VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JnanaSangama, Belagavi – 590014.

MINI PROJECT REPORT

ON

"Driving School Management System"

Submitted in partial fulfillment for the requirement of 6th semester for the

Degree of Bachelor of Engineering in

INFORMATION SCIENCE & ENGINEERING

For the academic year 2020-21

SUBMITTED BY:

SUNETHRA RANGANATH

[1DB18IS085]

Under the guidance of:

Dr. PRADEEP K.R.,

Associate Professor,

Dept. of ISE

DON BOSCO INSTITUTE OF TECHNOLOGY, BENGALURU-560074

DON BOSCO INSTITUTE OF TECHNOLOGY

Kumbalagodu, Bengaluru-560074

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Project Report entitled **"DRIVING SCHOOL MANAGEMENT SYSTEM"**is a bonafide Project work carried out by **SUNETHRA RANGANATH (1DB18IS085)**, in partial fulfillment of '6th' semester for the Degree of **Bachelor of Engineering in Information Science and Engineering** of Visvesvaraya Technological University, Belagavi, during the academic year 2021-22. It is certified that all corrections/suggestions indicated for Internal Assessments have been incorporated with the degree mentioned.

Project Guide	**Head of Department**
Dr. Pradeep K R	**Prof. Gowramma G.S**
Associate. Prof.	Head of Department
Dept. of ISE,	Dept. of ISE,
DBIT, Bangalore.	DBIT, Bangalore.

External Viva

Name of the Examiners	**Signature with Date**
1.________________	________________
2.________________	________________

DON BOSCO INSTITUTE OF TECHNOLOGY

Kumbalagodu, Bengaluru -560074

DECLARATION

SUNETHRA RANGANATH, student of sixth semester B.E, Department of Information Science and Engineering, Don Bosco Institute of Technology, Kumbalagodu, Bengaluru, declare, that Mini Project Work entitled **"DRIVING SCHOOL MANAGEMENT SYSTEM"** has been carried out by and submitted in partial fulfillment of the requirement of 6th semester April 2021-Aug 2022. The matter embodied in this report has been submitted to any university or institute for the award of any other degree or diploma.

Place: Bengaluru **SUNETHRA RANGANATH**

Date: **(1DB18IS085)**

ACKNOWLEDGEMENT

At the various stages in making the mini project, a number of people have given me invaluable comment on the manuscript. I take this opportunity to express my deepest gratitude and appreciation to all those who helped me directly or indirectly towards the successful completion of this mini project.

I would like to thank **Principal Dr HEMADRI NAIDU. T, DonBosco Institute of Technology** for his support though out this mini project.

I express my whole hearted gratitude to **Prof.GOWRAMMA G.S,** who is our respectable **Head of Dept. Department of Information Science and Engineering**. I wish to acknowledge for her valuable help and encouragement.

In this regard, a heartfelt gratitude to guide **Dr. PRADEEP K R Associate Professor, Department of Information Science and Engineering**, for her timely advice on the miniproject and regular assistance throughout the miniproject work. I would also like to thank the staff members of Department of **Information** Science and Engineering for their corporation.

ABSTRACT

The main aim and objective was to plan and program system application and to get rid of manual entry and to store it in a file so that easily available. We have to apply the best software engineering practice for system application. I developed “Driving School Management system” where the software Code Blocks for C++ is used and perform basic operations like insertion, deletion, display, update and search can be performed. Driving School Management system can handle all the details about a student of the driving school. The user can collect student information by adding, displaying, updating, removing and searching for details. This mini project contains limited features, but essential ones. These details include ID, name, course and many more. It tracks all the details of a student from the day one to the end of his course. It helps to coaches to make progress chart and student remarks. All the details regarding the student can be sent through SMS or email.

CONTENTS

Chapter-1

INTRODUCTION

I have developed driving school management system to get rid from manual entry and store it in a file which can be easily available. Here we perform various operations like inserting a record, deleting a record, search a record, updating a record and displaying a record. For this purpose, need the software which could easily and conveniently maintain the student details. The record of student can be stored in a single file.This project "driving school management system" includes some facilities such as id, name, course, phone number, location, etc .

1.1 PROBLEM STATEMENT

The main aim of designing this project is to get rid from manual entry and store it in a file which is easily available and perform operations like insert, delete, search, modify and display the records for driving school management system.

1.2 REQUIREMENTS

HARDWARE REQUIREMENT-

Minimum RAM	:-	2GB
Processor	:-	Intel Pentium 5
Operating System	:-	Windows 10

SOFTWARE REQUIREMENT-.

Language	:-	C++
Software used	:-	Code Blocks for C++

1.3 Scope

The software product "**Driving School Management System**" will be an application that will be used for maintaining the records in an organized manner and to replace old paper work system. This project aims at automating the student details for smooth working of the database by automating almost all the activities. Updations and modifications will be easily achievable.

Chapter-2.

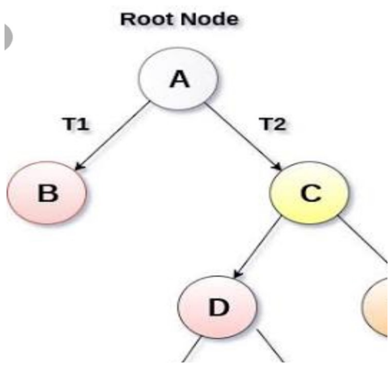
FILE STRUCTURE INTRODUCTION

File Structures is the Organization of Data in Secondary Storage Device in such a way that minimize the access time and the storage space. A File Structure is a combination of representations for data in files and of operations for accessing the data. A File Structure allows applications to read, write and modify data.

SHORT HISTORY OF FILE STRUCTURE DESIGN:

General goals of research and development in file structures:

- To find the target information with as few access as possible (i.e., 2 or 3 accesses).
- To get the information we need with one access to the disk.
- To group information to get everything with only one access.
- Trees -in the early 1960s, the idea of applying tree structures emerged as a potential solution. Unfortunately, trees can grow very unevenly as records are added and deleted, resulting in long searches requiring many disk accesses to find a record.



- Sequential access -> Direct access

  Tape -> Disk.

  Index : <key, pointer> in smaller file.

- As files grew intolerably large for unaided sequential access an as storage devices such as disk drives became available, indexes are added to files. Indexes made it possible to keep a list of keys and pointers in a smaller file that could be searched more quickly .With the key and pointer, user had direct access to large primary file.

- AVL tree is a self -adjusting binary tree for data in memory .Other researchers began to look for ways to apply AVL trees, or something like them to files.

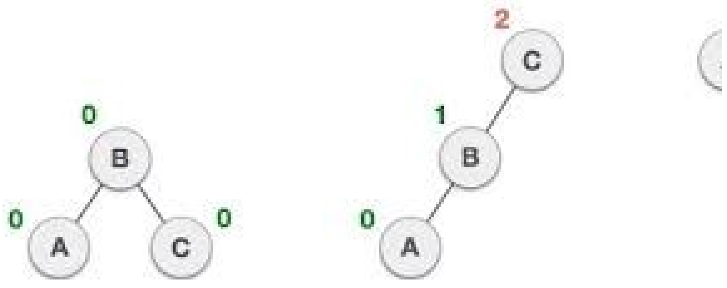


Fig: Examples for AVL tree

- It took nearly 10 years of design work before a solution emerged in the form of the B-tree. B-tree is a balanced tree structure and provide excellent access performance, but sequential access with a cost.

