Unit-1: Introduction to C++

Objectives: Exposure to the basics of the Object Oriented Mode, C++ Programming and I/O in C++

Topics to be covered: Part 1

Introduction to C++, Difference between C and C++, The Object Oriented Technology, Disadvantages of Conventional programming, Concepts of OOP, Advantages of OOP, Structure of C++ program, Header files and Libraries.

Introduction to C++

The C++ programming was developed by Bjarne StrousStrup in the year 1979 at AT & T Bell laboratories in Murray Hill, New Jersey (USA) as part of his PhD thesis. C++ runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. He added new features taking from the language called "**SIMULA 67**" to the c language to enhance its performance, and originally named it as " **C with classes**", but later he added some features from the language called "**ALGOL 68**", it was renamed by Rick Mascitti as "**C++**" in the year 1983. The name C++, is thought as the C post increment operator ++. C++ is a superset of c, hence any legal c program can be executed in the C++.

Bjarne Stroustrup

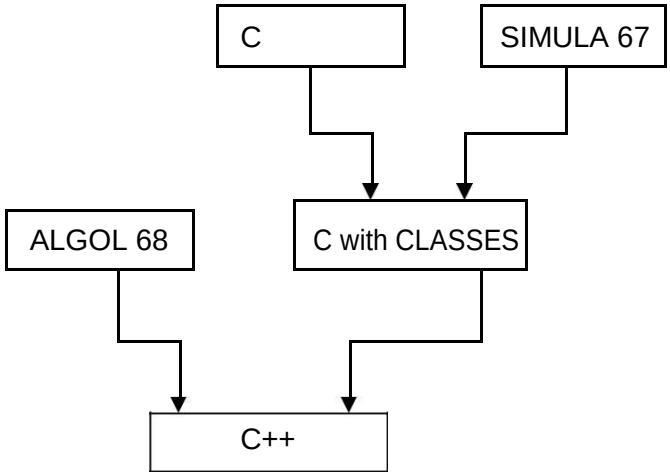


Fig 1. The Evolution of C++

Difference between C and C++

Sl NO	C Language	C++ Language
1	C is a Procedural Oriented Language	C++ is Object Oriented language
2	Data is not Protected in C.	Data is Secured in C++
3	It uses Top Down approach	It uses Bottom Up Approach
4	In C, same name cannot be given to two function	With function overloading it is possible to give same name to two or more functions
5	scanf() and printf() are used for reading and writing data respectively	cin and cout are used for reading and writing data respectively
6	C uses "stdio.h" header file for input & output operations	C++ uses, "iostream.h" for the same purpose.
7	C has no constructor & destructor	C++ provides constructors and destructors
8	Inline functions can be used as macros	Inline functions are supported by C++

The Object Oriented Technology

The nature is composed of various objects such as rocks, plants, trees, animals, and people etc. These **objects** can be further divided into two groups: living things and Non-living things. The rocks, chairs are the best example for Non-Living things which nothing but physical **objects**. The animals, trees, and people are the best example for Living things, which are again physical **objects**.

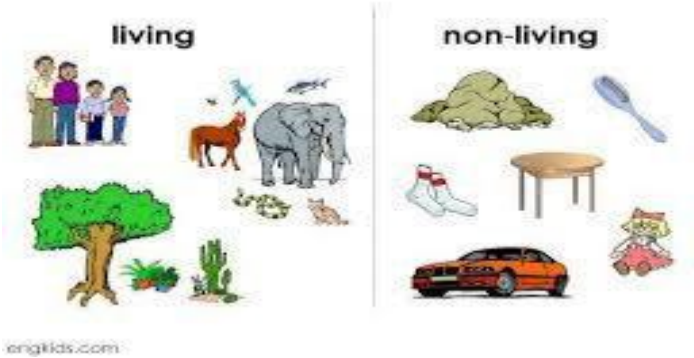


Fig 2. Living and Non-Living things

To understand the object oriented technology, let us take an analogy of the education institute which has two different working sections: Teaching and Non-Teaching. The institute will be having different departments such as CSE, ECE, MEC, EEE, CIVIL and LIBRARY. Each Department contains Teaching and Non-Teaching staff. Each department is considered as **Object** and is working for specific goals and objectives. Each department carries its own activities, and serves other departments when even needed. Departments can "**communicate"** with each other to carry out inter-departmental activities such as organizing the "**workshops**" and "**conferences**" though they are performing their own activities. The departments interact with other departments by sending **messages.** All these things can be represented in the C++. The departments can be represented as "**class**" from which different objects can be derived. The CSE Department is called an **instance** of the "department" which is called as **Object**. These objects contain their own data and methods to carry out their task and to communicate with other department whenever needed.

Fig 3. Relationship between departments

Disadvantages of Conventional programming

Traditional programming languages such as C, COBOL, FORTRAN are called Procedure Oriented (**POP**) language and also called as Conventional Languages.

The program written using these languages contains ***sequence of instructions*** that tell the compiler or interpreter to perform the task.

When the program size is large, it is a bit difficult to manage and debug.

To overcome this problem it is divided into ***smaller functions*** that carryout some specific task.

Each function can call other function during the execution.

As the functions are executing they may access same data, and may modify the data which in turn affects the entire task.

Most of the functions are allowed to access the global variables.

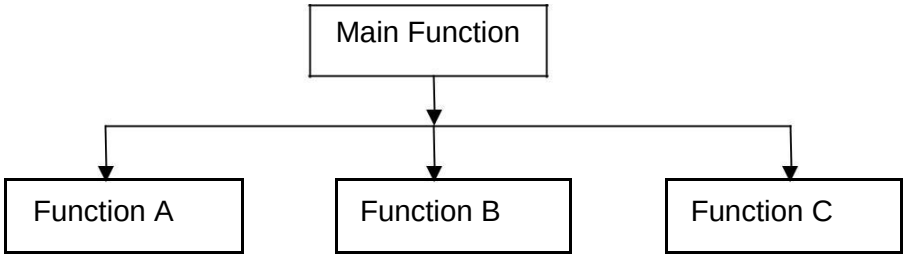


Fig 4. Flow of Functions in POP

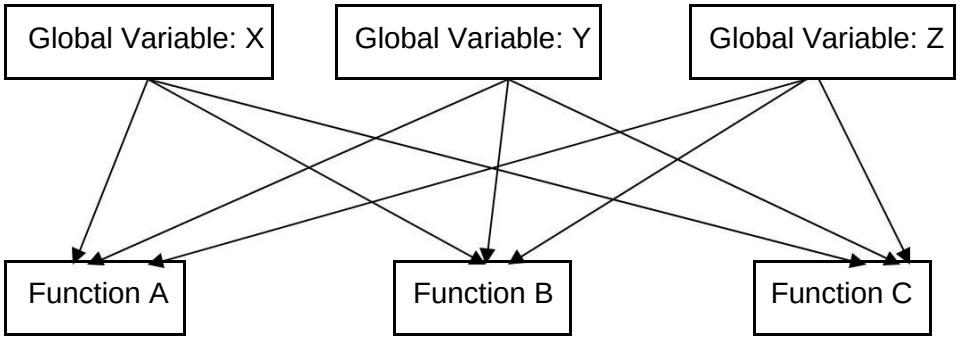


Fig 5. Sharing of Data by functions in POP.

