8...

Working with Files

Objectives...

- To learn Stream Classes for File operations.
- To study different File operations.
- To understand File updating with random access.
- To learn Error handling during file operations.
- To study command line argument.

8.1 INTRODUCTION [S-18]

- It is difficult to handle large amount of data only with use of keyboard and screen. So, we need to store data on secondary storage device as data structures called files.
- A computer system stores programs and data in secondary storage in the form of files. A file is a collection of related information.
- A file is a sequence of bits, bytes, lines or records whose meaning is defined by its creator and user. A file is named and is referred to by its name.
- To define a file properly, it is necessary to consider the operations which can be performed on files. The Operating System (OS) provides most of the essential file manipulation services such as create, open, write, read, rewind, close and delete.
- A program typically involves data communication between the console and the program or between the files and program or even both. The program must at least perform data exchange between processor and main memory.
- Computer programs are written to store the data (a write operation on file) and to read the data (a read operation on file).
- A computer program involves following two types of data communication:
  - Data transfer between the keyboard/monitor (console) and the program.
  - Data transfer between the program and the disk file.
- Fig. 8.1 shows the concept of data communication.

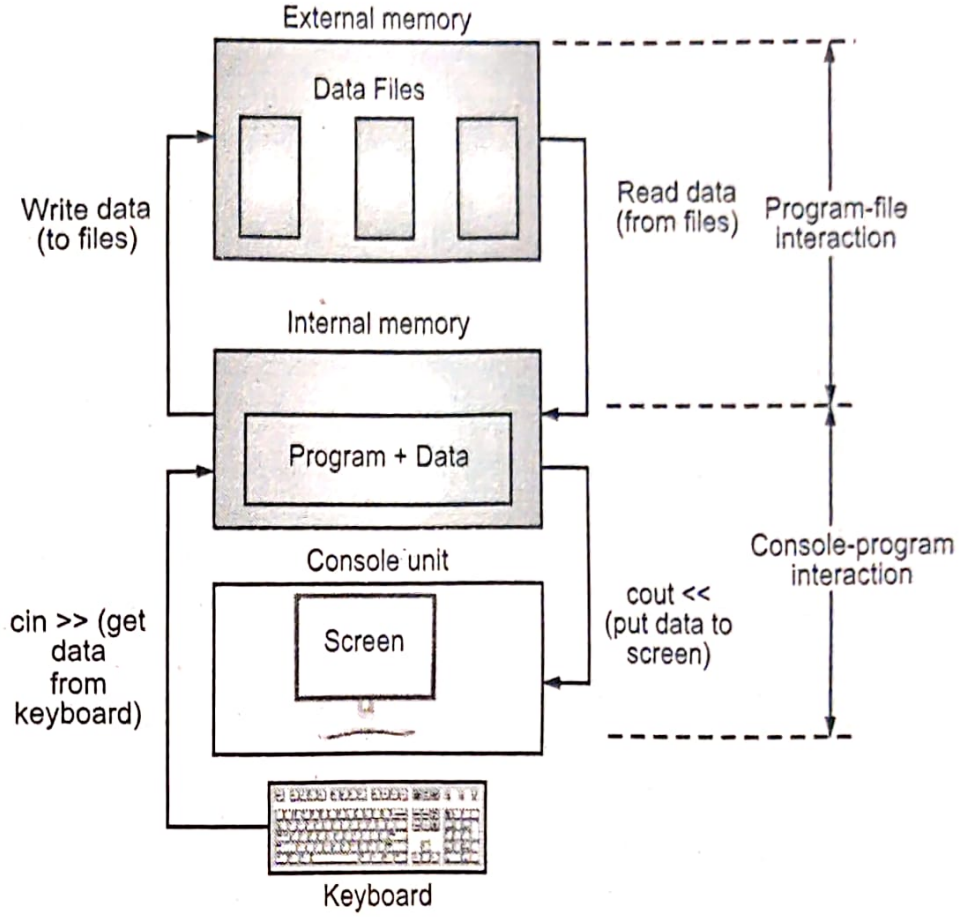

Fig. 8.1: Console Program and File Interaction

- C++ uses the concept of stream and stream classes to implement its I/O operations with the console and disk files.
- Files are either text files or binary files. A text files can be created using sophisticated word processor. Text files can be easily read or printed. Binary files are data files and program files.
- Binary files can not be easily read or write but size of binary file is smaller than that of text file. The speed at which programs can read data from binary file is very high as compare to text file.
- In this chapter, we will discuss various methods available for storing and retrieving the data from files. Using files we can transfer data from program to file on secondary storage or vice versa.

8.2 STREAM CLASSES FOR FILE OPERATIONS

- The file handling techniques of C++ support file manipulation in the form of stream objects. The stream object `cin` and `cout` are used extensively to deal with the standard input and output devices.
- The C++ input/output system support a hierarchy of classes that are used to manipulate both the console and disk files, called stream classes.
- The input/output system of C++ has a group of classes that defines all the file handling functions. These classes are:
  - `ifstream`: For handling input files.
  - `ofstream`: For handling output files.
  - `fstream`: For handling files on which both input and output can be performed.

