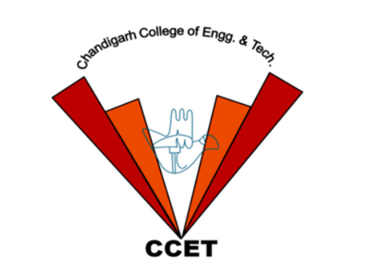
**CHANDIGARH COLLEGE OF ENGINEERING & TECHNOLOGY (DEGREE WING)**

****

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:                                                Submitted to:**

Bhavyam Dhand                                              Dr. R.B. Patel

(CO23316)                 (Professor)

**CODE**

#include <bits/stdc++.h>

#include <vector>

#include <string>

#include <fstream>

using namespace std;

//to depict empty entries as empty

#define EMPTY -1

#define DELETED -2

//function to verify name is alphabetical

bool VerifyName(string name) {

    for (char c : name) {

        if (!isalpha(c) && c != '.' && c != ' ') return false;

    }

    return true;

}

struct EmployeeHashNode {

    int key;

    string name;

    string gender;

    string mobile;

    string email;

    string qualifications;

    bool occupied;

    // Constructor to initialize an empty node

    EmployeeHashNode() : key(EMPTY), name(""), gender(""), mobile(""), email(""), qualifications(""), occupied(false) {}

    //Function to input Node data

    void Input(int Key) {

        this->key = Key;

        do {

            cout << "Enter name: ";

            getline(cin, name);

        } while (!VerifyName(name));

        cout << "Enter gender: ";

        getline(cin, gender);

        cout << "Enter Phone Number: ";

        getline(cin, mobile);

        cout << "Enter Email-id: ";

        getline(cin, email);

        cout << "Enter Qualifications: ";

        getline(cin, qualifications);

        occupied = true;

    }

    //function which displays Node Data

    void Display() const {

        if (occupied) {

            cout << "Key: " << key << "  Name: " << name

                 << "  Gender: " << gender

                 << "  Mobile: " << mobile << "  Email: " << email

                 << "  Qualifications: " << qualifications << endl;

        } else {

            cout <<" Empty" << endl;

        }

    }

};

//Datatype which depicts HashTable

struct HashTable {

    vector<EmployeeHashNode> Table;

    int M;

    HashTable(int MaxSize) : M(MaxSize) {

        Table.resize(MaxSize);

    }

};

// Hash function to compute the index

int HashFunction(int key, int M) {

    return key % M;  // Simple modulus hash function

}

// Insert a node into the hash table

void insert(HashTable& H, const EmployeeHashNode& Node) {

    int index = HashFunction(Node.key, H.M);

    int OGIndex = index;

    while (H.Table[index].occupied && (H.Table[index].key != EMPTY && H.Table[index].key != DELETED)) {

        index = (index + 1) % H.M;

        if (index == OGIndex) {

            cout << "Hash table is full!\n";

            return;

        }

    }

    H.Table[index] = Node;

}

// Delete a node from the hash table

void Delete(HashTable& H, int key) {

    int index = HashFunction(key, H.M);

    int originalIndex = index;

    while (H.Table[index].occupied) {

        if (H.Table[index].key == key) {

            H.Table[index] = EmployeeHashNode();

            H.Table[index].key = DELETED;

            cout << "Deleted employee with key: " << key << "\n";

            return;

        }

        index = (index + 1) % H.M;

        if (index == originalIndex)

            break;

    }

    cout << "Employee with key " << key << " not found.\n";

}

// Search for a node by key in the hash table

void Search(const HashTable& H, int key) {

    int index = HashFunction(key, H.M);

    int OGindex = index;

    while (H.Table[index].occupied) {

        if (H.Table[index].key == key) {

            cout << "Employee found:\n";

            H.Table[index].Display();

            return;

        }

        index = (index + 1) % H.M;

        if (index == OGindex)

            break;

    }

    cout << "Employee not found.\n";

}

// Display all nodes in the hash table

void Display(const HashTable& H) {

    for (int i = 0; i < H.M; ++i) {

            cout << "Index " << i << ": ";

            H.Table[i].Display();

    }

}

// Update the details of an employee

void Update(HashTable& H, int key) {

    int index = HashFunction(key, H.M);

    int originalIndex = index;

    while (H.Table[index].occupied) {

        if (H.Table[index].key == key) {

            cout << "Employee found. Enter new details:\n";

            H.Table[index].Input(key);

            cout << "Employee details updated successfully.\n";

            return;

        }

        index = (index + 1) % H.M;

        if (index == originalIndex)

            break;

    }

    cout << "Employee with key " << key << " not found.\n";

}

//load data from emp.dat file

void LoadFromFile(HashTable& H, const string& filename) {

    ifstream infile(filename);

    if (!infile) {

        cout << "Error opening file: " << filename << endl;

        return;

