Progressive Education Society's
Modern College of Engineering
DEPARTMENT OF COMPUTER ENGINEERING

ONE DAY ONLINE SYLLABUS IMPLEMENTATION

FACULTY DEVELOPMENT PROGRAM

ON

DATA SRTUCTURES AND ALGORITHMS LAB

SE COMP, 2019 Course

Date – 27th January 2021

Ms. Pallavi V. Baviskar

Assistant Professor, Computer Engineering Department

PES MCOE Pune-5

210256: Data Structures and Algorithms Laboratory

Teaching Scheme Practical: 04 Hours/Week

Examination Scheme and Marks

Term Work: 25 Marks

Practical: 25 Marks

Companion Course :

210252: Data Structures and Algorithms (TH)

210255:Priciples of Programming Languages

- **Objectives:**

1.The local and physical organization of files.

2. To understand the concept of sequential files.

3. To know about File Operations

4. To understand the use of sequential files.

5. Sequential file handling methods.

6. Creating, reading, writing and deleting records from a variety of file structures.

7. Creating code to carry out the above operations.

- **Mapping of assignments:**

1. Analyse the algorithmic solutions for resource requirement and optimisation.
2. Use appropriate modern tools to understand and analyse the functionalise confine to the secondary storage.

Outcome

- **CO1: Understand** the ADT/libraries, hash tables and dictionary to design algorithms for a specific problem.
- **CO2:** Choose most appropriate data structures and **apply** algorithms for graphical solutions of the problems.
- **CO3: Apply** and **analyze** non linear data structures to solve real world complex problems.
- **CO4: Apply** and **analyze** algorithm design techniques for indexing, sorting, multi-way searching, file organization and compression.
- **CO5: Analyze** the efficiency of most appropriate data structure for creating efficient solutions for engineering design situations.

Group F

1. 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 student. 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 main the data.**

2. 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.**

Group F

3.Implementation of a direct access file -Insertion and deletion of a record from a **direct access file.**

4. Assume we have two input and two output tapes to perform the sorting. The internal memory can hold and sort m records at a time. Write a program in java for **external sorting**. Find out time complexity.

External Sort- Consequential processing and merging two lists, multiway merging- a k way merge algorithm

Problem Statement:

Department maintains a student information. The file contains roll number, name, and division and address. Allow user to add, delete information of student. Display information of particular student. 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 main the data.

Objective:

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

Outcome:

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

Software & Hardware Requirements:

64-bit Open source Linux or its derivative

Open Source C++ Programming tool like G++/GCC

Online mode: online GDB classroom, Google classroom

Theory Contents:

Types of File Organization-

1. Sequential access file organization
2. Indexed sequential access file organization
3. Direct access file organization

Theory Contents:

Sequential access file organization

•Storing and sorting in contiguous block within files on tape or disk is called as **sequential access file organization**.

•In sequential access file organization, all records are stored in a sequential order. The records are arranged in the ascending or descending order of a key field.

•Sequential file search starts from the beginning of the file and the records can be added at the end of the file.

•In sequential file, it is not possible to add a record in the middle of the file without rewriting the file.

Primitive Operations on Sequential files :

- **Open**—This opens the file and sets the file pointer to immediately before the first record
- **Read-next**—This returns the next record to the user. If no record is present, then EOF condition will be set.
- **Close**—This closes the file and terminates the access to the file.
- **Write-next**—File pointers are set to next of last record and write the record to the file.
- **EOF**—If EOF condition occurs, it returns true, otherwise it returns false.
- **Search**—Search for the record with a given key.
- **Update**—Current record is written at the same position with updated values.

Sample Input Set:

Record Type Name	Field Name	Data Type
Student	Roll No.	Integer
	Name	Character or String
	Division	Character or String
	Address	Character or String

Sample Output Set:

*******Student Records System**********

1. Add new Record
2. Display All Records
3. Delete record by roll no.
4. Search a Record by Roll no.
5. Enter your choice 1

Enter a new Record-
Student Roll No: 210054
Student Name: Sangram Kulkarni
Division: SE A
Address: Pune

Initialization of Input Set parameters:

```
class Student
{
 int rollno;
 char name[20],address_city[20];
 char div;
 int year;
public:
 Student()
{
  strcpy(name,“ ");
  strcpy(address_city,“ ");
  rollno=year=div=0;
}
```

Algorithms:

1. **Algorithm for main function**

MAIN FUNCTION ()

S1: Read the filenames from user from database.

