CHAPTER-1

INTRODUCTION

1.1 RECORD

A record is a collection of fields, possibly of differ data types, typically in fixed number and sequence. The fields of a record may also be called members, particularly in oriented programming. Fields may also be called elements, through the risk confusion with the elements of the collection.

For example, a data could be stored as record containing a numeric year field, a month field represented as a string, and a numeric day-of-month field. A personnel record might contain a name, a salary, and a rank. A circle of record contains center and a radius in this instance, the center itself might be represented as point record containing x and y coordinates.

Record are distinguished from arrays be the fact that their number of fields is typically fixed, each field has a name, and that field may have a different type.

A record type is a data type that describes such values and variable. Most modern computer languages allow the programmer to define new data record types. The definition includes specifying the data types of each field and the identified by which it can be accessed. In type theory, product types are generally preferred due to the simplicity, but the proper record types are studied in languages such as System F-sub. Since type-theoretical records may contain first-class function-typed fields in addition to data, they can be express many features of object-oriented programming.

Records can exist in any storage medium, including main memory and mass storage devices such as magnetic tapes or hard disk. Records are the fundamental component of most Structure, especially liked data structure. Many computer files are organized as array of logical records, often grouped into larger physical records or block for efficiency.

1.1.1 RECORD DEFINITION

A record can be defined as set of fields that belongs together when the file is viewed in term of higher level of organization. For example we can define

structure as follows:

```
Struct date
    {
     int  year; int    month; int day;
    };
```

1.2 TYPES OF RECORDS

1.2.1 Fixed length records

1.2.2 Variable length records

1.2.1 Fixed Length Record:

Each record is stored in fixed size. The size can be determined by adding the maximum space occupied by each field and some space reserved for the header data.

1.2.2 Variable Length Record and Keys

In many applications information associated with a key varies in length. Secondary indexes that reference inverted lists are one of excellent example for this. One way to handle this variability is to place the associated information in a separate, variable-length record file.

Another approach is to allow a variable number of keys and records in a B-tree page.

1.3 Relative Record Number (RRN)

It's an integer that is used to represent a fixed length record. RRN starts with followed by 1, 2, 3…… where RRN 0 represent 1st record. RRN 1 represents-2nd record and so on.

The Relative Record Number identifies which position in a file the record is in. If, for example, a record has the RRN of 10 there does not have to be nine records before it. When a record is deleted its space in the file is retained, and it is not freed until the file is reorganized using the RGZPFM command, or if the file is set to reuse deleted records a new record is added to the file. We can use RRN options on most commands that access data sets. RRN specifies that the record identification field contains the relative record number of the record to be accessed. RRN starts with 0 followed by 1, 2, 3…. Where RRN 0 represents first record, RRN 1 represents second record and so on.

If length of a record is N and RRN of that record is R then address of that record is calculated as R*N.

1.4 Relative record Data Set (RRDS)

A relative record data set is a type of data set organization used by the VSAM computer data storage system. Records are accessed based on their ordinal position in the file (RRN, relative record number). For example, the desired record to be accessed might be the 42nd record in the file out of 999 total.

The concept of RRDS is similar to sequential access method, but it can access with data in random access and dynamic access.

1.5 RRDS Structure

An RRDS consist of data records in sequence, with the record number indicating the records the logical position in the data set. A program can access records randomly using this positional number or access record sequentially. But unlike a Key Sequenced Data Set, an RRDS has no keys, so the program cannot access records by key values. Keys may be used to access records in an RRDS by defining an alternate index.

1.6 Key Sequenced Data Set (KSDS)

A Key Sequenced Data Set (KSDS) is a type of data set used by the IBM VSAM computer data storage system. Each record in a KSDS data file is embedded with unique key. A KSDS consists of two parts, the data component and a separate index file known as the index component which allows the system to physically locate the record in the data file by its key value. Together, the data and index components are called a cluster.

1.7 Entry Sequenced Data Set (ESDS)

An Entry Sequenced Data Set (ESDS) is a type of data set used by the VSAM computer data storage system. Records are accessed based on their sequential order that is the order in which they were written to the file. Which means that accessing a particular record involves searching all the records sequentially until it is located, or by using a relative physical address(Relative Byte Address, RBA),that is the number of bytes from the beginning of file to start reading.

1.8 Direct Access

- With direct access, we can seek directly to the beginning of the record.

Time is in O (1) for n records.

- Index may be the beginning of the required record.
- We know the relative record number RRN

First record has RRN 0, the next has RRN 1, and so forth.

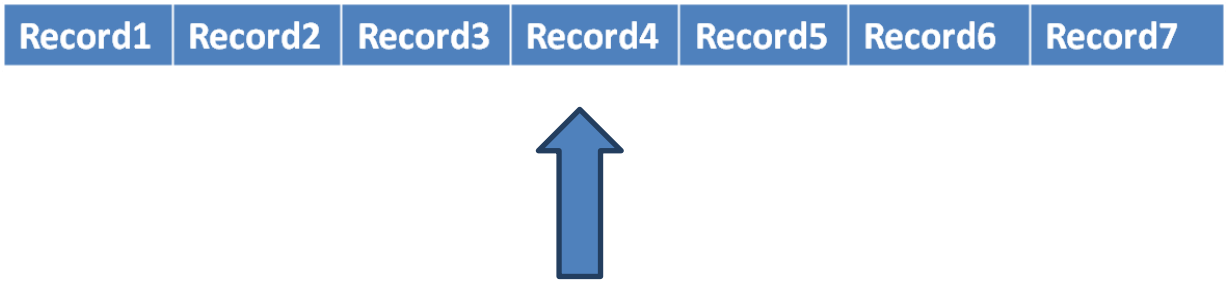


Fig 1.1 DIRECT ACCESS

1.9 Direct Access by RRN

RRN is not useful when working with variable length

record The access is still sequential.

To get the record we want, we have to read sequentially through the file, counting records as we go.

Time is in O (n).

With Fixed length record RRN is useful,

If the records are all the same length, we can use records RRN to calculate byte offset of the start of the record relative to the start of the file.

Example: RRN=30, record length=100bytes: byte offset=30*100=3000

CHAPTER-2

2.1 DESIGN OF PRISON MANAGEMENT SYSTEM

Prison management concerns in developing a prison management system that is the collection of details of the prisoner and his details.

