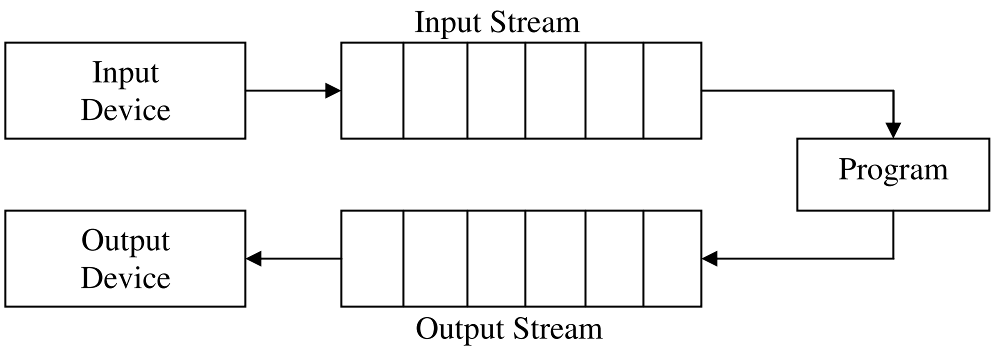
Chapter 11

Files and Streams

Stream

The I/P, O/P system in C++ is designed to work with a wide variety devices including terminates, Disk etc. Although each device is different the I/O system supplies an interface that is independent of the actual device which is accessed. This is known as Stream.

A Stream is a sequence of bytes. It acts as a source from which the I/P data can be obtain or as a destination to which the O/P data can be send. The source Stream that provides data to the program is called the I/P Stream and the destination Stream that receives O/P from the program is called the O/P Stream. In other words a program extracts data from I/P Stream and inserts data in to an O/P Stream. This is as shown below.



The data in the I/P Stream can come from the keyboard o any other storage device. Similarly the data in the O/P Stream can go to the screen or any other storage device.

C++ contains several predefined Stream that are automatically open when a program begins its execution. These include Cin & Cout which are automatically created when the program begins its execution. Cin represents the I/P Stream connected to the standard I/P device (Keyboard) and Cout represents the O/P Stream connected to the standard O/P stream connected to the standard O/P device (Screen).

Overloading the << and >> Operators

We have used the objects cin and cout (pre-defined in the iostream.h file) for the input and output of data of various types. This has been made possible by overloading the operators >> and << recognize all the basic C++ types. The >> operator is overloaded in the istream class and << is overloaded in the ostream class. The following is the general format of overloading >> operator.

```
cin>>variable1>>variable2>> . . . . >>variableN;
```

variable1,variable2, ... are valid C++ variable names that have been declared already. The input data are separated by white spaces and should match the type of variable in the cin list. Spaces, newlines and tabs will be skipped.

The operator >> reads the data character by character and assigns it to the indicated location. The reading for a variable will be terminated at the encounter of a white space or a character that does not match the destination type. For example, consider the following code.

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

Suppose the following data is given as input.

```
2345E
```

The operator will read the characters up to 5 and the value 2345 is assigned to code. The character E remains in the input stream and will be input to the next cin statement.

The following is the general format of overloading << operator.

```
cout<<item1<<item2<< . . . . <<itemN;
```

The items item1 through itemN may be variables or constants of any basic type.

<u>Character I/O with put( ) and get( ) Functions</u>

The classes istream and ostream define two member functions get( ) and put( ) respectively to handle the single character input/output operations. There are two types of get( ) functions. We can use both get(char *) and get(void) prototypes to fetch a character including the blank space, tab and the newline character. The get(char *) version assigns the input character to its argument and the get(void) version returns the input character.

e.g.
```
char c;
cin.get(c);
while(c!='\n')
{
    cout<<c;
    cin.get(c);
}
```

This code reads and displays a line of text (terminated by a newline character). Remember, the operator >> can also be used to read a character but it will skip the white spaces and newline character.

The get(void) version is used as follows.

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

The value returned by the function get( ) is assigned to the variable c.

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

e.g
```
cout.put(ch);
```

Write a Program to Character I/O with get( ) and put( ).