- Input stream is declared as a class ifstream. It is derived from istream. Output stream is declared as a class ofstream. It is derived from ostream. Input/Output stream should belong to fstream class. It is derived from iostream.
- The hierarchy of C++ file stream classes is shown in Fig. 8.2

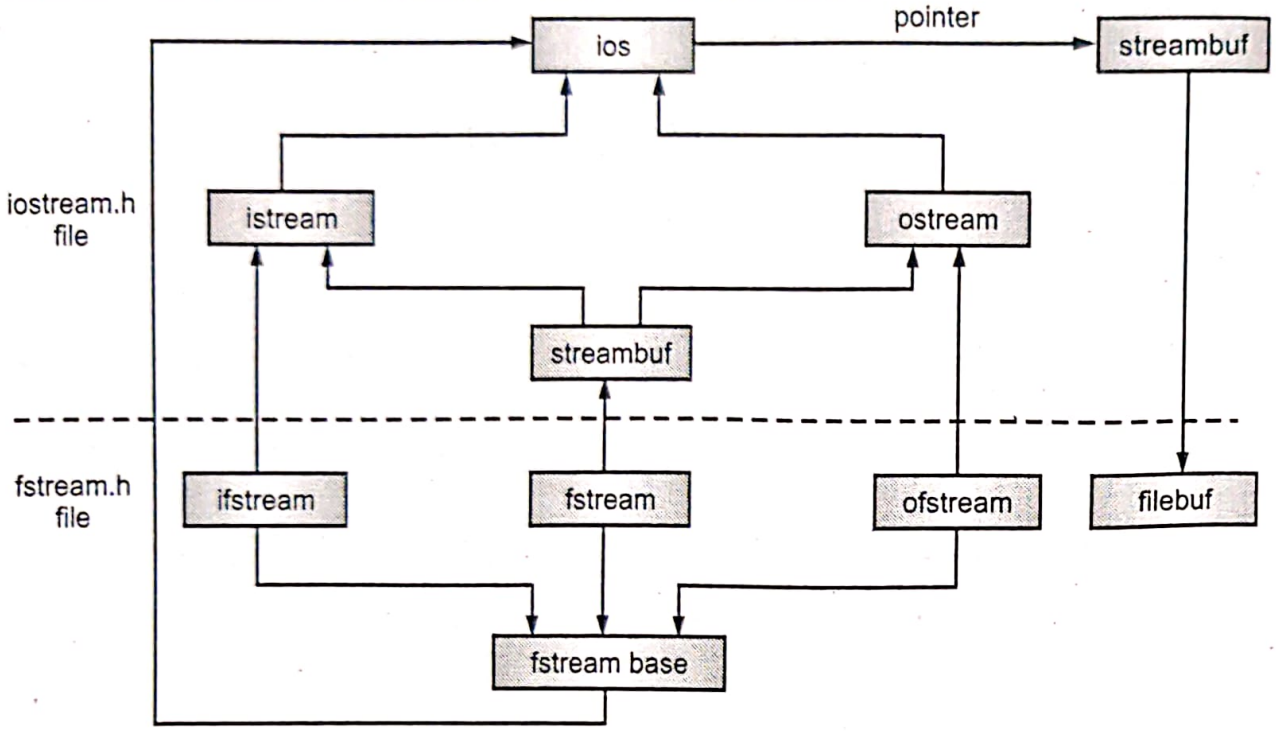

Fig. 8.2: Hierarchy of File Stream Classes

- The classes ifstream, ofstream, and fstream are designed exclusively to manage the disk files and their declaration exists in the header file fstream.h. To use these classes, include the following statement is used in the program,

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

- C++ views each file as a sequence of bytes. Each file ends either with an end of file marker or at a specific byte number. For file processing iostream.h and fstream.h must be included. fstream is used for read (ifstream) and write (ofstream).
- The functionalities of istream, ostream and iostream are inherited to ifstream, ofstream and fstream respectively. Details of file stream classes are given in the following table.

Table 8.1: Shows the details of file stream classes

Class	Contents
`filebuf`	Its purpose is to set the file buffers to read and write. Contains close() and open() as its members.
`fstreambase`	It is the base class of fstream, ifstream and ofstream classes. Contains close() and open() functions.
`ifstream`	It provides input operations. Contains open() with default input mode. Inherits the functions get(), getline(), read(), seekg() and tellg() from istream.

contd. ...

`ofstream`	It provides output operations. Contains open() with default output mode. Inherits put(), seekp(), tellp() and write() from ostream.
`fstream`	It allows simultaneous input and output operations. Contains open() with default input mode. Inherits all the functions from istream and ostream classes through iostream.

Methods for File Stream Classes:

- To associate a file with a stream, we can provide the file name when initializing a file stream object or we can first create a file stream object, then use the **open()** method to associate the stream with a file.
- The **close()** method terminates the connection between a stream and a file. The class constructors and the open() method take an optional second argument that provides the file mode.
- The file mode determines such things as whether the file is to be read and/or written to whether opening a file for writing truncates it or not, whether attempting to open a non-existing file is an error or not and whether to use the binary or text mode.
- The I/O class library provides a variety of useful methods. The istream class defines versions of the **extraction operator (>>)** that recognize all the basic C++ types and that convert character input to those types.
- The **get()** family of methods and the getline() method provide further support for single-character input and for string input.
- Similarly, the ostream class defines versions of the **insertion operator (<<)** that recognize all the basic C++ types and that convert them to suitable character output. The **put()** method provides further support for single-character output.
- A text file stores all information in character form. For example, numeric values are converted to character representations. The usual insertion and extraction operators, along with **get()** and **getline()**, support this mode.
- A binary file stores all information using the same binary representation the computer uses internally. Binary files store data, particularly floating point values, more accurately and compactly than text files, but they are less portable. The **read()** and **write()** methods support binary input and output.
- The **seekg()** and **seekp()** functions provide random access for files. These class methods let us to position a file pointer relative to the beginning of a file, relative to the end or relative to the current position. The **tellg()** and **tellp()** methods report the current file position.

8.3 FILE OPERATIONS

- There are various operations available on data files:
  1. Opening and closing files.
  2. Reading and writing files.
  3. Detecting End of file.
  4. Updating of file.

8.3.1 Opening a File [S - 18]

- Generally, if we are using files in the program, we must think about the name of file, the structure of file and opening mode of file.
- The file name is a string with at the most *eight* characters.
- It can have optional three characters for extension and both i.e. file name and extension are separated by period (•).
- To open a file:
  1. File stream is created and then linked to the file, which has two part: primary and extension.

