-

UNIT -5

File Handling-

• In File Handling, data is permanently stored in a secondary storage device (Hard disk). Open, close, read, write keys are used. • File Handling

Earlier, both cin and cout have been using iostream.h for these objects. When iostream does not use this header file then cout has no value. File handling in C++ is a topic which has been given separate header file, its name is fstream. cin of istream class and of ostream class stay | In the same way, File Handling has also been done.

File Handling Of you use why do ?

In the program, cin and cout store the memory for a short time, that is, when the programmer closes the program, then all the data of the program is destroyed. Programmer uses some variables, arrays, structures, unions to store data in the program, but this data is not permanently stored. File Handling is used to store it permanently. Files created during file handling, whether they are of different types (.txt, .doc etc.), are portable. It is used in other computers as well. Ultimately this header file fstream has to be used for File Handling. fstream There are no classes in this header file.

1. ifstream
2. ofstream
3. fstream

ifstream: ifstream is used to read the file. **ofstream :** ifstream is used to write to the file. **fstream:** fstream is used to read and write the file. There are two classes in the fstream class, ifstream and ofstream. If the file is to be read, then ifstream is required and if data is to be written on the file, ofstream is required. But it should be understood in which mode to open the file.

Seven modes of file have been created.

Modes	Type of Stream	Description
ios::in	ifstream	The file is opened in read mode.
ios::out	ofstream	The file is opened in write mode.
ios::binary	ifstream ofstream	Binary file is done open in no extension is given to the file, then the file is opened in default .txt mode.
ios::ate	ifstream ofstream	Data is appended at the end-of-file and the entire data in the file is controlled with the existing data.
ios::app	ofstream	Data is appended at the end-of-file. It is used to add meaning data.
ios::trunk	ofstream	Data is reduced from the file.
ios::nocreate	ofstream	If the file does not already exist, the open function fails and the file is not created.
ios::noreplace	ofstream	If the file already exists, it open function fails and new file create doesn't

If stream class 'ifstream' is used then file open in default ios::in(read) mode if stream class 'ofstream' is used then file open in default ios::out(write) mode.

-

Opening File

Syntax For Opening File

```
stream-class stream-object;

stream-object.open("file_name");
```

for eg.

```
downstream fault;                    // create stream

fout.open("sample.txt"); // default mode is ios::out
```

Let's open Multiple modes can also be used with the Bitwise OR Operator(|) go to File.

for eg.

```
fstream error;

fout.open("sample.txt", ios::in|ios::out);
```

There is a

When File ***closing*** file, then it has to be closed as well. When the file is closed, the file that allocates open memory it is freed.

Syntax For Closing File

```
stream-object.close();
```

for eg.

```
fout.close();
```

Check File Open or not!

open fail() member functions to check whether the file is open or not. If so, it has to check whether the file is opened or not. When file , its is_open() or functions return boolean values.

Source Code :

```
#include <iostream.h>

#include <fstream.h>

using namespace std;

int main(){

ifstream error;

fout.open("sample.txt");

if(fout.is_open()){                    // or if(!fout.fail()){

        cout<<"File is Opened."<<endl;

}else{

        cout<<"Unable to open file."<<endl;

}

fout.close();
```

```
return 0;

}
```

Output:

```
File is Opened.
```

Read Data from a File The

given program is used to read a File. There is a variable of

character data type, in which it is done to store the data of the file. Ifstream's stream-object name is fouted and later file is used with no argument of mode in function then ios::in is already there (default).

open What have you done? But

open member is because, if istream class

Later whether the file is open or not is checked by the condition. If the file is open, then the program continues and if the file is not open, then the program will be terminated by exit() (stdlib.h) function. Later, while loop is used to check whether the file has reached end-of-file or not.

sample.txt

```
Hello World!
```

Source Code :

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

-


```
#include <fstream.h>

#include <stdlib.h>

using namespace std;



int main(){



char ch[30];



ifstream error;

fout.open("sample.txt") ;



if(!fout){

        cout<<"Unable to opened file!";

        exit(1);                    //Program terminate if file is not opened.

}

        cout<<"File Contents : ";



while(!fout.eof()){
```

```
        error>>ch;

        cout<<ch<<" ";

}

fout.close();


return 0;

}
```

Output :

```
File Contents : Hello World!
```

Write data on a File

```
#include <iostream.h>

#include <fstream.h>

#include <stdlib.h>

using namespace std;


int main(){
```

```
downstream fault;

fout.open("sample.txt") ;



if(!fout){

        cout<<"Unable to opened file!";

        exit(1);                    //Program terminate if file is not opened.

}



        fout<<"Hello Friends!";



fout.close();



return 0;

}
```

Output:
sample.txt

```
Hello Friends!
```

-

SIR) ***get() Member Function***

Syntax for get() for File Handling

```
stream-object_of_ifstream_class.get(char ch);                    or

char ch = stream-object_of_ifstream_class.get();
```

Syntax for get() for Console

```
cin.get(char ch); or

char ch = cin.get();
```

Single character is inputted from the get() member function. The

get() function is defined in the istream class. This means to use this function iostream this header file has to be included. get() are unformatted stream IO functions. get() function is used to read the data. **cin>>ch and cin.get(ch); What is mefic?** cin >> ch; It does not accept newline and whitespace. cin.get; It takes in newlines and whitespaces.

get() Example

```
#include <iostream.h>

#include <fstream.h>

using namespace std;
```

-


```
int main(){



char ch;

        cout<<"Enter one character : ";

        cin.get(ch);

        cout<<"Entered character : ";

        cout<<ch;

return 0;

}
```

Output:

```
Enter one character : o

Entered character : o
```

get() with file

Source Code :

sample.txt

```
Hello Friends!
```

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

```
#include <fstream.h>

#include <stdlib.h>

using namespace std;



int main(){



char ch;



ifstream fin;

fin.open("sample.txt") ;



if(!fin){

        cout<<"Unable to opened file!";

        exit(1);                    //Program terminate if file is not opened.

}

            cout<<"File Contents : ";

while(!fin.eof()){



        fin.get(ch);
```

-


```
        cout<<ch;

}

fin.close();



return 0;

}
```

Output:

```
File Contents : Hello Friends!
```

put() Member Function

Syntax for put() for File Handling

```
stream-object_of_ofstream_class.put(char ch);
```

Syntax for put() for Console

```
cout.put(char ch);
```

A single character is outputted from the put() member function. put()

function is defined in ostream class. This means to use this function iostream this header file has to be included. put() are unformatted stream IO functions.

(KASHYAP SIR) put() function is used to write data.

put() Example

Source Code :

```
#include <iostream.h>

#include <fstream.h>

using namespace std;



int main(){



char ch='H', c = 65;



        cout<<"Value of ch : ";

        cout.put(ch)<<endl;



        cout<<"Value of c : ";

        cout.put(c);                    // ASCII value of a is 65



return 0;

}
```

-

Output:

```
Value of ch : H

Value of c : A
```

put() with file

Source Code :

```
#include <iostream.h>

#include <fstream.h>

#include <stdlib.h>

using namespace std;


int main(){



char ch='H';



downstream fault;

fout.open("sample.txt") ;
```

-


```
if(!fout){

        cout<<"Unable to opened file!";

        exit(1);                    //Program terminate if file is not opened.

}

        fout.put(ch);



fout.close();



return 0;

}
```

Output: *sample.txt*

```
H
```

write() Member Function

Syntax for write() for File Handling

```
stream-object_of_ofstream_class.write(line_or_string, stream_size);
```

-

Syntax for write() for Console

```
cout.write(line_or_string, stream_size);
```

Concatenate write() function with dot operator

```
cout.write(line_or_string, stream_size).write(line_or_string,
stream_size);
```

write() function is the output function, which displays the line. write() function is a member function of the iostream class. The write() function has two arguments.

