CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY
(DEGREE WING)
Government Institute under Chandigarh (UT) Administration, Affiliated to Panjab University , Chandigarh
Sector-26, Chandigarh.PIN-160019

Data Structure Assignments

Submitted By: - Abhay Pratap Singh

Roll No: - CO23306

Branch: - Computer Science and Engineering

Professor In charge of Subject: - Dr R.B Patel

Subject Incharge
Dr. R.B Patel

Name :- Abhay Pratap Singh
Branch :- CSE
Roll No :- C023306

Index Problem :- 10

SNo	Content	Pg No
1.)	Discussion About Problem	1-2
2.)	Block Diagram	3
3.)	Pseudocode And Algorithms	4-8
4.)	Code	9-14
5.)	Data file	15
6.)	Outputs	16-18
7.)	lesson learned from problem	19

Date of Assignment :- 11/11/2024

Date of Submission :- 22/11/2024

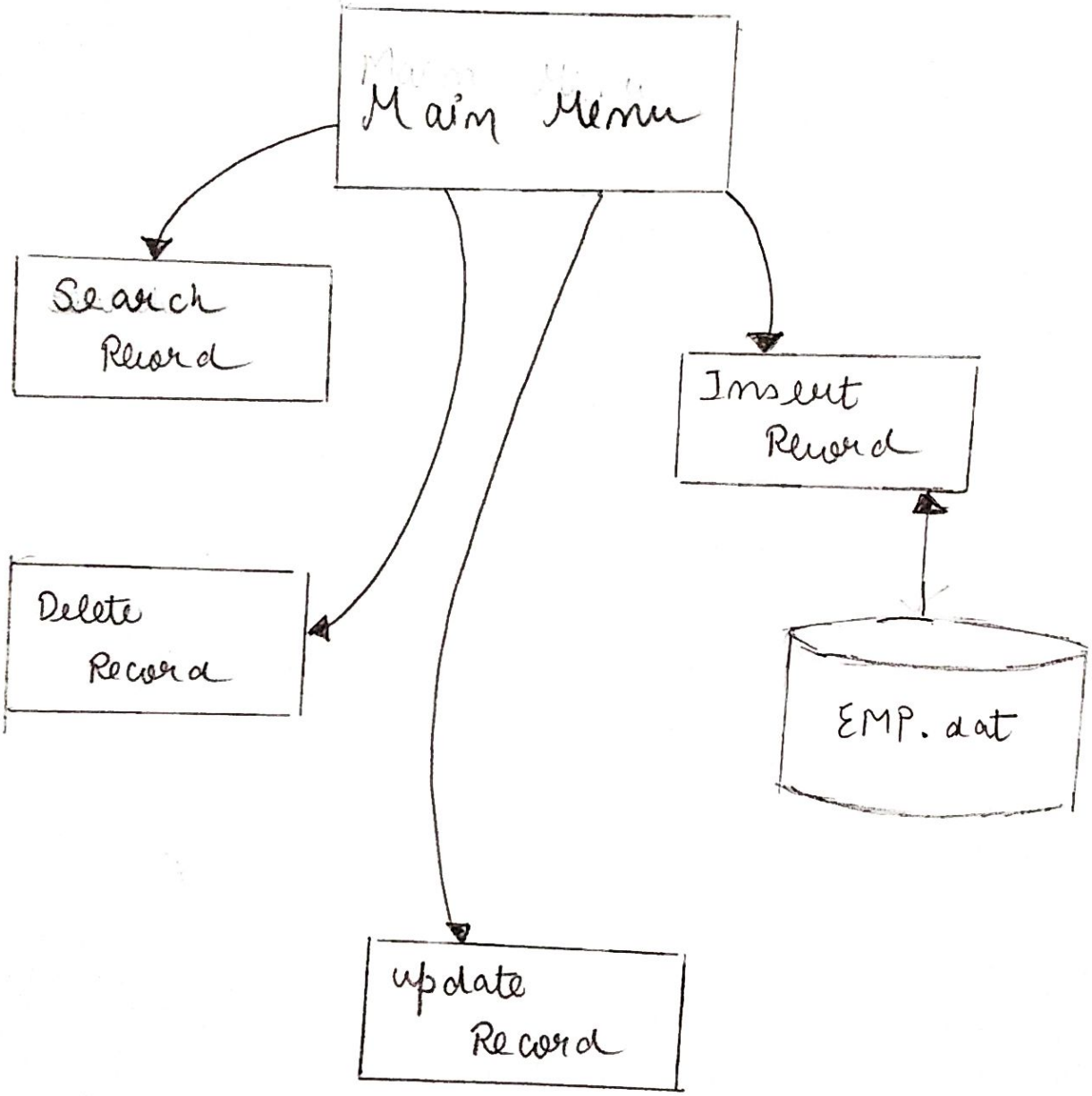
Date of Assignment: 11.11.20204
Date of Submission: 22.11.2024

