WEEK5 MODULE5

NAME:MD.Amrath **CLASS:**AB2-305 **ID: B**181094

q1: Closed Adressing-division Method:

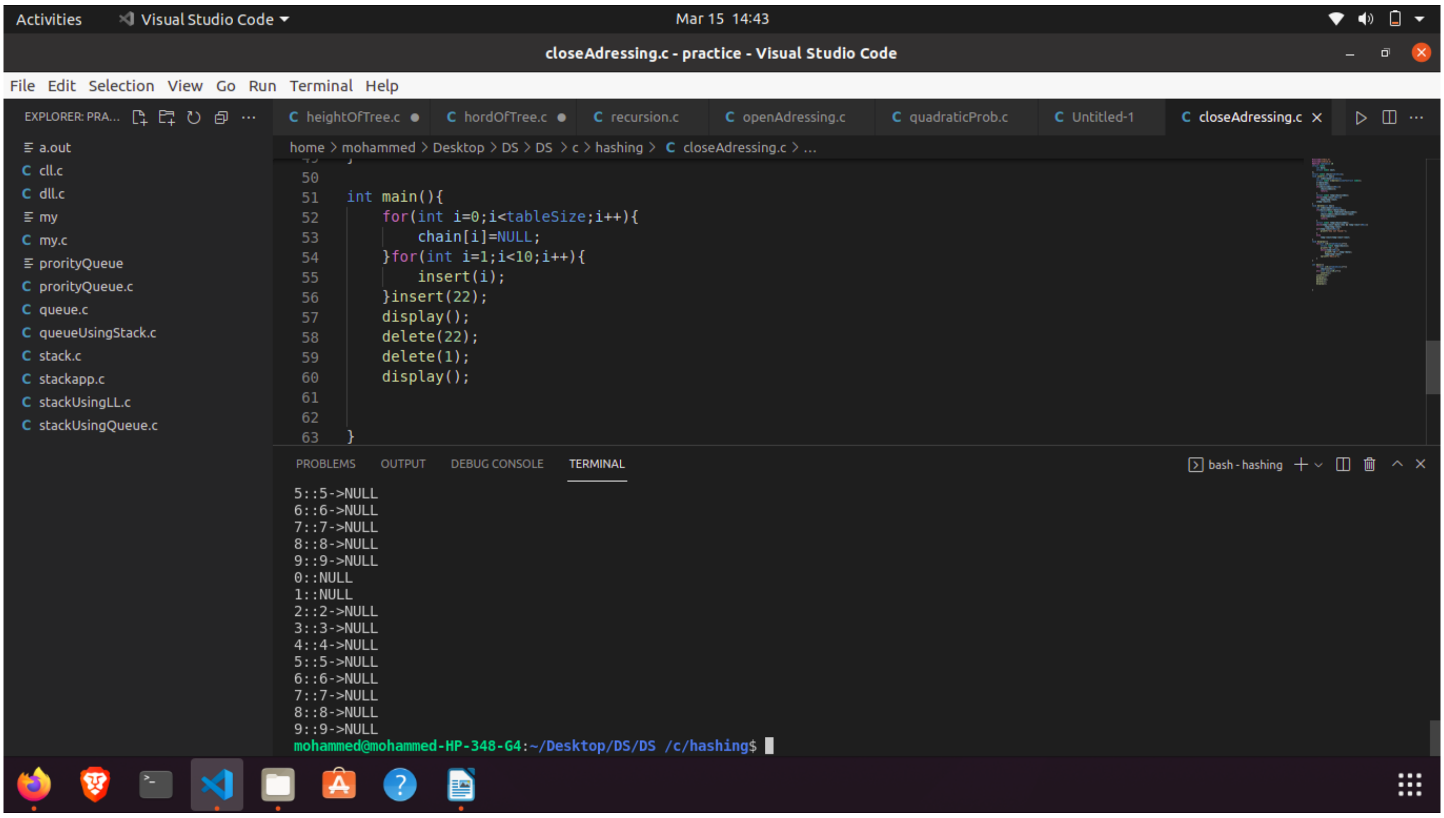
pseudocode:

```
1. Declare an array of a linked list with the hash table size.
2. Initialize an array of a linked list to NULL.
3. Find hash key.
4. If chain[key] == NULL
        Make chain[key] points to the key node.
5. Otherwise(collision),
        Insert the key node at the end of the chain[key].
```

