Date : 07.04.2025
Lakshana Baskaran
24011103026

LAB - 11
Hash ADT (Hash Table)

QUESTION 1:

A. Write a separate C++ menu-driven program to implement Hash ADT with Linear Probing. Maintain proper boundary conditions and follow good coding practices. The Hash ADT has the following operations,

1. Insert
2. Delete
3. Search
4. Display
5. Exit

SOURCE CODE:

```
//hash tables using linear probing
#include <stdio.h>
#include <iostream>
#include <stdbool.h>
#define SIZE 10
using namespace std;


class hash_table {
    private:
        int m = SIZE;
        int table[SIZE] = {};

    public:
        int hash_function(int);
        int linear_probing(int, int);

        void insert(int);
        int deletion(int);
```

```
        bool search(int);
        bool is_full();
        void print();
};

int main() {
    hash_table tab;
    int key, choice = 0;

    printf("MENU\n1 - Insert\n2 - Delete\n3 - Search\n4 - Exit\n");
    printf("Zero is not allowed in the hash table unless it is
representing empty space\n");
    while (true) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter key to insert: ");
                scanf("%d", &key);
                tab.insert(key);
                break;

            case 2:
                printf("Enter key to delete: ");
                scanf("%d", &key);

                printf("%d\n", tab.deletion(key));
                break;

            case 3:
                printf("Enter key to search: ");
                scanf("%d", &key);
                if (tab.search(key))
                    printf("key present in hash table\n");
                else
                    printf("key not present in hash table\n");
                break;

            case 4:
                printf("Exiting...\n");
                return 0;
                break;
```

```
            default:
                printf("\nInvalid choice. Enter again.\n");
                break;
        }
        printf("\n\tThe hash table: ");
        tab.print();
    }
}

//hash function definition
int hash_table::hash_function(int key) {
    return key % m;
}

//linear probing if collisions occur
int hash_table::linear_probing(int key, int i) {
    if (hash_function(key) + i > m-1) {
        return (hash_function(key) + i) % m;
    }
    return hash_function(key) + i;
}

//insertion of keys into hash table
void hash_table::insert(int key) {
    if (is_full()) {
        printf("OverFlowError: The hash table is Full\n");
        return;
    }
    int i = hash_function(key);
    int j, collision = 0;
    if (table[i] != 0) {
        while (true) {
            collision = collision + 1;
            j = linear_probing(key, collision);
            if (table[j] == 0) {
                table[j] = key;
                break;
            }
        }
    }
    else {
        table[i] = key;
    }
```

```
}

//deletes a specified key from the hash table
int hash_table::deletion(int key) {
    int del;
    for (int i = 0; i < m; ++i) {
        if (table[i] == key) {
            del = table[i];
            table[i] = 0;
            return del;
        }
    }
    printf("key not present in table\n");
    return 0;
}

//checks if a key is present in the table or not
bool hash_table::search(int key) {
    for (int i = 0; i < m; ++i) {
        if (table[i] == key) {
            return true;
        }
    }
    return false;
}

//checks if the hash table is full or not
bool hash_table::is_full() {
    for (int i = 0; i < m; ++i) {
        if (table[i] == 0) {
            return false;
        }
    }
    return true;
}

//displays the hash table
void hash_table::print() {
    for (int i = 0; i < m; ++i) {
        printf("%d ", table[i]);
    }
    printf("\n");
}
```

Date : 07.04.2025
Lakshana Baskaran
24011103026

OUTPUT:

```
lemon@jupiter:~/workspace/college/DSA/Lab-11$ g++ -o out linear_probing.cpp
lemon@jupiter:~/workspace/college/DSA/Lab-11$ ./out
MENU
1 - Insert
2 - Delete
3 - Search
4 - Exit
Zero is not allowed in the hash table unless it is representing empty space

Enter your choice: 1
Enter key to insert: 12

        The hash table: 0 0 12 0 0 0 0 0 0 0

Enter your choice: 1
Enter key to insert: 32

        The hash table: 0 0 12 32 0 0 0 0 0 0

Enter your choice: 1
Enter key to insert: 3

        The hash table: 0 0 12 32 3 0 0 0 0 0

Enter your choice: 145

Invalid choice. Enter again.

        The hash table: 0 0 12 32 3 0 0 0 0 0

Enter your choice: 1
Enter key to insert: 45

        The hash table: 0 0 12 32 3 45 0 0 0 0

Enter your choice: 2
Enter key to delete: 12
12

        The hash table: 0 0 0 32 3 45 0 0 0 0

Enter your choice: 3
Enter key to search: 3
key present in hash table

        The hash table: 0 0 0 32 3 45 0 0 0 0

Enter your choice: 4
Exiting...
lemon@jupiter:~/workspace/college/DSA/Lab-11$
```

