Data Structures (15B11CI311)

Odd Semester 2020

3rd Semester , Computer Science and Engineering

Jaypee Institute Of Information Technology (JIIT), Noida

Outline

- Standard Template Library (STL)
- Vector and its functionality

STL (Standard Template Library)

- 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.
- It is a generalized library and so, its components are parameterized.
- A working knowledge of template classes is a prerequisite for working with STL.
- STL has three components
  - ✔ Containers
  - ✔ Algorithms
  - ✔ Iterators

STL (Standard Template Library)

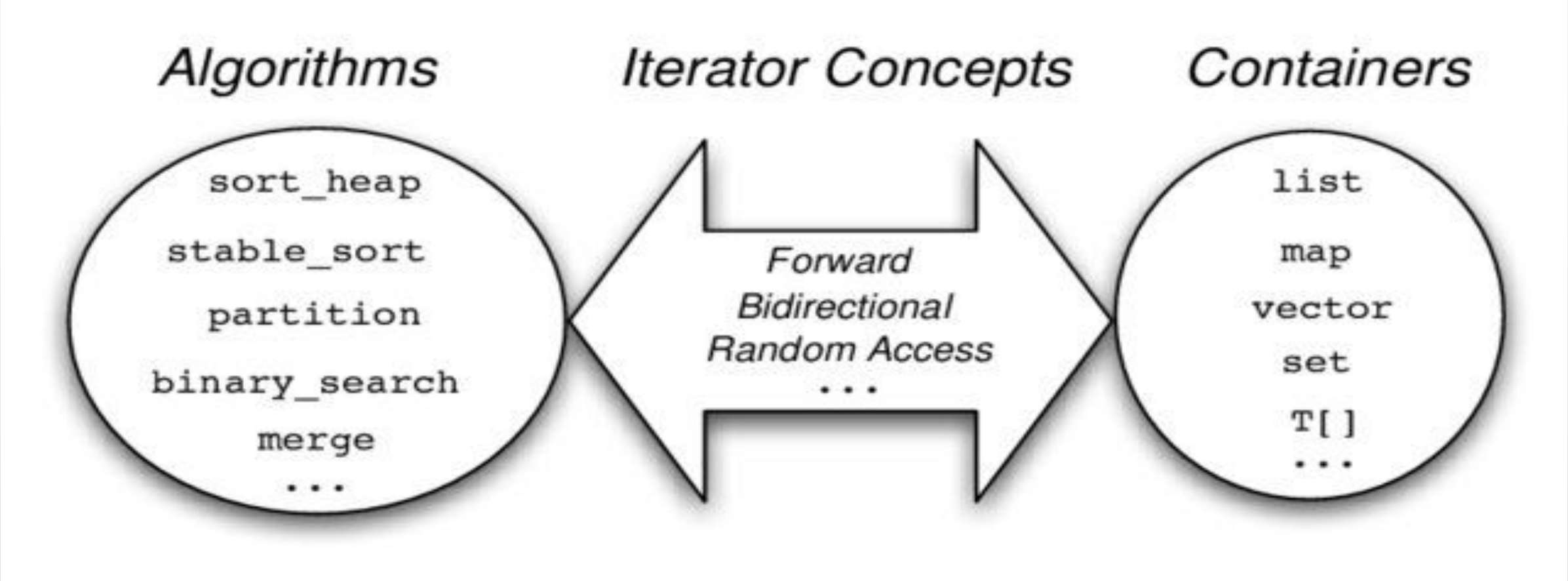
- These three components work together with one another in synergy to provide support to a variety of programming solutions.
- Algorithm employ iterators to perform operation stored in containers.

- **Container**
  - ✔ An object that stores data in memory into an organized fashion.
  - ✔ The containers in STL are implemented by template classes and therefore can be easily modified and customized to hold different types of data.

STL (Standard Template Library)

