Experiment No. : 7

Title: Hash Table implementation

Batch: B-1 **Roll No.: 16010422234** **Name: Chandana Ramesh Galgali**

Experiment No.: 7

Aim: Demonstrate the use of hash table in implementation of Dictionary using Circular Array

Resources Used: C/ C++ editor and compiler.

Theory:

Hash Table

Hash table is one of the most important data structures that uses a special function known as a hash function that maps a given value with a key to access the elements faster.

A hash table is a data structure that stores key-value pairs and provides fast access to values based on their keys. It employs a hash function to compute an index where each key-value pair is stored. Collisions, which occur when multiple keys map to the same index, are resolved through chaining, where each index holds a linked list of key-value pairs. Using a circular array for the hash table helps optimize memory utilization.

For example, suppose the key value is John and the value is the phone number, so when we pass the key value in the hash function shown as below:

Hash(key)= index;

When we pass the key in the hash function, then it gives the index.

Hash(john) = 3;

The above example adds the john at the index 3.

Hashing

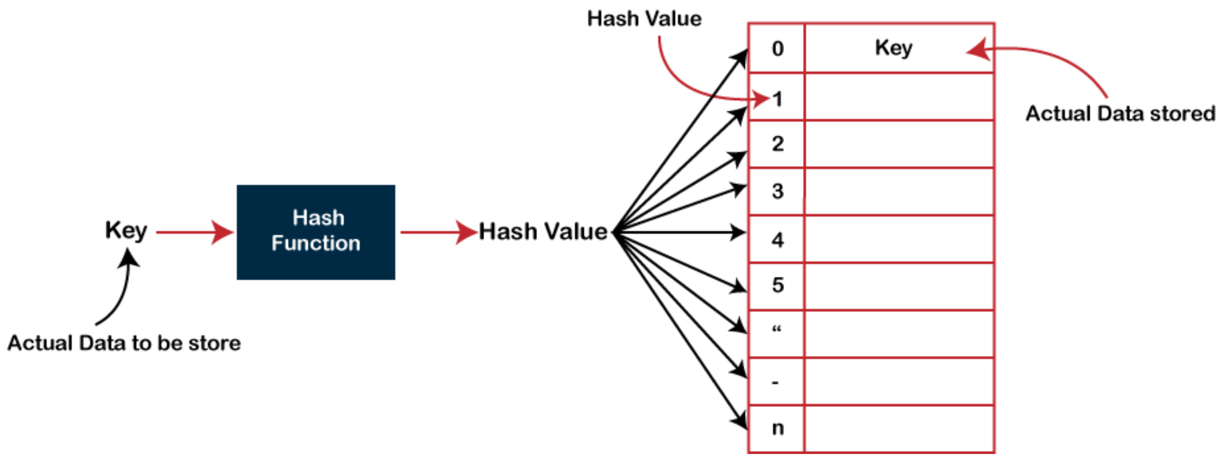
Hashing is one of the searching techniques that uses a constant time. The time complexity in hashing is O(1).

The worst time complexity in linear search is O(n), and O(logn) in binary search. In both the searching techniques, the searching depends upon the number of elements but we want the technique that takes a constant time. So, hashing technique is used which provides a constant time.

In Hashing technique, the hash table and hash function are used. Using the hash function, we can calculate the address at which the value can be stored.

The main idea behind the hashing is to create the (key/value) pairs. If the key is given, then the algorithm computes the index at which the value would be stored. It can be written as:

Index = hash(key)



Drawback of Hash function

A Hash function assigns each value with a unique key. Sometimes hash table uses an imperfect hash function that causes a collision because the hash function generates the same key of two different values.

