DEPARTMENT OF COMMERCE (CA)

OBJECT ORIENTED PROGRAMMING WITH C++ (SEMESTER-IV)

II-B.COM (CA) **SUBJECT CODE - 18BCA42C**

UNIT V

Inheritance – extending classes – defining derived classes – single, multilevel, multiple, hierarchical and hybrid inheritance – classes for file stream Operations – opening and closing a file - sequential I/O operations.

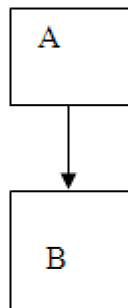
Inheritance

- Inheritance is a process of creating new classes from an existing class.
- It is a mechanism to derive a new class from an existing class.
- The existing (old) class is called as base class and the newly created class is called as derived class(subclass).
- The derived class inherits all properties from the base class.
- The derived class can also add some more features to this class.
- The base class is unchanged by its process.

Forms of inheritance:

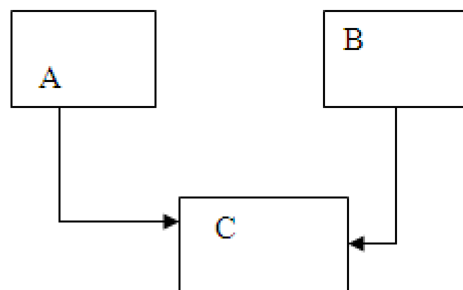
Single inheritance

A derived class with only one base class is called, single inheritance



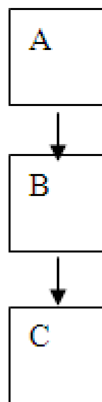
Multiple inheritance

One derived class with several Base classes is called multiple inheritance.



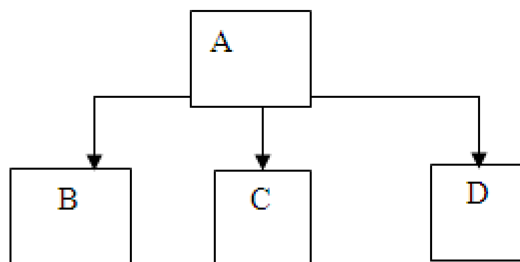
Multilevel inheritance

Deriving a class from another derived class is called as multilevel inheritance



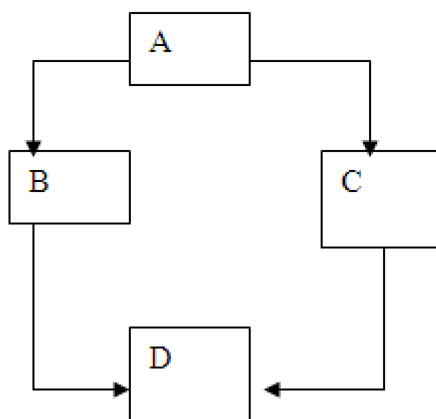
Hierarchical inheritance

Many derived class from single base class is called as hierarchical inheritance



Hybrid inheritance

A derived class can have more than one base class from more than one level.



Defining derived class:

A derived class can be defined by specifying its relationship with the base class along with its own details

General format:

```
Class derived_ Class_ name   visibility_mode  base _class _name
{

----        data members and member functions  of derived class
          ----
}
```

visibility mode:

It specifies how the features of the base class is derived. They are: Private (by default) and Public.

Private mode :

- When we specify 'private' mode, the 'public members' of the base class become private members of the derived class.
- So it can be accessed only be the member functions of the derived class.
- They can not be accessed by the object of the derived class.

Public mode:

- When we specify ' public' mode , the 'public members ' of the base class become 'public members' of the derived class and therefore they are accessible to the objects of the derived class

Note: In both cases, the private members are not inherited.

Example : class alpha : public beta

```
        {
1.          statements:
         }


            class abc :: xyz
            {
2.                statements;
            }
```

Single inheritance:

It is a process of creating new class from an existing base class.

The existing base class is known as the direct base class and the newly created class is called as a singly derived class.

