VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Proctor Management System

An J Component Report

Submitted by

Team Members:

NAME	REG. No.
Kaustubh Shrivastava	19BCE0617
Saurabh Singh	19BCI0184
Shashwat Sinha	19BCE0684

in partial fulfillment for the award of the degree of

B.Tech

In

COMPUTER SCIENCE ENGINEERING

Under the Guidance

Of

Faculty: Prof. Delhi Babu R

School of Computer Science and Engineering

JUNE 2020

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science

and Engineering

DECLARATION

I hereby declare that the J Component report entitled "**PROCTOR MANAGEMENT SYSTEM"** submitted by me to *Vellore Institute of Technology, Vellore-14* in partial fulfilment of the requirement for the award of the degree of **B.Tech** in **Computer Science and Engineering** is a record of bonafide undertaken by me under the supervision of **Dr. R. Delhi Babu.** I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Signature
Name: KAUSTUBH SHRIVASTAVA Reg. No. : 19BCE0617

Signature
Name: SHASHWAT SINHA
Reg. No. : 19BCE0684

Signature
Name: SAURABH SINGH
Reg. No. : 19BCI0184

TABLE OF CONTENTS

1. OBJECTIVES

- This project will automate the manual system used for management and maintenance of the critical information.
- It will reduce the numerous paper forms.
- As the database is centralized, it reduces the redundancy and the inconsistencies in the data.
- It aims at standardizing data, consolidating data, ensuring data integrity by the use of data structures and searching information in O(1) time by hashing algorithms.

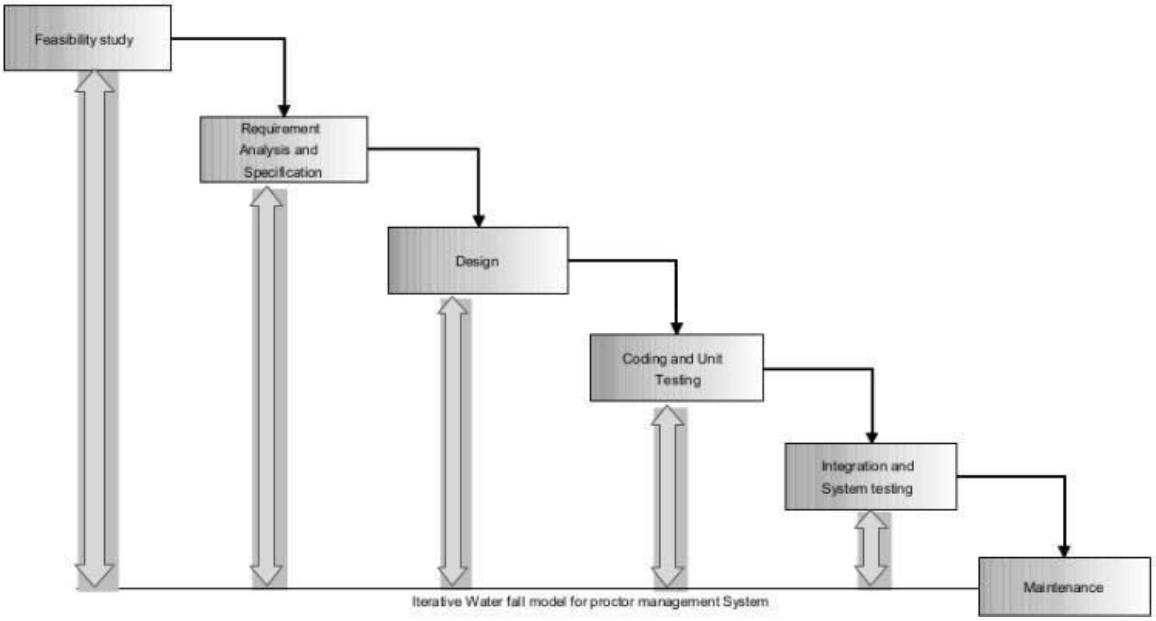
2. INTRODUCTION

A Proctor Management System **(PMS)** is a system that manages information typically involving student's personal information, marks, employee information, supervisor information and Daily Proctor Report **(DPR)** , reports and informing the students.

PMS Involves:

- Student Information Management
- Proctor Student Data Exchange Management
- Proctor – Student Management
- Supervisor – Faculty Management
- Parent Data Access and Exchange Management