Collision

When the two different values have the same value, then the problem occurs between the two values, known as a collision. In the above example, the value is stored at index 6. If the key value is 26, then the index would be:

h(26) = 26%10 = 6

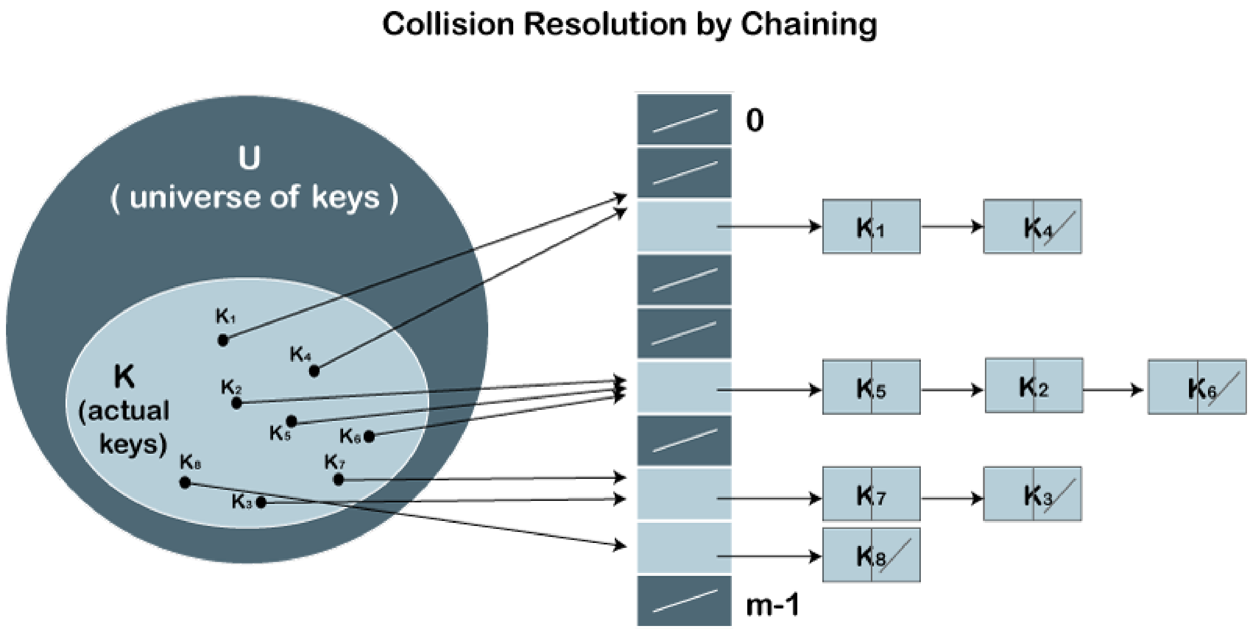
Therefore, two values are stored at the same index, i.e., 6, and this leads to the collision problem. To resolve these collisions, we have some techniques known as collision techniques.

The following are the collision techniques:

- Open Hashing: It is also known as closed addressing.
- Closed Hashing: It is also known as open addressing.

Open Hashing

- In Open Hashing, one of the methods used to resolve the collision is known as a chaining method



Basic Operations

Following are the basic primary operations of a hash table.

- **Search** − Searches an element in a hash table.
- **Insert** − inserts an element in a hash table.
- **Delete** − Deletes an element from a hash table.

DataItem

Define a data item having some data and key, based on which the search is to be conducted in a hash table.

```
struct DataItem {
   int data;
   int key;
};
```

Hash Method

Define a hashing method to compute the hash code of the key of the data item.

```
int hashCode(int key){
   return key % SIZE;
}
```

Search Operation

Whenever an element is to be searched, compute the hash code of the key passed and locate the element using that hash code as index in the array. Use linear probing to get the element ahead if the element is not found at the computed hash code. The search function attempts to find a specific value associated with a given key within the hash table. It calculates the initial index using the hash function, and then employs linear probing to navigate through the circular array. Linear probing involves moving to the next slot if the current slot is occupied until an empty slot is found or the entire array is traversed.

Insert Operation

Whenever an element is to be inserted, compute the hash code of the key passed and locate the index using that hashcode as an index in the array. Use linear probing for empty location, if an element is found at the computed hash code. The insert function enables the addition of new key-value pairs to the hash table. It calculates the initial index using the hash function and then employs linear probing to find an empty slot for insertion. If the array is full and no empty slot is found after a complete cycle, a message is displayed indicating that the hash table is full.