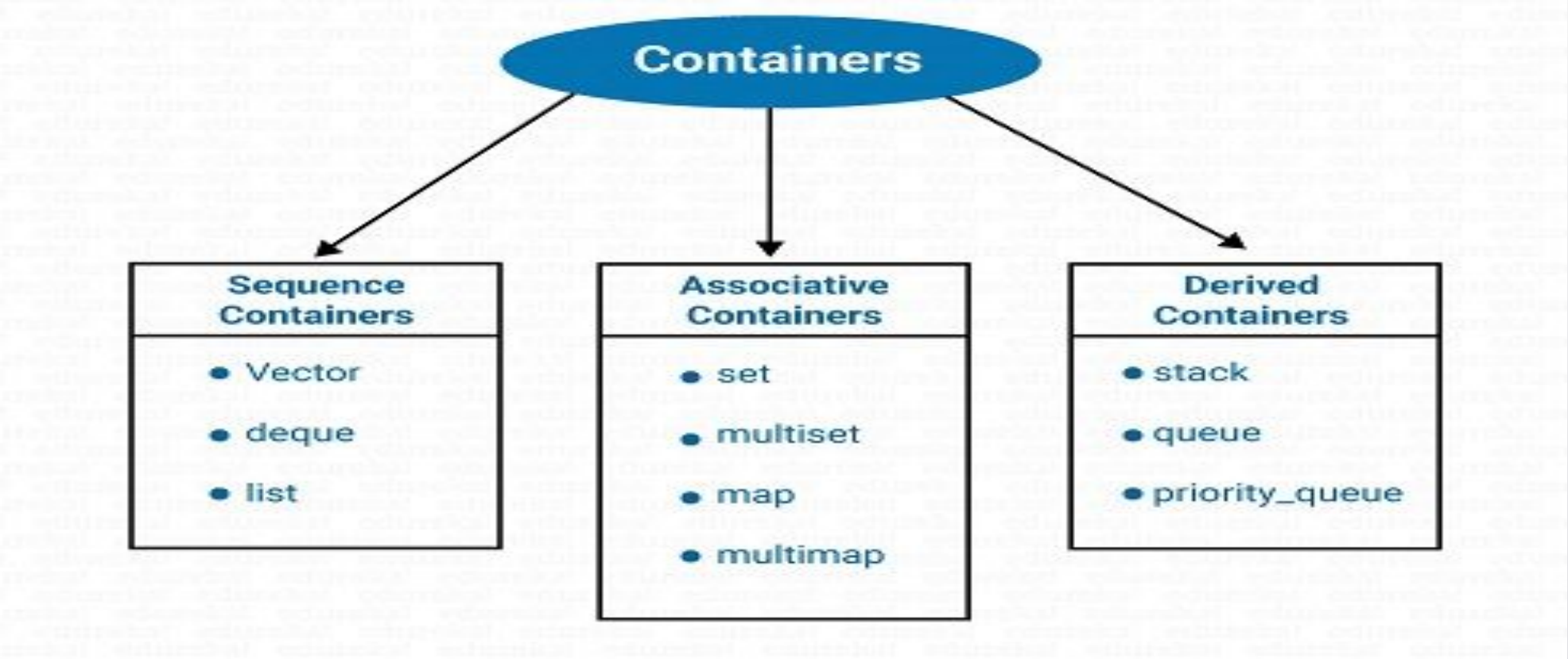
- **Procedure (Algorithms)**
  - ✔ used to process the data contained in the containers is defined as an algorithm.
  - ✔ The STL includes many different kinds of algorithms to provide support to tasks such as initializing, searching, copying, sorting, and merging, copying, sorting, and merging.
  - ✔ Algorithms are implemented by template functions.
- **Iterator**
  - ✔ can be defined as an object that points to an element in a container.
  - ✔ Iterators can be used to move through the contents of containers.
  - ✔ Iterators are handled just like pointers.
  - ✔ We can increment or decrement them.
  - ✔ Iterators connect algorithm with containers and play a key role in the manipulation of data stored in the containers.

STL (Standard Template Library)



Containers

- STL defines ten containers which are grouped into three categories.



Containers

Container : Vector

- ✔ Description: It can be defined as a dynamic array. It permits direct access to any element.
- ✔ Header File: <vector>
- ✔ Iterator: Random access

Container : List

- ✔ Description: It is a bidirectional linear list. It allows insertion and deletion anywhere
- ✔ Header File: <list>
- ✔ Iterator: Bidirectional

Containers

Container : deque

- ✔ Description: It is a double-ended queue. Allows insertions and deletions at both the ends. Permits direct access to any element.
- ✔ Header File: <deque>
- ✔ Iterator: Random access

Container : set

- ✔ Description: It is an associate container for storing unique sets. Allows rapid lookup.
- ✔ Header File: <set>
- ✔ Iterator:Bidirectional

Containers

Container : multiset

- ✔ Description: It is an associate container for storing non-unique sets.
- ✔ Header File: <set>
- ✔ Iterator: Bidirectional

Container : map

- ✔ Description: It is an associate container for storing unique key/value pairs. Each key is associated with only one value.
- ✔ Header File: <map>
- ✔ Iterator: Bidirectional

Container : Multimap

- ✔ Description: It is an associate container for storing key/value in which one key may be associated with more than one value (one-to-many mapping). It allows a key-based lookup.
- ✔ Header File: <map>
- ✔ Iterator: Bidirectional