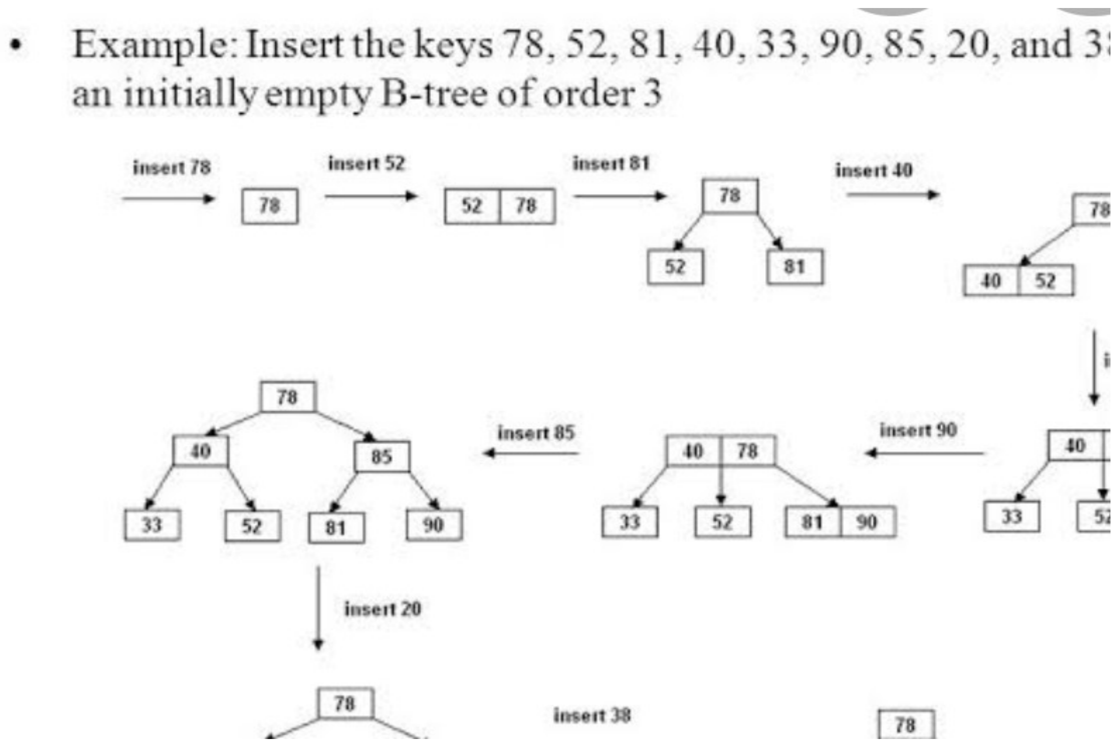

- Example: Insert the keys 78, 52, 81, 40, 33, 90, 85, 20, and 3
an initially empty B-tree of order 3

Fig:Example of B-trees

- The combination of a B-tree and sequential linked list is called a B +tree. Over the next 10 years, B-trees and B+ trees became the basis for many commercial file systems.

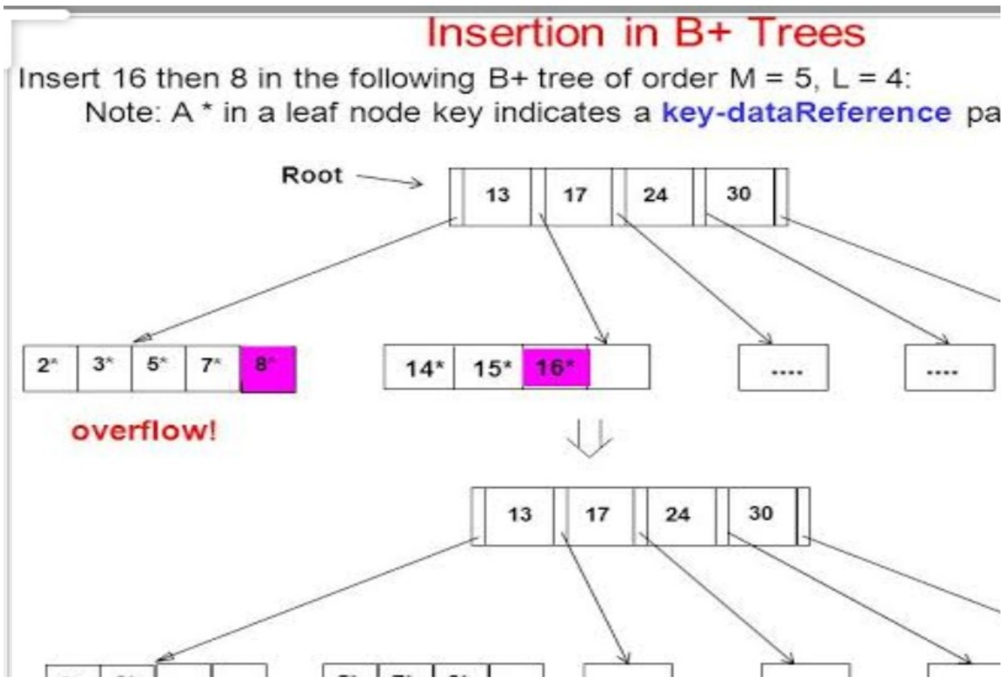

Insertion in B+ Trees
Insert 16 then 8 in the following B+ tree of order M = 5, L = 4:
Note: A * in a leaf node key indicates a key-dataReference pa
Root
13
17
24
30
2*
3*
5*
7*
8*
14*
15*
16*
....
....
overflow!
13
17
24
30

Fig: Example of B+ trees

- An approach called hashing is a good way to do that with the files that do not change size greatly over time. From early on, hashed indexes were used to provide fast access to files.

- After the development of B-trees, researchers turned to work on systems for extendible, dynamic hashing that could retrieve information with one ,or at most ,two disk access no matter how big the file became.

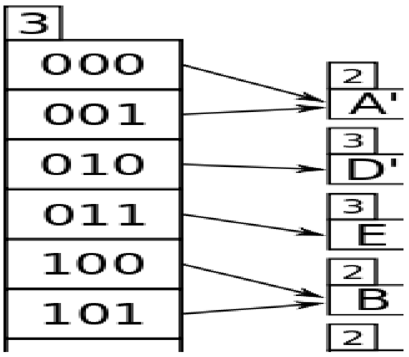


Fig:Extendible hashing

2.1 FILE STRUCTURE USED IN PROJECT:INDEXES

INDEXES:

All indexes are based on same basic concepts-keys and reference fields. The types of indexes we see are called simple indexes because they are represented using simple arrays of structures that contain the keys and reference fields.

A Simple Index for Entry-Sequenced Files

Simple index: An index in which the entries are a key ordered linear list.

- Simple indexing can be useful when the entire index can be held in memory.
- Changes (additions and deletions) require both the index and the data file to be changed.
- Updates affect the index if the key field is changed, or if the record is moved.
- An update which moves a record can be handled as a deletion followed by an addition.

Object Oriented Support for Indexed, Entry Sequenced Files

Entry-sequenced file: A file in which the record order is determined by the order in which they are entered. The index should be read into memory when the data file is opened.