Problem 10: Case Study of Hashing: Given a File of N employee records of company CCET with a set K of Keys(4-digit) which uniquely determine the records in file EMP.dat. Assume that file EMP.dat is maintained in memory by a Hash Table (HT) of **M** memory locations with **L** as the set of memory addresses (2-digit) of locations in HT. Let the keys in **K** and addresses in **L** are integers. Write a modular program in C/C++ that uses Hash function H: K → L as H(K)=K mod M (remainder method), and implement hashing technique to map a given key **K** to the address space **L**. Resolve the collision (if any) using linear probing.

0→ Discussion About Problem :-

The problem involves implementing a modular hashing system to store employee records using hashtable (HT) with M memory locations, where each employee is identified by a unique 4-digit key. The hash function H(K) = K mod M is used to map the 4-digit employee key K to a memory location L, but collisions can occur when multiple keys map to the same location. To resolve this linear probing is employed, where, in case of a collision, the algorithm checks the next available location subsequently until an empty slot is found. This approach ensures that the hash table efficiently stores & retrieves employee records by minimizing search times. The solution requires careful implementation of key-to-location mapping, collision handling, and ensuring the table scales with the number of records (N) and available locations (M). The use of C/C++ makes it possible to optimize for memory & access speed while ensuring a functional & efficient system.

Block Diagram



Topic _____ Date _____

0→ Pseudocode And Growth functions :-

1) HashTable Constructor :-

Pseudocode :-

```
function HashTable (size) {                    0
Initialize M as size                           m
Initialize table as an empty vector of size M }  m
```

- Growth function :- 2m.
- Time Complexity :- O(M)
- Space Complexity :- O(M)

2) Hash function :-

Pseudocode :-

```
function hashFunction (key) {                  0
Return key % M }                               1
```

- Growth function :- 1
- Time Complexity :- O(1)
- Space Complexity :- O(1)

3) Insert Employee :-

Pseudocode :-

```
function insert (emp) {                        0
set Index = hashfunction (emp.key)             1
Set originalIndex = Index                      1
while table [Index] is not empty {             m
 if (table [index].key = emp.key ) {           .1
  Print "Duplicate Key !" and return }         1
```

Topic _____ Date _____

```
Increment Index (handle collision with linear probing)
If index equals original Index {                    1
Print "Hash Table is full!" and return }            1
Insert emp at table[index] ;                        1
```

- Growth function :- 2n+7
- Time Complexity :- O(n)
- Space Complexity :- O(1)

4.) Search Employee :-

Pseudocode :-

```
Function search (key) {                             0
set index = hashfunction(key)                       1
set original Index = Index                          1
while table[index] is not empty {                   n
if (table[index].key = key) {                       n
return table[index] ;                               n
incrementIndex                                      n
if (index = originalIndex) {                        n
return nullptr ; } }                                n
```

- Growth function :- 6n+2
- Time Complexity :- O(n)
- Space Complexity :- O(1)

5.) Delete Employee :-

Pseudocode :-

```
Function deleteRecord (key) {                                   0
Set index = hashfunction(key)                                   1
Set originalIndex = Index                                       1
While table[index] is not empty {                               n
If (table[index].key equals key) {                              1
Set table[index].key to 0                                       1
Print "Employee deleted"                                        1
return }                                                        1
increment index                                                 n
If (index equals originalIndex)(i.e loop through
the table) {                                                    n
Print "Employee not found"                                      n
return }                                                        n
```

