|  |
| --- |
| detail of persons hands with scissors, markers, workingEmployee Management System |

|  |  |  |
| --- | --- | --- |
| DATA STRUCTURE & ALGORITHM DESIGN |  |  |
| Rabia Batool (2022-BSE-064)  Sawaira Saeed (2022-BSE-067)  Hafsah Tahir (2022-BSE-052) | **Group members:** |  |

## Employee management system:

## Introduction:

## This data structure report aims to provide a comprehensive overview of the implementation details and challenges encountered in developing the Employee Management System project. This system, developed using the C++ language, offers an efficient and effective solution for managing employee records.

## To ensure a seamless experience for users, the Employee Management System is designed with user-friendly interfaces. The system incorporates advanced features, such as a search option for quickly locating employee records, a login feature for secure access, and an edit option for updating employee information.

## Importance:

## To enhance the performance and reliability of the system, data is stored and manipulated using a Linked List data structure. The Linked List is a versatile and dynamic data structure that allows for efficient insertion, deletion, and search operations. Each employee record in the Linked List is represented by a node, which contains the employee's attributes, such as their name, employee ID, department, post, and salary.

## Challenges:

## One of the significant challenges faced during the development of the Employee Management System was ensuring the system's scalability. The system is designed to handle a large number of employee records without compromising performance. This is achieved by implementing efficient data manipulation algorithms, such as insertion and deletion, which operate in constant time.

## The security of the Employee Management System is a priority. To protect sensitive employee information, the system includes a secure login feature that authenticates user credentials before granting access to the system.

## Conclusion:

## The report concludes by discussing the future scope and improvements that can be made to the Employee Management System. Potential enhancements include implementing a Graphical User Interface (GUI) to provide an even more intuitive user experience, incorporating advanced data validation techniques to prevent data corruption, and implementing cloud-based storage to enable real-time data access and collaboration among employees.

**CODE:**

// proj.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include<iostream>

#include<string>

using namespace std;

class Node

{

public:

string Name, Post, Department;

int Emp\_Id;

float Salary;

Node\* Next\_Add;

};

class Employee

{

private:

Node\* head;

public:

Employee();

void Introduction();

void Login();

void Control\_panel();

void insert();

void search\_Id();

void edit();

void delete\_Id();

void display();

void salary\_slip();

void search\_dept();

};

Employee::Employee()

{

head = nullptr;

}

void Employee::Introduction()

{

cout << "\n\n\n\n\n\n\n";

cout << "\t\t\t=====================================";

cout << "\n\t\t\t=====================================";

cout << "\n\n\t\t\tEMPLOYEE MANAGEMENT SYSTEM";

cout << "\n\t\t\t VERSION : 1.1";

cout << "\n\n\t\t\t======================================";

cout << "\n\t\t\t=======================================";

}

void Employee::Login()

{

string user, pass;

cout << "\n\n";

cout << "\t\t\t==============================================";

cout << "\n\n\t\t\t\t LOGIN PANEL";

cout << "\n\n\t\t\t===============================================";

cout << "\n\n\n ENTER USER NAME :";

cin >> user;

cout << "\n\n ENTER PASSWORD:";

cin >> pass;

if (user == "hafsah" && pass == "12345")

{

cout << "\n\n\n\t\t\t CONGRATULATION LOGIN SUCCESS";

cout << "\n\n\n\t\t\t\ LOADING";

for (int i = 1; i <= 6; i++)

{

cout << ".\n";

}

Control\_panel();

}

else

{

cout << "\n\n\n USER NAME or PASSWORD IS WRONG";

}

}

void Employee::Control\_panel()

{

int x;

cout << "\n\n\t\t\t===============================================";

cout << "\n\n\t\t\t EMPLOYEE CONTROL PANEL";

cout << "\n\n\t\t\t====================================================";

cout << "\n\n\n 1. INSERT RECORD";

cout << "\n 2. SEARCH RECORD";

cout << "\n 3. EDIT RECORD";

cout << "\n 4. DELETE RECORD";

cout << "\n 5. DISPLAY RECORD";

cout << "\n 6. SALARY SLIP";

cout << "\n 7. SEARCH DEPARTMENT";

cout << "\n 8. EXIT";

cout << "\n\n YOUR CHOICE:";

cin >> x;

switch (x)

