Standard Template Library

Prepared By

- Sreejith S -

- Rahul Babu R -

Contents

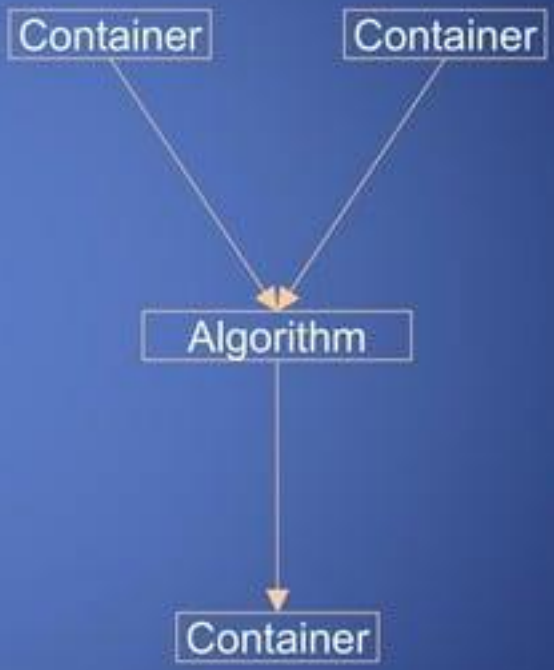
- Introduction To STL
- Containers
- Iterators
- Algorithms
- Function Objects

Introduction To STL

- STL is Standard Template Library
  - Powerful, template-based components
    - Containers: template data structures
    - Iterators: like pointers, access elements of containers
    - Algorithms: data manipulation, searching, sorting, etc.
  - Object- oriented programming: reuse, reuse, reuse
  - Only an introduction to STL, a huge class library

STL components overview

- Data storage, data access and algorithms are separated
  - *Containers* hold data
  - *Iterators* access data
  - *Algorithms*, *function objects* manipulate data
  - *Allocators*... allocate data (mostly, we ignore them)



Container

- A *container* is a way that stored data is organized in memory, for example an array of elements.

Container ctd-

- Sequence containers
  - **`vector`**
  - **`deque`**
  - **`list`**
- Associative containers
  - **`set`**
  - **`multiset`**
  - **`map`**
  - **`multimap`**
- Container adapters
  - **`stack`**
  - **`queue`**

Sequential Container

- `vector<T>` – dynamic array
  - Offers random access, back insertion
  - Should be your ***default choice, but choose wisely***
  - Backward compatible with C : `&v[0]` points to the first element
- `deque<T>` – double-ended queue (usually array of arrays)
  - Offers random access, back and front insertion
  - Slower than vectors, no C compatibility
- `list<T>` – 'traditional' doubly linked list
  - Don't expect random access, you can insert anywhere though

Some functions of vector class

-size()

    -provides the number of elements

-push_back()

    -appends an element to the end

-pop_back()

    -Erases the last element

-begin()

    -Provides reference to last element

-end()

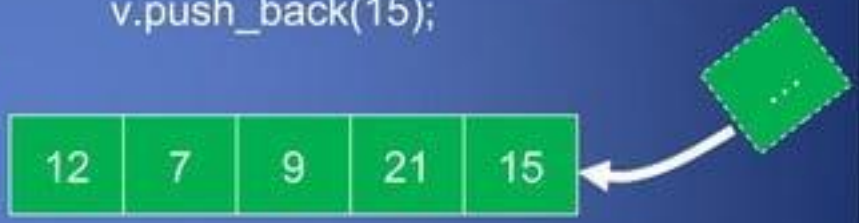
    -Provides reference to end of vector

Vector container

```
int array[5] = {12, 7, 9, 21, 13 };
Vector<int> v(array,array+5);
```

v.pop_back();

v.push_back(15);



