

PROJECT REPORT ON
“COACHING INSTITUTE MANAGEMENT SYSTEM
using File Structures”

INDEX

CONTENTS

1	Introduction
2	Objective
3	Scope
4	Key Concepts: <ul style="list-style-type: none">File StructuresIndexingMultilevel Indexing
5	Program Implementation: <ul style="list-style-type: none">Project StructureHeader FilesFunctionalities
6	Program Output
7	Conclusion
8	References

INTRODUCTION

The education plays a pivotal role in the life of a student. Especially in a growing country like India, education is developing and becoming a big part. Learning and Education is imbibed within our culture from the ancient times and a great importance is placed on academic success in our society. This has resulted in expansive rise of Coaching Centres.

Coaching Industry in India is a industry that's worth 150,000 crores. It's prevalence can be found in every part of India.

These coaching institutes stretch and encompass a wide range. They include elementary coaching for students of grade 5 and up. But due to the significance of competitive examinations such as JEE and NEET, they take quite a large amount of share as they require extra and specialised learning. Moreover certain international exams and higher level studies such as GRE, IELTS, GMAT, CAT and so on are solely taught by coaching centres. These all makes the coaching centres very significant.

OBJECTIVE

The project objective is to help manage records of the Coaching Centre. This project is implemented by utilizing the concepts of File Structures. The key topic it makes use of is Indexing. The data is stored in the files that is structured by means of indexing the entries. It allows for easy and convenient access to the records for retrieval and modification purposes.

The main two data to manage in this project are Students information and Educators information. This data is subjected to grow as new batches are started. Thus it becomes necessary to find a way to store these huge files. This project aims to do exactly that and provides an efficient way for storing the files.

SCOPE

There is a growing demand and market for the coaching institutes in the developing country like India. This can attributed to the importance placed on education as it is the tool of the intellectual.

The Indian Coaching industry is a 1500 crore industry growing at Compound Annual Growth Rate of 34%. Indian literacy rate has risen from 65% to 73% in the last five years. However the brightest minds in India keeps rising and nourished with the help of coaching institutes. The statistics reveals an average Indian middle-class family spends 1/3rd of their income on their child's education. With this importance, coaching centres have a lot of scope.

With the rise in online tutoring as well as increased strength of a coaching centre, it becomes necessary for it to keep track of their data. Many of these private coaching centres are either medium scaled or lack the modern technological infrastructure and instead rely on traditional way. With this project we can help them digitalize their records and store their data in an efficient manner so they can carry out their coaching in a much convenient manner and increase productivity

KEY CONCEPTS

File Structures

File Structure is a combination of representations for data in files and operation to access the data. File Structure is the organization of data in secondary storage structures such as disks. A successful file structure organizes your data and code with the goal of repeatability, making it easier for you and your collaborators to revisit, revise and develop your project. File structures are not fixed entities, but rather build a framework that communicates the function and purpose of elements within a project by separating concerns into a hierarchy of folders and using consistent, chronological, and descriptive names. An improvement in file structure design may make an application hundred times faster. The details of the representations of the data and the implementation of operations determine the efficiency of the file structure.

Indexing

File Indexing is a technique to efficiently retrieve records from the stored files based on some attributes on which the indexing has been done. There are several types of single level ordered indexes.

Primary index is an index specified on the ordering key field of a sorted file of records. Clustering index is an index built on the records that are sorted on the non-key field. It is since different records have different values in the key field, but for a non-key field, some records may share the same value.

Secondary index can be specified on any non-ordering field of a file. A file can have several secondary indexes in addition to its primary access path. These indexes do not affect the physical organisation of records.

