PIMPRI CHINCHWAD EDUCATION TRUST's.

PIMPRI CHINCHWAD COLLEGE OF ENGINEERING

(An Autonomous Institute)

Class : SY BTech **Acad. Yr. 2025-26** **Semester : I**

Name of the student: Om Jitendra Khalane **PRN :** 124B1B040

Department: Computer Engineering **Division :** A

Course Name : Data Structures and Laboratory

Course Code: BCE23PC02

Completion Date : 12/08/2025

Problem Statement:

Consider an employee database of *N* employees containing **Employee ID** and **Name** as data members.

Implement a hash table to quickly look up an employee's ID number using **hashing with linear probing** for collision resolution.

Source Code :

https://github.com/omkhalane/DSAL-SY-PCCOE/blob/main/lab_assignments/assignment12.cpp

```cpp
// static
#include <bits/stdc++.h>
using namespace std;

// Employee
class Employee
{
public:
    int empId;
```

```
    string name;
    bool occupied;

    Employee()
    {
        empId = -1;
        name = "";
        occupied = false;
    }
};

class HashTable
{
private:
    int size;
    Employee *table;
    int insertCount;

    int hashFunction(int key)
    {
        return key % size;
    }

public:
    HashTable(int s)
    {
        size = s;
        table = new Employee[size];
        insertCount = 0;
    }

    void insertEmployee(int id, const string &name)
```

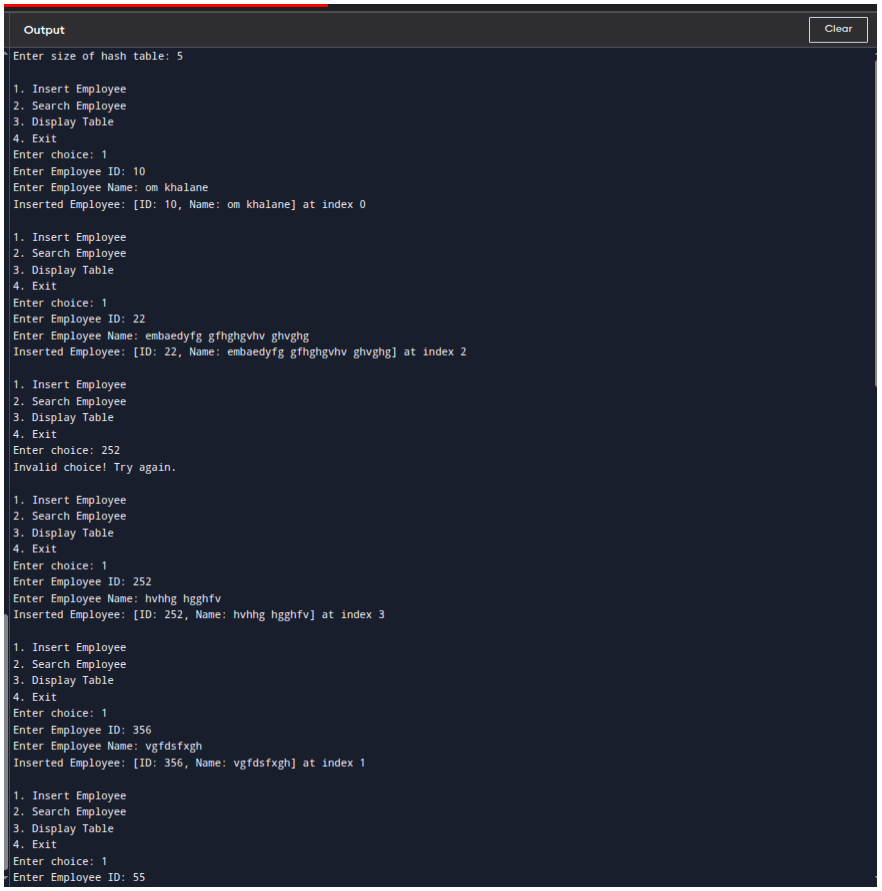
```
    {
        int index = hashFunction(id);
        int startIndex = index;

        while (table[index].occupied)
        {
            index = (index + 1) % size;
            if (index == startIndex)
            {
                cout << "Hash table is full! Cannot insert employee.\n";

                // static code and overwrite the first inserted employee
if table is full

                // int replaceIndex = insertCount % size;
                // cout << "Table full! Replacing oldest employee at
index "
                //      << replaceIndex << " (ID: " <<
table[replaceIndex].empId << ")\n";

                // table[replaceIndex].empId = id;
                // table[replaceIndex].name = name;
                // table[replaceIndex].occupied = true;

                // insertCount++;

                return;
            }
        }

        table[index].empId = id;
        table[index].name = name;
        table[index].occupied = true;
```