S2: Read the operations to be performed from the keyboard

S3: If the operation specified is create go to the create function, if the operation specified is display go to the display function, if the operation specified is add go to the add function, if the operation specified is delete go to delete function, if the operation specified is display particular record go to the search function, if the operation specified is exit go to step 4.

S4: Stop

```
//==========File Operations ==========
class FileOperations
{
 fstream file;
public:
 FileOperations(char* filename)
```

Algorithm for create function

S1: Open the file in the write mode, if the file specified is not found or unable to open then display error message and go to step5, else go to step2.

S2: Read the no: of records N to be inserted to the file.

S3: Repeat the step4 N number of times.

S4: Read the details of each student from the keyboard and write the same to the file.

S5: Close the file.

S6: Return to the main function

Algorithm for add a record

S1: Open the file in the append mode, if the file specified is not found or unable to open then

display error message and go to step5 , else go to step2

S2: Scan all the student details one by one from file until end of file is reached.

S3: Read the details of the form the keyboard and write the same to the file

S4: Close the file.

S5: Return to the main function

```
void insertRecord(int rollno, char name[MAX],int year,
char div,char city[MAX])
 {
  Student s1(rollno,name,year,div,city);
  file.seekp(0,ios::end);
  file.write((char *)&s1,sizeof(Student));
  file.clear();
 }
```

Algorithm for deleting a record

S1: Open the file in the append mode, if the file specified is not found or unable to open then display error message and go to step5, else go to step2

S2: Accept the roll no from the user to delete the record

S3: Search for the roll no in file. If roll no. exits, copy all the records in the file except the one to be deleted in another temporary file.

S4: Close both files

S5: Now, remove the old file & name the temporary file with name same as that of old file name.

```
void deleteRecord(int rollno)
 {
 ofstream outFile("new.dat",ios::binary);
 file.seekg(0,ios::beg);
 bool flag=false;
 Student s1;

 while(file.read((char *)&s1, sizeof(Student)))
 {
  if(s1.getRollNo()==rollno)
  {
   flag=true;
   continue;
  }
  outFile.write((char *)&s1, sizeof(Student));
 }
 if(!flag)
 {
  cout<<"\nRecord of "<<rollno<<" is not present.";
 }
```

Algorithm for displaying particular record(search)

S1: Open the file in the read mode, if the file specified is not found or unable to open then

display error message and go to step6, else go to step2.

S2: Read the roll number of the student whose details need to be displayed.

S3: Read each student record from the file.

S4: Compare the students roll number scanned from file with roll number specified by the user.

S5: If they are equal then display the details of that record else display required roll number

not found message and go to step6.

S6: Close the file.

S7: Return to the main function.

```
while(file.read((char*)&s1,sizeof(Student)))
 {
  if(s1.getRollNo()==rollNo)
  {
   s1.displayRecord();
   flag=true;
   break;
  }
 }
 if(flag==false)
 {
  cout<<"\nRecord of "<<rollNo<<"is not present.";
 }
 file.clear();
}
```

Test Conditions:

1. Input valid filename.

2. Input valid record.

3. Check for opening / closing / reading / writing file errors

Successful Test Conditions:

Create

Accept the records and write into the file. File created successfully

Display

Display all records present in Master file.

Add

Accept the record to be add into the file at the end of the file. Record inserted successfully

Delete

Accept the record to be deleted. Record deleted successfully

Search

Accept the record to be displayed by search. Display whether the record if it is present else record not present.

2. 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.

Objective:

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

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.