Some function of list class

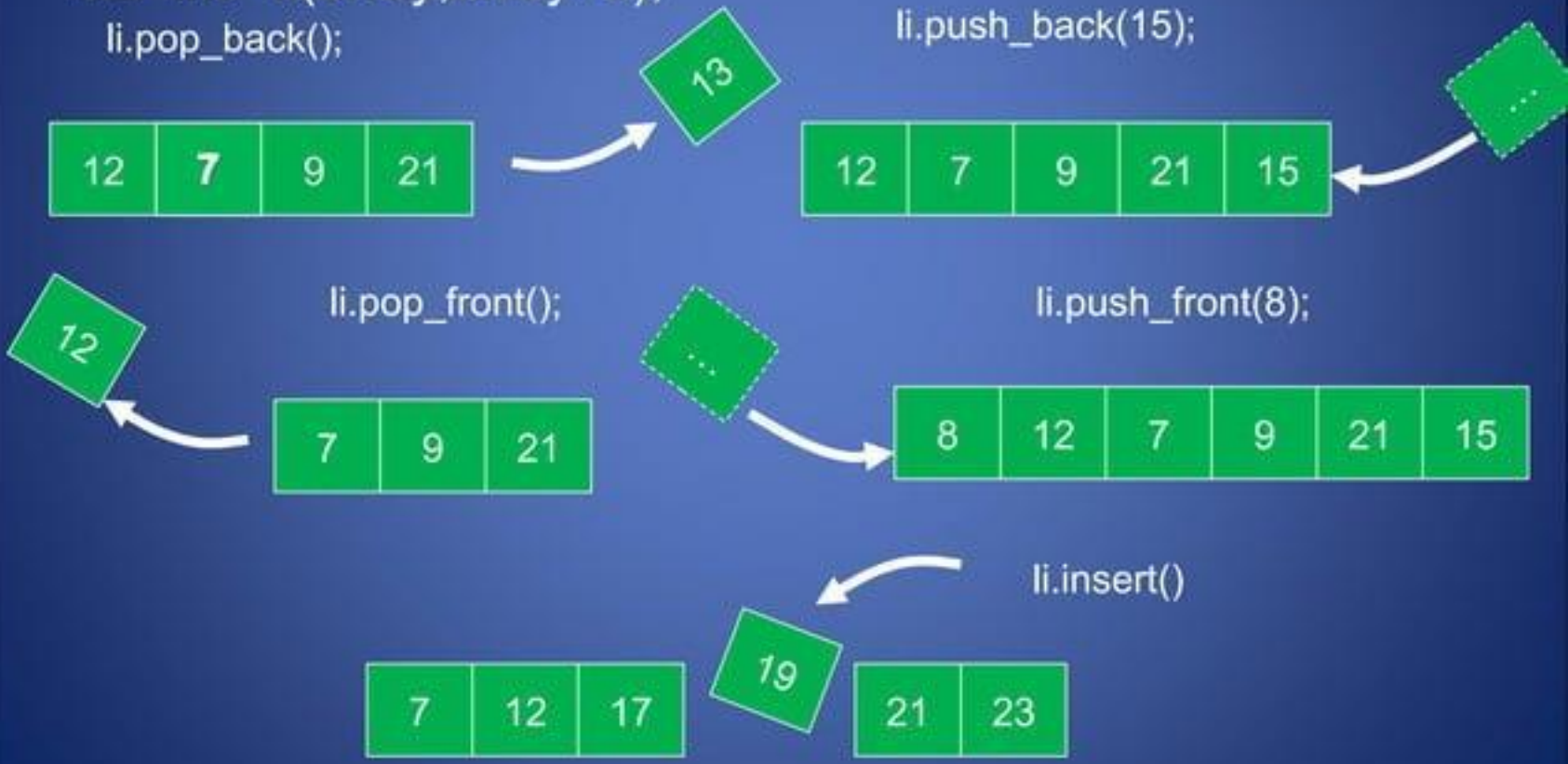
- **list** functions for object **t**
  - **t.sort()**
    - Sorts in ascending order
  - **t.splice(iterator, otherObject);**
    - Inserts values from **otherObject** before **iterator**
  - **t.merge(otherObject)**
    - Removes **otherObject** and inserts it into **t**, sorted
  - **t.unique()**
    - Removes duplicate elements

Functions of list class cntd-

- **list** functions
  - **t.swap(otherObject);**
    - Exchange contents
  - **t.assign(iterator1, iterator2)**
    - Replaces contents with elements in range of iterators
  - **t.remove(value)**
    - Erases all instances of **value**

List container

```
int array[5] = {12, 7, 9, 21, 13 };
list<int> li(array,array+5);
```



Functions of dequeue class

- dequeue functions for object d

  -**d.front()**

  -Return a reference (or const_reference) to the first component of d

  -**d.back()**

  -Return a reference (or const_reference) to the last component of d.

  -**d.size()**

  -Return a value of type size_type giving the number of values currently in d.

Functions of dequeue class contd-

-**d.push_back(val)**

-Add val to the end of d, increasing the size of d by one.

-**d.push_front(val)**

-Add val to the front of d, increasing the size of d by one.

-**d.pop_back()**

-Delete the last value of d. The size of d is reduced by one.

-**d.pop_front()**

-Delete the first value of d. The size of d is reduced by one.