- Growth functions :- $5n+6$
- Time Complexity :- $O(M)$
- Space Complexity :- $O(1)$

6) Update :-

Pseudo code :-

```
Function update (key, updateEmp) {                              0
Set index = hashfunction(key)                                   1
Set originalIndex = Index                                       1
while table[Index] is not empty {                               n
if (table[index].key = key) {                                   n
Replace table[Index] with updated Emp                           m
Print "Employee updated"                                        m
Return }                                                        n
```

Topic _____ Date _____

```
Increment index (handle collision)                    n
if (index = original Index) {                         n
  print "Employee not found!"                         n
  return; }                                           n
```

- Growth function :- 9n+2
- Time Complexity :- O(N)
- Space Complexity :- O(1)

7.) Display :-

Pseudocode :-

```
function display() {                                  0
for each index from 0 to M-1 {                        n
if (table[index] ! empty) {                           n
print the employee's details at table[index] }        n
  Else {                                              n
print "empty" for that index } }                      n
```

- Growth function :- 5n.
- Time Complexity :- O(N)
- Space Complexity :- O(1)

8.) Load from file :-

Pseudocode :-

```
function load from file (filename) {                  0
open file with name filename                          1
if (file cannot be opened) {                          1
  print "Failed to open file"                         1
  return }                                            1
```

Topic _____ Date _____

```
While (!EOF) {                                          0
  Print Data                                            n
Create a new Employee object with parsed data           n
Insert the Employee object into hash table using insert n
  (emp) ;                                               n
Close the file                                          1
Print "Employees loaded from file successfully!" ;      1
```

- Growth function :- 4n + 6
- Time Complexity :- O(M)
- Space Complexity :- O(N)

9.) Enter Employee Details :-

```
function enterEmployeeDetails (ht) {                    0
Create an empty Employee emp                            1
Repeat until valid key is entered {                     n
Prompt user for employee key                            n
If (key exists in hash table) {                         n
Print "Duplicate key! Please enter a different key" ;   n
else if (key is not a 4-digit number) {                 n
Print "Invalid input! Please enter 4 digit key"         n
else { Break; }                                         1
Prompt user for details                                 1
Return emp                                              1
```

- Growth function :- 6n + 4
- Time Complexity :- O(M)
- Space Complexity :- O(1)

