Assignment No: E-23

Aim: To understand File organization concepts

Problem Statement: Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to maintain the data.

Input:
1. Students Record stored in Sequential File

Output:
1. Add student information in file.
2. Delete the student record from file.
3. Display information of particular student.
4. If record of student does not exist then display an appropriate message.

Theory:

File: A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

File Access Mechanisms: File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files –

- Sequential access
- Direct/Random access
- Indexed sequential access

Sequential file organization: In sequential file organization, records are arranged sequentially. A sequential file is a file whose records can be accessed on the order of their appearance in the file. The order in which the records are stored is determined by the order in which they are written when the file was prepared. This order does not charge. Records may be added at the end of file only. The records may be accessed in the order on which they were originally written into a file. A magnetic tape file, such as printer, can only have a sequential organization. A sequentially organized file may be stored on either a serial –access or direct access storage medium.

Advantages of sequential file organization: -
(i) Simple to implement
(ii) Requires very low software support
(iii) Efficiency of blocking is good
(iv) Blocking results in saving in terms of input-output time required handling a file.

(v) A substantial amount of storage space on the disk can be saved.

Disadvantages of sequential file organization: -

(i) Updates are not easily accommodated

(ii) Random access is not possible

(iii) All records must be structurally identical, if a new field has to be added, and then every record must be rewritten to provide space for the new field.

(iv)Continuous areas may not be possible because both the primary data file and the transaction file must be looked during merging.

Classes for file stream operation

ofstream: Stream class to write on files **ifstream**:

Stream class to read from files

fstream: Stream class to both read and write from/to files.

A) Opening A File

Opening File Using Constructor ofstream

fout(*"results"*); //output only ifstream

fin(*"data"*); //input only

Opening File Using Open()

Stream-object.open(*"filename"*, mode)

fstream *ofile*; *ofile*.open(*"data1"*);

ifstream *ifile*; *ifile*.open(*"data2"*);

File mode parameter	Meaning
ios::app	Append to end of file
ios::ate	go to end of file on opening
ios::binary	file open in binary mode
ios::in	open file for reading only

ios::out	open file for writing only
ios::nocreate	open fails if the file does not exist
ios::noreplace	open fails if the file already exist
ios::trunc	delete the contents of the file if it exist

All these flags can be combined using the bitwise operator OR (|). For example, if we want to open the file example.bin in binary mode to add data we could do it by the following call to member function open(): fstream *file*;

file.open (*"example.bin"*, ios::out | ios::app | ios::binary);

B) CLOSING FILE

fout.close();

fin.close();

C) INPUT AND OUTPUT OPERATION put() and get() function

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

example :*file*.get(ch); *file*.put(ch); **write() and read() function**

write() and read() functions write and read blocks of binary

data.

example: *file*.read((char *)&*obj*, sizeof(*obj*)); *file*.write((char *)&*obj*, sizeof(*obj*));

D) ERROR HANDLING FUNCTION

FUNCTION	RETURN VALUE AND MEANING
eof()	returns true (non zero) if end of file is encountered while reading; otherwise return false(zero)
fail()	return true when an input or output operation has failed
bad()	returns true if an invalid operation is attempted or any

	unrecoverable error has occurred.
good()	returns true if no error has occurred.

E) FILE POINTERS AND THEIR MANIPULATION

All i/o streams objects have, at least, one internal stream pointer:

ifstream, like istream, has a pointer known as the get pointer that points to the element to be read in the next input operation.

ofstream, like ostream, has a pointer known as the put pointer that points to the location where the next element has to be written.

Finally, fstream, inherits both, the get and the put pointers, from iostream (which is itself derived from both istream and ostream).

These internal stream pointers that point to the reading or writing locations within a stream can be manipulated using the following member functions:

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

The other prototype for these functions is:

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

The parameter offset represents 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

ios::end	end of the file	

```
class reservation
{
public:
        char name[10],
 sex[10], source[10],
 destination[10];
        int age;
};
```

```
class SQ
{
public:
        fstream tab, tab1;
        char tabname[10];
        reservation r;
        void insert (reservation);
        void display();
        void  search();
        void modify();
        void dele();
};
```

ALGORITHMS:

Algorithm Insert (r1)

// this algorithm is used to add a record "r1" into sequential file

1. {
2. tab.open(tabname,ios::app|ios::binary);
3. tab.write((char *)&r1, sizeof(r1));
4. tab.close();
5. Write(" Record Added Successfully");
6. }

Algorithm Display ()

//This algorithm is used to display records of sequential file

1. {
2. tab.open(tabname,ios::binary|ios::in|ios::nocreate);
3. while(1)
4. {

```
5.  tab.read((char *)&r, sizeof(r));
6.  if(tab.eof()) break;
7.  Write(r.name, r.sex, r.source, r.destination, r.age);
8.  }
9.  tab.close();
10. }
```

Algorithm Search ()

// This algorithm is used to search and display particular record from sequential file

```
1.  {
2.  Read name of record to be searched in Key
3.  tab.open (tabname, ios::binary|ios::in);
4.  while(1)
5.  {
6.  tab.read((char *)&r, sizeof(r));
7.  if(tab.eof())break;
8.  if(strcmp(r.name,key)==0)
9.  {
10. Write(r.name, r.sex, r.source, r.destination, r.age);
11. break;
12. }
13. }
14. if(tab.eof())
15. Write(Data NOt Found);
16. tab.close();
17. }
```

Algorithm Modify ()

// This algorithm is used to modify and display particular record from sequential file

```
1.  {
2.  Read name of record to be modified in Key

3.  tab.open (tabname, ios::binary|ios::in|ios::out);
4.  int i=0;

5.  while(1)
6.  {

7.  tab.read((char *)&r, sizeof(r));
8.  if(tab.eof())break;
```

```
9.      if(strcmp(r.name,key)==0)
10.     {
11.     Read( New Data);
12.     Read (r.name, r.sex, r.source, r.destination, r.age);
13.     tab.seekg(i*sizeof(r), ios::beg);
14.     tab.write((char *)&r, sizeof(r));
15.     break;
16.     }
17.     }
18.     if(tab.eof())
19.     Write(Data NOt Found);
20.     tab.close()
21.     }
```

Algorithm Dele ()

// This algorithm is used to Delete particular record from sequential file

```
1.              {
2.              tab.open(tabname,ios::in|ios::binary);
3.              tab1.open("temp.txt",ios::out|ios::binary);
4.              while(1)
5.              {
6.              tab.read((char*)&r,sizeof(r));
7.              if(tab.eof()) break;
8.              if(strcmp(r.name,key)!=0)
9.              {
10.             tab1.write((char*)&r,sizeof(r));
11.             }
12.             Else
13.             {
14.             flag=1;
15.             }
16.             }
17.             tab.close();
18.             tab1.close();
19.             remove(tabname);
20.             rename("temp.txt",tabname);
```

```
21.     if(flag==1)
22.     Write( " deleted successfully.");
23.     Else
24.     Write("not found." );
25.
```

Flowchart:}

Conclusion: