Bansilal Ramnath Agarwal Charitable Trust's
Vishwakarma Institute of Technology
(An Autonomous Institute affiliated to Savitribai Phule Pune University)
PUNE-411037
THE ROBOTICS FORUM

TRF FINAL TASK 2021 – PROGRAMMING DOMAIN

Hash Map

Group Members:

SR No.	Name	Year	GR No.
1.	Pratham Ghule	TY	11910546
2.	Gaurav Pawar	TY	12020233
3.	Kapil Bhale	TY	11910408

Project GitHub Link:

https://github.com/RoboSpark-2021/robospark-2021-FT-Hash-Map

Project Algorithm:

Linear Probing uses a regular one dimensional array.

1) Insertion

STEP 1: START

STEP 2: Calculate the hash key. Key = data % size;

STEP 3: If hashTable[key] is empty, store the value directly. hashTable[key] = data.

STEP 4: If the hash index already has some value, check for next index.
With key = (key+1) % size;

STEP 5: If the next index is available hashTable[key], store the value. Else go to step 4

STEP 7: STOP

2) Searching

```
STEP 1: START

STEP 2: Calculate the hash key. Key = data % size;

STEP 3: If hashTable[key] is not empty, and hashTable[key] is equal to searchKey then
return the value and go to step 6 else continue.

STEP 4: check for next index.
        With key = (key+1) % size;
        Go to step 3.

STEP 5: if whole array is traversed then return Key not found. Go to step 6.

STEP 6: STOP
```

3) Delete

```
STEP 1: START

STEP 2: Calculate the hash key. Key = data % size;

STEP 3: If hashTable[key] is not empty, and hashTable[key] is equal to delKey then return
assign key with -1and value with empty string and go to step 6 else continue.

STEP 4: check for next index.
        With key = (key+1) % size;
        Go to step 3.

STEP 5: if whole array is traversed then return Key not found. Go to step 6.

STEP 6: STOP
```

Methods:

Display (): This will print the Hash Table.

Mod Function (): Function to find location for key to place in hashmap.

Insert (k, value): Keep probing until the slot's key doesn't have value or an empty slot is reached.

Get Value (k): If a key is present it will return the value.

Search (k): Keep probing until the slot's key doesn't become equal to k or an empty slot is reached.

Modify (k, value): By using the search method search for the key and then modify that value.

Delete (k): If we simply delete a key, then the search may fail. So slots of deleted keys are marked specially as "NULL".
The insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot.

Delete All (): Will delete complete Hash Table.

HeapSort (): Will print the key-value pair in the sorted order according to key as well as value.

Problems Faced:

1. Collision in HashMap is possible because hash function uses hashCode() of key object and equals() and hashCode() contract doesn't guarantee different hashCode for different objects. Resulting in overridden values
2. Sorting the hashmap array directly was removing the significance of the hashmap as the indexes were modified while doing that so if we sort the hashmap then it will be no longer of use.
3. When creating the KeyMap array and valueMap array by default it stores the garbage values. So we were not able to know if the key is filled or not so it was difficult to keep track of filled key.

TRF FINAL TASK 2021 – PROGRAMMING DOMAIN

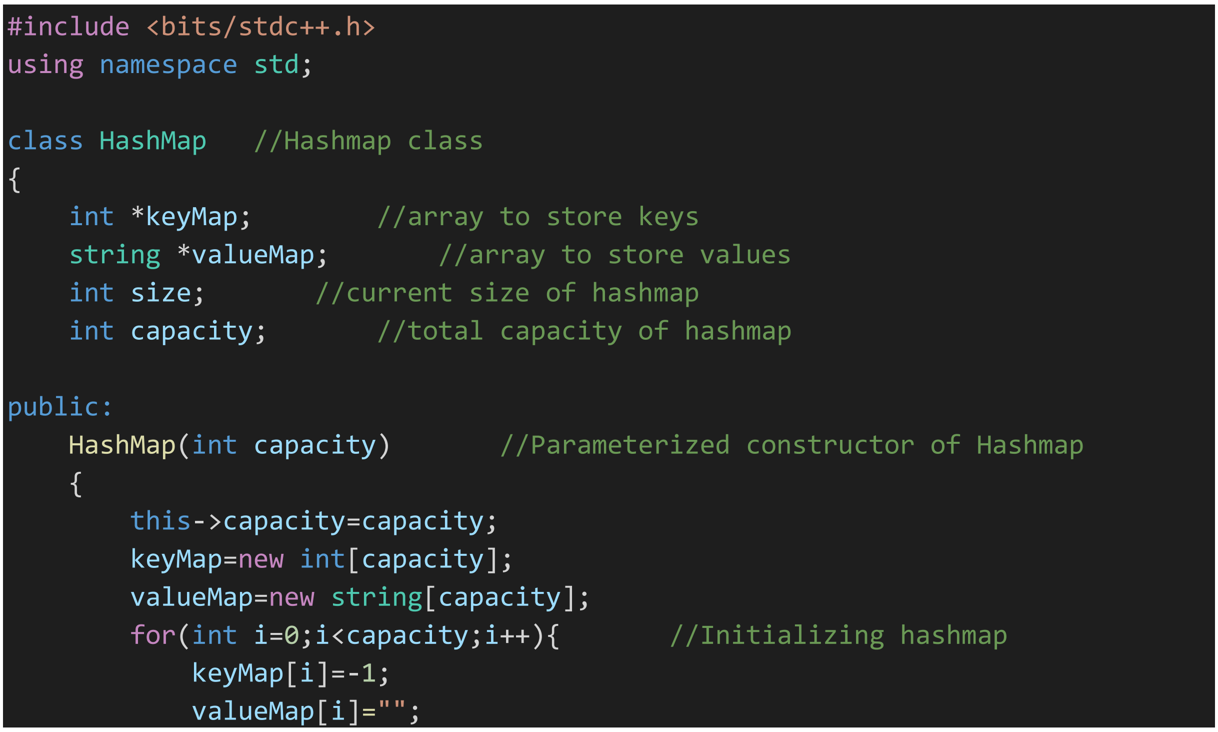
Alternative Solutions found for problems mentioned above:

1. To solve the collision problem we used the linear probing algorithm In this technique, if a value is already stored at a location generated by h(k), it means collision occurred then we do a sequential search to find the empty location. Here the idea is to place a value in the next available position. Because in this approach searches are performed sequentially so it’s known as linear probing. Here array or hash table is considered circular because when the last slot reached an empty location not found then the search proceeds to the first location of the array.