CODE

```
1: /*Problem-10 Code
2: Roll-No :- co23306
3: Name :- Abhay Pratap Singh
4: Cse, 2nd Year*/
5:
6: #include <bits/stdc++.h>
7: using namespace std;
8:
9: struct Employee {
10:     int key;             // Unique 4-digit key for each employee
11:     string name;
12:     int age;
13:     string gender;
14:     string phoneNumber;
15:     string email;
16:     string qualification;
17:
18:     // Default constructor for initialization
19:     Employee() : key(0), age(0) {}
20: };
21:
22: // HashTable Class
23: class HashTable {
24: private:
25:     vector<Employee> table;
26:     int M; // Number of slots in the hash table
27:
28:     // Hash function using remainder method (K % M)
29:     int hashFunction(int key) {
30:         return key % M;
31:     }
32:
33: public:
34:     // Constructor to initialize hash table with size M
35:     HashTable(int size) : M(size) {
36:         table.resize(M);  // Resize vector to M slots
37:     }
38:
39:     // Insert an employee record into the hash table
40:     void insert(const Employee& emp) {
41:         int index = hashFunction(emp.key);
42:         int originalIndex = index;
43:
44:         // Linear probing to handle collisions
45:         while (table[index].key != 0) { // 0 means the slot is empty
46:             if (table[index].key == emp.key) {
47:                 cout << "Duplicate key! Employee with key " << emp.key << " already
    exists." << endl;
48:                 return;  // Return early if key already exists
49:             }
50:             index = (index + 1) % M;
51:             if (index == originalIndex) {
52:                 cout << "Hash Table is full!" << endl;
53:                 return;
54:             }
```

```
        }
        table[index] = emp;
    }

    // Search for an employee by key
    Employee* search(int key) {
        int index = hashFunction(key);
        int originalIndex = index;

        while (table[index].key != 0) {
            if (table[index].key == key) {
                return &table[index];
            }
            index = (index + 1) % M;
            if (index == originalIndex) {
                break;
            }
        }
        return nullptr;  // Employee not found
    }

    // Delete an employee record by key
    void deleteRecord(int key) {
        int index = hashFunction(key);
        int originalIndex = index;

        while (table[index].key != 0) {
            if (table[index].key == key) {
                table[index].key = 0;  // Set the slot to empty
                cout << "Employee with key " << key << " deleted." << endl;
                return;
            }
            index = (index + 1) % M;
            if (index == originalIndex) {
                break;
            }
        }
        cout << "Employee with key " << key << " not found!" << endl;
    }

    // Update an existing employee's record
    void update(int key, const Employee& updatedEmp) {
        int index = hashFunction(key);
        int originalIndex = index;

        while (table[index].key != 0) {
            if (table[index].key == key) {
                table[index] = updatedEmp;  // Update employee details
                cout << "Employee with key " << key << " updated." << endl;
                return;
            }
            index = (index + 1) % M;
            if (index == originalIndex) {
                break;
            }
```

```
110:         }
111:         cout << "Employee with key " << key << " not found!" << endl;
112:     }
113: 
114:     // Display the hash table
115:     void display() {
116:         for (int i = 0; i < M; i++) {
117:             if (table[i].key != 0) {
118:                 // Print employee details
119:                 cout << "Index " << i << ": "
120:                      << table[i].key << " "
121:                      << table[i].name << " "
122:                      << table[i].age << " "
123:                      << table[i].gender << " "
124:                      << table[i].phoneNumber << " "
125:                      << table[i].email << " "
126:                      << table[i].qualification << endl;
127:             } else {
128:                 // Print "empty" for unused slots
129:                 cout << "Index " << i << ": empty" << endl;
130:             }
131:         }
132:     }
133: 
134:     // Load employees from file
135:     void loadFromFile(const string& filename) {
136:         ifstream file(filename);
137:         if (!file) {
138:             cout << "Failed to open file: " << filename << endl;
139:             return;
140:         }
141: 
142:         string line;
143:         while (getline(file, line)) {
144:             stringstream ss(line);
145:             Employee emp;
146:             ss >> emp.key;
147:             ss.ignore();  // Ignore space after key
148:             getline(ss, emp.name, '\t');
149:             ss >> emp.age;
150:             ss.ignore();  // Ignore space after age
151:             getline(ss, emp.gender, '\t');
152:             getline(ss, emp.phoneNumber, '\t');
153:             getline(ss, emp.email, '\t');
154:             getline(ss, emp.qualification);
155: 