3. METHODOLOGY



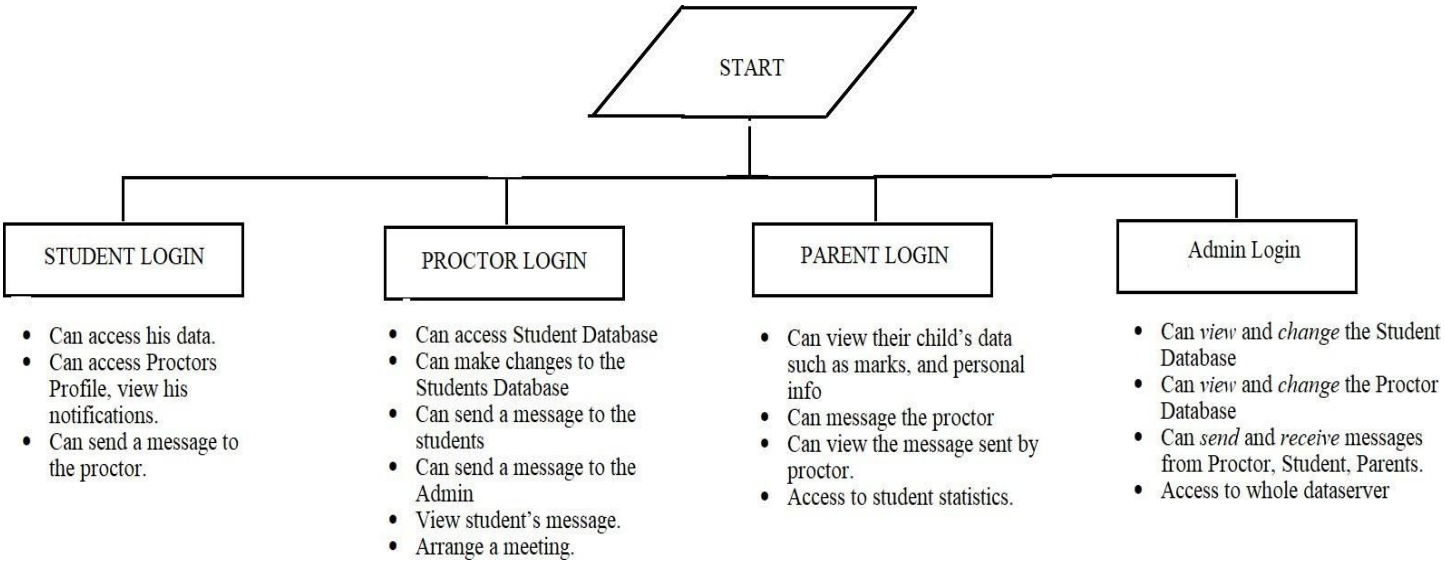
Iterative Waterfall Model

The iterative waterfall model provides feedback paths from every phase to its preceding phases, which is the main difference from the classical waterfall model. In the classical waterfall model, there are no feedback paths, so there is no mechanism for error correction. But in iterative waterfall model feedback path from one phase to its preceding phase allows correcting the errors that are committed and these changes are reflected in the later phases. Iterative waterfall model is very simple to understand and use. That's why it is one of the most widely used software development models.

All the phases of **the iterative waterfall model** are almost the same as they were in the classical waterfall model, and these phases are:

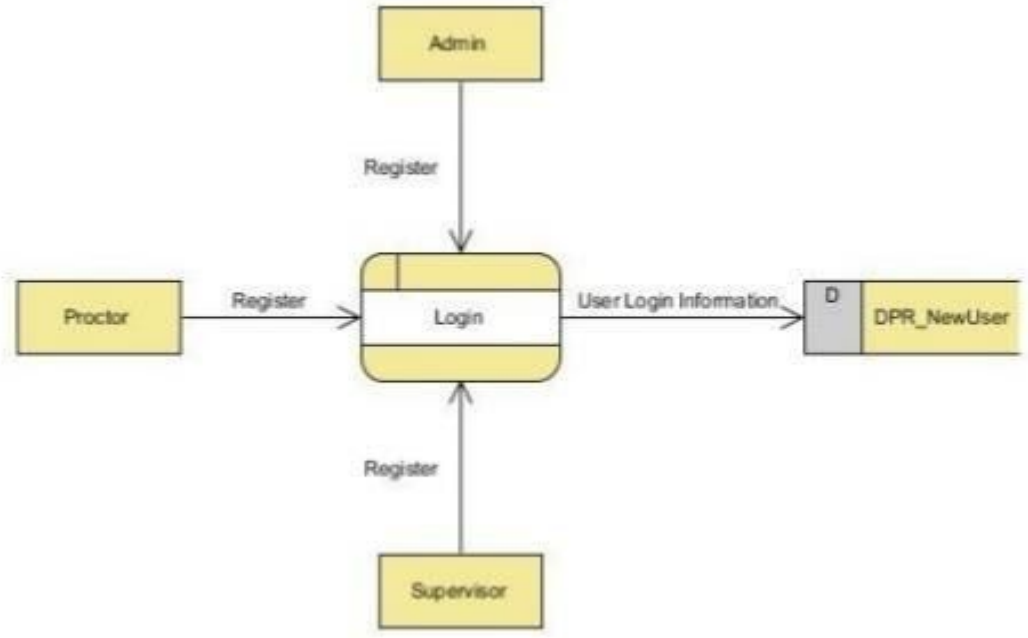
I. Requirement analysis and specification
II. Design
III. Implementation (Coding and unit testing)
IV. Integration and system testing
V. Deployment of systems
VI. Maintenance