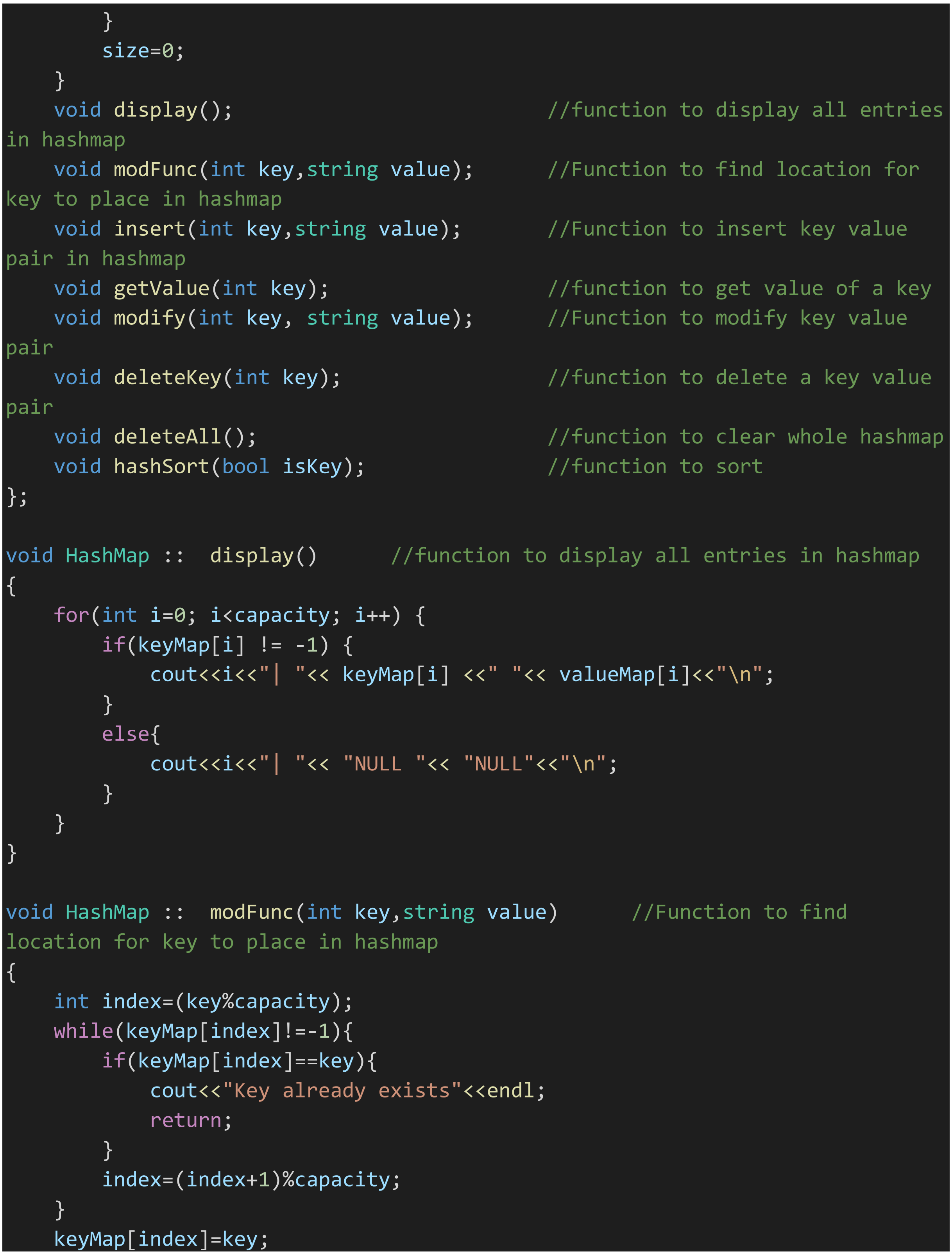
2. We created new array at the time of sorting and after sorting them we printed the sorted array so that indexes in the hash table won’t get affected by that.

3. To solve this problem we initialized hashmap with default values to the keyMap array and valueMap array so that we can keep track of key values are filled or not.

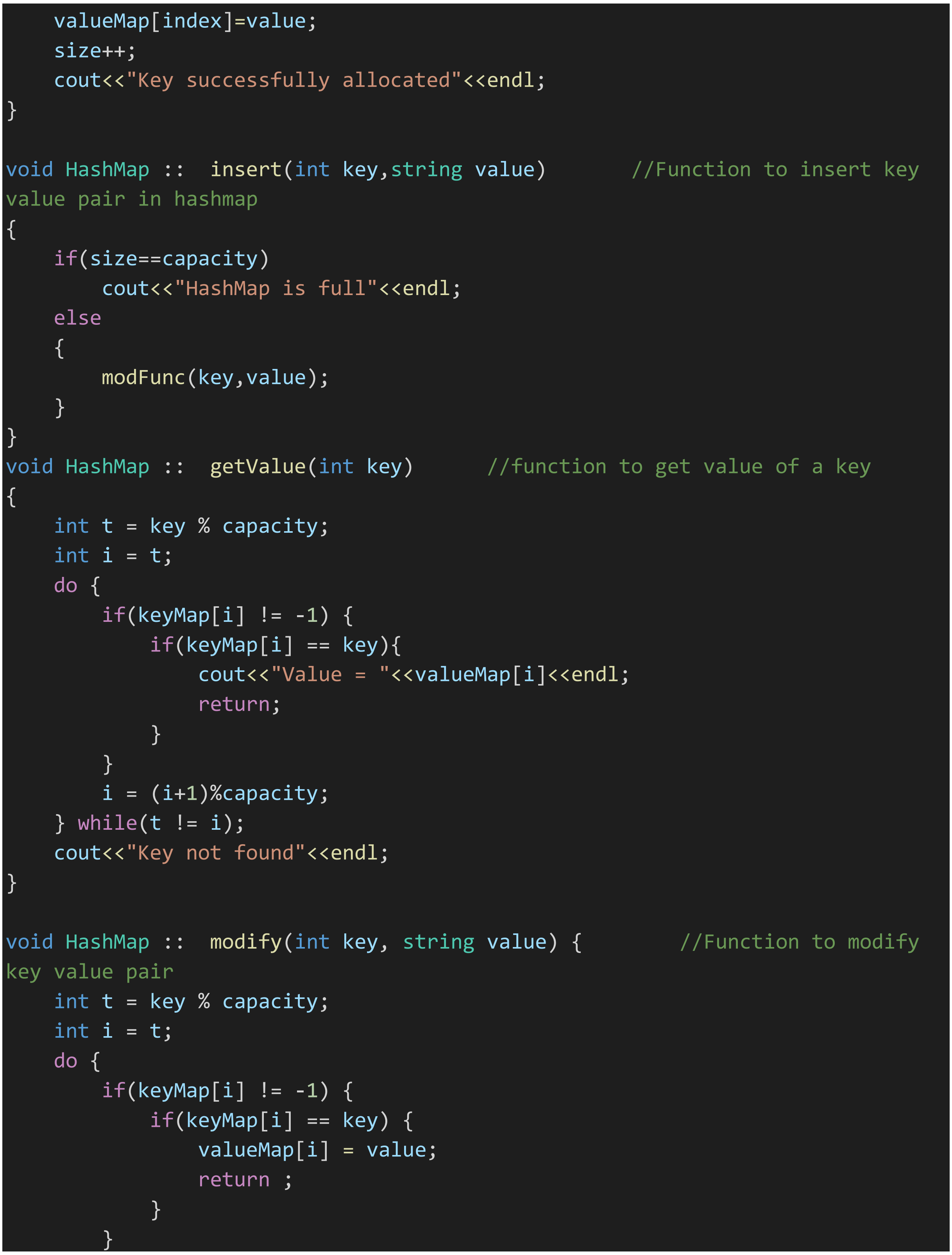
Code snippet:



```
#include <bits/stdc++.h>
using namespace std;

class HashMap   //Hashmap class
{
    int *keyMap;        //array to store keys
    string *valueMap;       //array to store values
    int size;       //current size of hashmap
    int capacity;       //total capacity of hashmap

public:
    HashMap(int capacity)       //Parameterized constructor of Hashmap
    {
        this->capacity=capacity;
        keyMap=new int[capacity];
        valueMap=new string[capacity];
        for(int i=0;i<capacity;i++){       //Initializing hashmap
            keyMap[i]=-1;
            valueMap[i]="";
```

