device in c. The general form of printf looks like :

Syntax:

printf("format string", list of variables);

The format string can contain :

- characters that are simply printed .
- conversion specifications that begin with a % sign
- escape sequences that begin with a\ sign.

Let us see a very simple example.

➢ **To display the message.**

#include<stdio.h>
#include<conio.h>
void main ()
{
printf("welcome to the world of c");
getch();
}
	displays
	welcome to the world of c

Explanation :

The print statement ,has the string **"**welcome to the world of c". enclosed within double quotes. No format specifies are used

➢ **To display average and percentage**

#include<stdio.h>	Output of above program would be

#include<conio.h>
void main ()
{int avg= 346;
float per = 69.2;
printf("Average =%d\n Percentage=%f",avg, per);
}
	Average = 346
	Percentage=69.200000

Now after taking a look at above example, let us study how printf () function interprets the contents of the **format string**. For this it examines the format string from **left to right**. So long as it doesn't come across either a % or a \ it continues dump characters that it encounters, on to the screen.

In the above example Average = is dumped on the screen. The moment it comes across a conversion specification (represented by %) in the **format string** it picks up the first variable in the list of variables and prints its value in the specified format. In the example, the moment %d is met the variable avg is picked up and its value is printed.

Similarly ,the moment an **escape sequence** (represented by \) is met it takes the appropriate action. In this example, the moment \n is met it places the cursor at the beginning of the next line. This process continues till the end of **format string** is reached.

Now, let us learn in detail about conversion specifications & escape sequences as follows:

• Conversion Specifications :

In the above Example, the %d and %f used in the printf () are called conversion characters. They tell printf () to print the value of avg as a decimal integer and the value of per as a float. Following is the list of conversion characters that are commonly used with the input/output functions.

Conversion Character	Meaning
c	Data item is or is displayed as a single character.
d	Data item is or is displayed as a signed decimal integer.
e	Data item is or is displayed as a floating-point value with an exponent.
f	Data item is or is displayed as a floating-point value without an exponent.
g	Data item is or is displayed as a floating-point value using either e- type or f- type conversion, depending on value. Trailing zeros and trailing decimal point 1 will not be displayed.
o	Data item is or is displayed as an octal integer without a leading zero.
s	Data item is or is displayed as a string.
u	Data item is or is displayed as an unsigned decimal integer.
X	Data item is or is displayed as a hexadecimal integer without a leading zero.

• Escape Sequences :

We saw earlier how the newline character \n when inserted in a printf() ***format string***, takes the cursor to the beginning of the next line. This newline character is an 'escape sequence', so called because the backslash symbol (\) is considered an escape character : it causes an escape from the normal interpretation of a string, so that the next character is recognized as one having a special meaning.

The following example shows usage of \n and a new escape sequence \t called tab. A \t moves the cursor to the next tab stop. A 80 column screen usually has 10 tab stops. In other words, the screen is divided into 10 zones of 8 columns each. Printing a tab takes the cursor to the beginning of next printing zone. For example, if cursor is positioned in column 5, then printing a tab takes it to column 8.

Example:

```
main()
{
printf("you\t must\t be\t crazy\n to\t hate\t this\t book");
}
```

And here is the output:

```
you     must    be      crazy
to      hate    this    book
```

In above example, the cursor jumps over eight columns when it encounters the \t character. The \n character causes a new line to begin following 'crazy'. The tab and newline are probably the most commonly used escape sequences, but there are others as well.

The following table shows a complete list of these escape sequences :

Escape Sequence	Character it represents
\a	**Bell (alert)**
\b	**Backspace**
\t	**Horizontal tab**
\v	**Vertical tab**
\n	**Newline (line feed)**
\f	**Form feed**
\r	**Carriage return**
\"	**Double quotation mark (")**
\'	**Single quote (')**
\?	**Question mark (?)**
****	**Backslash (\)**
\0	**Null**

Amongst this table, the first few escape sequences are more or less self-explanatory. Some confusing ones are explained below :

i) \a- alerts the user by sounding the speaker inside the computer.

ii) \b- moves the cursor one position to the left of its current position.
iii) \f- form feed advances the computer stationery attached to the printer to the top of the next page.
iv) \r- takes the cursor to the beginning of the line in which it is currently placed.