Associative Containers

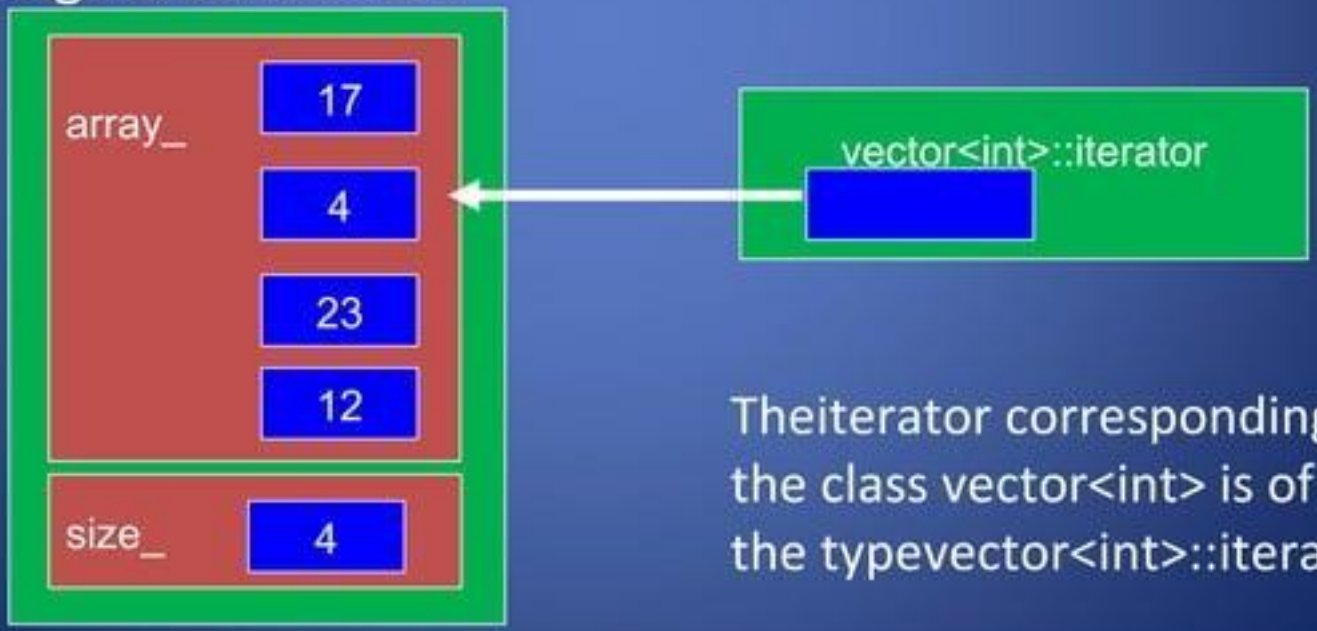
- Offer *O(log n)* insertion, suppression and access
- Store only weakly strict ordered types (eg. numeric types)
  - Must have `operator<()` and `operator==()` defined and !(a<b) && !(b<a) ≡ (a==b)
- The sorting criterion is also a template parameter
- `set<T>` – the item stored act as key, no duplicates
- `multiset<T>` – set allowing duplicate items
- `map<K,V>` – separate key and value, no duplicates
- `multimap<K,V>` – map allowing duplicate keys
- hashed associative containers *may* be available

Container adaptors

- Container adapters
  - **stack**, **queue** and **priority_queue**
  - Not first class containers
    - Do not support iterators
    - Do not provide actual data structure
  - Programmer can select implementation
  - Member functions **push** and **pop**

Iterators

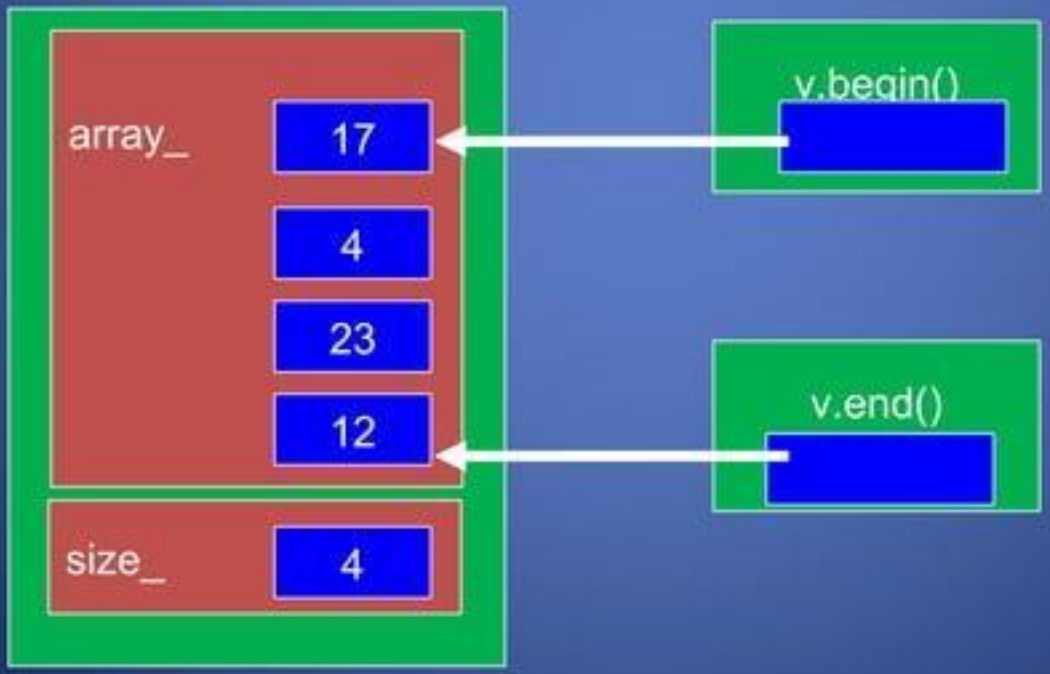
- Iterators are pointer-like entities that are used to access individual elements in a container.
- Often they are used to move sequentially from element to element, a process called *iterating* through a container.