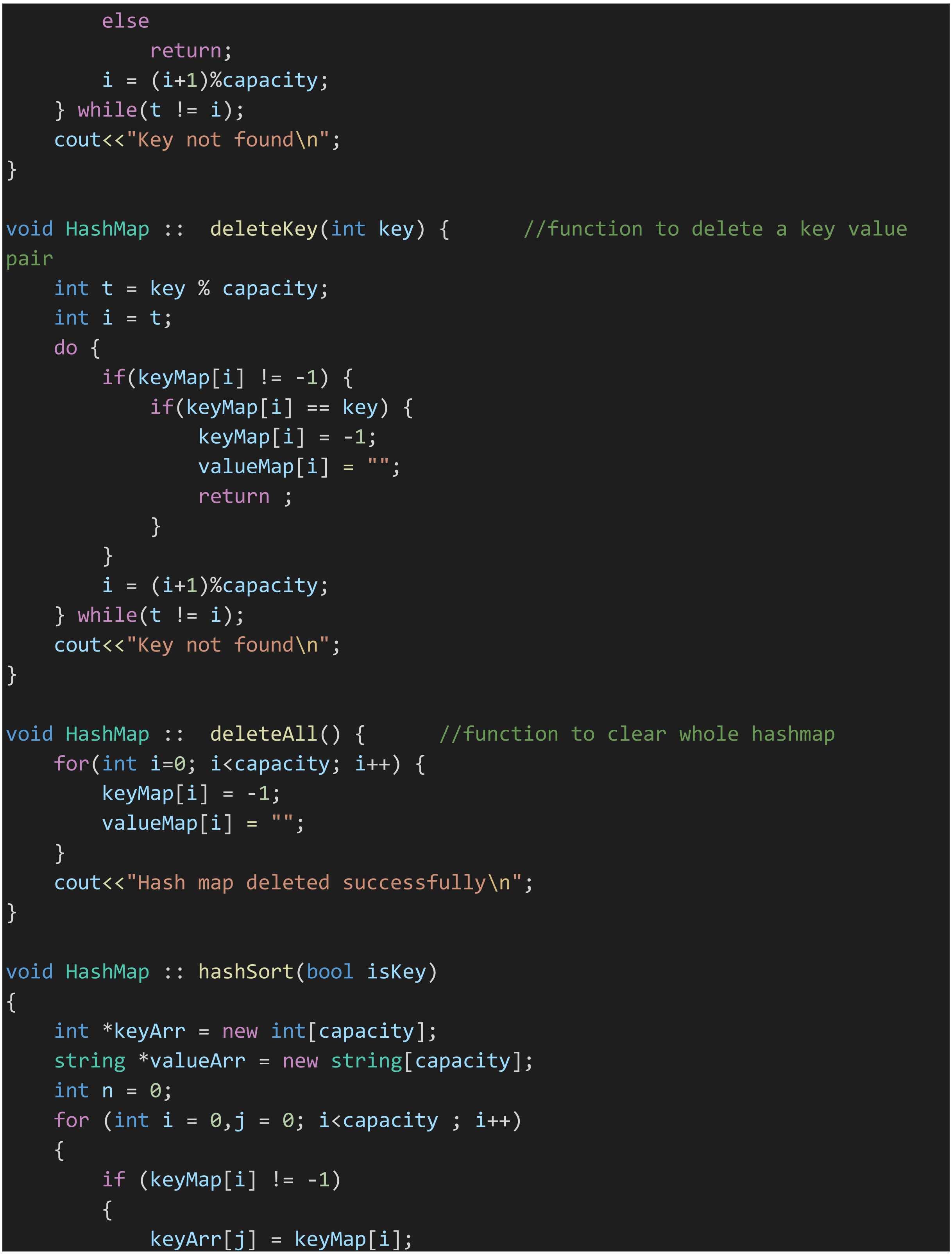


```
        }
        size=0;
    }
    void display();                          //function to display all entries
in hashmap
    void modFunc(int key,string value);      //Function to find location for
key to place in hashmap
    void insert(int key,string value);       //Function to insert key value
pair in hashmap
    void getValue(int key);                  //function to get value of a key
    void modify(int key, string value);      //Function to modify key value
pair
    void deleteKey(int key);                 //function to delete a key value
pair
    void deleteAll();                        //function to clear whole hashmap
    void hashSort(bool isKey);               //function to sort
};

void HashMap ::  display()      //function to display all entries in hashmap
{
    for(int i=0; i<capacity; i++) {
        if(keyMap[i] != -1) {
            cout<<i<<"| "<< keyMap[i] <<" "<< valueMap[i]<<"\n";
        }
        else{
            cout<<i<<"| "<< "NULL "<< "NULL"<<"\n";
        }
    }
}

void HashMap ::  modFunc(int key,string value)      //Function to find
location for key to place in hashmap
{
    int index=(key%capacity);
    while(keyMap[index]!=-1){
        if(keyMap[index]==key){
            cout<<"Key already exists"<<endl;
            return;
        }
        index=(index+1)%capacity;
    }
    keyMap[index]=key;
```