- Indexes That are too Large to Hold in Memory
- Searching of a simple index on disk takes too much time.
- Maintaining a simple index on disk in sorted order takes too much time.
- Tree structured indexes such as B-trees are a scalable alternative to simple indexes.
- Hashed organization is an alternative to indexing when only a primary index is needed.

Indexing to Provide Access by Multiple Keys

Secondary key: A search key other than the primary key.

Secondary index: An index built on a secondary key.

- Secondary indexes can be built on any field of the data file, or on combinations of fields.
- Secondary indexes will typically have multiple locations for a single key.
- Changes to the data may now affect multiple indexes.
- The reference field of a secondary index can be a direct reference to the location of the entry in the data file.
- The reference field of a secondary index can also be an indirect reference to the location of the entry in the data file, through the primary key.
- Indirect secondary key references simplify updating of the file set.
- Indirect secondary key references increase access time.
- We could build a catalog for our record collection consisting of entries for album title, composer, and artist. These fields are secondary key fields . Just as the library catalog relates an author entry (secondary key) to a card catalog number (primary key) , so can we build an index file that relates Composer to Label ID, as illustrated in Fig 2.2.1.
- Along with the similarities, there is an important difference between this kind of secondary key index and the card catalog in a library. In a library, once you have the catalog number you can usually go directly to the stacks to find the book since the books are arranged in order by catalog number. In other words, the books are sorted by primary key. The actual data records in our file, on the other hand, are entry sequenced. Consequently, after consulting the composer index to find the Label ID, you must consult one additional index, our primary key index, to find the actual byte offset of the record that has this particular Label Id.
- The procedure is summarized in Fig. 2.2.2. Clearly it is possible to relate secondary key references (e.g., Beethoven) directly to a byte offset (211) rather than to a primary key (DG1 8807). However, there are excellent reasons for postponing this binding of a secondary key to a specific address for as long as possible. These reasons become clear as we discuss the way that fundamental file operations such as record deletion and updating are affected by the use of secondary indexes. Record Addition When a

secondary index is present, adding a record to the file means adding a record to the secondary index.

Composer index

Secondary key	Primary key
BEETHOVEN	ANG3795
BEETHOVEN	DG139201
BEETHOVEN	DG18807
BEETHOVEN	RCA2626
COREA	WAR23699
DVORAK	COL31809
PROKOFIEV	LON2312
RIMSKY-KORSAKOV	MER750 16
SPRINGSTEEN	COL38358
SWEET HONEY IN THE R	FF245

Fig 2.2.1 Secondary key index organized by composer.

```
PROCEDURE   search_on_secondary ( KEY )
        search  for KEY in the secondary index
        once the correct secondary index record is found , set LABEL_ID to the
        primary key value in the record's reference field
        call retrieve_record (LABEL_ID ) to get the data record

end PROCEDURE
```

Fig 2.2.2: Search_on_secondary: an algorithm to retrieve a single record from Datafile through a secondary key index .

Retrieval Using Combinations of Secondary Keys

- The search for records by multiple keys can be done on multiple index, with the combination of index entries defining the records matching the key combination.

- If two keys are to be combined, a list of entries from each key index is retrieved.
- For an "or" combination of keys, the lists are merged.
- I.e., any entry found in either list matches the search.
- For an "and" combination of keys, the lists are matched.
- I.e., only entries found in both lists match the search.

Improving the Secondary Index Structure: Inverted Lists

Inverted list: An index in which the reference field is the head pointer of a linked list of reference items.

Selective Indexes: An index which contains keys for only part of the records in a data file.

Title index

Secondary key	**Primary key**
COQ D'OR SUITE	MER75016
GOOD NEWS	FF245
NEBRASKA	COL38358
QUARTET IN C SHARP	M RCA2626
ROMEO AND JULIET	LON2312
SYMPHONY NO. 9	ANG3795
SYMPHONY NO. 9	COL3 1809
SYMPHONY NO. 9	DG18807
TOUCHSTONE	WAR23699
VIOLIN CONCERTO	DG139201

Fig 2.2.3 Secondary key index organized by recording title.

The secondary index structures that we have developed so far result in two distinct difficulties :

1.We have to rearrange the index file every time a new record is added to the file, even if the new record is for an existing secondary key.

For example, if we add another recording of Beethoven's Symphony No. 9 to our collection, both the composer and title indexes would have to be rearranged, even though both indexes already contain entries for secondary keys (but not the Label IDs) that are being added.

2. If there are duplicate secondary keys, the secondary key field is repeated for each entry. This wastes space, making the files larger than necessary. Larger index files are less likely to be able to fit in electronic memory.

2.2 FILES FUNCTIONS USED IN PROJECT:

File handling :-

Many real-life problem handle large volume of data and ,in such situations, we need to use some devices such as Floppy disk or hard disk to store the data. The data is stored in these devices 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.

Class for file stream operation :-

The I/O system of CPP contain a set of classes that define the file handling methods. This include ifstream , ofstream and fstream. These classes are derived from fstream base and the corresponding iostream class, the classes ,designed manage the disk ,are declared in fstream and therefore we must include this file in any program that uses file.

Opening file Syntax:-

File stream-class stream object;

Stream object .open("filename,ios::mode);

Mode for opening the file:-

1) ios ::app ->To append at the end of file

2)ios::out->it open file in write only mode.

3) ios::in->it open file in read only mode.

Read file syntax:-

Stream –object .read((char*)&class-object,sizeof(class-object));

Write files syntax:

Stream -object.write((char*)&class-object,sizeof(class -object));

- **f.close():-** It closes the stream. All buffers associated with the stream are flushed before closing. System allocates buffers are freed upon closing. Buffers are assigned with setbuf are not automatically freed.

Chapter-3

PSEUDOCODE

3.1 INSERTION FUNCTION:-

```
void addStudent() {

    Student student;

    cout << "\n";

    cout << "\n\tEnter student name : ";

    cin.get();

    getline(cin, student.name);

    cout << "\n\tEnter student course : ";

    cin >> student.course;

    cout << "\n\tEnter student Ph. No : ";

    cin >> student.phno;

    cout << "\n\tEnter student location : ";

    cin >> student.loc;

    cout << "\n";

    ID++;

    ofstream write;

    write.open("student.txt", ios::app);

    write << "\n" << ID;

    write << "\n" << student.name ;

    write << "\n" << student.course ;
    write << "\n" << student.phno ;
    write << "\n" << student.loc;
    write.close();
    write.open("id.txt");
    write << ID;
    write.close();
    cout << "\n";
    cout << "\n\tData save to file";
    cout << "\n";
}
```

3.2 DELETE FUNCTION:-

```
void deleteData() {
    int id = searchData();
    cout << "\n";
    cout << "\n\tYou want to delete record (y/n) : ";
    char choice;
    cin >> choice;
    if (choice == 'y') {
        Student student;
        ofstream tempFile;
        tempFile.open("temp.txt");
        ifstream read;
        read.open("student.txt");
```

```
while (!read.eof()) {
    read >> student.id;
    read.ignore();
    getline(read, student.name);
    read >> student.course;
    read >> student.phno;
    read >> student.loc;
    cout << "\n";
    if (student.id != id) {
        tempFile << "\n" << student.id;
        tempFile << "\n" << student.name;
        tempFile << "\n" << student.course;
        tempFile << "\n" << student.phno;
        tempFile << "\n" << student.loc;
    }
}
read.close();
tempFile.close();
remove("student.txt");
rename("temp.txt", "student.txt");
cout << "\n";
cout << "\n\tData deleted successfuly";
cout << "\n";
```

```
    }
    else {
        cout << "\n";
        cout << "\n\tRecord not deleted";
        cout << "\n";
    }
}
```