Example.

```
// program for  single inheritance
#include <iostream.h>
class  basic_info
```

```
{
        int  rollno;
        char name[30];
        public:
                void getdata( );
                void                    display(                    );
};
class  physical_fit : public  basic_info
{
        float height, weight;
        public:
                void accept(  );
                void  show ( );
};

void  basic_info :: getdata( )
{
        cout<< “Enter Rollno  :”<<endl;
        cin >>rollno;
        cout<<”Enter name “<<endl’
        cin >>name;
}

void basic_info :: display( )
{
        cout <<rollno << name;
}

void physical_fit : : accept ( )
{
        getdata ( );
        cout <<”Enter Height and  Weight  “
        cin >>height >> weight;
}

void physical_fit : : show ( )
{
        display  ( );
        cout << height << weight ;
}
int main ( )
{
        physical_fit a;
        cout <<”Enter the details “<<endl;
a.      accept( );
        a. show   ( );
        return 0;         }
```

In this example , base class (basic_info) data members and member functions are inherited into the derived class (physical_fit). This derived class has its own data members and member Functions.

Making a private member inheritable:

- C++ supports three visibility modifier i.e. private, public, protected.
- With the help of the modifier 'protected' , we can access the private member .
- A data member declared as 'protected' is accessible by the member functions with its class and any class immediately derived from it.
- It can not be accessed by the functions outside these two classes.
- A class with all modifiers:

```
Class alpha
{
    private:      //   optional, visible to member function with in its class.

    protected:    // visible to member function of its own and
                     derived class

    public  :     //visible to all functions in  the program.
```

When the protectedmember is inherited in public mode , it becomes protected in the derived class and it can be access by the member functions of the derived class. It is ready for further inheritance

When the protectedmember is inherited in the private mode, it becomes private in the derived class.It is available to the member functions of the derived class, but is not available for further inheritance.

It is also possible to inherit a base class in protected mode. In protected derivation, both the public and protected members of the base class become protected member of the derived class

Example:

// program to access private member in private mode.

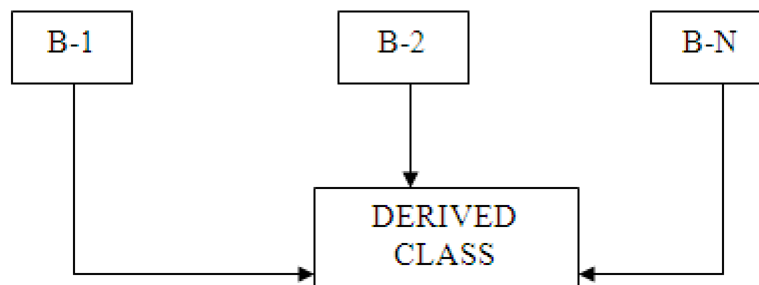
```
#include <iostream.h>
class  baseA
{
        protected  :
                        int value;
        public      :

                        baseA( )
                        {
                                cout<<"Enter a  number ";
                                cin >> value;
                        }
};        //  end of base class declaration
```

```
class  deriveB   : :  private  baseA
{
        public:
                void display ( )
                {
                        ++ value;            //   accessing  private  member
                        cout <<" The Value  is   : "<<value<< endl;
                }
};                  //  end of derived class definition

int main ( )
{
     deriveB  d;
    d. display( );
}
```

output:
Enter a number : 10   , The Value is : 11

Multiple inheritance:

It is a process of creating a new class which is derived from more than one base classes. A derived class can inherit attributes of two or more base classes.



Syntax:

```
Class  D : : visibility B-1, visibility  B-2,    . . .
        {
                ----
                ----      //  derived class data members and functions
        };
```

Example:

```
//  program for multiple inheritance
# include <iostream.h>
# include<iomanip.h>
class  basic_info
```

