Section 23 : STL

15 April 2022 16:49

© Rajat Kumar
https://www.linkedin.com/in/imRajat/
https://github.com/im-Rajat

Section 23 : STL

STL :

- Standard Template Library.
- STL i.e. building classes in C++.

Data Structure :

- Data Structure is a collection of data and the arrangement of their data for its efficient utilization.
- Depending on our utilization, we can arrange the data so that it can be utilized efficiently.
- Efficiency in terms of time and space. So we want the data to be stored and retrieved easily and also occupy less space.

Types of Data Structure :

- Array : Problem is size is fixed.
- Singly Linked List (Only Forward Pointer) : Size is variable.
- Doubly Linked List (Forward and Backward Pointers)
- Stack
- Queue
- Deque
- Priority Queue
- Map
- Set

- C++ provides built-in library of classes for all of these things and that is a collection of classes called as STL.

STL - Standard Template Library has :

- Algorithms :
    - Built-in algorithms/functions that are meant for managing container.
    - Performing operations on the containers.
    - Example :
        - Search()
        - Sort()
        - Binary_search()
        - Reverse()
        - Concat()
        - Copy()
        - Union()
        - Intersection()
        - Merge()
        - Heap()
- **Containers** :
    - Array, List, Stack, Queue, etc.
    - Containers contain collection of data, list of data.
    - Available containers (All these are template classes, generic, can work for any type of data) :
        - **Vector** : Like array but size is not fixed.
            - Functions available :
                - Push_back()
                - Pop_back()
                - Insert() // can mention the index and we can insert at that place.
                - Remove()
                - Size()
                - Empty()
        - **List** : Doubly linked list.
            - Functions available :
                - Same as vector + additional like insertion and deletion is possible from both the ends.
                - Push_front()
                - Pop_front()
                - Front()
                - Back()

        - **Forward_list** : Singly linked list (Introduce in C++ 11)
            - Functions available :
                - Same as List.
                - But push_back is not available as it's singly linked list.
        - **Deque** : Double ended Queue. Same as vector, it's an array only, but we can insert from both the ends (front and back)
            - Functions available:
                - Same as List.

          List, forward_list, deque have same set of functions, only in vector we can't insert or delete from front.
        - **Priority_queue** : For Heap Data Structure, MAX heap - whenever we pop() the largest element will be deleted. Deleting always maximum element.
            - Functions available :
                - Push()
                - Pop()
                - Empty()
                - Size()
        - **Stack** : LIFO, Last In First Out
            - Functions available :
                - Same as priority_queue
        - **Set** : Collection of elements which will contain only unique elements. Duplicates are not allowed. Order not maintained.
            - Functions available :
                - Insert()
        - **Mutiset** : same as set but allows duplicate.
        - **Map** : used for storing key-value pair. It uses hash table. It contains unique keys.
        - **MultiMap** : same as map but keys can be duplicate. But same key value pair should not be there.
        - **Queue :** FIFO, First In First Out
            - Functions available :
                - Empty()
                - Size()
                - Swap() - Exchange the contents of two queues but the queues must be of the same type, although sizes may differ.
                - Front()
                - Back()
                - Push(x) - Adds the element 'x' at the end of the queue.
                - Pop() - Deletes the first element of the queue.
                - Emplace() - Insert a new element into the queue container, the new element is added to the end of the queue.
- Iterators :
    - There are iterators for iterating through the collection of values.
    - For accessing containers, iterators are available.

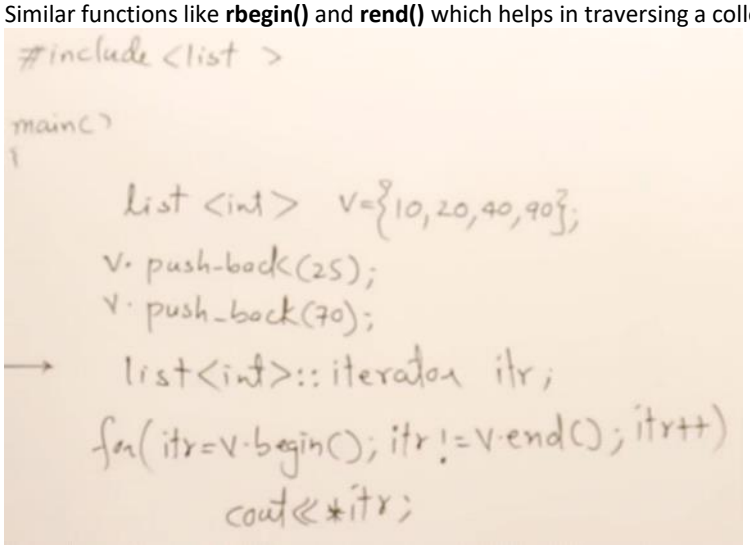
How to use them :

- Include Header file : **#include<vector>**
- Create object : **vector<int> v**;
    - Can also mention size :
        - vector<int> v(size);
    - Can also mention initial values :
        - vector<int> v = {10, 20, 30, 40};
    - Insertion :
        - v.push_back(50); // will insert at the end.
    - Deletion :
        - v.pop_back(); // will delete last element.

How to iterate/access them :

- Using for each loop introduce in C++ 11.
    - For (int x : v) { cout<<x; }
- Using iterator classes.
    - **Vector<int>::iterator itr = v.begin()**;
    - Or vector<int>::iterator itr;
    - For (itr = v.begin(); itr!=v.end(); itr++) { cout<<*itr; }
        - Need to use *, because iterator is like a pointer to the element inside the collection (here vector). Need to dereference.
- Iterators are available in every collection.
- **begin()** and **end()** are the functions that are available in all containers.

- Similar functions like **rbegin()** and **rend()** which helps in traversing a collection from the rare end so reverse traversing is possible.



- Using iterators, We can modify the values also

```
cout<<"Using Iterator: ";
vector<int>::iterator itr;
for (itr = v.begin(); itr != v.end(); itr++) {
    cout<<++*itr<<" ";
}
```

- The value of elements in vector will also increase by one, because we are using increment operator **++*itr**.
- So this becomes a very powerful feature of C++. This is available in Java also. And those sort of classes are called as collection framework.

Map in STL :

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


int main()
{
    map<int, string> m;

    m.insert(pair<int, string>(1, "Ravi"));
    m.insert(pair<int, string>(2, "John"));
    m.insert(pair<int, string>(3, "Rock"));

    map<int, string>::iterator itr;

    for (itr = m.begin(); itr != m.end(); itr++) {
        cout<<itr->first<<" "<<itr->second<<endl;
    }

    map<int, string>::iterator itr2;
    itr2 = m.find(3);
    cout<<"Value found is: "<<itr2->first<<" "<<itr2->second<<endl;

    cout<<m[2]<<endl;



    return 0;
}
```

- From where does the insertion and deletion of elements get accomplished in Queues?
  - Rear for insertion & Front for deletion.
- Which among the below mentioned entities is / are essential for an Array Representation of a Queue?
  - An array to hold queue elements.
  - A variable to hold the index of front element.
  - A variable to hold the index of rear element.
- What is the ' next ' field of structure node in the Queue?
  - Results into the storage of address of next node by holding the next element of queue.
- Which among the below mentioned assertions is / are mainly associated with the feature of Spooling?
  - Maintenance of a queue of jobs to be printed.
- Where is the root directory of a disk placed?
  - At a fixed location on the system disk.