3.3 DISPLAY FUNCTION:-

```
void print(Student s) {
    cout << "\n";
    cout << "\n\t---Stuent Data---";
    cout << "\n\tID is : " << s.id;
    cout << "\n\tName is : " << s.name;
    cout << "\n\tCourse is : " << s.course;
    cout << "\n\tPh. No. is : " << s.phno;
    cout << "\n\tLocation is : " << s.loc;
    cout << "\n";
}
```

3.4 SEARCH FUNCTION:-

```
int searchData() {
    int id;
    cout << "\n";
    cout << "\n\tEnter student id want to search : ";
    cin >> id;

    cout << "\n";

    Student student;

    ifstream read;

    read.open("student.txt");

    while (!read.eof()) {

        read >> student.id;

        read.ignore();

        getline(read, student.name);

        read >> student.course;

        read >> student.phno;

        read >> student.loc;

        if (student.id == id) {

            print(student);

            return id;

        }

    }

}
```

3.5 DELETE FUNCTION:

```
void deleteData() {

    int id = searchData();

    cout << "\n";

    cout << "\n\tYou want to delete record (y/n) : ";

    char choice;

    cin >> choice;
```

```
if (choice == 'y') {
    Student student;
    ofstream tempFile;
    tempFile.open("temp.txt");
    ifstream read;
    read.open("student.txt");
    while (!read.eof()) {
        read >> student.id;
        read.ignore();
        getline(read, student.name);
        read >> student.course;
        read >> student.phno;
        read >> student.loc;
        cout << "\n";
        if (student.id != id) {
            tempFile << "\n" << student.id;
            tempFile << "\n" << student.name;
            tempFile << "\n" << student.course;
            tempFile << "\n" << student.phno;
            tempFile << "\n" << student.loc;
        }
    }
    read.close();
    tempFile.close();
    remove("student.txt");
```

```
        rename("temp.txt", "student.txt");

        cout << "\n";

        cout << "\n\tData deleted successfuly";

        cout << "\n";

    }

    else {

        cout << "\n";

        cout << "\n\tRecord not deleted";

        cout << "\n";

    }

}
```

3.6 UPDATE FUNCTION:

```
void updateData() {

    int id = searchData();

    cout << "\n";

    cout << "\n\tYou want to update record (y/n) : ";

    char choice;

    cin >> choice;

    if (choice == 'y') {

        Student newData;

        cout << "\n";

        cout << "\n\tEnter student name : ";

        cin.get();

        getline(cin, newData.name);

        cout << "\n\tEnter student course : ";
```

```
cin >> newData.course;
cout << "\n\tEnter student Ph. No. : ";
cin >> newData.phno;
cout << "\n\tEnter student location : ";
cin >> newData.loc;
cout << "\n";
Student student;
ofstream tempFile;
tempFile.open("temp.txt");
ifstream read;
read.open("student.txt");
while (!read.eof()) {
   read >> student.id;
   read.ignore();
   getline(read, student.name);
   read >> student.course;
   read >> student.phno;
   read >> student.loc;
   if (student.id != id) {
      tempFile << "\n" << student.id;
      tempFile << "\n" << student.name;
      tempFile << "\n" << student.course;
      tempFile << "\n" << student.phno;
      tempFile << "\n" << student.loc;
   }
```

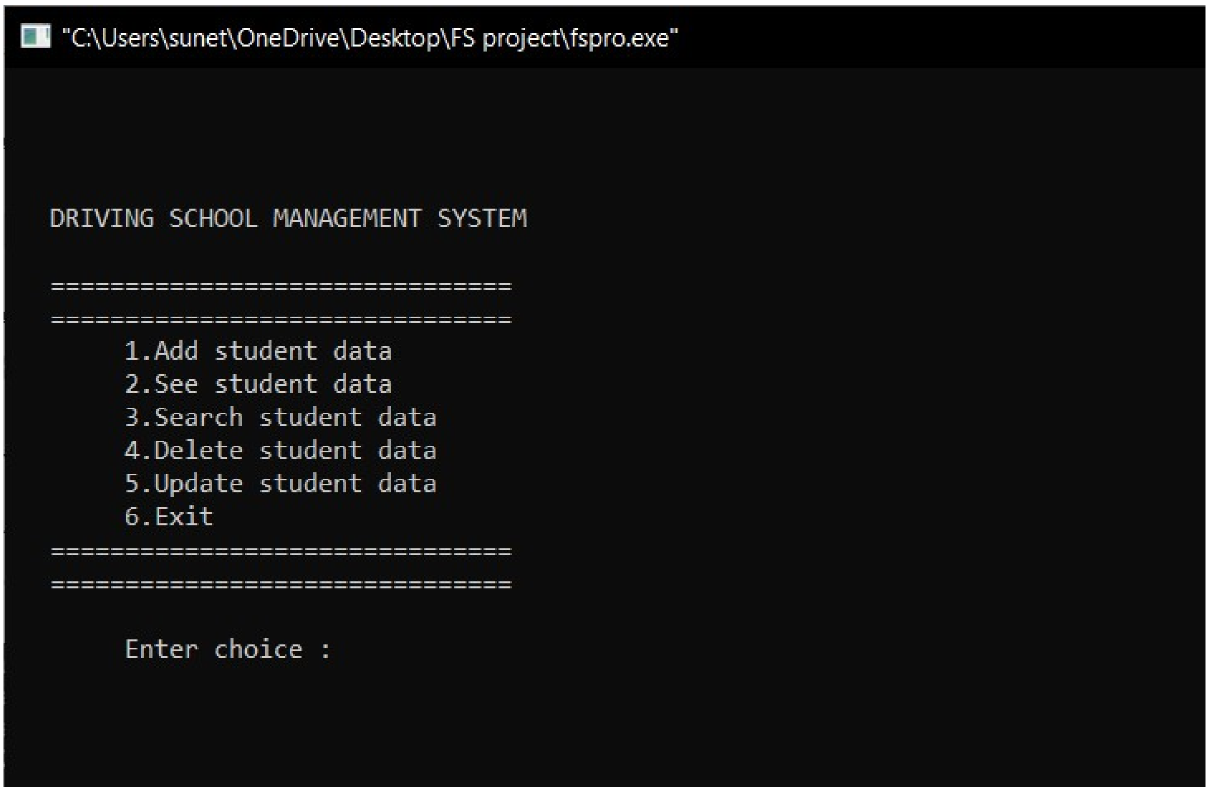
```
            else {
                tempFile << "\n"<< student.id;
                tempFile << "\n"<< newData.name;
                tempFile << "\n"<< newData.course;
                tempFile << "\n" << newData.phno;
                tempFile << "\n"<< newData.loc;
            }
        }
        read.close();
        tempFile.close();
        remove("student.txt");
        rename("temp.txt", "student.txt");
        cout << "\n";
        cout << "\n\tData updated successfuly";
        cout << "\n";
    }
    else {
        cout << "\n";
        cout << "\n\tRecord not deleted";
        cout << "\n";
    }
}
```