Drawbacks of these languages:

Large program is divided into smaller functions. These functions can call one another. Hence **security** is not provided.

No importance is given to the data security.

Data passes globally from one function to other function. Most function access global data.

Programming Paradigms

The change in the development of the software had taken place because of the 4 reasons:

1. The complexity of the problem domain
2. The difficulty of managing development process
3. The flexibility possible through the software
4. The problems of characterizing the behavior of discrete systems

To address different problems of the above, 4 programming paradigms have been designed.

1. Monolithic Programming
2. Procedural Programming
3. Structural Programming
4. Object Oriented Programming

Monolithic Programming

These programs contain global data and sequential code.

Program flow control is achieved through Jump statements. There is no support of subroutine concept.

It is suitable for small and simple applications.

There is no support for data abstraction

Examples: Assembly language and BASIC

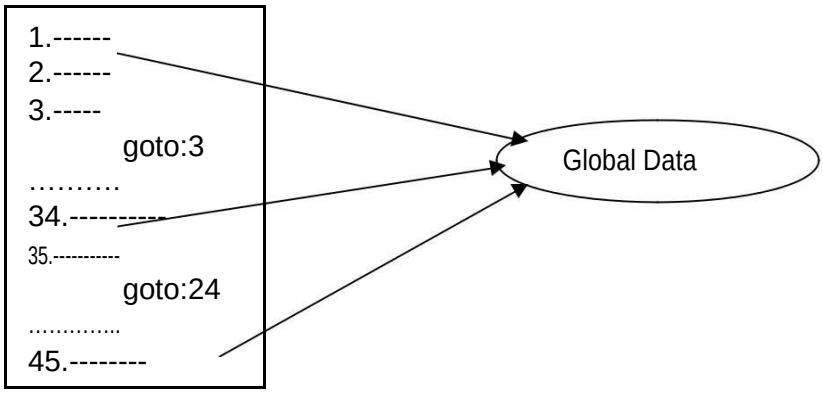


Fig 6. Monolithic Programming

Procedural Programming

The programs are organized in the form of subroutines and all the data items are global These programs are very easy to understand and modify.

These programs follow Top Down approach.

These programs focus on the functions apart from data. Data is globally accessed to functions.

Data is transferred by means of functions.

It is difficult to implement simultaneous processes/parallelism

Examples: FORTRAN and COBOL

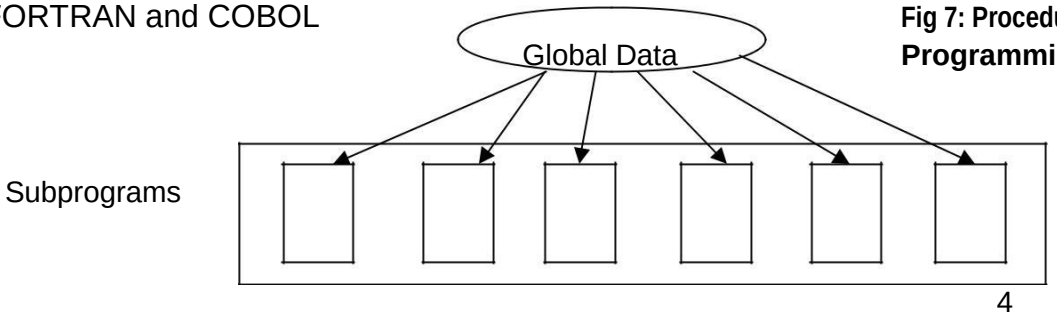


Fig 7: Procedure Programming

Structured Programming

Programs are divided into individual procedures that perform specific task. Each procedure is independent of other procedure

Procedures have their own local data and processing logic

The projects can be broken up into modules; each module consists of different set of functions.

Data access is controlled across the modules **Examples:** Pascal and C

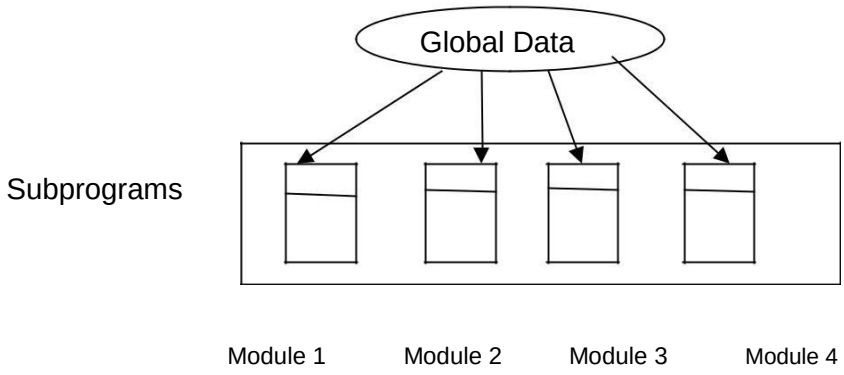


Fig 8: Procedure

Programming Object Oriented Programming

One of the main drawbacks of the procedure oriented programming is, it cannot provide data abstraction. It provides functional abstraction.

To address the increasing complexity with software for the data abstraction Object oriented programming came into existence.

Depending on the object features the languages are classified into two categories:

- Object –Based Programming (OBP), example: JavaScript
- Object-Oriented Programming(OOP), examples: Java and C++

The OBP languages support **Encapsulation**, and object **Identity,** but do not support Inheritance and polymorphism.

The OOP languages incorporate all the features of the OOP.

