GROUP: F

ASSIGNMENT NO: 11

Title: Study of Sequential files

Objective: To Study Sequential file system.

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.

Outcomes:

To use effective and efficient data structures in solving various Computer Engineering domain problems.

Theory :

Many real life problems use large volume of data and in such cases we require to use some devices such as floppy disk or hard disk to store the data. The data in these devices is stored using the concept of **files .** A file is a collection of related data stored in a particular area on the disk. Programs can be designed to perform the read and write operations on these files.

A program involves either or both of following types of data communication.

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

The I/O system of C++ contains a set of classes that define the file handling methods.

These include **ifstream, ofstream** and **fstream .** These classes are derived from **fstream base** and from corresponding **stream** classes. These classes are declared in **fstream.h** header file. We must include this file in the program that uses file.

Details of file stream classes

Fstreambase	**Serves as a base for fstream, ifstream** and **ofstream** class. Contains **close( )** and **open( )**
Ifstream	Provides input operations. Contains **open( )** with default input mode.Inherits the

functions **get(), getline( ), read( ), seekg( ) and tellg( )** functions from **istream.**

Ofstream	Provides output operations. Contains **open( )** with default output mode. Inherits the functions **put( ), write( )seekp( ) and tellp( )** functions from **ostream.**
Fstream	Provides support for i/p and o/p operations.Contains **open( )** without default mode. Inherits all the functions from **istream** and **ostream** classes through **iostream.**
Filebuf	To set file buffers to read and write

A file can be opened in two ways:

- Using the constructor function of the class : This method is useful when we use only one file in the stream.
- Using the member function **open( )** of the class. : This method is used when we want to manage multiple files using one stream.

Opening Files Using Constructor

Filename is used to initialize the file stream object. There are 2 steps :

Create a file stream object to manage the stream using appropriate class
Initialize the file object with the desired filename.
Ex. Ofstream outfile("result");

Open : File Modes

The general form of the function **open( )** with two arguments is :

stream-object.open("filename", mode);
The second argument mode specifies the purpose for which the file is opened. The prototype of the class member functions contain default values for the second argument .

The default values are as :
ios :: in for ifstream functions open for reading only
ios:: out for ofstream functions open for writing only

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

Parameter	**Meaning**
ios :: app	Append to end-of –file.
ios::ate	Go to end-of-file on opening.
ios::in	Open file for reading only
ios::nocreate	Open fails if file does not exist
ios::noreplace	Open fails if file already exists
ios::out	Open file for writing only
ios::trunc	Delete the contents of the file if it exists
ios::binary	Binary file

File Pointers and Their Manipulations

Each file has two associated pointers known as file pointers. One of them is called 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 or writing . The input pointer is used for reading the contents of a given location and the output pointer is used for writing to a given location.

Each time an input or output operation takes place , the appropriate pointer is automatically adjusted

Default actions

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.

When we open a file in write-only mode, the existing contents are deleted and the output pointer is set at the beginning of the file. This enables to write to the file from the start

When 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 file.

The file stream classes support the functions to manage movements of the file pointers seekg( ) Moves get pointer to a specified location.

seekp( ) Moves put pointer to a specified location.

tellg( ) Gives the current position of the get pointer.

tellp( ) Gives the current position of the put pointer.