1. **line_or_string:** This is an array of charactetype. 2.
stream_size : How many characters to take from the line is written here. If stream_size is exceeded then whitespaces are also done.

write() Example

Source Code :

```
#include <iostream.h>

#include <fstream.h>

#include <stdlib.h>

using namespace std;



int main(){
```

-


```
char ch[12] = "Hello World!";


        cout.write(ch, 12);


return 0;

}
```

Output: *sample.txt*

```
Hello World!
```

write() with file

Source Code :

```
#include <iostream.h>

#include <fstream.h>

#include <stdlib.h>

using namespace std;


int main(){
```

-


```
char ch[12] = "Hello World!";



downstream error;

fout.open("sample.txt") ;



if(!fout){

        cout<<"Unable to opened file!";

        exit(1);                    //Program terminate if file is not opened.

}

        fout.write(ch, 12);



fout.close();



return 0;

}
```

Output: *sample.txt*

```
Hello World!
```

-

getline() Member Function

Syntax for getline() for File Handling

```
stream-object_of_ifstream_class.getline(line_or_string,
stream_size);
```

Syntax for getline() for Console

```
cin.getline(line_or_string, stream_size);
```

getline() function This is the input function, which inputs the line from the keyboard.

getline() function is a member function of istream class. There is NULL character(\0) at the end of the string of getline() function.

getline() Example

sample.txt

```
Hello World!
```

Source Code :

```
#include <iostream.h>

#include <fstream.h>

#include <stdlib.h>
```

```
using namespace std;



int main(){



char ch[12];

        cout<<"Enter String : ";

        cin.getline(ch, 12);

        cout<<"Entered String : "<<ch;



return 0;

}
```

Output:

```
Enter String : Hello World!

Entered String : Hello World
```

getline() with file

sample.txt

```
Hello World!
```

-

Source Code :

```
#include <iostream.h>

#include <fstream.h>

#include <stdlib.h>

using namespace std;



int main(){



char ch[12];



ifstream fin;

fin.open("sample.txt") ;



if(!fin){

        cout<<"Unable to opened file!";

        exit(1);                    //Program terminate if file is not opened.

}

        fin.getline(ch, 12);

        cout<<ch;
```

-


```
fin.close();


return 0;

}
```

Output:

```
Hello World
```

Exception Handling

Runtime Errors

Exceptions are a type of error that occurs at run time or at the time of execution. Exception handling is used program when a specific code cannot be handled or an abnormal condition occurs.

-

SIR) These Exceptions may be due to divided by zero, out of bound array, out of memory or other reasons.

Exceptions are of two types.

• Compile-time Error •
Run-time Exceptions

Compile-time Errors: Compile-

time Errors are called when an error occurs at the time of compile time. For example, Logical Errors, Syntax Errors.

Run-time Errors(Exceptions): When

an error occurs at run-time, it is called Exceptions. For example, divided by zero, out of memory, array out of bounds. Exception or run time error can also occur when the program

executes successfully. This can also happen if the IDE(Application) you are using crashes.

Simple Example for Exception

On example 'Divided by zero' exception has occurred. It will be handled by try-catch block.

```
#include <iostream>

using namespace std;



int main(){

int a = 5;

int b = 0;

int c;
```

```
  c = a/b;

  return 0;

}
```

Output :

[Warning] division by zero [-Wdiv-by-zero]

Try to Handle Exception/ Run-time Error

'try catch throw' statement is used to handle exceptions.

here. This • **Try :** There is a possibility of exception coming from the source code, that source code is given from which any block the exception is thrown.

• **Throw :** The throw keyword is used to throw the exception. It provides information about the error. The parameter given with the throw keyword is passed to the handler.

• **Catch :** The catch block is used to catch the exception thrown by throw statement. The exception is handled on the catch block.