{

case 1:

insert();

break;

case 2:

search\_Id();

break;

case 3:

edit();

break;

case 4:

delete\_Id();

break;

case 5:

display();

break;

case 6:

salary\_slip();

break;

case 7:

search\_dept();

break;

case 8:

exit(0);

default:

cout << "\n\n \* INVALID CHOICE...PLEASE TRY AGAIN\*\*\*";

}

}

void Employee::insert()

{

Node\* temp = new Node;

cout << "\n\n\t\t\t=========================================";

cout << "\n\n\t\t\t INSERT EMPLOYEE RECORD";

cout << "\n\n\t\t\t=========================================";

cout << "\n\n EMPLOYEE ID :";

cin >> temp->Emp\_Id;

// Check for duplicate Employee ID

Node\* duplicateCheck = head;

while (duplicateCheck != nullptr)

{

if (temp->Emp\_Id == duplicateCheck->Emp\_Id)

{

cout << "\n\n\*DUPLICATE EMPLOYEE RECORD\*";

delete temp;

return;

}

duplicateCheck = duplicateCheck->Next\_Add;

}

cout << "\n\n EMPLOYEE NAME :";

cin.ignore(); // Clear the newline character from the buffer

getline(cin, temp->Name);

cout << "\n\n EMPLOYEE POST :";

getline(cin, temp->Post);

cout << "\n\n EMPLOYEE DEPARTMENT :";

getline(cin, temp->Department);

cout << "\n\n EMPLOYEE SALARY :";

cin >> temp->Salary;

temp->Next\_Add = nullptr;

if (head == nullptr || temp->Emp\_Id < head->Emp\_Id)

{

// Insert at the beginning if the list is empty or the new employee ID is smaller than the head's employee ID

temp->Next\_Add = head;

head = temp;

}

else

{

Node\* s1 = head;

Node\* s2 = head;

while (s1 != nullptr)

{

if (s1->Emp\_Id > temp->Emp\_Id && s2->Emp\_Id < temp->Emp\_Id)

{

s2->Next\_Add = temp;

temp->Next\_Add = s1;

break;

}

s2 = s1;

s1 = s1->Next\_Add;

}

if (s1 == nullptr)

{

s2->Next\_Add = temp;

}

}

cout << "\n\n\*EMPLOYEE RECORD INSERTED\*";

}

void Employee::search\_Id()

{

int found = 0;

int EmpId;

cout << "\n\n\t\t======================================";

cout << "\n\n\t\t SEARCH EMPLOYEE RECORD";

cout << "\n\n\t\t======================================";

if (head == nullptr)

{

cout << "\n\n\*\* LINKED LIST IS EMPTY \*\*";

}

else

{

cout << "\n\n EMPLOYEE ID FOR SEARCH:";

cin >> EmpId;

Node\* ptr = head;

while (ptr != nullptr)

{

if (EmpId == ptr->Emp\_Id)

{

cout << "\n\n\t\t\t===================================";

cout << "\n\n\t\t\t SEARCH EMPLOYEE RECORD";

cout << "\n\n\t\t\t===================================";

cout << "\n\n EMPLOYEE ID:" << ptr->Emp\_Id;

cout << "\n\n EMPLOYEE NAME:" << ptr->Name;

cout << "\n\n EMPLOYEE POST:" << ptr->Post;

cout << "\n\n EMPLOYEE DEPARTMENT:" << ptr->Department;

cout << "\n\n EMPLOYEE SALARY:" << ptr->Salary;

found++;

}

ptr = ptr->Next\_Add;

}

if (found == 0)

{

cout << "\n\n \*EMPLOYEE ID IS NOT FOUND\*";

}

}

}

void Employee::edit()

{

int found = 0;

int EmpId;

cout << "\n\n\t\t\t=======================================";

cout << "\n\n\t\t\t MODIFY EMPLOYEE RECORD";

cout << "\n\n\t\t\t===========================================";

if (head == nullptr)

{

cout << "\n\n\*\* LINKED LIST IS EMPTY\*\*";

}

else

{

cout << "\n\n EMPLOYEE ID FOR MODIFY:";

cin >> EmpId;

Node\* ptr = head;

while (ptr != nullptr)

{

if (EmpId == ptr->Emp\_Id)