Chapter-4

SNAPSHOTS

4.1 WELCOME PAGE :-

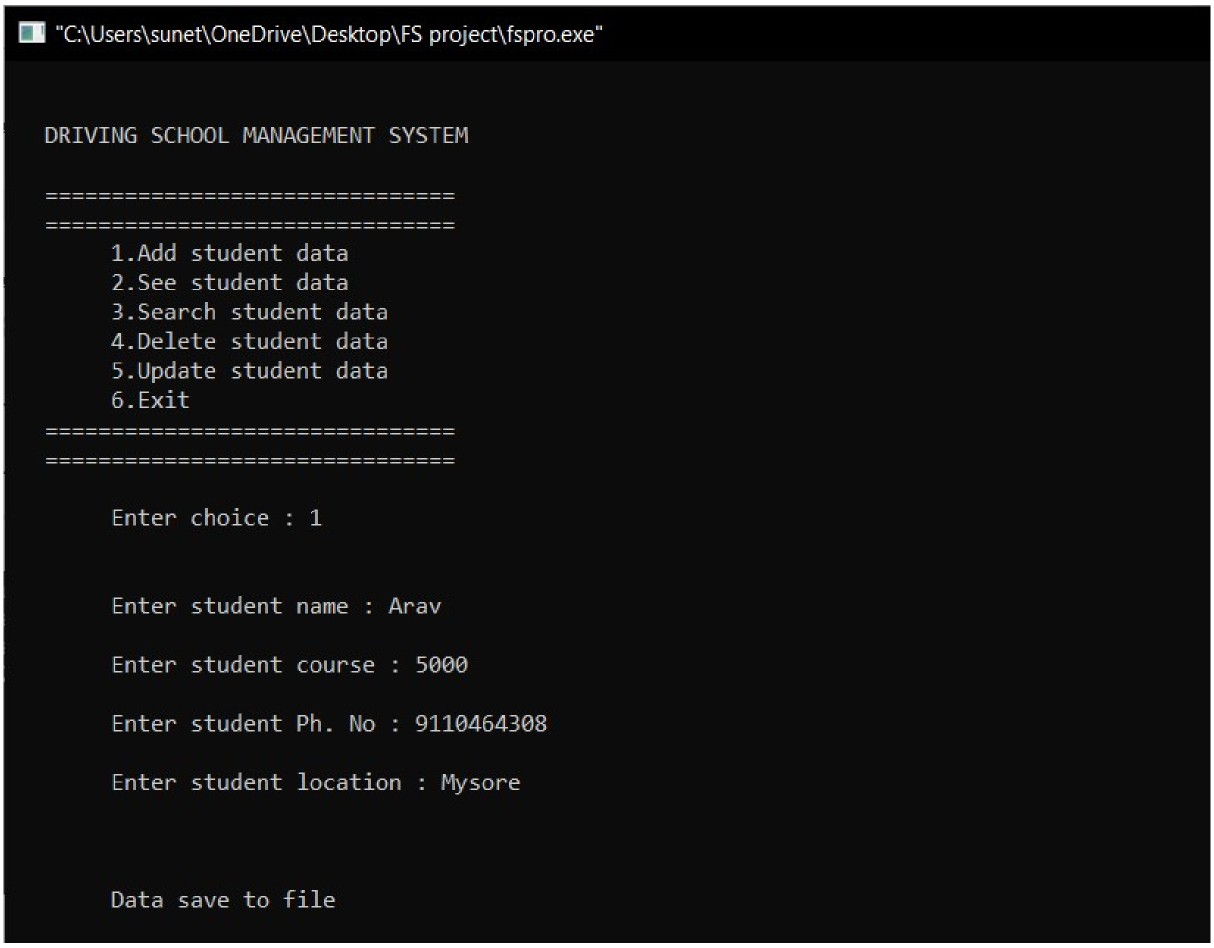
This is the Welcome page where all the title of the project and all the operations such as Add, See, Search, Delete and Update are displayed. The user is required to choose one of the given operations that is to be performed.


"C:\Users\sunet\OneDrive\Desktop\FS project\fspro.exe"

```
DRIVING SCHOOL MANAGEMENT SYSTEM

===============================
===============================
     1.Add student data
     2.See student data
     3.Search student data
     4.Delete student data
     5.Update student data
     6.Exit
===============================
===============================

     Enter choice :
```

4.2 INSERTION OPERATION:-

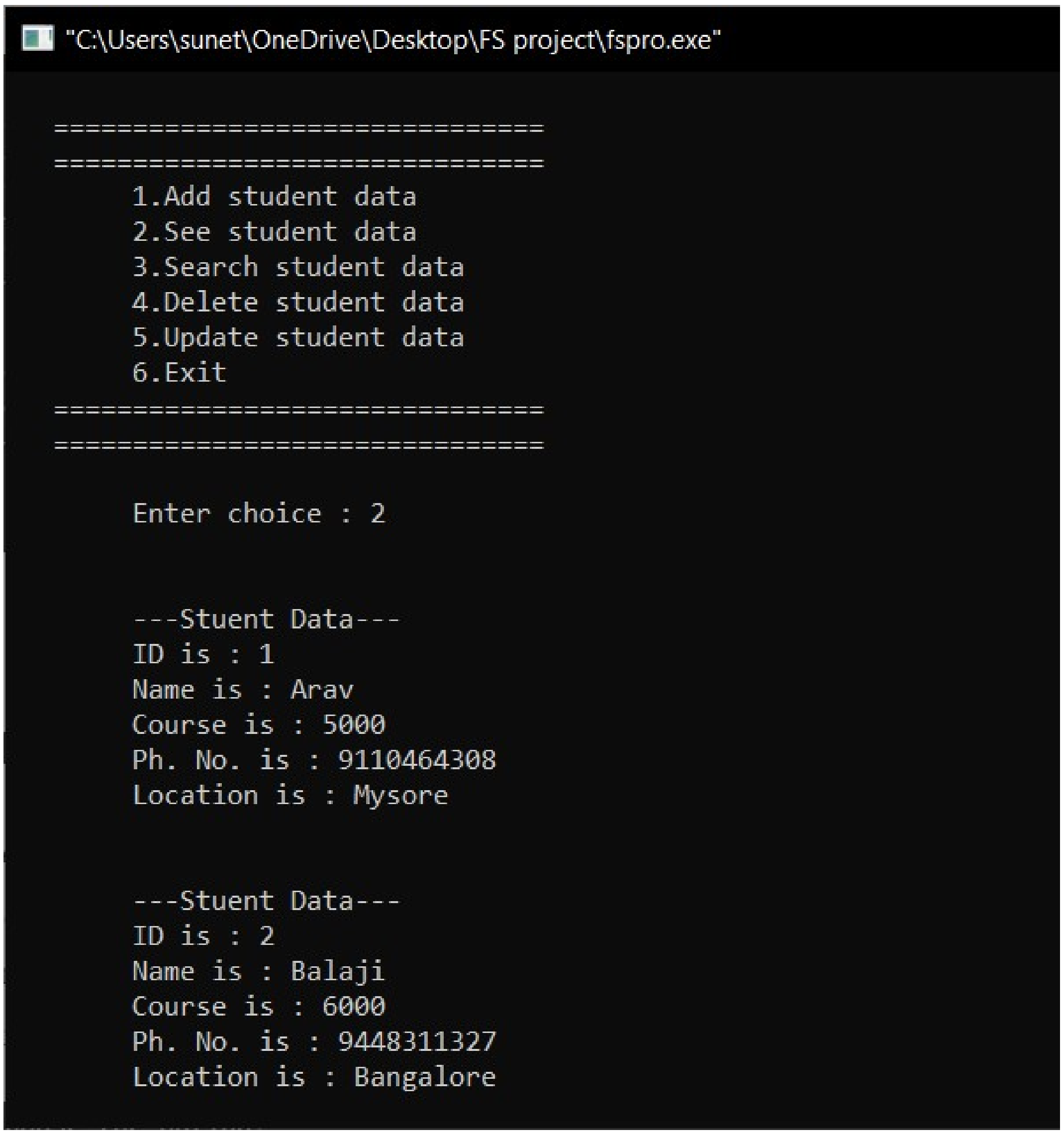
Administrator can insert or add a record by opting the first choice. During Insertion, the user has to enter details like student name, course, phone number and location. Thus, the inserted record is stored.



```
"C:\Users\sunet\OneDrive\Desktop\FS project\fspro.exe"

DRIVING SCHOOL MANAGEMENT SYSTEM

===============================
===============================
     1.Add student data
     2.See student data
     3.Search student data
     4.Delete student data
     5.Update student data
     6.Exit
===============================
===============================

     Enter choice : 1


     Enter student name : Arav

     Enter student course : 5000

     Enter student Ph. No : 9110464308

     Enter student location : Mysore



     Data save to file
```