Delete Operation

Whenever an element is to be deleted, compute the hash code of the key passed and locate the index using that hashcode as an index in the array. Use linear probing to get the element ahead if an element is not found at the computed hash code. When found, store a dummy item there to keep the performance of the hash table intact. The delete function allows the removal of a key-value pair based on a given key. Similar to the search function, it calculates the initial index and uses linear probing to locate the key-value pair. Once found, the status of the pair is updated to "deleted."

Activity: Write a C program for implementation of Dictionary using Circular Array in a Hash Table

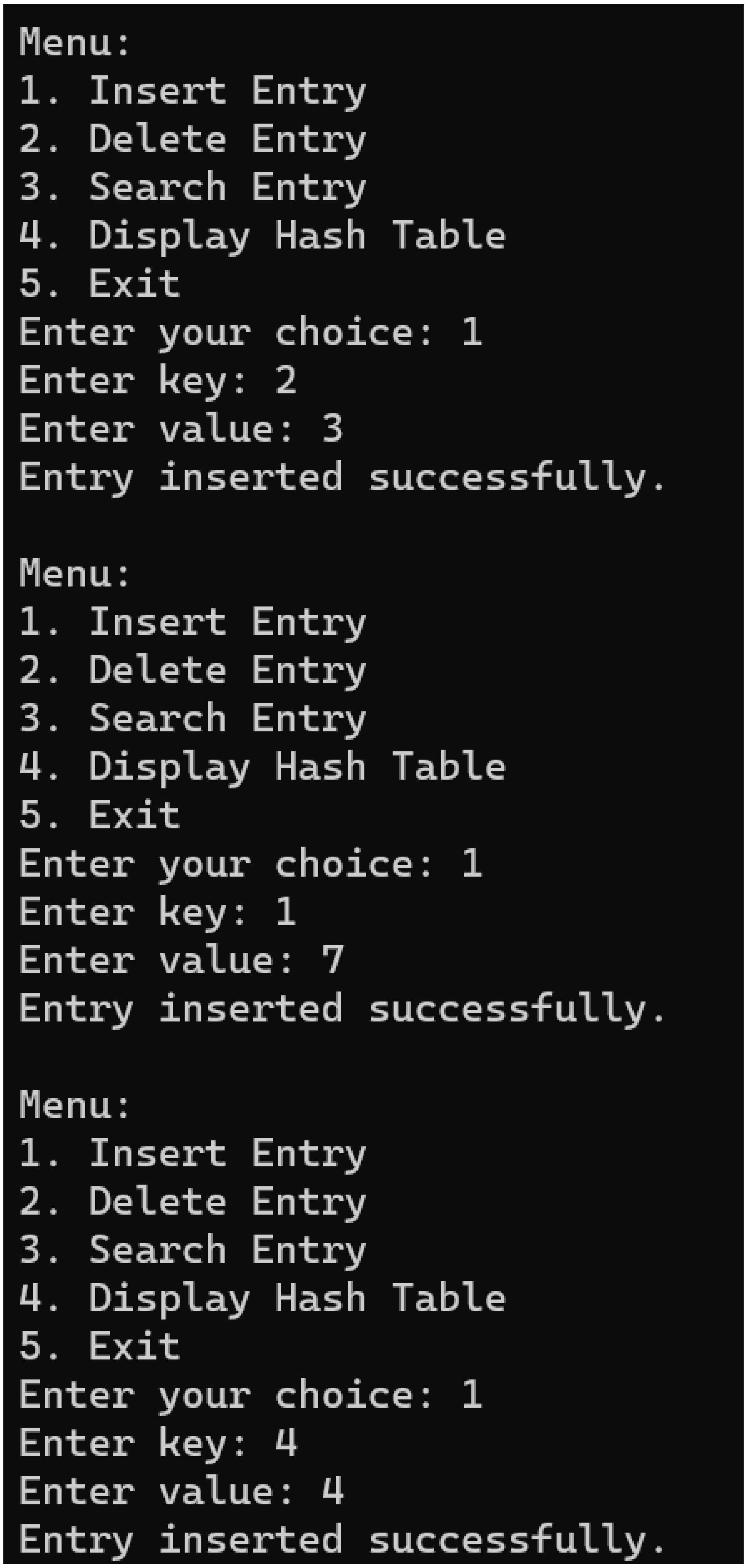
---

Results:

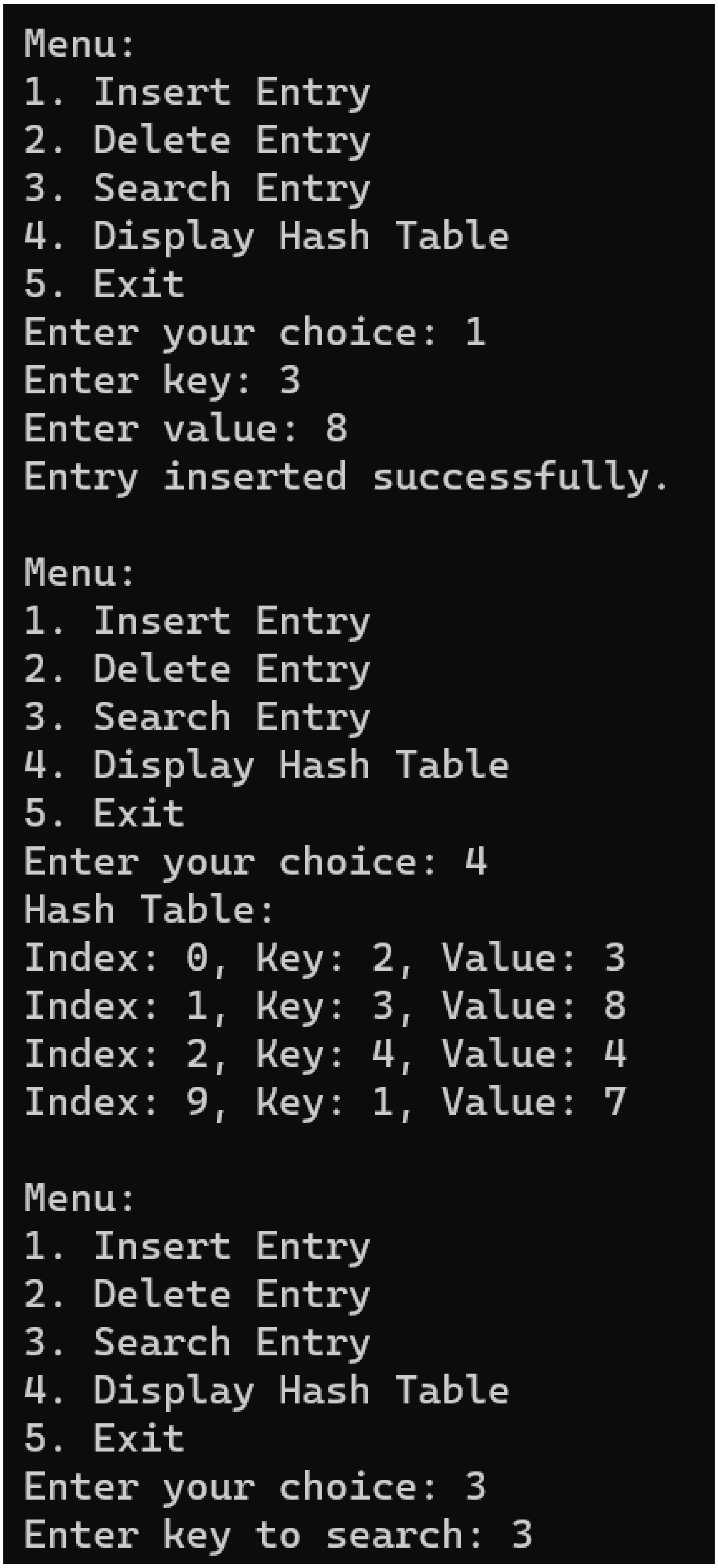
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 10
typedef struct {
   char key[20];
   char value[20];
} Entry;
Entry hashTable[SIZE];
void initializeHashTable() {
   int i;
   for (i = 0; i < SIZE; i++) {
      strcpy(hashTable[i].key, "");
      strcpy(hashTable[i].value, "");
   }
}
```

