C Program to Implement Hash Tables with Linear Probing

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

/* to store a data (consisting of key and value) in hash table array */
struct item
{
   int key;
   int value;
};

/* each hash table item has a flag (status) and data (consisting of key and value) */
struct hashtable_item
{

   int flag;
   /*
    * flag = 0 : data does not exist
    * flag = 1 : data exists
    * flag = 2 : data existed at least once
   */
```

```
    struct item *data;


};


struct hashtable_item *array;
int size = 0;
int max = 10;


/* initializing hash table array */
void init_array()
{
    int i;
    for (i = 0; i < max; i++)
    {
            array[i].flag = 0;
            array[i].data = NULL;
    }
}


/* to every key, it will generate a corresponding index */
int hashcode(int key)
{
    return (key % max);
}
```

```
/* to insert an element in the hash table */
void insert(int key, int value)
{
   int index = hashcode(key);
   int i = index;

   /* creating new item to insert in the hash table array */
   struct item *new_item = (struct item*) malloc(sizeof(struct item));
   new_item->key = key;
   new_item->value = value;

   /* probing through the array until we reach an empty space */
   while (array[i].flag == 1)
   {

         if (array[i].data->key == key)
      {

                  /* case where already existing key matches the given key */
                  printf("\n Key already exists, hence updating its value \n");
                  array[i].data->value = value;
                  return;

         }
```

```
            i = (i + 1) % max;
            if (i == index)
        {
                        printf("\n Hash table is full, cannot insert any more item \n");
                        return;
            }

    }

    array[i].flag = 1;
    array[i].data = new_item;
    size++;
    printf("\n Key (%d) has been inserted \n", key);

}



/* to remove an element from the hash table */
void remove_element(int key)
{
    int index = hashcode(key);
    int  i = index;

    /* probing through array until we reach an empty space where not even once an element had been present */
    while (array[i].flag != 0)
```

```
    {

        if (array[i].flag == 1  &&  array[i].data->key == key )
      {

                // case when data key matches the given key
                array[i].flag =  2;
                array[i].data = NULL;
                size--;
                printf("\n Key (%d) has been removed \n", key);
                return;

        }
        i = (i + 1) % max;
        if (i == index)
      {
                break;
        }

    }

    printf("\n This key does not exist \n");

}
```

```
/* to display all the elements of hash table */

void display()

{

   int i;

   for (i = 0; i < max; i++)

   {

          struct item *current = (struct item*) array[i].data;



          if (current == NULL)

      {

             printf("\n Array[%d] has no elements \n", i);

          }

          else

      {

             printf("\n Array[%d] has elements -: \n  %d (key) and %d(value) ", i, current->key, current-
>value);

          }

   }



}



int size_of_hashtable()

{

   return size;

}
```

```
void main()
{
        int choice, key, value, n, c;
        clrscr();

        array = (struct hashtable_item*) malloc(max * sizeof(struct hashtable_item*));
        init_array();

        do {
                printf("Implementation of Hash Table in C with Linear Probing \n\n");
                printf("MENU-: \n1.Inserting item in the Hashtable"
                 "\n2.Removing item from the Hashtable"
                 "\n3.Check the size of Hashtable"
                 "\n4.Display Hashtable"
                      "\n\n Please enter your choice-:");

                scanf("%d", &choice);

                switch(choice)
         {

                case 1:

                    printf("Inserting element in Hashtable\n");
                    printf("Enter key and value-:\t");
```

```
        scanf("%d %d", &key, &value);
        insert(key, value);

        break;

    case 2:

        printf("Deleting in Hashtable \n Enter the key to delete-:");
        scanf("%d", &key);
        remove_element(key);

        break;

    case 3:

        n = size_of_hashtable();
        printf("Size of Hashtable is-:%d\n", n);

        break;

    case 4:

        display();

        break;
```

```
                default:

                    printf("Wrong Input\n");

                }

                printf("\n Do you want to continue-:(press 1 for yes)\t");
                scanf("%d", &c);

        }while(c == 1);

        getch();

}
```

Output

Implementation of Hash Table in C with Linear Probing

MENU-:

1. Inserting item in the Hashtable

2. Removing item from the Hashtable

3. Check the size of Hashtable

4. Display Hashtable

Please enter your choice-: 3

Size of Hashtable is-: 0

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C with Linear Probing

MENU-:

1. Inserting item in the Hashtable

2. Removing item from the Hashtable

3. Check the size of Hashtable

4. Display Hashtable

Please enter your choice-: 1

Inserting element in Hashtable

Enter key and value-: 12 10

Key (12) has been inserted

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C with Linear Probing

MENU-:

1. Inserting item in the Hashtable

2. Removing item from the Hashtable

3. Check the size of Hashtable

4. Display Hashtable

Please enter your choice-: 1

Inserting element in Hash table

Enter key and value-: 122 4

Key (122) has been inserted

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C with Linear Probing

MENU-:

1. Inserting item in the Hashtable

2. Removing item from the Hashtable

3. Check the size of Hashtable

4. Display Hashtable

Please enter your choice-: 3

Size of Hashtable is-: 2

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C with Linear Probing

MENU-:

1. Inserting item in the Hashtable

2. Removing item from the Hashtable

3. Check the size of Hashtable

4. Display Hashtable

Please enter your choice-: 4

Array[0] has no elements

Array[1] has no elements

Array[2] has elements-:

12 (key) and 10 (value)

Array[3] has elements-:

122(key) and 5(value)

Array[4] has no elements

Array[5] has no elements

Array[6] has no elements

Array[7] has no elements

Array[8] has no elements

Array[9] has no elements

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C with Linear Probing

MENU-:

1. Inserting item in the Hashtable

2. Removing item from the Hashtable

3. Check the size of Hashtable

4. Display Hashtable

Please enter your choice-: 2

Deleting in Hashtable

Enter the key to delete-: 122

Key (122) has been removed

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C with Linear Probing

MENU-:

1. Inserting item in the Hashtable

2. Removing item from the Hashtable

3. Check the size of Hashtable

4. Display Hashtable

Please enter your choice-: 2

Deleting in Hashtable

Enter the key to delete-: 56

This key does not exist

Do you want to continue-:(press 1 for yes) 2