4.3 DISPLAY OPERATION:-

Once the record is inserted, the next would be displaying the record. It displays the record of previously inserted which includes all the attributes like student name, course, phone number and location.

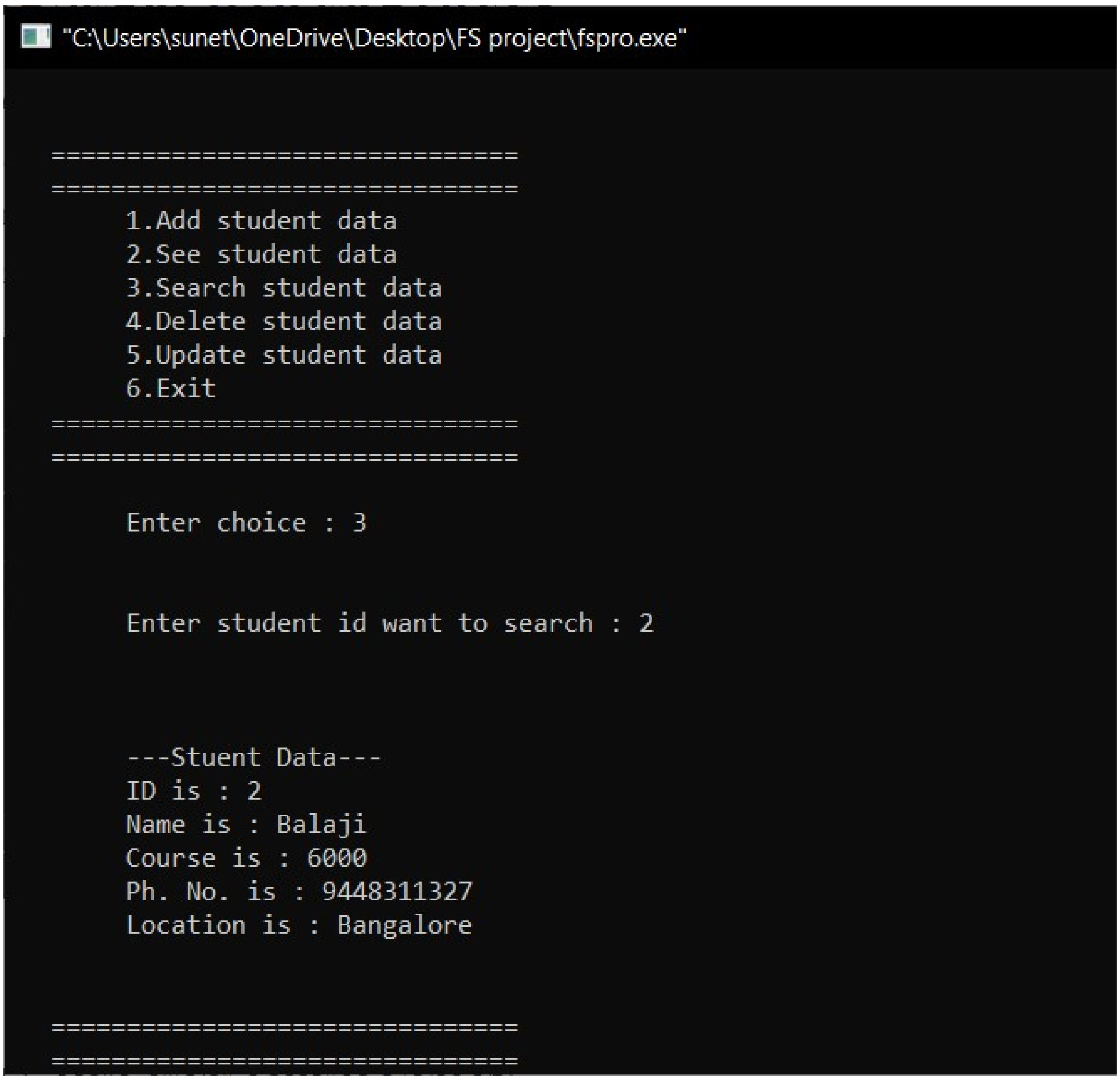


"C:\Users\sunet\OneDrive\Desktop\FS project\fspro.exe"

```
================================
================================
     1.Add student data
     2.See student data
     3.Search student data
     4.Delete student data
     5.Update student data
     6.Exit
================================
================================

     Enter choice : 2


     ---Stuent Data---
     ID is : 1
     Name is : Arav
     Course is : 5000
     Ph. No. is : 9110464308
     Location is : Mysore


     ---Stuent Data---
     ID is : 2
     Name is : Balaji
     Course is : 6000
     Ph. No. is : 9448311327
     Location is : Bangalore
```

