COMPONENTS OF STL

CONTAINERS

Containers can be described as the objects that hold the data of the same type. Containers are used to implement different data structures for example arrays, list, trees, etc.

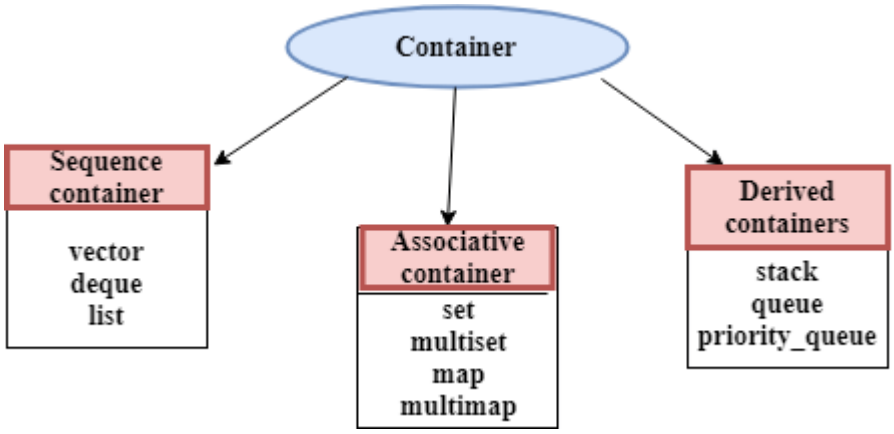
Following are the containers that give the details of all the containers as well as the header file and the type of iterator associated with them :

Container	Description	Header file	iterator
vector	vector is a class that creates a dynamic array allowing insertions and deletions at the back.	<vector>	Random access
list	list is the sequence containers that allow the insertions and deletions from anywhere.	<list>	Bidirectional
deque	deque is the double ended queue that allows the insertion and deletion from both the ends.	<deque>	Random access
set	set is an associate container for storing unique sets.	<set>	Bidirectional
multiset	Multiset is an associate container for storing non- unique sets.	<set>	Bidirectional
map	Map is an associate container for storing unique key-value pairs, i.e. each key is associated with only one value(one to one mapping).	<map>	Bidirectional

multimap	multimap is an associate container for storing key- value pair, and each key can be associated with more than one value.	<map>	Bidirectional
stack	It follows last in first out(LIFO).	<stack>	No iterator
queue	It follows first in first out(FIFO).	<queue>	No iterator
Priority-queue	First element out is always the highest priority element.	<queue>	No iterator

Classification of containers :

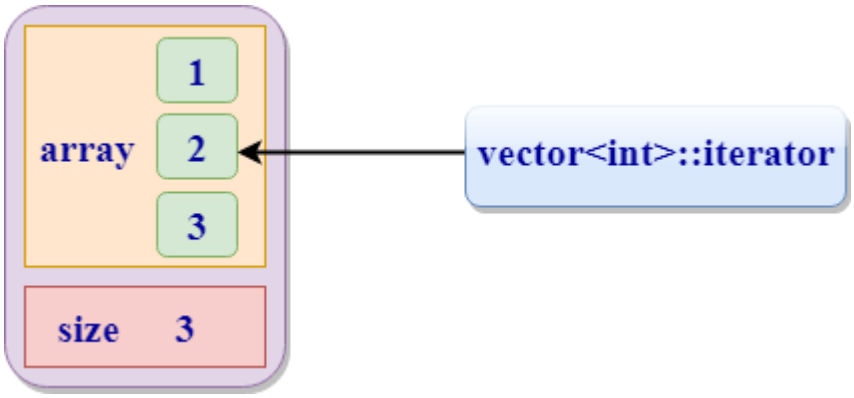
- Sequence containers
- Associative containers
- Derived containers



Note : Each container class contains a set of functions that can be used to manipulate the contents.

ITERATOR

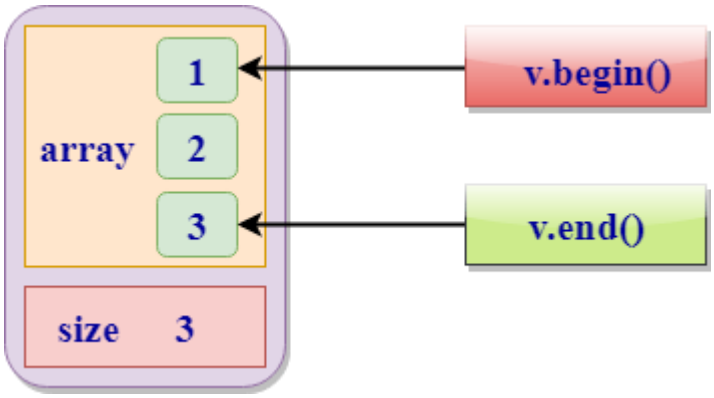
- Iterators are pointer-like entities used to access the individual elements in a container.
- Iterators are moved sequentially from one element to another element. This process is known as iterating through a container.