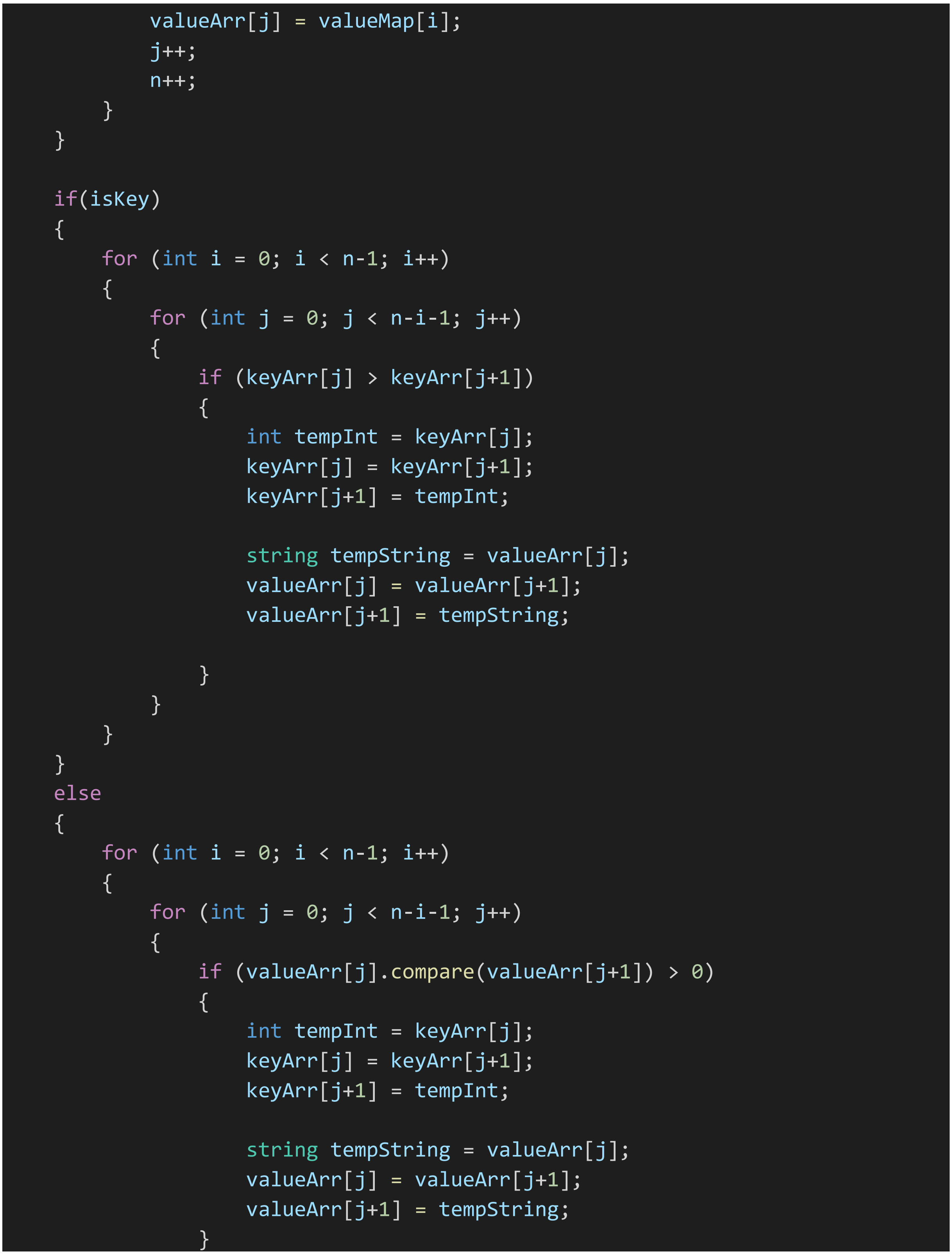


```
    valueMap[index]=value;
    size++;
    cout<<"Key successfully allocated"<<endl;
}

void HashMap ::  insert(int key,string value)       //Function to insert key value pair in hashmap
{
    if(size==capacity)
        cout<<"HashMap is full"<<endl;
    else
    {
        modFunc(key,value);
    }
}
void HashMap ::  getValue(int key)      //function to get value of a key
{
    int t = key % capacity;
    int i = t;
    do {
        if(keyMap[i] != -1) {
            if(keyMap[i] == key){
                cout<<"Value = "<<valueMap[i]<<endl;
                return;
            }
        }
        i = (i+1)%capacity;
    } while(t != i);
    cout<<"Key not found"<<endl;
}

void HashMap ::  modify(int key, string value) {        //Function to modify key value pair
    int t = key % capacity;
    int i = t;
    do {
        if(keyMap[i] != -1) {
            if(keyMap[i] == key) {
                valueMap[i] = value;
                return ;
            }
        }
```

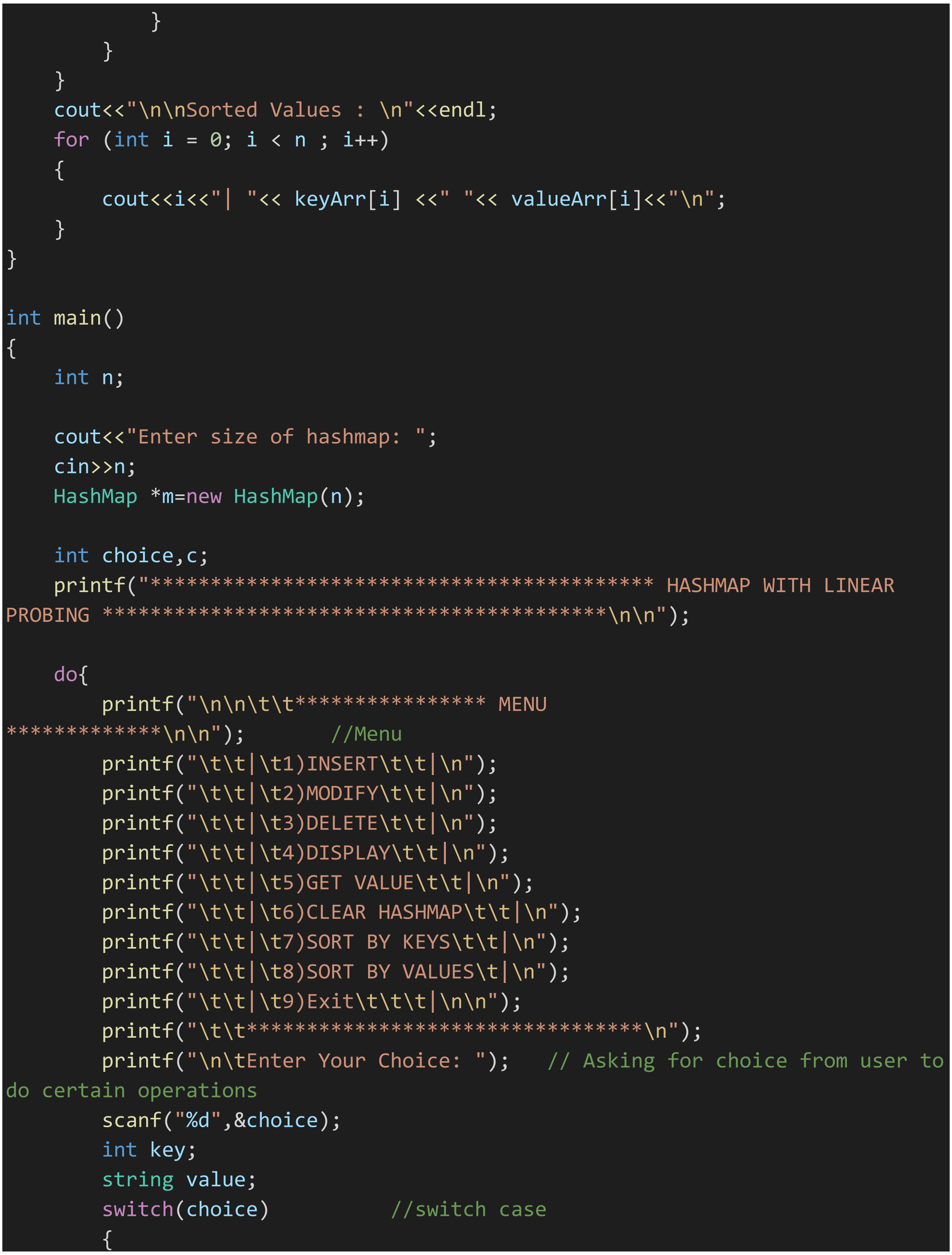
TRF FINAL TASK 2021 – PROGRAMMING DOMAIN



