Exercise 12: Implementation of Hash Table using Closed and Open addressing methods

The HashTableADT contains hash table and its size. Hash function to be used for the insertion of elements is ***x mod tableSize***. Use Separate chaining method to resolve the collision.

- void init(HashTableADT *H) – To initialize the size of Hash Table
- void insertElementL (HashTableADT *H, int x)– To insert the input key into the hash table
- int searchElement(HashTableADT *H, int key) – Searching an element in the hash table, if found return 1, otherwise return -1
- void displayHT(HashTableADT *H) – Display the elements in the hash table

Note:

1. Implement HashTableADT with the specified operations in HashTableADTImpl.h
2. Write a menu driven application to utilize the HashTableADT.

1. Demonstrate ADT with the following test case
insert 23, 45, 69, 87, 48, 67, 54, 66, 53

Contents of Hash Table
(Separate Chaining)
0 :
1 :
2 :
3 : 23 → 53
4 : 54
5 : 45
6 : 66
7 : 87 → 67
8 : 48
9 : 69

2. Create another hash table ADT with following functions for open addressing methods, namely, Quadratic probing and Double Hashing.

- void insertElementL (HashTableADT *H, int x)– To insert the input key into the hash table
- void displayHT(HashTableADT *H) – Display the elements in the hash table

Note: For Double hashing, the second hash function is 7-(x%7)

Demonstrate the ADT with the following testcase
(i) Contents of Hash Table
(Quadratic Probing)
0 →
1 →67
2 →53
3 →23
4 →54
5 →45
6 →66
7 →87
8 →48
9 →69

(ii) Contents of Hash Table
(Double Hashing)
0 →67
1 →
2 →53
3 →23
4 →54
5 →45
6 →66
7 →87
8 →48
9 →69

CODE:

```
#include <stdio.h>
#include <stdlib.h>

// Node structure for linked list
struct node
{
   int data;
   struct node *next;
};

// Hash Table structure
struct hash_table
{
   int size;
```