Syntax for try-catch Block

```
try{

  some_statements;

  throw exception_parameter;
```

```
}

catch (data_type e){

  some_statements;

}
```

Syntax for multiple catch Block

```
try{

  some_statements;


  throw exception_parameter;

}

catch (data_type e1){

  some_statements;

}

catch (data_type e2){

  some_statements;

}

.....

catch (data_type eN){
```

```
  some_statements;

}
```

Handle Divided By Zero Exception(Single catch Block)

```
#include <iostream>

using namespace std;



int main(){

int a = 5;

int b = 0;

int c;

  try{

    if(b == 0){

      throw b;

    }

    c = a/b;

      cout<<"Value of c :                    "    <<c;

  }

  catch(int ex){
```

```
    cout<<"You cannot declare "<<ex<<" as denominator.";

  }

  return 0;

}
```

Output :

You cannot declare 0 as denominator.

Example for Multiple catch Block

```
#include <iostream>

using namespace std;



int main(){

int a = 5;

int b = 10;

int c;

  try{

    if(b == 0){

      throw b;

    }else if(b > a){
```

```
        throw "Not allowed - denominator is greater than numerator.";

    }

    c = a/b;

      cout<<"Value of c :                    "    <<c;

}

catch(int ex1){

   cout<<"You cannot declare "<<ex1<<" as denominator.";

}

   catch(char const* ex2){

   cout<<ex2;

}

return 0;

}
```

Output :

Not allowed - denominator is greater than numerator.

Exceptions	**Description**
bad_alloc	An exception is thrown when memory is not allocated.
bad_cast	An exception is thrown when dynamic cast fails.

bad_exception	**(KASHYAP SIR)** Exception is thrown by unexpected handler.
bad_function_call	Exception is thrown on bad call.
bad_typeid	Exception is thrown by typeid.
bad_weak_ptr	An exception is thrown when there is a bad weak pointer.
ios_base::failure	This stream is the base class of the exception.
logic_error	An exception is thrown when the source code fails to read.
runtime_error	Exception is thrown at runtime.
domain_error	An exception is thrown if there is an invalid domain.
future_error	An exception is thrown when there is a future error.
invalid_argument	An exception is thrown if there is an invalid argument.
length_error	An exception is thrown when there is a length error.
out_of_range	out of range Exception is thrown when it happens.
overflow_error	An exception is thrown when an Arithmetic overflow error occurs.
range_error	An exception is thrown when there is a range error in the internal computation.
system_error	An exception is thrown when there is a system error.
underflow_error	An exception is thrown when there is a mathematical underflow error.

Templates:-

Templates are an important feature of C++. Templates for generic programming in C++ provide ability. In generic programming, you can use generic type functions and Let's create classes.

Generic type functions and classes independent of other types (int, float, double etc.) It happens That's why the data type you pass as a parameter is generic function and class becomes of that type. By doing this you will get functions and functions for each data type separately. classes are not required and your program does not contain unnecessary code Is.

Through Templates, you create generic type functions and classes in C++. Template classes and functions are defined with parameters. This parameter is replaced by actual data type and then functions and classes are of that type Let's become

Template Functions

A template function is just like a normal function except that A template function can be executed for many types of data types (int, float etc.).

-

(KASHYAP SIR) If you perform an identical task with different data types through functions

If you want, you can create a template function.

For example, if you create a function of addition, then every data type (int, float etc.

) you have to create a separate function. Created by Integer type

The addition function will not add float values.

As you know the task is one (addition) but you have to create different functions.

Have to do In this situation, you can just create a template function for addition.

Which will add both integers and floats if needed.

Although you can do this work by function overloading also, but in that you will have to

Some code has to be written. By creating template function you have to write less code.

It will be necessary and you will be able to do the work easily.

Let us now see how you can create a template function in C++.

