Chapter – 7 File Handling

Introduction- The idea of storing the information has laid down the concept of file handling. **<u>A file is a bunch of bytes stored on some storage devices like hard disk, floppy disk etc</u>**. Most of the application programs process large volume of data which is permanently stored in files. We can write program that can read data from file(s) and write data to file(s). Data transfer is generally done in two ways:

i. Reading data from Input device (memory) and writing into file(s).
ii. Reading data from file(s) and writing into or display on Output device(memory).

Need of file handling:

i. Fast response – There are applications where the required info. Is needed very fast. In such situation if the data is stored in a file(s) it can be utilized immediately.
ii. Ease of Use : There are different ways of utilizing the data by different users. Data can be shared by users if it is stored in the form of files.
iii. Constraint on performance: in real time systems, there are constraints on the performance of the system. The data must be processed very fast.
iv. Saving of repetitive typing of data – if the same data is needed again and again for processing we need not to type it repeatedly, if stored in files.
v. Correctness of data – the contents of a data file are verified and the errors, if any, can be removed.

Types of Data files – TEXT FILE and BINARY FILE

Text Files : It contains alphanumeric data input using a text editor program. ASCII file format is used, so it can be displayed on screen or manipulated easily (Less Data Security) . Data translation is required, resulting in slower speed.

Binary File : It allows us to write alphanumeric data to the disk file in less number of files as compared to text files. The data is stored in Binary (Machine) language, not readable without program code, data is secure as no easy manipulation is there. Speed is faster as data needs no translation.

<u>Basic File Operations on Text Files</u>

1. Creating an empty file – First time when a file is created with some valid filename it is empty therefore it contains only EOF marker and a location pointer pointing to it.
2. Opening a file – A file is opened for reading/writing or manipulation of data on it. If a file exists then only it can be opened , when a file is opened the location pointer points to the Beginning of file.
3. Closing a file – After the file operations done , the file should be closed. If we don’t close the file it gets automatically closed when the program using it comes to an end.
4. Writing text into file – Once a file is created , data elements can be stored to it permanently. The already existing contents are deleted if we try to write data to it next time, rather we can append data to it and keep the existing data.
5. Reading of text from an already existing text file (accessing sequentially)
6. Manipulation of text file from an already existing file – An existing file is opened first and then the manipulation is done in sequential order. for example – counting of words.
7. Detecting EOF – When the data from the file is read in sequential order, the location pointer will reach to the

end of file. After reaching at the EOF no attempt should be made to read data from the file.

8. Copying of one text file to other text file

Binary file operations

1. Creation of file – A binary file is always opened in binary mode for both reading or writing. Upon successful creation, the pointer is set to the beginning of the file.

2. Writing data into file – A binary file is opened in output mode for writing data in it. A binary file contains non readable characters and write( ) function is used to write the records into file.
3. Searching for required data from file – a binary file is opened in input mode for searching data. The file can be read sequentially one by one each record or randomly by going to that particular location.
4. Appending data to a file – appending means addition of new records to an already existing file.
5. Insertion of data in sorted file
6. Deletion of data
7. Modification/Updation of data

COMPONENTS OF C++ TO BE USED WITH FILE HANDLING

Header file – fstream.h, In C++ file input/output facilities are performed by a header file fstream.h, which exists in the C++ standard library. C++ provides specific classes for dealing with user defined streams. Every file in C++ is linked to a stream. A stream must be obtained before opening a file. The file fstream.h is inherited from iostream.h, thus includes all the classes included in iostream.h .

User defined streams – The three classes for file Input/Output are :

i. Ifstream – derived from istream and used for file input(reading). It inherits the functions get(), getline() and read() and functions supporting random access(seekg() and tellg() and >> operator. It also contains open(),close() and eof().
ii. ofstream - – derived from ostream and used for file output(writing). It inherits the functions put() and write() functions along with functions supporting random access (seekp() and tellp() from ostream class. It also contains open(),close() and eof().
iii. fstream – derived form iostream and used for both input and output. It inherits all the functions from istream and ostream classes through iostream.h.
iv. Filebuf – it sets the buffers to read and write and contains close() and open() member functions in it.

