// Online C++ compiler to run C++ program online

#include <bits/stdc++.h>

using namespace std;

// template for generic type

template <typename K, typename V>

// Hashnode class

class HashNode {

public:

 V value;

 K key;

 // Constructor of hashnode

 HashNode(K key, V value)

 {

 this->value = value;

 this->key = key;

 }

};

// template for generic type

template <typename K, typename V>

// Our own Hashmap class

class HashMap {

 // hash element array

 HashNode<K, V>\*\* arr;

 int capacity;

 // current size

 int size;

 // dummy node

 HashNode<K, V>\* dummy;

public:

 HashMap()

 {

 // Initial capacity of hash array

 capacity = 20;

 size = 0;

 arr = new HashNode<K, V>\*[capacity];

 // Initialise all elements of array as NULL

 for (int i = 0; i < capacity; i++)

 arr[i] = NULL;

 // dummy node with value and key -1

 dummy = new HashNode<K, V>(-1, -1);

 }

 // This implements hash function to find index

 // for a key

 int hashCode(K key)

 {

 return key % capacity;

 }

 // Function to add key value pair

 void insertNode(K key, V value)

 {

 HashNode<K, V>\* temp = new HashNode<K, V>(key, value);

 // Apply hash function to find index for given key

 int hashIndex = hashCode(key);

 // find next free space

 while (arr[hashIndex] != NULL

 && arr[hashIndex]->key != key

 && arr[hashIndex]->key != -1) {

 hashIndex++;

 hashIndex %= capacity;

 }

 // if new node to be inserted

 // increase the current size

 if (arr[hashIndex] == NULL

 || arr[hashIndex]->key == -1)

 size++;

 arr[hashIndex] = temp;

 }

 // Function to delete a key value pair

 V deleteNode(int key)

 {

 // Apply hash function

 // to find index for given key

 int hashIndex = hashCode(key);

 // finding the node with given key

 while (arr[hashIndex] != NULL) {

 // if node found

 if (arr[hashIndex]->key == key) {

 HashNode<K, V>\* temp = arr[hashIndex];

 // Insert dummy node here for further use

 arr[hashIndex] = dummy;

 // Reduce size

 size--;

 return temp->value;

 }

 hashIndex++;

 hashIndex %= capacity;

 }

 // If not found return null

 return NULL;

 }

 // Function to search the value for a given key

 V get(int key)

 {

 // Apply hash function to find index for given key

 int hashIndex = hashCode(key);

 int counter = 0;

 // finding the node with given key

 while (arr[hashIndex] != NULL) { // int counter =0; // BUG!

 if (counter++ > capacity) // to avoid infinite loop

 return NULL;

 // if node found return its value

 if (arr[hashIndex]->key == key)

 return arr[hashIndex]->value;

 hashIndex++;

 hashIndex %= capacity;

 }

 // If not found return null

 return NULL;

 }

 // Return current size

 int sizeofMap()

 {

 return size;

 }

 // Return true if size is 0

 bool isEmpty()

 {

 return size == 0;

 }

 // Function to display the stored key value pairs

 void display()

 {

 for (int i = 0; i < capacity; i++) {

 if (arr[i] != NULL && arr[i]->key != -1)

 cout << "key = " << arr[i]->key

 << " value = "

 << arr[i]->value << endl;

 }

 }

};

// Driver method to test map class

int main()

{

 HashMap<int, int>\* h = new HashMap<int, int>;

 h->insertNode(1, 1);

 h->insertNode(2, 2);

 h->insertNode(2, 3);

 h->display();

 cout << h->sizeofMap() << endl;

 cout << h->deleteNode(2) << endl;

 cout << h->sizeofMap() << endl;

 cout << h->isEmpty() << endl;

 cout << h->get(2);

 return 0;

}