CHANDIGARH COLLEGE OF ENGINEERING & TECHNOLOGY (DEGREE WING)

Government institute under Chandigarh (UT) Administration, affiliated to Punjab University, Chandigarh

Department of Computer Science & Engineering

Semester: CSE 3rd

SUBJECT: Data Structures Practical (CS351)

Problem 10: Case Study of Hashing

Submitted by:

Bhavyam Dhand

(CO23316)

Submitted to:

Dr. R.B. Patel

(Professor)

CODE

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

#define M 10
#define STRING_LEN 50
#define EMPTY -1
#define DELETED -2

struct Employee {
    int KeyID;
    char name[STRING_LEN];
    char Department[STRING_LEN];
    bool Occupied;
};

void LogFile(const string& event) {
    ofstream BinFile("Machine_Code_Stack.txt", ios_base::app); // Open in
append mode
    if (BinFile.is_open()) {
        for (char c : event) {
            BinFile << bitset<8>(c) << " "; // Convert each character to
binary
        }
        BinFile << endl;
        BinFile.close();
    }
}

int HashFunction(int key) {
    return key % M;
}

void InitializeHashTable(Employee HT[]) {
    for (int i = 0; i < M; i++) {
        HT[i].KeyID = EMPTY;
        strcpy(HT[i].name, "");
        strcpy(HT[i].Department, "");
        HT[i].Occupied = false;
    }
    LogFile("Hash table initialized.");
}

void InsertEmployee(Employee HT[], int key, const char name[], const char
Department[]) {
    int index = HashFunction(key);
    int originalIndex = index;

    while (HT[index].Occupied && HT[index].KeyID != EMPTY) {
        index = HashFunction(index + 1);
        if (index == originalIndex) {
```

```
            LogFile("Failed to insert employee: Hash table is full.");
            cout << "Hash Table is Full!" << endl;
            return;
        }
    }

    HT[index].KeyID = key;
    strcpy(HT[index].name, name);
    strcpy(HT[index].Department, Department);
    HT[index].Occupied = true;

    stringstream ss;
    ss << "Inserted employee with KeyID: " << key << " at index: " << index;
    LogFile(ss.str());
    cout << ss.str() << endl;
}

void SearchEmployee(Employee HT[], int key) {
    int index = HashFunction(key);
    int originalIndex = index;

    while (HT[index].Occupied) {
        if (HT[index].KeyID == key) {
            stringstream ss;
            ss << "Found employee with KeyID: " << key << " at index: " <<
index;
            LogFile(ss.str());
            cout << "Employee found at index " << index << ": "
                 << HT[index].name << ", Department: " << HT[index].Department
<< endl;
            return;
        }
        index = HashFunction(index + 1);
        if (index == originalIndex) {
            break;
        }
    }
    LogFile("Search failed for employee with KeyID: " + to_string(key));
    cout << "Employee not found." << endl;
}

void DeleteEmployee(Employee HT[], int key) {
    int index = HashFunction(key);
    int originalIndex = index;

    while (HT[index].Occupied) {
        if (HT[index].KeyID == key) {
            HT[index].KeyID = DELETED;
            strcpy(HT[index].name, "");
            strcpy(HT[index].Department, "");
            HT[index].Occupied = false;
```

```
            stringstream ss;
            ss << "Deleted employee with KeyID: " << key << " from index: " <<
index;
            LogFile(ss.str());
            cout << ss.str() << endl;
            return;
        }
        index = HashFunction(index + 1);
        if (index == originalIndex) break;
    }
    LogFile("Failed to delete: Employee with KeyID " + to_string(key) + " not
found.");
    cout << "Employee not found. Cannot delete element." << endl;
}

void DisplayTable(Employee HT[]) {
    cout << "\nHash Table:" << endl;
    for (int i = 0; i < M; i++) {
        if (HT[i].Occupied && HT[i].KeyID != EMPTY) {
            cout << "Index: " << i << " Key: " << HT[i].KeyID
                 << " ,Name: " << HT[i].name
                 << " ,Department: " << HT[i].Department << endl;
        } else {
            cout << "Index: " << i << " Empty" << endl;
        }
    }
    LogFile("Displayed hash table contents.");
}

int main() {
    Employee HT[M];
    InitializeHashTable(HT);

    int choice, key;
    char name[STRING_LEN], department[STRING_LEN];

    do {
        cout << "\n--- Hash Table Menu ---\n";
        cout << "1. Insert Employee\n";
        cout << "2. Search Employee\n";
        cout << "3. Delete Employee\n";
        cout << "4. Display Hash Table\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter Key, Name, and Department: ";
                cin >> key;
                cin.ignore();
                cin.getline(name, STRING_LEN);
```