     For example: `x.cpp` – Here x, is a filename and .cpp is extension.
  2. A filestream is defined by using classes ifstream, ofstream and fstream which are present in header file fstream.h. These classes are used depending on operations read or write.
  3. When a file is opened for write only, a new file is created if there is no file exist of that name.
- When we want to write certain text or information, the file should be created and opened, and even if we want to read the contents from the file, the file should be opened.
- In C++, a file can be opened in two ways:
  1. Opening file using constructor.
  2. Opening file using open( ) member function.

8.3.1.1 Opening File using Constructor

- In order to perform file operations, the file should be open. As we know, constructor is used to initialize an object while it is being created.
- Here, a constructor is used to initialize the file name which is used with the file stream object.
- The creation of file using file stream object involves the following steps:
  1. **Create a file stream object using stream classes as follows:**

```
ifstream ob1 ;        //input
ofstream ob2;         //output
fstream ob;           //input and output
```

  2. **File stream is identified by a file name:**

     For example, `ifstream ob1("stud.dat");`
- This statement declares **ob1** as an **ifstream** object and attaches it to the file **stud.dat** for reading (input) as shown in Fig. 8.3.

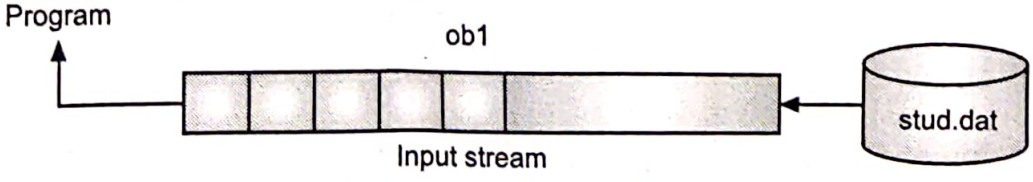

Fig. 8.3: Input Stream Object

- The statement `ofstream ob2("stud.dat");` opens a file in output mode i.e. in write mode. After opening a file, it is attached to output stream ob2 as shown in Fig. 8.4.

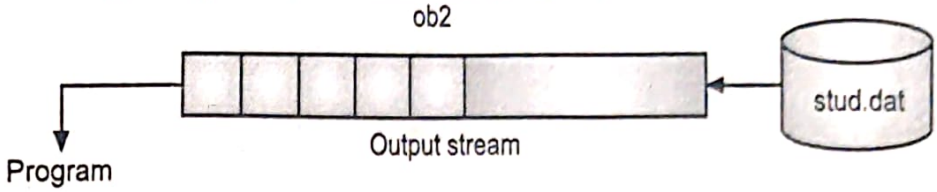

Fig. 8.4: Output Stream Object

Program 8.1: Program for open file in output mode.

```
#include<fstream.h>
#include<iostream.h>
int main()
{
    ofstream ob1("stud.dat");
    ifstream ob2 ("stud.dat");
    char str[80];
    //writing string to the file
    ob1<<"This is student database";
    //Read string from the file
    while (ob2)
    {
        ob2.getline(str, 80);      //read string using ob2
        cout<<str;                 //write on console
    }
    return 0;
}
```

Output:

```
This is student database
```

- Here ob1, ob2 are objects. Two different streams are working on the same file. This is shown in Fig. 8.5 where reading and writing are the operations performed on **stud.dat**.

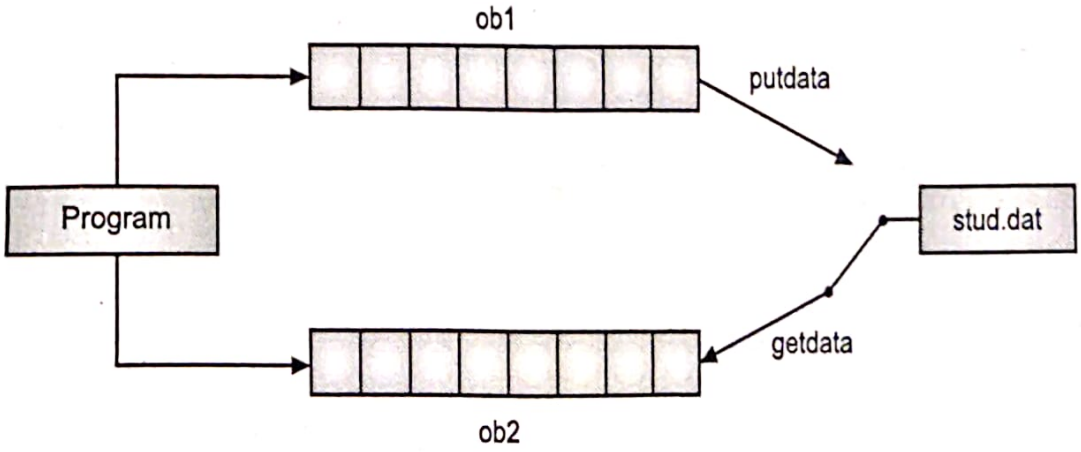

Fig. 8.5: Two file streams working on stud.dat

- When a stream object expires, the connection with a file is closed automatically i.e. when the program terminates, compiler automatically closes a file. Here, to print the message into file we use the statement.

```
ob1 << "This is student database";
```

8.3.1.2 Open File Explicitly using Open( )

- The file is open explicitly using open() function instead of constructor.
- **The syntax for opening a file is:**

```
stream_class   stream_object;
stream_object.open ("filename");
```

For example:

```
ofstream ofile;                 //create stream for output
ofile.open("stud");             //connect stream to stud
-------------
-------------
ofile.close();                  //close the file stud
```

- The stream is connected to only one file at a time. Hence, using same stream object we cannot open many files simultaneously. If we want to open two files, think first file should be close before we open the second file. If we want to open many files same time then we can use different stream_objects to open files.