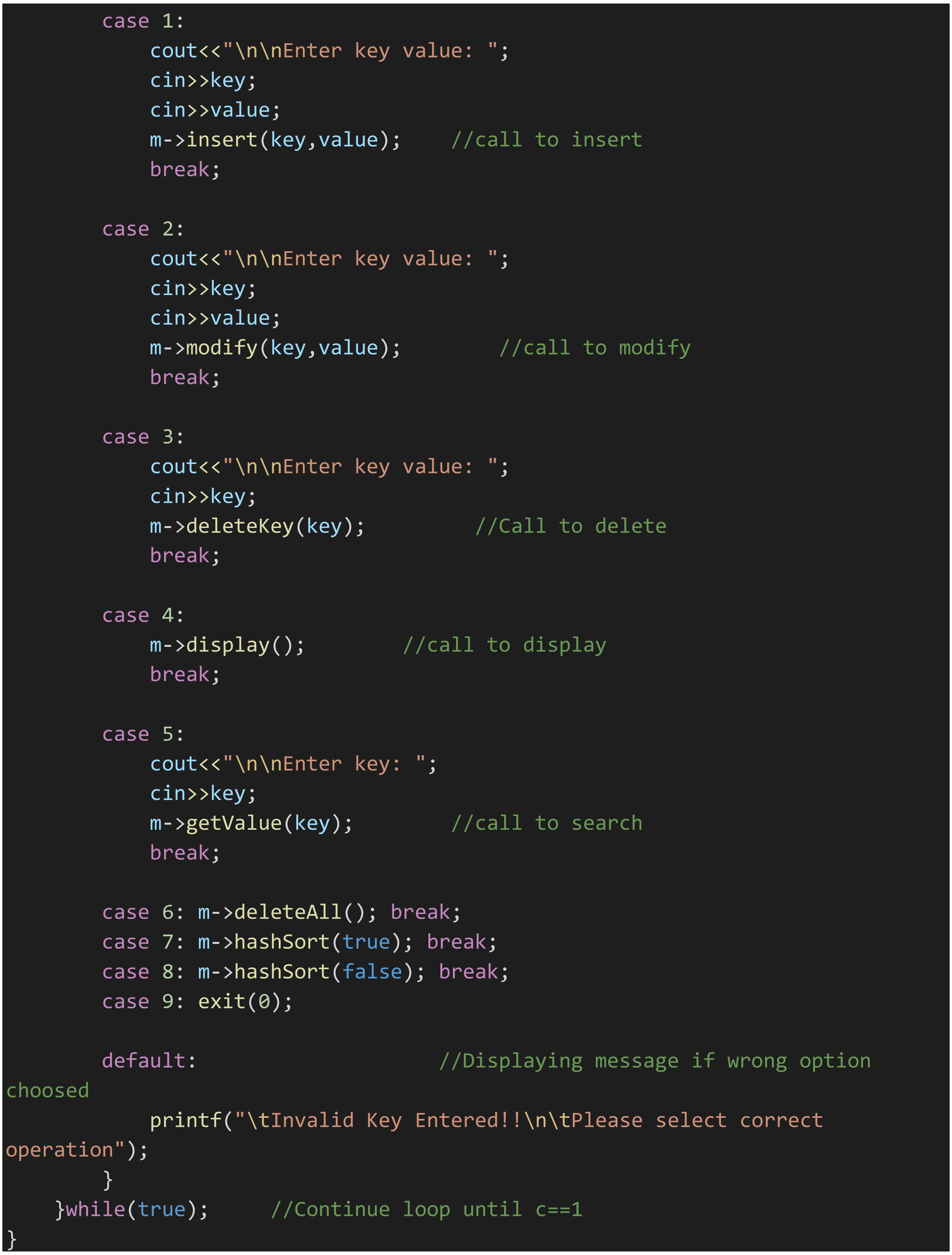
```
        else
            return;
        i = (i+1)%capacity;
    } while(t != i);
    cout<<"Key not found\n";
}

void HashMap ::  deleteKey(int key) {      //function to delete a key value
pair
    int t = key % capacity;
    int i = t;
    do {
        if(keyMap[i] != -1) {
            if(keyMap[i] == key) {
                keyMap[i] = -1;
                valueMap[i] = "";
                return ;
            }
        }
        i = (i+1)%capacity;
    } while(t != i);
    cout<<"Key not found\n";
}

void HashMap ::  deleteAll() {      //function to clear whole hashmap
    for(int i=0; i<capacity; i++) {
        keyMap[i] = -1;
        valueMap[i] = "";
    }
    cout<<"Hash map deleted successfully\n";
}

void HashMap :: hashSort(bool isKey)
{
    int *keyArr = new int[capacity];
    string *valueArr = new string[capacity];
    int n = 0;
    for (int i = 0,j = 0; i<capacity ; i++)
    {
        if (keyMap[i] != -1)
        {
            keyArr[j] = keyMap[i];
```



```
                valueArr[j] = valueMap[i];
                j++;
                n++;
            }
        }

    if(isKey)
    {
        for (int i = 0; i < n-1; i++)
        {
            for (int j = 0; j < n-i-1; j++)
            {
                if (keyArr[j] > keyArr[j+1])
                {
                    int tempInt = keyArr[j];
                    keyArr[j] = keyArr[j+1];
                    keyArr[j+1] = tempInt;

                    string tempString = valueArr[j];
                    valueArr[j] = valueArr[j+1];
                    valueArr[j+1] = tempString;

                }
            }
        }
    }
    else
    {
        for (int i = 0; i < n-1; i++)
        {
            for (int j = 0; j < n-i-1; j++)
            {
                if (valueArr[j].compare(valueArr[j+1]) > 0)
                {
                    int tempInt = keyArr[j];
                    keyArr[j] = keyArr[j+1];
                    keyArr[j+1] = tempInt;

                    string tempString = valueArr[j];
                    valueArr[j] = valueArr[j+1];
                    valueArr[j+1] = tempString;
                }
```