The records in the program are **inmate, prison.**

2.1.1 INMATE RECORDS

In the inmate record, the fields are prison_id, inmate_id, inmate_name, nationalism, crime, health_status, datein

prison_id	inmate_id	inmate_name	nationalism	crime	health_status	datein

2.1.2 PRISON RECORDS

In the prison records, the fields are prison_id, prison_name.

prison_id	prison_name

CHAPTER-3

ALGORITHM AND IMPLEMENTATION

This chapter algorithm and implementation of the Prison Management System using file structure concept of relative record number.

3.1 ALGORITHM

3.1.1 MAIN ( ) FUNCTION

It will give the choice to select among inmate, prison. Once the choice is selected it gives further choice to insert a record or search a record.

1. In a while loop

   Print Prison management system using

   rrn Print enter the choice for inmate,

   prison details Switch (expression)

   Case 1: In a do while loop

   Print prison operations.

   If the choice is 1, call read(), pack(), write_to_file()

   method. If the choice is 2, call display_rrn_list()

   If the choice is 3, call create_rrn_list(), search method.

   If the choice is 4, call remove_rrn_list(), remove

   method. If the choice is 5, go to home.

   Case 2: In a do while loop

   Print inmate operations.

   If the choice is 1, call read1(), pack1(), write_to_file1()

   method. If the choice is 2, call display_rrn_list1()

   If the choice is 3, call create_rrn_list1(), search method.

   If the choice is 4, call remove_rrn_list1(), remove

   method. If the choice is 5, go to home.

   Case 3:Exit

2. End while
3. End

read()

Read the bus details by asking the input from the user like:

1. Read `prison_id, prison_name.`
2. End

pack( )

Store all the volumes of bus details

1. Erase the buffer garbage value

2. Store `prison_id, prison_name` in the buffer along with the delimiter between them. End of the record is identified by $ symbol.

3. End

write_to_file( )

1.Open a txt file in the output and in append mode.

2. Redirect contents of buffer to file.

3. Close the file.

4. End

Create_rrn_list()

1. Initialize rrn to 1 and no_of_records to 0.

2. Open a file filename.txt in input mode.

3. Print RRN no Record.

4. In do while loop

a. get the current position of pointer, store in pos.

b. and initialize pos to rrn_array.

c. get the contents of buffer.

d. if end of the file reached, break.

e. print rrn and buffer

f. increment rrn

5. Until true.

6. End do while loop.

7. close the file.

8. End.

Search() function:

1. If rrn_no is negative or zero or greater than no_of_recs then print record not found.

2. Else

    a. Put the volume of rrn_array to pos.

    b. Open the file prison.txt in input mode.

    c. move the get pointer to beginning of pos.

    d. get the content of buffer.

    e. print the buffer content.

    f. close the file.

3. End if

4. End

Display() function

1. Creates the RRN for each record in prison.txt file.
2. And unpacks it in the console.

Remove() function

1. First rrn is created for the set of records in prison.txt file.
2. Enter the rrn no corresponding to the record which has to be deleted.
3. Call search function to search the rrn no.

Have the * value corresponding to the record that has to search all data.