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#define tableSize 10
struct node{
    int data;
    struct node* next;
};
struct node* chain[tableSize];
void insert(int key){
    int index=key%tableSize;
    struct node* n=malloc(sizeof(struct node));
    n->data=key;
    n->next=NULL;
    if(chain[index]==NULL){
        chain[index]=n;
        return;
    }
    struct node* temp=chain[index];
    while(temp->next!=NULL){
        temp=temp->next;
    }temp->next=n;
}
void delete(int key){
    int index=key%tableSize;
    if(chain[index]->data==key){
        struct node* todelete=chain[index];
        chain[index]=chain[index]->next;
        free(todelete);
        return;
    }
    struct node* temp=chain[index];
    while(temp->next->data!=key && temp->next!=NULL){
            temp=temp->next;
    }if(temp->next==NULL){
        printf("key not found!");
    }
    else
        temp->next=temp->next->next;
}
void display(){
    for(int i=0;i<tableSize;i++){
        struct node* temp=chain[i];
        printf("%d::",i);
        while(temp!=NULL){
            printf("%d->",temp->data);
            temp=temp->next;
        }printf("NULL\n");
    }
}

int main(){
    for(int i=0;i<tableSize;i++){
        chain[i]=NULL;
    }for(int i=1;i<10;i++){
        insert(i);
    }insert(22);
    display();
    delete(22);
    delete(1);
    display();


}
```

OUTPUT:



```
int main(){
    for(int i=0;i<tableSize;i++){
        chain[i]=NULL;
    }for(int i=1;i<10;i++){
        insert(i);
    }insert(22);
    display();
    delete(22);
    delete(1);
    display();


}
```

```
5::5->NULL
6::6->NULL
7::7->NULL
8::8->NULL
9::9->NULL
0::NULL
1::NULL
2::2->NULL
3::3->NULL
4::4->NULL
5::5->NULL
6::6->NULL
7::7->NULL
8::8->NULL
9::9->NULL
mohammed@mohammed-HP-348-G4:~/Desktop/DS/DS /c/hashing$
```

Q2:Linear probing:

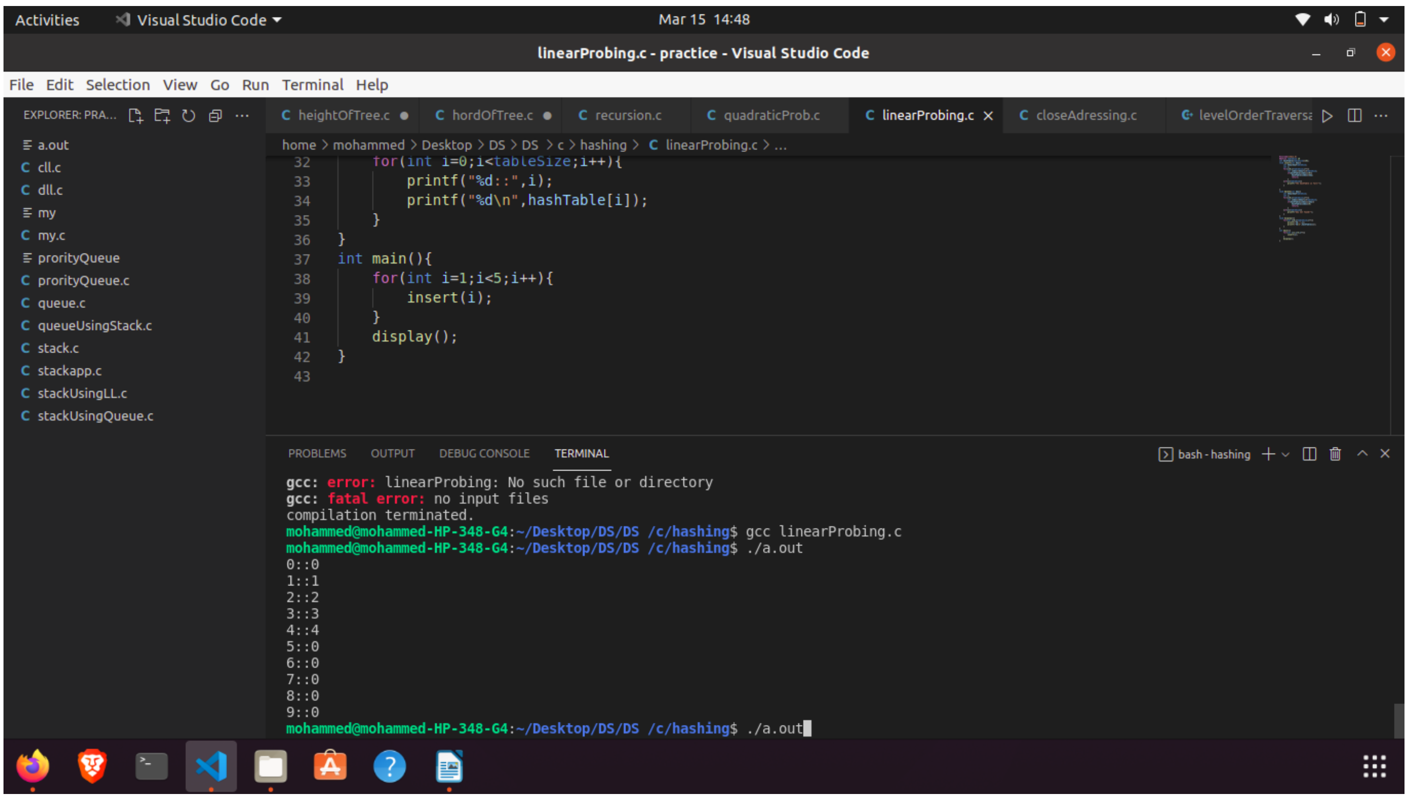
pseudocode:

