Non- Linear Data Structures-MAP

Maps

- A **Map** models a searchable collection of key-value entries
- The main operations of a map are for searching, inserting, and deleting items
- Multiple entries with the same key are **not** allowed
- Applications:
  - address book
  - student-record database

MAP

- A map is any data structure that groups a dynamic number of *key-value pairs* together,

- Map allows us to
  - retrieve values by key,
  - to insert new key-value pairs, and
  - to update the values associated with keys.

Courtesy: https://noncombatant.org/2014/03/02/maps-and-their-applications/

MAP

- A *Map* is a type of fast key lookup data structure that offers a flexible means of indexing into its individual elements.
- Unlike most **array data structures** that
  - **only allow access to the elements by means of integer** indices,
  - **the indices for a Map can be nearly any scalar numeric value or a character vector.**

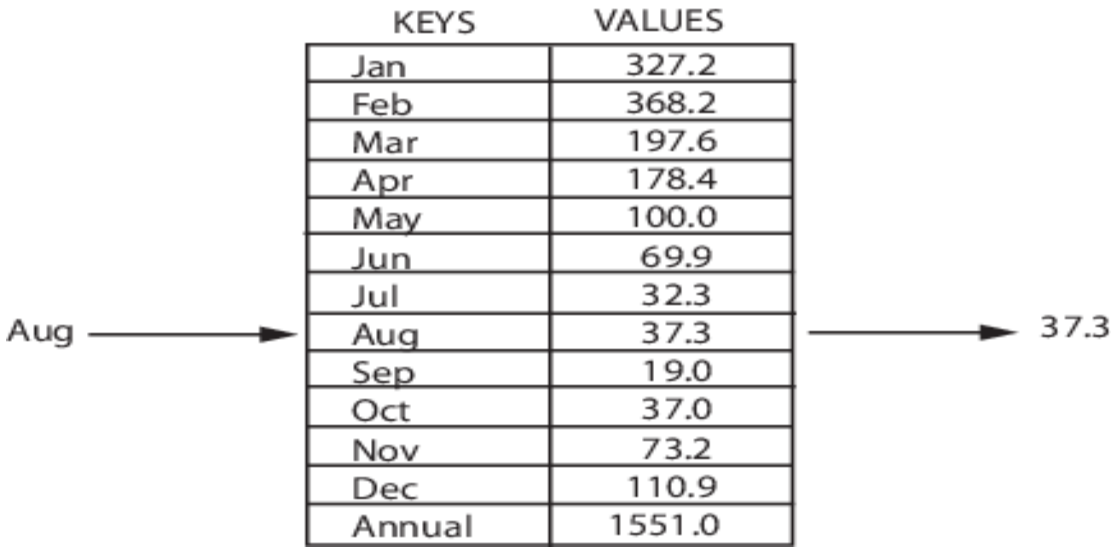
Courtesy: https://in.mathworks.com/help/matlab/matlab_prog/overview-of-the-map-data-structure.html

MAP

- Indices into the elements of a Map are called *keys*.
- These keys, along with the data *values* associated with them, are stored within the Map.
- **Each entry of a Map contains exactly one unique key and its corresponding value.**
- No two mapped values can have same key values.
- **A map cannot contain duplicate keys**

MAP-Example

- **Indexing into the Map of rainfall statistics shown below with a character vector representing the month of August yields the value internally associated with that month, 37.3.**
- **Mean monthly rainfall statistics (mm)**



MAP

- **Keys are not restricted to integers as they are with other arrays.**
- Specifically, a key may be any of the following types:
  - 1-by-N character array
  - **Scalar real double** or single
  - **Signed or unsigned scalar integer**

MAP

- The values stored in a Map can be of any type.
- This includes
  - **arrays of numeric values,**
  - **structures,**
  - **cells,**
  - **character arrays,**
  - objects, or
  - **other Maps.**

Standard Template Library (STL)

- The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc.
- It is a library of container classes, algorithms, and iterators.

https://www.geeksforgeeks.org/the-c-standard-template-library-stl/

MAP

- Maps are part of the C++ STL (Standard Template Library).
- A C++ map is a way to store a key-value pair.

MAP ADT Functions

Some basic functions associated with Map:

- begin() – Returns an iterator to the first element in the map
- end() – Returns an iterator to the theoretical element that follows last element in the map
- size() – Returns the number of elements in the map
- max_size() – Returns the maximum number of elements that the map can hold
- empty() – Returns whether the map is empty
- pair insert(keyvalue, mapvalue) – Adds a new element to the map
- erase(iterator position) – Removes the element at the position pointed by the iterator
- clear() – Removes all the elements from the map

Courtesy:https://www.geeksforgeeks.org/map-associative-containers-the-c-standard-template-library-stl/

begin() function

- Used to return an iterator pointing to the first element of the map container.
- begin() function returns a bidirectional iterator to the first element of the container.

Syntax :

***mapname*.begin()**

- **Parameters :** No parameters are passed.
- **Returns :** This function returns a bidirectional iterator pointing to the first element.