For example:

1.
```
ofstream ofile;
ofile.open ("Teacher");
.........
.........
ofile.close();
ofile.open("stud");
.........
.........
ofile.close();
```

2.
```
ofstream ofile, ofout;
ofile.open("Teacher");
ofout.open("stud");
.........
.........
```

Program 8.2: Program for opening two files with same stream and storing the contents in it.

```
#include<iostream.h>
#include<fstream.h>
int main()
{
   ofstream outfile;
```

```
        outfile.open("Empname");
        outfile<<"Ganesh\n";
        outfile<<"Mahesh\n";
        outfile<<"Harish\n";
        outfile<<"Dinesh\n";
        outfile.close();                    //closing the file
        outfile.open("ACC");                //opening the file
        outfile<<"001";
        outfile<<"022";
        outfile<<"333";
        outfile<<"444";
        outfile.close();
        //Reading from a file
        char line[20];
        ifstream infile;                    //create input stream
        infile.open("Empname");
        cout<<"Name of Employees:\n";
        while(infile)
        {
           infile>>line;                    //or infile.getline(line,20);
           cout<<line<<"\n";
        }
        infile.close();
        infile.open("ACC");
        cout<<"\n the account numbers:\n";
        while(infile)
        {
           infile>>line;
           cout<<line<<"\n";
        }
        infile.close();
    }
```

Output:

```
Name of Employees
Ganesh
Mahesh
Harish
Dinesh
The account numbers
001
022
333
444
```

More about open( ): File Modes:

- C++ provides a mechanism of opening a file in different modes where second argument is explicitly passed.
- The **syntax** is:

```
stream_object.open("filename", openingmode);
```

- Here, the open( ) function takes two arguments, the first argument is file name and second argument is used for the file mode or file mode parameter, which are shown in Table. 8.1. We know that, ios::in & ios::out are default values for opening of ifstream and ofstream.
- The file modes are shown in Table. 8.2.

Table 8.2: File Open Modes

Parameter	Meaning
ios::ate	Go to end-of-file on opening.
ios::app	Append to end-of-file or all writes occurs at end of file.
ios::in	open file for reading only.
ios::out	open file for writing only.
ios::binary	Binary file (open file is binary mode).
ios::trunc	Delete contents of the file if it exists.
ios::nocreate	open fails if file does not exist.
ios::noreplace	open files if the file already exists.

- The various file modes in above table are explained below:
  1. **ios::ate:** It is used to sets a pointer to the end-of-file at opening. It allows to add or modify the existing data.
  2. **ios::app:** It is used to sets a pointer to the end-of-file similar to ate, but it add the data to the end of file.
  3. **ios::in:** This mode should be specified for input files.
  4. **ios::out:** This mode should be specified for output files.
     For the fstream class, mode should be explicitly specified.
     For example:
     ```
     fstream f;
     f.open("Emp", ios::in | ios::out);
     ```
  5. **ios::binary:** It is the most used mode for large applications and databases as compared to text file reading and writing.
  6. **ios::trunc:** It is a default mode if the file is opened for output.
  7. **ios::nocreate:** It is relevant with direct access files.
  8. **ios::noreplace:** This mode not allow us to open a file with the same name hence avoid destroying the previous file.
  9. We can separate more than one modes using bitwise OR or | symbol.
     For example, `out.open("Employee", ios::in | ios::app)`

Program 8.3: Program for ios::trunc file open mode.

```
#include<fstream.h>
#include<iomanip.h>
int main()
{
    float a = 122.25, b = 27, c = 33.5;
    ofstream fout;
    fout.open("num.data", ios::trunc);
    fout<<setw(6)<<a<<endl;
    fout<<setw(6)<<b<<endl;
    fout<<setw(6)<<c<<endl;
}
```

Output:

```
122.25
27.00
33.50
```

8.3.2 Closing a File

- To close a file we should use the input and output stream object name.
- For example,

```
ob1.close();
ob2.close();
```

- Here, the member function close() is used to close a file which has been open either for reading or writing or both purposes. The close() function is not contain any arguments. It is called explicitly as above. It also called automatically by the destructor function.
- Program 8.4 uses a single file for both writing and reading the data. First it takes data from the keyboard and writes it to the file. After writing is completed, the file is closed. The program again opens the same file, reads the information already written to it and displays the same on the screen.

Program 8.4: Program uses single file for both writing and reading data.

```
#include<iostream.h>
#include<fstream.h>
int main()
{
    ofstream ofile ("Employee");
    cout<<"Enter emp.name";
    char name[20];
    cin>>name;
    ofile <<name<<"\n";  //write to file employee
```

```
    cout<<"Enter salary of emp";
    double salary;
    cin>>salary;
    ofile<<salary<<"\n";
    ofile.close();
    ifstream ifile ("Employee");
    ifile>>name;
    ifile>>salary;
    cout<<"employee name:"<<name<<"\n";
    cout<<"employee salary:"<<salary<<"\n";
    ifile.close();
}
```

Output:

```
Enter emp name
Mahesh
Enter salary of emp
26500
employee name: Mahesh
employee salary:26500
Press any key to continue . . .
```

- When a file is opened for write only, a new file is created if there is no file exist of that name. If a file is exists already, then its contents are deleted.

8.3.3 Detecting the End-of-File

- When file is read, it is necessary to detect the end-of-file.
- The eof( ) function is used to check whether, file pointer is reached at end of file or not.
- It is member function of **ios** class. If eof is detected, then function return a non-zero value otherwise returns zero.
- **Syntax:**

```
ifstream in1;
in1.open("Hindiname");
while (! in1.eof())
{
    ---------------
}
```

- We can also write,

```
if(in1.eof()!=0) {exit (1);}
```

- Here, if end-of-file is detected then function returns non-zero value. This is illustrating in program 8.5.