```
1. Apply hash function on the key value and get the address of the location.
    2. If the location is free, then
        i) Store the key value at this location, else
        ii) Check the remaining locations of the table one after the other till an
        empty location is reached. Wrap around on the table can be used. When
        we reach the end of the table, start looking again from the beginning.
        iii) Store the key in this empty location.
  3.End
```

CODE:

```
#include<stdio.h>
#define tableSize 10
int hashTable[tableSize]={0};
void insert(int key){
    int hkey=key%tableSize;
    int i;
    for(i=0;i<tableSize;i++){
        int index=(hkey+i)%tableSize;
        if(hashTable[index]==0){
            hashTable[index]=key;
            return;
        }
    }if(i==tableSize){
        printf("The hashTable is full!");
    }

}
void delete(int key){
    int hkey=key%tableSize;
    int i;
    for(i=0;i<tableSize;i++){
        int index=(hkey+i)%tableSize;
        if(hashTable[index]==key){
            hashTable[index]=0;
            return;
        }
    }if(i==tableSize){
        printf("key not found!");
    }
}
void display(){
    for(int i=0;i<tableSize;i++){
        printf("%d::",i);
        printf("%d\n",hashTable[i]);
    }
}
int main(){
    for(int i=1;i<5;i++){
        insert(i);
    }
    display();
}
```

OUTPUT:



```c
    for(int i=0;i<tableSize;i++){
        printf("%d::",i);
        printf("%d\n",hashTable[i]);
    }
}
int main(){
    for(int i=1;i<5;i++){
        insert(i);
    }
    display();
}
```