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

main()
{
int count=0;
clrscr();
char c;
cout<<"Input Text"<<endl;
cin.get(c);
while(c!='\n')
{
   cout.put(c);
   count++;
   cin.get(c);
}
cout<<"\nNumber of Characters = "<<count;
getch();
return(0);
}
```

String I/O with getline( ) and write( ) Functions

We can read and display a line of text more efficiently using the line-oriented input/output functions getline( ) and write( ). The getline( ) function reads a whole line of text that ends with a newline character (terminated by the Return key). This function can be invoked by using the object cin as follows.

cin.getline(line,size);

This function call invokes the function getline( ) which reads character input into the variable line. The reading is terminated as soon as either the newline character '\n' is encountered or size-1 characters are read (whichever occurs first). The newline character is read but not saved. Instead, it is replaced by the null character.

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

Write a Program to String I/O with getline( ).

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

main()
{
int size=20;
clrscr();
char name[20];
cout<<"Enter name = ";
cin>>name;
cout<<"Name = "<<name<<endl;
cout<<"Enter name again = ";
cin.getline(name,size);
cout<<"\nName = "<<name<<endl;
cout<<"Enter another name = ";
cin.getline(name,size);
cout<<"Name = "<<name;
getch();
return(0);
}
```

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 locates the number of characters to display. Note that it does not stop displaying the characters automatically when the null character is encountered. If the size is greater than the length of line, then the displays beyond the bounds of line.

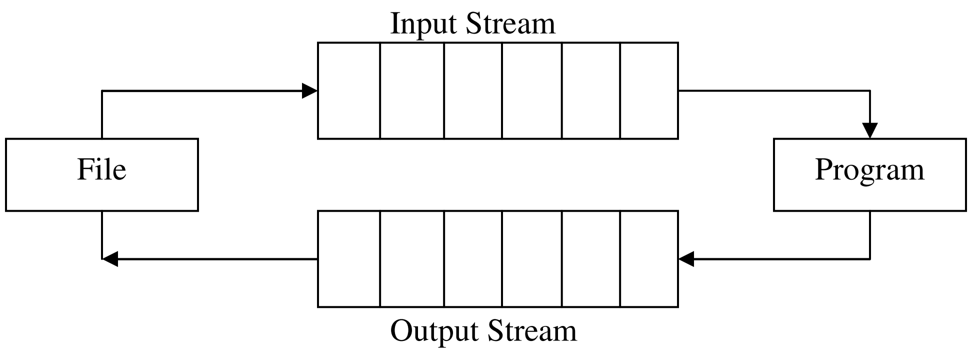
Write a Program to String I/O with write( ).

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

main()
{
char *str1="Amol ";
char *str2="Patil";
clrscr();
int m=strlen(str1);
int n=strlen(str2);
for(int i=1;i<n;i++)
{
   cout.write(str2,i);
   cout<<endl;
}
for(i=n;i>0;i--)
{
   cout.write(str2,i);
   cout<<endl;
}
cout.write(str1,m).write(str2,n);
cout<<"\n";
cout.write(str1,5);
getch();
return(0);
}
```

Files

A file is a collection of interrelated data stored on same of a disk. The I/P system of C++ handles file operations which are very much similar to the consol input-output operation. It uses file stream as an interface between the program and the files. The stream that supplies data to the program is known as an input stream and the stream which receives data from the program is known as O/P stream. In other words the I/P stream extracts data from the file and the O/P stream inserts data to the file as shown below.



Opening and Closing File

If we want to use a disk file, we need to specify following things about a file.

1. Suitable name for the File
2. Data type and structure
3. Purpose
4. Opening method

The filename is a stream of characters. It may contain two parts, a primary file name and an optional extension. The primary name and extension are separated by a period ".".

e.g.

Stud, Letter.txt, Emp.dat

For opening a file first of all we have to create a file stream and link it to the file name. A file stream can be define using the classes "ifstream" and "ofstream". These classes are declared in the header file fstream.h. The class to be used depends upon the purpose i.e. if we want to read data from the file, we have to create an I/P stream and if we want to write data to a file, we have to create an O/P stream.

A file can be opened in two different ways.

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

The first method is used when we want to manage multiple files using one stream and the second method is useful when we use only one file.

After performing the read and write operations on a file, we must close the file. It is necessary because all the data related to the file is written to the disk. The member function close( ) is used to close any file.

e.g.

The statement fout.close( ) breaks the link between O/P stream fout and the file STUD. It also writes the data to the file STUD. Similarly fin.close( ) close the link between fin and the file.

Opening Files using Constructor

A constructor is used to initialize an object when it is created. In this method a file name is used to initialize the file stream object. This involves the following steps.

1. Create a file stream either by using "ifstream" or "ofstream".
2. Initialize the file object with the desired file name.

e.g. The following statement opens a file STUD for output.