Container : stack

- ✔ Description: A standard stack follows last-in-first-out(LIFO)
- ✔ Header File: <stack>
- ✔ Iterator: No iterator

Containers

Container : queue

- ✔ Description: A standard queue follows first-in-first-out(FIFO)
- ✔ Header File: <queue>
- ✔ Iterator: No iterator

Container : priority-queue

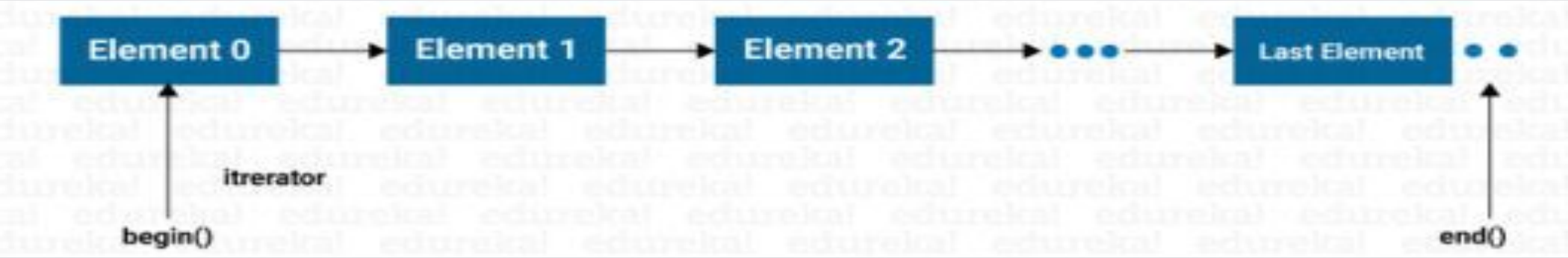
- ✔ Description: The first element out is always the highest priority element
- ✔ Header File: <queue>
- ✔ Iterator: No iterator

Sequence Containers

- Sequence containers store elements in a linear order.
- All elements are related to each other by their position along the line.
- They allow insertion of element and all of them support several operations on them.

The STL provides three types of sequence elements:

- Vector
- List
- Deque



Associative containers:

- They are designed in such a way that they can support direct access to elements using keys. They are not sequential.
- There are four types of associative containers:
  - Set
  - Multiset
  - Map
  - Multimap
- All the above containers store data in a structure called tree which facilitates fast searching, deletion, and insertion unlike sequential.
- Container set or multiset can store various items and provide operations for manipulating them using the values as the keys.
- And map or Multimap are used to store items in pair, one called the key and other called the value.

Derived containers:

- The STL provides three derived containers namely, stack, queue, and priority_queue. These are also known as container adaptors.
- There are three types of derived containers:
  - Stack
  - Queue
  - Priority_quue
- Stacks, queue and priority queue can easily be created from different sequence containers.
- The derived containers do not support iterators and therefore we cannot use them for data manipulation.
- However, they support two member function pop() and push() for implementing deleting and inserting operations.

Algorithms

- Algorithms are functions that can be used generally across a variety of containers for processing their content.
- Although each container provides functions for its basic operations, STL provides more than sixty standard algorithms to support more extended or complex operations.
- Standard algorithms also permit us to work with two different types of containers at the same time.
- STL algorithms save a lot of time and effort of programmers
- To access STL algorithms, we must include <algorithm> in our program.
- STL algorithm, based on the nature of operations they perform, may be categorized as under :
  - Nonmutating algorithms
  - Mutating algorithms
  - Sorting algorithms
  - Set algorithms
  - Relational algorithm

Iterators

- Iterators act like pointers and are used to access elements of the container.
- We use iterators to move through the contents of containers.
- Iterators are handled just like pointers.
- We can increment or decrement them as per our requirements.
- Iterators connect containers with algorithms and play a vital role in the manipulation of data stored in the containers.
- They are often used to pass through from one element to another, this process is called iterating through the container.
- There are five types of iterators:
  - Input
  - Output
  - Forward
  - Bidirectional
  - Random

Iterators

Iterator	Access method	Direction of movement	I/O capability	Remark
Input	Linear	Forward only	Read-only	Cannot be saved
Output	Linear	Forward only	Write only	Cannot be saved
Forward	Linear	Forward only	Read/Write	Can be saved
Bidirectional	Linear	Forward and backward	Read/Write	Can be saved
Random	Random	Forward and backward	Read/Write	Can be saved

Iterators

- Different types of iterators must be used with the different types of containers such that only sequence and associative containers are allowed to travel through iterators.
- Each type of iterators is used for performing certain functions.
- The input and output iterators support the least functions.
- They can be used only to pass through in a container.
- The forward iterators support all operations of input and output iterators and also retain its position in the container.
- A Bidirectional iterator, while supporting all forward iterators operations, provides the ability to move in the backward direction in the container.