4. STRUCTURE CHAT OF PMS

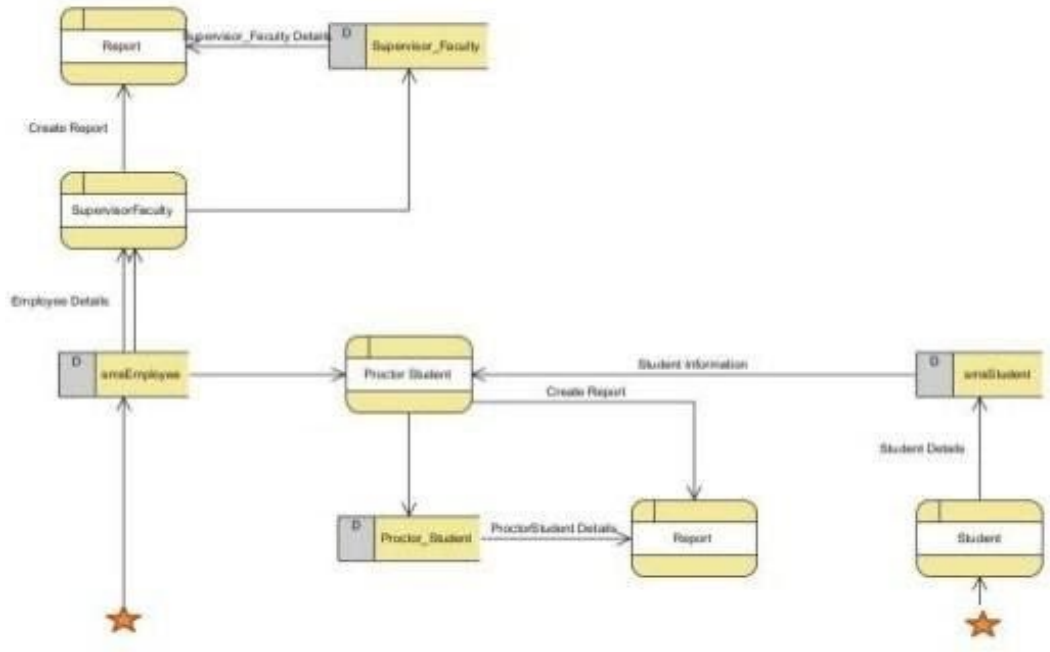


5. DATA FLOW DIAGRAM OF PMS

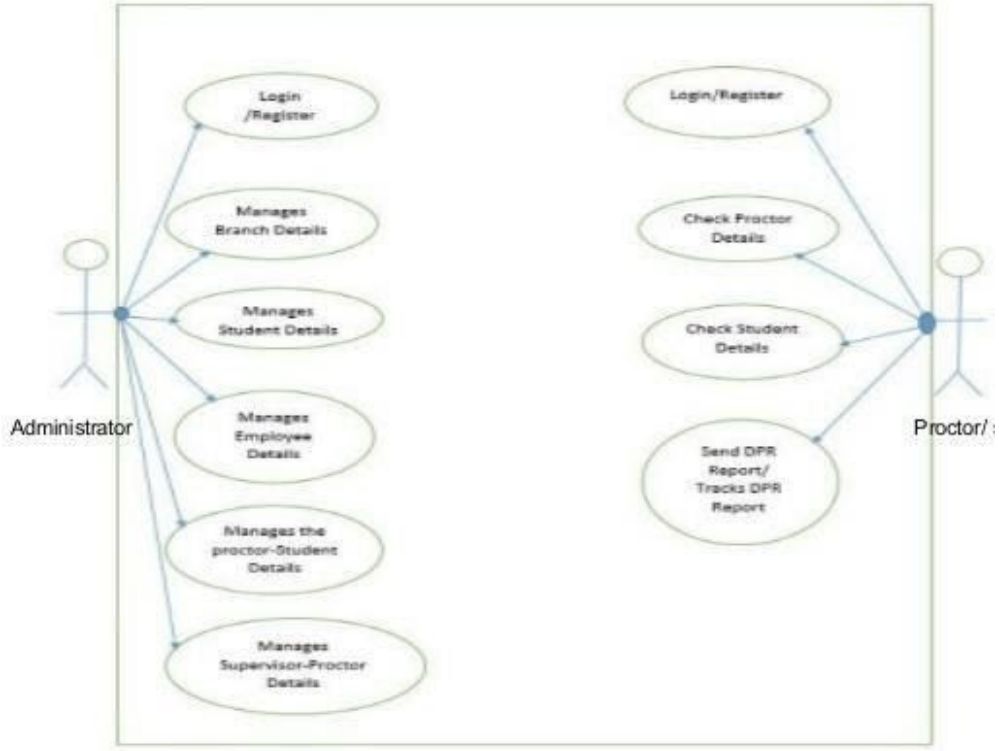
- LEVEL 1



- LEVEL 2



6. DIAGRAM OF DATABASE EXAMINATION MODULE



7. DATA STRUCTURES USED

7.1 QUEUE USING LINKED LIST

Queue has been used many times in the program in order to store students and their data. The hash table is made up of a queue where all the index numbers in the hash table points to the front of the queue.

```
class Link_lst_queue
{
    student *front;
    student *rear;
    int count;
public:
    Link_lst_queue()
    {
        front = NULL;
        rear = NULL;
        count = 0;
    }
```

```
void add(student *ptr_temp)
{
    count++;
    if(front == NULL)
    {
        front = ptr_temp;
        rear = ptr_temp;

    }
    else
    {
        rear -> next = ptr_temp;
        rear = ptr_temp;
    }
}
```

```
    student *get_student_front_ptr()
    {
        return front;
    }

    int get_count()
    {
        return count;
    }
};
```

As we can see in the above code snippet there is a function to add more students to the as well as we can get the front pointer of the queue.

There is also a function that tells how many students are present in that particular index.

7.2 TIME COMPLEXITY OF QUEUE

The time complexity for adding elements to the queue is **O(1).**

If we want to find a student in the queue then we have used linear search, so the time complexity of linear search is **O(n).**

8. HASHING ALGORITHM

There are two databases in the program one database is directly under the "proctor class" and the other database is the hashing table. Hashing table were mainly used so that the search done by any student, parent or administrator can be done in very less time.

