K. J. Somaiya College of Engineering, Mumbai-77
(A constituent College of Somaiya Vidyavihar University)

Batch: B4	**Roll. No.: 16010122828**
Experiment: 7	
Grade: AA / AB / BB / BC / CC / CD /DD	

Title: Implementation of map using linked list

Objective: To understand map as a data structure

Expected Outcome of Experiment:

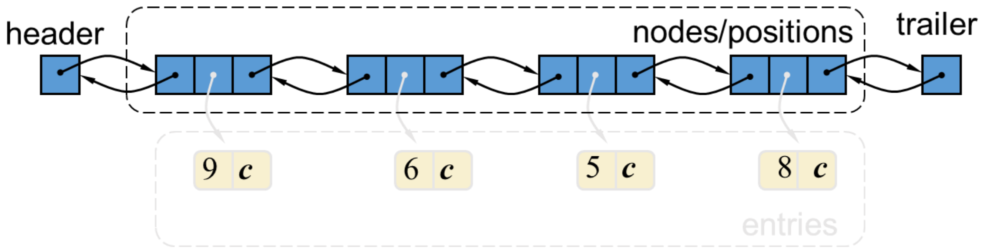
CO	Outcome
CO2	Describe concepts of advanced data structures like set, map & dictionary.

Websites/books referred:

1. Michael T. Goodrich, Roberto Tamassia, and David M. Mount. 2009. Data Structures and Algorithms in C++ (2nd. ed.). Wiley Publishing

Map:

A map allows us to store elements so they can be located quickly using keys. The motivation for such searches is that each element typically stores additional useful information besides its search key, but the only way to get at that information is to use the search key. Specifically, a map stores key-value pairs (k,v), which we call entries, where k is the key and v is its corresponding value. In addition, the map ADT requires that each key be unique, so the association of keys to values defines a mapping. In order to achieve the highest level of generality, we allow both the keys and the values stored in a map to be of any object type. In a map storing student records (such as the student's name, address, and course grades), the key might be the student's ID number. In some applications, the key and the value may be the same.



For example, if we had a map storing prime numbers, we could use each number itself as both a key and its value. In either case, we use a key as a unique identifier that is assigned by an application or user to an associated value object. Thus appropriate in situations where each key is to be viewed as a kind of unique index address for its value, that is, an object that serves as a kind of location for that value. For example, if we wish to store student records, we would probably want to use student ID objects as keys (and disallow two students having the same student ID). In other words, the key associated with an object can be viewed as an "address" for that object. Indeed, maps are sometimes referred to as associative stores or associative containers, because the key associated with an object determines its "location" in the data structure

Algorithm :

createMap() : Creates a map and initialize it to empty map.

Int size(): Return the number of elements in the map.

Boolean empty(): Return true if the map is empty and false otherwise.

Typedef find(int k): Find the entry with key k and return an iterator to it; if no such key exists return end.

Void insert(pair(k,v)): If key k is already present on the Map, replaces the value with v; otherwise inserts the pair (k,v) at the beginning.

Typedef erase(int k): Remove the element with key k and return associated value.

Typedef begin(): Return an iterator to the beginning of the map.

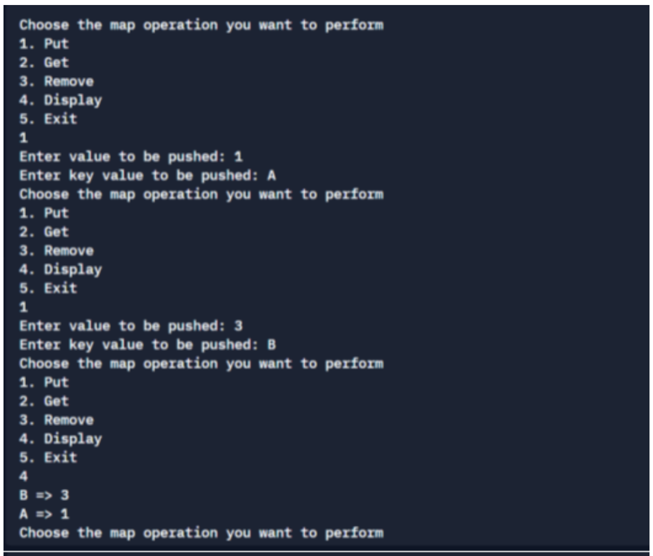
Tyepdef end(): Return an iterator just past the end of the map

Sourcecode and output screenshots:

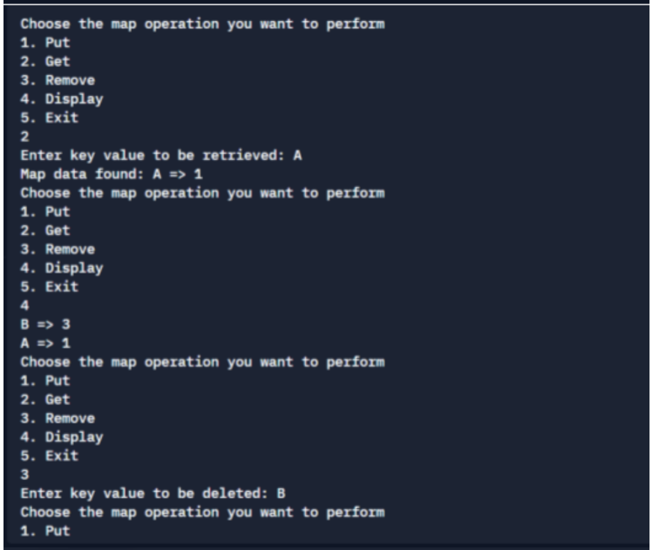
(Students could implement map using singly or doubly linked list. It should handle all possible boundary conditions)

```
#include<bits/stdc++.h>
 using namespace std;
 class Node {
     public:
     Node *prev;
     int data;
     char key;
     Node *next;
     Node(int value, char key1)
     {
         next = nullptr;
         prev = nullptr;
         data = value;
         key = key1;
    }
 };
 class Map
 {
     public:
     Node* head;
     int size;
     Map()
     {
        head = nullptr;
        size = 0;
     }
    bool isEmpty()
    {
        return head == nullptr;
    }
    void put(int value, char key)
    {
        Node* newNode = new Node(value, key);
        Node* current = head;
        while (!isEmpty() && current->next != nullptr && current->key !=
key)
        {
            current = current->next;
        }
        if (isEmpty())
        {
            head = newNode;
        }
        else if (current->key == key && current != nullptr)
        {
            current->data = value;
            size--;
        }
        else
        {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
        size++;

    }
    void remove(char key)
    {
        Node* temp;
        if (isEmpty())
        {
            cout << "Map is Empty!" << endl;
        }
        else if (head->key == key && size == 1)
        {
            temp = head;
            head = nullptr;
            free(temp);
            cout << "Map data was successfully deleted." << endl;
            size--;

        }
        else if (head->key == key)
        {
            temp = head;
            head = head->next;
            free(temp);
        }
        else
        {
            Node* current = head;
            while (current->key != key && current->next != nullptr)
            {
                current = current->next;
            }
            if (current->key == key)
            {
                temp = current;
                if (current->next != nullptr)
                {
                    current->next->prev = current->prev;
                }
                if (current->prev != nullptr)
                {
                    current->prev->next = current->next;
                }
                free(temp);
                size--;
                cout << "Map data was successfully deleted."<< endl;
            }
            else
            {
                cout << "Map data not found!" << endl;
            }
        }
    }
    void get(char key)
    {
        Node* current = head;
        while (current->key != key && current->next != nullptr)
        {
            current = current->next;
        }
        if (current->key == key)
        {
            cout << "Map data found: " << current->key << " => "<<
current->data << endl;

        }
        else
        {
            cout << "Map data not found!" << endl;
        }
    }
    void display()
    {
        Node* currentNode = head;
        if (isEmpty())
        {
            cout << "Empty Map";
        }
        else
        {
            while (currentNode->next != nullptr)
            {
                cout << currentNode->key << " => " << currentNode->data
<< endl;
                currentNode = currentNode->next;
            }
            cout << currentNode->key << " => " << currentNode->data <<
endl;
        }
    }
    int getSize()
    {
        return size;
    }

};
int main()
{
    Map obj;
    bool flag = true;
    int value;
    char key;
    while (flag)
    {
        cout << "Choose the map operation you want to perform\n"; cout <<
"1. Put\n2. Get\n3. Remove\n4. Display\n5. Exit" << endl;
        int choice;
        cin >> choice;
        switch(choice)
        {
            case 1:
                cout << "Enter value to be pushed: ";
                cin >> value;
                cout << "Enter key value to be pushed: ";
                cin >> key;
                obj.put(value, key);
                break;
            case 2:
                cout << "Enter key value to be retrieved: ";
                cin >> key;
                obj.get(key);
                break;
            case 3:
                cout << "Enter key value to be deleted: ";
                cin >> key;
                obj.remove(key);
                break;
            case 4:
                obj.display();
                break;
            case 5:
                flag = false;
                break;
            default:
                cout << "Invalid choice!\n";
        }

    }
    return 0;
}
```