    }

    string name, gender, mobile, email, qualifications;

    int key;

    while (infile >> key) {

        // Skip the newline character after the key

        infile.ignore();

        // Read the name, gender, mobile, email, and qualifications

        getline(infile, name, ' ');  // Read until the first space for name

        getline(infile, gender, ' '); // Read until the next space for gender

        getline(infile, mobile, ' '); // Read until the next space for mobile

        getline(infile, email, ' ');  // Read until the next space for email

        getline(infile, qualifications); // Read the rest for qualifications

        // Create an EmployeeHashNode with the parsed data

        EmployeeHashNode Node;

        Node.key = key;

        Node.name = name;

        Node.gender = gender;

        Node.mobile = mobile;

        Node.email = email;

        Node.qualifications = qualifications;

        Node.occupied = true;

        // Insert the node into the hash table

        insert(H, Node);

    }

    infile.close();

    cout << "Data loaded from file successfully.\n";

}

//user interface

int main() {

    int size;

    cout << "Enter hash table size: ";

    cin >> size;

    cin.ignore();

    HashTable table(size);

    int choice;

    do {

        cout << "\nMenu:\n1. Insert\n2. Delete\n3. Search\n4. Display\n5. Load from file\n6. Update\n7. Exit\nEnter your choice: ";

        cin >> choice;

        cin.ignore();

        switch (choice) {

            case 1: {

                EmployeeHashNode Node;

                int key;

                cout << "Enter key: ";

                cin >> key;

                cin.ignore();

                Node.Input(key);

                insert(table, Node);

                break;

            }

            case 2: {

                int key;

                cout << "Enter key to delete: ";

                cin >> key;

                Delete(table, key);

                break;

            }

            case 3: {

                int key;

                cout << "Enter key to search: ";

                cin >> key;

                Search(table, key);

                break;

            }

            case 4:

                Display(table);

                break;

            case 5:

                LoadFromFile(table, "EMP.dat");

                break;

            case 6: {

                int key;

                cout << "Enter key to update: ";

                cin >> key;

                Update(table, key);

                break;

            }

            case 7:

                cout << "Exiting program.\n";

                break;

            default:

                cout << "Invalid choice. Please try again.\n";

        }

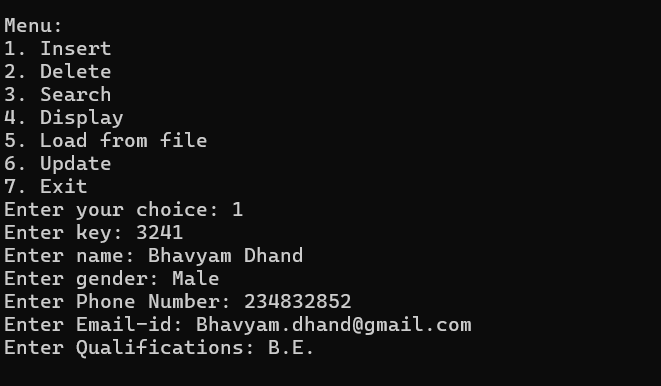
    } while (choice != 7);

    return 0;

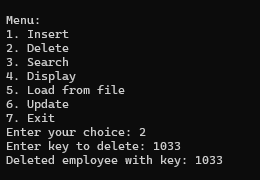
}

**Output**

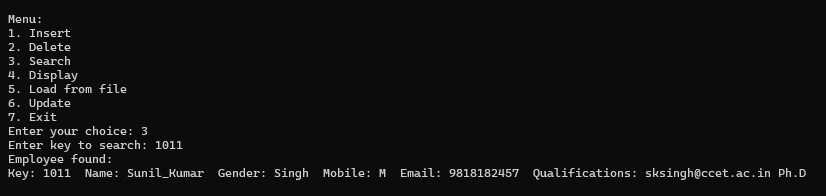
1. **Insert a record using Linear probing**

****

1. **Delete a Record from Hash Table**

****

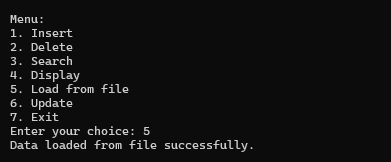
1. **Search a Record in Hash Table**

****

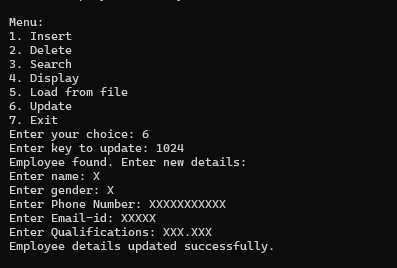
1. **Display Hash Table**

****

1. **Load from EMP.dat file:**

****

1. **Update Record**

****

**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**