So the proctor database is the main database and then we have to take the database from there and then feed it into the hash table that we have made for quick search of the student.

```
void hashing_the_student_info()
{

    for(int i = 0; i < 3; i++)
    {
        student *ptr = proctor_data[i].get_front_student();
        while(ptr != NULL)
        {
            string s = ptr -> get_reg_num();
            int sum = 0;
            for(auto x : s)
            {
                sum += (int)x;
            }

            int temp_num = sum % 10;
```

```
student *ptr_temp = new student;
ptr_temp -> set_name(ptr -> get_name());
ptr_temp -> set_proctor_id(ptr -> get_proctor_id());
ptr_temp -> set_reg_num(ptr -> get_reg_num());
ptr_temp -> set_phone_num(ptr -> get_phone_num());
ptr_temp -> set_password(ptr -> get_password());
ptr_temp -> set_father_name(ptr -> get_father_name());
ptr_temp -> set_mother_name(ptr -> get_mother_name());
ptr_temp -> set_message_to_proctor(ptr -> get_message_to_proctor());
ptr_temp -> set_message_from_proctor(ptr -> get_message_from_proctor());
```

```
marks *m = ptr -> get_student_marks();
for(int itr = 0; itr < 3; itr++)
{
    ptr_temp -> set_student_marks(m -> get_course_name(), m -> get_course_id(), m -> get_cat1(), m -> get_cat2(), m -> get_fat());
    m++;
}
ptr_temp -> set_password_parent(ptr -> get_password_parent());
ptr_temp -> set_mssg_prnt_proc(ptr -> get_mssg_prnt_proc());
ptr_temp -> set_mssg_proc_prnt(ptr -> get_mssg_proc_prnt());
```

```
ptr_temp -> next = NULL;

hashing_students[temp_num].add(ptr_temp);
hashing_by_parent_name[hash_func(ptr -> get_father_name())].add(ptr_temp);

ptr = ptr -> next;
```

The above code is the loading of the student data in the hash table from the main database.

```
int hash_func(string s)
{
    int sum = 0;
    for(auto x : s)
    {
        sum += (int)x;
    }

    return sum % 10;
}
```

This is the **Hash Function**. It takes in the string argument. It converts every character of the string into its **ASCII** number and adds all of it. After adding all of it modulo it with 10 as the hashing table's size is 10.

It assigns it a number between 0 to 9.

```
student *hash_search(student *ptr)
{
    student *temp = hashing_students[hash_func(ptr -> get_reg_num())].get_student_front_ptr();
    while(temp != NULL)
    {
        if(temp -> get_reg_num() == ptr -> get_reg_num())
        {
            break;
        }
        temp = temp -> next;
    }

    return temp;
}
```

This is the **Hash Search** which uses the hash function. In order to assign students to the hash table we are using their registration number. So, this hash search searches the student's registration number in the hash table that we have made.

It takes a student pointer as argument and uses that student's registration through the function to check the database.

```
student *hash_search(string reg)
{
    student *temp = hashing_students[hash_func(reg)].get_student_front_ptr();
    while(temp != NULL)
    {
        if(temp -> get_reg_num() == reg)
        {
            break;
        }
        temp = temp -> next;
    }

    return temp;
}
```

This is the overloaded function of the hash search but its argument is different from the above. It directly takes the registration number as an argument to find the student in the database.

8.1 TIME COMPLEXITY OF HASH TABLE

The complexity of putting the data in the hash table is **O(1)**. But when the index number of the registration number overlaps then we have started pushing it in a queue that we have made to store the data in the hash table when the number overlaps.

So, in order to find the student in the queue the time complexity is **O(n)**. But there are very less students that overlap so it is obviously faster from a linear search.

So in all it takes O(n) in searching the student data.