Program 8.5: Program for eof().

```
#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>
int main()
{
    char line[20];
    ifstream ifile1, ifile2;
    ifile1.open("Employee");
    ifile2.open("ACC");
    for(int i=1;i<=5;i++)
    {
       if(ifile1.eof()!=0)
       {
        cout<<"Exit from Employee";
        exit(1);
       }
       ifile1>>line;
       cout<<"The account no. of employee"<<line<<"is";
       if(ifile2.eof()!=0)
       {
        cout<<"exit from ACC";
        exit(1);
       }
       ifile2>>line;
       cout<<line<<"\n";
    }
}
```

Output:

```
The account no. of employee Ganesh is 001
The account no. of employee Mahesh is 022
The account no. of employee Harish is 333
The account no. of employee Dinesh is 444
```

8.3.4 Updating a File

- We can add a new item, deleting an item, display an item, modifying (updating) an item in a program.

Program 8.6: Program illustrate the concept of updating files.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
   int k=0;
   string line;
   string find;
   char name[25];
   int id=0;
   float gpa=0;
   ofstream myfile;
   myfile.open("data.txt");
   while(k!=3){
   cout<<"press 1 for adding data"<<endl;
   cout<<"press 2 for update "<<endl;
   cin>>k;
   if(k==1)
   {
      cout<<"enter ID "<<endl;
      cin>>id;
      cout<<"enter Name"<<endl;
      cin>>name;
      cout<<"enter GPA "<<endl;
      cin>>gpa;
      myfile<<name<<endl;
      myfile<<id<<endl;
      myfile<<gpa<<endl<<endl<<endl;
   }
   if(k==2)
   {
      cout<<"name u want to update "<<endl;
      cin>>find;
      ifstream file;
      file.open("data.txt");
      while (!file.eof() && line!=find)
      {
      getline(file,line);
      }
```

```
        cout<<"enter ID "<<endl;
        cin>>id;
        cout<<"enter Name"<<endl;
        cin>>name;
        cout<<"enter GPA "<<endl;
        cin>>gpa;
        myfile<<name<<endl;
        myfile<<id<<endl;
        myfile<<gpa<<endl<<endl<<endl;
    }
    if(k==3){
    myfile.close();
    }
    }
    return 0;
}
```

8.4 FILE POINTERS AND THEIR MANIPULATORS

- To understand the working of files, (binary and text) we have to study 'file pointer'. When we open any file, the operating system maintains a pointer for that file.
- There are two pointers associated with each file, called file pointers. These are get pointer (input pointer) and put pointer (output pointer). These pointers are move through a file when read and write operations are perform.
- The get pointer points to the current reading operation and put pointers points to current writing operations.
- When the file is opened in read only mode the input pointers is initalised to point to the beginning of the file. So we can read file from start which is shown in Fig. 8.6.

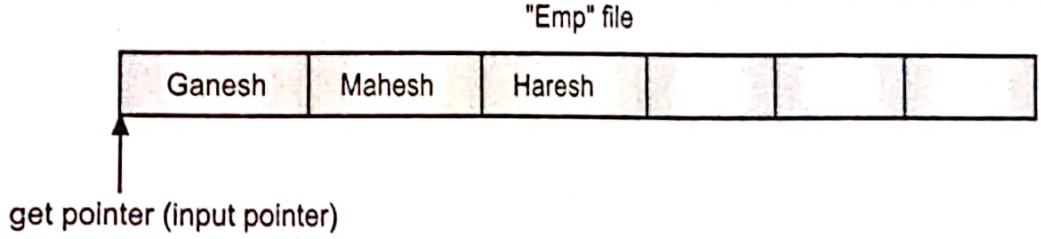

Fig. 8.6: Read Mode

- When the file is open in write only mode, then output pointer is set to beginning of the file for writing. If file is already exists then the existing contents are deleted as shown in Fig. 8.7.

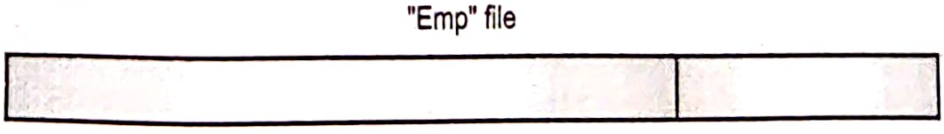

Fig. 8.7: Write Mode

- When the file is open is append mode, then the new data is added at the end of the file and output pointer are set to end of the file as shown in Fig. 8.8.

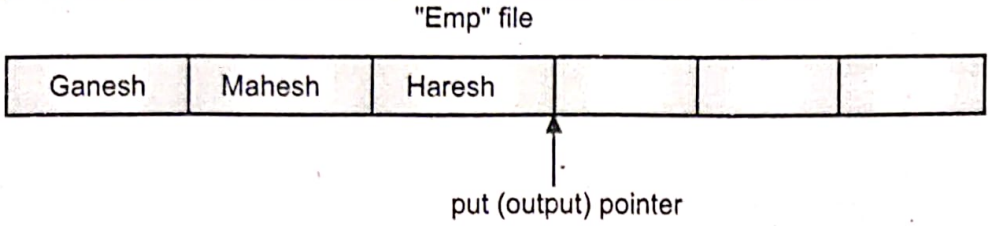

Fig. 8.8: Append Mode

- This is how the default actions take place with input and output pointers.

8.4.1 Functions for Manipulation of File Pointers [W-22]

1. **seekg( ):**
   - This is member function of ifstream. It moves get pointer (input) to a specified location.
   - **Syntax:** `seekg (position);`
   - **For example:** `in.seekg(20);` Where `in` is ifstream object
   - It moves the file pointer to byte number 20. i.e. the pointer points to the 21st byte in the file since first byte is zero.
2. **seekp( ):**
   - It is member function of ofstream.
   - It moves put pointer to a specified location.
3. **tellg( ):** [W-18; S-23]
   - It is member function of ifstream. It gives current position of a get pointer.
   - **For example:** `long n = in.tellg( );`
     Here, variable n stores the position of get pointer.
4. **tellp( ):** [W-18; S-23]
   - It is member function of ofstream. It gives the current position of put pointer.
   - For Example:

```
out.open ("stud", ios::app);
long p = out.tellp();
```

   - Here, the output pointer is moved to the end of the file "stud" and the value of `P` gives the number of types in the file. We also specify the offset with `seekg()` and `seekp()` functions.
   - **Syntax:**

```
seekg (offset, pointer direction);
seekp (offset, pointer direction);
```

- Where, offset is the number of bytes the file pointer is to be moved from the location specified by pointer_direction. The pointer_direction is one of the following defined in the ios class:

Table 8.3

ios::beg	Seek from beginning of the file.
ios::cur	Seek from current location.
ios::end	Seek from end of file.

- For example, [S - 18]

```
(i)   in.seekg(o, ios::beg);      //Go to start
(ii)  in.seekg(m, ios::beg);      //move to (m+1)th byte in the file
(iii) in.seekg(-m, ios::endl);    //go backward by m byte from the end
(iv)  in.seekpg(m, ios::cur);     //go forward m byte from current position.
(v)   in.seekg(-10, ios::end);    //moves the get pointer 10 bytes backward
                                  from the end of the file.
```

Program 8.7: Functions for manipulation of file pointer.

```
#include<fstream.h>
#include<iostream.h>
int main()
{
    ifstream infile;
    infile.open("stud",ios::in "ios::binary);
    if(!infile)
    {
       cerr<<"Error opening"<<endl;
       exit(0);
    infile.seekg (o, ios::end);
    cout<<"File size ="<<infile.tellg();
    infile.close();
    return 0;
}
```

Output:

```
File size = 371
```

- Here, keyword "cerr" is used for displaying error.

8.5 FILE UPDATING WITH RANDOM ACCESS

- Random access means moving the file pointer directly to any location in the file instead of moving it sequentially.
- The random access technique is often used with the database files.

- Fig. 8.9 shows the classification of files.

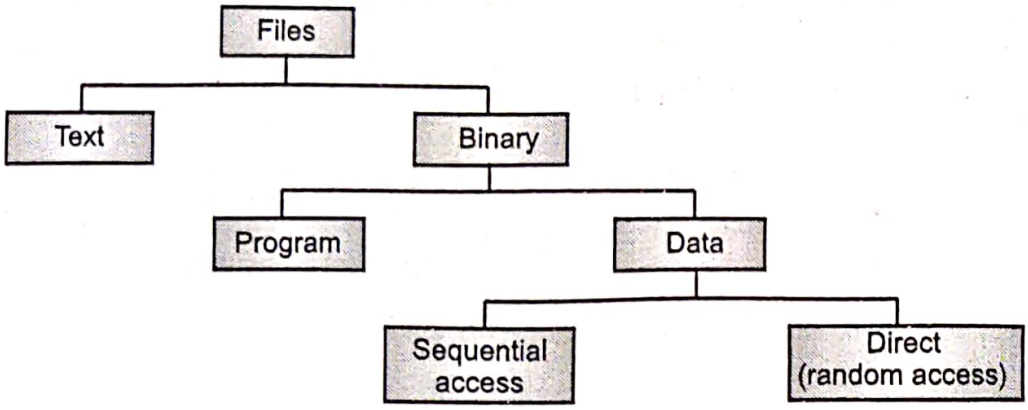

Fig. 8.9: Classification of Files

- In sequential access, we must open a file for reading and use file pointer to access records one by one sequentially.
- Binary data files can be accessed in direct or random manner. These files should be opened in binary mode.
- Random access performs the operations on file such as adding a new item, displaying the contents of a file, deleting an existing item, modifying an existing item, read and write.
- These requires that a file pointer to be move to an absolute position as specified by the parameter move.
- This can performed by using seekg( ), seekp( ), tellg( ), tellp( ), read( ) and write( ) functions. We use the following formulae in program.
- **Formula:**

1. To find size of object (*l*),

```
int l  = size of (object);
```

2. To find the location at which m^{th} object is stored,

```
int location = m * sizeof (object);
```

or `int location = m * l;`

3. To find total number of objects in file,

```
int n = filesize / sizeof (object);
```

Program 8.8: Program for random access approach.

```
//Basic File handling with add, modify, display operations
#include<iostream.h>
#include<fstream.h>
//using namespace std;
class student
{
   int roll;
   char name[20];
   public:
```

```
void getdata()
{
    cout<<endl<<"Enter Roll no:";
    cin>>roll;
    cout<<end1<<"Enter Name:";
    cin>>name;
}
void putdata()
{
    cout<<end1<<"Roll no: "<<roll;
    cout<<end1<<"Name: "<<Name;
};
int main()
{
   ifstream f;
   int n,i;
   char ch;
   student s[5],s1[5];
   clrscr();
   f.open("temp.txt",ios::out   ios::in);
   cout<<"writing data into file"
   for (i=0;i<2;i++)
   {
      s[i].getdata();
      f.write((char*)&s[i],sizeof(s[i]));
   }
   f.seekg(0);
   cout<<"reading from file & display on console"
   for(i=0;i<2;i++)
   {
      f.read((char*)&s[i],sizeof(s1[i]));
      s1[i].putdata();
      cout<<endl<<endl;
   }
   f.close();
   cout<<endl<<endl<<"Do you wish to update the file?(Y/N):";
   cin>>ch;
```