Seek functions seekg( ) and seekp( ) can also be used with two arguments:

```
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 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

The **seekg()** function moves associated file's 'get' pointer while the **seekp()** function moves the associated file's 'put' pointer.

Sequential Input and Output Operations

The file stream classes support a number of member functions for performing the input and output operations on files.

The first pair of functions , **put( )** and **get( )** are designed for handling a single character at a time

Second pair of functions, **write( )** and **read( )** are designed to write and read a blocks of binary data.

Put( ) and get( ) Functions

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

Write( ) and read( ) Functions

The functions write( ) and read( ) , unlike the functions put( ) and get( ) handle the data in binary form. The binary format is more accurate for storing the numbers as they are stored in the exact internal representation.

The binary input and output functions takes form as:

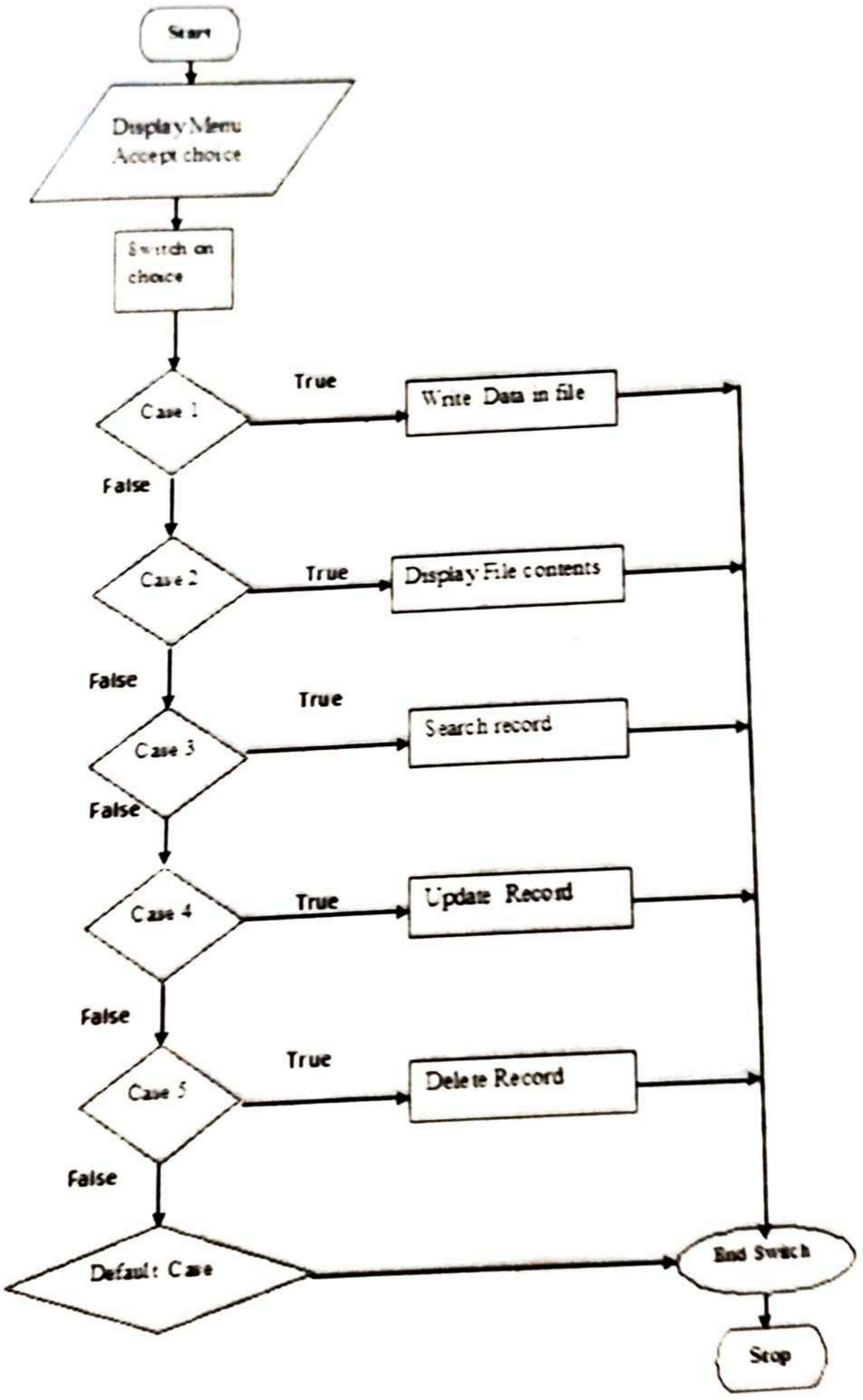
```
infile.read ((char *) &V, sizeof (V) );
outfile.write ((char *) &V, sizeof (V) );
```

The functions takes two arguments . The first is the address of the variable V and the second is the length of variable in bytes.

Algorithm :

1. Open the file
2. Write the student record in file.
3. Display all contents of filename
4. Delete the required student record from file and d display the contents
5. Search for specific student record and display respective record.

Flowchart:


Start
Display Menu
Accept choice
Switch on choice
Case 1
True
Write Data in file
False
Case 2
True
Display File contents
False
Case 3
True
Search record
False
Case 4
True
Update Record
False
Case 5
True
Delete Record
False
Default: Case
End Switch
Stop

Assignment No: 12

Title: Study of Use index sequential file

Objective: To understand the concept and basic of index sequential file and its use in Data structure

Problem Statement: Company maintains employee information as employee ID, name, designation and salary. Allow user to add, delete information of employee. Display information of particular employee. If employee does not exist an appropriate message is displayed. If it is, then the system displays the employee details. Use index sequential file to maintain the data

Outcome:
To implement the concept and basic of index sequential file and to perform basic operation as adding record, display all record, search record from index sequential file and its use in Data structure.

Theory :
Indexed sequential access file organization
•Indexed sequential access file combines both sequential file and direct access file organization.
•In indexed sequential access file, records are stored randomly on a direct access device such as magnetic disk by a primary key.
•This file have multiple keys. These keys can be alphanumeric in which the records are ordered is called primary key.
•The data can be access either sequentially or randomly using the index. The index is stored in a file and read into memory when the file is opened.

Primitive operations on Index Sequential files:
• Write (add, store): User provides a new key and record, IS file inserts the new record and key.
• Sequential Access (read next): IS file returns the next record (in key order)
• Random access (random read, fetch): User provides key, IS file returns the record or "not there"
• Rewrite (replace): User provides an existing key and a new record, IS file replaces existing record with new.
• Delete: User provides an existing key, IS file deletes existing record

Algorithm:
Step 1 - Include the required header files (iostream.h, conio.h, and windows.h for colors).
Step 2 - Create a class (employee) with the following members as public members. emp_number, emp_name, emp_salary, as data members. get_emp_details(), find_net_salary() and show_emp_details() as member functions.
Step 3 - Implement all the member functions with their respective code (Here, we have used scope resolution operator ::). Step 3 - Create a main() method.
Step 4 - Create an object (emp) of the above class inside the main() method.
Step 5 - Call the member functions get_emp_details() and show_emp_details().
Step 6 - return 0 to exit form the program execution.

Approach:
1. For storing the data of the employee, create a user define datatype which will store the information

regarding Employee. Below is the declaration of the data type:

```
2. struct employee
{ 3. string name;
4. long int Employee_id;
5. string designation;
6. int salary;
7. };
```

Building the Employee's table:
For building the employee table the idea is to use the array of the above struct datatype which will use to store the information regarding employee.
For storing information at index i the data is stored as:

```
struct employee emp[10];
emp[i].name = "GeeksforGeeks"
emp[i].code = "12345"
emp[i].designation = "Organisation"
emp[i].exp = 10 emp[i].age = 10
```

Deleting in the record: Since we are using array to store the data, therefore to delete the data at any index shift all the data at that index by 1 and delete the last data of the array by decreasing the size of array by 1

Searching in the record: For searching in the record based on any parameter, the idea is to traverse the data and if at any index the value parameters matches with the record stored, print all the information of that employee.

Conclusion: Index Sequential Files are implemented successfully for Student database system.