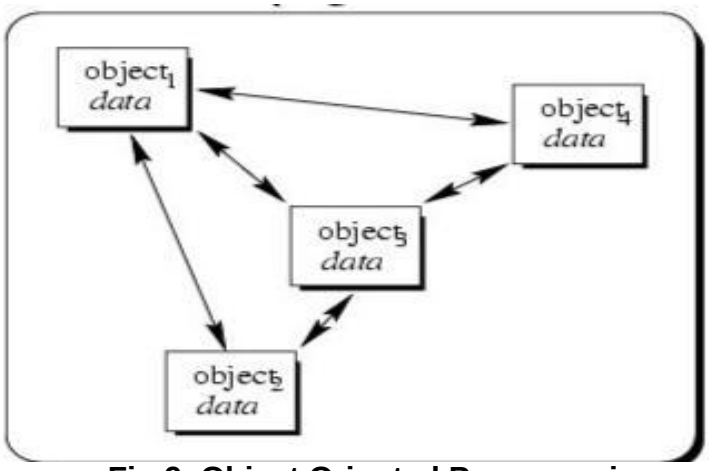


Fig 9. Object Oriented Programming

Key Concepts of Object Oriented Programming

There are several fundamental concepts in object oriented programming. They are shown as follow:

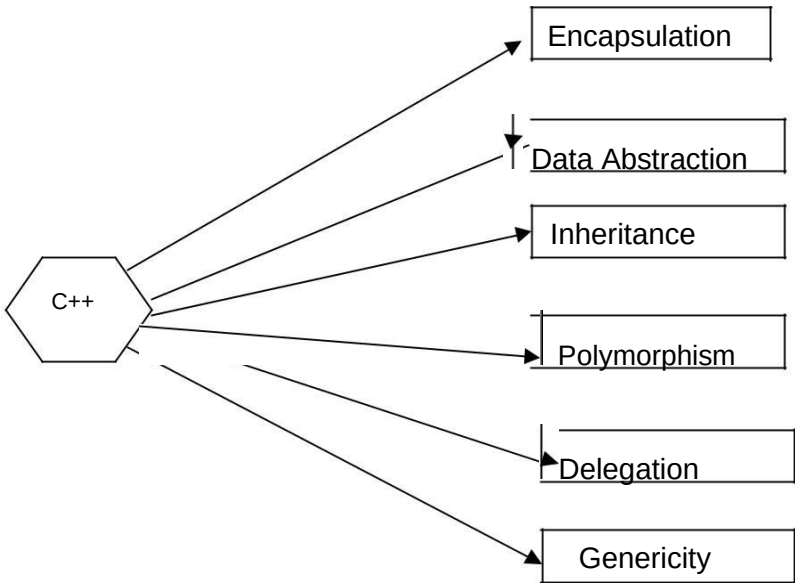


Fig 10. Key Concepts of OOP

Encapsulation:

***Definition*:** The process of binding Data and functions into a single unit is called Encapsulation.

C++ supports the feature of Encapsulation using classes.

The data in the class is not accessed by outside functions.

The functions that are defined along with the data within the same class are allowed to access the data.

Class defines the structure of data and member functions. The variables declared inside the class are called **instance variables**, and functions are called **member functions**.

Objects are created from the class. An **Object** is specimen of a class.

Class is logical structure, and object is physical structure.

Each member in the class may be public or private.

Only the member functions can access the private data of the class. However the public members can be accessed by the outside class.

Class declarations

A class definition starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. A Class is used to pack the data and member functions together. For example:

```
class class_name
{
    type variable1;
    type variable2;
    type variable3;

    type name_of_function(parameter_list);
    type name_of_function(parameter_list);
};
```

instance variables

member functions

Data abstraction

Definition: It the process of providing essential features without providing the background or implementation details.

Example:

It is not important for the user how TV is working internally, and different components are interconnected. The essential features required to the user are how to start, how to control volume, and change channels.

Inheritance

Definition: It the process by which an object of one class acquires the properties of another class. When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class, and the new class is referred to as the **derived** class. The actual power of the inheritance is that it permits the programmer to **reuse** the existing class to create new class.

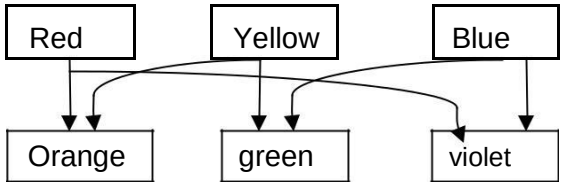


Fig 11: Inheritance

In Fig 11, Red, Yellow and Blue are main colors. From these colors orange is created from red and yellow. Green is created from the Yellow and Blue. The violet is created from the blue and red. So here a new color is inheriting the properties from two or more colors.

Polymorphism

Definition: polymorphism is a technique in which various forms of a single function can be defined and shared by different objects to perform an operation.

A function "**display()"** can display the shape of line, rectangle and square. Here the function "display()" has three different forms to perform the task of displaying the different things.

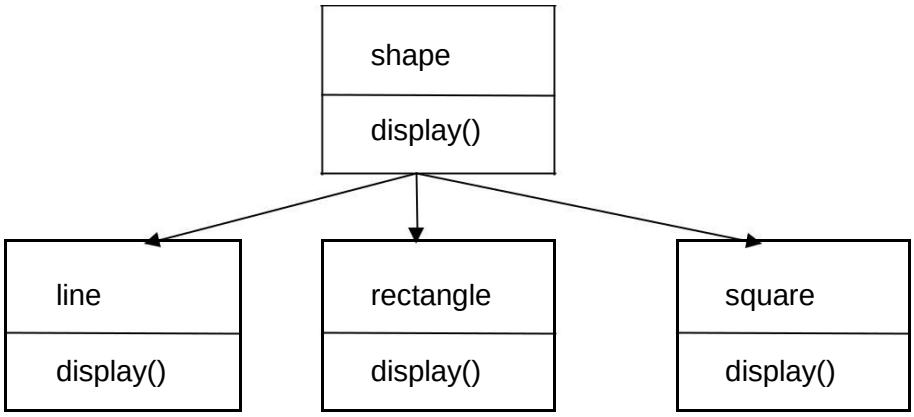


Fig 12. Polymorphism in OOP

Delegation

In OOP, two classes can be joined in two ways: (a) Inheritance (b) Delegation. In inheritance a new class can be derived from the existing class. The relationship between these two classes is called "***Kind of relationship".*** For example class Y has been derived from class X, then class Y is called kind of X.

Definition:The second type of relations ship is " has a relationship". When object of one class is used as data member in another class, such composition of objects is known as "***Delegation***"

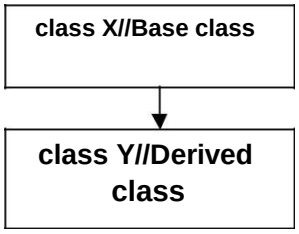


Fig 13 (a). Inheritance

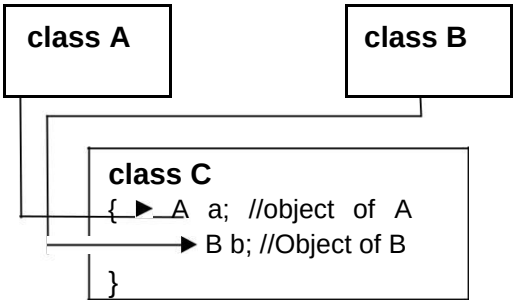


Fig 13 (b). Delegation

Genericity

We can write a program that contains more than one version depending on the data types of the arguments provided at run time. For example we can write a program that can perform addition on integers, and the same program can also perform addition on floating point numbers. This feature allows declaration of variables without specifying exact data types. The compiler identifies the data at run time. The "***template***" feature of C++ allows this genericity.