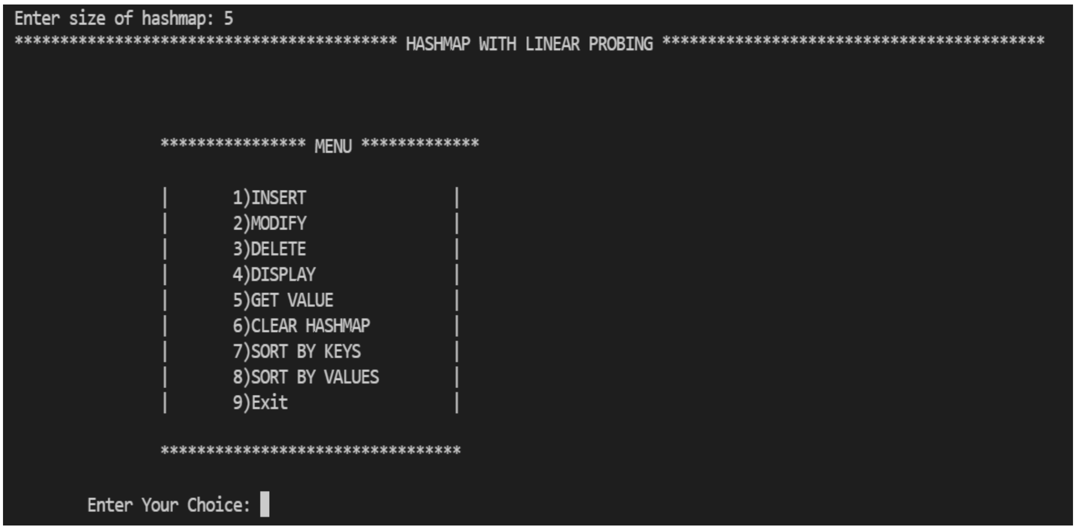


```
            }
        }
    }
    cout<<"\n\nSorted Values : \n"<<endl;
    for (int i = 0; i < n ; i++)
    {
        cout<<i<<"| "<< keyArr[i] <<" "<< valueArr[i]<<"\n";
    }
}

int main()
{
    int n;

    cout<<"Enter size of hashmap: ";
    cin>>n;
    HashMap *m=new HashMap(n);

    int choice,c;
    printf("****************************************** HASHMAP WITH LINEAR
PROBING ******************************************\n\n");

    do{
        printf("\n\n\t\t**************** MENU
*************\n\n");       //Menu
        printf("\t\t|\t1)INSERT\t\t|\n");
        printf("\t\t|\t2)MODIFY\t\t|\n");
        printf("\t\t|\t3)DELETE\t\t|\n");
        printf("\t\t|\t4)DISPLAY\t\t|\n");
        printf("\t\t|\t5)GET VALUE\t\t|\n");
        printf("\t\t|\t6)CLEAR HASHMAP\t\t|\n");
        printf("\t\t|\t7)SORT BY KEYS\t\t|\n");
        printf("\t\t|\t8)SORT BY VALUES\t|\n");
        printf("\t\t|\t9)Exit\t\t\t|\n\n");
        printf("\t\t*********************************\n");
        printf("\n\tEnter Your Choice: ");   // Asking for choice from user to
do certain operations
        scanf("%d",&choice);
        int key;
        string value;
        switch(choice)          //switch case
        {
```

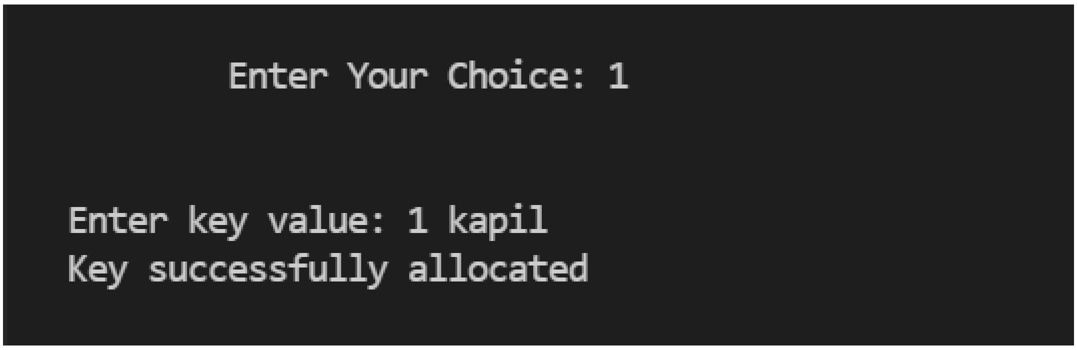


```
        case 1:
            cout<<"\n\nEnter key value: ";
            cin>>key;
            cin>>value;
            m->insert(key,value);    //call to insert
            break;

        case 2:
            cout<<"\n\nEnter key value: ";
            cin>>key;
            cin>>value;
            m->modify(key,value);        //call to modify
            break;

        case 3:
            cout<<"\n\nEnter key value: ";
            cin>>key;
            m->deleteKey(key);         //Call to delete
            break;

        case 4:
            m->display();        //call to display
            break;

        case 5:
            cout<<"\n\nEnter key: ";
            cin>>key;
            m->getValue(key);        //call to search
            break;

        case 6: m->deleteAll(); break;
        case 7: m->hashSort(true); break;
        case 8: m->hashSort(false); break;
        case 9: exit(0);

        default:                    //Displaying message if wrong option
choosed
            printf("\tInvalid Key Entered!!\n\tPlease select correct
operation");
        }
    }while(true);     //Continue loop until c==1
}
```

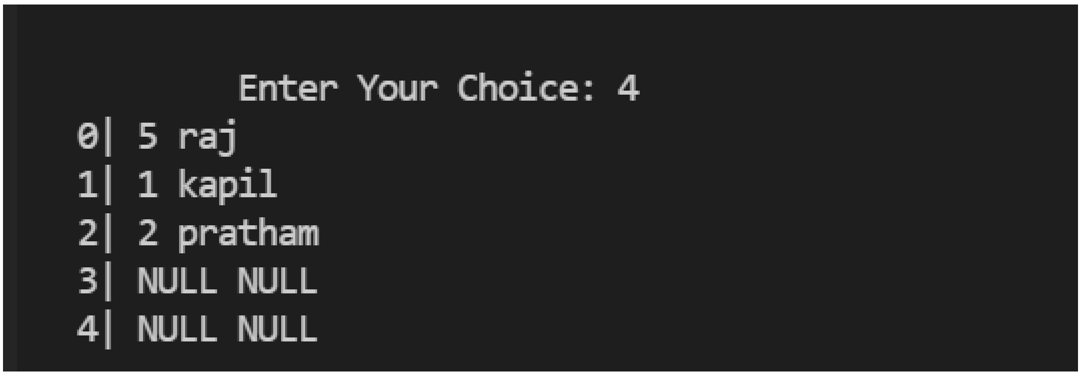
Output:



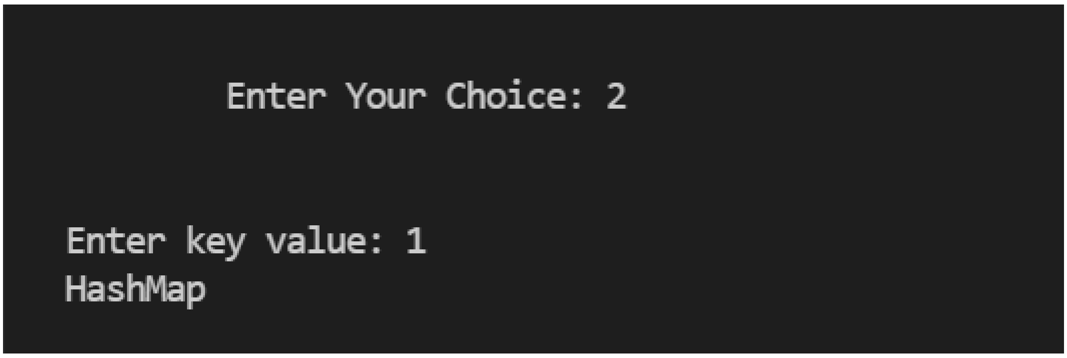
```
Enter size of hashmap: 5
****************************************** HASHMAP WITH LINEAR PROBING ******************************************



                 **************** MENU *************

                 |      1)INSERT                 |
                 |      2)MODIFY                 |
                 |      3)DELETE                 |
                 |      4)DISPLAY                |
                 |      5)GET VALUE              |
                 |      6)CLEAR HASHMAP          |
                 |      7)SORT BY KEYS           |
                 |      8)SORT BY VALUES         |
                 |      9)Exit                   |

                 **********************************

        Enter Your Choice:
```



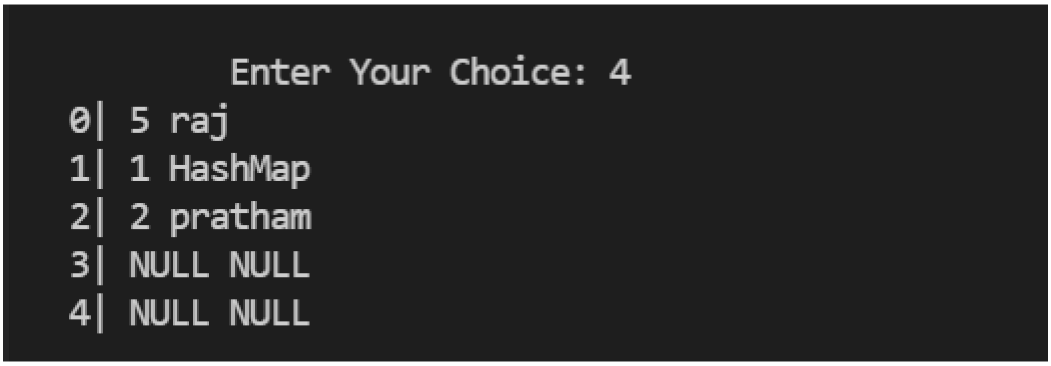
```
        Enter Your Choice: 1


Enter key value: 1 kapil
Key successfully allocated
```



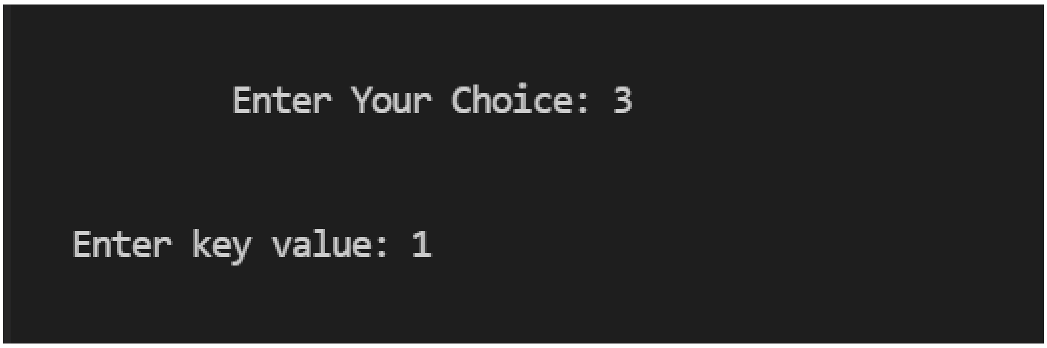
```
        Enter Your Choice: 4
0| 5 raj
1| 1 kapil
2| 2 pratham
3| NULL NULL
4| NULL NULL
```



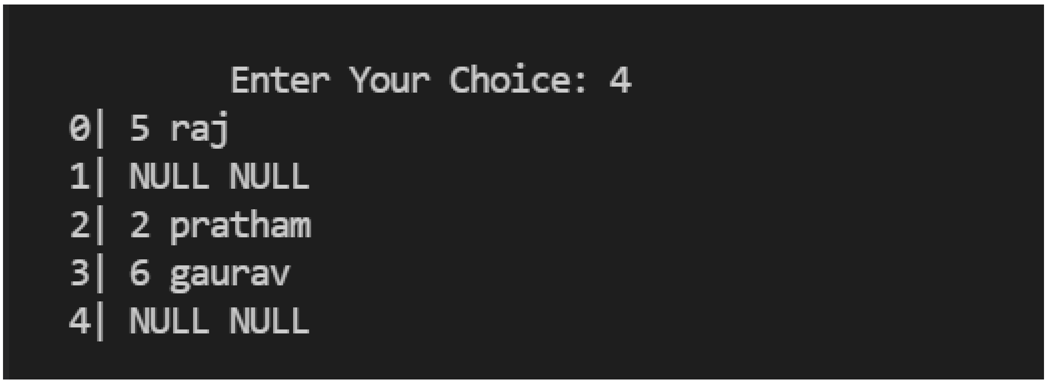
```
        Enter Your Choice: 2


Enter key value: 1
HashMap
```



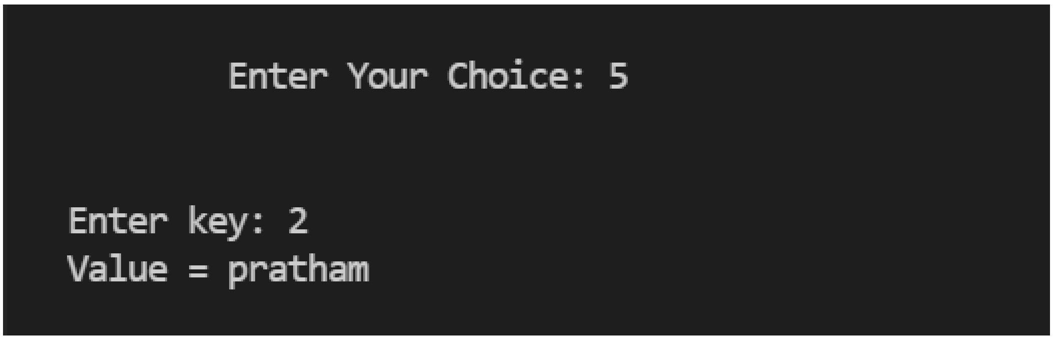
```
        Enter Your Choice: 4
0| 5 raj
1| 1 HashMap
2| 2 pratham
3| NULL NULL
4| NULL NULL
```