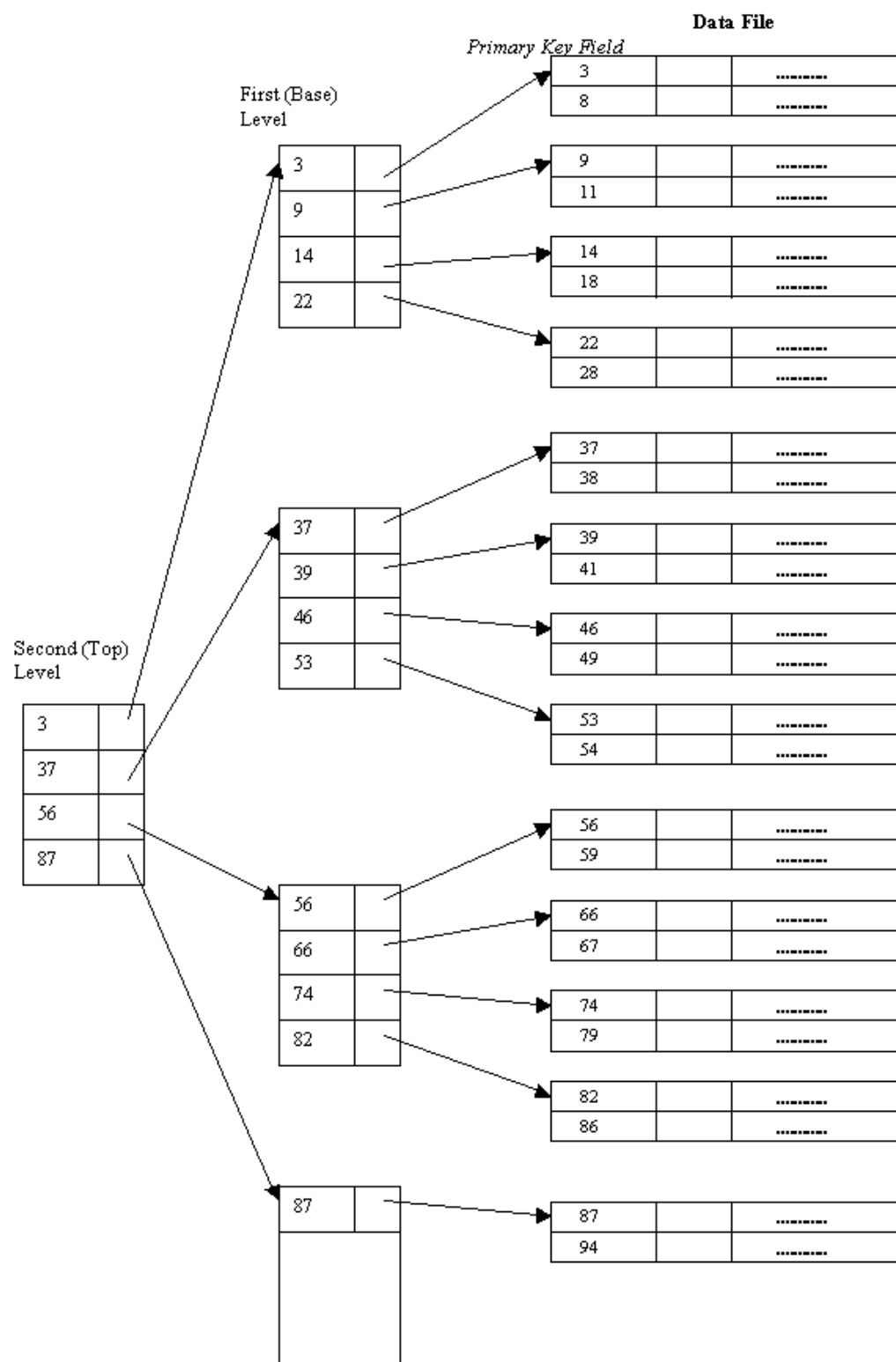
Multi-level Indexing

The idea behind a multilevel index is to reduce the part of the index that we have to continue to search by blocking factor for the index, which is almost always larger than 2. Thus, the search space can be reduced much faster. Searching a multilevel index requires block accesses, which is a smaller number than for a binary search if the blocking factor of the index is bigger than 2.