```
ofstream fout("STUD");
```

This statement create an O/P stream fout and link it to the file STUD. Similarly the following statement creates an I/P stream fin and link it to the file STUD for reading.

```
Ifstream fin("STUD");
```

We can write the statements as

```
fout<<"xyz\n";
fout<<"pqr";
fin>>name;
```

Opening File using Open Function

The function open( ) can be used to open multiple files that use the stream object. This is done as follows.

```
File-stream-class Stream Object;
Stream Object.open("File name");
```

e.g.

The following statements create an O/P stream fout and link it to the file STUD using open( ) function.

```
ofstream fout;
fout.open("STUD");
```

Similarly, the following statements creates an I/P stream fin and link it to the file ITEM;

```
ifstream fin;
fin.open("ITEM");
```

When a file is opened for writing only, a new file is created if there is no file of that name. If a file by that name exists already, then its contents are deleted and the file is presented as a clean file.

Write a Program to Create a file Item and store the Information about an Item

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

main()
{
int no;
float cost;
clrscr();
ofstream fout("Item");
cout<<"Enter Item no & cost = ";
cin>>no>>cost;
fout<<no<<endl;
fout<<cost;
fout.close();
getch();
return(0);
}
```

Write a Program to Read the Content of a file Item and Print them on the Screen.

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

main()
{
int n;
float c;
clrscr();
ifstream fin("Item");
fin>>n;
fin>>c;
cout<<"Item no = "<<n<<endl;
cout<<"Item cost = "<<c;
fin.close();
getch();
return(0);
}
```

Write a Program to Write & Read the Content of a file Item and Print them on the Screen.

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

```
#include<fstream.h>

main()
{
int no;
float cost;
clrscr();
ofstream fout("Item");
cout<<"Enter Item no & cost = ";
cin>>no>>cost;
fout<<no<<endl;
fout<<cost;
fout.close();

ifstream fin("Item");
fin>>no;
fin>>cost;
cout<<"Item no = "<<no<<endl;
cout<<"Item cost = "<<cost;
fin.close();
getch();
return(0);
}
```

Write a Program to Store the Information about 5 Student in a file stud and to Read the Content and Print them on the Screen.

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

main()
{
int rn;
char name[30];
clrscr();
ofstream fo("stud");
for(int i=1;i<=3;i++)
{
   cout<<"Enter Roll no & name of "<<i<<" Student = ";
   cin>>rn>>name;
   fo<<rn<<"\t";
   fo<<name<<"\n";
}
fo.close();
ifstream fin("stud");
cout<<"Roll no\tName"<<endl;
for(i=0;i<3;i++)
{
   fin>>rn;
   fin>>name;
   cout<<rn<<"\t";
   cout<<name<<endl;
}
fin.close();
getch();
return(0);
}
```

Rewrite the above Program by Checking the end of file Condition.

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

main()
{
int rn;
char name[30];
clrscr();
ofstream fo("stud");
for(int i=1;i<=3;i++)
{
   cout<<"Enter Roll no & name of "<<i<<" Student = ";
   cin>>rn>>name;
   fo<<rn<<"\t";
   fo<<name<<"\n";
}
fo.close();
ifstream fin("stud");
cout<<"Roll no\tName"<<endl;
while(fin)
{
   fin>>rn;
   fin>>name;
   cout<<rn<<"\t";
   cout<<name<<endl;
}
fin.close();
getch();
return(0);
}
```

Write a Program to Store the Information about 5 Student in a file stud.

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

main()
{
int rn;
char name[30];
clrscr();
ofstream fo("stud");
fo<<"Roll no\tName\n";
for(int i=1;i<=3;i++)
{
   cout<<"Enter Roll no & name of "<<i<<" Student = ";
   cin>>rn>>name;
   fo<<rn<<"\t";
   fo<<name<<"\n";
}
fo.close();
getch();
return(0);
}
```

Write a Program to Store 10 nos. in a file no and print them on the Screen.