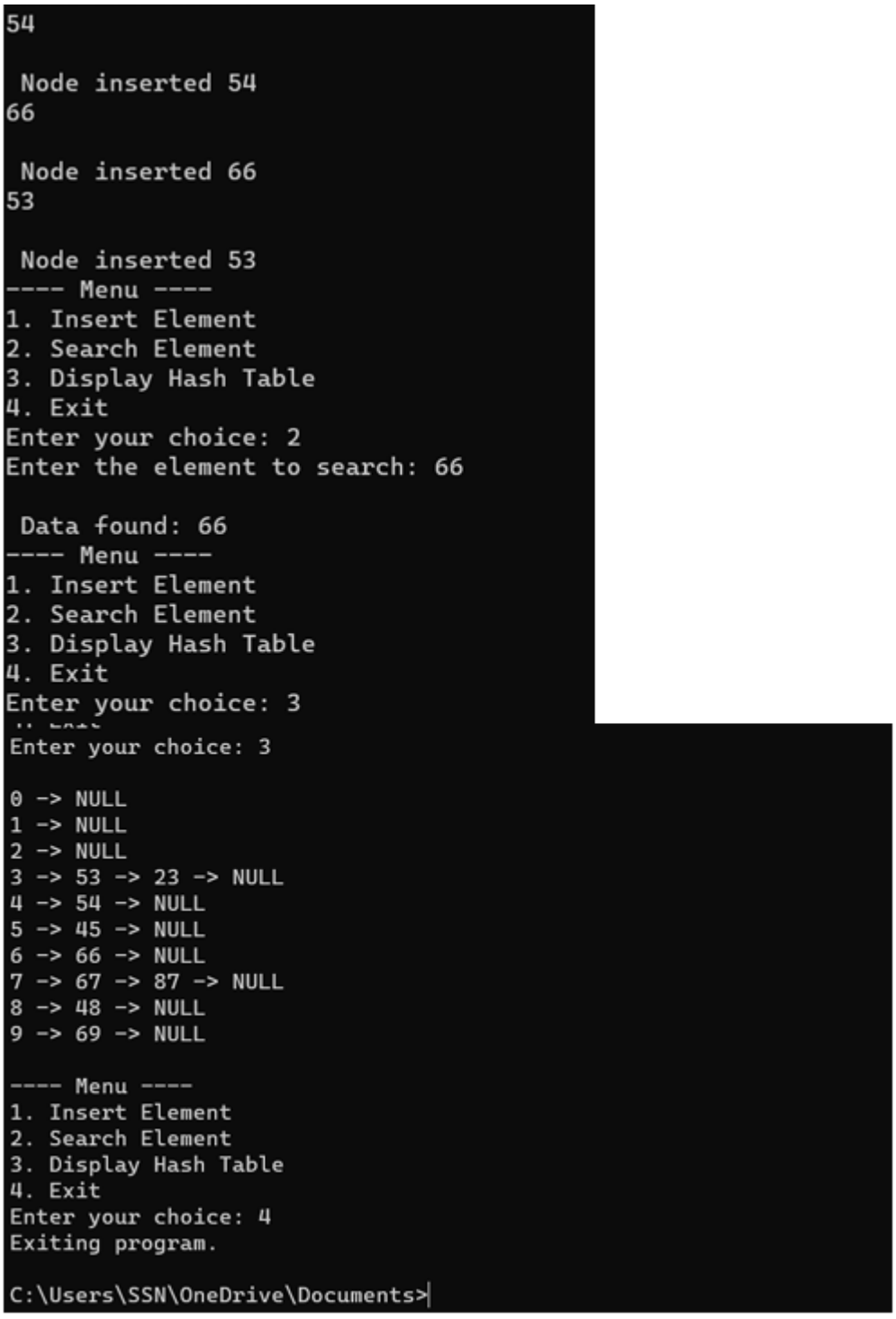
```
struct node *list[100];
};
// Function to initialize the size of the hash table
void init(struct hash_table *H, int size)
{
H->size = size;
for (int i = 0; i < H->size; i++)
{
H->list[i] = (struct node *)malloc(sizeof(struct node));
H->list[i]->next = NULL;
}
}
// Function to insert an element into the hash table
void insertElementL(struct hash_table *H, int x)
{
int h;
struct node *temp;
h = x % H->size;
temp = (struct node *)malloc(sizeof(struct node));
temp->data = x;
temp->next = H->list[h]->next;
H->list[h]->next = temp;
printf("\n Node inserted %d ", temp->data);
}
// Function to search for an element in the hash table
int searchElement(struct hash_table *H, int key)
{
struct node *temp;
int h;
h = key % H->size;
temp = H->list[h]->next;
while (temp != NULL)
{
if (temp->data == key)
{
printf("\n Data found: %d", temp->data);
return 1;
}
temp = temp->next;
}
return -1;
}
// Function to display the elements in the hash table
void displayHT(struct hash_table *H)
```

```
{
for (int i = 0; i < H->size; i++)
{
struct node *temp = H->list[i]->next;
printf("\n%d -> ", i);
while (temp != NULL)
{
printf("%d -> ", temp->data);
temp = temp->next;
}
printf("NULL");
}
printf("\n");
}
// Menu-driven application to demonstrate HashTableADT
int main()
{
struct hash_table *hashTable = (struct hash_table *)malloc(sizeof(struct hash_table));
init(hashTable, 10);
int choice, key,k;
do
{
printf("\n---- Menu ----");
printf("\n1. Insert Element");
printf("\n2. Search Element");
printf("\n3. Display Hash Table");
printf("\n4. Exit");
printf("\nEnter your choice: ");
scanf("%d", &choice);
switch (choice)
{
case 1:
printf("Enter the number of element to insert: ");
scanf("%d", &key);
for(int i=0;i<key;i++)
{
scanf("%d",&k);
insertElementL(hashTable, k);}
break;
case 2:
printf("Enter the element to search: ");
scanf("%d", &key);
if (searchElement(hashTable, key) == -1)
{
```

```
printf("\n Data not found");
}
break;
case 3:
displayHT(hashTable);
break;
case 4:
printf("Exiting program.\n");
break;
default:
printf("Invalid choice. Please enter a valid option.\n");
break;
}
} while (choice != 4);
free(hashTable);
return 0;
}
```