A multi-level index considers the index file, which we referred as a single-level ordered index or it's referred as the first level (or) base level of the multi-level structure, as a sorted file with a distinct value for each primary key field. We can build a primary index for an index file itself (on top of the index at the first level). This new index to the first level is called the second level of the multilevel index. Because the second level is a primary index, we can use block anchors so that the second level has one entry for each block of the first-level index entries.

The above process can be repeated and a third level index can be created on top of the second level one. The third level, which is a primary index for the second level, has an entry for each second level block.

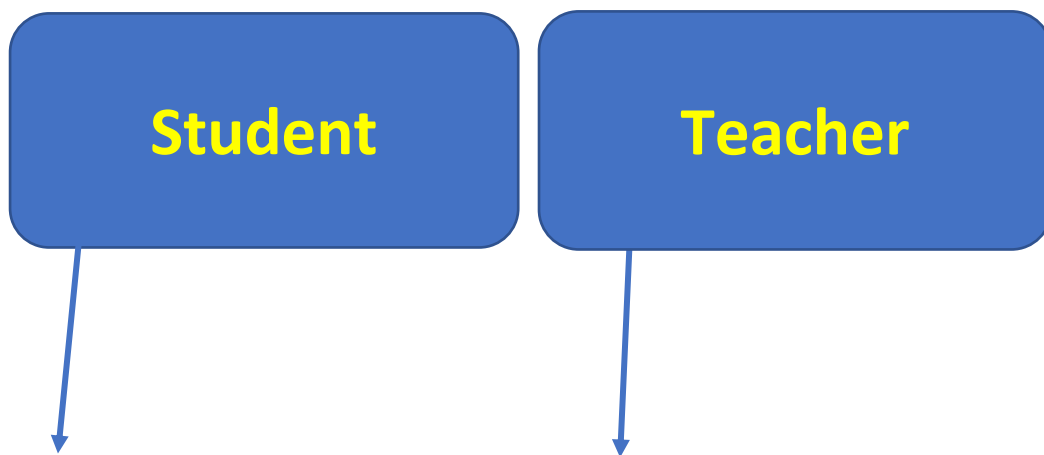
The multilevel structure can be used on any type of index as long as the first-level index has distinct values for primary key field and fixed-length entries. The figure below depicts a multilevel index built on top of a primary index



PROGRAM IMPLEMENTATION

Project Structure

The coaching institute will have two disparate sets of data to keep track and store. Those would be Student and Teachers



First Name	First Name
Last Name	Last Name
Registration ID	Teacher ID
Age	Qualification
Fee Status	Pay
Phone Number	Phone Number
Batch	Subject
Student Index	Teacher index

Header Files

#include <iostream>

iostream is the standard input output stream. It contains definitions of objects like cout to insert data into a stream (output), cin to extract data from a stream (input) and to control the format of data

#include <fstream>

It mainly describes the file stream. This header file is used to handle the data being read from a file as input or data being written into the file as output.

#include <string.h>

The string. h header defines one variable type, one macro, and various functions for manipulating arrays of characters

#include <conio.h>

It is a header file that includes inbuilt functions like getch() and clrscr(). It stand for console input ouput. it takes input from keyboard and displays it on screen

#include <cstdlib>

It stands for C Standard General Utilities Library. This defines several general purpose functions, including dynamic memory management, searching, sorting and converting to facilitate efficient, high-performing, standardized C++ code across platforms

INDEXING in the CODE

Primary Indexing is used in this project. We declare the INDEX as a Structure. The struct members “instudent” and “inteacher” are used throughout the program and functions to access, sort and modify the index file of student and teacher

```
// Primary Index Definition (for Student)
struct index
{
    char registrationid[20];           //Registration ID
    char indstudent[20];               //Corresponding Student Index
}instudent[20], tempstudent;

// Primary Index Definition (for Teacher)
struct index1
{
    char teacherid[20];                //Teacher ID
    char indteacher[20];               //Corresponding Teacher Index
}inteacher[20], tempteacher;
```

Sort_Index()

Indexing is the process of associating a key with the location of a data record. An external sort uses the concept of a key sort, in which an index file is created which consist of key and pointer pairs.