Standard Template Library in C++

Container

- Sequence Containers
- Associative Containers
- Container Adapters
- Unordered Associative Containers

Iterator

- begin()
- next()
- prev()
- advance()
- end()

Algorithm

- Sorting Algorithms
- Search algorithms
- Non modifying algorithms
- Modifying algorithms
- Numeric algorithms
- Minimum and Maximum operations

Vector

- Vectors in C++ function the same way as Arrays in a **dynamic manner** i.e. vectors can resize itself automatically whenever an item is added/deleted from it.
- The data elements in Vectors are placed in contagious memory locations and Iterator can be easily used to access those elements.
- Moreover, **insertion of items in takes place at the end of Vector**.

Vector Syntax:

```
vector<data_type> vector_name;
```

Vector Functions:

- **vector.begin():** It returns an iterator element which points to the first element of the vector.
- **vector.end():** It returns an iterator element which points to the last element of the vector.
- **vector.push_back():** It inserts the element into the vector from the end.
- **vector.pop_back():** It deletes the element from the end of the vector.
- **vector.size():** This function gives the size i.e. the number of elements in the vector.
- **vector.empty():** Checks whether the vector is empty or not.
- **vector.front():** It returns the first element of the vector.
- **vector.back():** It returns the last element of the vector.
- **vector.insert():** This function adds the element before the element at the given location/position.
- **vector.swap():** Swaps the two input vectors.

Example: Vector

```
#include <iostream>

#include <vector>

 using namespace std;

 int main()

{

    vector<int> V1;

     for (int i = 1; i <= 4; i++)

        V1.push_back(i);

     cout << "Displaying elements of vector using begin() and
end():\n";

    for (auto i = V1.begin(); i != V1.end(); ++i)

        cout << *i << " ";

cout << "\nSize of the input Vector:\n" << V1.size();

     if (V1.empty() == false)
            cout << "\nVector isn't empty";
    else
            cout << "\nVector is empty";

    cout << "\nvector.front() function:\n" << V1.front();

    cout << "\nvector.back() function:\n" << V1.back();

    V1.insert(V1.begin(), 8);

    cout<<"\nVector elements after the insertion of element using
vector.insert() function:\n";

    for (auto x = V1.begin(); x != V1.end(); ++x)
        cout << *x << " ";

    return 0;
}
```

Output:

- Statement **V1.insert(V1.begin(), 8)** inserts the element (8) at the beginning i.e. before the first element of the vector.

```
Displaying elements of vector using begin() and end(): 1 2 3 4

Size of the input Vector: 4

Vector isn't empty

vector.front() function:1

vector.back() function:4

Vector elements after the insertion of element using vector.insert() function:8 1 2 3 4
```

Vector Example

```
#include <iostream>
#include <vector>
using namespace std;
 int main() {
   // create a vector to store int
   vector<int> vec;
   int i;
   // display the original size of vec
   cout << "vector size = " << vec.size() << endl;
   // push 5 values into the vector
   for(i = 0; i < 5; i++) {
      vec.push_back(i);
   }
   // display extended size of vec
   cout << "extended vector size = " << vec.size() << endl;

   // access 5 values from the vector
   for(i = 0; i < 5; i++) {
      cout << "value of vec [" << i << "] = " << vec[i] << endl;
   }

   // use iterator to access the values
   vector<int>::iterator v = vec.begin();
   while( v != vec.end()) {
      cout << "value of v = " << *v << endl;
      v++;
   }

   return 0;
}
```

Output:

vector size = 0

extended vector size = 5

value of vec [0] = 0

value of vec [1] = 1

value of vec [2] = 2

value of vec [3] = 3

value of vec [4] = 4

value of v = 0

value of v = 1

value of v = 2

value of v = 3

value of v = 4

- ✔ The push_back( ) member function inserts value at the end of the vector, expanding its size as needed.
- ✔ The size( ) function displays the size of the vector.
- ✔ The function begin( ) returns an iterator to the start of the vector.
- ✔ The function end( ) returns an iterator to the end of the vector.

- List Code in Codeblocks
- Dequeue --- Homework
- A vector is a single contiguous memory block.
- A deque is a set of linked memory blocks, where more than one element is stored in each memory block.
- A list is a set of elements dispersed in memory, i.e.: only one element is stored per memory "block".

References

- https://www.geeksforgeeks.org/the-c-standard-template-library-stl/
- https://www.tutorialspoint.com/cplusplus/cpp_stl_tutorial.htm
- https://www.hackerearth.com/practice/notes/stacks-and-queues/