OUTPUT:

```
C:\Users\SSN\OneDrive\Documents>g++ hasecode.c

C:\Users\SSN\OneDrive\Documents>a

---- Menu ----
1. Insert Element
2. Search Element
3. Display Hash Table
4. Exit
Enter your choice: 1
Enter the number of element to insert: 9
23

 Node inserted 23
45

 Node inserted 45
69

 Node inserted 69
87

 Node inserted 87
48

 Node inserted 48
67

 Node inserted 67
```



2) CODE:

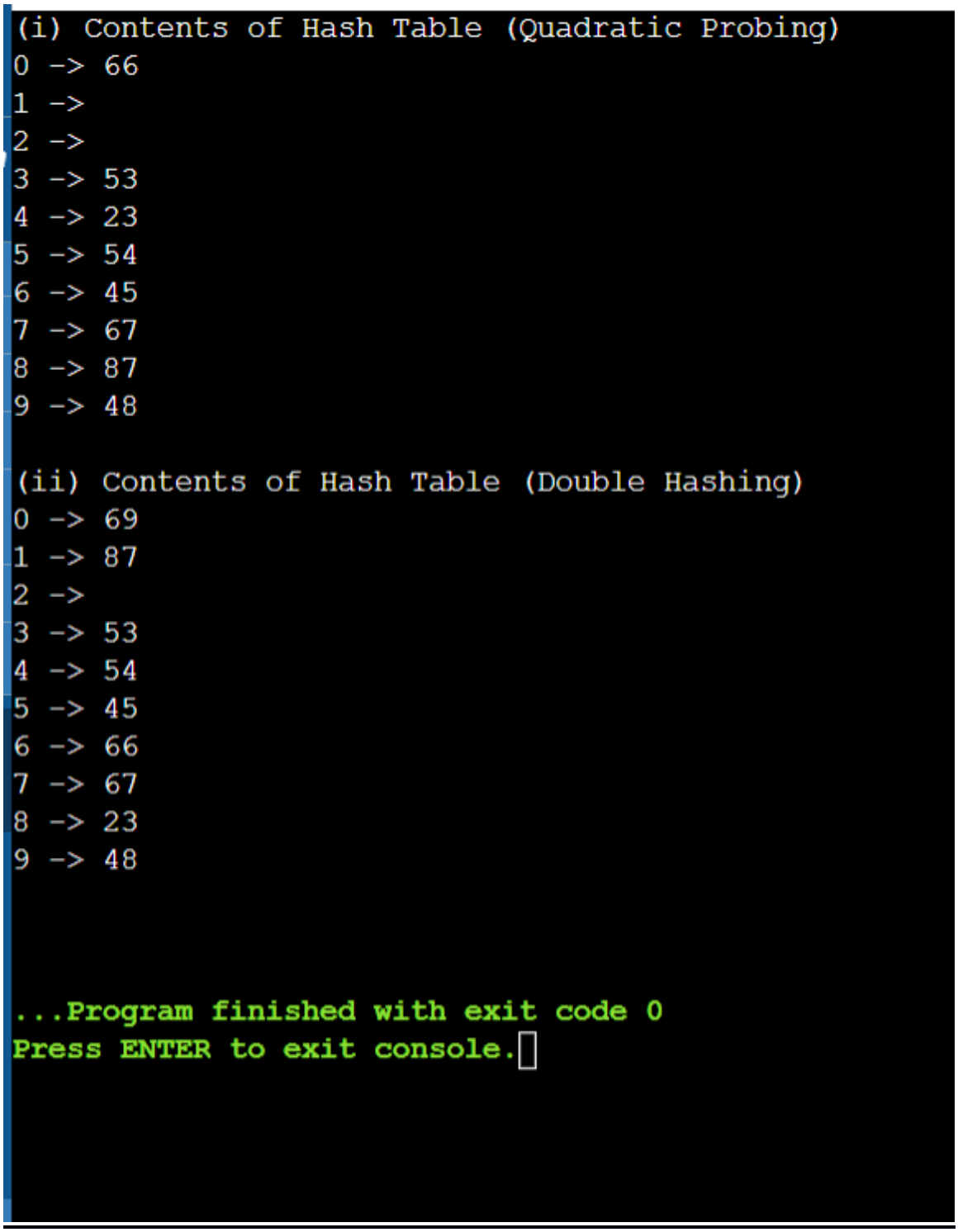
```
#include <stdio.h>
#include <stdlib.h>
```

```
// Node structure for open addressing
struct node {
   int data;
};

// Hash Table structure
struct hash_table {
   int size;
   struct node *table[100];
};

// Initialize hash table
void init(struct hash_table *H, int size) {
   H->size = size;
   for (int i = 0; i < H->size; i++) {
      H->table[i] = NULL;
   }
}

// Quadratic Probing insertion
void insertElementQuad(struct hash_table *H, int x) {
   int h = x % H->size;
   int i = 0;

   while (H->table[(h + i * i) % H->size] != NULL) {
      i++;
   }

   H->table[(h + i * i) % H->size] =
      (struct node *)malloc(sizeof(struct node));
   H->table[(h + i * i) % H->size]->data = x;
}

// Double Hashing insertion
void insertElementDouble(struct hash_table *H, int x) {
   int h = x % H->size;
   int i = 0;
   int secondHash = 7 - (x % 7);

   while (H->table[(h + i * secondHash) % H->size] != NULL) {
      i++;
   }
```

```
    H->table[(h + i * secondHash) % H->size] =
        (struct node *)malloc(sizeof(struct node));
    H->table[(h + i * secondHash) % H->size]->data = x;
}