Demonstrates begin() and end()

```
#include <iostream>
#include <map>
using namespace std;

int main()
{
   // declaration of map container named mymap
   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:

a = 1

b = 2

c = 3

Standard Containers

- A container is a holder object that stores a collection of other objects (its elements).

  The container manages the storage space for its elements and provides member functions to access them, either directly or through iterators (reference objects with similar properties to pointers).

https://cplusplus.com/reference/iolibrary/

end() function

- end() function is used to return an iterator pointing to past the last element of the map container.
- Since it does not refer to a valid element, it cannot de-referenced end() function returns a bidirectional iterator.

Syntax :

***mapname*.end()**

- **Parameters :** No parameters are passed.
- **Returns :** This function returns a bidirectional iterator pointing to the next of last element.

insert()

- A built-in function in C++ STL which is used to insert elements with a particular key in the map container.

Syntax:

- iterator map_name.insert({key, element})

Parameters:

- The function **accepts a pair that consists of a key and element** which is to be inserted into the map container.
- The function does not insert the key and element in the map if the key already exists in the map.

Return Value:

- The function returns an iterator pointing to the new element in the container.

bits/stdc++.h

- It is basically a header file that includes every standard library. In programming contests, Using this file is a good idea, when you want to reduce the time wasted in doing chores; especially when your rank is time sensitive.

- In programming contests, people do focus more on finding the algorithm to solve a problem than on software engineering.
  - From, software engineering perspective, it is a good idea to minimize the include.
  - If you use it actually includes a lot of files, which your program may not need, thus increases both compile time and program size unnecessarily.

insert()

```
// 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 inserts 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;
}
```

OUTPUT-

KEY ELEMENT

1 40

2 30

3 60

5 50

size() function

- In C++, **size()** function is used to return the total number of elements present in the map.

Syntax:

- **map_name.size()**

Return Value: It returns the number of elements present in the map.

size() function

```
Input : map1 = {
                {1, "India"},
                {2, "Nepal"},
                {3, "Sri Lanka"},
                {4, "Myanmar"}
               }
        map1.size();
```

Output: 4

```
Input : map2 = {};
        map2.size();
```

Output: 0

clear()

- clear() function is used to remove all the elements from the map container and thus leaving it’s size 0.

Syntax:

map1.clear() where map1 is the name of the map.

Parameters:

No parameters are passed.

Return Value:

None

clear()

Input : map1 = {
{1, "India"},
{2, "Nepal"},
{3, "Sri Lanka"},
{4, "Myanmar"}
}

```
map1.clear();
```

Output: map1 = {}

clear()

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

int main()
{
   // Take any two maps
  // initialize container

   map<int, string> map1, map2;

   // Inserting values
   map1[1] = "India";
   map1[2] = "Nepal";
   map1[3] = "Sri Lanka";
   map1[4] = "Myanmar";

  // Print the size of map
   cout<< "Map size before running function: \n";
   cout << "map1 size = " << map1.size() << endl;
   cout << "map2 size = " << map2.size() << endl;;

      // Deleting the map elements
   map1.clear();
   map2.clear();

   // Print the size of map
   cout<< "Map size after running function: \n";
   cout << "map1 size = " << map1.size() << endl;
   cout << "map2 size = " << map2.size();
   return 0;
}
```

Output:

```
Map size before running function:
map1 size = 4
map2 size = 0

Map size after running function:
map1 size = 0
map2 size = 0
```

erase()

- A built-in function in C++ STL which is used to erase element from the container.
- It can be used to **erase keys, elements** at any specified position or a given range.

Syntax :

- map_name.erase(key)

Parameters:

- The function accepts **one mandatory parameter *key*** which specifies the key to be erased in the map container.

Return Value:

- The function **returns 1 if the key element is found** in the map else returns 0.

erase()

```
#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 });

// prints 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 given keys
   mp.erase(1);
   mp.erase(2);

   // prints the elements
   cout << "\nThe 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;
}
```

erase()

The map before using erase() is :

KEY ELEMENT

1 40

2 30

3 60

5 50

The map after applying erase() is :

KEY ELEMENT

3 60

5 50

empty()

- Used to check if the map container is empty or not.

Syntax :

- ***mapname*.empty()**

Parameters :

- No parameters are passed.

Returns :

- True, if map is empty
- False, Otherwise

empty()

Examples:

Input : map

```
mymap['a']=10;
mymap['b']=20;
mymap.empty();
```

Output : False

Map ADT

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

Example

Operation	*Output*	*Map*
empty()	true	$\emptyset$
put$(5,A)$	$p_1 : [(5,A)]$	$\{(5,A)\}$
put$(7,B)$	$p_2 : [(7,B)]$	$\{(5,A),(7,B)\}$
put$(2,C)$	$p_3 : [(2,C)]$	$\{(5,A),(7,B),(2,C)\}$
put$(2,E)$	$p_3 : [(2,E)]$	$\{(5,A),(7,B),(2,E)\}$
find(7)	$p_2 : [(7,B)]$	$\{(5,A),(7,B),(2,E)\}$
find(4)	end	$\{(5,A),(7,B),(2,E)\}$
find(2)	$p_3 : [(2,E)]$	$\{(5,A),(7,B),(2,E)\}$
size()	3	$\{(5,A),(7,B),(2,E)\}$
erase(5)	–	$\{(7,B),(2,E)\}$
erase(p_3)	–	$\{(7,B)\}$
find(2)	end	$\{(7,B)\}$