Its general syntax is not being checked.

```
template<class T>

return-type function-name (arguments with type T)

{

        // Statements to be executed (with T type)
```

```
}
```

In the above syntax T is a parameter which can be replaced by different data types is given and class is a keyword. You can use typename instead of class.

Basically it tells that the name of the data type will be passed here. let's do it now

An example is for understanding.

```
#include<iostream>

using namespace std;



template<class T>                              //Template Function

      T add(T x, T y)

       {

              return x + y;

        }



int main()

{

      cout<<"Sum of 5 & 3 is :"<<add(5,3); // Adding integers

      cout<<"Sum of 5.2 & 3.2 is :"<<add(5.2,3.2);                    //Adding floats
```

```
    return 0;

}
```

As you can see in the above example, integer and floating by the same function

point is being added to both types of values. This is possible through template functions.

It is possible. This program generates the given output.

```
Sum of 5 & 3 is : 8

Sum of 5.2 & 3.2 is : 8.4
```

C++ Template Classes

Sometimes it happens that you create similar classes with different data types.

Is. Like you create a class which number is taken as argument while constructing object

and later display this number by display() function. This

Thus, you have to create a separate class for each type of number (int, float, double).

In this situation you can create template class.

Sometimes it may happen that you do not know the type of user while making the program.

will enter the values. You may not even know that the data members of your class

(variables) will be of which type. For example, the current bank balance whole number of the user

Can be integer and can also be floating point number. In this situation you template

You can create classes.

SIR) When you create a template class, you can easily template all its functions.

Can create functions. Let us now see how you can create a template class.

Can Its general syntax is not being checked.

```
template<class T>
class class-name
{
        //Statements with T data type as required
}
```

In the above syntax as you know template is a keyword and actual data type

K is the placeholder.

The method of creating objects of Template class is also different. its syntax should not be checked

Used to be.

```
class-name <data-type> object-name(arguments-list);
```

Let us now try to understand template classes through a complete example.

```
#include <iostream>
using namespace std;
```

```
template<class T>

class NumDisplay

{

        private:

                        T whether;

        public:

                        NumDisplay(T n)

                        {

                                num = n;

                        }



                        void display()

                        {

                                cout<<"Number is : "<<num;

                          }

};



int main()

{
```

```
        NumDisplay<float> obj(5.2); //Passing data type & float value to constructor

        obj.display();



        return 0;

}
```

In the above example when object is constructed then T will be replaced by float and

The num variable declared in the class will be made of float type. template like this

Through classes, you can also handle such situations when you do not know that the user

What kind of value will enter. This program generates the given output.

```
Number is : 5.2
```

Stream Classes –

The ios, istream & ostream

Stream Classes in C++: The general name of the flow of data in Input and Output is Stream. For this reason, streams in C++ are called iostreams. an iostream as an object of a particular class Can be represented. For example, till now we have seen many examples of cin and cout stream objects. People who have learned “C” may wonder how these Stream Classes are of much use in place of Traditional I/O. So the answer is that stream classes are less error prone than traditional I/O. If you have studied “C” then you would know that if you use %f Control String instead of %d Control String by mistake, you do not get the desired result. But which type of control string is not used in “C++” stream, because stream

Object itself knows in what way it has to display which data or which type of value

-

Input is to be done in Object. Using streams mostly solves the problem of writing syntax in the program. Mistakes do not

happen. Which Existing Operator or Function can we overload and use with our created class, such as Insertion (<<) and Extraction (>>) Operators have been done. C++

Provides the ability to use Classes in the same way we use Basic Data Types. This helps us in programming and our program is more error free. If we want to do Graphics Programming, then how can we not use C's Traditional I/O, whereas iostream also proves to be fully useful in Graphics Programming. Because they provide us a best way, by which we can write data in files and format in memory, so that these formatted data can be used later in any other GUI and Dialog Box.