CHAPTER 4

RESULT ANALYSIS AND OBSERVATION

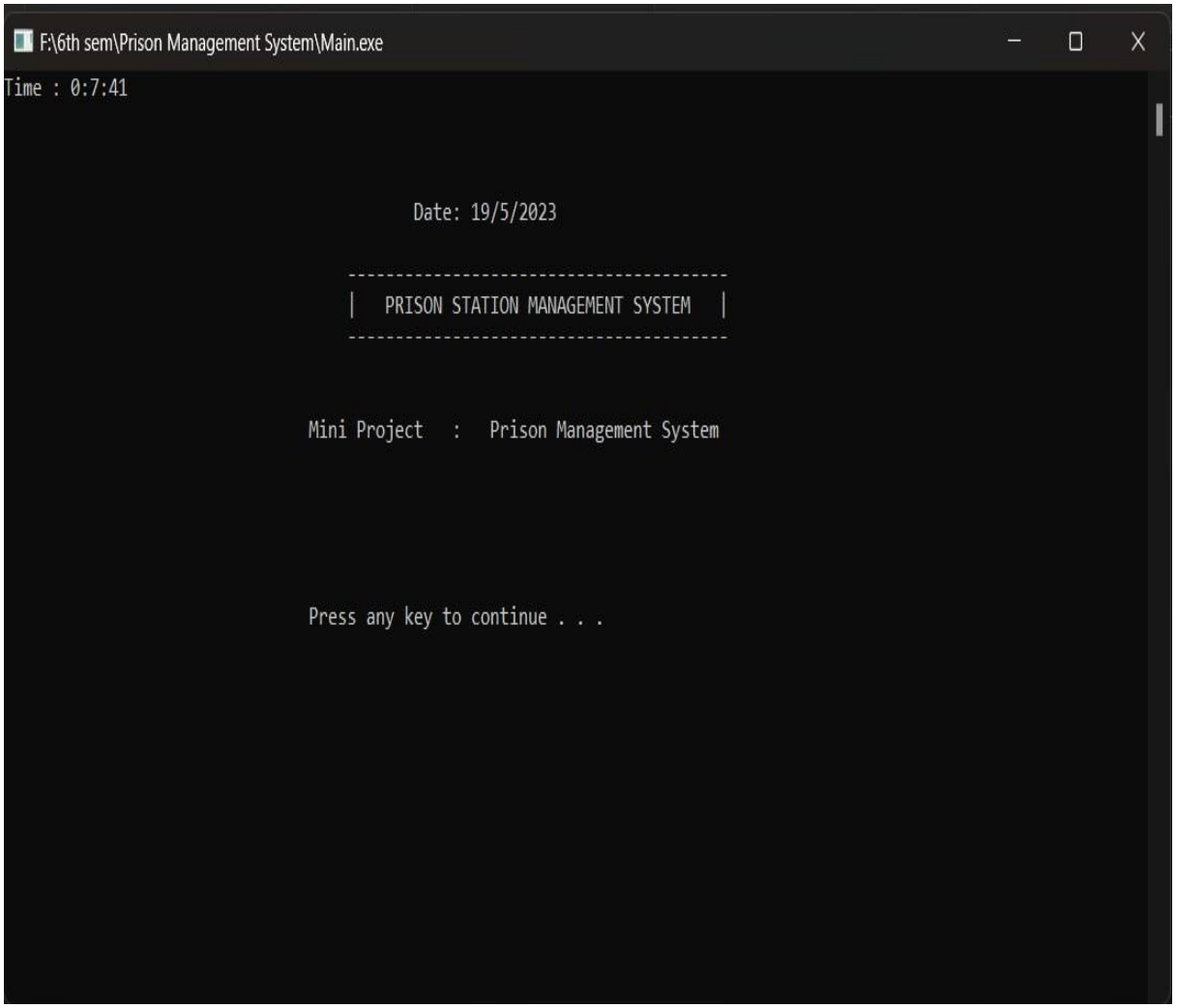


Fig 1: Main Menu

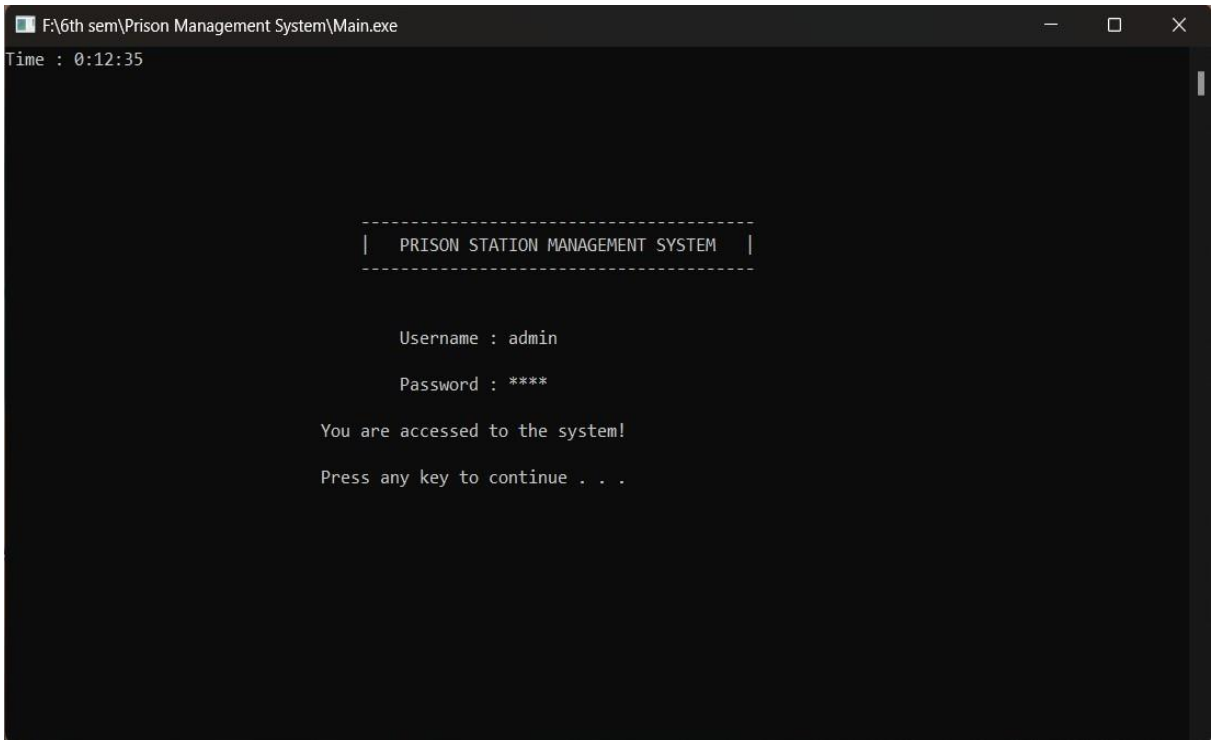