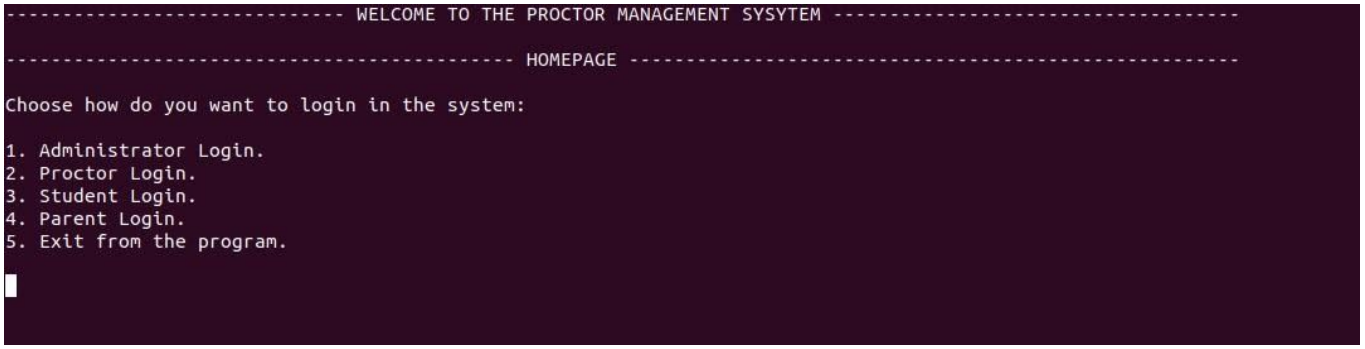
Hash table

Type	Unordered associative array
Invented	1953

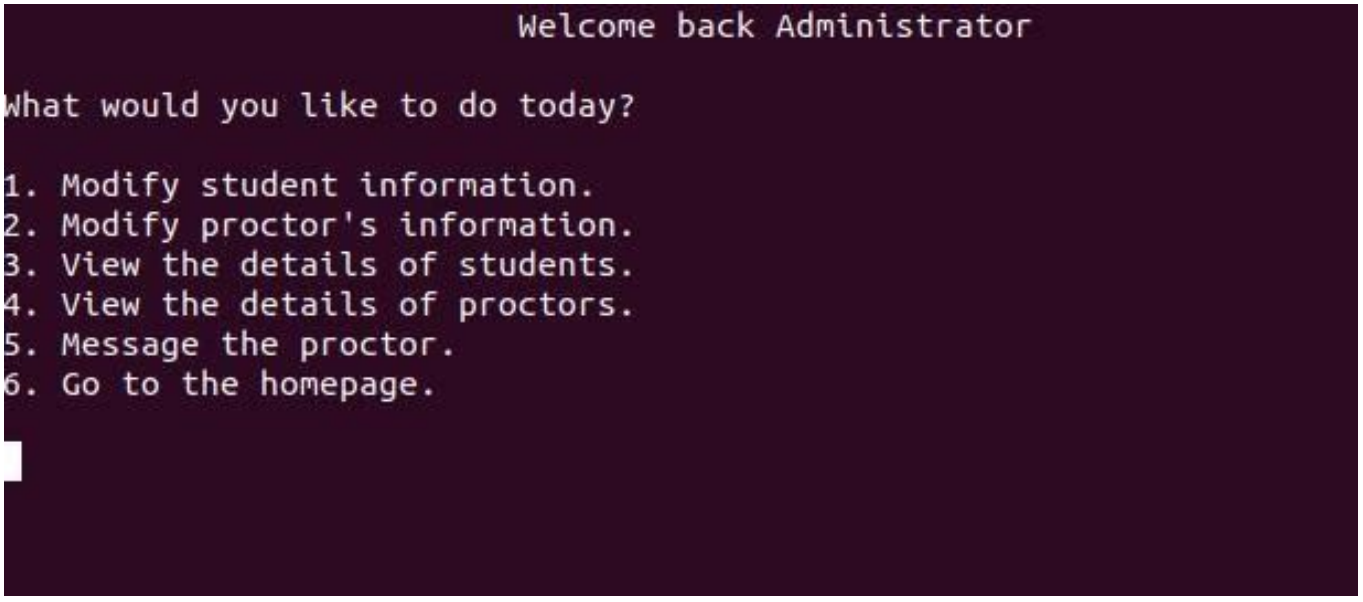
Time complexity in big O notation

Algorithm	Average	Worst Case
Space	O(*n*)[1]	O(*n*)
Search	O(1)	O(*n*)
Insert	O(1)	O(*n*)
Delete	O(1)	O(*n*)

9. SCREENSHOTS OF THE PROGRAM



9.1 ADMINISTRATOR LOGIN



```
The details of the students are as follows:

Name                : Kaustubh
Registration No.    : 19BCE0617
Phone No.           : 8868888227
Father's Name       : Jay Shankar
Mother's Name       : Priya Shankar

Name                : Shaswat
Registration No.    : 19BCE0673
Phone No.           : 8235387342
Father's Name       : Delhi
Mother's Name       : Dhaka

Name                : Saurabh
Registration No.    : 19BCE0634
Phone No.           : 6377288941
Father's Name       : Madrid
Mother's Name       : Kyoto

Name                : Moscow
Registration No.    : 19BCE0654
Phone No.           : 4523974267
Father's Name       : Colombo
Mother's Name       : Montreal

Name                : Denver
Registration No.    : 19BCE0645
Phone No.           : 4556871025
Father's Name       : Warsaw
Mother's Name       : Harare

Name                : Tokyo
Registration No.    : 19BCE0621
Phone No.           : 8962475334
Father's Name       : Congo
Mother's Name       : Istanbul

Name                : Oslo
Registration No.    : 19BCE0631
Phone No.           : 4886044452
Father's Name       : Dubai
Mother's Name       : Riyadh

Name                : Helsinki
Registration No.    : 19BCE0637
Phone No.           : 8883456025
Father's Name       : Ottawa
Mother's Name       : CapeTown
```

```
Who do you want to send a message to? Please enter the Proctor ID.

18BHDE999
```

```
Enter the message below :

Is everything going on fine? Do you want any help.
The message has been successfully sent to Mr.Srinivasan Krishnan.

Press enter to go the main menu.....
```

9.2 PROCTOR LOGIN

```
                                    Welcome back Mr.Srinivasan Krishnan

What would you like to do today?

1.  Modify student information.
2.  View message from the administrator.
3.  View any message from students.
4.  View any message from the student's parents
5.  View the details of student.
6.  View the marks of student.
7.  Message a student.
8.  Message a parent.
9.  Log out from the account.
10. Go to the homepage.
```

```
The messages from the students are :

There is no message from Kaustubh 19BCE0617.

There is no message from Shaswat 19BCE0673.

There is no message from Saurabh 19BCE0634.

There is no message from Moscow 19BCE0654.


Press any key to go to main menu....
```

```
Enter the message below:

Hey! Your son is doing quite well.
The message has been successfully sent to Kaustubh' s parent.

Press enter to go to the main menu.......
```

```
Is everything going on fine? Do you want any help.


Press any key to head back to the main menu......
```

