STL

What is STL in C++?

The Standard Template Library or STL in C++ is a collection of template classes and template functions that provide a generic way of programming. It is a library of container classes, algorithms, and iterators.

It is commonly used for efficiently programming data structures, algorithms, and functions.

Components of STL in C++

There are four major components of STL in C++:

1. Containers
2. Iterators
3. Algorithms
4. Function objects or Functors

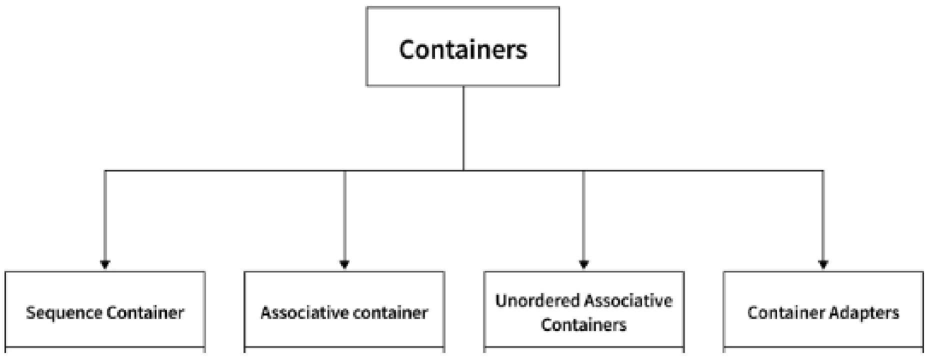
1. Containers

A container is a holder object that stores other objects as elements. It is used to implement different data structures.

They are implemented as class templates (templates use data types as parameters), allowing greater flexibility in the types supported as elements.

The containers manage storage space for its elements and provide member functions to access them directly or through iterators.

They replicate the most commonly used data structures like queues (queue), dynamic arrays (vector), stacks (stack), linked lists (list), queues (queue), heaps (priority_queue), trees (set), associative arrays (map), and many others.



There are 4 types of containers of STL in C++:

1. Sequence Containers - Array - Vector - Deque - List

2. Associative Containers - Set - MultiSet - Map - Multimap

3. Unordered Associative Containers - Unordered Set - Unordered Multiset - Unordered Map - Unordered Multimap

4. Container Adapters - Stack - Queue - Priority Queue

1. Sequence Containers

Sequence containers implement data structures that can be accessed sequentially via their position.

It preserves the insertion order of elements.

These are internally implemented as arrays or linked lists.

The five sequence containers supported by STL are:

a. Array

Arrays are sequential homogeneous containers of fixed size. The elements are stored in contiguous memory locations.

Syntax:

array<object_type, size> array_name;

b. Vector

Vectors are dynamic arrays, allowing the insertion and deletion of data from the end. They can grow or shrink as required. Hence their size is not fixed, unlike arrays.

Syntax:

```
vector<object_type> vector_name;
```

c. Deque

Deque is double-ended queue that allows inserting and deleting from both ends. They are more efficient than vectors in case of insertion and deletion. Its size is also dynamic.

Syntax:

```
deque<object_type> deque_name;
```

d. List

The list is a sequence container that allows insertions and deletions from anywhere. It is a doubly linked list. They allow non-contiguous memory allocation for the elements.

Syntax:

```
list<object_type> list_name;
```

e. Forward List

Forward Lists are introduced from C++ 11. They are implemented as singly linked list in STL in C++. It uses less memory than lists and allows iteration in only a single direction.

Syntax:

```
forward_list<object_type> forward_list_name;
```

2. Associative Containers

Associative container is an ordered (sorted) container that provides a fast lookup of objects based on the keys, unlike a sequence container which uses position.

A value is stored corresponding to each key.

They are internally implemented as binary tree data structures. This results in logarithmic time operations –O (log n)

Types of associative containers:

a. Set

The set is used to store unique elements. The data is stored in a particular order (increasing order, by default).

Syntax:

```
set<object_type> set_name;
```

b. Map

The map contains elements in the form of unique key-value pairs. Each key can be associated with only one value. It establishes a one-to-one mapping. The key-value pairs are inserted in increasing order of the keys.

Syntax:

```
map<key_object_type, value_object_type> map_name;
```

c. Multiset

Multiset is similar to a set but also allows duplicate values.

Syntax:

```
multiset<object_type> multiset_name;
```

d. Multimap

Multimap is similar to a map but allows duplicate key-value pairs to be inserted. Ordering is again done based on keys.

Syntax:

multimap<key_object_type, value_object_type> multimap_name;

3. Unordered Associative Containers

Unordered Associative Container is an unsorted version of Associative Container.

It is important to note that insertion order is not maintained. Elements are in random order.

These are internally implemented as a hash table data structure. This results, on average, in constant time operations – O (1)

a. Unordered Set

It is used to store unique elements.

Syntax:

unordered_set<object_type> unordered_set_name;

b. Unordered Map

It is used to store unique key-value pairs.

Syntax:

unordered_map<key_object_type, value_object_type> unordered_map_name;

c. Unordered Multiset

It is similar to an unordered set, but the elements need not be unique.

Syntax:

unordered_multiset<object_type> unordered_multiset_name;

d. Unordered Multimap

It is similar to an unordered map, but the duplicate key-value pairs can be inserted here.

Syntax:

unordered_map<key_object_type, value_object_type> unordered_map_name;

4. Derived Containers (Container Adapters in C++)

STL in C++ also consists of a special type of container that adapts other containers to give a different interface with restricted functionality. Hence the name Container Adapters or Derived containers.

The underlying container is encapsulated in such a way that its elements are accessed by the members of the container adaptor independently of the underlying container class used.

Unlike other containers, values are not directly initialized but with the help of other supported methods.

a. Stack

A Stack is a container that provides Last-In-First-Out (LIFO) access. All the operations occur at the same place called top of the stack. It is implemented on top of a deque by default.

Syntax:

stack<data_type> stack_name;

b. Queue

A queue is a container that provides First-In First-Out access. The insertion is done at the rear (back) position and deletion at the front. It is implemented on top of a deque by default.

Syntax:

queue<data_type> queue_name;

c. Priority Queue

A priority Queue is similar to a queue, but every element has a priority value. This value decides what element is present at the top, which is, by default, the greatest element in the case of STL in C++. This is similar to the max heap data structure. It is implemented on top of the vector by default.

Syntax:

priority_queue<object_type> priority_queue_name;