```
int hashFunction(char* key) {
   int sum = 0;
   int i;
   for (i = 0; i < strlen(key); i++) {
      sum += key[i];
   }
   return sum % SIZE;
}
void insertEntry() {
   char key[20];
   char value[20];
   printf("Enter key: ");
   scanf("%s", key);
   printf("Enter value: ");
   scanf("%s", value);
   int index = hashFunction(key);
   while (strcmp(hashTable[index].key, "") != 0) {
      index = (index + 1) % SIZE;
   }
   strcpy(hashTable[index].key, key);
   strcpy(hashTable[index].value, value);
   printf("Entry inserted successfully.\n");
}
void deleteEntry() {
   char key[20];
   printf("Enter key to delete: ");
   scanf("%s", key);
   int index = hashFunction(key);
   while (strcmp(hashTable[index].key, key) != 0) {
      index = (index + 1) % SIZE;
      if (strcmp(hashTable[index].key, "") == 0) {
         printf("Entry not found.\n");
         return;
      }
   }
   strcpy(hashTable[index].key, "");
   strcpy(hashTable[index].value, "");
   printf("Entry deleted successfully.\n");
}
void searchEntry() {
   char key[20];
   printf("Enter key to search: ");
   scanf("%s", key);
   int index = hashFunction(key);
   while (strcmp(hashTable[index].key, key) != 0) {
      index = (index + 1) % SIZE;
      if (strcmp(hashTable[index].key, "") == 0) {
         printf("Entry not found.\n");