The index file is sorted or organized using a tree structure, thereby imposing a logical order on the records without physically rearranging them.

```
void sort_index_student()
{
    int i,j;

    for(i=0;i<no_student-1;i++)
    for(j=0;j<no_student-i-1;j++)
    if(strcmp(instudent[j].registrationid,instudent[j+1].registrationid)>0)
    {
        tempstudent=instudent[j];
        instudent[j]=instudent[j+1];
        instudent[j+1]=tempstudent;
    }
}
```

```
void sort_index_teacher()
{
    int i,j;

    for(i=0;i<no_teacher-1;i++)
    for(j=0;j<no_teacher-i-1;j++)
    if(strcmp(inteacher[j].teacherid,inteacher[j+1].teacherid)>0)
    {
        tempteacher=inteacher[j];
        inteacher[j]=inteacher[j+1];
        inteacher[j+1]=tempteacher;
    }
}
```

Retrieve_Record()

This facilitates the client who want to get every details of any particular number of student or teacher. They can specify their request and this function runs through the index and returns the data with very fast access.

```
void retrieve_record_student(char *ind)
{
    for(int i=0;i<no_student;i++)
    {
        if(strcmp(studentData[i].ind,ind)==0)
        {
            strcpy(rec_ind_student,ind);
            rec_flag_student=1;
            cout<<"Record Found:\n";
            cout<<studentData[i].ind<<"\t"<<studentData[i].fname \
            <<"\t"<<studentData[i].lname<<"\t"<<studentData[i].registrationid \
            <<"\t"<<studentData[i].age<<"\t"<<studentData[i].cel_no \
            <<"\t"<<studentData[i].batch<<"\n";
            return;
        }
    }
    cout<<"Press any key to continue: \n";
    getch();
}
```

```
void retrieve_record_teacher(char *ind)
{
    for(int i=0;i<no_teacher;i++)
    {
        if(strcmp(teacherData[i].ind,ind)==0)
        {
            strcpy(rec_ind_teacher,ind);
            rec_flag_teacher=1;
            cout<<"Record found:\n";
            cout<<teacherData[i].ind<<"\t"<<teacherData[i].fst_name<<"\t"<<teacherData[i].lst_name<<"\t"<<teacherData[i].teacherid<<"\t" \
            <<teacherData[i].qualification<<"\t"<<teacherData[i].pay<<"\t"<<teacherData[i].batch<<"\t"<<teacherData[i].cel_no<<"\n";
            return;
        }
    }
    cout<<"Press any key to continue: \n";
    getch();
}
```

Search_Index()

This function allows to look for the primary index, whether it exists or not and return the corresponding record if it exists. Registration ID for students and Teacher ID for teachers is the primary index

```
int search_index_student(char *registrationid)
{
    int flag=0;
    for(int i=0;i<no_student;i++)
    {
        if(strcmp(instudent[i].registrationid,registrationid)==0)
        {
            retrieve_record_student(instudent[i].indstudent);
            flag=1;
        }
    }
    if(flag)
        return 1;
    else
        return -1;
}
```

```
int search_index_teacher(char *teacherid)
{
    int flag=0;
    for(int i=0;i<no_teacher;i++)
    {
        if(strcmp(inteacher[i].teacherid,teacherid)==0)
        {
            retrieve_record_teacher(inteacher[i].indteacher);
            flag=1;
        }
    }
    if(flag)
        return 1;
    else
        return -1;
}
```

Delete()