```
        cout<<"ADDING A RECORD"
        if(ch=='Y'||  ch=='y')
        {
           f.open("temp.txt",ios::app || ios::out);
           s[i].getdata();
           f.write((char*) &s[i],sizeof(s[i]));
           f.close();
           clrscr();
           f.open("temp.txt",ios::in);
           f.seekg (0, ios::beg);
           cout<<"Updated file is::"<<endl;
           for(i=0;i<3;i++)
           {
              f.read ((char*)&s1[i],sizeof(s1[i]));
              s1[i].putdata();
              cout<<endl;
           }
           cout<<"finding filesize & no. of objects"
           int l = f.tellg();
           int m = l/size of (s1[pi]);
           cout<<"no.of objects="<<m<<endl;
           cout<<"file size = "<<l<<"bytes"<<endl;
           f.close();
        }
        else
         f.close():
        return 0;
     }
```

Output:

```
Writing data into file
   200 Riya  //file contents
   201 Rucha
reading from file and display on console
   200 Riya
   201 Rucha
Do you wish to update the file? (Y/N): Y
ADDING A RECORD
   202 Deepa
```

```
updated file is:
    200 Riya
    201 Rucha
    202 Deepa
finding file size & no. of objects
no. of objects = 3
file size = 9 bytes
```

8.6 ERROR HANDLING DURING FILE OPERATIONS [W-18, S-19]

Errors Handling:

- When we use files different errors occurs such as:
  1. A file to be opened does not exist.
  2. Invalid filename.
  3. A filename used for a new file may already exist.
  4. Insufficient disc space.
- Hence, once the file is opened it has to be closed and reopen is possible using same stream.
- We can use following functions for error handling during file operation:
  - clearerr function
  - feof function
  - ferror function
  - perror function

1. The clearerr function:

- The clearerr function clears the end-of-file and error indicators for the stream pointed to by stream.

```
#include <stdio.h>
void clearerr(FILE *stream);
```

2. The feof function:

- The feof function tests the end-of-file indicator for the stream pointed to by stream and returns nonzero if and only if the end-of-file indicator is set for stream, otherwise it returns zero.

```
#include <stdio.h>
int feof(FILE *stream);
```

3. The ferror function

- The ferror function tests the error indicator for the stream pointed to by stream and returns nonzero if and only if the error indicator is set for stream, otherwise it returns zero.

```
#include <stdio.h>
int ferror(FILE *stream);
```

4. The perror function

- The perror function maps the error number in the integer expression errno to an error message. It writes a sequence of characters to the standard error stream thus: first, if s is not a null pointer and the character pointed to by s is not the null character, the string pointed to by s followed by a colon (:) and a space; then an appropriate error message string followed by a new-line character. The contents of the error message are the same as those returned by the strerror function with the argument errno, which are implementation defined.

```
#include <stdio.h>
void perror(const char *s);
```

8.7 COMMAND LINE ARGUMENTS

- File names may be supplied as argument to the main() at the time of invoking the program, these argument are known as Command Line Argument.
- These arguments are called as command line argument because they are passed from the command line during run time.
- The main() function look like this:

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

- In above the function main( ) can have two arguments, named argc (Argument counter) and argv (Argument vector), where argv is an array of pointers to strings and argc is an integer whose value is equal to the number of strings to which argv points.
- When the program is executed, the strings on the command line are passed to main( ).
- For example, `filecpy source.c destination.c`
- Here, argc contain:

  argv[0] → base address of string filecpy.
  argv[1] → base address of string "source.c".
  argc[2] → base address of string "destination.c".

Program 8.9: Program to copy file using command line argument.

```
#include<iostream.h>
#include<fstream.h>
int main (int argc, char * argv[])
{
    if (argc ! = 3)
    {
        cout<<"Bad command or file name";
        exit(1);
    }
    char ch;
    ifstream fin;
    fin.open (argc[1]);
```

```
    if(fin.fail())
    {
        cout<<"could not open";
        exit(1);
    }
    ofstream fout;
    fout.open (argv[2]);
    if(fout.fail())
    {
        cout<<"could not open destination";
        exit(1);
    }
    while(fin)
    {
        fin.get(ch);
        fout.put(ch);
    }
return 0;
}
```

Output:

```
This is file copy program        //source.c
This is file copy program        //destination.c
```

Program 8.10: Counting the number of vowels in the text file. [S-18]

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
void main()
{
    int count;
    ifstream in;
    in.open ("xyz.txt")
    ofstream out;
    out.open ("x.text")
    while (!in.eof())
    {
        char ch = (char) in.get();
        if (ch=='a' ||  ch=='e' || ch=='i'||  ch='=o' ||  ch=='u')
        count ++;
    }
    cout<<"no. of vowels ="<<count;
    in.close()'
}
```

Output:

```
This is program to                  //input file
Count number of vowels
No. of vowels  =  12                //on screen
```

Program 8.11: Write a program to count number of characters, lines and words in a file. **[S-19, 22]**

```
#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>
void main()
{
    char ch;
    int noc = 0; nol = 0; now = 0;
    ifstream myfile;
    myfile.open("countf.dat");
    while(1)
    {
        myfile.get(ch);
        if(ch==eof)
        break;
        noc++;
        if((ch=='\b') || (ch=='\t'))
        now++;
        if(ch=='\n')
        nol++;
    }
    cout<<"no. of characters = "<<noc<<endl;
    cout<<"no. of words = "<<now<<endl;
    cout<<"no. of lines = "<<nol<<endl;
}
```

Output:

```
This is C++ program //input file of counting characters words and lines
no. of characters    =  36
no. of words         =  9
no. of lines         =  2
```

Program 8.12: A program to perform copy from one text file to another text file.