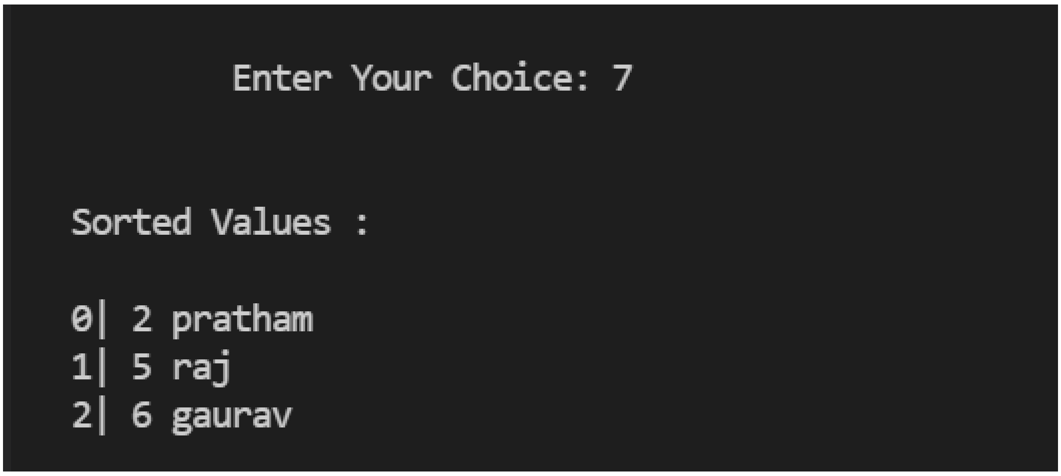
```
        Enter Your Choice: 3


Enter key value: 1
```



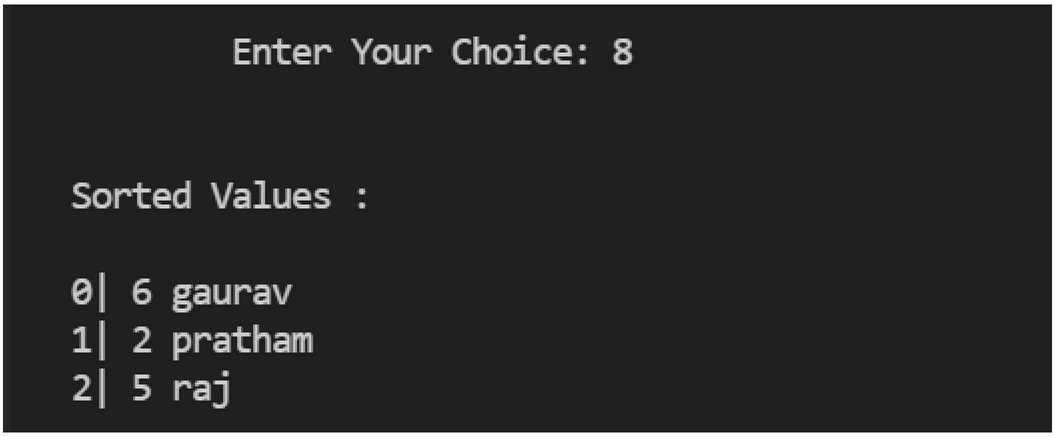
```
        Enter Your Choice: 4
0| 5 raj
1| NULL NULL
2| 2 pratham
3| 6 gaurav
4| NULL NULL
```



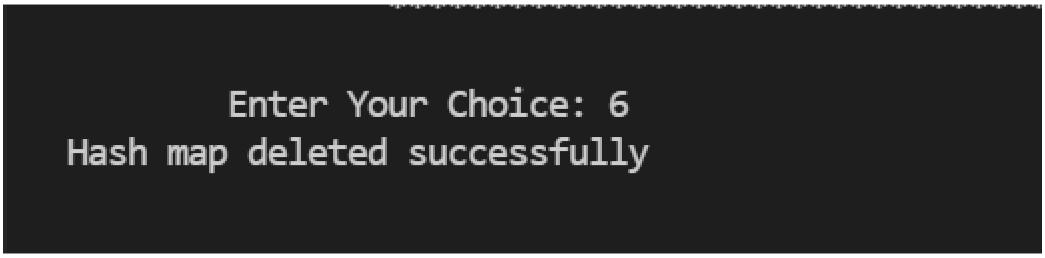
```
        Enter Your Choice: 5


Enter key: 2
Value = pratham
```



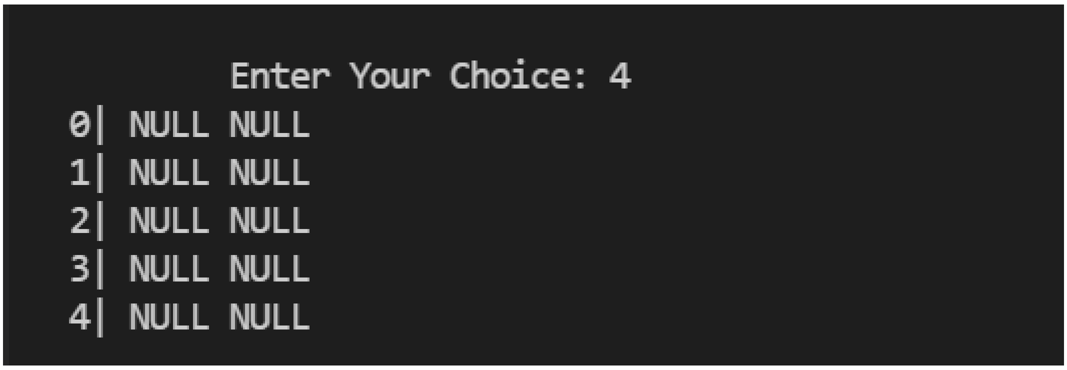
```
        Enter Your Choice: 7


Sorted Values :

0| 2 pratham
1| 5 raj
2| 6 gaurav
```



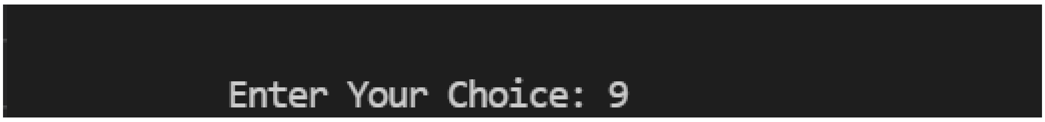
```
        Enter Your Choice: 8


Sorted Values :

0| 6 gaurav
1| 2 pratham
2| 5 raj
```



```
        Enter Your Choice: 6
Hash map deleted successfully
```



```
          Enter Your Choice: 4
  0| NULL NULL
  1| NULL NULL
  2| NULL NULL
  3| NULL NULL
  4| NULL NULL
```



```
          Enter Your Choice: 9
```