156:             insert(emp); // Insert the employee into the hash table
157:         }
158: 
159:         file.close();
160:         cout << "Employees loaded from file successfully!" << endl;
161:     }
162: };
163: 
164: // Function to prompt user to enter employee details and ensure the key is not a duplicate
```

```
165: Employee enterEmployeeDetails(HashTable& ht) {
166:     Employee emp;
167:     cout << "Enter employee details:" << endl;
168: 
169:     // Ensure the key is a 4-digit number and not already in the hash table
170:     do {
171:         cout << "Employee Key (4-digit number): ";
172:         cin >> emp.key;
173: 
174:         // Check if the key exists in the hash table
175:         if (ht.search(emp.key)) {
176:             cout << "Duplicate key! Employee with key " << emp.key << " already exists.
    Please enter a different key." << endl;
177:         } else if (emp.key < 1000 || emp.key > 9999) {
178:             cout << "Invalid input! Please enter a 4-digit key." << endl;
179:         } else  break;
180:             break; // Valid key and no duplicates
181:         }
182:     } while (true);  // Repeat until valid key is entered
183: 
184:     cin.ignore();  // Ignore the newline character after the key input
185: 
186:     cout << "Name: ";
187:     getline(cin, emp.name);
188:     cout << "Age: ";
189:     cin >> emp.age;
190:     cin.ignore();  // Ignore the newline after age input
191:     cout << "Gender (M/F): ";
192:     getline(cin, emp.gender);
193:     cout << "Phone Number: ";
194:     getline(cin, emp.phoneNumber);
195:     cout << "Email: ";
196:     getline(cin, emp.email);
197:     cout << "Qualification: ";
198:     getline(cin, emp.qualification);
199:     return emp;
200: }
201: 
202: int main() {
203:     // Step 1: Get the size of the hash table from the user
204:     int M;
205:     cout << "Enter the size of the hash table: ";
206:     cin >> M;
207: 
208:     // Step 2: Create a hash table with size M
209:     HashTable ht(M);
210: 
211:     int choice;
212:     do {
213:         cout << "\nMenu:\n";
214:         cout << "1. Insert Employee Manually\n";
215:         cout << "2. Load Employees from File\n";
216:         cout << "3. Search Employee\n";
217:         cout << "4. Update Employee\n";
218:         cout << "5. Delete Employee\n";
```

```
219:         cout << "6. Display Hash Table\n";
220:         cout << "7. Exit\n";
221:         cout << "Enter your choice: ";
222:         cin >> choice;
223:         cin.ignore();  // Ignore the newline character after choice input
224:
225:         switch (choice) {
226:             case 1: {
227:                 // Insert employee manually
228:                 Employee emp = enterEmployeeDetails(ht);  // Pass hash table to check for
    duplicate keys
229:                 ht.insert(emp);
230:                 break;
231:             }
232:             case 2: {
233:                 // Load employees from file
234:                 ht.loadFromFile("EMP.dat");
235:                 break;
236:             }
237:             case 3: {
238:                 // Search employee by key
239:                 int keyToSearch;
240:                 cout << "Enter the employee key to search: ";
241:                 cin >> keyToSearch;
242:                 Employee* emp = ht.search(keyToSearch);
243:                 if (emp) {
244:                     cout << "\nFound employee with key " << keyToSearch << ": "
245:                          << emp->name << " " << emp->age << " "
246:                          << emp->gender << " " << emp->phoneNumber << " "
247:                          << emp->email << " " << emp->qualification << endl;
248:                 } else {
249:                     cout << "Employee with key " << keyToSearch << " not found!" << endl;
250:                 }
251:                 break;
252:             }
253:             case 4: {
254:                 // Update employee details
255:                 int keyToUpdate;
256:                 cout << "Enter the employee key to update: ";
257:                 cin >> keyToUpdate;
258:                 Employee updatedEmp = enterEmployeeDetails(ht);  // Pass hash table to
    check for duplicate keys
259:                 ht.update(keyToUpdate, updatedEmp);
260:                 break;
261:             }
262:             case 5: {
263:                 // Delete employee
264:                 int keyToDelete;
265:                 cout << "Enter the employee key to delete: ";
266:                 cin >> keyToDelete;
267:                 ht.deleteRecord(keyToDelete);
268:                 break;
269:             }
270:             case 6: {
271:                 // Display the hash table
```

```
272:                 cout << "\nHash Table Contents:" << endl;
273:                 ht.display();
274:                 break;
275:             }
276:             case 7:
277:                 cout << "Exiting program." << endl;
278:                 break;
279:             default:
280:                 cout << "Invalid choice! Please try again." << endl;
281:         }
282:     } while (choice != 7);
283: 
284:     return 0;
285: }
```