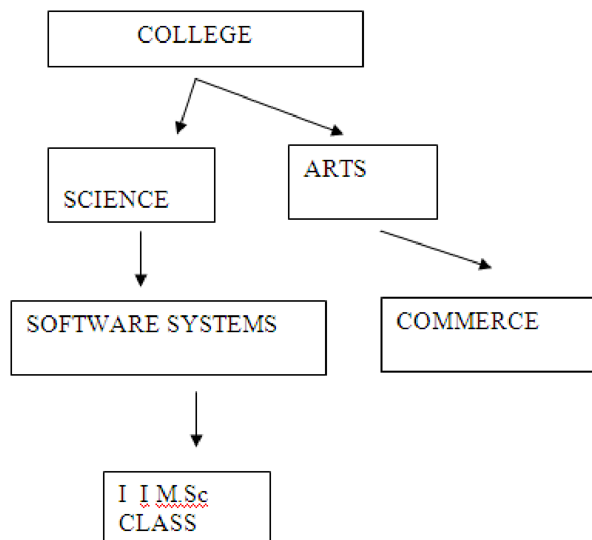
```
{
        int rollno;
        char name[20];
        public:
                void getdata( );
                void display( );
};      //  end of class definition…
class  academic_fit
{
        char course[20];
        char semester[10];
        public:
                void getdata( );
                void display( );
};      //  end of class   definition…
class financial_assit    : : private basic_info, private  academi_fit
{
        float amount;
        public:
                void getdata( );
                void display( );
};      //  end  of  class definition with multiple  inheritance
void basic_info : :  getdata ( )
{
        cout <<"Enter Rollno :"<<endl;
        cin >>rollno;
        cout<<"Enter name :"
        cin >> name;
}
void basic_info  : : display ( )
{
        cout <<rollno << name;
}
void academic_fit ( ) :: getdata( )
{
        cout <<"Enter course name:";
        cin >>course;
        cout<< "Enter semester";
        cin >>semester;
}
void academic_fit( ) :: display
{
        cout <<course<<semester;
}
void financial_assit( ) :: getdata( )
{
        basic_info : : getdata ( );
        academic_fit : : getdata( );
        cout <<"Enter amount in rupess :";
        cin>> amount;
```

```
        }
        void financial_assit : : display( )
        {
                basic_info  : : display ( );
                academic_fit : : display ( ) ;
                cout << amount ;
        }
        int  main ( )
        {
         financial_assit   f;
         cout<<"Enter the following details for financial assistance …\n";
         f. getdata( )'
         cout <<"Rollno     Name     course     semester    amount  \n";
         f.display  ( );
        return 0;
}
}
```

Hierarchical inheritance:

In hierarchical inheritance , a base class will include all features that are common to the Subclasses.
A subclass can inherit properties of the base class.
A subclass can serve as a base class for the lower level classes and so on.

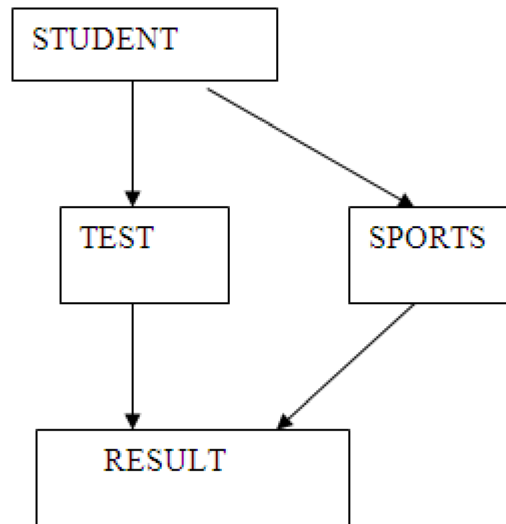
Example: Hierarchical classification of students



Hybrid inheritance:

* Sometimes two or more type of inheritance can be needed.

- This type of inheritance is called as hybrid inheritance or multilevel , multiple inheritance
- Let us consider the student result processing based on sports weightage.