Software & Hardware Requirements:

64-bit Open source Linux or its derivative

Open Source C++ Programming tool like G++/GCC

Online mode: online GDB classroom, Google classroom

Theory Contents:

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

Sample Input Set:

Record Type Name	Field Name	Data Type
Employee	Employee ID	Long Integer
	Name	Character or String
	Designation	Character or String
	Salary	Integer

Sample Output Set:

*******Employee Management System**********

1. Add new Record
2. Display All Records
3. Delete record by roll no.
4. Search a Record by Employee ID.
5. Enter your choice 1

Enter a new Record-
Employee ID: 101
Employee Name: Sangram Kulkarni
Designation: HR Manager
Salary: 60000\-

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 - returnn 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. };

```
using namespace std;

class employee
{
        int   emp_number;
        char  emp_name[20];
        float emp_basic;
        float emp_da;
        float emp_it;
        float emp_net_sal;

        public:

                void get_emp_details();
                float find_net_salary(float
basic, float da, float it);
                void show_emp_details();
};
```

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

```
void build()
{
   cout << "Build The Table\n";
   cout << "Maximum Enteries can be "
        << max << "\n";

   cout << "Enter the number of "
        << "Enteries required";
   cin >> num;

   if (num > 20) {
      cout << "Maximum number of "
           << "Enteries are 20\n";
      num = 20;
   }
   cout << "Enter the following data:\n";

   for (int i = 0; i < num; i++) {
      cout << "Name ";
      cin >> emp[i].name;

      cout << "Employee ID ";
      cin >> emp[i].code;

      cout << "Designation ";
      cin >> emp[i].designation;
 cout << “Salary ";
      cin >> emp[i].salary;
   }

   showMenu();
}
```

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.

```
// Function to delete record at index i
void deleteIndex(int i)
{
    for (int j = i; j < num - 1; j++) {
        emp[j].name = emp[j + 1].name;
        emp[j].code = emp[j + 1].code;
        emp[j].designation
            = emp[j + 1].designation;
        emp[j].exp = emp[j + 1].exp;
        emp[j].age = emp[j + 1].age;
    }
    return;
}

// Function to delete record
void deleteRecord()
{
    cout << "Enter the Employee ID "
         << "to Delete Record";
    int code;
cin >> code;
    for (int i = 0; i < num; i++) {
        if (emp[i].code == code) {
            deleteIndex(i);
            num--;
            break;
        }
    }
    showMenu();
}
```

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.

```
void searchRecord()
{
    cout << "Enter the Employee"
         << " ID to Search Record";

    int code;
    cin >> code;

    for (int i = 0; i < num; i++) {

        // If the data is found
        if (emp[i].code == code) {
            cout << "Name "
                 << emp[i].name << "\n";

            cout << "Employee ID "
                 << emp[i].code << "\n";

            cout << "Designation "
                 << emp[i].designation << "\n";

            cout << "Experience "
                 << emp[i].exp << "\n";

            cout << "Age "
                 << emp[i].age << "\n";
            break;
        }
    }
```

Problem Statement:

Assume we have two input and two output tapes to perform the sorting. The internal memory can hold and sort m records at a time. Write a program in java for external sorting. Find out time complexity.

Objective:

To understand the concept and basic of Sorting, internal and external memory handling.

To analyse time complexity of sorting algorithm.

Software & Hardware Requirements:

64-bit Open source Linux or its derivative

Open Source C++ Programming tool like G++/GCC

Java applets.

Online mode: online GDB classroom, Google classroom

Theory Contents:

External Sorting:

External Sorting is sorting the lists that are so large that the whole list cannot be contained in the internal memory of a computer.

Assume that the list(or file) to be sorted resides on a disk. The term block refers to the unit of data that is read form or written to a disk at one time. A block generally consists of several records. For a disk, there are three factors contributing to read/write time:

(i) Seek time: time taken to position the read/write heads to the correct cylinder. This will depend on the number of cylinders across which the heads have to move.

(ii) Latency time: time until the right sector of the track is under the read/write head.

(iii) Transmission time: time to transmit the block of data to/from the disk.

Example of External Sorting: 2-way Merge Sort, k-way Merge Sort, Polyphase Merge sort etc.