Note: the last five escape sequences of the table. Characters that are ordinarily used as delimiters : the single quote, double quote, and the backslash can be printed by preceding them with the backslash. Thus, the statement :

```
printf("He said ,\"Let's do it!\"");
```

will print :

He said,"Let's do it!"

(2) scanf( ) :

The function scanf () is used to read data into variables from the standard input device, namely, a keyboard attached to a video terminal.

The general form of scanf () statement is as follows :

scanf("format string",list of addresses of variables);

where *format string* gives information to the computer on the type of data to be stored in the list of variables .For example, in the *statement*;

scanf("%d%f%c",&c,&a,&ch);

note that sending addresses of variables (addresses are obtained by using '&' the 'address of 'operator)to scanf function .this is necessary because the values received from keyboard must be dropped into variables corresponding to these addresses. The values supplied thought keyboard must be separated by either blank(s),tab(s),or newline(s).do not include these escape sequences in the format string.

All the format specification that we learned in printf() function apply for scanf() function as well.

Character Input and Output:

(1) getchar():

single character can be enter into the computer using c++ library function getchar. The getchar function is part of the standard c++ language i/o function library. It returns a single character from a standard input device (typically a keyword).The function does not require any arguments, through a pair of empty parentheses must follow the word getchar. In general terms, a reference to getchar function is written as

character variable =getchar();

character variable-some previasly declared character variable. the getchar function can also be used to read multicharacter strings, by reading one character at a time within a multipass loop.

(2) putchar():

single character can displayed (i.e. written out of the computer) using the c++ library function putchar this function is complementary to character input function getchar. Putchar function like getchar, is part of the standard c language i/o library. It transmits a single character to standard output device (typically a TV monitor or a time sharing terminal). The character being Transmitted will normally be represented as a character to function ,enclosed in parentheses following the word putchar

Putchar(character variable)

character variable refer previously declared character variable. The putchar function can be used to output a string constant by storing the string within a one dimensional character type array. Each character can then be written separately within a loop. The most convenient way to do this utilize a for statement.

(3) Gets() and puts():

C contains a number of other library functions that permit some from of data transfer into or out of the computer ,e.g. gets and puts function which facilitate the transfer of strings between the computer & the standard input/output devices.

Each of these functions accepts a single functions a single argument. The argument must be a data item that represents a string (e.g. character array).the string may include whitespace characters. In case gets, the string will be enterd from the keyboard, & will terminate with a newline character (i.e string will end when the user presses the RETURN Key)

The gets & puts functions offer simple alternatives to the use of scanf and printf for reading and displaying strings.

(4) getch(),getche(),getchar(),putch()&putchar():

some situation scanf() function are weakness. You need to hit Enter key before the function can digest what you have typed. However we often went a function that will read a single character the in stent it is typed without waiting for enter key to be hit. getch() and getche() are 2 function which serve this purpose. These function means it displays the character that you typed on screen . as against this 'e' is getche() function mean display character screen. getche() just returns the character that you typed on screen . getchar() works similarly and display character that typed on screen ,but unfortunately requires the enter key to be hit following the character to be typed.

Putch() and putchar() from the other side the coin they print a character on the screen .as for as working is concerned they are exactly same.they can output only one character at a time

- program to use of getch(),getche(),getchar(),putch()&putchar():

```
#include<stdio.h>
#include<conio.h>
void  main ()
{
char  ch='A';
clrscr();
putch(ch);
cout<<"press any key  to continue"<<endl;
ch=getch();//will not echo the character
cout<<"you enter "<<endl;
putchar(ch);//will display the character
cout<<"Type any character"<<endl;
getch();//will echo the character typed
cout<<"Type any character "<<endl;
getchar();//will echo character,must be followed by enter key
}
```

```
*****output*****
press any key to
continue
you entered a
type any character f
type any character f
```

➢ Data Input and Output in C ++

Communication through Console:

The console is the basic interface of computers, normally it is the set composed of the keyboard & the screen. The keyboard is generally the standard input device & the screen the standard.