// PROGRAM FOR HYBRID INHERITANCE

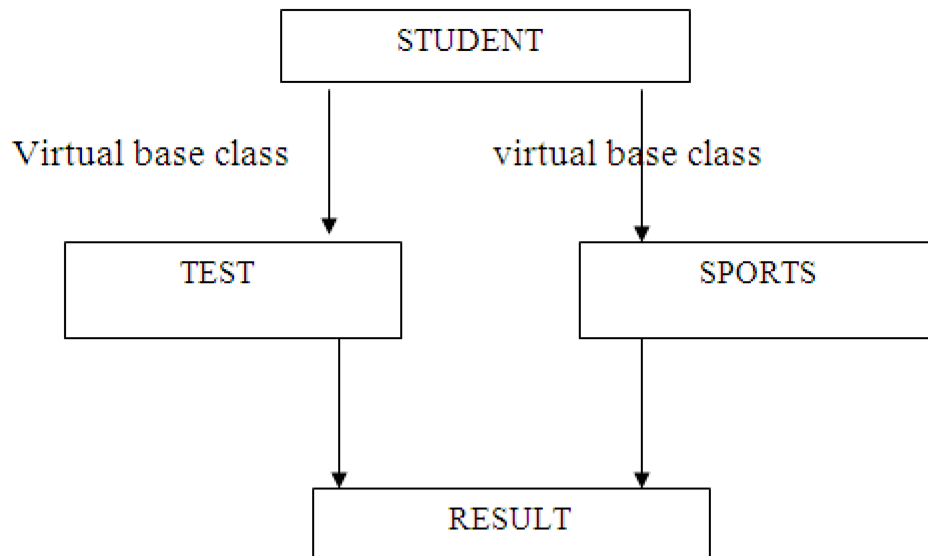
```
# include <iostream.h>
class student
{
        protected:
                int rollno;
        public:
                void getno( )
                {
                        cout <<"Enter Rollno";
                        cin >> rollno;
                }
                void putno( )
                {
                        cout<<"Roll  number "<<rollno<<endl;
                }
};
class test  :  public  student
{
        protected:
                float  part1,part2;
        public:
                void get_marks( )
                {
                        cout <<"Enter   two   marks ";
                        cin >>part1>>part2;
                }
                void put_marks( )
                {
```

```
                    cout <<"Marks scored  "<<part1 << part2;
               }
};
class  sports
{
          protected:
                    float score:
          public:
                    void get_score( )
                    {
                              cout <<"Enter sports weightage:";
                              cin >>score ;
                    }
                    void put_score( )
                    {
                              cout<<"Sports weightage"<<score;
                    }
};
class result  : public test, public sports   //….HYBRID
                                                       INHERITANCE
{
          float total;
          public:
                    void display ( )
};
void result :: display ( )
{
          total =part1 +  part2 + score;
          putno( );
          put_marks( );
          put_score( );
          cout <<"Total mark  : "<<total ;
}

int main ( )
{
          result   s1;
          s1.getno( );
          s1.get_marks( );
          s1.get_score( );
          s1.display ( );

          return 0;
}
```

Virtual base classes:

Virtual base class is a base class which is used to avoid multiple repetition of data members in the multiple inheritance.

Example :



Here we are assuming the class SPORTS derives rollno from the class STUDENT.

```
//  PROGRAM FOR VIRTUAL BASE CLASS

# include <iostream.h>

class student
{
        protected:
                int rollno;
        public:
                void getno( )
                {
                        cout <<"Enter Rollno";
                        cin >> rollno;
                }
                void putno( )
                {
                        cout<<"Roll  number "<<rollno<<endl;
                }
};
class test :  virtual  publicstudent
{
        protected:
                float  part1,part2;
```

```
        public:
                void get_marks( )
                {
                        cout <<"Enter   two   marks ";
                        cin >>part1>>part2;
                }
                void put_marks( )
                {
                        cout <<"Marks scored  "<<part1 << part2;
                }
};

class  sports  :  public virtual student
{
        protected:
                float score:
        public:
                void get_score( )
                {
                        cout <<"Enter sports weightage:";
                        cin >>score ;
                }
                void put_score( )
                {
                        cout<<"Sports weightage"<<score;
                }
};

class result  : public test, public sports
{
        float total;
        public:
                void display ( )
};

void result :: display ( )
{
        total =part1 +  part2 + score;
        putno( );
        put_marks( );
        put_score( );
        cout <<"Total mark  : "<<total ;
}

int main ( )
{
        result   s1;
        s1.getno( );
        s1.get_marks( );
        s1.get_score( );
```

```
        s1.display ( );

        return 0;
}
```

In this example ,the class student is inherited in test and sports classes. So the object of the derived class(result) can have only one copy of the student class.

Abstract classes:

- An abstract class is a class that is not used to create objects.
- An abstract class is used as a base class that is to be inherited by other classes.In the previous example "STUDENT " class is an abstract class since it was not used to create any object.

Managing Console I/O operations

- ✓ Every program takes some data as input and generates the processed data as output following the familiar input-output-process cycle.
- ✓ Cin and cout with the operators >> and << for the input and output operations. C++ supports a rich set of I/O functions and operations to control the I/O operations.
- ✓ C++ uses the concept of stream and stream classes to implement its I/O operations with the console and disk files.

C++ Streams :

- ➢ The I/O system in C++ is designed to work with a wide variety of devices including terminals, disks and tape drives. The I/O system, supplies an interface to the programmer that is independent of the actual device being accessed. This interface is known as stream.

- ➢ A stream is a sequence of bytes. It acts either as a source from which the input data can be obtained or as a destination to which the output data can be sent. The source stream that provides data to the program is called the input stream and the destination stream that receives output from the program is called the output stream.