Stream Classes Hierarchy

Stream Classes can be seen as a Complex Hierarchy. However, we do not need to know this Hierarchy completely as I/O does. But even a little understanding is useful. We have already used some stream Classes. The extraction operator >> is a member of the istream class and the insertion operator << is a member of the ostream class. Both these classes are derived from an ios class. The cout object represents the standard output stream, which is responsible for the video display and is a predefined object of the ostream_withassign class, which is derived from the ostream class. In this way cin is object of istream_withassign class which is derived from istream class. Classes that work to send Output to Video Display and take Input from Keyboard are declared in Header File named iostream.h. We have included this Header File in each of our programs. Those classes which are mainly used for disk files are called

Defined in the Header File named FSTREAM.H. ios

Class is Base Class of iostream Hierarchy. It has many Constants and Member Functions, which are common to all types of Input and Output. Some of these, such as showpoint and fixed formatting flags, we have already seen. There is also a Pointer of streambuf class in ios Class which contains Actual Memory Buffers. Data is read through this memory buffer and data is written in this memory buffer. Also, various types of Low-Level Routines handle these data and take them from Input and print them on the screen in Output. There are also some low level routines in the streambuf class, which handle the data of the buffer. Normally we don't need to think about streambuf class. This class is automatically used by other classes as needed. But sometimes it is very convenient to use this buffer.

SIR) istream Class and ostream Class are derived from ios Class and they are responsible for Input and Output respectively. istream Class has Member Functions like get(), getline(), read() and Extraction (>>) Operators whereas ostream Class has put(), write() and Insertion (<<) Operators. iostream Class is derived from both istream and ostream classes by Multiple Inheritance. Classes derived from the iostream class can be used with devices such as disk files, which can perform both input and output functions at the same time. istream_withassign, ostream_withassign and iostream_withassign These nine classes are inherited or derived from istream, ostream and iostream respectively. In these, Assignment Operators have been added, so that cin, cout and other such operators can be assigned to others.
Can also be assigned with Streams.

Stream Classes in C++ : The ios Class

The ios class is delimited from all other classes and has the most features that operate on C++ streams. Features are there. The most important features under this class are Formatting Flags, Error-Status Bits and File There are Operation Modes.

Formatting Flags The Formatting Flags are a set of enum definitions in the ios class. These specify the choices of various aspects of Input and Output Format and Operations, acting like On/Off Switches. The complete list of these Formatting Flags is as follows: Skip (ignore) whitespace on input.

skipws	
left	Left adjust output [12.34].
right	Right adjust output [12.34].
internal	Use padding between sign or base indicator and number [+12.34].
dec	Convert to decimal.
oct	Convert to octal.
hex	Convert to hexadecimal.
showbase	Use base indicator on output (0 for octal, 0x for hex).
showpoint	Show decimal point on output.
uppercase	Use uppercase X, E, and hex output letters ABCDEF (the default is lowercase).
showpos	Display '+' before positive integers.

scientific	Use exponential format on floating-point output [9.1234E2].
fixed	Use fixed format on floating-point output [912.34].
unitbuf	Flush all streams after insertion.
stdio	Flush stdout, stderror after insertion.

There are many ways to set the formatting flags and different flags can be set from different angles. Since, all these Flags are members of the ios class, the Flags are generally used as a prefix to the ios name and the Flag is used after the Scope Resolution Operator. like

Ios::skipws

All Flags can be set by setf() and unsetcf() ios Member Functions. for example

```
cout.setf(ios::left);  cout >> “This                    // left justify output text
text is left-justified"; cout.unsetf(ios::left);
                                                        // return to default (right justified)
```

Many Formatting Flags are used with Manipulators, so let's look at Manipulators first.

Manipulators

Manipulators are those Formatting Instructions that can be directly inserted into a stream. We have used the endl manipulator which provides a new line. like:

cout << “Nandlal Gopal” << endl; **We**

have also used the setsiosflags() Manipulator. vehicle :

```
cout << setiosflags(ios::fixed) <<
  setiosflags(ios::showpoint) << var
```

From both these examples we can see that there are two types of Manipulators. First one which takes one Argument and second one which does not take Argument. No-Argument Manipulators are summarized in the following table.