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

main()
{
int n,i;
clrscr();
ofstream fo("no");
cout<<"Enter 10 nos = ";
for(i=0;i<10;i++)
{
   cin>>n;
   fo<<n<<"\n";
}
fo.close();
ifstream fin("no");
cout<<"Number in File = ";
while(fin)
{
   fin>>n;
   cout<<n<<" ";
}
fin.close();
getch();
return(0);
}
```

Write a Program to read the Contents of a file no. Check every no, Wheather it is even no or odd. If an no is even then write that no in a file even otherwise write it in a file odd.

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

```
#include<conio.h>
#include<fstream.h>

main()
{
int n,i;
clrscr();
ifstream fin("no");
ofstream f1,f2;
f1.open("even");
f2.open("odd");
while(fin)
{
   fin>>n;
   if(n%2==0)
     f1<<n<<"\n";
   else
     f2<<n<<"\n";
}
fin.close();
f1.close();
f2.close();
fin.open("even");
cout<<"Content of Even file = ";
while(fin)
{
  fin>>n;
  cout<<n<<" ";
}
fin.close();
fin.open("odd");
cout<<"\nContent of Odd file = ";

while(fin)
{
  fin>>n;
  cout<<n<<" ";
}
fin.close();
getch();
return(0);
}
```

Write a Program to Working with Multiple Files.

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

main()
{
```

```
ofstream fout;
clrscr();
fout.open("country");
fout<<"United States of America\n";
fout<<"Kingdom\n";
fout<<"Korea\n";
fout.close();

fout.open("capital");
fout<<"Washington\n";
fout<<"London\n";
fout<<"Seoul\n";
fout.close();

const int n=80;
char line[n];
ifstream fin;
fin.open("country");
cout<<"Content of Country File"<<endl;

while(fin)
{
   fin.getline(line,n);
   cout<<line<<endl;
}
fin.close();
fin.open("capital");

cout<<"Content of Capital File"<<endl;

while(fin)
{
   fin.getline(line,n);
   cout<<line<<endl;
}
fin.close();
getch();
return(0);
}
```

Write a Program to Reading from two Files Simultaneously.

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
#include<stdlib.h>

main()
{
const int size=80;
char line[size];
clrscr();
ifstream fin1,fin2;
fin1.open("country");
fin2.open("capital");
for(int i=1;i<=10;i++)
{
   if(fin1.eof()!=0)
   {
     cout<<"Exit from Country\n";
     exit(1);
   }
   fin1.getline(line,size);
   cout<<"Capital of "<<line<<endl;
   if(fin2.eof()!=0)
   {
     cout<<"Exit from Capital\n";
     exit(1);
   }
   fin2.getline(line,size);
   cout<<line<<endl;
}
getch();
return(0);
}
```

Detecting end of File

In the above program, the end of file is detected by the following statement.

while(fin);.

Detection of the end of file condition is necessary for preventing any further attempt to read data from the file. The ifstream object fin returns a value zero if any error occurs in the file operation including the end of file condition. Thus the while loop terminates when fin returns a value zero on reading the end of file condition.

File Modes

We have used ifstream and ofstream constructor and the function open( ) to create new files as well as to open exiting files. Remember, in both these methods, we used only one argument that was the filename. However, these functions can take two arguments, the second one for specifying the file mode. The general form of the function open( ) with two arguments is:

Stream-Object.open("File name",mode);

The second argument mode (called file mode parameter) specifies the purpose for which the file is opened. The prototype of these class member functions contain default values for the second argument and therefore they use the default value in the absence of the actual values. The default values are as follows

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

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

Table File Mode Parameters

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 it exists

Note

1. Opening a file in ios::out mode also opens it in the ios::trunc mode by default.
2. Both ios::app and ios::ate take us to the end of the file when it is opened. But the difference between the two parameters is that the ios::app allows us to add data to the end of the file only, while ios::ate mode permits us to add data or to modify the existing data anywhere in the file. In both the cases, a file is created by the specified name, if it does not exist.
3. The parameter ios::app can be used only with the files capable of output.
4. Creating a stream using ifstream implies input and creating a stream using ofstream implies output. So in these cases it is not necessary to provide the mode parameters.
5. The mode can combine two or more parameters using the bitwise OR operator (symbol |) as shown below.

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

File Pointers

Each file has two associated pointers known as the File Pointers. One of them is called the input pointer (or get pointer) and the other is called the output pointer (or put pointer). We can use these pointers to move through the files while reading and writing. The input pointer is used for reading the content of a given file location and the output pointer is used for writing to a given file location. Each time an input or output operation takes place, the appropriate pointer is automatically advanced.

When we open a file in read-only mode, the input pointer is automatically set at the beginning so that we can read the file from the start. Similarly, when we open a file write-only mode, the existing contents are deleted and the output pointer is set at the beginning. This enables us to write to the file from the start. In case, we want to open an existing file to add more data, the file is opened in ‘append’ mode. This moves the output pointer to the end of the file.

Functions for Manipulation of File Pointers

All the actions on the file pointer take place automatically by default. How do we then move a file pointer to any other desired position inside the file? This is possible if we can take control of the movement of the file pointers ourselves.

seekg( )	Moves get pointer (input) to a specified location.
seekp( )	Moves put pointer (output) to a specified location.
tellg( )	Gives the current position of the get pointer.
tellp( )	Gives the current position of the put pointer.

e.g. infile.seekg(10);

moves the file pointer to the byte number 10. Remember, the bytes in a file are numbered beginning from zero. Therefore, the pointer will be pointing to the 11th byte in the file. Consider the following statements