// Display hash table
void displayHT(struct hash_table *H) {
    for (int i = 0; i < H->size; i++) {
        printf("%d -> ", i);
        if (H->table[i] != NULL) {
            printf("%d", H->table[i]->data);
        }
        printf("\n");
    }
    printf("\n");
}

int main() {
    struct hash_table *hashTableQuad =
        (struct hash_table *)malloc(sizeof(struct hash_table));
    struct hash_table *hashTableDouble =
        (struct hash_table *)malloc(sizeof(struct hash_table));

    init(hashTableQuad, 10);
    init(hashTableDouble, 10);

    // Quadratic Probing elements
    int quadElements[] = {67, 53, 23, 54, 45, 66, 87, 48, 69};
    int n1 = sizeof(quadElements) / sizeof(quadElements[0]);

    for (int i = 0; i < n1; i++) {
        insertElementQuad(hashTableQuad, quadElements[i]);
    }

    printf("(i) Contents of Hash Table (Quadratic Probing)\n");
    displayHT(hashTableQuad);

    // Double Hashing elements
    int doubleElements[] = {67, 53, 23, 54, 45, 66, 87, 48, 69};
    int n2 = sizeof(doubleElements) / sizeof(doubleElements[0]);

    for (int i = 0; i < n2; i++) {
        insertElementDouble(hashTableDouble, doubleElements[i]);
    }
```

```
    printf("(ii) Contents of Hash Table (Double Hashing)\n");
    displayHT(hashTableDouble);

    return 0;
}
```

Output:



Learning Outcome:

This experiment helped me understand hashing and collision resolution using Quadratic Probing and Double Hashing. I learned how open addressing techniques reduce collisions and improve the performance of hash tables.