Manipulator	**Purpose**
ws	Turn on whitespace skipping on input.
dec	Convert to decimal.
oct	Convert to octal.
hex	Convert to hexadecimal.
endl	Insert new line and flush the output stream.
ends	Insert null character to terminate an output string.
flush	Flush the output stream.

lock	Lock file handle.
unlock	Unlock file handle.

We can insert these Manipulators directly into the stream. for example the value of var To display the value in hexadecimal format, we can write the following statement:

cout << hex << var;

This statement keeps the No-Argument Manipulator in effect until the stream is destroyed. That's why we can print many numbers in Hexadecimal Format by using hex manipulator once. Manipulators of the following table take Arguments.

Manipulator	Argument	Purpose
setw()	field width (int)	Set field width for output.
setfill()	fill character (int)	Set fill character for output (default is a space).
setprecision()	Precision (int)	Set precision (number of digits displayed).
setiosflags()	formatting flags (long)	Set specified flags.
resetiosflags()	formatting flags (long)	Clear specified flags.

Those Manipulators which take Argument only have Effect on the Item before which they have been used. For example, if we want to set the number of Fields to be displayed, then we use setw() Use the Manipulator. If we have to decide the number of Fields of second Number, then again we have to Manipulator has to be used. like:

cout << "TTime " << setw(2) << hours << setw(2) << minutes;

Functions

There are also many functions in Ios class which can be used to set formatting flags and do various things. The following table lists the functions that do not deal with errors.

Function	Purpose
ch = fill();	Return the fill character (fills unused part of field; default is space).
fill(ch);	Set the fill character.
p = precision()	Get the precision (number of digits displayed for floating point).
precision(p);	Set the precision.
w = width();	Get the current field width (in characters).
width(w);	Set the current field width.

setf(flags);	Set specified formatting flags (e.g., ios::left).
unsetf(flags);	Unset specified formatting flags.
setf(flags, field);	First clear field, then set flags.

These functions are called using the Dot operator of the object of the specific stream. For example, to set the field width to 21, we can write the following statement:

```
cout.width(21);
```

Thus the following statement sets the Fill character to (*) Asterisk:

```
cout.fill(*');
```

We can use many functions to Directly Manipulate ios Formatting Flags. like
To set Left Justification, we can write the following statement:

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

To set back the Right Justification, we can write the following statement:

```
cout.unsetf(ios::left);
```

The 2-Argumnent Version of the setf() Function Uses the Second Argument to Reset All Flags of a Particular Type or Field. Then the Flag specified in the first Argument gets set. This function resets all set flags before setting a new flag. The following table is showing this Arrangement:

First argument: flags to set	Second argument; field to clear
dec, oct, hex	basefield
left, right, internal	adjustfield
scientific, fixed	floatfield

like:

```
cout.set(ios::left, ios::adjustfield);
```

This statement clears all the flags, then sets the left flag as Left Justification. In this way we can do the formatting of Input and Output. This Formatting is not only done for Display and Keyboard, it can also be done for Bust Files.

Stream Classes in C++ : The istream Class Derived from the ios

Class istream Class Performs Different Types of Functions or Extractions Related to Input Activities. We can get confused in relation to Extraction (>>) and its related Output Activities and Insertion. That's why an attempt has been made to show the difference between their children in the following table:

Function	Purpose
>>	Formatted insertion for all basic and overloaded types

get(ch);	Extract one character into ch.
get(str);	Extract characters into array str, until '\0'.
get(str, MAX);	Extract up to MAX characters into array.
get(str, SHARE)	Extract characters into array str until specified delimiter (typically '\n'). Leave delimiting char in stream.
get(str, MAX, DELIM)	Extract characters into array str until MAX characters or the DELIM character. Leave delimiting char in stream.
getline(str, MAX, DELIM)	Extract characters into array str until MAX characters or the DELIM character. Extract delimiting character.
putback(ch)	Insert last character read back into input stream.
ignore(MAX, DELIM)	Extract and discard up to MAX characters until (and including) the specified delimiter (typically '\n').
peek(ch)	Read one character, leave it in stream.
count = gcount()	Return number of characters read by a (immediately preceding) call to get(), getline(), or read().
read(str, MAX)	For files. Extract up to MAX characters into str until EOF.
seekg(position)	Sets distance (in bytes) of file pointer from start of file.
seekg(position, seek_dir)	Sets distance (in bytes) of file pointer from specified place in file: seek_dir can be ios::beg, ios::cur, ios::end.
position = tellg(pos)	Return position (in bytes) of file pointer from start of file.

We have already seen some of these functions. Most of them are used with Functions cin Object, which represents Keyboard as well as Disk Files. then the last four Functions' names can be used with files only.

Stream Classes in C++: The ostream class ostream

class handles the activities of Output or Insertion. Most commonly used member functions are given in the following table:

Function	**Purpose**
<<	Formatted extraction for all basic and overloaded types.
put(ch);	Insert character ch into stream.

flush();	Flush buffer contents and insert new line.
write(str, SIZE)	Insert SIZE characters from array str into file.
seekp(position)	Sets distance in bytes of file pointer from start of file.
seekp(position, seek_dir)	Set distance in bytes of file pointer from specified place in file. seek_dir can be ios::beg, ios::cur, or ios::end.
position = tellp()	Return position of file pointer, in bytes.

Its last four functions are used only with Disk Files.

Stream Classes in C++ : The iostream and the _withassign Classes Derived from istream and ostream, the iostream class mainly serves as the base class for other classes, from which other classes, especially iostream_withassign, can be derived.

It does not have any functions of its own except Constructors and Destructors. The class derived from the iostream class can perform both Input and Output functions. There is *_withassign* Class in our Library :

istream_withassign, Derived from stream

ostream_withassign, Derived from our enemy

iostream_withassign, Derived from iostream

_withassign classes are derived from classes except those classes which have overloaded assignment operator. That's why their objects can be copied. streams

Objects of some Classes in Library cannot be copied while Objects of some Class can be copied. Stream Library is designed for this, because which Stream Class

Object should not be copied. Because every object of this type is related to some streambuf object Associated stays. It

uses some memory area for storing data in streambuf object memory. So if we copy the Stream Object, then this fact; There will be confusion regarding whether we are copying the streambuf object as well or not. However, in some cases it is important to copy stream objects, as in the case of redirection, the predefined objects cin and cout are copied. The istream, ostream and iostream classes have been made uncopyable by making their overloaded copy constructor and assignment operator private, while objects of _withassign classes derived from them can be copied.

Stream Classes in C++: Predefined Stream Objects Through cin and cout Objects,

we have learned to use Derived Predefined Stream Objects from _withassign Classes. These are usually objects related to Keyboard and Monitor. Two other predefined objects are cerr and clog. These four are explained in the following table:

Class	Used for

-

eating	**SIR)** istream_withassign (Keyboard input)
cout	ostream_withassign (Normal screen output)
cerr	ostream_withassign (Error output)
clog	ostream_withassign (Log output)

The cerr object is generally used to display error messages and program diagnostics. Whatever output is sent to cerr, it is immediately displayed on the monitor. while we output cout When sent to the Output Object, that Output first goes to the Buffer of cout, then it is displayed on the Monitor. Also the output sent to cerr cannot be redirected. We can use the cerr object to see the output in the event that our program is terminating before completion. The clog object is similar to the cerr object. Note that it also cannot be redirected, but its output goes to the Buffer, whereas the output of cerr does not go to the Buffer.