Date : 07.04.2025
Lakshana Baskaran
24011103026

QUESTION 2:

B. Write a separate C++ menu-driven program to implement Hash ADT with Quadratic Probing. Maintain proper boundary conditions and follow good coding practices. The Hash ADT has the following operations,

1. Insert
2. Delete
3. Search
4. Display
5. Exit

SOURCE CODE:

```
//hash tables using quadratic probing
#include <stdio.h>
#include <iostream>
#include <stdbool.h>
#define SIZE 10
using namespace std;


class hash_table {
    private:
        int m = SIZE;
        int table[SIZE] = {};

    public:
        int hash_function(int);
        int quadratic_probing(int, int);

        void insert(int);
        int deletion(int);
        bool search(int);
        bool is_full();
        void print();
};


int main() {
    hash_table tab;
```

Date : 07.04.2025
Lakshana Baskaran
24011103026

```
    int key, choice = 0;

    printf("MENU\n1 - Insert\n2 - Delete\n3 - Search\n4 - Exit\n");
    printf("Zero is not allowed in the hash table unless it is
representing empty space\n");
    while (true) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter key to insert: ");
                scanf("%d", &key);
                tab.insert(key);
                break;

            case 2:
                printf("Enter key to delete: ");
                scanf("%d", &key);

                printf("%d\n", tab.deletion(key));
                break;

            case 3:
                printf("Enter key to search: ");
                scanf("%d", &key);
                if (tab.search(key))
                    printf("key present in hash table\n");
                else
                    printf("key not present in hash table\n");
                break;

            case 4:
                printf("Exiting...\n");
                return 0;
                break;

            default:
                printf("\nInvalid choice. Enter again.\n");
                break;
        }
        printf("\n\tThe hash table: ");
        tab.print();
    }
```

```
}

//hash function definition
int hash_table::hash_function(int key) {
    return key % m;
}

//quadratic probing if collisions occur
int hash_table::quadratic_probing(int key, int i) {
    if (hash_function(key) + i*i > m-1) {
        return (hash_function(key) + i*i + i) % m;
    }
    return hash_function(key) + i*i;
}

//insertion of keys into hash table
void hash_table::insert(int key) {
    if (is_full()) {
        printf("OverFlowError: The hash table is Full\n");
        return;
    }
    int i = hash_function(key);
    int j, collision = 0;
    if (table[i] != 0) {
        while (true) {
            collision = collision + 1;
            j = quadratic_probing(key, collision);
            if (table[j] == 0) {
                table[j] = key;
                break;
            }
        }
    }
    else {
        table[i] = key;
    }
}

//deletes a specified key from the hash table
int hash_table::deletion(int key) {
    int del;
    for (int i = 0; i < m; ++i) {
        if (table[i] == key) {
```

```
            del = table[i];
            table[i] = 0;
            return del;
        }
    }

    printf("key not present in table\n");
    return 0;
}

//checks if a key is present in the table or not
bool hash_table::search(int key) {
    for (int i = 0; i < m; ++i) {
        if (table[i] == key) {
            return true;
        }
    }
    return false;
}

//checks if the hash table is full or not
bool hash_table::is_full() {
    for (int i = 0; i < m; ++i) {
        if (table[i] == 0) {
            return false;
        }
    }
    return true;
}

//displays the hash table
void hash_table::print() {
    for (int i = 0; i < m; ++i) {
        printf("%d ", table[i]);
    }
    printf("\n");
}
```

Date : 07.04.2025
Lakshana Baskaran
24011103026

OUTPUT:

```
lemon@jupiter:~/workspace/college/DSA/Lab-11$ g++ -o out quadratic_probing.cpp
lemon@jupiter:~/workspace/college/DSA/Lab-11$ ./out
MENU
1 - Insert
2 - Delete
3 - Search
4 - Exit
Zero is not allowed in the hash table unless it is representing empty space

Enter your choice: 1
Enter key to insert: 2

        The hash table: 0 0 2 0 0 0 0 0 0 0

Enter your choice: 1
Enter key to insert: 12

        The hash table: 0 0 2 12 0 0 0 0 0 0

Enter your choice: 1
Enter key to insert: 32

        The hash table: 0 0 2 12 0 0 32 0 0 0

Enter your choice: 1
Enter key to insert: 3

        The hash table: 0 0 2 12 3 0 32 0 0 0

Enter your choice: 2
Enter key to delete: 2
2

        The hash table: 0 0 0 12 3 0 32 0 0 0

Enter your choice: 3
Enter key to search: 32
key present in hash table

        The hash table: 0 0 0 12 3 0 32 0 0 0

Enter your choice: 4
Exiting...
lemon@jupiter:~/workspace/college/DSA/Lab-11$
```