- ➢ A program extracts the bytes from an input stream and inserts bytes into an output stream . The data in the input stream can come from the keyboard or any other storage device. A stream acts as an interface between the program annd the input/output device. Therefore, a C++ program handles data independent of the devices used.

- ➢ C++ contains several pre-defined streams that are automatically opened when a program begins its execution. Some of the these include cin and cout. Cin represents

input stream connected to the standard input device (i.e. keyboard) and cout represents output stream connected to the standard output device(i.e. screen).

C++ Stream Classes:

The C++ I/O system contains a hierarchy of classes that are used to define various streams to deal with both the console and disk files. These classes are called stream classes.

The following figure shows the hierarchy of the stream classes used for input and ouptput operations with the console unit. These classess are declared in the header file iostream.h. This file should be included in all the programs that communicate with the console unit.

ios is the base class for istream and ostream which are, in turn, base classes for iostream. The class ios is declared as virtual base class so that only one copy of its members are inherited by the iostream. The class ios provides the basic support for formatted and unformatted input while the class ostream provides the facilities for formatted output. The class iostream provides the facilities for handling both input and ouput streams.

Class : ios

- Contains the basic facilities that are used by all the input and output classes
- Declares constants and functions that are necessary for handling formatted input and output operations

Class: istream

- Inherits the properties of ios.
- Declares input functions such as get(), getline() and read()
- Contains overloaded extreaction operator >>.

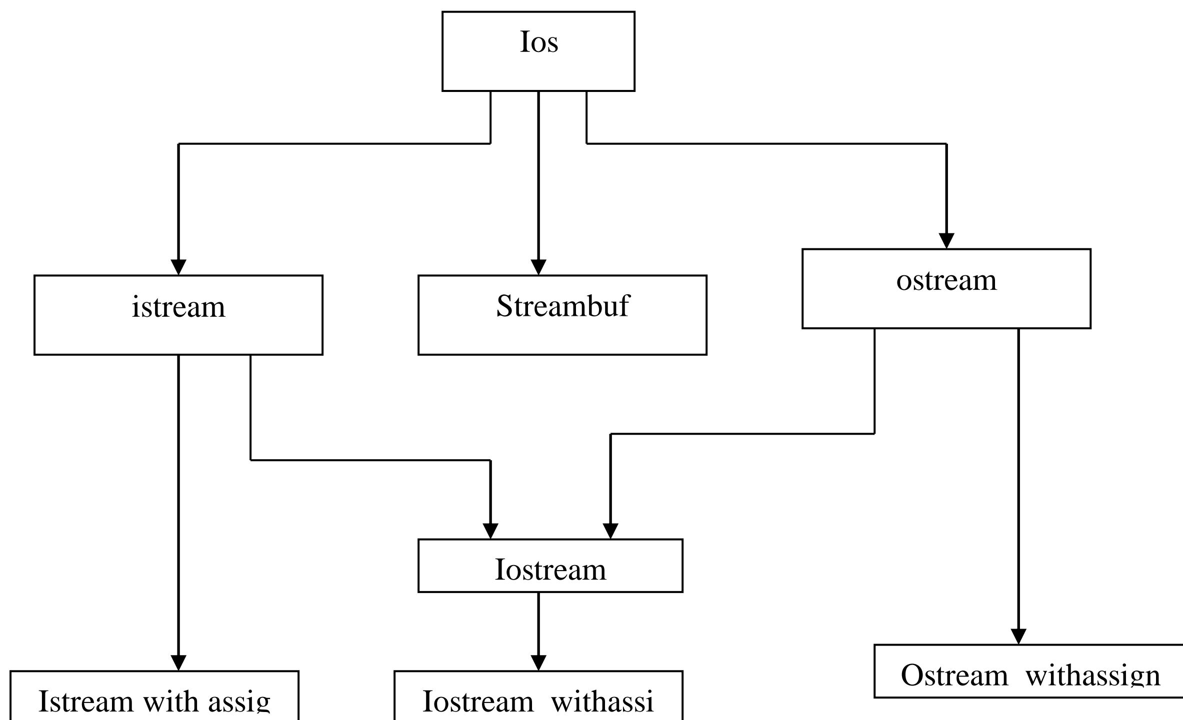
Class : ostream

- Declares for the output functions such as put() and write().
- Contains the overloaded insertion operator <<.
- Inherits the properties of ios.