Output and input device:

- In the iostream C++ library, standard input and output operations for a program are supported by two data streams: cin for input and cout for output.
- Therefore cout (the standard output stream) is normally directed to the screen and cin (the standard input stream) is normally assigned to the keyboard.
- By handling these two streams you will be able to interact with the user in your programs Since you will be able to show messages on screen & receive his/her input from keyboard.

1) Output (cout)

- The cout stream is used in conjunction with the overloaded operator << (a pair of "less than" signs).
- The << operator is known as **insertion operator** since it inserts the data that follows it into the stream that precedes it

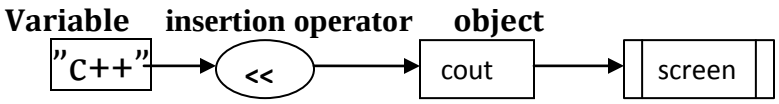


Fig : insertion operator

e.g.

cout << "Output sentence"; **// prints Output sentence on screen**

cout << 120; // **prints number 120 on screen**

cout << x; // **prints the content of variable x on screen**

In the examples above it inserted the constant string Output sentence, The numerical constant 120 and the variable x into the output stream cout. Notice that the first of the two sentences is enclosed between double quotes (") because it is a string of characters. Whenever we want to use constant strings of characters we must enclose them between double quotes (") so that they can be clearly distinguished from variables.

For example, these two sentences are very different:

cout << "Hello"; // **prints Hello on screen**

cout << Hello; // **prints the content of Hello variable on screen**

The insertion operator (<<) may be used more than once in a same sentence:

```
cout << "Hello, " << "I am " << "a C++ sentence";
```

This last sentence would print the message on the screen.

Hello, I am a C++ sentence

The utility of repeating the insertion operator(<<) is demonstrated when we want to print out a combination of variables and constants or more than one variable:

```
cout << "Hello, I am " << age << " years old and my zipcode is " << zipcode;
```

If we suppose that variable age contains the number 24 and the variable zipcode contains 90064 the output of the previous sentence would be:

Hello, I am 24 years old and my zipcode is 90064

It is important to notice that cout does not add a line break after its output unless we explicitly indicate it, therefore, the following sentences:

```
cout << "This is a sentence.";
cout << "This is another sentence.";
```

will be shown followed in screen:

This is a sentence.This is another sentence.

even if we have written them in two different calls to cout. So,in order to perform a line break on output we must explicitly order it by inserting a new-line character that in C++ can be written as \n:

```
cout << "First sentence.\n ";
cout << "Second sentence.\nThird sentence.";
```

produces the following output:

First sentence.
Second sentence.
Third sentence.

Additionally, to add a new-line, you may also use the endl manipulator. For example:

```
cout << "First sentence." << endl;
cout << "Second sentence." << endl;
```

would print out:

First sentence.
Second sentence.

The endl manipulator has a special behavior when it is used with buffered streams: they are flushed. But anyway cout is unbuffered by default .You may use either the \n escape character or the endl manipulator in order to specify a line jump to cout

Standard Input :(Cin)

cin>>no1;

(2) Input(cin)

it is an input statement and causes the program to wait for the user to type in number.
The operator of extraction (>>) or get from operator. it extracts(or takes) the values from the keyboard and assign it to the variable on right .
This must be followed by the variable that will store the data that is going to be read. For example:

int age;
cin >> age;

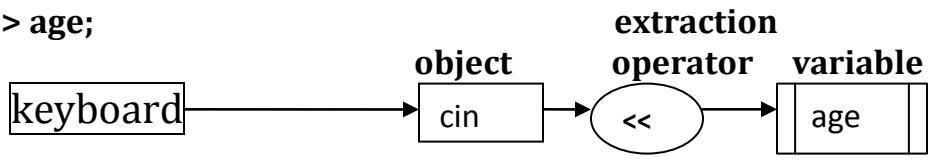


fig: **extraction operator**

Declares the variable **age** as an **int** and then waits for an input from cin (keyboard) in order to store it in this integer variable.cin can only process the input from the keyboard once the RETURN key has been pressed.Therefore, even if you request a single character cin will not process the input until the user presses RETURN once the character has been introduced.