Fig 2: Prison User Authentication

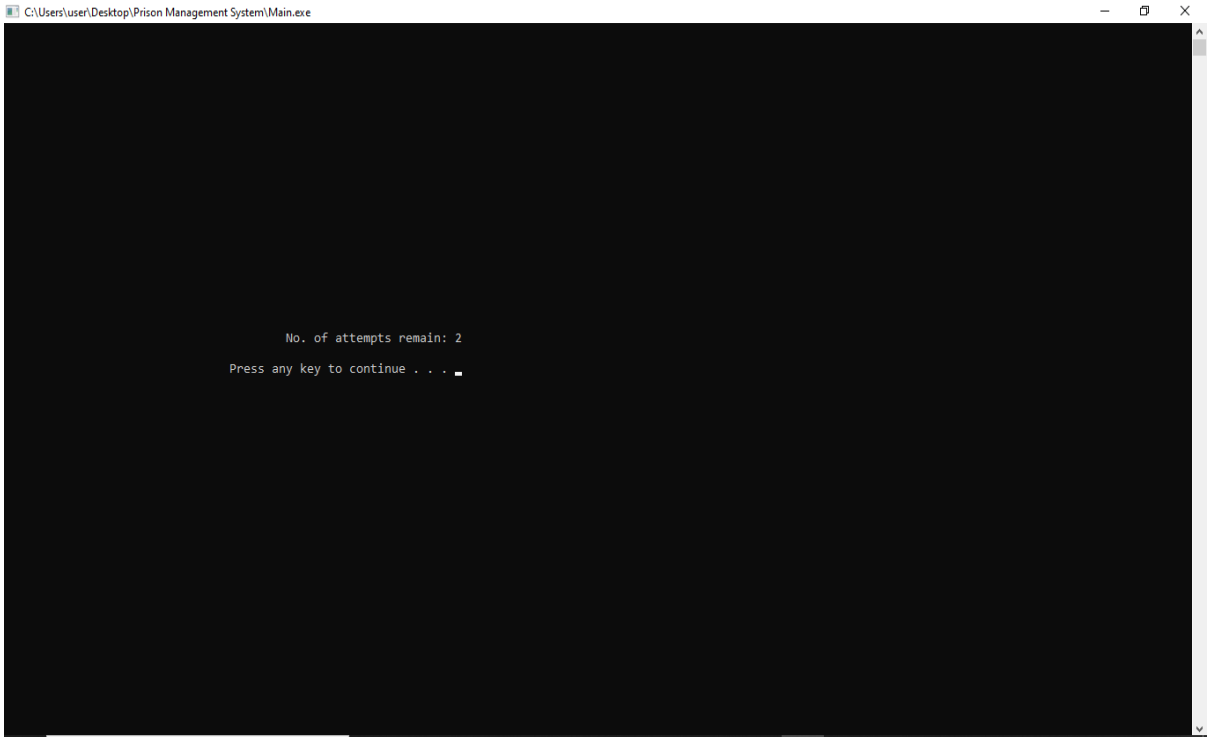


Fig 3: Wrong User Authentication

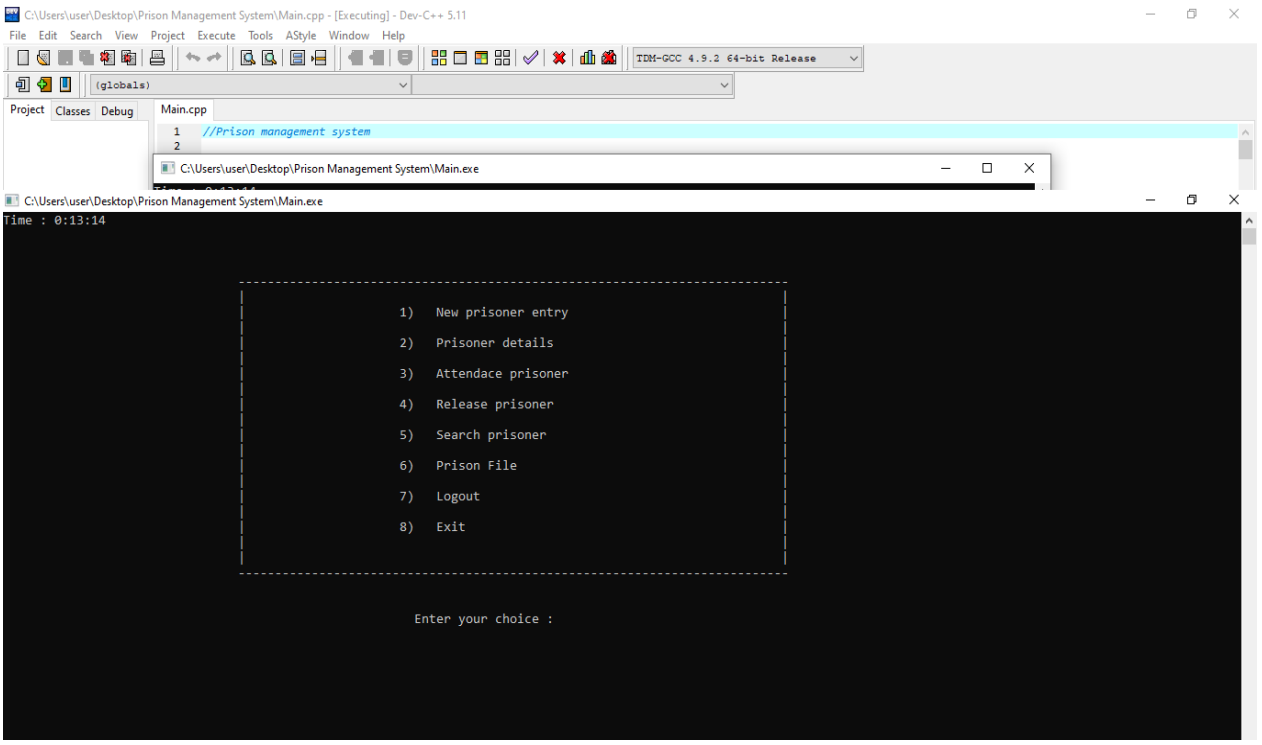