Text File Operations

Reading Operation

Reading a File Character by character (including space,'\n','\t')

void readfile()
{
Ifstream fin;
fin.open("Text1.Txt");
	
	1. Create input file object
	2. Open the file, input and text mode is default

if(!fin)
{   cout<<"error in opening file"; getch();
exit(1);   }
char ch;
while(!fin.eof())
{
fin.get(ch);
cout<<ch;
}
fin.close();
getch(); }
	mode
	3. Check for file opening
	
	4. Declare a character
	5. Reading the file until end of file encountered
	6. Read the character from file and the character is stored in the memory.
	7. Display the character on the screen.
	
	8. Close the file.

<u>Reading a text file character by character excluding space</u>. Step 1 to 5 from the previous question is same, only the file name to be opened can be changed. To read only character **>> operator is used i.e. fin>>ch will only read a single character.** Once the character is read from the file it gets stored in the variable , and can be used according to question. Some of the possibilities are given below: a. Uppercase character – if(isupper(ch)) b. Lowercase Character – if(islower(ch)) c. Digit – if(isdigit(ch)) d. Vowel/Consonant/any single character The question can be asked to count/display the desired character.	

<u>Reading a text file word by word which starts with uppercase letter</u>
void readword()
{
Ifstream fin;
fin.open("Text1.Txt");
if(!fin)
{   cout<<"error in opening file";
exit(1);   }
char w[20];
while(!fin.eof())
{
fin>>w;
if(isupper(w[0]))
cout<<w<<' ';
	Step 1 to 5 are same.
	
	6. read word from file.
	7, Display on the screen.
	Note :- Once the word is in memory desired operation can be done for ex. counting of words, counting of words starting with vowel/uppercase/lowercase/any specific character or ending with vowel/uppercase/lowercase/any specific character.
	<u>Never compare string(word) with == sign use strcmp/strcmpi function.</u>

}
fin.close();
getch(); }	

<u>Reading text file line by line and count the number of lines.</u>
 	
void readline()
{
Ifstream fin;
fin.open("Text1.Txt");
if(!fin)
{   cout<<"error in opening file";
exit(1);   }
char line[80];
while(!fin.eof())
{
fin.getline(line,80);
cout<<line<<'\n ';
}
fin.close();
getch(); }
	Step 1 to 5 are same.
	
	6. read a line from file.
	7, Display on the screen.
	Note :- Once the line comes in memory desired operation can be done for ex. counting of lines, counting of lines starting with vowel/uppercase/lowercase/any specific character or ending with vowel/uppercase/lowercase/any specific character.
	Never compare string(word) with == sign use strcmp/strcmpi function.

Writing in Text File

void writefile()
{
ofstream fout;
fout.open("Text1.Txt");
if(!fout)
{   cout<<"error in opening file";
exit(1);   }
char line[80];
cout<<"enter a line";
cin.getline(line,80);
fout<<line<<'\n';
fout.close();
getch(); }
	1. Create output file object using ofstream class.
	2. Open the file, output and text mode is default mode
	3. Check for file opening
	
	4. Declare a line/word/character
	5. Accept the variable.
	6. Write to file.(Similar to write to screen using cout)
	7. Close the file.

<u>Copying of text file to another file (only lines wihich start with uppercase characters)</u>
 	
void copyline()
{
Ifstream fin;
fin.open("Text1.Txt");//file to be read
	Read the file until eof() and read character/word/line from file , apply the condition if asked and write to another file.

if(!fin)
{ cout<<"error in opening file";
exit(1); }
ofstream fout;
fout.open("Text2.Txt");
if(!fout)
{ cout<<"error in opening file";
exit(1); }
 	
char line[80];
while(!fin.eof())
{
fin.getline(line,80);
if(isupper(line[0]))
fout<<line<<'\n ';
}
fin.close(); fout.close();
getch(); }	

Binary Files are also called fixed length files or packed files as all the fields in a binary file occupy fixed number of bytes.