Advantages of OOP

OOP provides many advantages to the programmer and use. This solves many problems related to the software development, and provides improved quality and low cost.

1. These programs can be easily upgraded.
2. Using Inheritance, we can avoid writing the redundant code, and reuse already existing code.
3. It allows designing and developing safe programs using the data hiding.
4. Using the encapsulation feature of OOP, we can define new class with many functions and only few functions can be exposed to the user.
5. All the OOP language allows crating extended and reusable parts of the programs.
6. It changes the thinking of the programmer and results in rapid development of the software in a short time.
7. Objects communicate with each other and pass messages.

The structure of C++ Program

C++ program consists of objects, classes, variables, functions and other program statements. It contains 4 main parts. (1) The preprocessor directives (2) class declaration or definition (3) class function definition (4) The main() function.

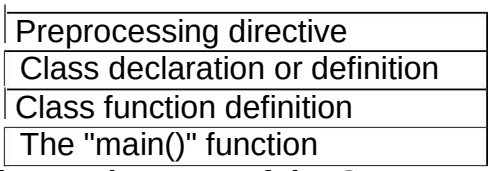


Fig 14. The parts of the C++ program

(1) Preprocessor Directives(Include header file section):

The preprocessor directive must be included at the beginning of program. It begins with symbol #(hash). It can be placed anywhere, but quite often it is placed at the beginning before the main() function. In traditional C, # must begin at the first column.

Example:

```
#include<iostream>
```

(2) Declaration of the class:

Declaration of the class is done in this section. The class declaration starts with keyword "**class**" and contains some name followed by a pair of curly ({ }) braces with instance variable and member functions. The class ends with semicolon (;);

Example:

```
#include<iostream>
using namespace std;
        class Box
        {
                public:
                double l;
                double w;
                double b;
                void volume()
                {//body of the function
                }
        };
```

(3) Class functions declaration:

This part contains the definition of the functions. The definition can be also written outside the function with class name and scope operator before the function name.

Example:

```
#include<iostream>
using namespace
std; class Box
        {
                public:
                double l;
                double w;
                double b;
                void volume(void); //function
        declaration }; //end of the class
void Box:: volume()  //definition outside the class
{
        // body of the function
}
```

(4) The main() function:

Like C, C++ also starts with main function. The execution of every program starts with "***main()"*** functions. The programming rules for C++ are same as C language.

Example:

```
#include<iostream>
using namespace
std; class Box
        {
                public:
                double l;
                double w;
                double b;
                void volume(void); //function
        declaration }; //end of the class
```

```
        void Box:: volume() //definition outside the
        class {// body of the function
                cout<< "The volume of Box is:"<<(l*w*b);
        }
int main()
{
        Box b;
        b.l=12.3;
        b.w=13.4;
        b.b=14.5;
        b.volume();
        return 0;
}
```

Simple C++ program:

hello.cpp

```
int main()
{
        //simple c++ program
        cout<<"\n Hello! Welcome to C++
        programming:"; return 0;
}
```

Compiling:

```
        $>g++ hello.cpp ENTER  //compiling
        $>./a.out              // seeing the output
```

Explanation of above program:

Program must be saved as ".cpp" as extension.

Every program executes from main() function.

A Comment can be placed in anywhere of the program. We can use single line comments (Begins with //), and multiple comment line start with /*Comment here…….*/

Header files are needed at the beginning of the program.

The statement **cout<<"\n Hello! Welcome to C++ program:";** displays the string enclosed between the quotations. The **"\n"** is called escape sequence used to print the string in the next line.

The << operator is called "**insertion**" operator.

Header files and Libraries

The library functions perform operations such as managing memory, reading and writing to disk files, input/output, mathematical operations, etc.

Name of the Header File	Function of the Header File	List of few functions supported by header files
alloc.h	Memory management functions	Calloc(), malloc(), free(),realloc() etc;
complex.h	Complex math functions	asin(),atan(),arg(),atanh() etc;
ctype.h	automatic type conversion	toupper(), tolower(), islower(), isupper() etc;
dos.h	Perform DOS functions	getdate(), gettime(), int86(), sleep() etc;
graphics.h	Graphic functions	arc(), bar(), circle(), getx() etc;
process.h	Process functions	exit(), abort(), system() etc;
stdio.h	Standard input/output functions	puts(),gets(),fopen(),fclose(), printf(),getchar(),putchar() etc;
stdlib.h	Standard library functions	ato(), atoi(),time(),abort() etc;
string.h	Manipulate various string functions	strcpy(),strcat(), strlen() etc;
time.h	Time manipulate functions	time(),stime(),difftime(),localtime() etc

Introduction to Input and Output in C++

Part –II
INPUTAND OUTPUTIN C++:
Introduction, Streams In C++ and Stream Classes, Pre-Defined Streams, Stream Classes, 'Formatted and Unformatted Data, Unformatted Console I/O Operations, Member Functions of Istream Class, Formatted Console I/O Operations, Bit Fields, Flags without Bit Field, Manipulators, User Defined Manipulators

Generally computer applications use large amount of Data to be read from input devices and writing them to the output devices. To facilitate reading and writing operations C++ support number of inbuilt functions. These functions are stored in the library. The library is a set of .OBJ file that are linked during the execution of the program. This library is also called "***iostream***" library.

What is a stream? : stream is an intermediator between I/O devices and user. It is a flow of data, measured in bytes, in sequence.
Input Stream: If the data is received from the input devices, it is called "source stream" and also called input stream. Examples for the input stream are Keyboard and disk.
Output Stream: If the Data is written or passed to the output devices, then that stream is called "destination stream" or "output stream". Examples for the output device are Monitor and disk.

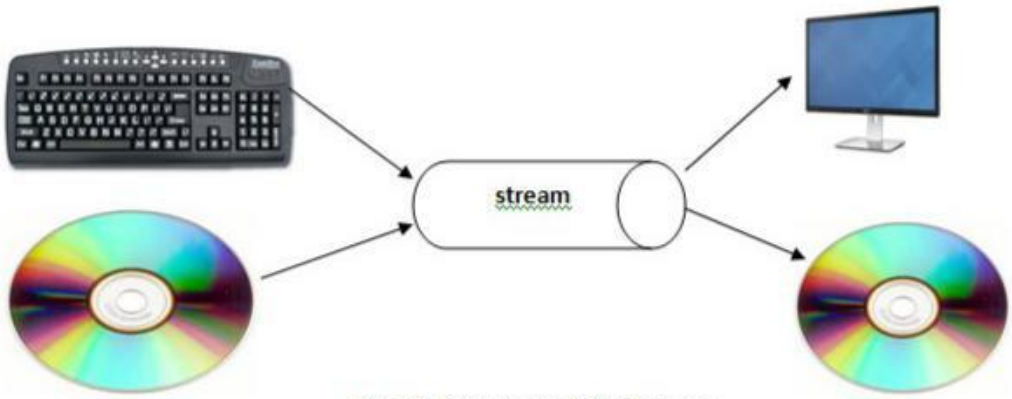


Fig 15. Streams and I/O Devices

Predefined Streams

C++ has number of predefined streams. These streams are also called as Standard I/O Objects. These streams are automatically activated when the program execution starts.