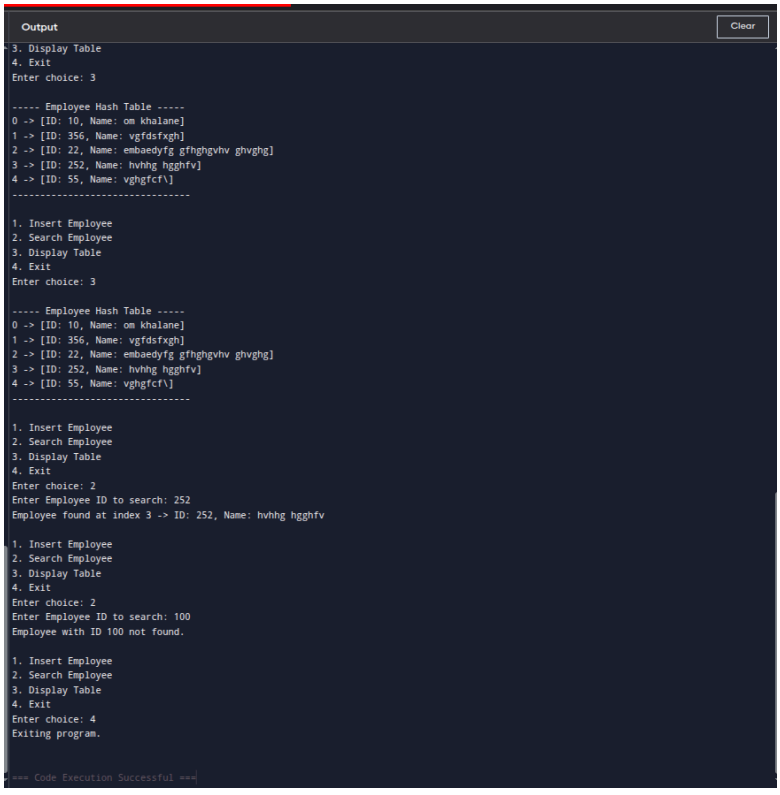
```
            insertCount++;
            cout << "Inserted Employee: [ID: " << id << ", Name: " << name
<< "] at index " << index << endl;
        }

        void searchEmployee(int id)
        {
            int index = hashFunction(id);
            int startIndex = index;

            while (table[index].occupied)
            {
                if (table[index].empId == id)
                {
                    cout << "Employee found at index " << index
                        << " -> ID: " << table[index].empId
                        << ", Name: " << table[index].name << endl;
                    return;
                }
                index = (index + 1) % size;
                if (index == startIndex)
                    break;
            }
            cout << "Employee with ID " << id << " not found.\n";
        }

        void displayTable()
        {
            cout << "\n----- Employee Hash Table -----\n";
            for (int i = 0; i < size; i++)
            {
```

```
                    if (table[i].occupied)
                        cout << i << " -> [ID: " << table[i].empId << ", Name: "
<< table[i].name << "]\n";
                    else
                        cout << i << " -> [Empty]\n";
                }
                cout << "--------------------------------\n";
            }
        };

        int main()
        {
            int size;
            cout << "Enter size of hash table: ";
            cin >> size;

            HashTable ht(size);

            int choice, id;
            string name;

            do
            {
                cout << "\n1. Insert Employee\n2. Search Employee\n3. Display
Table\n4. Exit\nEnter choice: ";
                cin >> choice;

                switch (choice)
                {
                case 1:
                    cout << "Enter Employee ID: ";
                    cin >> id;
```

```
                cout << "Enter Employee Name: ";
                cin.ignore();
                getline(cin, name);
                ht.insertEmployee(id, name);
                break;

            case 2:
                cout << "Enter Employee ID to search: ";
                cin >> id;
                ht.searchEmployee(id);
                break;

            case 3:
                ht.displayTable();
                break;

            case 4:
                cout << "Exiting program.\n";
                break;

            default:
                cout << "Invalid choice! Try again.\n";
            }

        } while (choice != 4);

        return 0;
    }
```