External Sorting with Disks:

The most popular method for sorting on external storage devices is merge sort.

This method consists of essentially two distinct phases.

First Phase(Run Generation Phase): Segments of the input file are sorted using a good internal sort method. These sorted segments, known as runs, are written out onto external storage as they are generated.

Second Phase(Merge Phase): The runs generated in phase one are merged together following the merge tree pattern , until only one run is left.

2-way Merging/ Basic External Sorting Algorithm

Assume unsorted data is on disk at start.

Let M=maximum number of records that can be sorted & sorted in internal memory at one time.

Algorithm:

Repeat

1. Read M records into main memory & sort internally.

2. Write this sorted sub-list into disk. (This is one “run”).

Until data is processed into runs.

Repeat

1. Merge two runs into one sorted run twice as long.

2. Write this single run back onto disk

Until all runs processed into runs twice as long

Merge runs again as often as needed until only one large run: The sorted list.

Pseudo Code for merging of M records:

```
open N input files and one output file
(* initialize buffers *)
loop from i = 1 to N
if end_of_file (file i)
 then buffer[i] <- invalid_key else buffer[i] <- first record of
file i;
(* merge *)
stop <- false
repeat
s <- index of smallest buffer element
if buffer[s] = invalid_key
 then stop <- true else write buffer[s]
 if end_of_file (file s)
 then buffer[s] <- invalid_key
 else buffer[s] <- next record from file s
until stop = true
close files
```

Measuring Performance: Time Complexity:

The size of a run and number of initial runs (nr) is dictated by the number of file blocks (b) and the available buffer space (nb)

- Nr = b/nb
- E.g b = 1024 blocks and nb = 5 blocks then 205 initial runs will be needed

The degree of merging (dm) is the number of runs that can be merged together in each pass.

- One buffer block is needed to hold one block from each run being merged
- One buffer block is needed for containing one block of merged result
- Dm is the smaller of (nb – 1) and nr
- Number of passes = logdm(nr)

Analysis

his algorithm requires log(N/M) passes with initial run pass. Therefore, at each pass the N records are processed and at last we will get a **time complexity as O(N log(N/M).**

Sample of Assessment Parameters:

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total
2	2	2	2	2	10

FAQ's:

1. Define File? What are the factors affecting the file organization?
2. Compare Text and Binary File?
3. Explain the different File opening modes in C++?
4. What is indexed sequential file? Explain the primitive operations on indexed sequential file.

Write a note on Direct Access File?

6. What are the advantages and disadvantages of indexed-sequential file organization?
7. What are the advantages of Sequential File Organization?
8. What are serial and sequential files and how are they used in organization?
9. Distinguish between logical and physical deletion of records and illustrate it with suitable examples.
10. Compare and contrast sequential file and random access file organization?
11. Explain the different types of external storage devices?
12. With the prototype and example, explain following functions:

    i) seekg() ii) tellp() iii) seekp() iv) tellp()

Learning Resources

Links: https://syrah.eecs.harvard.edu/files/syrah/files/usenix-02-dafs.pdf (direct Access)
https://digital.library.unt.edu/ark:/67531/metadc663738/m2/1/high_res_d/1002772864-Rogers.pdf

Books: http://cruiserselite.co.in/downloads/btech/materials/second%20sem/2/ADS/UNIT-1.pdf (External Sort)

- Dr. Dobbs/books/book5/chap03.htm
- Aho, J. Hopcroft, J. Ulman, "Data Structures and Algorithms"
- Sartaj Sahani, "Data Structures, Algorithms and Applications in C++"
  https://www.cs.uct.ac.za/mit_notes/database/pdfs/chp11.pdf

Papers:

- https://digital.library.unt.edu/ark:/67531/metadc663738/m2/1/high_res_d/1002772864-Rogers.pdf
  https://www.kean.edu/~gchang/tech2920/http___professor.wiley.com_CGIBIN_JSMPROXY_DOCUMENTDIRECTORDEV+DOCUMENT

Thank You…

Presented by, Ms. Pallavi V. Baviskar PES MCOE Pune-5