stream	Description
Cin	standard input stream, usually the keyboard
Cout	standard output stream, usually the monitor
Cerr	standard error (output) stream, usually the monitor
Clog	standard logging (output) stream, usually the monitor

Write an Example program to display a message using predefined objects

```
#include<iostream>
int main() {
        cout<<"\\Hello displayed on monitor";
        cerr<<"Errors also printed to the monitor":
        clog<<"Displayed to the monitor";
        return 0; }
```

Stream Classes

C++ streams are based on classes and object theory. C++ has number of classes that work together with console and file operations. These classes are known as stream classes. All these classes are declared inside the header file "**iostream.h**". The ***istream*** and ***ostream*** are the derived classes of the "***ios"*** base class. The ***ios*** contains member variable ***streambuf***. The ***iostream*** is derived from ***istream*** and ***ostream*** classes using the multiple inheritance.

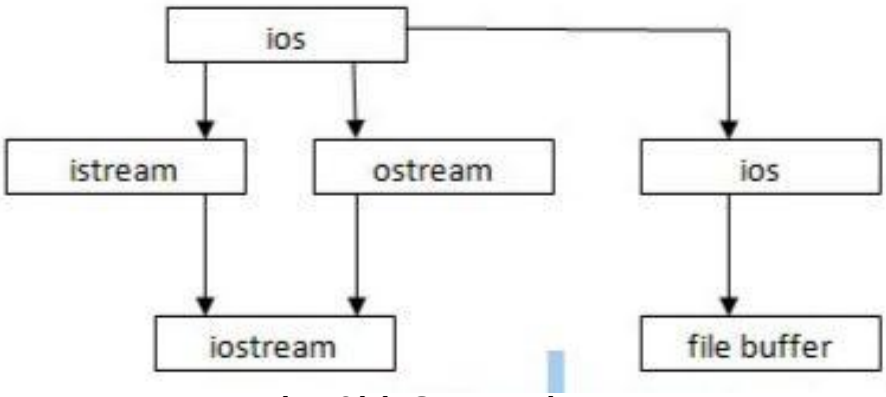


Fig 16(a). Stream classe

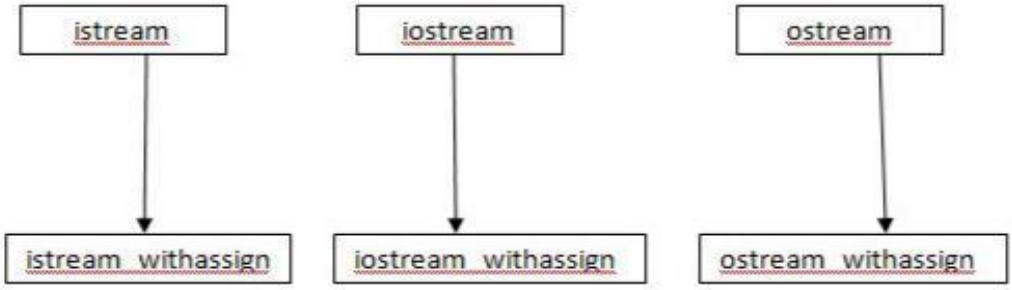


Fig 16(b). Hierarchy of stream classes

Functions and contents of stream classes

Class	Function/Contents
Ios	(1) it is an input and output stream class (2) It is used to implement a buffer (3) It maintains information on the state of streambuf
istream	(1) it provides the formatted input

	(2) used to handle formatted as well as unformatted conversion characters from streambuf (3) properties of ios are inheritated to istream class (4) it provides the functions such as peek(), tellg(), seek(),getline() (5) it overloads the '>>' operator
ostream	(1)It used for general purpose output. (2)used to declare output functions such as tellp(), put(),write(), seekp() etc. (3)It is the parent of all output streams (4)it overloads '<<' operator
iostream	It is used to handle both input and output streams
istream_withassign	(1). It is derived from istream (2) It is used for cin input
iostream_withassign	It is a bidirectional stream

Formatted and Unformatted Data

Formatting means representation of data with different settings according to the user requirements. Various settings that can be done are number format, field width, decimal points, etc. The data accepted or printed with default setting of I/O functions of the language is known as "**Unformatted Data**". For example, when cin is executed is asks for the number and if the number is entered, then the cout will display it on the screen. By default the I/O functions represent the data in decimal format.

If the user wants to accept the data in the HEXADECIMAL format, manipulator with I/O functions should be used. The obtained data with these formats is called "**Formatted Data**". For example, hex is the manipulator.

```
cout<<hex<<15;
```

The above statement is converted to decimal to hexadecimal format.

Unformatted Console I/O Operations

Input and Output Streams

The input stream uses "cin" object to read the data from the keyboard.

The output stream uses "cout" object to write data to the screen.

The "cin" and "cout" are called predefined streams for input and output data.

The operator "<<" is used after the cout. It is called "**insertion**" operator.

The operator ">>" is used before the variable name. It is called "**extraction**" operator. These two means "<<" and ">>" are called overloading operators.

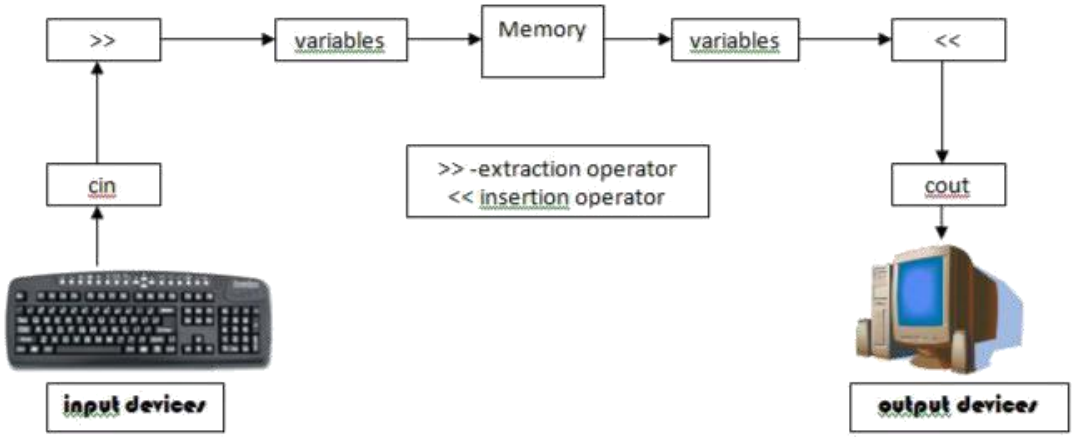


Fig 17. Working of cin and cout statements

Input stream

The input stream reads the data using the cin object. The cin statement uses >> (extraction operator) before the variable name. The Syntax is as follows:

```
cin>>variable;
```