```
gcc: error: linearProbing: No such file or directory
gcc: fatal error: no input files
compilation terminated.
mohammed@mohammed-HP-348-G4:~/Desktop/DS/DS /c/hashing$ gcc linearProbing.c
mohammed@mohammed-HP-348-G4:~/Desktop/DS/DS /c/hashing$ ./a.out
0::0
1::1
2::2
3::3
4::4
5::0
6::0
7::0
8::0
9::0
mohammed@mohammed-HP-348-G4:~/Desktop/DS/DS /c/hashing$ ./a.out
```

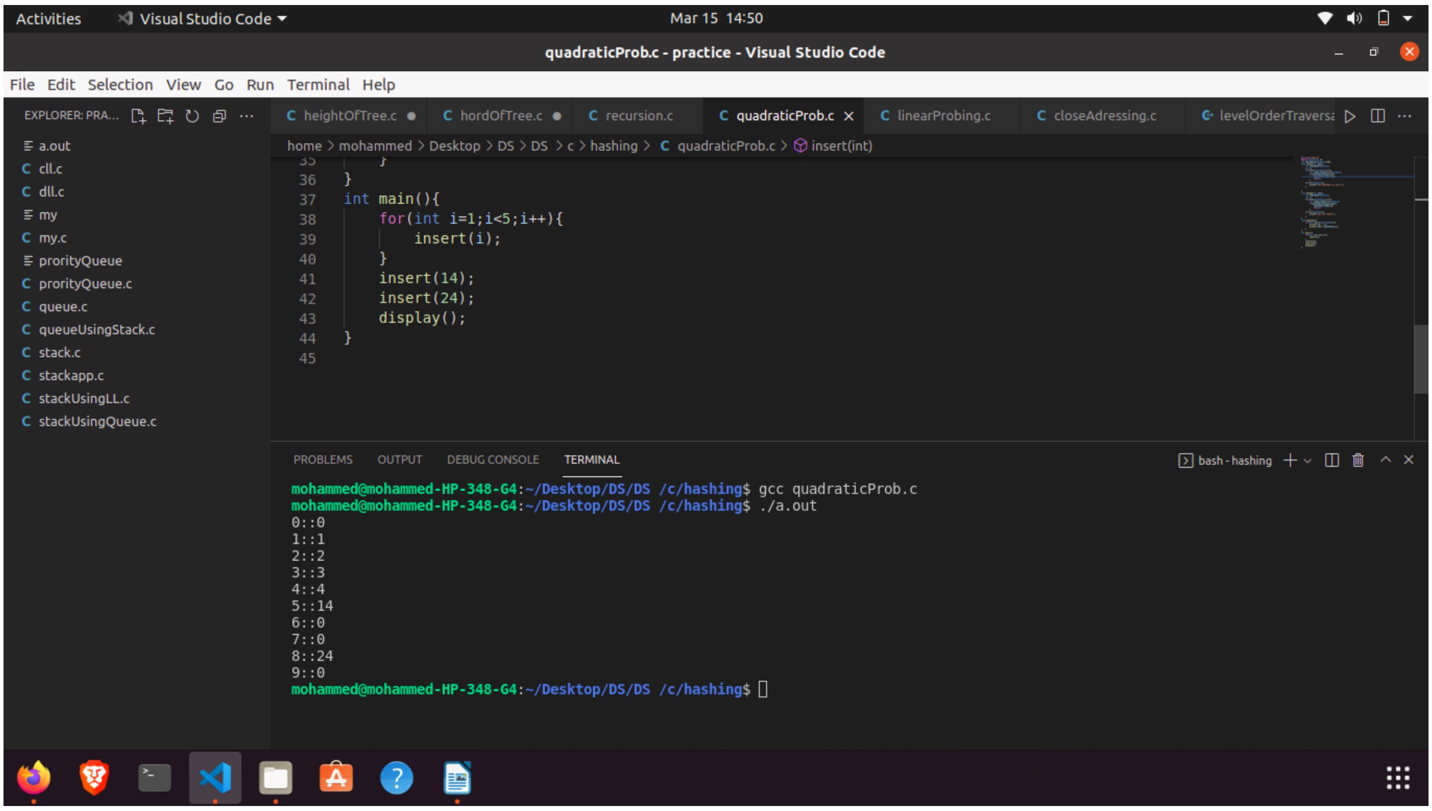
Q3: Quadratic Probing:

```
PSEUDOCODE:
1. Create an array of structure (i.e a hash table).
2. Take a key and a value to be stored in hash table as input.
3. Corresponding to the key, an index will be generated i.e every key is stored in a particular array index.
4. Using the generated index, access the data located in that array index.
5. In case of absence of data, create one and insert the data item (key and value) into it and increment the size of hash table.
6. In case the data exists, probe through the subsequent elements (looping back if necessary) for free space to insert new data item.
Note: This probing will continue until we reach the same element again (from where we began probing)
Note: Here, unlike Linear Probing, probing will be done according to the following formula –
(currentPosition + h) % arraySize => Linear Probing
(currentPosition + (h * h)) % arraySize => Quadratic Probing
where h = 1, 2, 3, 4 and so on.
7. To display all the elements of hash table, element at each index is accessed (via for loop).
8. To remove a key from hash table, we will first calculate its index and delete it if key matches, else probe through elements until we find key or an empty space where not a single data has been entered (means data does not exist in the hash table).
9. Exit
```

CODE:

```
#include<stdio.h>
#define tableSize 10
int hashTable[tableSize]={0};
void insert(int key){
    int hkey=key%tableSize;
    int i;
    for(i=0;i<tableSize;i++){
        int index=(hkey+i*i)%tableSize;
        if(hashTable[index]==0){
            hashTable[index]=key;
            return;
        }
    }if(i==tableSize){
        printf("The hashTable is full!");
    }

}
void delete(int key){
    int hkey=key%tableSize;
    int i;
    for(i=0;i<tableSize;i++){
        int index=(hkey+i)%tableSize;
        if(hashTable[index]==key){
            hashTable[index]=0;
            return;
        }
    }if(i==tableSize){
        printf("key not found!");
    }
}
void display(){
    for(int i=0;i<tableSize;i++){
        printf("%d::",i);
        printf("%d\n",hashTable[i]);
    }
}
int main(){
    for(int i=1;i<5;i++){
        insert(i);
    }
    insert(14);
    insert(24);
    display();
}
```

OUTPUT:



```c
    }
}
int main(){
    for(int i=1;i<5;i++){
        insert(i);
    }
    insert(14);
    insert(24);
    display();
}
```

```
mohammed@mohammed-HP-348-G4:~/Desktop/DS/DS /c/hashing$ gcc quadraticProb.c
mohammed@mohammed-HP-348-G4:~/Desktop/DS/DS /c/hashing$ ./a.out
0::0
1::1
2::2
3::3
4::4
5::14
6::0
7::0
8::24
9::0
mohammed@mohammed-HP-348-G4:~/Desktop/DS/DS /c/hashing$
```

Q4:DoubleHashing:

PSEUDOCODE:

```
Hashtable is an array of size = TABLE_SIZE
Step 1: Read the value to be inserted,key
Step 2: let i = 0
Step 3: Let R be the nearest prime < TABLE_SIZE
Step 4: hkey=key%TABLE_SIZE
Step 5: compute the index at which the value has to be inserted in hash table
        index = (hkey + i * (R- (value % R))% TABLE_SIZE
Step 6: if there is no element at that index then  insert the value at index and STOP
Step 7: If there is already an element at that index
        step 6.1: i = i+1
step 8: if i < TABLE_SIZE then go to step 5
```

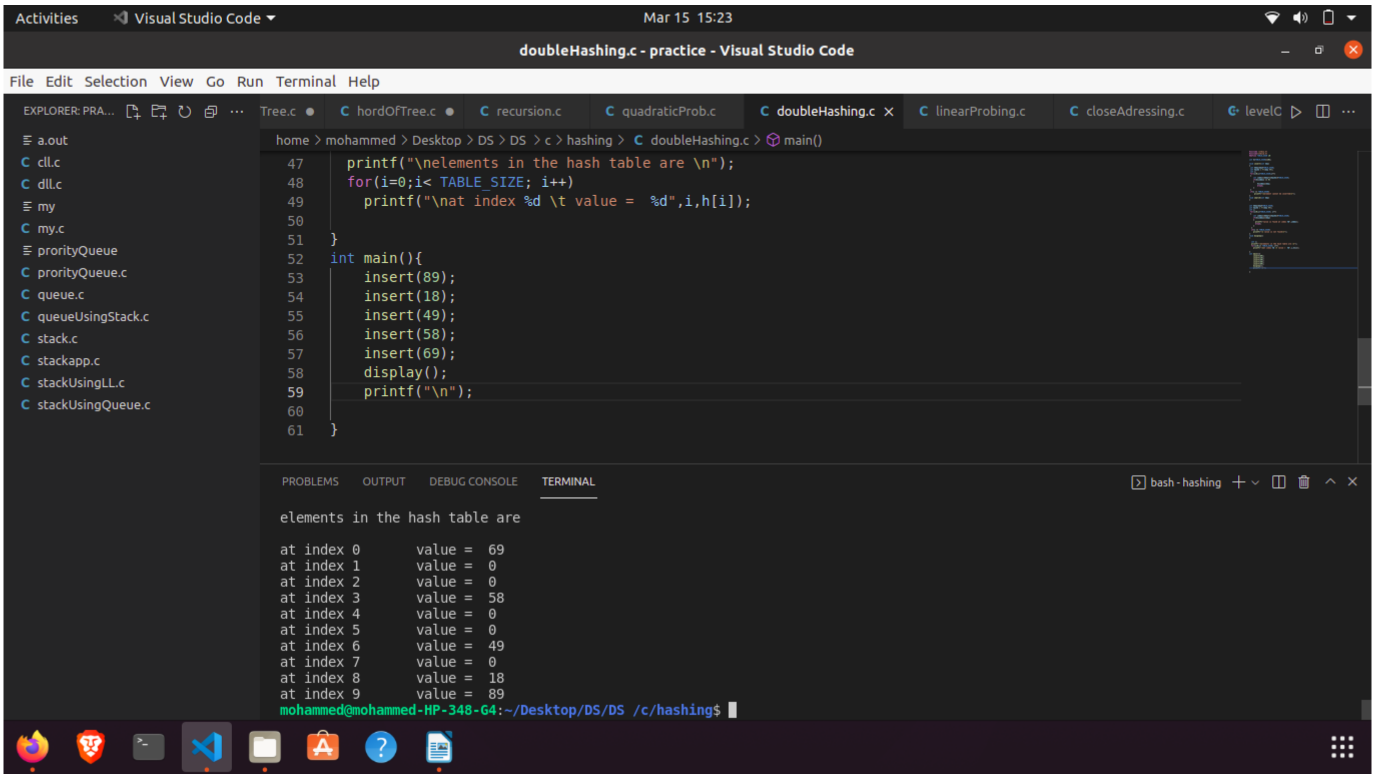
CODE:

```c
#include <stdio.h>
#include<stdlib.h>
#define TABLE_SIZE 10

int h[TABLE_SIZE]={0};
```

```
void insert(int key)
{
 int hkey=key%TABLE_SIZE;
 int hash2 = 7-(key %7);
 int i;
 for(i=0;i<TABLE_SIZE;i++)
 {
    int index=(hkey+i*hash2)%TABLE_SIZE;
    if(h[index] == 0)
    {
        h[index]=key;
        break;
    }
 }
 if(i == TABLE_SIZE)
     printf("\nelement cannot be inserted\n");
}
void search(int key)
{


int hkey=key%TABLE_SIZE;
int hash2 = 7-(key %7);
int i;
 for(i=0;i<TABLE_SIZE; i++)
 {
    int index=(hkey+i*hash2)%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(){
    insert(89);
    insert(18);
    insert(49);
    insert(58);
    insert(69);
    display();
    printf("\n");
}
```

OUTPUT:



```
    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(){
    insert(89);
    insert(18);
    insert(49);
    insert(58);
    insert(69);
    display();
    printf("\n");

}
```

```
elements in the hash table are

at index 0      value =  69
at index 1      value =  0
at index 2      value =  0
at index 3      value =  58
at index 4      value =  0
at index 5      value =  0
at index 6      value =  49
at index 7      value =  0
at index 8      value =  18
at index 9      value =  89
mohammed@mohammed-HP-348-G4:~/Desktop/DS/DS /c/hashing$
```