**Batch: B3 Roll. No.: 121**

**Experiment: 9**

**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

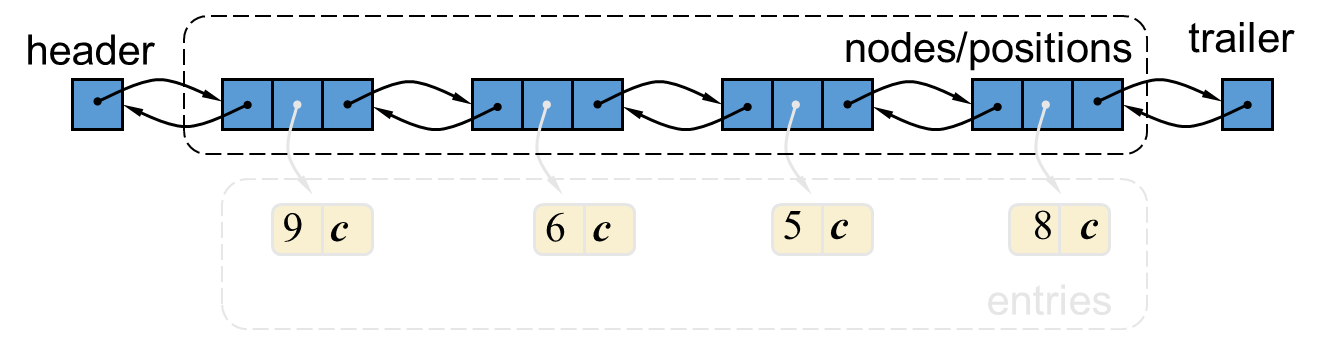
**2.** Aaron M. Tenenbaum, Yedidyah Langsam, and Moshe J. Augenstein. Data structures using C.

**3.** [**https://classroom.google.com/u/0/c/NTM5NDM0NzUxMTQ2**](https://classroom.google.com/u/0/c/NTM5NDM0NzUxMTQ2)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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)*

**Code 1 (*use of begin() function)***

#include<iostream>

#include<map>

using namespace std;

int main()

{

//declaration of map container

map<char, int> mymap;

mymap['a'] = 1;

mymap['b'] = 2;

mymap['c'] = 3;

//using begin() to print map

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

{

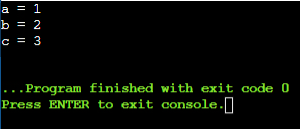
cout<<it->first<<" = "<<it->second<<"\n";

}

return 0;

}

**Output:**



**Code 2: *(use of insert() function)***

//C++ program to illustrate map::insert({key, element})

#include<bits/stdc++.h>

using namespace std;

int main(){

//initialize container

map<int, int> mp;

//insert elements in random order

mp.insert({2, 30});

mp.insert({1, 40});

mp.insert({3, 60});

//does not insert key 2 with element 20

mp.insert({2, 20});

mp.insert({5, 50});

//prints the elements

cout<<"Key\tElement\n";

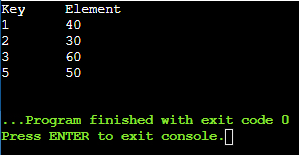
for(auto itr = mp.begin(); itr != mp.end(); itr++){

cout<<itr->first<<"\t"<<itr->second<<"\n";

}

return 0;

}



**Code 3: *(use of clear() function)***

#include<bits/stdc++.h>

using namespace std;

int main(){

map<int, string> map1, map2; //Take any two maps

//inserting values

map1[1] = "India";

map1[2] = "Nepal";

map1[3] = "Sri Lanka";

map1[4] = "Myanmar";

cout<<"Map size before running function:\n"; //Print the size of map

cout<<"map1 size = "<<map1.size()<<endl;

cout<<"map2 size = "<<map2.size()<<endl;

//Deleting the map elements

map1.clear();

map2.clear();

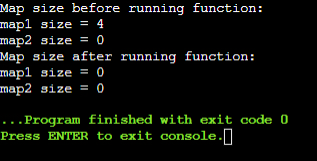
cout<<"Map size after running function:\n"; //Print the size of map

cout<<"map1 size = "<<map1.size()<<endl;

cout<<"map2 size = "<<map2.size();

return 0;

}



**Code 4: *(using erase() function)***

#include<bits/stdc++.h>

using namespace std;

int main()

{

//initialize container

map<int, int> mp;

//insert elements in random order

mp.insert({2, 30});

mp.insert({1, 40});

mp.insert({3, 60});

mp.insert({5, 50});