Class : iostream

- Inherits the properties of ios, istream, ostream through multiple inheritance and thus contains all the input and output functions.

Stream classes for console I/O operations:



Unformatted I/O Operations:

Overloaded operators >> and <<

Objects cin and cout for the input and output of data of various types. This has been made possible by overloading the operators .. and << to recognize all the basic data types. The input operator >> is overloaded in the istream class and the >> is overloaded in the ostream class.

Statement for reading data from the keyboard.

cin>>variable1>>variable2>>…..

The input can be entered separated by white spaces or by line space like

Data1 data2 data3…

The >> operator gets character by character and assigns to the indicated location. The reading for a variable is terminated when the operator encounters an white space or a character that does not match the destinationtype.

For e.g.

```
int code;
cin>>code;
```

then the data such as

45

or

45D causes a termination.

The general format for displaying a data is

```
cout<<item1<<item2<<item3<<….
```

put() and get() functions:

The istream and ostream classes define two member functions get() and put() respectively to handle single character input/output operations.

Two types of get() functions.

1. get(char *) – Assigns the input character to its argument.

```
char c;
cin.get(c);     //gets a character from the keyboard
```

2. get((void) – Returns the input character.

```
char c;
c=cin.get();     //cin.get(c) is replaced
```

In this case, the value returned by the function get() is assigned to the variable c.

The function put(), a member ostream class is used to output a line of text, character by character.

For e.g.,

```
cout<<put('x');          // displays the character x
cout.put(ch);            // displays the value of the variable ch.
```

Number can be as an argument to the function put(). For e.g.,

```
cout.put(68);
```

Getline() And Write() Functions:

A line of text can be read and displayed using line-oriented input/output functions getline() and write() functions. The getline() function reads a whole line of text that ends with a newline chracter which is transmitted by the RETURN key. It is invoked by using the object cin as follows

```
cin.getline(line, size);
```

This function reads character input into the variable line. The reading is terminated when it encounters a newline character such as '\n' or when the size-1 characters are read.

```
For e.g.,
char name[20];
cin.getline(name,20);
```

'How are u' is typed as input and if return is pressed then the input is assigned to the array variable name. If the length of the text exceeds 20 characters then the reading is terminated. Here the blank spaces are also included in the text length.

In case of cin, it can read strings that do not contain white spaces.

The write() function displays an entire line and has the following form:

```
cout.write(line,size);
```

The first argument line represents the name of the string to be displayed and the second argument size indicates the number of characters to be displayed.

For eg.
```
cout.write(string1,3);
```
Displays the 3 characters of the string1.

Working With Files

Introduction

A file is a collection of related data stored in a particular area on the disk. Programs can be designed to perform the read and write operations on these files.

A program involes either or both of the following kinds of data communication:

1.Data transfwer between the console unit and the program.

2. Data transfer between the program and a disk file.

The I/O system of C++ handles file operations which are very similar to the console input and output operations. It uses file streams as an interface between the programs and the files.

The stream that supplies data to the program is known as input stream and the one that receives data from the program is known as output stream. In other words, input streams extracts data from the file and output stream inserts data to the file.

Classes for File Stream Operations

The I/O system of C++ contains a set of classes that define the file handling methods. They include ifstream, ofstream and fstream. They are derived from fstreambase class and from the corresponding iostream.h class. These classes are designed to manage the disk files and are declared in fstream.h and therefore this file should be included in any program that uses these files. The other file stream classes available are:

Class	Contents
filebuf	Its purpose is to set the file buffers to read and write.
Fstreambase	Provides operations common to file streams. Serves as a base for fstream, ifstream and ofstream classes. Contain open() and close() functions.
Ifstream	Provides for input operations. It contains open() with default input mode. Inherits the functions get(),getline(), read(),seekg() and tellg() from istream.
ofstream	Provides for output operations. It contains open() as the default output mode. It inherits put(),seekp(),tellp() and write() functions from ostream.
fstream	Provides support for simultaneous input and output operations.Contains open() as the default input mode. It inherits all the functions from istream and ostream classes through iostream.

Opening and Closing a file:

A filestream can be defined using the classes ifstream, ofstream and fstream that are contained in the header file fstream.h. The class to be used depends upon the purpose i.e., to read data from the file or write data to it. The two ways of opening a file are:

1.Using the constructor function of the class

2.Using the member function open () of the class

The first method is useful only one file in the stream is used and Second method is used when multiple files are to be managed using one stream.