```
ofstream fileout;
fileout.open("hello",ios::app);
int p = fileout.tellp( );
```

On execution of these statements, the output pointer is moved to the end of the file "hello" and the value of p will represent the number of bytes in the file.

Specifying the Offset

We have just now seen how to move file pointer to a desired location using the 'seek' functions. The argument to these functions represents the absolute position in the file. 'Seek' functions seekg( ) and seekp( ) can also be used with two arguments as follows.

```
seekg(offset,refposition);
seekp(offset,refposition);
```

The parameter offset represent the number of bytes the file pointer is to be moved from the location specified by the parameter refposition. The refposition takes one of the following three constants defined in the ios class.

ios::beg	Start of the file
ios::cur	Current position of the pointer
iso::end	End of the file

Sequential Input and Output Operator

The file stream class support a number of member functions for performing the input and output operations on files. One pair of functions, put( ) and get( ), are designed for handling a single character at a time. Another pair of functions, write( ) and read( ), are designed to write and read blocks of binary data.

The function put( ) writes a single character to the associated stream. Similarly, the function get( ) reads a single character from the associated stream.

Note

We have used an fstream object to open a file. Since an fstream object can handle both the input and output simultaneously, we have opened the file in ios: :in / ios: :out mode. After writing the file, we want to read the entire file and display its contents. Since the file pointer has already moved to the end of the file, we must bring it back to the start of the file. This is done by the statement.

```
file.seekg(0);
```

The functions write( ) and read( ), unlike the functions put( ) and get( ), handle the data in binary form. This means that the values are stored in the disk file in the same format in which they are stored in the internal memory.

Binary Format	00001010 00100010	(2 Bytes)
Character Format	2 0 9 4	(4 Bytes)

The above format shows how an int value 2094 is stored in the binary and character formats. An int takes two bytes to store its value in the binary form, irrespective of its size. But a 4-digit int will take four bytes to store it in the character form.

The binary format is more accurate for storing the numbers as they are stored in the exact internal representation. There are no conversion while saving the data and therefore saving is much faster. The input and output functions takes the following form.

```
infile.read((char *) &v, sizeof(v));
outfile.write((char *) &v, sizeof(v));
```

Write a Program to I/O Operation on Characters.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<fstream.h>
#include<string.h>

main()
{
char str[80];
clrscr();
cout<<"Enter a String = ";
cin>>str;
int len=strlen(str);
fstream file;
file.open("text",ios::in | ios::out);
for(int i=0;i<len;i++)
   file.put(str[i]);
file.seekg(0);
char ch;
while(file)
{
   file.get(ch);
   cout<<ch;
}
getch();
return(0);
}
```

Write a Program to I/O Operation on Binary Files.