This function is used to delete the record that is no longer needed. It checks if the inputted key exists or not. If it does, then that record is deleted from the file permanently. The other records and their indices are sorted then.

```
void delet_student(char *st_registrationid)
{
    rec_flag_student=0;
    int fr=0;
    search_index_student(st_registrationid);
    if(!rec_flag_student)
    {
        cout<<"Deletion failed. Record not found\n";
        cout<<"Press any key to continue: \n";
        getch();
        return;
    }
    for(int i=0;i<no_student;i++)
    {
        if(strcmp(studentData[i].ind,rec_ind_student)==0)
        {
            fr=i;
            break;
        }
    }
    for(int i=fr;i<no_student-1;i++)
    {
        studentData[i]=studentData[i+1];
        char str[3];
        sprintf(str,"%d",i);
        strcpy(studentData[i].ind,str);
    }
}
```

```

no_student--;
fstream f1,f2;
f1.open("record_student.txt",ios::out);
f2.open("index_student.txt",ios::out);
for(int i=0;i<no_student;i++)
{
    strcpy(instudent[i].registrationid,studentData[i].registrationid);
    strcpy(instudent[i].indstudent,studentData[i].ind);
}
sort_index_student();
for(int i=0;i<no_student;i++)
{
    f1<<studentData[i].ind<<"|"<<studentData[i].fname<<"|" \
    <<studentData[i].lname<<"|"<<studentData[i].registrationid<<"|"<< \
    studentData[i].feestatus<<"|"<<studentData[i].age<<"|"<<\
    studentData[i].cel_no<<"|"<<studentData[i].batch<<"\n";
    f2<<instudent[i].registrationid<<"|"<<instudent[i].indstudent<<"\n";
}
f1.close();
f2.close();
cout<<"Deletion Successful\n";
cout<<"Press any key to continue:\n";
getch();
}

```


Update_Status()

This is used to update the Fee status of a student. Oftentimes, students who opt in for coaching pay partial fees at the beginning and later pay the full fee. We maintain fee status as whether paid or unpaid. When the student pays the fees, we update it. The details of students who haven't paid the fees can be retrieved which is helpful.

```
void update_status(char *id)
{
    for (int k = 0; k < no_student; k++)
    {
        if (strcmp(studentData[k].registrationid, id) == 0)
        {
            if (strcmp(studentData[k].feestatus, "Unpaid") == 0)
            {
                strcpy(studentData[k].feestatus, "Paid");
            }
        }
    }
    fstream f1;
    f1.open("record_student.txt", ios::out);
    for (int k = 0; k < no_student; k++)
    {
        f1 << studentData[k].ind << " | " << studentData[k].fname << " | " \
        << studentData[k].lname << " | " << studentData[k].registrationid << " | " \
        << studentData[k].feestatus << " | " << studentData[k].age << " | " << \
        studentData[k].cel_no << " | " << studentData[k].batch << "\n";
    }
    f1.close();
    cout << "Updation Successful";
    cout << "Press any key to continue:";
    getch();
}
```

Main()

The brief description of main() function implementation are as follows

- Local Declaration of Record Fields
- Creation of record and index file for students
- Structuring record of student files
- Key sorting the student index
- Creation of record and index file for teachers
- Structuring record of teacher files
- Key sorting the teacher index
- Get input whether client wants to access Students section (or) Teachers section
- Use the WHILE loop for getting data on repeat and exit out of the loop when user specifies the exit option
- Use the Switch case to jump to either Students section or Teachers section
- The same idea is repeated inside of the both sections

- Student Information

- 1) New Admission: Get details of new students admitted and insert it into the file
- 2) Find Student: Get the ID of student and check whether it exists or not
- 3) Check Fee Status: Display the details of students who have paid or not paid the fee
- 4) Remove: Delete the student record of specified ID from the file
- 5) Display: Display the details of all the students or display the data batchwise
- 6) Update Fee Status: Modify the fee status as paid or unpaid for a student
- 7) Exit

- Teachers Information

- 1) Create New Entry: Get details of the new teachers and insert it into the file
- 2) Find Teacher: Search for a teacher using the key teacher ID
- 3) Remove: Delete the details of the teacher that no longer works in the institute
- 4) Display: Display the particulars of all the teachers or batchwise display the details
- 5) Students Assigned: See the students assigned for a particular teacher to coach
- 6) Exit