EMP.dat

```
 1: 1000 Varun Gupta 45 Male 9646047091 varungupta@ccet.ac.in Ph.D
 2: 1001 Manveen Kaur 35 Female 9988957007 manveenkaur@ccet.ac.in Ph.D
 3: 1002 Parul Aggarwal 43 Female 8437911722 parul@ccet.ac.in Ph.D
 4: 1003 Neha 35 Female 9646614209 neha@ccet.ac.in M.Sc.
 5: 1004 Rajesh Kumar 51 Male 9478548248 rajeshaastha@ccet.ac.in Ph.D
 6: 1005 Aradhana Mehta 57 Female 8054977561 amehta@ccet.ac.in Ph.D
 7: 1006 Mohammad Sakib Perwez Khan 37 Male 7839452836 sakib786@ccet.ac.in
    M.Tech
 8: 1007 Poonam 40 Female 8968399719 poonam@ccet.ac.in M.Tech
 9: 1008 Anil Kumar 42 Male 9816290720 anilkumar@ccet.ac.in M.Tech
10: 1009 Karuna Sharma 41 Female 8283833589 karunasharma@ccet.ac.in M.E.
11: 1010 Arfat Ahmed 39 Male 8860736206 arfat@ccet.ac.in M.Tech
12: 1011 Sunil Kumar Singh 42 Male 9818182457 sksingh@ccet.ac.in Ph.D
13: 1012 Ram Bahadur Patel 56 Male 9416932840 rbpatel@ccet.ac.in Ph.D
14: 1013 Dheerendra Singh 45 Male 9876439071 dsingh@ccet.ac.in Ph.D
15: 1014 Gulshan Goyal 41 Male 9417506206 gulshangoyal@ccet.ac.in Ph.D
16: 1015 Sunita 42 Female 9041059379 sunita@ccet.ac.in Ph.D
17: 1016 Amit Chhabra 41 Male 9888623825 amitchhabra@ccet.ac.in Ph.D
18: 1017 Ankit Gupta 41 Male 9412314479 ankit@ccet.ac.in Ph.D
19: 1018 Sarabjeet Singh 44 Male 9463739413 ssingh@ccet.ac.in Ph.D
20: 1019 Sudhakar Kumar 34 Male 8434518635 sudhakar@ccet.ac.in M.Tech
21: 1020 Animesh Singh 38 Male 9584035345 animeshsingh@ccet.ac.in M.Tech
22: 1021 Davinder Singh Saini 46 Male 8146730369 dssaini@ccet.ac.in Ph.D
23: 1022 Krishan Gopal Sharma 44 Male 9414403565 kgsharma@ccet.ac.in Ph.D
24: 1023 Bhasker Gupta 42 Male 9855908643 bgupta@ccet.ac.in Ph.D
25: 1024 Anil Kumar 43 Male 9416234853 anilrose@ccet.ac.in Ph.D
26: 1025 Dr Parvinder Kaur 40 Female 8295688911 pkaur@ccet.ac.in Ph.D
27: 1026 Shilpa Jindal 41 Female 9463328881 shilpajindal@ccet.ac.in Ph.D
28: 1027 Dinesh Sharma 39 Male 9671721850 dsharma@ccet.ac.in Ph.D
29: 1028 Irfan Ahmad Khan 43 Male 7835847022 iakhan@ccet.ac.in Ph.D
30: 1029 Sarita Sharma 45 Female 9988292611 saritasharma@ccet.ac.in Ph.D
31: 1030 Hardeep Saini 36 Male 9914611106 hsaini@ccet.ac.in M.Tech
32: 1031 Anil Kumar Vaghmare 44 Male 6284561607 anilvaghmare@ccet.ac.in Ph.D
33: 1032 Jatinder Madan 52 Male 9041291970 jatindermadan@ccet.ac.in Ph.D
34: 1033 Vettivel S C 42 Male 9865822376 scvettivel@ccet.ac.in Ph.D
35: 1034 Radhey Sham 46 Male 9888040982 radheysham@ccet.ac.in Ph.D
36: 1035 Mukesh Kumar 40 Male 9478420561 mukeshkumar@ccet.ac.in Ph.D
37: 1036 Vinod Chauhan 40 Male 9466736896 vinodchauhan@ccet.ac.in M.E.
38: 1037 Ashwani Kumar 40 Male 9872823250 ashwanikumar@ccet.ac.in Ph.D
39: 1038 Rajiv Kumar 37 Male 9877887402 rajivkumar@ccet.ac.in M.Tech
40: 1039 Nipun Sharma 39 Male 9877726260 nipun@ccet.ac.in M.Tech
```