Example:

```
int v1;
float v2;
cin >> v1>>v2;
```

Where v1 and v2 are variables. If the user enters data 4 and 5, then 4 is stored in v1 and 5 is stored in v2. More than one variable can be used in cin statement then it is called "***cascading input operations***"

Output Streams

The output streams manage the output of the streams. The cout object is used as output stream. It uses the << operator before the variable or string to display it on the screen. The Syntax is as follows:

```
cout<< variable;
```

Example:

```
cout<<v1<<v2;
or
cout<<"Hello";
```

Write a program to accept the string from the keyboard and display it on the screen. Use cin and cout statements.

exp.cpp

```
#include<iostream.h>
using namespace std;
int main()
{
    char name[20];
```

```
        cout<<"Enter your name:";
        cin>>name;
        cout<<name;
        return 0;
}
```

Write a program to read int, float, char and string using cout to display them on the screen.

```
#include<iostream.h>
using namespace std;
int main()
{
        char c;
        int a;
        float b;
        char city[10];
        cout<<"Enter Character:\n";
        cin>>c;
        cout<<"Enter integer a\n";
        cin>>a;
        cout<<"Enter String :";
        cin>>city;
        cout<<" Char is:\n"<<c<<"The integer is :\n"<<a<<"The float is \n:"<<b<<"\n the city
        name is :"<<city;
        return 0;
}
```

Write a C++ program to carry out all the arithmetic operations on integers.

```
#include<iostream.h>
using namespace std;
int main()
{
        int a,b,c;
        cout<<"Enter integer a and b\n";
        cin>>a>>b;
        c=a+b;
        cout<<"The sum is :"<<c;
        c=a*b;
        cout<<"The Multiplications is :"<<c;
        c=a/b;
        cout<<"The Division is :"<<c;
        return 0;
}
```

Type casting with cout statement

The type casting refers to the conversion of one basic type into another by applying external use data type keywords. C++ provides the type casting operator in the following format:

cout<<(type)variable; // here the type refers to the destination type and variable refers to the data of one basic type.

Example:

```
float f=2.34;
cout<<(int)<<f;
```

Output: displays 2

Write a c++ program to use different formats of the type casting and display the converted value.

```
#include<iostream.h>
using namespace std;
int main()
{
        int a=66;
        float f=2.34;
        double d=85.45;
        char c='A';
        cout<<"\n int in char format:"<<(char)a;
        cout<<"\n float in in t format:"<<(int)f;
        cout<<"\n double in char format:"<<(char)d;
        cout<<"\n char in int format:"<<(int)c;
        return 0;
}
```

Write a c++ program to display A to Z alphabets using the ASCII values

```
#include<iostream.h>
using namespace std;
int main()
{
        int j;
        cout<<"\n the alphabets from A to Z are :\n";
        for (j=65;j<91;j++)
        cout<<(char)j<<"\t";
        return 0;
}
```

Note 1: The & operator is used to print the address of the operator. For example, int x=12; the variable is stored at some address of the memory. That address can be printed using the & operator before the variable name.

Example:

```
cout<<"The address of the variable is:"<<&x;
```

output :0x887ffff4, it is the hexadecimal format of the address.

Note 2: The & and * operators are used with string to display string.

Example:

```
char *name=" Ashok Kamthen";
cout<<name<<"\n";
cout<<&name[0]<<"\n";
```

Output statements will display the Same output. If we write only the &name, then it displays the address, if we use & along with the base address of the variable then it displays the string present at that address.

Member functions of the istream class

The istream class contains the following functions using cin object.

Function name	Description
cin.get()	Used to read a character from the keyboard
cin.peek()	It returns the succeeding character without extraction
cin.ignore(number, character)	Used to ignore the maximum number of characters
cin.putback()	Replaces the given character into the input stream
cin.gcount()	Return the number of characters extracted from the stream to a variable
cin.getline()	Contains the two arguments: variable and size , used to get the text from the variable with specified number of character in 2nd argument

Example programs:

Write a c++ program to demonstrate the use of get(), peek() and ignore() functions.

```
#include<iostream.h>
using namespace std;
int main()
{
        char c;
        cout<<"Enter the chars : And PRESS F6 to end";
        while(cin.get( c ))
        {
        cout<<c;
                while(cin.peek()=='#')
                {
                        cin.ignore(1,'#');
                }
        }
        return 0;
}
```

Output:

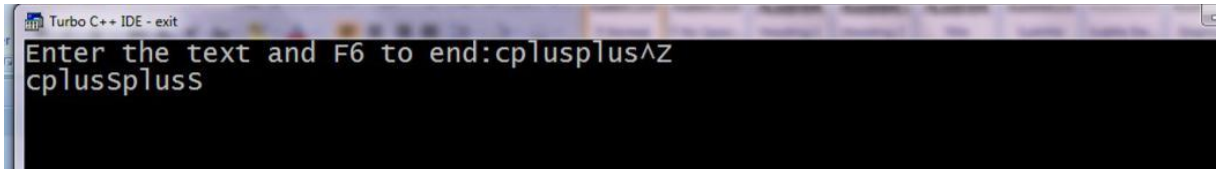
```
C:\TC\BIN\KSR>exit
Enter chars and F6 to endABCDE###FGH^Z
ABCDEFGH
```

Write a c++ program to demonstrate the use of putback() function

```
#include<iostream.h>
#include<conio.h>
int main()
{char c;
      clrscr();
      cout<<"Enter the text and F6 to end:";
      while(cin.get(c))
      {
            if(c=='s')
            cin.putback('S');
            cout.put(c);
      }
      getch();
      return 0;

}
```

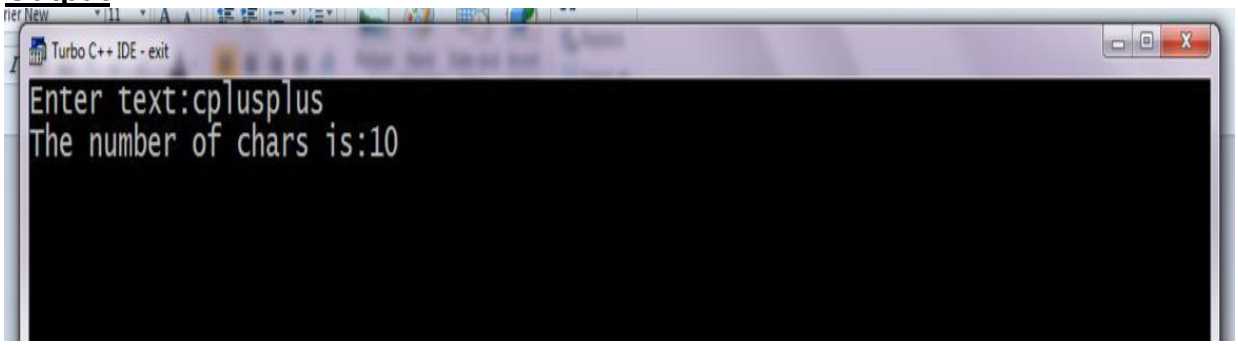
Output:



Write a c++ program to demonstrate the use of gcount() function

```
#include<iostream.h>
#include<conio.h>
int main()
{
      char name[20];
      int len;
      clrscr();
      cout<<"Enter text:";
      cin.getline(name,20);
      len= cin.gcount();
      cout<<"The number of chars
      is:"<<len; return 0;
}
```

Output



Formatted Console I/O Operations

C++ provides various formatted console I/O functions for formatting the output. There are three types of Formatted I/O Functions:

1. IOS class functions flags
2. Manipulators
3. User defined output functions

IOS Functions

Sl No	Function	Description
1	width()	Used to set the required field width
2	precision()	Used to set the number of decimal points to a float value
3	fill()	Used to set the character in the blank spaces
4	setf()	Used to set various formatting flags
5	unsetf()	Used to remove the flag setting

Note: all these functions are used along with "cout" object. For example, cout.width().

The **cout::width()** function can be declared in two ways:

i) **int width()**- used to return the current field width.
ii) **int width(int)**- used to set the width with given integer.

The **cout::precision()** can be declared in two ways:

i) **int precision()** –used to return the current precision
ii) **int precision(int)** –used to set the precision

i) **char fill()**-used to return the current fill character
ii) **char fill(char)** –used to set the filling character

The **cout::setf()** function has the following form:

i) cout.setf(V1,V2) –where v1 is flag, and v2 is bit field
ii) cout.unsetf() –used to clear the formatting flags

Flag and Bit Field are as follow:

Format	Flag (V1)	Bit Field (V2)
Left Justifications	Ios::left	Ios::adjustfield
Right Justification	Ios::right	Ios:: adjustfield
Padding after sign and base	Ios::internal	Ios:: adjustfield
Scientific notation	Ios::scientific	Ios::floatfield
Fixed point notation	Ios::fixed	Ios::floatfield
Decimal base	Ios::dec	Ios::basefield
Octal base	Ios::oct	Ios::basefield
Hexadecimal base	Ios::hex	Ios::basefield

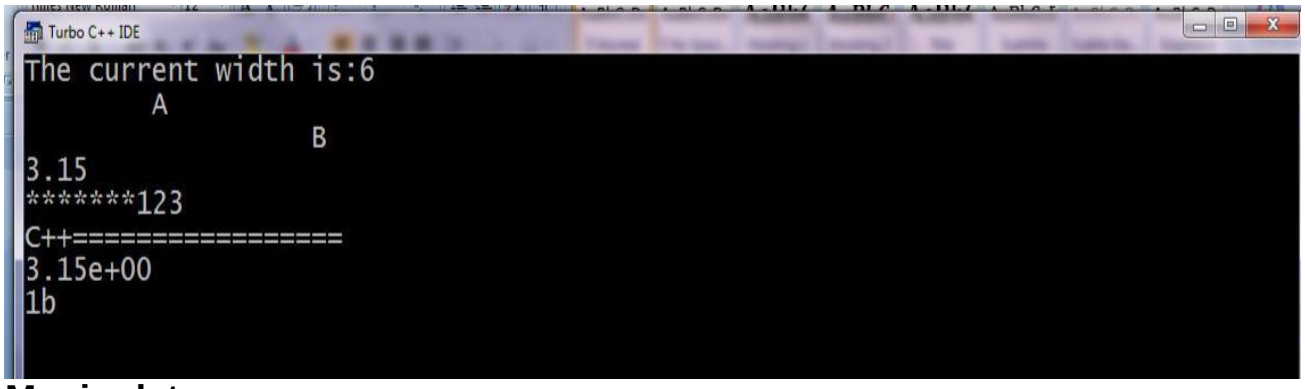
Example Program:

fop1.cpp

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

```
{
      clrscr();
      cout.width(6);
      int x=cout.width(); //returns the current width
      cout<<"The current width is:"<<x <<"\n";
      cout.width(10);
      cout<<"A\n";
      cout.width(20); //sets the width 20
      cout<<"B\n";
      cout.precision(2); //keeps only 2 decimal points
      cout<<3.1472<<"\n";
      cout.fill('*'); //used to set the blank spaces with *
      cout.width(10);
      cout<<123<<"\n";
      cout.fill('=');
      cout.width(20);
      cout.setf(ios::left,ios::adjustfield); //sets the justification left
      cout<<"C++";
      cout.setf(ios::scientific,ios::floatfield); //sets the float to
      //scientific notation
      cout<<"\n";
      cout<<3.14734<<"\n";
      cout.setf(ios::hex,ios::basefield);//sets input number to
hexadecimal
      cout<<27;
      getch();
}
```

Output:



Manipulators

The output format can be controlled using manipulators. The header file "**iomanip.h**" has set of functions. The effect of these manipulators is similar to ios class member functions. Every ios member function has two formats. The first format is used for setting and the second format is used to understand the previous setting. But the manipulator does not return previous setting. The manipulator can be used along with cout() statement as follows: cout<<m1<<m2<<v1;
where m1 and m2 are manipulators and v1 is the variable.

Manipulator	Function
setw(int n)	The field width is fixed to n
setbase	Sets the base of the number system.
setprecision(int p)	Sets the precision point to p
setfill(char f)	Sets the filling character to f
setiosflags(long l)	Format flag is set to l
sesetiosflags(long l)	Removes the flags indicated by l
endl	Splits the new line
skipws	Omit whitespace on input
noskipws	Does not omit white space on input
ends	Adds a null character to close output string
flush	Flushes the buffer stream
lock	Locks the file associated with file handle
ws	Omit the leading white space
hex, oct, dec	Displays the numbers in hexadecimal, octal and decilmal format

Write a program to display the formatted output using the manipulator

man1.cpp

```
#include<iostream.h>
#include<iomanip.h>
#include<conio.h>
void main()
{
      clrscr();
      //setting the width and precision at a time
      cout<<setw(10)<<setprecision(2)<<3.14567;
      cout<<endl; //splits the new line
      cout<<setiosflags(ios::hex);
      cout<<"The hexadecimal value of the 23 is:"<<23<<endl;
      //converting hex into oct
      cout<<"The octal value of 23 is:"<<oct<<23<<endl;
      //reading a number into hexa decimal format int
      x;
      cout<<"Enter a number:"<<endl;
      cin>>hex>>x;
      cout<<x<<endl;
      getch();
}
```

