PIMPRI CHINCHWAD EDUCATION TRUST's.

PIMPRI CHINCHWAD COLLEGE OF ENGINEERING

(An Autonomous Institute)

S.Y. B. TECH **Year:** 2024 – 25 **Semester:** I
Name: Abhishek Joshi **PRN:123B1B150**
Department: Computer Engineering **Division:** C (C1)
Course: Data Structures Laboratory **Course Code:** BCE23PC02
Date:

Assignment -11

Aim: Consider an employee database of N employees. Make use of a hash table implementation to quickly look up the employer's id number.

- **Source Code :**

```
#include<iostream>
using namespace std;

class Employee {
   int EmpID;
   string Name;
   int contact;

public:
   int index;
   Employee() {
      EmpID = -1;
      Name = "";
      contact = -1;
      index = -1;
   }
   void setID(int id) { EmpID = id; }
   void setName(string n) { Name = n; }
   void setContact(int c) { contact = c; }
   void setIndex(int i) { index = i; }

   int getID() { return EmpID; }
   void setEmployee(int EmpID, string N, int contact, int index) {
```

```
        this->EmpID = EmpID;
        Name = N;
        this->contact = contact;
        this->index = index;
    }
    void printEmployee() {
        if (EmpID != -1)
            cout << "Details: " << EmpID << " - " << Name << " - " << contact << endl;
    }
};

class HashTable {
    int tableSize;
    Employee *ht;

public:
    HashTable(int size) {
        tableSize = size;
        ht = new Employee[tableSize];
    }

    int hash(int value) {
        return (value % tableSize);
    }

    void insertIntoHT(int EmpID, string N, int contact) {
        int ToBeInsertedAt = hash(EmpID);
        if (ht[ToBeInsertedAt].index == -1) {
            ht[ToBeInsertedAt].setEmployee(EmpID, N, contact, ToBeInsertedAt);
        } else {
            for (int i = 0; i < tableSize; i++) {
                ToBeInsertedAt = (ToBeInsertedAt + 1) % tableSize;

                if (ht[ToBeInsertedAt].index == -1) {
                    ht[ToBeInsertedAt].setEmployee(EmpID, N, contact, ToBeInsertedAt);
                    return;
                }
            }
            cout << "HashTable is full" << endl;
        }
    }

    void searchInHT(int EmpID) {
        int ToBeInsertedAt = hash(EmpID);
        if (ht[ToBeInsertedAt].getID() == EmpID) {
            ht[ToBeInsertedAt].printEmployee();
        } else {
```

```
        for (int i = 0; i < tableSize; i++) {
            ToBeInsertedAt = (ToBeInsertedAt + 1) % tableSize;

            if (ht[ToBeInsertedAt].getID() == EmpID) {
                ht[ToBeInsertedAt].printEmployee();
                return;
            }
        }
        cout << "Details for EmpID " << EmpID << " not found" << endl;
    }
}

void displayHT() {
    for (int i = 0; i < tableSize; i++) {
        cout << "Index " << i << ": ";
        ht[i].printEmployee();
    }
}
};

int main() {
    int size, choice, EmpID, contact;
    string Name;

    cout << "Enter the size of the hash table: ";
    cin >> size;
    HashTable ht1(size);

    do {
        cout << "\nMenu:\n";
        cout << "1. Insert Employee\n";
        cout << "2. Search Employee\n";
        cout << "3. Display Hash Table\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter Employee ID: ";
                cin >> EmpID;
                cout << "Enter Name: ";
                cin >> Name;
                cout << "Enter Contact: ";
                cin >> contact;
                ht1.insertIntoHT(EmpID, Name, contact);
                break;
```

```
            case 2:
                cout << "Enter Employee ID to search: ";
                cin >> EmpID;
                ht1.searchInHT(EmpID);
                break;

            case 3:
                cout << "Displaying Hash Table:\n";
                ht1.displayHT();
                break;

            case 4:
                cout << "Exiting program." << endl;
                break;

            default:
                cout << "Invalid choice. Please try again." << endl;
                break;
        }
    } while (choice != 4);

    return 0;
}
```

Screen Shot of Output :

Output Clear

```
Menu:
1. Insert Employee
2. Search Employee
3. Display Hash Table
4. Exit
Enter your choice: 1
Enter Employee ID: 123
Enter Name: Abhishek
Enter Contact: 1234567890

Menu:
1. Insert Employee
2. Search Employee
3. Display Hash Table
4. Exit
Enter your choice: 1
Enter Employee ID: 321
Enter Name: rajesh
Enter Contact: 0987654321

Menu:
1. Insert Employee
2. Search Employee
3. Display Hash Table
4. Exit
Enter your choice: 2
Enter Employee ID to search: 123
Details: 123 - Abhishek - 1234567890

Menu:
1. Insert Employee
2. Search Employee
3. Display Hash Table
4. Exit
Enter your choice: 3
Displaying Hash Table:
Index 0: Details: 123 - Abhishek - 1234567890
Index 1: Details: 321 - rajesh - 987654321
Index 2:
Menu:
1. Insert Employee
2. Search Employee
3. Display Hash Table
4. Exit
Enter your choice: 4
Exiting program.


=== Code Execution Successful ===
```

Conclusion: The program efficiently manages employee records using a hash table, supporting operations such as insertion, search, and display based on employee IDs.