C Program to Implement Hash Tables chaining with Singly Linked Lists

This is a C Program to implement hash tables using linked list. Hash table will have ‘n’ number of buckets. To insert a node into the hash table, we need to find the hash index for the given key. And it could be calculated using the hash function. Example: hashIndex = key % noOfBuckets Move to the bucket corresponds to the above calculated hash index and insert the new node at the end of the list. To delete a node from hash table, get the key from the user, calculate the hash index, move to the bucket corresponds to the calculated hash index, search the list in the current bucket to find and remove the node with the given key. Finally, remove the node with given key, if it is present.
Here is source code of the C Program to Implement Hash Tables chaining with Singly Linked Lists. The C program is successfully compiled and run on a Linux system. The program output is also shown below.

```
1. #include <stdio.h>
2. #include <string.h>
3. #include <stdlib.h>
4. struct hash *hashTable = NULL;
5. int eleCount = 0;
6. struct node {
7.     int key, age;
8.     char name[100];
9.     struct node *next;
10. };
11. struct hash {
12.     struct node *head;
13.     int count;
14. };
15. struct node * createNode(int key, char *name, int age) {
16.     struct node *newnode;
17.     newnode = (struct node *) malloc(sizeof(struct node));
18.     newnode->key = key;
19.     newnode->age = age;
20.     strcpy(newnode->name, name);
21.     newnode->next = NULL;
22.     return newnode;
```

```
}
void insertToHash(int key, char *name, int age) {
    int hashIndex = key % eleCount;
    struct node *newnode = createNode(key, name, age);
    /* head of list for the bucket with index "hashIndex" */
    if (!hashTable[hashIndex].head) {
        hashTable[hashIndex].head = newnode;
        hashTable[hashIndex].count = 1;
        return;
    }
    /* adding new node to the list */
    newnode->next = (hashTable[hashIndex].head);
    /*
     * update the head of the list and no of
     * nodes in the current bucket
     */
    hashTable[hashIndex].head = newnode;
    hashTable[hashIndex].count++;
    return;
}
void deleteFromHash(int key) {
    /* find the bucket using hash index */
    int hashIndex = key % eleCount, flag = 0;
    struct node *temp, *myNode;
    /* get the list head from current bucket */
    myNode = hashTable[hashIndex].head;
    if (!myNode) {
```

```
        printf("Given data is not present in hash Table!!\n");
        return;
    }
    temp = myNode;
    while (myNode != NULL) {
        /* delete the node with given key */
        if (myNode->key == key) {
            flag = 1;
            if (myNode == hashTable[hashIndex].head)
                hashTable[hashIndex].head = myNode->next;
            else
                temp->next = myNode->next;
            hashTable[hashIndex].count--;
            free(myNode);
            break;
        }
        temp = myNode;
        myNode = myNode->next;
    }
    if (flag)
        printf("Data deleted successfully from Hash Table\n");
    else
        printf("Given data is not present in hash Table!!!!\n");
    return;
}
void searchInHash(int key) {
    int hashIndex = key % eleCount, flag = 0;
```

```
    struct node *myNode;
    myNode = hashTable[hashIndex].head;
    if (!myNode) {
        printf("Search element unavailable in hash table\n");
        return;
    }
    while (myNode != NULL) {
        if (myNode->key == key) {
            printf("VoterID  : %d\n", myNode->key);
            printf("Name     : %s\n", myNode->name);
            printf("Age      : %d\n", myNode->age);
            flag = 1;
            break;
        }
        myNode = myNode->next;
    }
    if (!flag)
        printf("Search element unavailable in hash table\n");
    return;
}
void display() {
    struct node *myNode;
    int i;
    for (i = 0; i < eleCount; i++) {
        if (hashTable[i].count == 0)
            continue;
        myNode = hashTable[i].head;