Output :


Output
Clear
Enter size of hash table: 5
1. Insert Employee
2. Search Employee
3. Display Table
4. Exit
Enter choice: 1
Enter Employee ID: 10
Enter Employee Name: om khalane
Inserted Employee: [ID: 10, Name: om khalane] at index 0
1. Insert Employee
2. Search Employee
3. Display Table
4. Exit
Enter choice: 1
Enter Employee ID: 22
Enter Employee Name: embaedyfg gfhghgvhv ghvghg
Inserted Employee: [ID: 22, Name: embaedyfg gfhghgvhv ghvghg] at index 2
1. Insert Employee
2. Search Employee
3. Display Table
4. Exit
Enter choice: 252
Invalid choice! Try again.
1. Insert Employee
2. Search Employee
3. Display Table
4. Exit
Enter choice: 1
Enter Employee ID: 252
Enter Employee Name: hvhhg hgghfv
Inserted Employee: [ID: 252, Name: hvhhg hgghfv] at index 3
1. Insert Employee
2. Search Employee
3. Display Table
4. Exit
Enter choice: 1
Enter Employee ID: 356
Enter Employee Name: vgfdsfxgh
Inserted Employee: [ID: 356, Name: vgfdsfxgh] at index 1
1. Insert Employee
2. Search Employee
3. Display Table
4. Exit
Enter choice: 1
Enter Employee ID: 55


Output
Clear
3. Display Table
4. Exit
Enter choice: 3
----- Employee Hash Table -----
0 -> [ID: 10, Name: om khalane]
1 -> [ID: 356, Name: vgfdsfxgh]
2 -> [ID: 22, Name: embaedyfg gfhghgvhv ghvghg]
3 -> [ID: 252, Name: hvhhg hgghfv]
4 -> [ID: 55, Name: vghgfcf\]
1. Insert Employee
2. Search Employee
3. Display Table
4. Exit
Enter choice: 3
----- Employee Hash Table -----
0 -> [ID: 10, Name: om khalane]
1 -> [ID: 356, Name: vgfdsfxgh]
2 -> [ID: 22, Name: embaedyfg gfhghgvhv ghvghg]
3 -> [ID: 252, Name: hvhhg hgghfv]
4 -> [ID: 55, Name: vghgfcf\]
1. Insert Employee
2. Search Employee
3. Display Table
4. Exit
Enter choice: 2
Enter Employee ID to search: 252
Employee found at index 3 -> ID: 252, Name: hvhhg hgghfv
1. Insert Employee
2. Search Employee
3. Display Table
4. Exit
Enter choice: 2
Enter Employee ID to search: 100
Employee with ID 100 not found.
1. Insert Employee
2. Search Employee
3. Display Table
4. Exit
Enter choice: 4
Exiting program.
=== Code Execution Successful ===

Conclusion:

- A **hash table** is an efficient data structure used to store key-value pairs for fast lookup.
- **Linear Probing** is a type of **open addressing** where, upon collision, the algorithm checks the next slot sequentially until an empty slot is found.
- In this program, each employee's **ID** acts as the key, and the **name** represents the associated data.

Time Complexity (TC):

- **Insertion:** O(1) average, O(n) worst case (when table is nearly full).
- **Search:** O(1) average, O(n) worst case (due to probing).
- **Deletion:** O(1) average, O(n) worst case.

Space Complexity (SC):

- O(n) for storing all employee records.