Date : 07.04.2025
Lakshana Baskaran
24011103026

QUESTION 3:

C. Write a separate C++ menu-driven program to implement Hash ADT with Separate Chaining. Maintain proper boundary conditions and follow good coding practices. The Hash ADT has the following operations,

1. Insert
2. Delete
3. Search
4. Exit

SOURCE CODE:

```
//hash tables using seperate chaining
#include<cstdio>
#include<cstdlib>

class Hashtable{
    private:
    struct Node{
        int data;
        Node *next;
    };
     Node* arr[10];

     int hash(int key) {
        return key % 10;
    }

    public:
    Hashtable(){
        for (int i = 0; i < 10; i++) {
            arr[i] = NULL;
        }
    }

    void insert(int num){
        int index = hash(num);
        Node* newNode = (Node*) malloc(sizeof(Node));
        newNode->data = num;
```

```
    newNode->next = arr[index];
    arr[index] = newNode;
}

void remove(int key){
    int index = hash(key);
    Node* temp = arr[index];
    Node* prev = NULL;

    while (temp != NULL) {
        if (temp->data == key) {
            if (prev == NULL) {
                arr[index] = temp->next;
            } else {
                prev->next = temp->next;
            }
            free(temp);
            return;
        }
        prev = temp;
        temp = temp->next;
    }
    printf("Element not found!\n");
}

int search(int target){
    int index = hash(target);
    Node* temp = arr[index];
    int pos = 1;

    while (temp != NULL) {
        if (temp->data == target) {
            return pos;
        }
        temp = temp->next;
        pos++;
    }

    return -1;
}

void display(){
     printf("\nHashtable contents:\n");
```

```
        for (int i = 0; i < 10; i++) {
            printf("%d: ", i);
            Node* temp = arr[i];
            while (temp != NULL) {
                printf("%d -> ", temp->data);
                temp = temp->next;
            }
            printf("NULL\n");
        }
    }
};

int main(){
    Hashtable h;
    int choice;
    int value;
    int key;
    int result;
    printf("MENU\n1 - Insert\n2 - Delete\n3 - Search\n4 - Display\n5
- Exit\n");
    printf("-1 is not allowed in the hash table unless it is
representing empty space\n");
    while(true){
        printf("\nEnter your choice: ");
        scanf("%d" , &choice);

        switch(choice){
            case 1:
                printf("Enter the value to be inserted into the
table: ");
                scanf("%d", &value);
                h.insert(value);
                break;

            case 2:
                printf("Enter the value to be deleted: ");
                scanf("%d", &key);
                h.remove(key);
                break;

            case 3:
                printf("Enter the element to serach for: ");
                scanf("%d", &key);
```

Date : 07.04.2025
Lakshana Baskaran
24011103026

```
                if (h.search(key) == -1)
                    printf("the element %d is NOT present in the
table\n", key);
                else
                    printf("the element %d is present in the
table\n", key);
                break;

            case 4:
                h.display();
                break;

            case 5:
                printf("Exiting...\n");
                return 0;

            default:
                printf("Invalid Choice.\n");
        }

    }
}
```

Date : 07.04.2025
Lakshana Baskaran
24011103026

OUTPUT:

```
lemon@jupiter:~/workspace/college/DSA/Lab-11$ g++ -o out seperate_chaining.cpp
lemon@jupiter:~/workspace/college/DSA/Lab-11$ ./out
MENU
1 - Insert
2 - Delete
3 - Search
4 - Display
5 - Exit
-1 is not allowed in the hash table unless it is representing empty space

Enter your choice: 1
Enter the value to be inserted into the table: 12

Enter your choice: 1
Enter the value to be inserted into the table: 34

Enter your choice: 1
Enter the value to be inserted into the table: 4

Enter your choice: 1
Enter the value to be inserted into the table: 45

Enter your choice: 1
Enter the value to be inserted into the table: 22

Enter your choice: 4

Hashtable contents:
0: NULL
1: NULL
2: 22 -> 12 -> NULL
3: NULL
4: 4 -> 34 -> NULL
5: 45 -> NULL
6: NULL
7: NULL
8: NULL
9: NULL

Enter your choice: 2
Enter the value to be deleted: 34

Enter your choice: 3
Enter the element to serach for: 34
the element 34 is NOT present in the table

Enter your choice: 5
Exiting...
lemon@jupiter:~/workspace/college/DSA/Lab-11$
```

—