```

```
        if (!myNode)
            continue;
        printf("\nData at index %d in Hash Table:\n", i);
        printf("VoterID     Name          Age   \n");
        printf("--------------------------------\n");
        while (myNode != NULL) {
            printf("%-12d", myNode->key);
            printf("%-15s", myNode->name);
            printf("%d\n", myNode->age);
            myNode = myNode->next;
        }
    }
    return;
}
int main() {
    int n, ch, key, age;
    char name[100];
    printf("Enter the number of elements:");
    scanf("%d", &n);
    eleCount = n;
    /* create hash table with "n" no of buckets */
    hashTable = (struct hash *) calloc(n, sizeof(struct hash));
    while (1) {
        printf("\n1. Insertion\t2. Deletion\n");
        printf("3. Searching\t4. Display\n5. Exit\n");
        printf("Enter your choice:");
        scanf("%d", &ch);
```

```
131.         switch (ch) {
132.         case 1:
133.             printf("Enter the key value:");
134.             scanf("%d", &key);
135.             getchar();
136.             printf("Name:");
137.             fgets(name, 100, stdin);
138.             name[strlen(name) - 1] = '\0';
139.             printf("Age:");
140.             scanf("%d", &age);
141.             /*inserting new node to hash table */
142.             insertToHash(key, name, age);
143.             break;
144.         case 2:
145.             printf("Enter the key to perform deletion:");
146.             scanf("%d", &key);
147.             /* delete node with "key" from hash table */
148.             deleteFromHash(key);
149.             break;
150.         case 3:
151.             printf("Enter the key to search:");
152.             scanf("%d", &key);
153.             searchInHash(key);
154.             break;
155.         case 4:
156.             display();
157.             break;
```

```
158.         case 5:
159.             exit(0);
160.         default:
161.             printf("U have entered wrong option!!\n");
162.             break;
163.         }
164.     }
165.     return 0;
166. }
```


Output:

```
$ gcc HashTablesLL.c
$ ./a.out

  Enter the number of elements:3

  1. Insertion 2. Deletion
  3. Searching 4. Display
  5. Exit
  Enter your choice:1
  Enter the key value:3
  Name:Sally
  Age:23

  1. Insertion 2. Deletion
  3. Searching 4. Display
  5. Exit
  Enter your choice:1
  Enter the key value:33
  Name:Harry
  Age:25

1. Insertion 2. Deletion
3. Searching 4. Display
5. Exit
Enter your choice:1
Enter the key value:7
Name:Nick
Age:30

1. Insertion 2. Deletion
3. Searching 4. Display
5. Exit
Enter your choice:1
Enter the key value:35
Name:Raj
Age:28

1. Insertion 2. Deletion
3. Searching 4. Display
5. Exit
Enter your choice:4
```

```
Data at index 0 in Hash Table:
VoterID     Name          Age
--------------------------------
33          Harry          25
3           Sally          23

Data at index 1 in Hash Table:
VoterID     Name          Age
--------------------------------
7           Nick           30

Data at index 2 in Hash Table:
VoterID     Name          Age
--------------------------------
35          Raj            28

1. Insertion 2. Deletion
3. Searching 4. Display
5. Exit
Enter your choice:2
Enter the key to perform deletion:33
Data deleted successfully from Hash Table


1. Insertion 2. Deletion
3. Searching 4. Display
5. Exit
Enter your choice:4

Data at index 0 in Hash Table:
VoterID     Name          Age
--------------------------------
3           Sally          23

Data at index 1 in Hash Table:
VoterID     Name          Age
--------------------------------
7           Nick           30

Data at index 2 in Hash Table:
VoterID     Name          Age
--------------------------------
35          Raj            28

1. Insertion 2. Deletion
3. Searching 4. Display
5. Exit
Enter your choice:3
Enter the key to search:35
VoterID  : 35
Name     : Raj
Age      : 28

1. Insertion 2. Deletion
3. Searching 4. Display
5. Exit
Enter your choice:5
```