Outputs

- Loading data from file :-

```
Enter the size of the hash table: 50

Menu:
1. Insert Employee Manually
2. Load Employees from File
3. Search Employee
4. Update Employee
5. Delete Employee
6. Display Hash Table
7. Exit
Enter your choice: 2
Employees loaded from file successfully!
```

- Inserting data manually :-

```
Menu:
1. Insert Employee Manually
2. Load Employees from File
3. Search Employee
4. Update Employee
5. Delete Employee
6. Display Hash Table
7. Exit
Enter your choice: 1
Enter employee details:
Employee Key (4-digit number): 1001
Name: Abhay Pratap Singh
Age: 20
Gender (M/F): M
Phone Number: 6284881149
Email: co23306@ccet.ac.in
Qualification: Engineer
```

- Linear Probing in case of collision :-

```
Index 30: 1030 Hardeep Saini 36 Male 9914611106 hsaini@ccet.ac.in M.Tech 0
Index 31: 1031 Anil Kumar Vaghmare 44 Male 6284561607 anilvaghmare@ccet.ac.in Ph.D 0
Index 32: 1032 Jatinder Madan 52 Male 9041291970 jatindermadan@ccet.ac.in Ph.D 0
Index 33: 1033 Vettivel S C 42 Male 9865822376 scvettivel@ccet.ac.in Ph.D 0
Index 34: 1034 Radhey Sham 46 Male 9888040982 radheysham@ccet.ac.in Ph.D 0
Index 35: 1035 Mukesh Kumar 40 Male 9478420561 mukeshkumar@ccet.ac.in Ph.D 0
Index 36: 1036 Vinod Chauhan 40 Male 9466736896 vinodchauhan@ccet.ac.in M.E. 0
Index 37: 1037 Ashwani Kumar 40 Male 9872823250 ashwanikumar@ccet.ac.in Ph.D 0
Index 38: 1038 Rajiv Kumar 37 Male 9877887402 rajivkumar@ccet.ac.in M.Tech 0
Index 39: 1039 Nipun Sharma 39 Male 9877726260 nipun@ccet.ac.in M.Tech 0
Index 40: 5001 abhay 20 M 9815774411 abhaysps2004@gmail.com phd
Index 41: empty
Index 42: empty
Index 43: empty
Index 44: empty
Index 45: empty
```

❖ Displaying Data :-

```
Menu:
1. Insert Employee Manually
2. Load Employees from File
3. Search Employee
4. Update Employee
5. Delete Employee
6. Display Hash Table
7. Exit
Enter your choice: 6

Hash Table Contents:
Index 0: 1000 Varun Gupta 45 Male 9646047091 varungupta@ccet.ac.in Ph.D 0
Index 1: 1001 Manveen Kaur 35 Female 9988957007 manveenkaur@ccet.ac.in Ph.D 0
Index 2: 1002 Parul Aggarwal 43 Female 8437911722 parul@ccet.ac.in Ph.D 0
Index 3: 1003 Neha 35 Female 9646614209 neha@ccet.ac.in M.Sc. 0
Index 4: 1004 Rajesh Kumar 51 Male 9478548248 rajeshaastha@ccet.ac.in Ph.D 0
Index 5: 1005 Aradhana Mehta 57 Female 8054977561 amehta@ccet.ac.in Ph.D 0
Index 6: 1006 Mohammad Sakib Perwez Khan 37 Male 7839452836 sakib786@ccet.ac.in M.Tech 0
Index 7: 1007 Poonam 40 Female 8968399719 poonam@ccet.ac.in M.Tech 0
Index 8: 1008 Anil Kumar 42 Male 9816290720 anilkumar@ccet.ac.in M.Tech 0
Index 9: 1009 Karuna Sharma 41 Female 8283833589 karunasharma@ccet.ac.in M.E. 0
Index 10: 1010 Arfat Ahmed 39 Male 8860736206 arfat@ccet.ac.in M.Tech 0
Index 11: 1011 Sunil Kumar Singh 42 Male 9818182457 sksingh@ccet.ac.in Ph.D 0
Index 12: 1012 Ram Bahadur Patel 56 Male 9416932840 rbpatel@ccet.ac.in Ph.D 0
Index 13: 1013 Dheerendra Singh 45 Male 9876439071 dsingh@ccet.ac.in Ph.D 0
Index 14: 1014 Gulshan Goyal 41 Male 9417506206 gulshangoyal@ccet.ac.in Ph.D 0
Index 15: 1015 Sunita 42 Female 9041059379 sunita@ccet.ac.in Ph.D 0
Index 16: 1016 Amit Chhabra 41 Male 9888623825 amitchhabra@ccet.ac.in Ph.D 0
Index 17: 1017 Ankit Gupta 41 Male 9412314479 ankit@ccet.ac.in Ph.D 0
Index 18: 1018 Sarabjeet Singh 44 Male 9463739413 ssingh@ccet.ac.in Ph.D 0
Index 19: 1019 Sudhakar Kumar 34 Male 8434518635 sudhakar@ccet.ac.in M.Tech 0
Index 20: 1020 Animesh Singh 38 Male 9584035345 animeshsingh@ccet.ac.in M.Tech 0
Index 21: 1021 Davinder Singh Saini 46 Male 8146730369 dssaini@ccet.ac.in Ph.D 0
Index 22: 1022 Krishan Gopal Sharma 44 Male 9414403565 kgsharma@ccet.ac.in Ph.D 0
Index 23: 1023 Bhasker Gupta 42 Male 9855908643 bgupta@ccet.ac.in Ph.D 0
Index 24: 1024 Anil Kumar 43 Male 9416234853 anilrose@ccet.ac.in Ph.D 0
Index 25: 1025 Dr Parvinder Kaur 40 Female 8295688911 pkaur@ccet.ac.in Ph.D 0
Index 26: 1026 Shilpa Jindal 41 Female 9463328881 shilpajindal@ccet.ac.in Ph.D 0
Index 27: 1027 Dinesh Sharma 39 Male 9671721850 dsharma@ccet.ac.in Ph.D 0
Index 28: 1028 Irfan Ahmad Khan 43 Male 7835847022 iakhan@ccet.ac.in Ph.D 0
Index 29: 1029 Sarita Sharma 45 Female 9988292611 saritasharma@ccet.ac.in Ph.D 0
Index 30: 1030 Hardeep Saini 36 Male 9914611106 hsaini@ccet.ac.in M.Tech 0
Index 31: 1031 Anil Kumar Vaghmare 44 Male 6284561607 anilvaghmare@ccet.ac.in Ph.D 0
Index 32: 1032 Jatinder Madan 52 Male 9041291970 jatindermadan@ccet.ac.in Ph.D 0
```