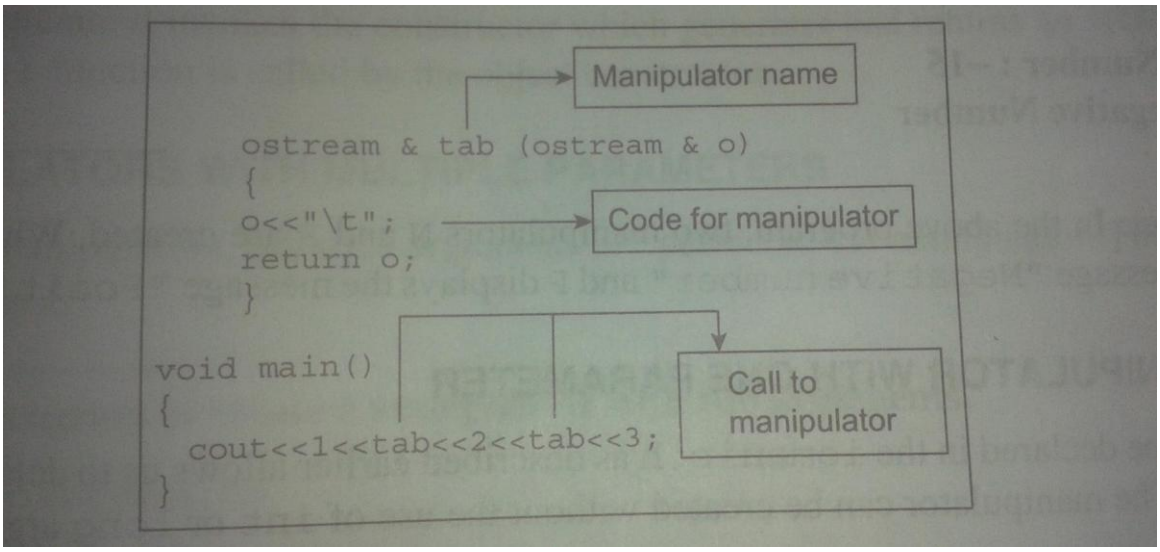
User defined manipulator

The programmer can also define his or her own manipulators according to the requirements. The syntax for creating the manipulators is as follows:

```
ostream & manip_name(ostream & o)
{
      statement 1;
      statement 2;
      return o;
```

}

where ***mani_name*** is the name of the manipulator used by the user.



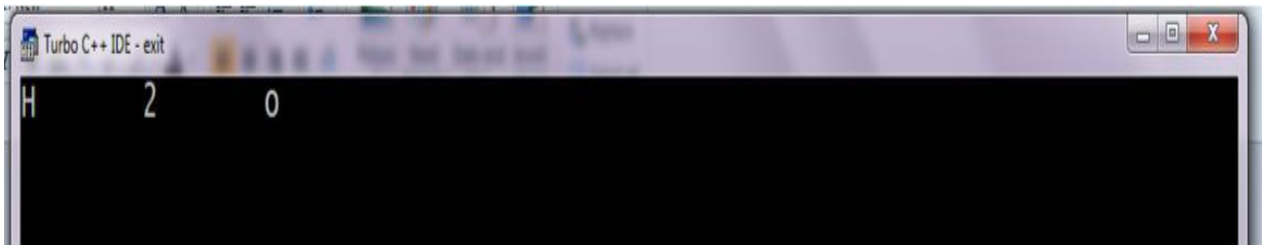
Write a C++ program to create a manipulator equivalent to the "\t" ?

tabtest.cpp

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
ostream & tab(ostream &o)
{
     //here tab is the name of the manipulator
     //when it is used it returns the o which contains the tab
     o<<"\t";
     return o;
}

void main()
{
     clrscr();
     cout<<1<<tab<<2<<tab<<3;
     getch();
}
```

Output:



Flags without Bit fields

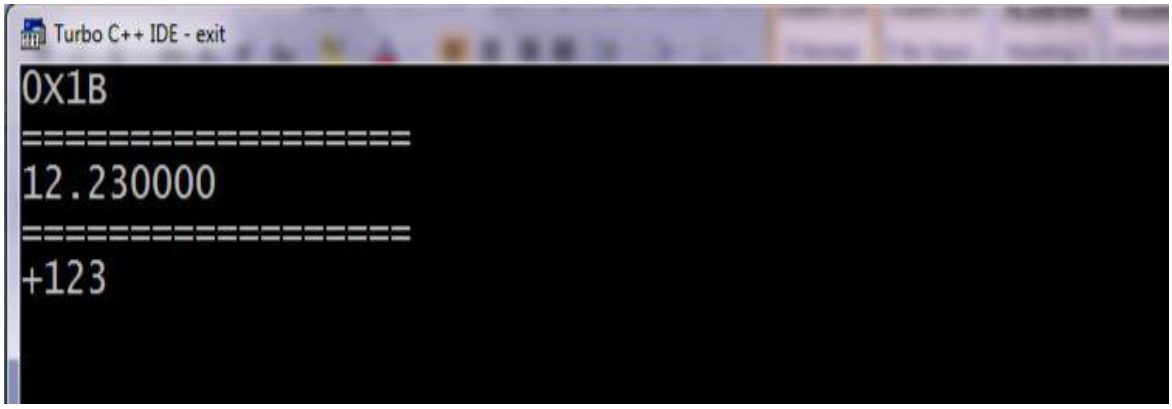
Sl No	Flag	Working
1.	ios::showbase	Uses base indicator on the output
2.	ios::showpos	Display + preceding positive number
3.	ios::showpoint	Shows trailing decimal point and zeros
4.	ios::uppercase	Uses capital case for hex output
5.	ios::skipws	Skips the white space
6.	ios::unitbuf	Flushes all the stream buffers
7.	ios::stdio	Flushes stdout and stdin later insertion
8.	ios::boolalpha	Converts the boolean value into text(True or False)

Note: all the flags are used with setf() function.

Write a C++ program that uses various Flags of ios ?

```
#include<iostream.h>
#include<conio.h>
void main()
{
      clrscr();
      //cout.setf(ios::hex,ios::basefield);
      cout.setf(ios::uppercase);
      //converts the hex value into uppercase
      cout.setf(ios::showbase);
      //showbase adds some base before the number such as 0x1B
      cout.setf(ios::hex,ios::basefield);
      cout<<27;
      cout<<"\n==================\n";
      cout.setf(ios::showpoint);
      //adds the zeros at the end of the number
      cout<<12.23;
      cout<<"\n==================\n";
      cout.setf(ios::showpos);
      //displays the sign operator before the number
      cout.setf(ios::dec,ios::basefield);
      cout<<123;
      getch();
}
```

Output:



*******end of 1st unit******