{

cout << "\n\n EMPLOYEE NAME:";

getline(cin, ptr->Name);

cout << "\n\n EMPLOYEE POST:";

getline(cin, ptr->Post);

cout << "\n\n EMPLOYEE DEPARTMENT:";

getline(cin, ptr->Department);

cout << "\n\n EMPLOYEE SALARY:";

cin >> ptr->Salary;

cout << "\n\n\*\* EMPLOYEE RECORD MODIFY\*\*";

found++;

}

ptr = ptr->Next\_Add;

}

if (found == 0)

{

cout << "\n\n\*EMPLOYEE ID NOT FOUND\*";

}

}

}

void Employee::delete\_Id()

{

int found = 0;

int id;

cout << "\n\n\t\t\t=======================================";

cout << "\n\n\t\t\t DELETE EMPLOYEE RECORD";

cout << "\n\n\t\t\t=======================================";

cout << "Employee ID: ";

cin >> id;

if (head == NULL)

{

cout << "Linked list is empty";

}

else if (id == head->Emp\_Id)

{

Node\* temp = head;

head = temp->Next\_Add;

temp->Next\_Add = NULL;

delete temp;

found++;

}

else

{

Node\* s1 = head;

Node\* s2 = s1->Next\_Add;

while (s2 != NULL)

{

if (s2->Emp\_Id == id)

{

s1->Next\_Add = s2->Next\_Add;

s2->Next\_Add = NULL;

delete s2;

found++;

break;

}

s1 = s2;

s2 = s2->Next\_Add;

}

if (found == 0 && s1->Emp\_Id == id)

{

Node\* temp = s1;

s1->Next\_Add = NULL;

delete temp;

found++;

}

}

if (found == 0)

{

cout << "\n\n Employee ID not found";

}

else

{

cout << "\n\n Employee record deleted";

}

}

void Employee::display()

{

cout << "\n\n\t\t\t=======================================";

cout << "\n\n\t\t\t DISPLAY EMPLOYEE RECORD";

cout << "\n\n\t\t\t=======================================";

if (head == nullptr)

{

cout << "\n\n\*\* LINKED LIST IS EMPTY\*\*";

}

else

{

Node\* ptr = head;

while (ptr != nullptr)

{

cout << "\n\n EMPLOYEE ID:" << ptr->Emp\_Id;

cout << "\n\n EMPLOYEE NAME:" << ptr->Name;

cout << "\n\n EMPLOYEE POST:" << ptr->Post;

cout << "\n\n EMPLOYEE DEPARTMENT:" << ptr->Department;

cout << "\n\n EMPLOYEE SALARY:" << ptr->Salary;

cout << "\n\n================================================";

cout << "\n\n==================================================";

ptr = ptr->Next\_Add;

}

}

}

void Employee::salary\_slip()

{

int found = 0;

int EmpId;

cout << "\n\n\t\t\t=======================================";

cout << "\n\n\t\t\t SALARY SLIP GENERATION";

cout << "\n\n\t\t\t=======================================";

if (head == nullptr)

{

cout << "\n\n\*\* LINKED LIST IS EMPTY\*\*";

}

else

{

cout << "\n\n EMPLOYEE ID FOR SLIP:";

cin >> EmpId;

Node\* ptr = head;

while (ptr != nullptr)

{

if (EmpId == ptr->Emp\_Id)

{

cout << "\n\t\t\t\*\*\*\*\*\*\*";

cout << "\n\t\t\t\* \*";

cout << "\n\t\t\t\* EMPLOYEE SALARY SLIP \*";

cout << "\n\t\t\t\* \*";

cout << "\n\t\t\t\*\*\*\*\*\*\*";

cout << "\n\n EMPLOYEE ID: " << ptr->Emp\_Id;

cout << "\n\n NAME: " << ptr->Name;

cout << "\n\n POST: " << ptr->Post;

cout << "\n\n DEPARTMENT: " << ptr->Department;

cout << "\n\n SALARY: " << ptr->Salary;

cout << "\n\t\t\t\*\*\*\*\*\*\*";

cout << "\n\t\t\t\*\*\*\*\*\*\*";

found++;

}

ptr = ptr->Next\_Add;

}

if (found == 0)

{

cout << "\n\n\\* EMPLOYEE ID NOT FOUND \*";

}

}

}

void Employee::search\_dept()

{

int found = 0;

string dept;

cout << "\n\n\t\t\t=======================================";

cout << "\n\n\t\t\t SEARCH DEPARTMENT RECORD ";

cout << "\n\n\t\t\t=======================================";

if (head == nullptr)