Fig 4: After User Authentication

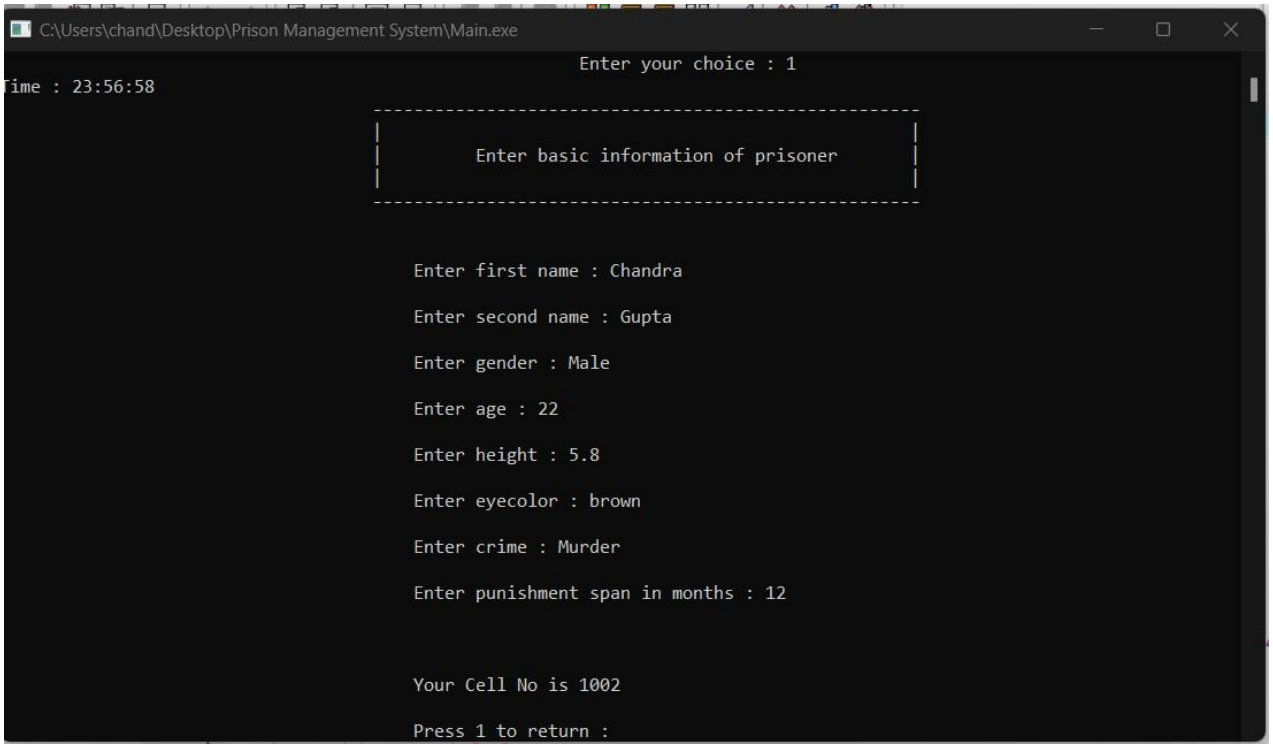


Fig 5: Insertion

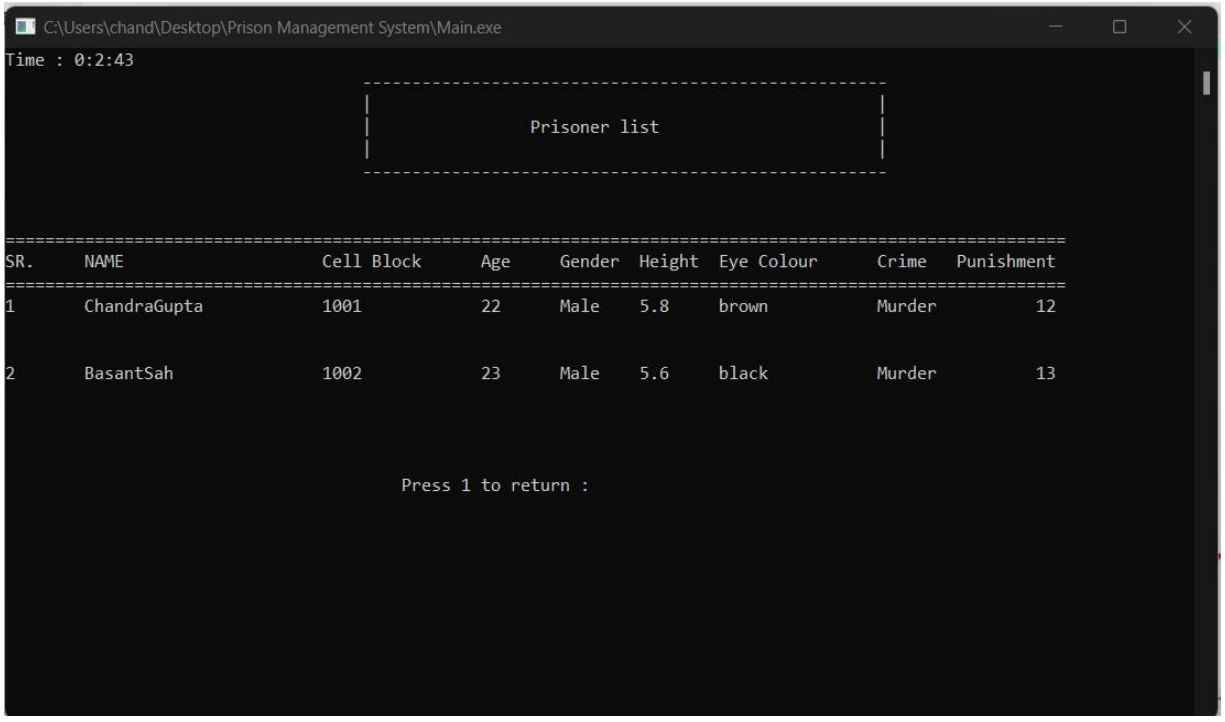