//print the elements

cout<<"The map before using erase() is -\n";

cout<<"Key\tElement\n";

for(auto itr = mp.begin(); itr != mp.end(); itr++)

{

cout<<itr->first<<"\t"<<itr->second<<"\n";

}

//function to erase keys

mp.erase(1);

mp.erase(2);

cout<<"The map after applying erase() is -\n";

cout<<"Key\tElement\n";

for(auto itr = mp.begin(); itr != mp.end(); itr++)

{

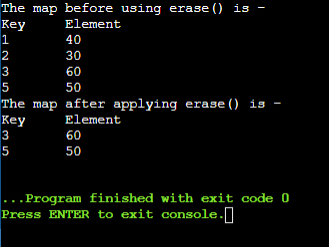
cout<<itr->first<<"\t"<<itr->second<<"\n";

}

return 0;

}

**Output:**



**Code for Assignment:**

**Output:**

#include<bits/stdc++.h>

using namespace std;

int main()

{

map<string, float> rainfall\_data;

int i, n, value, flag = 0;

string key, find\_key;

cout<<"Enter the number of records: ";

cin>>n;

for(i = 1; i <= n; i++)

{

fflush(stdin);

cout<<"\nEnter Key for record "<<i<<": ";

cin>>key;

if(i!=1)

cin>>key;

fflush(stdin);

cout<<"\nEnter Value for record "<<i<<": ";

cin>>value;

rainfall\_data.insert({key, value});

}

fflush(stdin);

cout<<"KEYS\tVALUES\n";

for(auto itr = rainfall\_data.begin(); itr != rainfall\_data.end(); itr++)

{

cout<<itr->first<<"\t"<<itr->second<<"\n";

}

cout<<"Enter Key whose Value is to be found: ";

cin>>find\_key;

cin>>find\_key;

for(auto itr = rainfall\_data.begin(); itr != rainfall\_data.end(); itr++)

{

if(find\_key.compare(itr->first) == 0)

{

cout<<"Value "<<itr->second<<" is found.";

flag++;

}

else

continue;

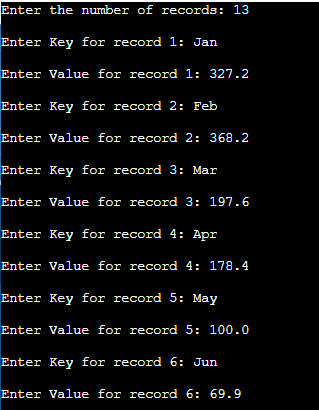
}

if(flag == 0)

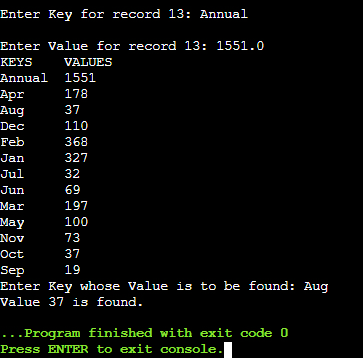
cout<<"Value for the given key was not found.";

return 0;

}







**Conclusion:** Thus, in this experiment, the concept of Maps in C++ has been learnt and implemented. It is good for those situations where redundancy of data is not good, for e.g., passwords of employees in a record, etc. However, it does become a problem when there is redundancy, for e.g., names of employees in a record. So the appropriateness of using Maps depends on the situation. Maps also offer an advantage of having keys which can be of any basic data type, and having values which can themselves be Maps. So, there are a wide range of applications for Maps.

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.

Ans.

* 1. 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. For example, Student ID numbers and last names.
  2. The dictionary data structure can store a large set of values for a specific key. Hence it is widely used to store a heterogeneous set of data for a key. For example, all the data for an employee can be stored using a dictionary.
  3. The set data structure is used to test whether elements belong to a set of values. It is used in pattern matching and other applications.

1. Write map as ADT

Ans. 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 one 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.

It can be implemented using associative arrays.

size(): Return the number of entries in M.

empty(): Returns true if M is empty and false otherwise.

find(k): If M contains an entry e = (k, v), with key equal to k, then return an iterator p referring to this entry, and otherwise return the special iterator end.

put(k, v): If M does not have an entry with key equal to k, then add entry (k, v) to M and otherwise, replace the value field of this entry with v; return an indicator to the inserted/modified entry.

erase(k): Remove from M the entry with key equal to k; an error condition occurs if M has no such entry.

erase(p): Remove from M the entry referenced by the iterator p; an error condition occurs if p points to the end sentinel.

begin(): Return an iterator to the first entry of M.

end(): return an iterator to a position just beyond the end of M.