{

cout << "\n\n\*\* LINKED LIST IS EMPTY\*\*";

}

else

{

cout << "\n\n DEPARTMENT NAME FOR SEARCH:";

getline(cin, dept);

Node\* ptr = head;

while (ptr != nullptr)

{

if (dept == ptr->Department)

{

cout << "\n\n EMPLOYEE ID:" << ptr->Emp\_Id;

cout << "\n\n EMPLOYEE NAME:" << ptr->Name;

cout << "\n\n EMPLOYEE POST:" << ptr->Post;

cout << "\n\n EMPLOYEE DEPARTMENT:" << ptr->Department;

cout << "\n\n EMPLOYEE SALARY:" << ptr->Salary;

cout << "\n\n================================================";

cout << "\n\n================================================";

found++;

}

ptr = ptr->Next\_Add;

}

if (found == 0)

{

cout << "\n\n \* DEPARTMENT NAME NOT FOUND\*\*\*";

}

else

{

cout << "\n\n TOTAL EMPLOYEE " << found;

}

}

}

int \_tmain(int argc, \_TCHAR\* argv[])

{Employee E;

E.Introduction();

E.insert();

E.insert();

E.insert();

E.insert();

E.insert();

E.display();

E.search\_Id();

E.Login();

E. Control\_panel();

E.edit();

E.delete\_Id();

E.delete\_Id();

E.delete\_Id();

E.display();

E.salary\_slip();

E.search\_dept();

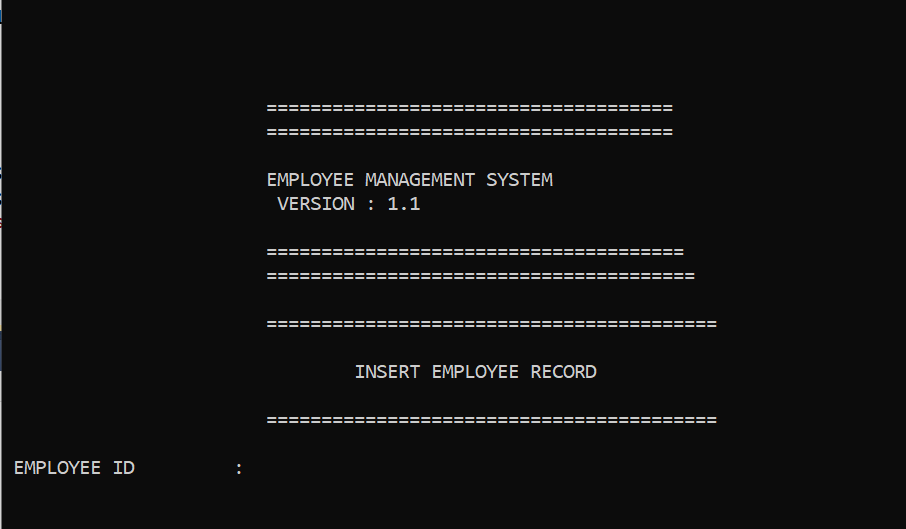
system("pause");

return 0;

}

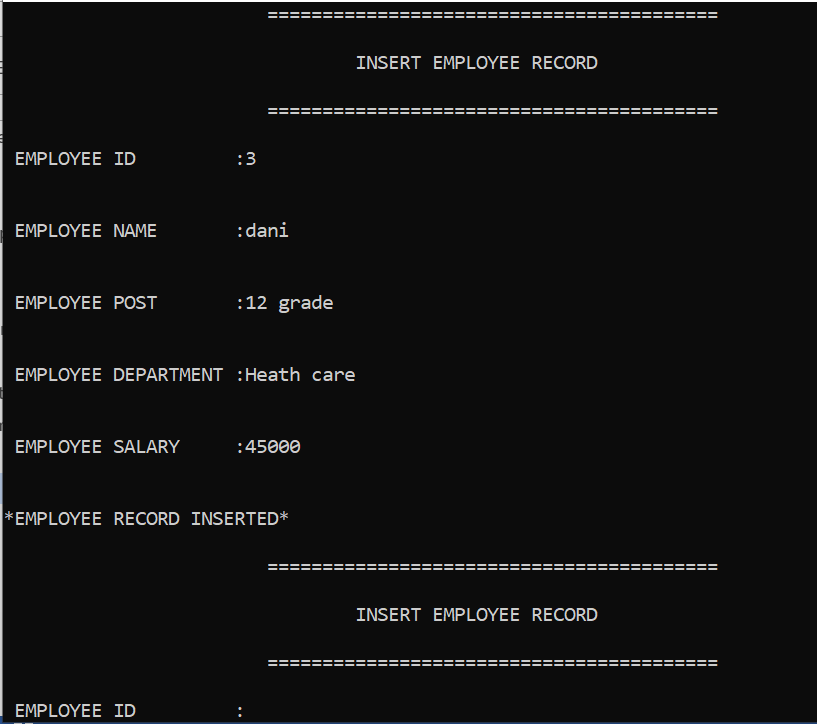
OUTPUT:

Insert record:



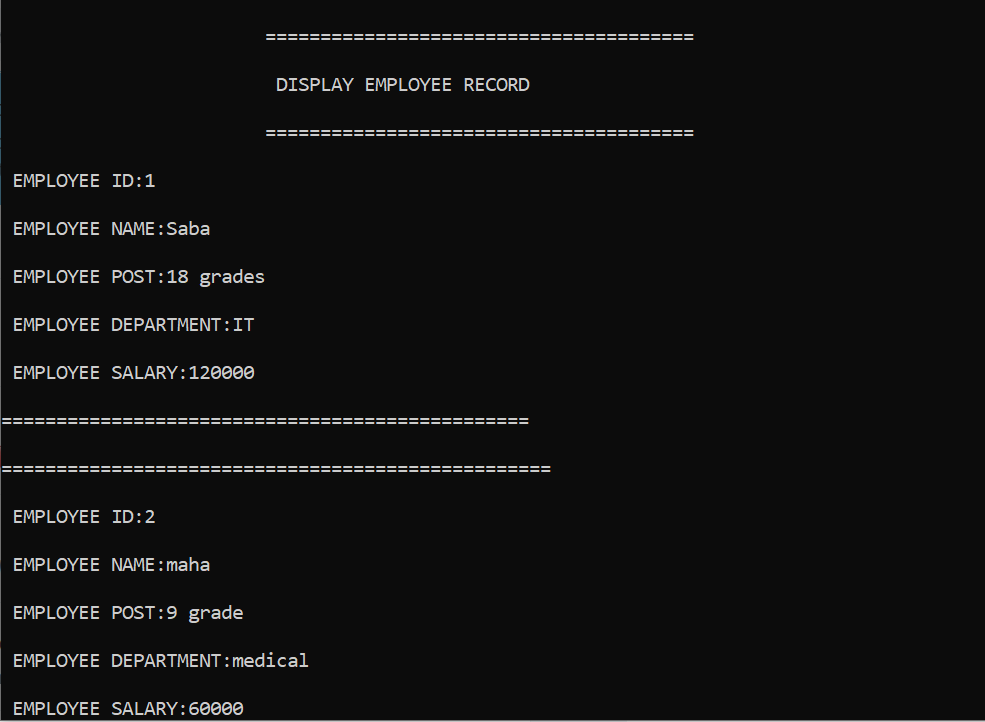




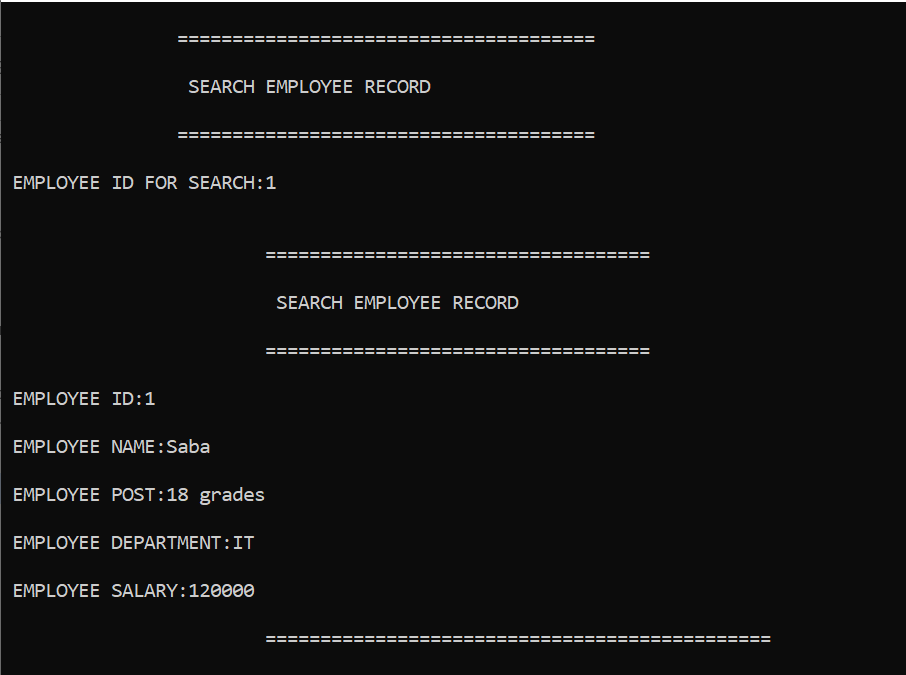




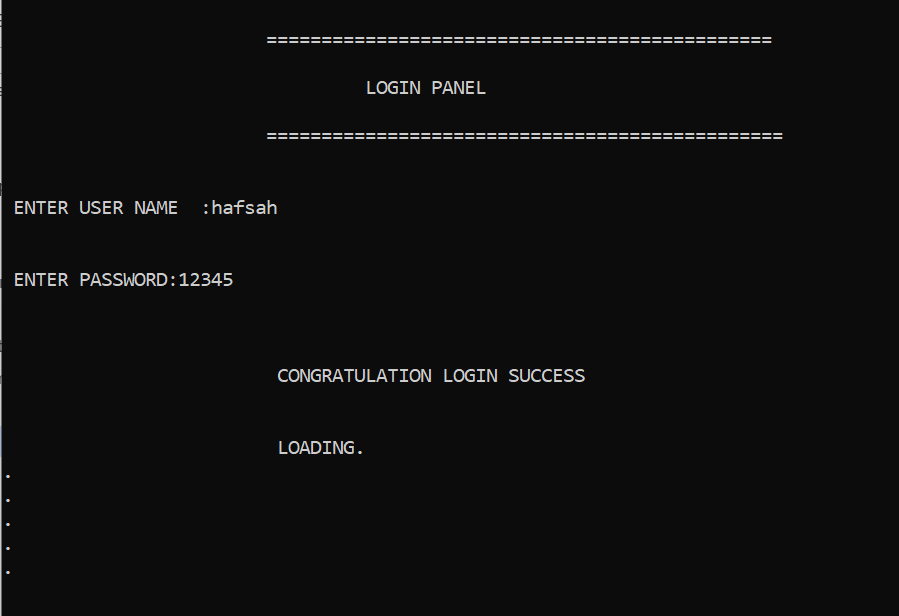
**Display Record:**



**Search through ID:**