PROGRAM OUTPUT

COACHING INSTITUTE MANAGEMENT SYSTEM

HOME PAGE

This is a Coaching System, used to maintain the records of students and teachers in the coaching institute

The two basic sections are:

- 1.Student Section
- 2.Teacher Section

Which section do you want to avail? Choose an option below

- 1) Student Information
- 2) Teacher Information
- 3) Exit Program

STUDENTS INFORMATION AND BIODATA SECTION

Enter your Choice:

- 1) New Admission
- 2) Find Student
- 3) Check Fee Status
- 4) Remove Student
- 5) Display Students Batchwise
- 6) Display all Students
- 7) Update Fee status
- 8) Jump to Main Menu

STUDENTS INFORMATION AND BIODATA SECTION

Enter your Choice:

- 1) New Admission
- 2) Find Student
- 3) Check Fee Status
- 4) Remove Student
- 5) Display Students Batchwise
- 6) Display all Students
- 7) Update Fee status
- 8) Jump to Main Menu

1

Enter the number of students: 3

Enter the Details below

Enter Student Details:

First Name: Aaryan

Last Name: Dhube

Registration ID: 16001

Fee Status: Paid

Age: 18

Batch Allotted: JEE

Contact Number: 9268217628

Enter Student Details:

First Name: Abhay

Last Name: Hegde

Registration ID: 16002

Fee Status: Unpaid

Age: 16

Batch Allotted: Boards

Contact Number: 9106729392

Enter Student Details:

First Name: Bhudharan

Last Name: Gowda

Registration ID: 16003

Fee Status: Paid

Age: 21

Batch Allotted: CAT

Contact Number: 7629195872

Do You Want to Enter More Data?

Press Y for Continue and N to Finish: _

Enter your Choice:

- 1) New Admission
- 2) Find Student
- 3) Check Fee Status
- 4) Remove Student
- 5) Display Students Batchwise
- 6) Display all Students
- 7) Update Fee status
- 8) Jump to Main Menu

2

Enter the ID of the student whose record is to be searched: 16001

Record Found:

Index	Fname	Lname	ID	Age	Phone	Batch
1	Aaryan	Dhube	16001	18	9387509275	JEE

Search Successful!

Press any key to continue:

STUDENTS INFORMATION AND BIODATA SECTION

Enter your Choice:

- 1) New Admission
- 2) Find Student
- 3) Check Fee Status
- 4) Remove Student
- 5) Display Students Batchwise
- 6) Display all Students
- 7) Update Fee status
- 8) Jump to Main Menu

4

Enter the ID of the Student whose Record is to be Deleted: 16002

Record Found:

Index	Fname	Lname	ID	Age	Phone	Batch
2	Abhay	Hegde	16002	16	9008636743	Boards

Deletion Successful

Press any key to continue:

-

TEACHERS INFORMATION AND BIODATA SECTION

Enter your Choice:

- 1) Create New Entry
- 2) Find Teacher
- 3) Remove Teacher
- 4) Batcheswise Display
- 5) Students Assigned
- 6) Display All Teachers
- 7) Jump to Main Menu

1

Enter the number of Teachers: 2

Enter the Details Below

Enter Teacher Details:

First Name: Ramesh

Last Name:

Doddmane

Qualification: BSc

Teacher ID: 5001

Pay: 20000

Batch Allotted: Boards

Contact Number: 9108732567

Enter Teacher Details:

First Name: Nagesh

Last Name: Bhat

Qualification: MSc

Teacher ID: 5005

Pay: 60000

Batch Allotted: JEE

Contact Number: 8762856104

Do you want to Enter further Data?

Press Y for Continue and N to Finish: N_

TEACHERS INFORMATION AND BIODATA SECTION

Enter your Choice:

- 1) Create New Entry
 - 2) Find Teacher
 - 3) Remove Teacher
 - 4) Batcheswise Display
 - 5) Students Assigned
 - 6) Display All Teachers
 - 7) Jump to Main Menu
- 2

Enter the ID of the Teacher whose record is to be Searched: 5001

Record Found:

Index	Fname	Lname	ID	Qualification	Pay	Batch	Phone
1	Ramesh	Doddmane	5001	BSc	20000	Boards	9108578321

Search Successful!