❖ Searching an employee :-

```
Menu:
1. Insert Employee Manually
2. Load Employees from File
3. Search Employee
4. Update Employee
5. Delete Employee
6. Display Hash Table
7. Exit
Enter your choice: 3
Enter the employee key to search: 1004

Found employee with key 1004: Rajesh Kumar 51 Male 9478548248 rajeshaastha@ccet.ac.in Ph.D 0
```

❖ Updating an employee :-

```
Menu:
1. Insert Employee Manually
2. Load Employees from File
3. Search Employee
4. Update Employee
5. Delete Employee
6. Display Hash Table
7. Exit
Enter your choice: 4
Enter the employee key to update: 1002
Enter employee details:
Employee Key (4-digit number): 1002
Name: Bhavyam
Age: 19
Gender (M/F): M
Phone Number: 9815774411
Email: co23316@ccet.ac.in
Qualification: engineer
Employee with key 1002 updated.
```

❖ Deleting an employee :-

```
Menu:
1. Insert Employee Manually
2. Load Employees from File
3. Search Employee
4. Update Employee
5. Delete Employee
6. Display Hash Table
7. Exit
Enter your choice: 6

Hash Table Contents:
Index 0: 1000 Varun Gupta 45 Male 9646047091 varungupta@ccet.ac.in Ph.D 0
Index 1: 1001 Manveen Kaur 35 Female 9988957007 manveenkaur@ccet.ac.in Ph.D 0
Index 2: 1002 Bhavyam 19 M 9815774411 co23316@ccet.ac.in engineer
Index 3: empty
```

Topic _____ Date _____

o→ lessons learned from Problem :-

1) Hash function Design :- The choice of an appropriate hash function is crucial. In this case, using $H(k) = k \bmod M$ effectively maps employee keys to memory locations, but it may still result in collisions, so handling those collisions efficiently is critical.

2) Collision Handling :- Collisions are inevitable in hashing, especially with a limited number of slots (M). Linear probing is a simple & effective method for collision resolution, but it can lead to clustering, which may degrade performance if not managed well.

3) Memory Management :- Managing memory carefully is critical to avoid issues like memory leaks or inefficient memory usage.

4) Efficiency Consideration :- The time complexity of insertions and lookups operations can degrade due to collisions.

5) Scalability :- As number of employee scale the hash table need to scale effectively.