**Login to account:**



**Control panel:**



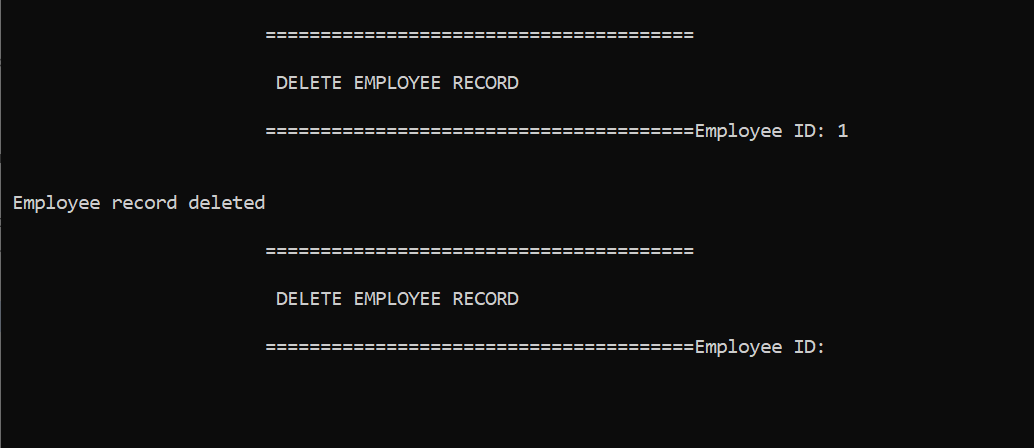
**Slip generation through ID:**



**Edit account:**



**Delete employee:**



**Search through department:**