- Iterator contains mainly two functions:

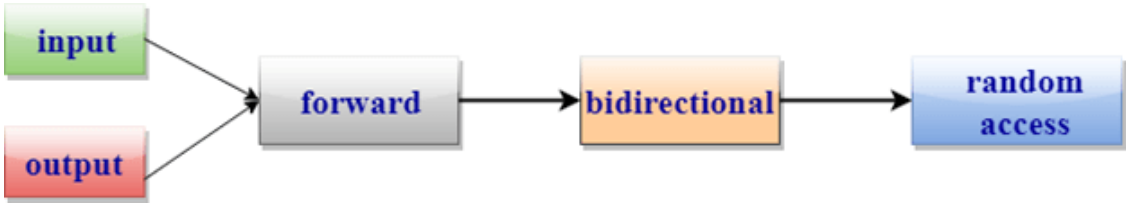
begin(): The member function begin() returns an iterator to the first element of the vector.

end(): The member function end() returns an iterator to the past-the-last element of a container.



Iterator Categories

Iterators are mainly divided into five categories:



1. **Input iterator:**
   - An Input iterator is an iterator that allows the program to read the values from the container.
   - Dereferencing the input iterator allows us to read a value from the container, but it does not alter the value.

- An Input iterator is a one way iterator.
- An Input iterator can be incremented, but it cannot be decremented.

2. **Output iterator:**
   - An output iterator is similar to the input iterator, except that it allows the program to modify a value of the container, but it does not allow to read it.
   - It is a one-way iterator.
   - It is a write only iterator.
3. **Forward iterator:**
   - Forward iterator uses the ++ operator to navigate through the container.
   - Forward iterator goes through each element of a container and one element at a time.
4. **Bidirectional iterator:**
   - A Bidirectional iterator is similar to the forward iterator, except that it also moves in the backward direction.
   - It is a two way iterator.
   - It can be incremented as well as decremented.
5. **Random Access Iterator:**
   - Random access iterator can be used to access the random element of a container.
   - Random access iterator has all the features of a bidirectional iterator, and it also has one more additional feature, i.e., pointer addition. By using the pointer addition operation, we can access the random element of a container.

Operations supported by iterators

iterator	Element access	Read	Write	Increment operation	Comparison
input	->	v = *p		++	==,!=
output			*p = v	++	
forward	->	v = *p	*p = v	++	==,!=

Bidirectional	->	v = *p	*p = v	++,--	==,!=
Random access	->,[ ]	v = *p	*p = v	++,--,+,-,+=,--=	==,!=, <,>,<=,>=