Press any key to continue:

TEACHERS INFORMATION AND BIODATA SECTION

Enter your Choice:

- 1) Create New Entry
 - 2) Find Teacher
 - 3) Remove Teacher
 - 4) Batcheswise Display
 - 5) Students Assigned
 - 6) Display All Teachers
 - 7) Jump to Main Menu
- 3

Enter the ID of the Teacher whose record is to be Deleted: 5001

Record Found:

Index	Fname	Lname	ID	Qualification	Pay	Batch	Phone
1	Ramesh	Doddmane	5001	BSc	20000	Boards	9108578321

Deletion successful

Press any key to continue

-

record_student.txt - Notepad

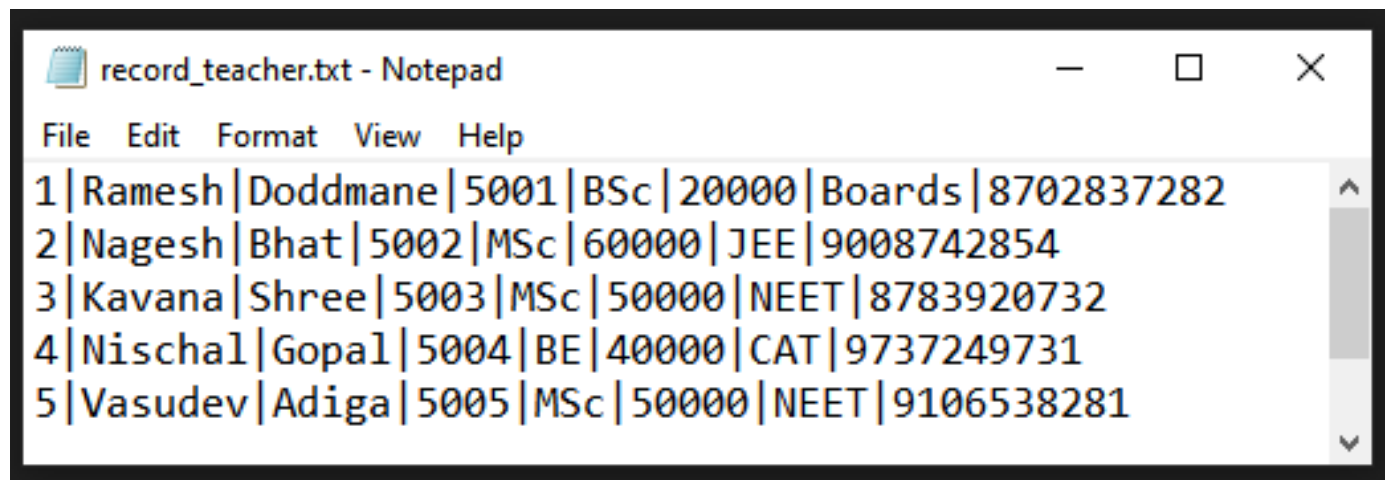
File Edit Format View Help

```
1|Aaryan|Gupta|16001|Paid|18|9109842378|JEE
2|Abhay|Hegde|16002|Unpaid|16|9008537218|Boards
3|Bhudharan|Gowda|16003|Paid|21|9107842374|CAT
4|Chandramouli|B|16004|Paid|17|900753728|NEET
5|Dheeraj|Singh|16005|Paid|18|907853728|JEE
6|Lakshmi|K|16006|Paid|17|987853728|NEET
7|Jack|Hughman|16007|Unaid|24|87053728|CAT
8|Fazal|Khan|16008|Paid|17|900854728|JEE
9|Anshika|M|16009|Paid|16|9106729374|Boards
10|Neeraj|Kumar|16010|Unpaid|18|8703246283|JEE
11|Vaibhav|Jain|16011|Paid|19|987853734|JEE
12|Shreyas|R|16012|Paid|18|8108246283|NEET|
```