[W-18, 22; S-19, 23]

```
#include<fstream.h>
void main()
{
    char source[50]; desti[50];
    char ch;
    cout<<"enter source file:"<<endl;
    cin>>source;
    cout<<"Enter destination file:"<<endl;
    cin>>desti;
    ifstream in(source);
    if(!in)
    {
        cerr<<"error opening file";
        exit (0);
    }
    ofstream out (disti);
    while (in)
    {
        in.get(ch);
        out.put(ch);
    }
}
```

Output:

```
This is file copy program        //input file
This is file copy program        //output file
```

Summary

- A file is a collection of related or associated data on a particular area of the storage device. Text files and binary files are the two different types of files.
- A data of a file is stored in the form of readable and printable characters then the file is known as text file. A file contains non-readable characters in binary code then the file is called binary file.
- The data in a text file is stored in the form of characters, i.e. in ASCII code. In binary files the data is stored in terms of a sequence of bytes, i.e., in 0's and 1's.
- File handling concept in C++ language is used for store a data permanently in computer. Using file handling we can store the data in secondary memory (hard disk).
- A stream is general name given to a flow of data.
- The stream that supplies data to the program is known as the input stream and the one that receives data from the program is known as the output stream. The input stream reads data from the file and the output stream writes data to the file. The third type of the stream takes input as well as gives output to the file.

- The C++ I/O system contains classes such as ifstream, ofstream and fstream to deal with file handling. These classes are derived from fstream base class and a declared in a header file iostream.
- The fstream class inherits all the functions from istream and ostream classes through iostream class defined inside iostream.h file. Also fstream inherits from the ios class, which is also the base class for istream and ostream classes.
- The ifstream class is derived from the istream class. Therefore, all the functions of ifstream class can operate on fstream class.
- A file can be opened in two ways by using the constructor function of a class and using the member function open() of the class.
- The function open() does the following:
  - It prepares the file for input or output operation.
  - It places the file pointer at the appropriate location in the file depending upon the mode in which file has been opened.
- The close() function closes all the opened stream files.
- The purpose of the close() function is:
  - It disconnects the stream with the file on the storage device.
  - No input or output operation can be performed on a file after it has been closed.
- The eof() function is used to check whether the end of a file character is reached or not. When the end of the file is reached, it returns non-zero value, otherwise it returns a zero.
- File names may be supplied as arguments to the main() function at the time of invoking the program. These arguments are known as command-line arguments.
- The file stream classes support some predefined functions that navigate the position of the file pointers. The relevant functions are seekg(), seekp(), tellg() and tellp().

Check Your Understanding

1. The stream that supplies data to a program is called ..... stream.
   (a) input (b) output
   (c) I/O (d) none of these
2. The stream that receives data from the program is called ..... stream.
   (a) input (b) output
   (c) I/O (d) none of these
3. Open() function is used to open ..... files which use the same stream objects.
   (a) multiple (b) console
   (c) print (d) none of these
4. ..... function is used to reached end of file.
   (a) exit() (b) close()
   (c) eof() (d) both (a) & (b)

5. The ..... function closes all the opened stream files.
   (a) exit() (b) close()
   (c) eof() (d) both (a) & (c)
6. A stream may be connected to more than one file at a time.
   (a) True (b) False
7. The ios::ate mode allows us to write data anywhere in the file.
   (a) True (b) False

Answers

1. (a)	2. (b)	3. (a)	4. (c)	5. (b)	6. (a)	7. (a)

Practice Questions

Q.I Answer the following questions in short:

1. What is the purpose of input stream and output stream?
2. What are the types of files?
3. Differentiate between seekg( ) and seekp( ).
4. Enlist various file operations.
5. Differentiate between ios::ate and ios::app.

Q.II Answer the following questions:

1. With the help of diagram explain file operations.
2. Explain various classes available for file Operations.
3. Differentiate between opening a file with a Constructor function and opening a file with open( ) function.
4. What is a File Mode? Describe the various file mode options available.
5. Write a C++ program that reads a text file and creates another file that is identical except that every sequence of consecutive blank spaces is replaced by a single space.
6. Write a C++ program that will create a data file containing the list of telephone numbers and name. Use a class object to store each set of data.
7. Write a C++ menu-driven program that will access the file created in exercise 2 and implement the following tasks:
   (a) Find telephone number of a specified person.
   (b) Find name of a specified telephone number.
   (c) Update the telephone number.
8. Write a menu-driven program that will allow us to create a file with following fields or information:
   (a) book-name
   (b) code
   (c) author-name
   (d) price and implement the following tasks
   (e) add a record
   (f) modify a record
   (g) search a record.
9. Write a C++ program to count no. of vowels in a text file.
10. Write a C++ program to count no. of characters and word from text file.

Q.III Define the term:

1. File
2. Stream
3. Input stream
4. Output stream

Previous Exam Questions

April 2018

1. Write seekg( ) function to: **[2 M]**

(i) Move get pointer 10 bytes backward from end of file.

Ans. Refer to Section 8.4.1.

(ii) Move get pointer to start of file.

Ans. Refer to Section 8.4.1.

2. What is file in C++? Explain two methods of opening a file with syntax. **[4 M]**

Ans. Refer to Section 8.1, 8.3.1.

3. Write a C++ program to display number of vowels present in a given file. **[4 M]**

Ans. Refer to Program 8.10.

October 2018

1. Define the following: **[2 M]**

(i) tellg( )

Ans. Refer to Section 8.4.1.

(ii) tellp( )

Ans. Refer to Section 8.4.1.

2. Write a program to append contents of one file to another file. **[4 M]**

Ans. Refer to Program 8.12.

3. Explain functions for error handing during file operation. **[4 M]**

Ans. Refer to Section 8.6.

April 2019

1. Write a C++ program to read contents of a text file and count number of characters, words and lines in a file. **[4 M]**

Ans. Refer to Section 8.11.

2. Explain various errors handling functions used during file operations. **[4 M]**

Ans. Refer to Section 8.6.

3. Write a C++ program to copy the content of one file to another file. **[4 M]**

Ans. Refer to Section 8.12.

■■■