Fig 6: Display Prisoner Details

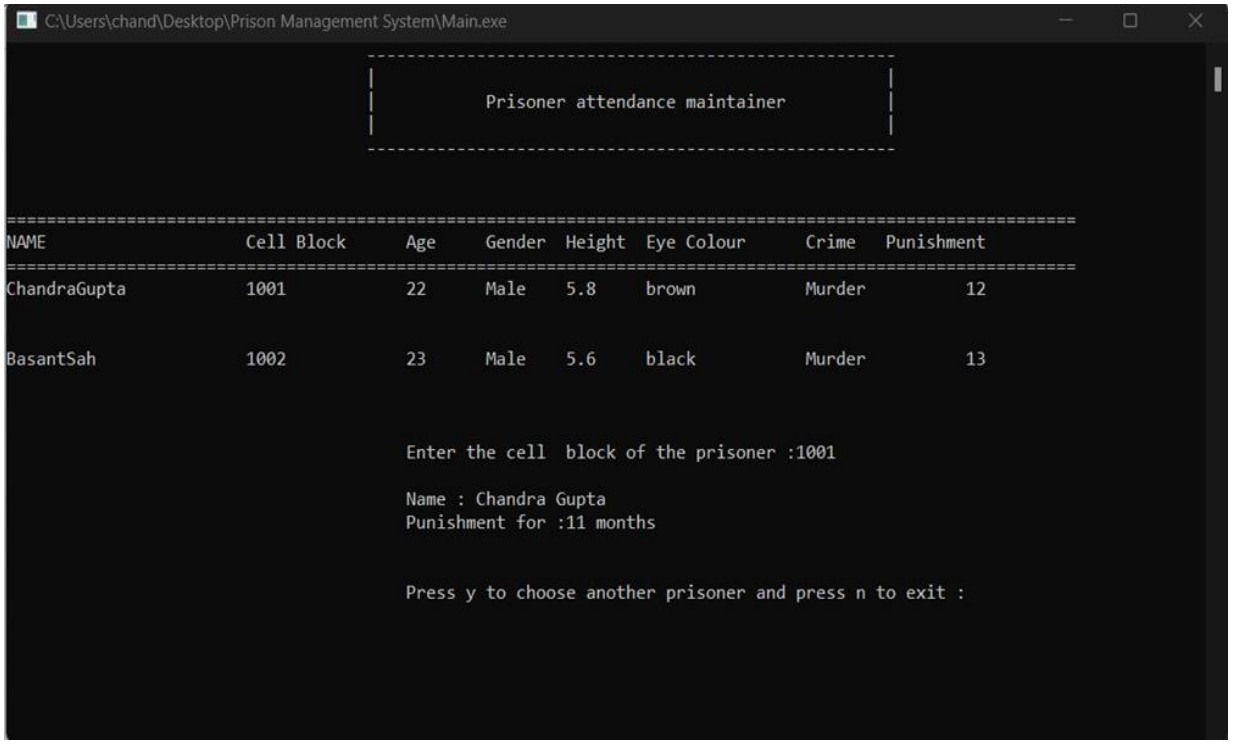


Fig 7: Display Attendance Maintainer