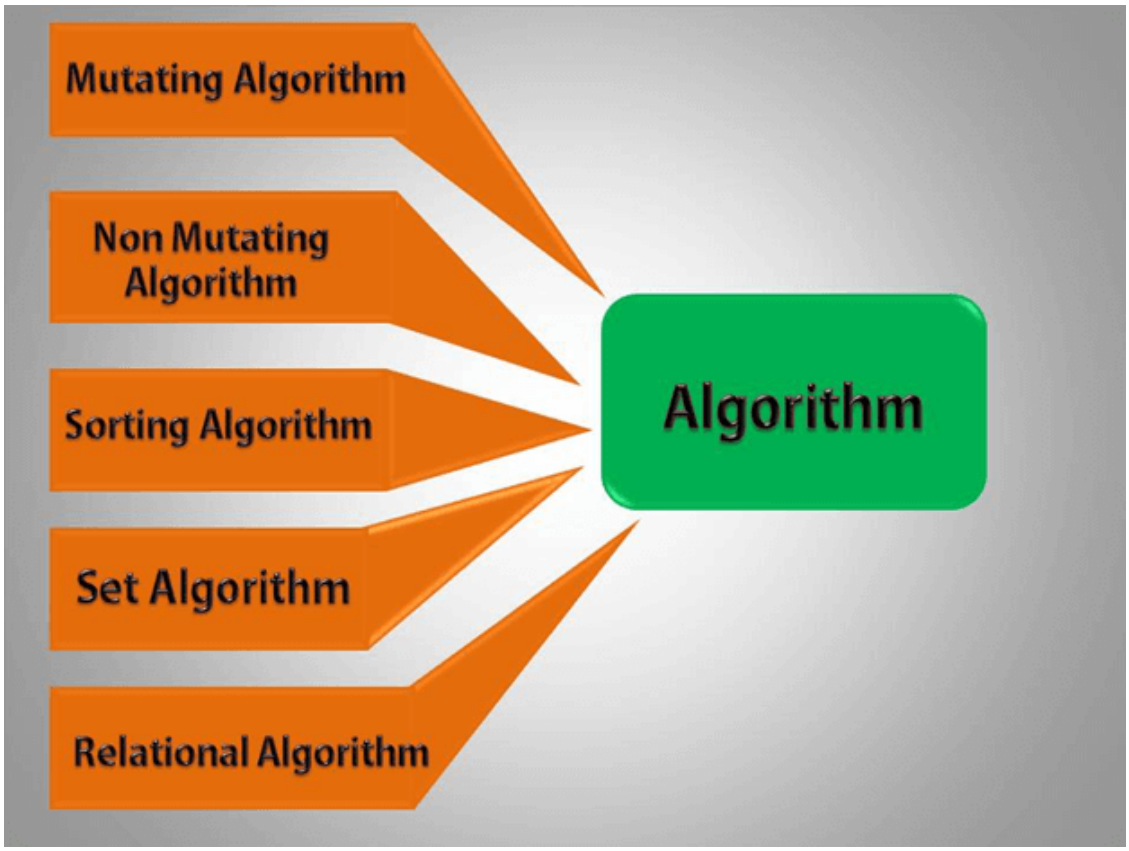
Algorithms

Algorithms are the functions used across a variety of containers for processing its contents.

Points to Remember:

- Algorithms provide approx 60 algorithm functions to perform the complex operations.
- Standard algorithms allow us to work with two different types of the container at the same time.
- Algorithms are not the member functions of a container, but they are the standalone template functions.
- Algorithms save a lot of time and effort.
- If we want to access the STL algorithms, we must include the <algorithm> header file in our program.

STL algorithms can be categorized as:



- **Nonmutating algorithms**: Nonmutating algorithms are the algorithms that do not alter any value of a container object nor do they change the order of the elements in which they appear. These algorithms can be used for all the container objects, and they make use of the forward iterators.
- **Mutating algorithms**: Mutating algorithms are the algorithms that can be used to alter the value of a container. They can also be used to change the order of the elements in which they appear.
- **Sorting algorithms**: Sorting algorithms are the modifying algorithms used to sort the elements in a container.
- **Set algorithms**: Set algorithms are also known as sorted range algorithm. This algorithm is used to perform some function on a container that greatly improves the efficiency of a program.
- **Relational algorithms**: Relational algorithms are the algorithms used to work on the numerical data. They are mainly designed to perform the mathematical operations to all the elements in a container.

FUNCTION OBJECTS

A Function object is a function wrapped in a class so that it looks like an object. A function object extends the characteristics of a regular function by using the feature of aN object oriented such as generic programming. Therefore, we can say that the

function object is a smart pointer that has many advantages over the normal function.

Following are the advantages of function objects over a regular function:

- Function objects can have member functions as well as member attributes.
- Function objects can be initialized before their usage.
- Regular functions can have different types only when the signature differs. Function objects can have different types even when the signature is the same.
- Function objects are faster than the regular function.

A function object is also known as a '**functor**'. A function object is an object that contains atleast one definition of **operator()** function. It means that if we declare the object 'd' of a class in which **operator()** function is defined, we can use the object 'd' as a regular function.

Suppose 'd' is an object of a class, operator() function can be called as:

```
d();

which is same as:

d.operator() ( );
```

Let's see a simple example:

```
#include <iostream>
using namespace std;
class function_object
{
   public:
  int operator()(int a, int b)
  {
     return a+b;
  }
};

 int main()
{
```

```
    function_object f;
    int result = f(5,5);
   cout<<"Addition of a and b is : "<<result;

  return 0;
}
```

Output:

```
Addition of a and b is : 10
```

In the above example, 'f' is an object of a function_object class which contains the definition of operator() function. Therefore, 'f' can be used as an ordinary function to call the operator() function.

← prev next →

Learn Latest Tutorials

OneNote

Data Ware.

VBA

SSIS

NGINX

Blockchain

ETL

Jenkins

Pytorch

Agile

JIRA

Tableau

Preparation

Aptitude

Reasoning

Verbal A.

Interview

Company

Trending Technologies


AI

AWS

Selenium

IoT


Cloud

Hadoop

ReactJS

React Native

Node.js

D. Science

Angular 7

B.Tech / MCA

DBMS

DS

DAA


OS


C. Network


Compiler D.

COA

D. Math.

E. Hacking

C. Graphics

Software E.

Web Tech.

Cyber Sec.

Automata

C

C++

Java

.Net

Python

Programs

Control S.