Admin's message received by Proctor

```
The details of your proctees are as follows:

Name                : Kaustubh
Registration No.    : 19BCE0617
Phone No.           : 8868888227
Father's Name       : Jay Shankar
Mother's Name       : Priya Shankar

Name                : Shaswat
Registration No.    : 19BCE0673
Phone No.           : 8235387342
Father's Name       : Delhi
Mother's Name       : Dhaka

Name                : Saurabh
Registration No.    : 19BCE0634
Phone No.           : 6377288941
Father's Name       : Madrid
Mother's Name       : Kyoto

Name                : Moscow
Registration No.    : 19BCE0654
Phone No.           : 4523974267
Father's Name       : Colombo
Mother's Name       : Montreal

Name                : Denver
Registration No.    : 19BCE0645
Phone No.           : 4556871025
Father's Name       : Warsaw
Mother's Name       : Harare

Press any key to go back to the main menu.......
```

10. CONCLUSION

Based on the results we can easily state that automation of the entire **Proctor Management System** improves the efficiency in the way of interaction between proctor, student, administrator and also keeps parents up to date about the well- being of their child. It provides a better user interface which is better that the conventional way of management. The program has proven to be very effective in overcoming the delays in the communication due to which each member of the data base could be reached without much burden. Automated Proctor Management System is structured in keeping the data safe without redundancy and inconsistency. It manages to maintain the privacy of the user without a risk of being stolen due to which users can now work with a free mind. With the use of data integrity and centralized data control this program can deal with the bulk databases. Search operation using Hash Tables in this management system has made it very fast and reliable for users to get their results.

So, by completing this project on Proctor Management System our objectives to provide the users with various assistance in day to day work has been successfully achieved using appropriate data structures and algorithms to solve some of the problems frequently reported by the users.

11. REFERENCES

https://github.com/

https://www.geeksforgeeks.org/

https://www.tutorialspoint.com/computer_programming/index.htm

https://in.linkedin.com/

https://www.codewithc.com/


VIT
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
PROCTOR MANAGEMENT SYSTEM

DATA STRUCTURE AND ALGORITHM J- COMPONENT

KAUSTUBH SHRIVASTAVA- 19BCE0617
SAURABH SINGH-19BCI0184
SHASHWAT SINHA-19BCE0684

FACULTY- R. DELHI BABU

ABSTRACT OF PMS

Definition:

-A Proctor Management System is system that manages information typically involves; students personal information , employee information, students report.

-Proctor is a faculty of the college who is responsible to monitor the performances of the students. Proctor generates reports of the performances and informs to the parents about it. This system provides parents an easy access to their ward's academic performance.

•Proctor can easily communicate with parents and students easily.

The application can be used by:-

• PROCTOR

•STUDENTS

•PARENTS

•ADMINISTRATOR

The admin has full access to this proctor management system.

OBJECTIVES

•This Project will automate the manual system used for management and maintenance by proctor of the critical information of students and the communication of proctor with students and parents.

•This database can reduce the work as it can be used by admin,proctor, students and parents.

•It will reduce the redundancy and the inconsistencies in the data,as the database is centralized redundancy.

INTRODUCTION

• This Proctor Management System provides a simple interface for the maintenance of student information controlled and maintained by proctor and administrator.

•. Achieving this objective is difficult using a manual system as the information is scattered, can be redundant and collecting relevant information may be very time consuming. . All these problems are solved using online Proctor Management System.

• This system focuses on presenting information in an easy and intelligible manner and it allows student and parents to directly contact proctor.

•All data is stored securely linked lists and classes.

LITERATURE REVIEW

•As earlier proctor have to maintain records of their proctees manually using paper and registers and also the attendance was taken manually. There were no exixting online system for proctor to get all those records.

• Proctors communicate with students and parents through phone calls and therefore sometimes they're not able to reach them.

•Sometimes there were errors in report generation

PROBLEMS

•As we have seen our proctors maintain our records on registers using pen ,paper.

•The time taken to maintain this paperwork is lot more .

•Proctors have to ask students of their results and how they performed.

•They cannot communicate with students and parents so easily.

•Inconsistency in the reports generated manually.

SOLUTIONS

- PMS helps proctors to maintain and can change records of the members online which can be changed if needed and results in less work time .
- Proctors can see the results and marks of students online and can send message to students and parents concerning marks.
- The interaction between proctor->students->parents will get easy.
- Paperwork and inconsistency gets less. Records are more secure here.

SOME MAJOR DATA STRUCTURE USED

HASH TABLE – HASHING

- Hashing table were mainly used so that the search done by any student, parent or administrator can be done in very less

```
void hashing_the_student_info()
{
    for(int i = 0; i < 3; i++)
    {
        student *ptr = proctor_data[i].get_front_student();
        while(ptr != NULL)
        {
            string s = ptr -> get_reg_num();
            int sum = 0;
            for(auto x : s)
            {
                sum += (int)x;
            }
            int temp_num = sum % 10;
```

```
student *ptr_temp = new student;
ptr_temp -> set_name(ptr -> get_name());
ptr_temp -> set_proctor_id(ptr -> get_proctor_id());
ptr_temp -> set_reg_num(ptr -> get_reg_num());
ptr_temp -> set_phone_num(ptr -> get_phone_num());
ptr_temp -> set_password(ptr -> get_password());
ptr_temp -> set_father_name(ptr -> get_father_name());
ptr_temp -> set_mother_name(ptr -> get_mother_name());
ptr_temp -> set_message_to_proctor(ptr -> get_message_to_proctor());
ptr_temp -> set_message_from_proctor(ptr -> get_message_from_proctor());
```

```
marks *m = ptr -> get_student_marks();
for(int itr = 0; itr < 3; itr++)
{
    ptr_temp -> set_student_marks(m -> get_course_name(), m -> get_course_id(), m -> get_cat1(), m -> get_cat2(), m -> get_fat());
    m++;
}
ptr_temp -> set_password_parent(ptr -> get_password_parent());
ptr_temp -> set_msg_prnt_proc(ptr -> get_msg_prnt_proc());
ptr_temp -> set_msg_proc_prnt(ptr -> get_msg_proc_prnt());
```

```
ptr_temp -> next = NULL;

hashing_students[temp_num].add(ptr_temp);
hashing_by_parent_name[hash_func(ptr -> get_father_name())].add(ptr_temp);

ptr = ptr -> next;
```

```
int hash_func(string s)
{
    int sum = 0;
    for(auto x : s)
    {
        sum += (int)x;
    }

    return sum % 10;
}
```