index_student.txt - Notepad

File Edit Format View Help

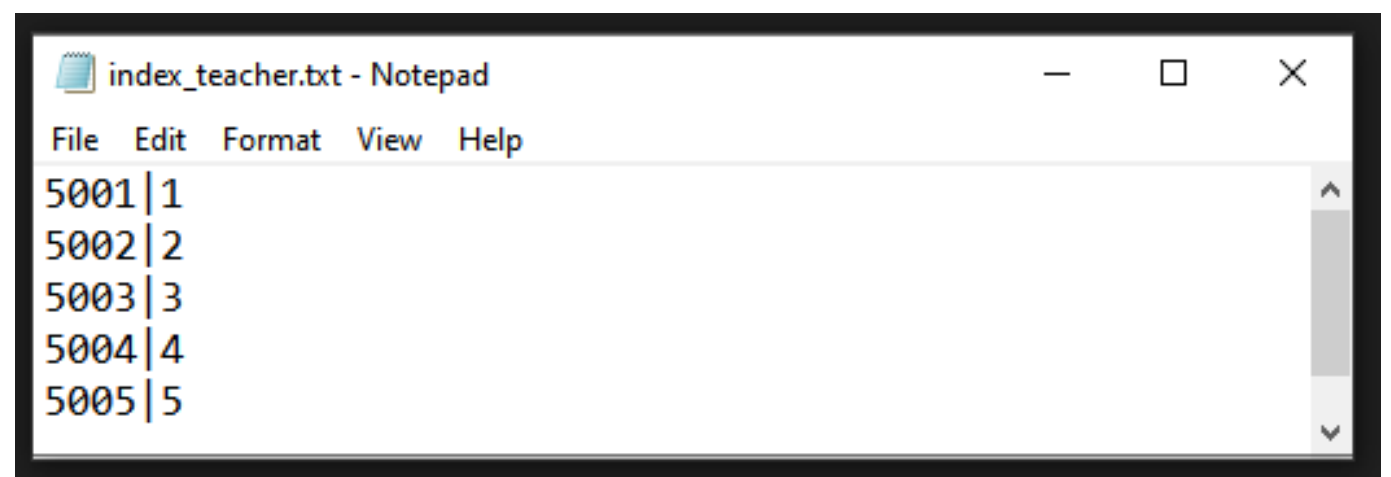
```
16001|1
16002|2
16003|3
16004|4
16005|5
16006|6
16007|7
16008|8
16009|9
16010|10
16011|11
16012|12
```



record_teacher.txt - Notepad

File Edit Format View Help

```
1|Ramesh|Doddmane|5001|BSc|20000|Boards|8702837282
2|Nagesh|Bhat|5002|MSc|60000|JEE|9008742854
3|Kavana|Shree|5003|MSc|50000|NEET|8783920732
4|Nischal|Gopal|5004|BE|40000|CAT|9737249731
5|Vasudev|Adiga|5005|MSc|50000|NEET|9106538281
```



index_teacher.txt - Notepad

File Edit Format View Help

```
5001|1
5002|2
5003|3
5004|4
5005|5
```

CONCLUSION

The Coaching Institute Management System has been made to aid the storage of details of students and teachers in the coaching centre. The records are obtained and efficiently stored in the file for easier and quick access whenever needed. Indexing of records is employed in this project that keeps index of corresponding records. This takes lesser storage and provides faster access during data retrieval. This project makes the file storage of the institute convenient.

REFERENCES

- 1) An Object Oriented Approach with C++ by Michael J. Folk, Bill Zoellick, Greg Riccardi
- 2) File Structures Using C++ by K.R. Venugopal, K.G. Srinivas, P.M. Krishnaraj
- 3) Indexing and File Organization Course by University of Cape Town (<https://www.cs.uct.ac.za>)
- 4) File Structures by GeeksforGeeks.org
- 5) Working with C++ Files by w3schools.com