For Opening files C++ provides mechanism for opening file in different modes:

Mode	Behavior
Ios::in	Opens the file for reading.
Ios::out	Open for writing(previous contents erased)
Ios::ate	Go to end of the file at the time of opening the file.
Ios::app	Appends records
Ios :: binary	File is opened in binary file.

Note: A file can be opened in more than one modes by making use of pipe sign as follows:
fstream file("ABC",ios::in|ios::binary|ios::out);

Using ordinary operator >> , data cannot be written to binary files. Write() function is used to write the records. The general syntax is

File_object.write((char*)&variable,sizeof(variable));

Where variable name is the name of that variable whose contents have to be written in the file and sizeof(variable) is the number of bytes that are to be written. (char*)&variable is explicit typecastingi.e. converting one data type to another datatype. The variable will explicitly converted into character stream and writes to the file specified.

If the function is a member function then in place of &variable “this” pointer is used. “this” is a special pointer that points to the current object i.e. thru which the member function is called. Object can be of any data type(Fundamental or derived)

Writing class object to binary File(assuming student class)

```
void writefl()
{
ofstream fout(“Stud.dat”,ios::binary|ios::out|ios::app);
If(!fout)
{
cout<<eror in opening file”;
exit(1);
}
Stud s; char choice=’y’;
do{
s.getdata();
fout.write((char*)&s,sizeof(s));
cout<<”do you want to add more records”;
cin>>choice;
}while(ch==’y’);
fout.close();
}
```

Steps:

1. Open the file in output and binary mode.(append mode for addition of records)
2. Create class Objects
3. Call the class function which accepts data from keyboard.
4. Write the records using wirte function.
5. Do steps 3,4 repeatedly until user wants to add records
6. Close the file.

Writing class object to binary File(assuming student class)and function is a member function

```
void student::writefl()
{
ofstream fout(“Stud.dat”,ios::binary|ios::out|ios::app);
If(!fout)
{
cout<<eror in opening file”;
exit(1);
}
Stud s; char choice=’y’;
do{
cout<<”Enter name”;
gets(name);
cout<<”Enter average”;
cin>>avg;
fout.write((char*)this,sizeof(s));
cout<<”do you want to add more records”;
cin>>choice;
}while(ch==’y’);
fout.close();
}
```

Steps:

1. Open the file in output and binary mode.(append mode for addition of records)
2. Create class Objects
3. Call the class function which accepts data from keyboard.
4. Write the records using wirte function.
5. Do steps 3,4 repeatedly until user wants to add records
6. Close the file.

Writing Structure object to binary File(assuming student is structure, containing name and avg as members)

```
void writefl()
{
ofstream fout(“Stud.dat”,ios::binary|ios::out|ios::app);
If(!fout)
{
cout<<eror in opening file”;
exit(1);
}
Stud s; char choice=’y’;
do{
cout<<”Enter name”;
gets(s.name);
cout<<”Enter average”;
cin>>s.avg;
fout.write((char*)&s,sizeof(s));
cout<<”do you want to add more records”;
cin>>choice;
}while(ch==’y’);
fout.close();
}
```

Steps:

1. Open the file in output and binary mode.(append mode for addition of records)
2. Create class Objects
3. Call the class function which accepts data from keyboard.
4. Write the records using wirte function.
5. Do steps 3,4 repeatedly until user wants to add records
6. Close the file.

Reading a Binary File

Sequential reading – Sequential reading means reading record one by one until end of file and all the records are displayed.

```
void readfl()
{
ifstream fin(“Stud.dat”,ios::binary|ios::in);
If(!fin)
{
cout<<eror in opening file”;
exit(1);
}
Stud s;
while(!fin.eof()) or while(fin)
{
fin.read((char*)&s,sizeof(Stud));
s.display();
getch();
```

Steps:

1. Open the file in input and binary mode.
2. Check for file opening, if error then exit
3. Create class Object.
4. Read the file until End of file
5. Call display function of class
6. Do steps 4,5 repeatedly until end of file reached.
7. Close the file.