4.4 SEARCH OPERATION:-

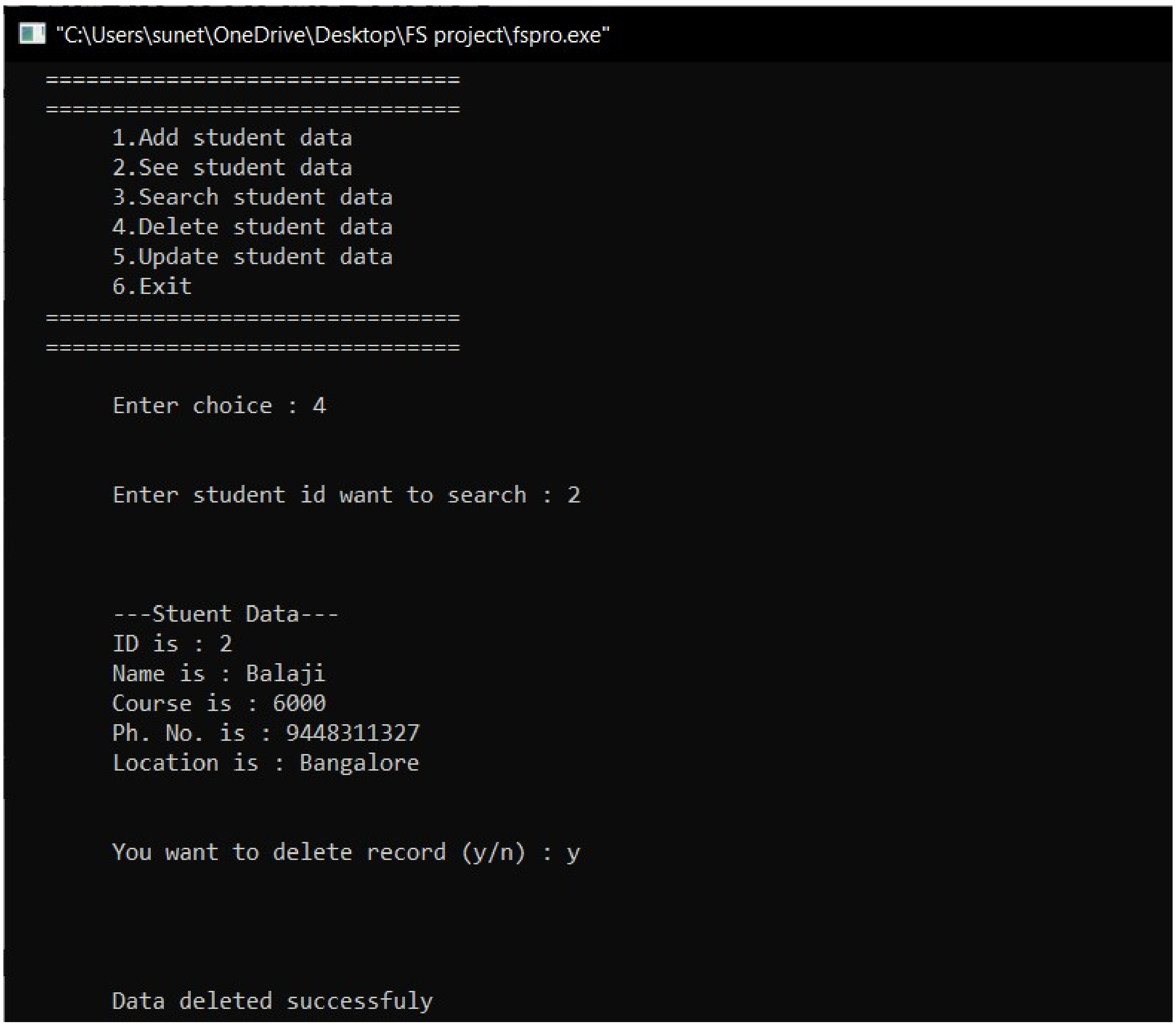
After displaying the record,the next operation would be search opeartion.Here the student record would be searched by his or her id. If the given student id is found then the search operation is succesful and it displays the details of student. If the given student name is not found the search is unsuccessful.



```
"C:\Users\sunet\OneDrive\Desktop\FS project\fspro.exe"

   ================================
   ================================
        1.Add student data
        2.See student data
        3.Search student data
        4.Delete student data
        5.Update student data
        6.Exit
   ================================
   ================================

        Enter choice : 3


        Enter student id want to search : 2



        ---Stuent Data---
        ID is : 2
        Name is : Balaji
        Course is : 6000
        Ph. No. is : 9448311327
        Location is : Bangalore


   ================================
   ================================
```

4.5 DELETE OPERATION:-

Delete operation includes deleting a record with the student id. If the given student id matches, then the record would be deleted and delete operation is successful.

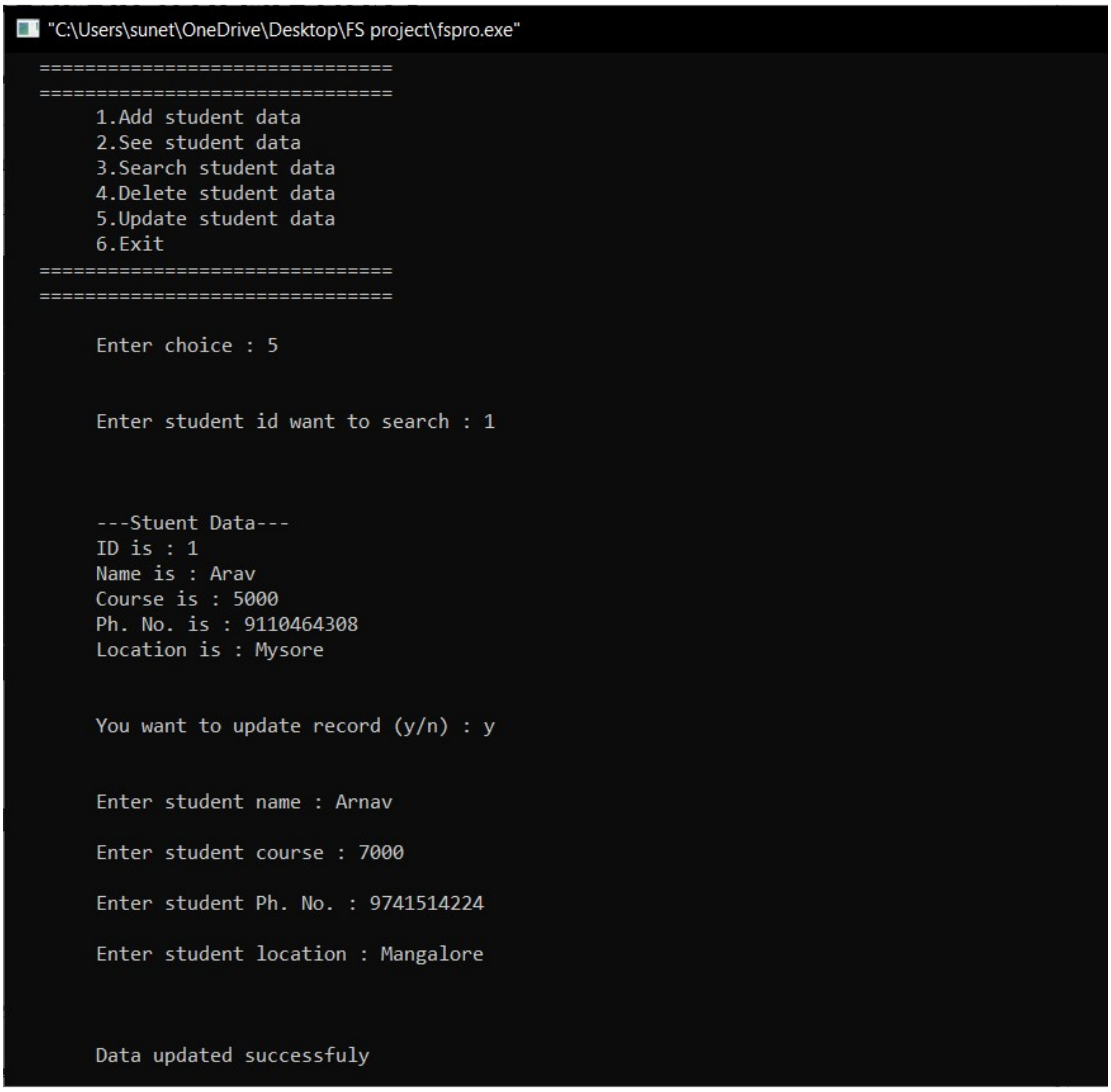


"C:\Users\sunet\OneDrive\Desktop\FS project\fspro.exe"

```
================================
================================
     1.Add student data
     2.See student data
     3.Search student data
     4.Delete student data
     5.Update student data
     6.Exit
================================
================================

     Enter choice : 4


     Enter student id want to search : 2



     ---Stuent Data---
     ID is : 2
     Name is : Balaji
     Course is : 6000
     Ph. No. is : 9448311327
     Location is : Bangalore


     You want to delete record (y/n) : y



     Data deleted successfuly
```