```

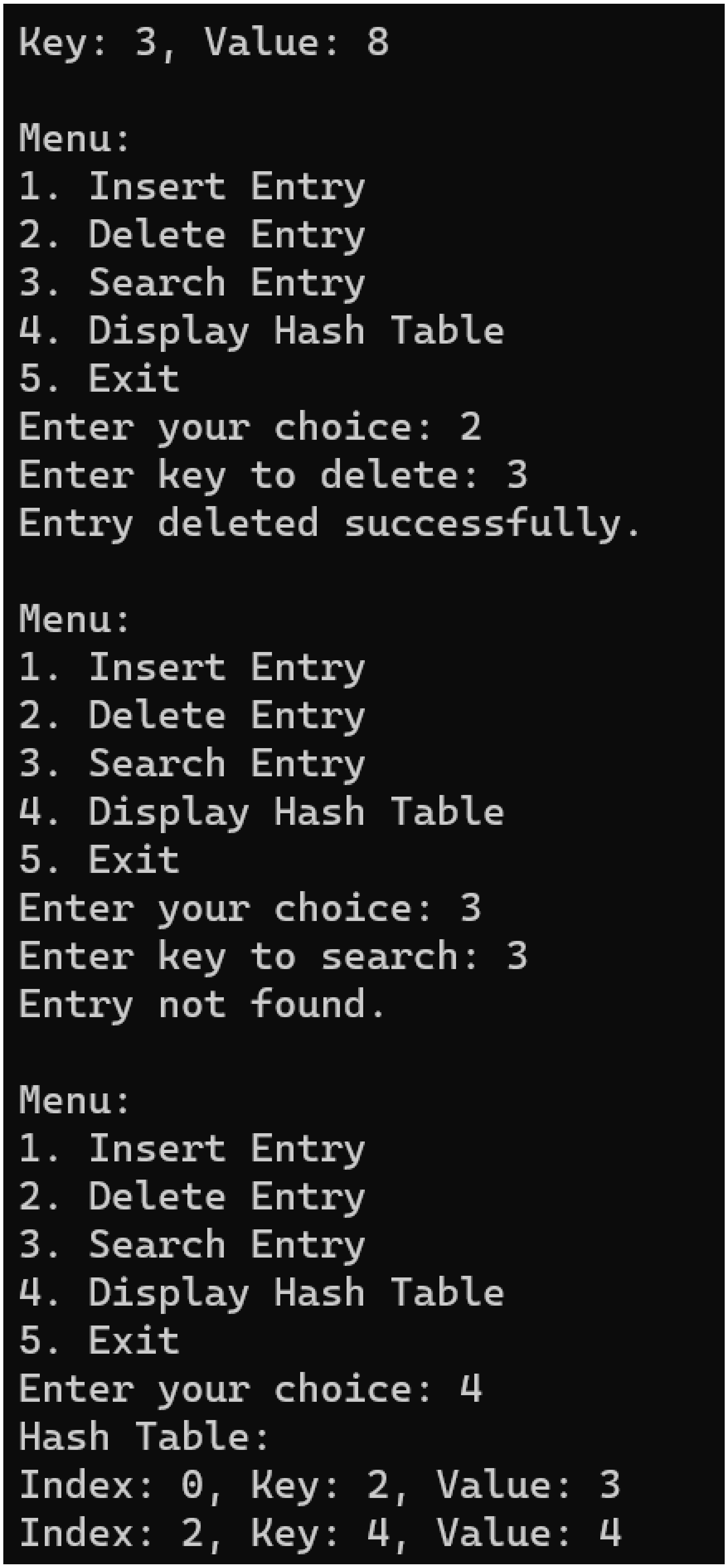
```
            return;
        }
    }
    printf("Key: %s, Value: %s\n", hashTable[index].key, hashTable[index].value);
}
void displayHashTable() {
    int i;
    printf("Hash Table:\n");
    for ( i = 0; i < SIZE; i++) {
        if (strcmp(hashTable[i].key, "") != 0) {
            printf("Index: %d, Key: %s, Value: %s\n", i, hashTable[i].key, hashTable[i].value);
        }
    }
}
int main() {
    initializeHashTable();
    int choice, i;
    do {
        printf("\nMenu:\n");
        printf("1. Insert Entry\n");
        printf("2. Delete Entry\n");
        printf("3. Search Entry\n");
        printf("4. Display Hash Table\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                insertEntry();
                break;
            case 2:
                deleteEntry();
                break;
            case 3:
                searchEntry();
                break;
            case 4:
                displayHashTable();
                break;
            case 5:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);
    return 0;
}
```



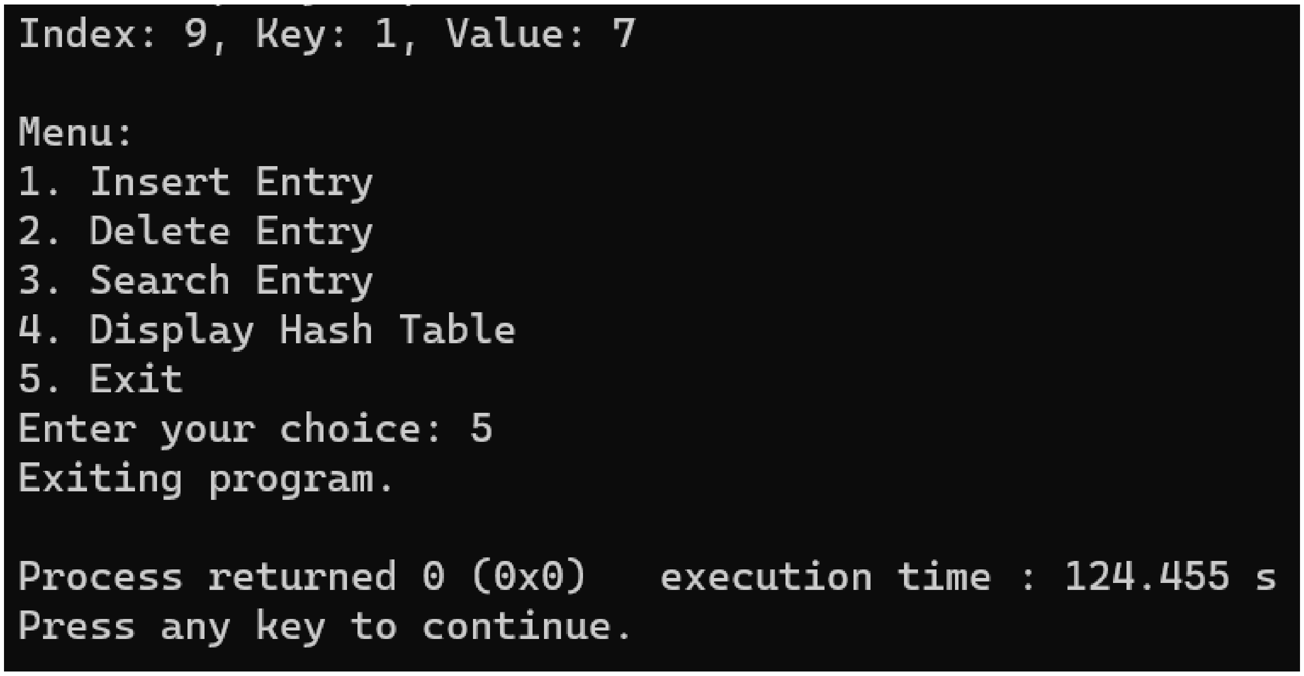
```
Menu:
1. Insert Entry
2. Delete Entry
3. Search Entry
4. Display Hash Table
5. Exit
Enter your choice: 1
Enter key: 2
Enter value: 3
Entry inserted successfully.