}
fin.close();}	

Displaying selected records - This is condition based display , where the condition is checked after reading. If the reading is concerned with reading class objects then there will be a function in class as a public member which will return that particular value that is to be evaluated. If the structure instance is to be read from file then the normal comparison can be done after executing the file read statement. If the condition is true then call display function or display the record.

Display Records from file where average marks are greater than 60

```
void readfl()
{
ifstream fout("Stud.dat",ios::binary|ios::in);
If(!fin)
{
cout<<eror in opening file";    exit(1);  }
Stud s;
while(!fin.eof())
{   fin.read((char*)&s,sizeof(Stud));
if(s.retavg()>60)  // here the condition can be specified
s.display();   getch();  }
fin.close();       }
```

Random Access

Up till now we had been writing and reading the files in sequential manner but binary files, being fixed length files, provides liberty to perform read and write operations randomly. C++ provides functions for it. When you open a file operating system assigns two exclusive pointers to file object. In C++ these pointers are called get pointer(input) and put pointer(output) . These pointers provide you the facility to move to the desired place in the file to perform read, write operations. The get pointer specifies a location from where the current reading operation is initiated. The put pointer specifies a location where the current write operation will take place.

Functions for manipulation of file pointers

Function	**Class Members**	**Action Performed**	
seekg()	**Ifstream**	**Moves get pointer to a specific location.**	
seekp()	**Ofstream**	**Moves put pointer to a specific location.**	
tellg()	**Ifstream**	**Returns the current position of the get pointer**	

tellp()	**Ofstream**	**Returns the current position of the put pointer**	
The general syntax for seekg() and seekp() is : file_object.seekg(location,origin); file_object.seekp(location,origin); where location is the byte position, origin is the position from where the location is to be calculated.			Origin Value Seek From ios::beg seek from beginning of file ios::cur seek from current position ios::end seek from end of file f.seekg(-10,ios::end) – will move 10 bytes back from end position. f.seekg(0,ios::end) - directly reach to end of file

<u>Random record reading-</u> File pointer position is set according to the record number to be displayed. The file pointer directly can be set to the position using seekg().

Note :- when file is opened in input or output mode the file pointer is at the starting i.e. at 0 position. In append mode the file pointer is at the end of file, and the writing continues by adding the records.

```
void readrec()
{
ifstream fin(“stud.dat”,ios::binary|ios::in);
stud s;
int rec;
cout<<”enter the record you want to display”;
cin>>rec;
fin.seekg((rec-1)*sizeof(stud),ios::beg);  // by default ios::beg is default argument
fin.read((char*)&s,sizeof(stud));
s.display();
getch();
}
```

<u>Updation of record in existing file –</u> In this the record to be updated is asked from the user, then appropriate position of file pointer is obtained and pointer is set and write operation is performed.

Eg. – writing a program to update structure/class object to update.

Insertion in Sorted file The record to be inserted in a sorted file is accepted as a separate object. The file in which the record is to be inserted, is opened in input mode. The records with record key smaller than the record key to be inserted are copied to temporary file and then record to be inserted is copied, following with rest of the records if any. After that original file is removed using remove() and temporary file is renamed with the original name with rename().

Deletion of record from file – The logic of deleting record is:

1. Accept the record key for the record you want to delete.
2. Read the file sequentially , from which the record is to be deleted and copied down the records to temporary file except the record you want to delete (by comparing the record value)
3. Remove the original file and rename the temporary file with the original file name.

Sorting the records of an existing file - First the records in the file are counted . an array of objects is created dynamically / statically with the approx. index value as compared to number of records. And read the file using the following statement:

```
while(fin.read(char*)&obj[i++],sizeof(class)));
```

After reading the file , sort the array with any sorting technique(Bubble/Insertion/Selection)

Then write the sorted array in to the file opened in output mode.

Notes:-

1. Multiple File Handling- In this case the point to remember is :

a)Identify the file mode to be opened . b) One object can open only one file at a time.

2. Sizeof() operator gives the sizeof data type.

3. The data written to a file using write() can only be read accurately using read().

4. The files are closed using destructor of file object, if not closed using close(). It is must to close the file. If the file pointer is opening another file in the same program otherwise it is a good practice to use close().