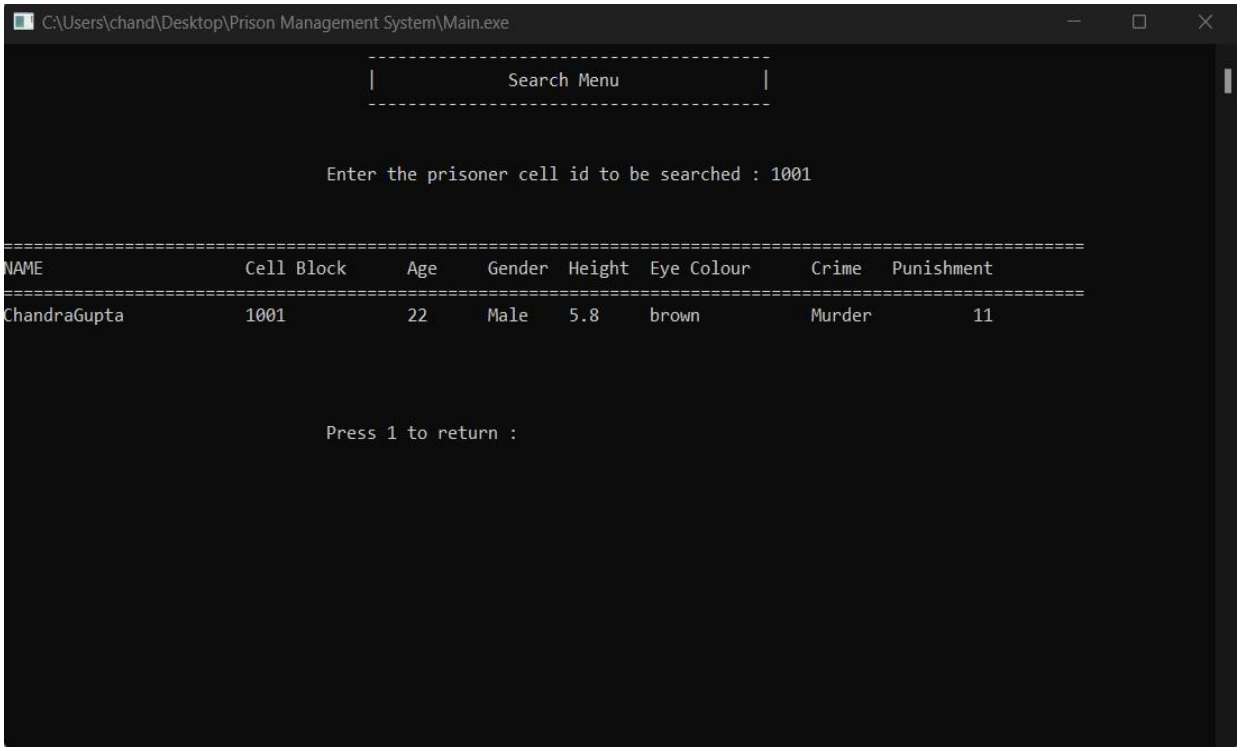


Fig 8: Searching the prisoner

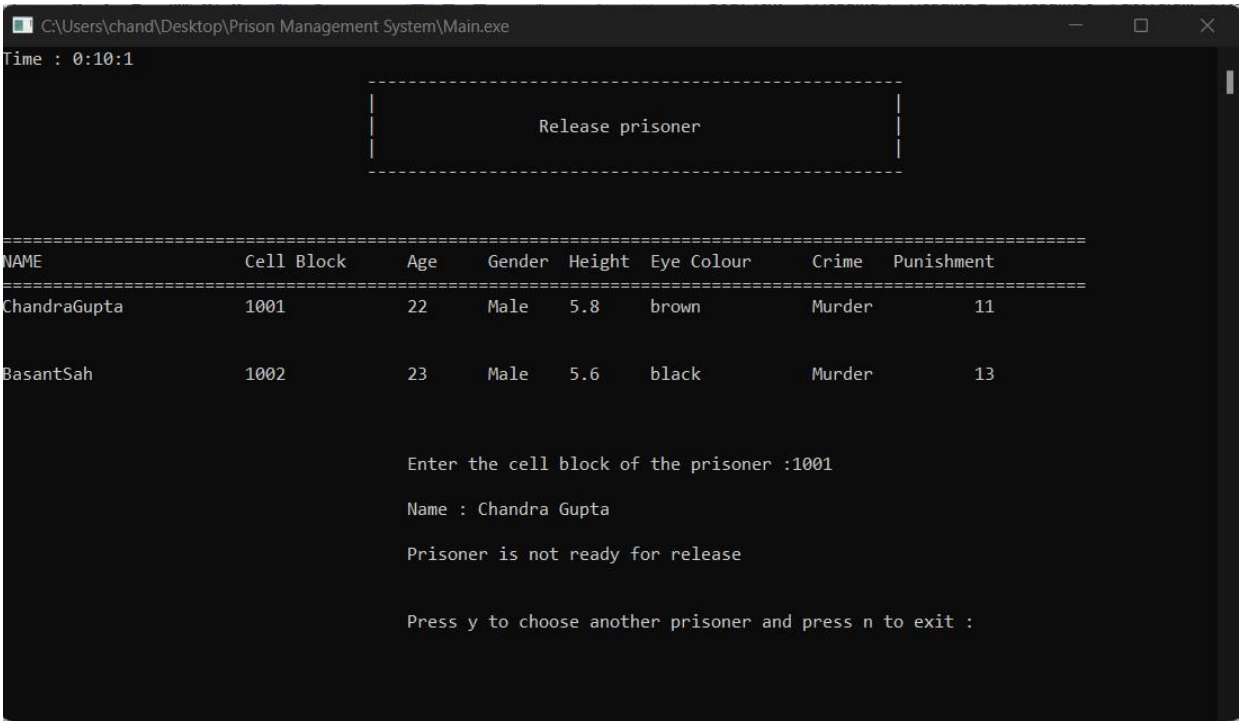
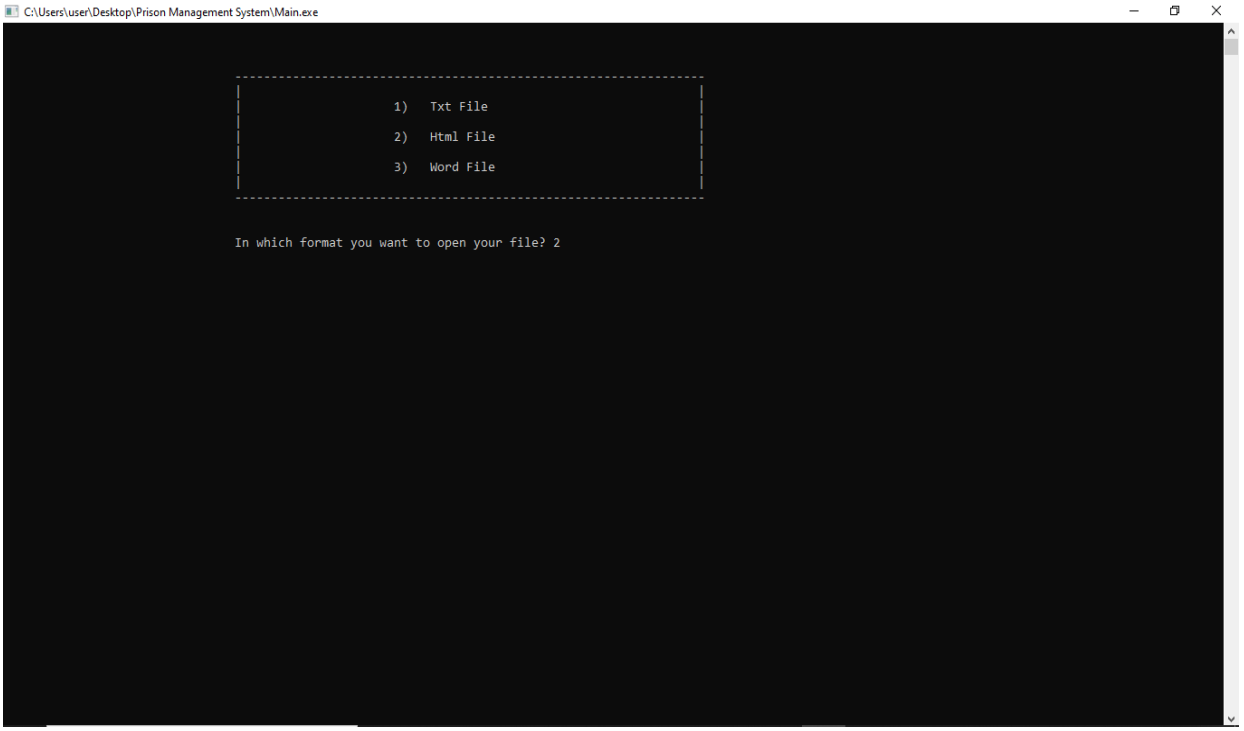