- This is the hash function. It takes in the string argument. It converts every character of the string into its ascii number and adds all of it. After adding all of it it modulo it with 10 as the hashing table's size is 10.
- It assigns it a number between 0 to 9.
- TIME COMPLEXITY:- O(n)

```
student *hash_search(student *ptr)
{
    student *temp = hashing_students[hash_func(ptr -> get_reg_num())].get_student_front_ptr();
    while(temp != NULL)
    {
        if(temp -> get_reg_num() == ptr -> get_reg_num())
        {
            break;
        }
        temp = temp -> next;
    }

    return temp;
}
```

This is the hash search it uses the hash function. In order to assign students to the hash table we are using their registration number. So, this hash search searches the student's registration number in the hash table that we have made.

It takes a student pointer as argument and uses that student's registration through the function to check the database.

QUEUE USING LINKED LIST

- To store data of members .
- The hash table is made up of a queue where all the index numbers in the hash table points to the front of the queue.

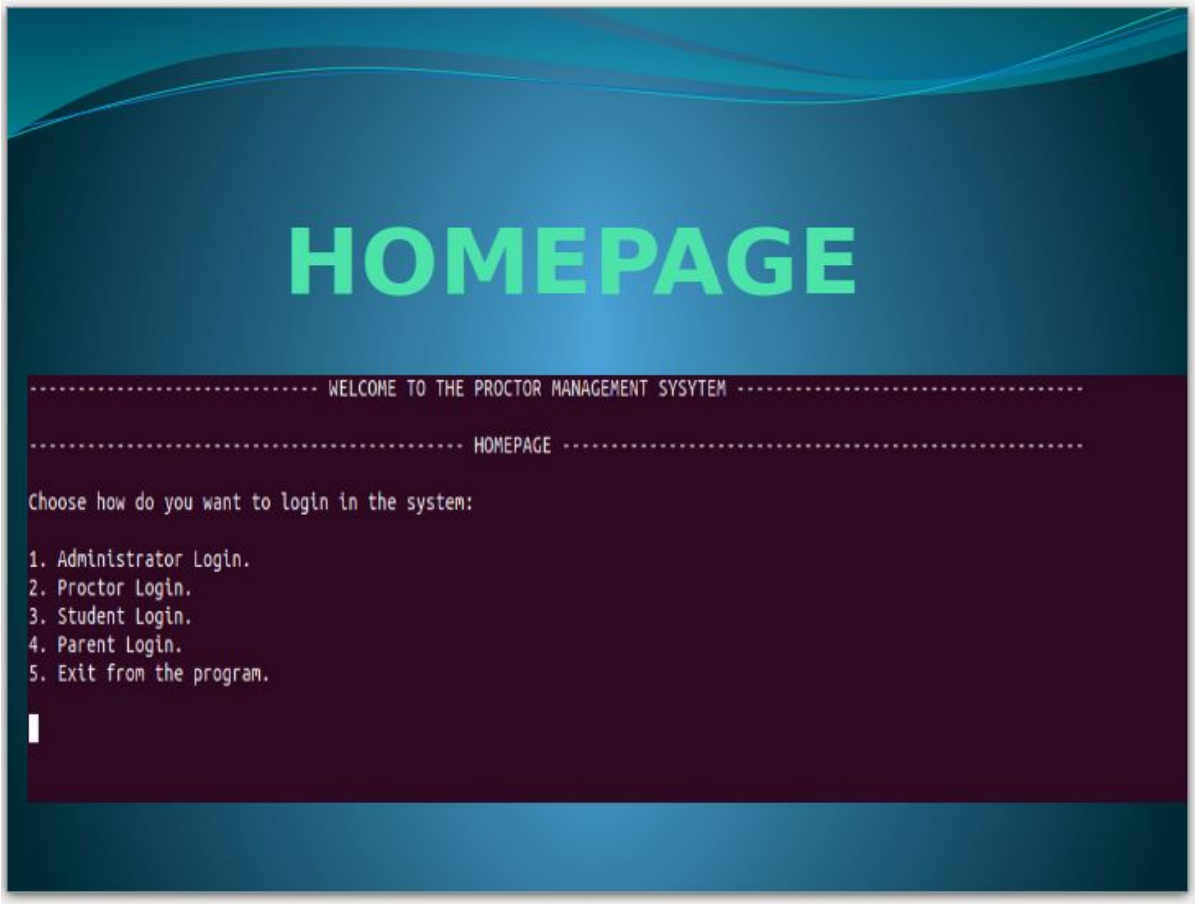
```
class Link_lst_queue
{
    student *front;
    student *rear;
    int count;
public:
    Link_lst_queue()
    {
        front = NULL;
        rear = NULL;
        count = 0;
    }
```

```
void add(student *ptr_temp)
{
    count++;
    if(front == NULL)
    {
        front = ptr_temp;
        rear = ptr_temp;

    }
    else
    {
        rear -> next = ptr_temp;
        rear = ptr_temp;
    }
}
```