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
#include<iomanip.h>

main()
{
float height[4]={175.56,875.55,654.65,458.25};
clrscr();
ofstream out("binary");
out.write((char *)&height,sizeof(height));
out.close();
for(int i=0;i<4;i++)
   height[i]=0;

ifstream in("binary");
in.read((char *)&height,sizeof(height));
for(i=0;i<4;i++)
{
   cout.setf(ios::showpoint);
   cout<<setw(10)<<setprecision(2)
        <<height[i];
}
in.close();
getch();
return(0);
}
```

Reading and Writing Class Object

We mentioned earlier that one of the shortcoming of the I/O system of C is that it cannot handle user defined data types such as class objects. Since the class objects are the central elements of C++ programming, it is quite natural that the language supports features for writing to and reading from the disk files objects directly. The binary input and output functions read( ) and write( ) are designed to do exactly this job.

These functions handle the entire structure of an object as a single unit, using the computers internal representation of data. For instance, the function write( ) copies a class object from memory byte by byte with no conversion. One important point to remember is that only data members are written to the disk file and the member functions are not.

Updating File : Random Access

Updating is a routine task in the maintenance of any data file. The updating would include one or more of the following tasks.

1. Displaying the content of a file.
2. Modifying an exiting item.
3. Adding a new item.
4. Deleting an exiting item.

These actions require the file pointers to move to a particular location that corresponds to the item/object under consideration. This can be easily implemented if the file contains a collection of item/objects of equal lengths. In such cases, the size of each object can be obtained using the statement.

int object_length = sizeof(object);

Then, the location of a desired object, say the mth object, may be obtained as follows.

int location = m * object_length;

The location gives the byte number of the first byte of the mth object. Now, we set the file pointer to reach this byte with the help of seekg( ) or seekp( ).

We can also find out the total number of objects in a file using the object_length as follows.

Int n = file_size/object_length;

The file_size can be obtained using the functions tellg( ) or tellp( ) when the file pointer is located at the end of the file.

Write a Program to File Updating & Random Access.

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
#include<iomanip.h>

class inv
{
  private:
       char name[10];
       int code;
       float cost;
  public:
       void getdata();
       void putdata();
};

void inv :: getdata()
{
   cout<<"Enter Name = ";
   cin>>name;
   cout<<"Enter Code = ";
   cin>>code;
   cout<<"Enter Cost = ";
   cin>>cost;
}
```

```
void inv :: putdata()
{
   cout<<setw(10)<<name
       <<setw(10)<<code
       <<setprecision(2)<<setw(10)<<cost
       <<endl;
}

main()
{
inv item;
clrscr();
fstream inout;
inout.open("stock.txt",ios::ate | ios::in | ios::out |
ios::binary);
inout.seekg(0,ios::beg);

cout<<"Current Content of Stock\n";
while(inout.read((char *)&item,sizeof item))
{
   item.putdata();
}
inout.clear();

cout<<"Add in Item\n";
item.getdata();
char ch;
cin.get(ch);
inout.write((char *)& item, sizeof item);

inout.seekg(0);
cout<<"Content of Append File\n";
while(inout.read((char *)&item,sizeof item))
{
   item.putdata();
}

int last=inout.tellg();
int n=last/sizeof(item);
cout<<"Number of Objects = "<<n<<endl;
cout<<"Total Bytes in File = "<<last<<endl;

cout<<"Enter Object no to be Updated\n";
int obj;
cin>>obj;
cin.get(ch);
int loc=(obj-1)*sizeof(item);
if(inout.eof())
  inout.clear();
inout.seekp(loc);
cout<<"Enter new Values of Object\n";
item.getdata();
cin.get(ch);
inout.write((char *)&item,sizeof item)<<flush;
inout.seekg(0);
cout<<"Content of Updated File\n";
while(inout.read((char *)&item,sizeof item))
{
   item.putdata();
}
inout.close();
getch();
return(0);
}
```

Error Handling During File Operations

So far we have been opening and using the files for reading and writing on the assumption that everything is fine with the files. This may not be true always. For instance, one of the following things may happen when dealing with the files.