Menu:
1. Insert Entry
2. Delete Entry
3. Search Entry
4. Display Hash Table
5. Exit
Enter your choice: 1
Enter key: 1
Enter value: 7
Entry inserted successfully.

Menu:
1. Insert Entry
2. Delete Entry
3. Search Entry
4. Display Hash Table
5. Exit
Enter your choice: 1
Enter key: 4
Enter value: 4
Entry inserted successfully.
```



```
Menu:
1. Insert Entry
2. Delete Entry
3. Search Entry
4. Display Hash Table
5. Exit
Enter your choice: 1
Enter key: 3
Enter value: 8
Entry inserted successfully.

Menu:
1. Insert Entry
2. Delete Entry
3. Search Entry
4. Display Hash Table
5. Exit
Enter your choice: 4
Hash Table:
Index: 0, Key: 2, Value: 3
Index: 1, Key: 3, Value: 8
Index: 2, Key: 4, Value: 4
Index: 9, Key: 1, Value: 7

Menu:
1. Insert Entry
2. Delete Entry
3. Search Entry
4. Display Hash Table
5. Exit
Enter your choice: 3
Enter key to search: 3
```



```
Key: 3, Value: 8

Menu:
1. Insert Entry
2. Delete Entry
3. Search Entry
4. Display Hash Table
5. Exit
Enter your choice: 2
Enter key to delete: 3
Entry deleted successfully.

Menu:
1. Insert Entry
2. Delete Entry
3. Search Entry
4. Display Hash Table
5. Exit
Enter your choice: 3
Enter key to search: 3
Entry not found.

Menu:
1. Insert Entry
2. Delete Entry
3. Search Entry
4. Display Hash Table
5. Exit
Enter your choice: 4
Hash Table:
Index: 0, Key: 2, Value: 3
Index: 2, Key: 4, Value: 4
```



```
Index: 9, Key: 1, Value: 7

Menu:
1. Insert Entry
2. Delete Entry
3. Search Entry
4. Display Hash Table
5. Exit
Enter your choice: 5
Exiting program.

Process returned 0 (0x0)   execution time : 124.455 s
Press any key to continue.
```

Outcomes: Describe concepts of advance data structures like set, map & dictionary.

Conclusion:

This experiment has provided a practical understanding of how hash tables can be utilized to implement a dictionary using a circular array, showcasing the benefits of this data structure in terms of efficient key-value pair storage and retrieval.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

- Michael T. Goodrich, Roberto Tamassia, and David M. Mount. 2009. Data Structures and Algorithms in C++ (2nd. ed.). Wiley Publishing.