NAME: K.MRUDHULA

REG NO: 192372290

//1.SEPARATE HASHING:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct node {
	char* key;
	char* value;
	struct node* next;
};
void setNode(struct node* node, char* key, char* value)
{
	node->key = key;
	node->value = value;
	node->next = NULL;
	return;
};
struct hashMap {
	int numOfElements, capacity;
	struct node** arr;
};
void initializeHashMap(struct hashMap* mp)
{
	mp->capacity = 100;
	mp->numOfElements = 0;
	mp->arr = (struct node**)malloc(sizeof(struct node*)
```

```
                                                        * mp->capacity);
        return;
}

int hashFunction(struct hashMap* mp, char* key)
{
        int bucketIndex;
        int sum = 0, factor = 31;
        for (int i = 0; i < strlen(key); i++) {
                sum = ((sum % mp->capacity)
                        + (((int)key[i]) * factor) % mp->capacity)
                        % mp->capacity;
                factor = ((factor % __INT16_MAX__)
                                * (31 % __INT16_MAX__))
                                % __INT16_MAX__;
        }
        bucketIndex = sum;
        return bucketIndex;
}
void insert(struct hashMap* mp, char* key, char* value)
{
        int bucketIndex = hashFunction(mp, key);
        struct node* newNode = (struct node*)malloc(
                sizeof(struct node));
        setNode(newNode, key, value);
        if (mp->arr[bucketIndex] == NULL) {
                mp->arr[bucketIndex] = newNode;
```

```
        }
        else {
                newNode->next = mp->arr[bucketIndex];
                mp->arr[bucketIndex] = newNode;
        }
        return;
}
void delete (struct hashMap* mp, char* key)
{
        int bucketIndex = hashFunction(mp, key);
        struct node* prevNode = NULL;
        struct node* currNode = mp->arr[bucketIndex];
        while (currNode != NULL) {
                if (strcmp(key, currNode->key) == 0) {
                        if (currNode == mp->arr[bucketIndex]) {
                                mp->arr[bucketIndex] = currNode->next;
                        }
                        else {
                                prevNode->next = currNode->next;
                        }
                        free(currNode);
                        break;
                }
                prevNode = currNode;
                currNode = currNode->next;
        }
        return;
```

```
}

char* search(struct hashMap* mp, char* key)
{
	int bucketIndex = hashFunction(mp, key);
	struct node* bucketHead = mp->arr[bucketIndex];
	while (bucketHead != NULL) {
		if (bucketHead->key == key) {
			return bucketHead->value;
		}
		bucketHead = bucketHead->next;
	}
	char* errorMssg = (char*)malloc(sizeof(char) * 25);
	errorMssg = "Oops! No data found.\n";
	return errorMssg;
}
int main()
{
	struct hashMap* mp
		= (struct hashMap*)malloc(sizeof(struct hashMap));
	initializeHashMap(mp);
	insert(mp, "Yogaholic", "Anjali");
	insert(mp, "pluto14", "Vartika");
	insert(mp, "elite_Programmer", "Manish");
	insert(mp, "GFG", "GeeksforGeeks");
	insert(mp, "decentBoy", "Mayank");
	printf("%s\n", search(mp, "elite_Programmer"));
```

```
        printf("%s\n", search(mp, "Yogaholic"));
        printf("%s\n", search(mp, "pluto14"));
        printf("%s\n", search(mp, "decentBoy"));
        printf("%s\n", search(mp, "GFG"));
        printf("%s\n", search(mp, "randomKey"));
        printf("\nAfter deletion : \n");
        delete (mp, "decentBoy");
        printf("%s\n", search(mp, "decentBoy"));
        return 0;
}
```

OUTPUT:

```
Manish
Anjali
Vartika
Mayank
GeeksforGeeks
Oops! No data found.
After deletion :
Oops! No data found.
```

//2.LINEAR PROBING:

```
#include <stdio.h>
 #include<stdlib.h>
 #define TABLE_SIZE 10
 int h[TABLE_SIZE]={NULL};
 void insert()
 {
  int key,index,i,flag=0,hkey;
```

```
 printf("\nenter a value to insert into hash table\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE;i++)
  {
   index=(hkey+i)%TABLE_SIZE;
   if(h[index] == NULL)
   {
    h[index]=key;
     break;
   }
  }
  if(i == TABLE_SIZE)
   printf("\nelement cannot be inserted\n");
}
void search()
{
 int key,index,i,flag=0,hkey;
 printf("\nenter search element\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE; i++)
 {
  index=(hkey+i)%TABLE_SIZE;
  if(h[index]==key)
  {
   printf("value is found at index %d",index);
```

```
   break;
  }
 }
 if(i == TABLE_SIZE)
  printf("\n value is not found\n");
}
void display()
{
 int i;
 printf("\nelements in the hash table are \n");
 for(i=0;i< TABLE_SIZE; i++)
 printf("\nat index %d \t value = %d",i,h[i]);
}
 int main()
{
  int opt,i;
  while(1)
  {
    printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
    scanf("%d",&opt);
    switch(opt)
    {
      case 1:
        insert();
        break;
      case 2:
        display();
```

```
        break;
      case 3:
        search();
        break;
      case 4:exit(0);
    }
  }
  return 0;
}
```

OUTPUT:

```
Press 1. Insert        2. Display   3. Search     4.Exit
1
enter a value to insert into hash table
10
Press 1. Insert        2. Display   3. Search     4.Exit
1
enter a value to insert into hash table
12
Press 1. Insert        2. Display   3. Search     4.Exit
1
enter a value to insert into hash table
23
Press 1. Insert        2. Display   3. Search     4.Exit
1
enter a value to insert into hash table
42
Press 1. Insert        2. Display   3. Search     4.Exit
```

1

enter a value to insert into hash table

53

Press 1. Insert 2. Display 3. Search 4.Exit

1

enter a value to insert into hash table

62

Press 1. Insert 2. Display 3. Search 4.Exit

1

enter a value to insert into hash table

74

Press 1. Insert 2. Display 3. Search 4.Exit

1

enter a value to insert into hash table

85

Press 1. Insert 2. Display 3. Search 4.Exit

1

enter a value to insert into hash table

96

Press 1. Insert 2. Display 3. Search 4.Exit

1

enter a value to insert into hash table

105

Press 1. Insert 2. Display 3. Search 4.Exit

1

enter a value to insert into hash table

116

element cannot be inserted

Press 1. Insert 2. Display 3. Search 4.Exit

2

elements in the hash table are

at index 0 value = 10

at index 1 value = 105

at index 2 value = 12

at index 3 value = 23

at index 4 value = 42

at index 5 value = 53

at index 6 value = 62

at index 7 value = 74

at index 8 value = 85

at index 9 value = 96

Press 1. Insert 2. Display 3. Search 4.Exit

3

enter search element

116

value is not found

Press 1. Insert 2. Display 3. Search 4.Exit

4