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

Batch: B2	**Roll. No.:** 110
Experiment: Map	
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:

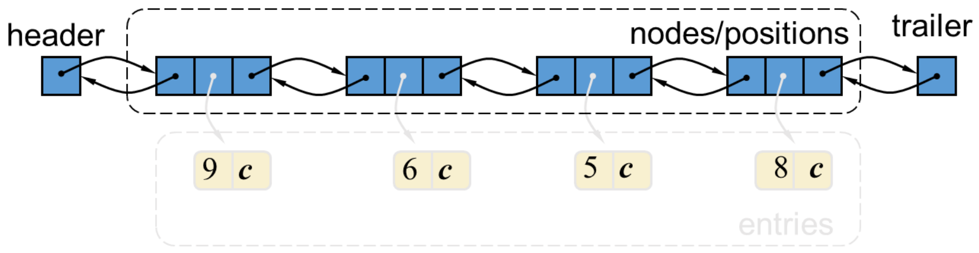
1. Michael T. Goodrich, Roberto Tamassia, and David M. Mount. 2009. Data Structures and Algorithms in C++ (2nd. ed.). Wiley Publishing

2.https://tildesites.bowdoin.edu/~ltoma/teaching/cs210/fall09/Slides/Maps.pdf

3. Geeks for Geeks

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, a map is most 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<map>

#include <iostream>


using namespace std;


int main()

{

   map<int, int> mymap;

   cout<<"\n Checking if map is empty......."<<mymap.empty();

   mymap.insert({1,2});

   cout<<"\n Inserting 1,2 into the map.......";

   mymap.insert({3,2});

   cout<<"\n Inserting 3,2 into the map.......";

   mymap.insert({3,5});

   cout<<"\n Inserting 3,5 into the map.......(wont work)";

   mymap.insert({5,3});
```

```
    cout<<"\n Inserting 5,3 into the map.......(wont work)";

    cout<<"\n Iterating through the map and printing each value.......\n";

    for(auto i=mymap.begin();i!=mymap.end();i++){

       cout<<"("<<i->first<<i->second<<")"<<"\n";

    }

    mymap.erase(3);

    cout<<"\n Erasing 3 key from the map.......";

    cout<<"\n Iterating through the map and printing each value.......\n";

    for(auto i=mymap.begin();i!=mymap.end();i++){

       cout<<"("<<i->first<<i->second<<")"<<"\n";

    }

    mymap.clear();

    cout<<"\n Clearing map.......";

    cout<<"\n Iterating through the map and printing each value(empty).......\n";

    for(auto i=mymap.begin();i!=mymap.end();i++){

       cout<<"("<<i->first<<i->second<<")"<<"\n";

    }




    return 0;

}
```

```
Checking if map is empty.......1

Inserting 1,2 into the map.......

Inserting 3,2 into the map.......

Inserting 3,5 into the map.......(wont work)

Inserting 5,3 into the map.......(wont work)

Iterating through the map and printing each value.......

(12)

(32)

(53)


Erasing 3 key from the map.......
```

Iterating through the map and printing each value.......

(12)

(53)

Inserting 3,2 into the map.......

Iterating through the map and printing each value.......

Conclusion: Thus we have implemented map using STL of C++. The Stl library of C++ provides inbuilt functions for the implementation of data structures like map, and dictionaries. Map is a data structure in which the key value pairs are stored. the keys cannot be repeated.

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.

Set - counting only unique pairs. (example while removing repetition)
Map- storing roll numbers and student names
Dictionary - Storing project numbers and projects assigned to students.

2. Write map as ADT

Value definition: Map is a collection of key value entries, with each value associated with a distinct key.
• Assumption: map provides a special pointer object, which permits us to reference entries of the map, called position.
• Iterator references entries and navigate around the map.
• Given a map iterator p, the associated entry may be accessed by dereferencing the iterator, *p.
• The individual key and value can be accessed using p->key() and p->value(), respectively.

• Can be implemented using associative arrays
Map ADT
• size()
• isEmpty()
• get(k):
• if M contains an entry with key k, return it; else return null
• put(k,v):
• if M does not have an entry with key k, add entry (k,v) and return null
• else replace existing value of entry with v and return the old value
• remove(k):
• remove entry (k,*) from M
(k,v) key=integer, value=letter