You must always consider the type of the variable that you are using as a container with cin extraction. If you request an integer you will get an integer, if you request a character you will get a character & if you request a string of characters you will get a string of characters.

- **i/o example**

#include <iostream.h>
int main ()
{
int i;
cout << "Please enter an integer value: ";
cin >> i;
cout << "The value you entered is " << i;
cout << "and its double is " << i*2 <<
"\n";
return 0;
}
	Please enter an integer value: 702
	The value you entered is 702 and its double is 1404.

- **Cascading of input in c++:**

cin >> n1>> n2>>n3;
is equivalent to:
cin >> n1;
cin >> n2;
cin>>n3;
In both cases the user must give two data, one for variable a and another for variable b that may be separated by any valid blank separator: a space, a tab character or a newline.

A special mention of header file required for input and output in c++:
To enable the use of out ,<<, cin and ,>> one needs to include a header file named **iostream.h** in the first line of every program, by the statement:**#include<iostream.h>**
This header file contains declarations that are needed by count & cin objects and << & >> operators. Without these declarations, the compiler won't recognize cout & cin and will Think << and >> are being used incorrectly in the program.

A Special Mention Of Manipulators :

Manipulators are special type of function that change certain characteristics of the input and output .if you want to use manipulator function then iomanip.h header file must be included by the statement.

#include<iomanip.h>

Following are the standard manipulators.

Manipulators	functionality	Equivalent ios Function
setw()	Set the field width of a field.	width()
setprecision()	To set the precision of a floating point value to be displayed.	precision()
endl	To insert line feed character '\n'	-

1)endl :

The endl is output manipulator. it is used to generate carriage return. it has same effect as using the newline character "\n".
consider following statements:
cout<<"amit ";
cout<<"kumar";
output of two statements: will be:
amit kumar
but ,if we write:
cout<<"amit "<<endl;
cout<<"kumar";
output of two statements: will be:
amit
kumar
This is because after"amit" we have written endl so that the cursor gets transferred to new line ,therefore, "kumar" is written form new line

- **Note:** for the use of endl, there is no need of including header file **iomanip.h**

2)Setw():

- setw(w) - sets output or input width to w;
- need of use including header file <iomanip> to be included.
- setw manipulator is used to specify the number of character position a variable will consume on the output field. The setw manipulator does this job ,it is follows:

cout<<setw(5)<<sum<<endl;

the manipulator setw(5) specifies a field width 5 for printing the value of the variable 'sum'. This value is right-justified within the field as shown below:

		3	**4**	**5**

➢ **Program to illustrate the use of endl and setw.**

```
# include <iostream.h>
#include<conio.h>
#include<iomanip.h>              //for setw
int main( )
{
clrscr();
int  Basic = 950, Allowance = 95, Total = 1045;
cout << setw(10)<<"Basic"<<setw(10)<< Basic<<endl
<< setw(10)<<" Allowance"<<setw(10)<< Allowance <<endl
<< setw(10)<<" Total"<<setw(10)<< Total <<endl;
getch();
return 0;
}
```

```
******output*****
      Basic       950
  Allowance        95
      Total      1045
```

3)Setprecision():

By using setprecision we can control the number of digits of an output stream display of a floating point value.

Syntax of setprecision is:

Setprecision(n)

This above statement sets the precision for floating point number to n. The default precision is 6.Let us illustrate the use of setprecision with example:

Consider the following program block:

```
#include<iostream.h>
#include<iomanip.h>
Void main( )
{
float   a,b,c;
a=8;
b=3;
c=a/b;
cout<< setprecision(6)<<c<<endl;
cout<< setprecision(3)<<c<<endl;
cout<< setprecision(2)<<c<<endl;
cout<< setprecision(1)<<c<<endl;
}
```

```
**********output**********
2.666667
2.667
2.67
2.7
```

(2) Write short note on **Manipulators**.(DEC 2008)

EXTRA QUESTIONS

(1) Explain format control worlds printf()&scanf() function ?

(2) Explain : Getch(), putchar(),gets(), puts()

(3) Explain the following terms with example.

1) Conversion specification 2)Escape sequences.

(4) Explain any three backspace characters in C ++ with suitable examples.