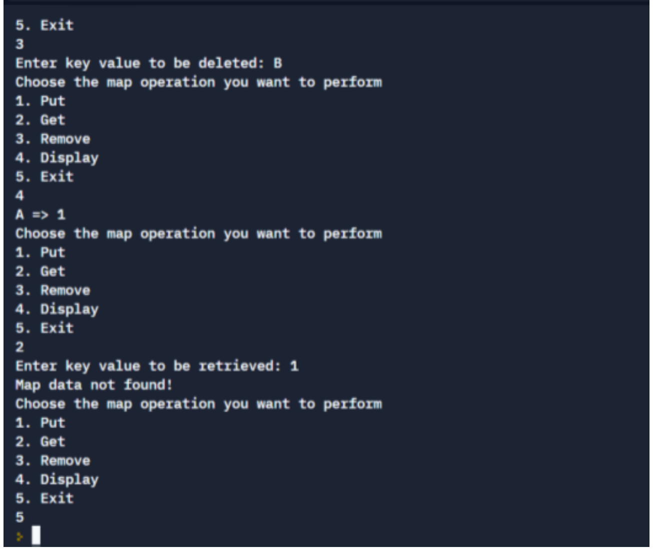
Output:



```
Choose the map operation you want to perform
1. Put
2. Get
3. Remove
4. Display
5. Exit
1
Enter value to be pushed: 1
Enter key value to be pushed: A
Choose the map operation you want to perform
1. Put
2. Get
3. Remove
4. Display
5. Exit
1
Enter value to be pushed: 3
Enter key value to be pushed: B
Choose the map operation you want to perform
1. Put
2. Get
3. Remove
4. Display
5. Exit
4
B => 3
A => 1
Choose the map operation you want to perform
```



```
Choose the map operation you want to perform
1. Put
2. Get
3. Remove
4. Display
5. Exit
2
Enter key value to be retrieved: A
Map data found: A => 1
Choose the map operation you want to perform
1. Put
2. Get
3. Remove
4. Display
5. Exit
4
B => 3
A => 1
Choose the map operation you want to perform
1. Put
2. Get
3. Remove
4. Display
5. Exit
3
Enter key value to be deleted: B
Choose the map operation you want to perform
1. Put
```



```
5. Exit
3
Enter key value to be deleted: B
Choose the map operation you want to perform
1. Put
2. Get
3. Remove
4. Display
5. Exit
4
A => 1
Choose the map operation you want to perform
1. Put
2. Get
3. Remove
4. Display
5. Exit
2
Enter key value to be retrieved: 1
Map data not found!
Choose the map operation you want to perform
1. Put
2. Get
3. Remove
4. Display
5. Exit
5
>
```

Postlab questions:

1. Give and explain at least one scenario each in which map, dictionary and set would be the most appropriate data structure to use.

Map:

Map data type is ideal to use in look-up type situations where there is an identifying value and an actual value that is represented by the identifying value. A few examples where the map data type can be used are:

- Student ID numbers and last names
- House numbers on a street and the number of pets in each house
- Postal codes and the names of cities
- Driving licenses and last names

Dictionary:

Dictionary data structure can be used in contact books, where the key can be the name of the person and the corresponding value can be their telephonic number.

Set:

One of the most well-known applications is its use in Kruskal's Minimal Spanning Tree algorithm, which has a lot of applications in many ways. Using a graph with random weights, Kruskal's algorithm generates random trees, and in particular, this can be used to easily generate mazes of all kinds.

2. Write map as ADT

Ans:

A map consists of (key, value) pairs
• Each key may occur only once in the map
• Values are retrieved from the map via the key
• Values may be modified
• Key, value pairs may be removed

```
/*Value definition*/
Abstract typedef <int value, char key,  NODE* next> NODE

/*Operator definition: CREATE*/
Abstract void createNode(value, key)
Precondition : none
Postcondition: none
```

/*Operator definition: PUT*/
Abstract void put(value, key);
Precondition : NODE->key should not be present, if present then update value
NODE->value = value
Postcondition: none

/*Operator definition: GET*/
Abstract void get(key);
Precondition : none
Postcondition: none

/*Operator definition: REMOVE*/
Abstract void remove(key);
Precondition : temp != NULL
Postcondition: none

Conclusion:

Through this experiment we learnt about the concept of maps and learnt to implement it using single linked list in C++.