Opening files using constructor

A constructor is used to initialize an object while it is created. In this case, a filename is used to initialize the file stream object. The steps involved here are:

1. Create a file stream object to manage the stream using the appropriate class. That is, the class ofstream is used to create the output stream.

2. Initialize the file object with the desired filename.

For e.g., if we want to open a file named “result”, then the command is

Program1

```
--------
--------
ofstream outfile(“results”);     //open for output only
---------
---------
```

The above statement creates an object to manage the output stream. If the file result is to be opened for reading only then the syntax could be of the form:

```
Program2
-------
-------
ifstream infile("result");        //open for input only.
-------
-------
```

Since the same file is used for both the purposes, the connection with a file is closed automatically when the stream object expires i.e, when the program terminates. In the above case the 'result' file is closed when the program1 is terminated and the file is disconnected from the outfile stream and the same process happens with the program2.

Instead of using two programs, for reading and writing of data, a single program to do both the functions. The use of a statement

```
outfile.close("result");
```

disconnects the file 'result' from the output stream.

```
Program1
    ofstream outfile("results");    //open for output only
    -------
    outfile.close("result");

    ifstream infile("result");       //open for input only.
    -------
    -------
    intfile.close("result");
```

When a file is opened for writing only, then a new file is created if there is no file of that name. If it already exists then the contents are deleted.

Opening files using open()

The function open() can be used to open multiple files that use the same stream object. In case of processing a set of sequential files a single stream object can be created and used to open each file in turn. The syntax for the same is:

```
filestreamclassstream-object;
stream-object.open("filename");
For eg.
ofstream outfile;        //create stream for output
outfile.open("data1);    //connect stream to data1
------
------
outfile.close();         //disconnect stream from data1
outfile.open("data2);    //connect stream to data2
------
```

```
outfile.close();            //disconnect stream from data2
```

Detecting End-Of-File

Detection fo the end-of-file is necessary for preventing any further attempt to read data from file. The syntax for is:

```
while(fin);
```

fin is an ifstream object and it returns 0 if any error occurs in the file operation including the end-of-file situation. Thus the while loop terminates when fin returns a value of zero on reaching the end of file.

The end of file can be detected by another method such as:

```
If (fin1.eof() != 0){exit(1);}
```

eof() is a member function of ios class. It returns a non-zero value if the end of file is encountered and a zero otherwise.

More about open() : file modes

The general form of open() with two arguments is

```
stream-object.open("filename",mode);
```

The mode parameter specifies the purpose for which the file is being opened. The mode has default values for reading and writing such as

ios::in for ifstream meaning open for reading only.

ios::out for ofstream meaning open for writing only.

The file mode parameter can take one or more of scuh constants defined in the class ios.:

Parameter	Meaning
ios::app	append to end of file
ios::ate	go to end of file on opening
ios::binary	binary file
ios::in	open file for reading only
ios::nocreate	open fails if the file does not exist
ios::noreplace	open fails if the file already exists
ios::out	open file for writing only
ios::trunc	delete contents of the file if its exists

The other points to be noted here are:

- Opening a file in ios::out mode also opens it in ios::trunc mode by default

- Both ios::app and ios::ate has the cursor at the end of the file when it is opened, but the difference is that
- ios::app allows to add data at the end of the file only and the ios::ate permits to add data or modify the existing data anywhere in the file.
- The parameter ios::app can be used only with the files capable of output
- Creating a stream using ifstream implies input and creating a stream using ofstream implies output.
- The fstream class does not provide a mode by default and therefore it should be stated explicitly
- The mode can combine two or more values using the bitwise OR operator. For e.g.

For eg.

```
fout.open("data",ios::app| ios::nocreate)
```

The above statement opens the file in the append mode but fails to open if does not exist.

REFERENCE:

1. E. Balaguruswamy , " Object oriented programming with C++", TataMcGraw Hill publishing company Limited, 1998.
2. K.R, Venugopal, Rajkumar, T. Ravishankar, "Mastering C++", tata mc graw – Hill publishing company Limited, 1998.
3. D. Ravichandran, Programming with C++", Tata McGraw – Hill published Company Limited.

Prepared By Dr.N.Shanmugavadivu