1. A file which we are attempting to open for reading does not exist.
2. The file name used for a new file may already exist.
3. We may attempt an invalid operation such as reading past the end-of-file.
4. There may not be any space in the disk for storing more data.
5. We may use an invalid file name.
6. We may attempt to perform an operation when the file is not opened for that purpose.

The class ios supports several member functions that can be used to read the status recorded in a file stream. These functions along with their meaning are listed in following table.

Function	**Return value and meaning**
eof( )	Returns true (non-zero value) if end-of-file is encountered while reading; otherwise returns false (zero).
fall( )	Returns true when an input or output operation has failed.
bad( )	Returns true if an invalid operation is attempted or any unrecoverable error has occurred. However, if it is false, it may be possible to recover from any other error reported and continue operation.
good( )	Returns true if no error has occurred. This means, all the above functions are false. For instance, if file.good( ) is true, all is well with the stream file and we can proceed to perform I/O operations. When it returns false, no further operations can be carried out.

Command-Line Arguments

Like C++ too supports a feature that facilities the supply of arguments to the main( ) function. These arguments are supplied at the time of invoking the program. They are typically used to pass the names of data files.

e.g.

```
C > exam data results
```

Here, exam is the name of the file containing the program to be executed, and data and results are the filenames passed to the program as command-line arguments.

The command-line arguments are typed by the user and are delimited by a space. The first argument is always the filename (command name) and contains the program to be executed. The main( ) functions which we have been using up to now without any arguments can take two arguments as shown below.

```
main(int argc,char *argv[])
```

The first argument argc (known as argument counter) represents the number of arguments in the command line. The second argument argv (known as argument vector) is an array of char type pointers that point to the command line arguments. The size of this array will be equal to the value of argc. For instance, for the command line.

```
C > exam data results
```

The value of argc would be 3 and the argv would be an array of three pointers to strings as shown below.

```
argv[0] - - > exam
argv[1] - - > data
argv[2] - - > results
```

Note that argv[0] always represents the command name that invokes the program. The character pointers argv[1] and argv[2] can be used as file names in the file opening statements as shown below.

```
....
infile.open(argv[1]);      // open data file for reading
....
....
outfile.open(argv[2]);    // open results file for writing
....
....
```

Write a Program to Command Line Argument.

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
#include<process.h>

main(int argc,char *argv[])
{
clrscr();
int no[9]={22,44,65,33,22,74,73,84,85};
if(argc!=3)
{
  cout<<"Argc = "<<argc<<endl;
  cout<<"Error in Arguments";
  exit(1);
}
ofstream fout1,fout2;

fout1.open(argv[1]);
if(fout1.fail())
{
  cout<<"Could not open the file "<<argv[1]<<endl;
  exit(1);
}

fout2.open(argv[2]);
if(fout2.fail())
{
  cout<<"Could not open the file "<<argv[2]<<endl;
  exit(1);
}

for(int i=0;i<9;i++)
{
   if(no[i]%2==0)
     fout2<<no[i]<<" ";
   else
     fout1<<no[i]<<" ";
}

fout1.close();
fout2.close();
ifstream fin;
char ch;

for(i=0;i<argc;i++)
{
   fin.open(argv[i]);
   cout<<"Content of "<<argv[i]<<endl;
   do
   {
     fin.get(ch);
     cout<<ch;
   }
   while(fin);
   cout<<endl;
   fin.close();
}
getch();
return(0);
}
```

Write a Program to Create Object ‘person’ consists of ‘Name’ and ‘Age’. Write this object onto disk.

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
#include<iomanip.h>
#include<stdio.h>

class person
{
  private:
      char name[30];
      float age;
  public:
      void getdata()
      {
       cout<<"Enter Person Name & Age = ";
       gets(name);
       cin>>age;
      }
      void putdata()
      {
       cout<<"Name = ";
       puts(name);
       cout<<"Age = "<<age;
      }
};

main()
{
person p;
clrscr();
fstream file;
file.open("person",ios::in | ios::out);
p.getdata();
file.write((char *)& p,sizeof(p));
cout<<"Output\n";
file.read((char *)& p,sizeof(p));
p.putdata();
getch();
return(0);
}
```