4.6 UPDATE OPERATION:-

After inserting student records, if we want to update the details of aparticular student, then we can do so by using the Update Operation. We search the student by id and then update their details.

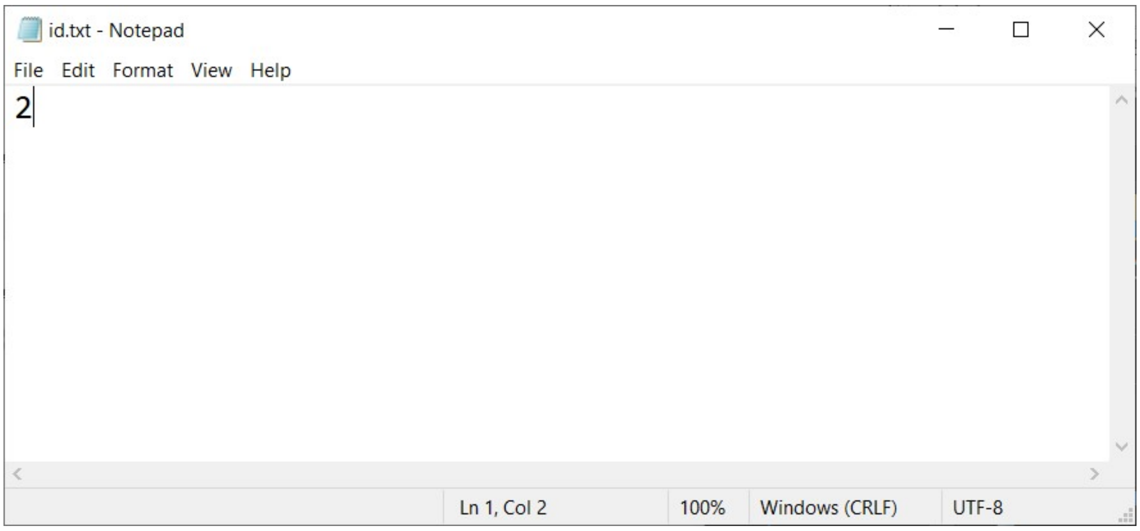


"C:\Users\sunet\OneDrive\Desktop\FS project\fspro.exe"

```
===============================
===============================
     1.Add student data
     2.See student data
     3.Search student data
     4.Delete student data
     5.Update student data
     6.Exit
===============================
===============================

     Enter choice : 5


     Enter student id want to search : 1



     ---Stuent Data---
     ID is : 1
     Name is : Arav
     Course is : 5000
     Ph. No. is : 9110464308
     Location is : Mysore


     You want to update record (y/n) : y


     Enter student name : Arnav

     Enter student course : 7000

     Enter student Ph. No. : 9741514224

     Enter student location : Mangalore


     Data updated successfuly
```

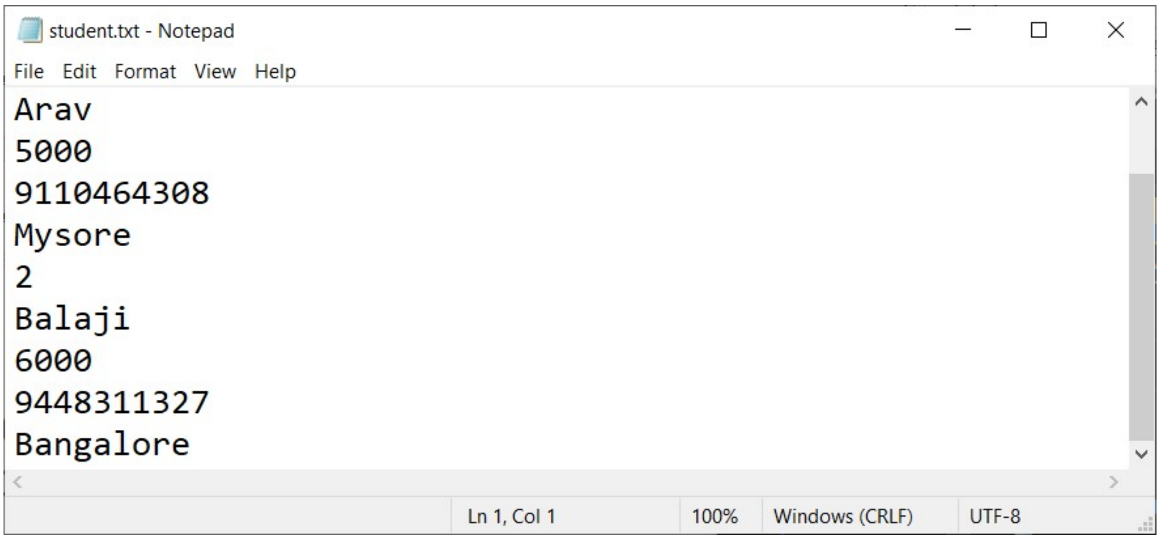
4.7 INDEXES FILES

1: This contains index page where id of the last inserted student is stored**.**



```
2
```

2: This contains details page which includes all the attributes in a sorted manner.



```
Arav
5000
9110464308
Mysore
2
Balaji
6000
9448311327
Bangalore
```

Chapter-5

ADVANTAGES OF FILE STRUCTURES

Advantages of ordered File Organization:

- To find a record in the sequential file is very efficient, because all files are stored in an order.
- It is fast and efficient when dealing with large volumes of data that need to be processed periodically.

Advantages of unordered File Organization:

- insert simple, records added at end of file.
- easier for retrievals a large proportion of records.
- effective for bulk loading data.

Advantages of hashing File Organization:

- Direct Access to the data.
- Hash function or randomizing function.
- Best if equality search is needed on hash-key.

Chapter-6

CONCLUSION

The Mini Project **“Driving School Management System**” is designed in order reduce the burden of maintaining bulk records of all the details in which Inserting, Retrieving, Searching, Updating and Deleting the details are easy when compared to the manual update and storing. This mini project helps in maintaining the student details in an Organized manner and to replace old paper work system. This is to conclude that the project that I undertook was worked upon sincere effort. Most of the requirement are fulfilled up to the mark.

Chapter-7

REFERENCES

1. Reference Book :- Michael J. Folk, Bill Zoellick - File Structures

(1991, Addison Wesley)

2. Website:-

http://www.cppforschool.com/projects.html

http://www.codeincodeblock.com/mini-projects.html