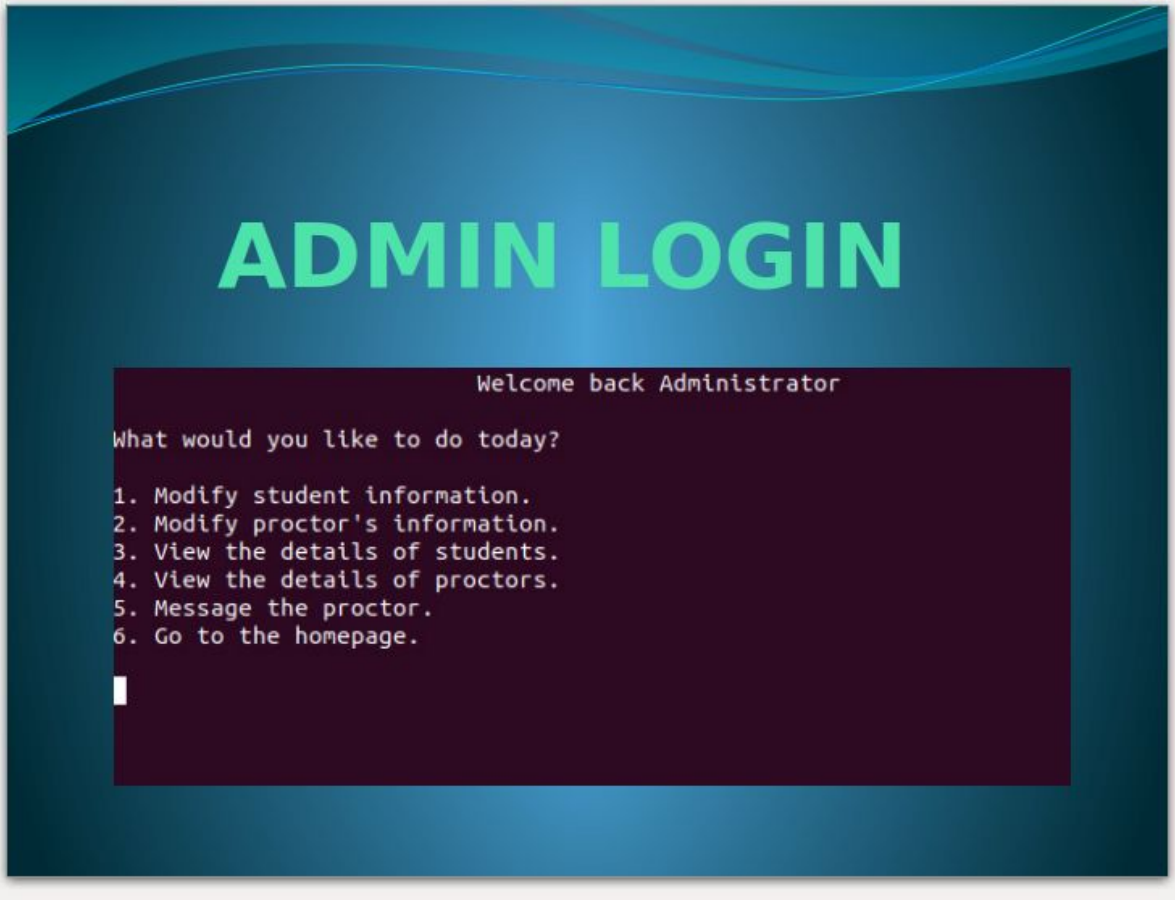
```
    student *get_student_front_ptr()
    {
        return front;
    }

    int get_count()
    {
        return count;
    }
};
```

- As we can see above there is a function to add more students to the as well as we can get the front pointer of the queue.
- And one more function to count the number of members.
- TIME COMPLEXITY:- O(n).

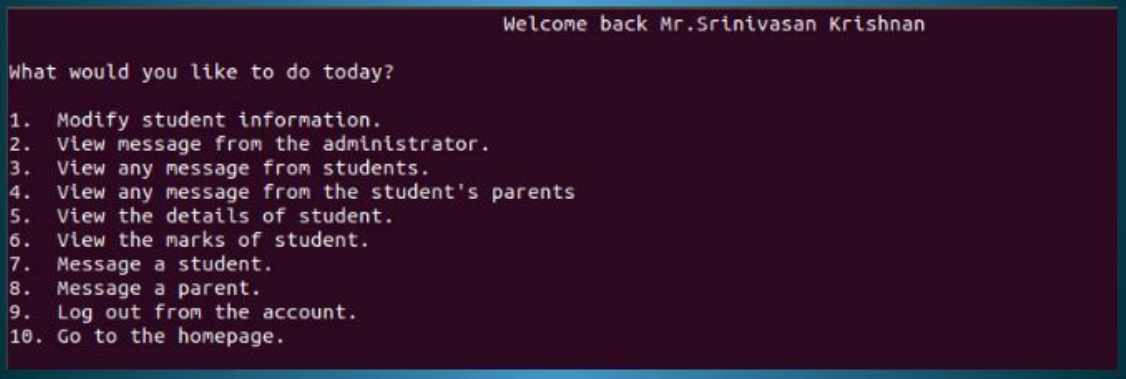
HOMEPAGE



ADMIN LOGIN



PROCTOR LOGIN



CONCLUSION

- Automation of the entire system improves the efficiency
- It effectively overcomes the delay in communications as one can directly send message to the other.
- Updating of information becomes so easier using this system.