```
                cin.getline(department, STRING_LEN);
                InsertEmployee(HT, key, name, department);
                break;
            case 2:
                cout << "Enter Key to search: ";
                cin >> key;
                SearchEmployee(HT, key);
                break;
            case 3:
                cout << "Enter Key to delete: ";
                cin >> key;
                DeleteEmployee(HT, key);
                break;
            case 4:
                DisplayTable(HT);
                break;
            case 5:
                LogFile("Exiting program.");
                cout << "Exiting program.\n";
                break;
            default:
                LogFile("Invalid menu option selected.");
                cout << "Invalid choice. Try again.\n";
        }
    } while (choice != 5);

    return 0;
}
```

Output

1. Insert a record using Linear probing

```
--- Hash Table Menu ---
1. Insert Employee
2. Search Employee
3. Delete Employee
4. Display Hash Table
5. Exit
Enter your choice: 1
Enter Key, Name, and Department: 3246
Vansh
Accounting
Inserted employee with KeyID: 3246 at index: 6
```

2. Delete a Record from Hash Table

```
--- Hash Table Menu ---
1. Insert Employee
2. Search Employee
3. Delete Employee
4. Display Hash Table
5. Exit
Enter your choice: 3
Enter Key to delete: 2025
Deleted employee with KeyID: 2025 from index: 5
```

3. Search a Record in Hash Table

```
--- Hash Table Menu ---
1. Insert Employee
2. Search Employee
3. Delete Employee
4. Display Hash Table
5. Exit
Enter your choice: 2
Enter Key to search: 2025
Employee found at index 5: Tarun, Department: CSE
```

4. Display Hash Table

```
--- Hash Table Menu ---
1. Insert Employee
2. Search Employee
3. Delete Employee
4. Display Hash Table
5. Exit
Enter your choice: 4

Hash Table:
Index: 0 Empty
Index: 1 Empty
Index: 2 Empty
Index: 3 Empty
Index: 4 Empty
Index: 5 Key: 2025 ,Name: Tarun ,Department: CSE
Index: 6 Key: 3246 ,Name: Harsh ,Department: Accounting
Index: 7 Empty
Index: 8 Empty
Index: 9 Empty
```

Machine Code:

01001000 01100001 01110011 01101000 00100000 01110100 01100001 01100010 01101100 01100101 00100000 01101001 01101110 01101001 01110100 01101001 01100001 01101100 01101001 01111010 01100101 01100100 00101110

01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000 01100101 01101101 01110000 01101100 01101111 01111001 01100101 01100101 00100000 01110111 01101001 01110100 01101000 00100000 01001011 01100101 01111001 01001001 01000100 00111010 00100000 00110011 00110100 00110010 00111000 00100000 01100001 01110100 00100000 01101001 01101110 01100100 01100101 01111000 00111010 00100000 00111000

01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000 01100101 01101101 01110000 01101100 01101111 01111001 01100101 01100101 00100000 01110111 01101001 01110100 01101000 00100000 01001011 01100101 01111001 01001001 01000100 00111010 00100000 00110010 00110010 00110100 00110001 00100000 01100001 01110100 00100000 01101001 01101110 01100100 01100101 01111000 00111010 00100000 00110001

01000100 01101001 01110011 01110000 01101100 01100001 01111001 01100101 01100100 00100000 01101000 01100001 01110011 01101000 00100000 01110100 01100001 01100010 01101100 01100101 00100000 01100011 01101111 01101110 01110100 01100101 01101110 01110100 01110011 00101110