Fig 9: Release prisoner



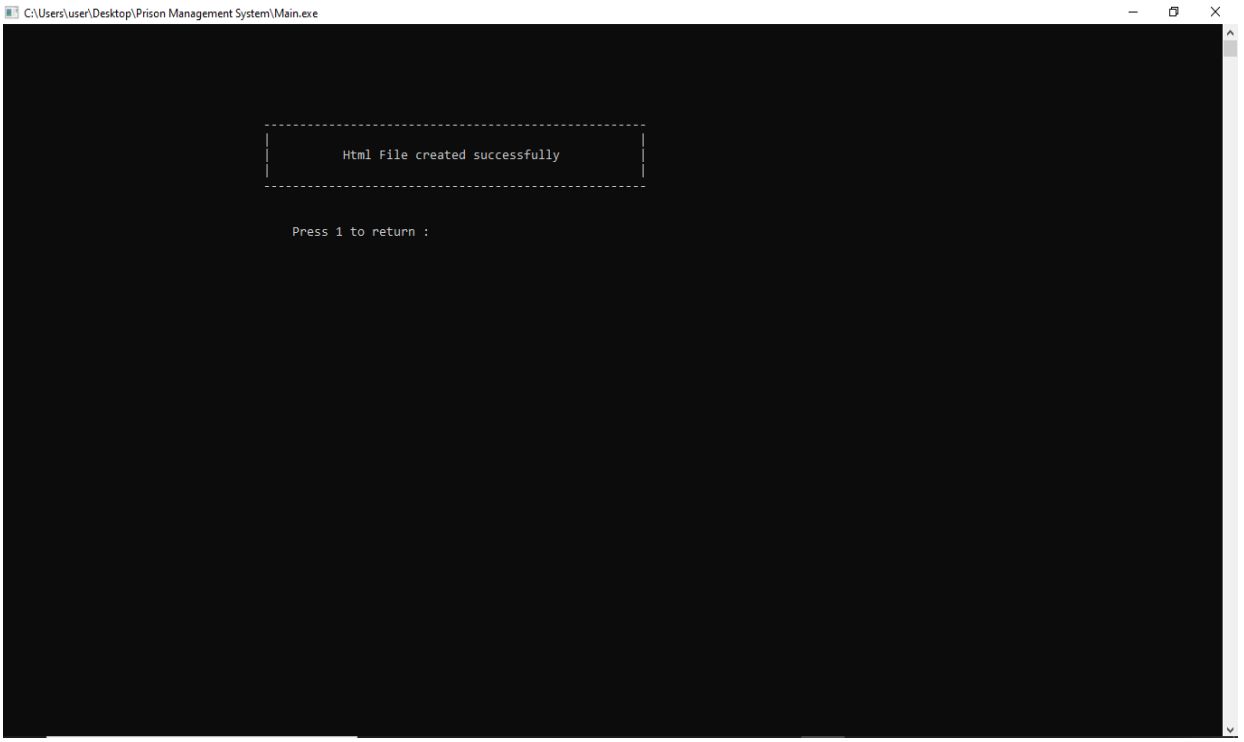


Fig 10: Saving the prisoner's files

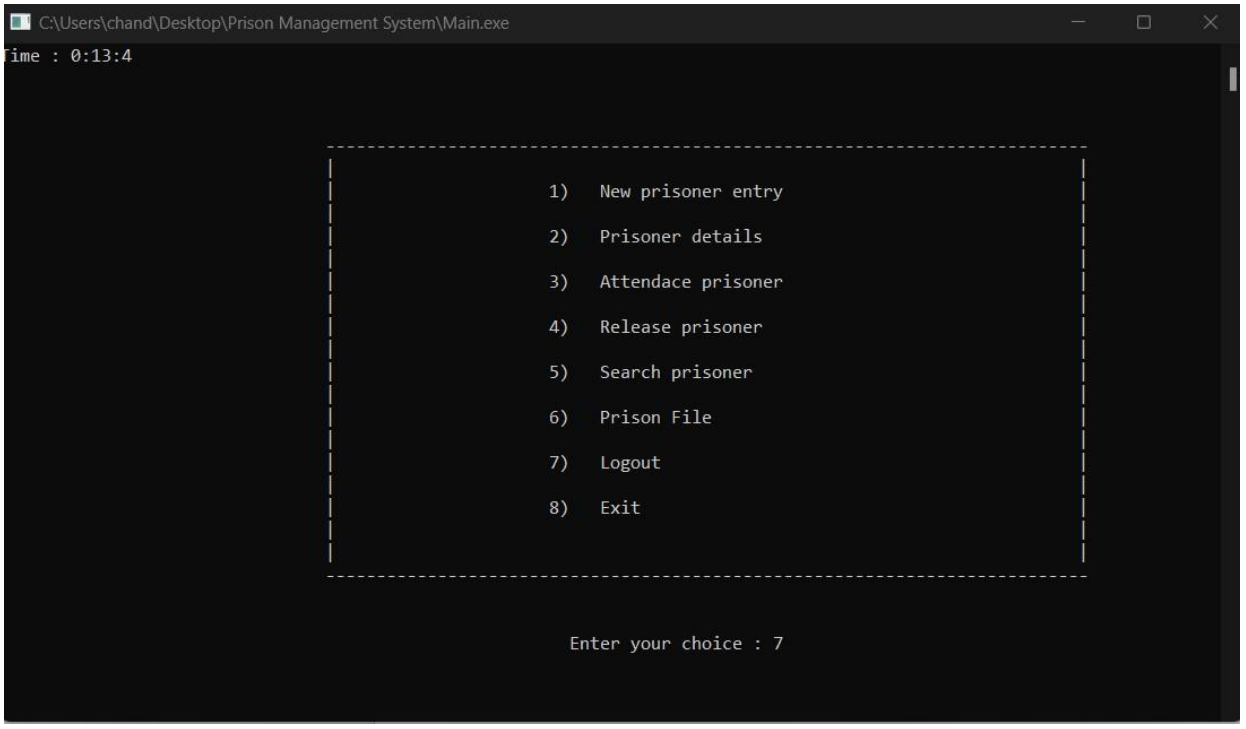


Fig 11: Select Logout

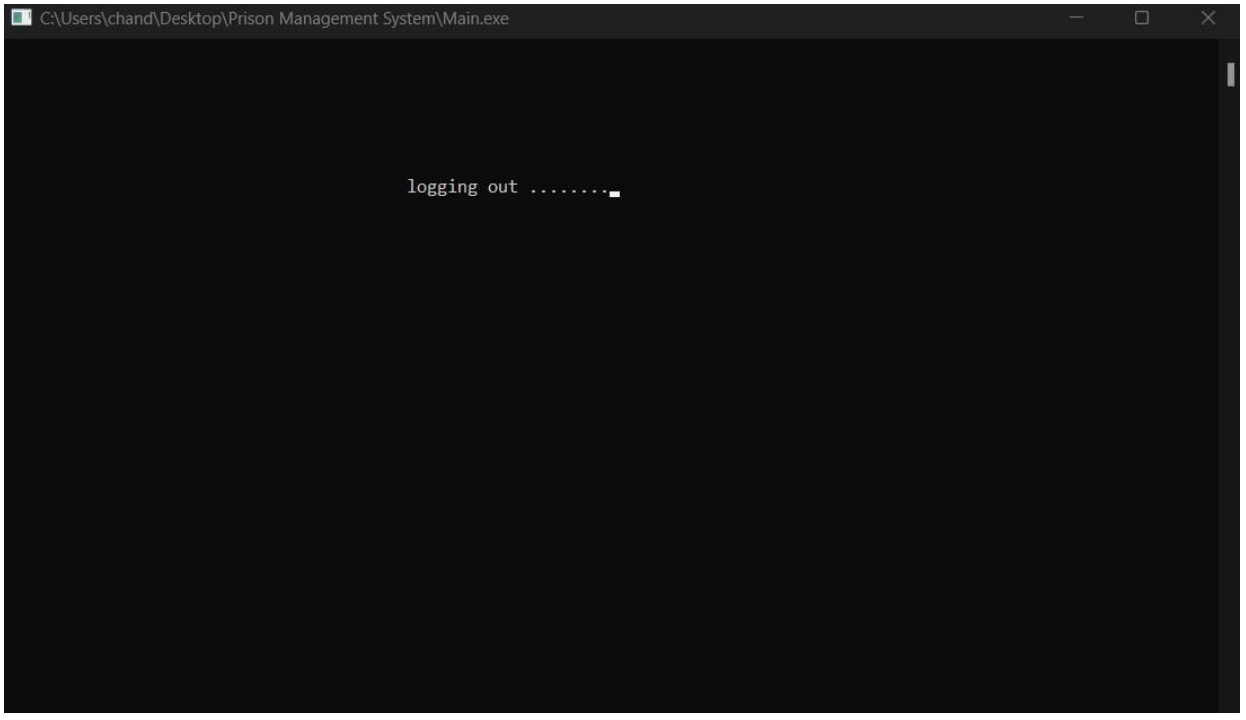


Fig 12: Logout

CHAPTER-5

CONCLUSION

The RRN technique is used for Prison Management System. The Relative Record Number identifies which position in a file the record is in.

- The purpose of conducting the study and doing the project is to know How the Prison information is maintained.
- The study of data reduces the response time.

Inmate Data: This directory contains subdirectories and files related to inmate information, including personal details, medical records, sentencing information, visitation logs, disciplinary records, and rehabilitation programs. Each inmate might have a separate folder or file within this directory.

Facility Operations: This directory includes subdirectories and files concerning the operational aspects of the prison facility. It may contain information about facility maintenance, staffing, schedules, security protocols, incident reports, and equipment inventory. Additionally, it may include data on inmate housing assignments and cell occupancy.

Administrative Files: This directory stores administrative documents and files related to the overall management of the prison system. It may include policy manuals, legal documents, financial records, budgeting information, audit reports, and correspondence.

User Access and Security: This directory focuses on user access control, security protocols, and authentication mechanisms. It might contain files related to user roles, permissions, login credentials, and audit logs to track system access and activities.

System Configuration: This directory stores configuration files related to the prison management system itself. It can include settings for system preferences, integration with other systems, database connections, and backup procedures.

BIBLIOGRAPHY

1. Michael j. Folk, Bill Zoellick, Greg Riccardi: File structures-An object oriented approach with C++, 3rd Edition, pearson Education, 1998.

2. K.R. Venugopal, K.G. Srinivas, P.M. Krishnaraj; File Structures Using C++, Tata McGraw- Hill, 2018.

3. Scot Robert Ladd: C++ Components and Algorithms, BPB Publications, 1993.

4. Wikipedia.com.