Theiterator corresponding to the class vector<int> is of the typevector<int>::iterator

Iterators contd-

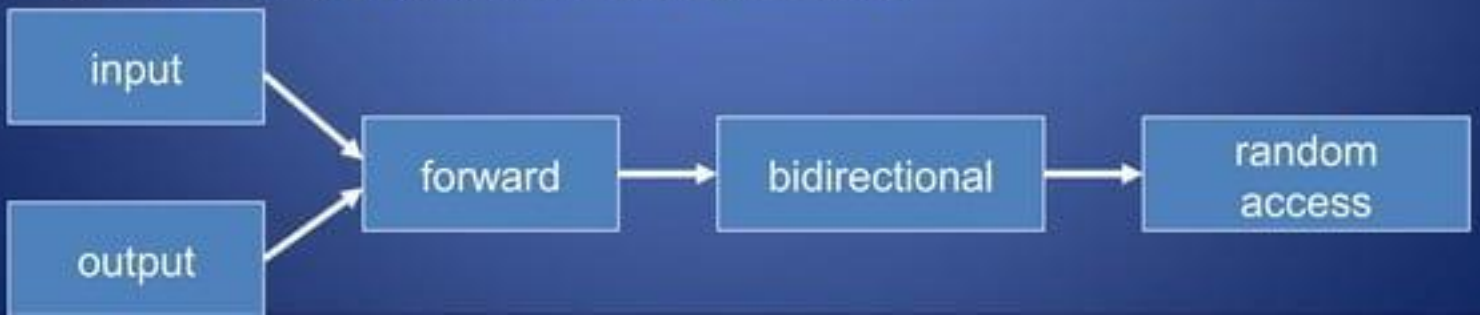
- The member functions begin() and end() return an iterator to the first and past the last element of a container



Iterators Categories

- Not every iterator can be used with every container for example the list class provides no random access iterator
- Every algorithm requires an iterator with a certain level of capability for example to use the [] operator you need a random access iterator
- Iterators are divided into five categories in which a higher (more specific) category always subsumes a lower (more general) category, e.g. An algorithm that

  accepts a forward iterator will also work with a bidirectional iterator and a random access iterator



Algorithms

Algorithms in the STL are procedures that are applied to containers to process their data, for example search for an element in an array, or sort an array.

For_Each() Algorithm

```
#include <vector>
#include <algorithm>
#include <iostream>
void show(int n)
{
  cout << n << " ";
}
 int arr[] = { 12, 3, 17, 8 };  // standard C array
vector<int> v(arr, arr+4);  // initialize vector with C array
for_each (v.begin(), v.end(), show); // apply function show
                // to each element of vector v
```

Find() Algorithm

```
#include <vector>
#include <algorithm>
#include <iostream>
int key;
int arr[] = { 12, 3, 17, 8, 34, 56, 9  };  // standard C array
vector<int> v(arr, arr+7);  // initialize vector with C array
vector<int>::iterator iter;
cout << "enter value :";
cin >> key;
iter=find(v.begin(),v.end(),key); // finds integer key in v
if (iter != v.end()) // found the element
   cout << "Element " << key << " found" << endl;
else
  cout << "Element " << key << " not in vector v" << endl;
```

Sort & Merge

- Sort and merge allow you to sort and merge elements in a container

```
#include <list>
int arr1[]= { 6, 4, 9, 1, 7 };
int arr2[]= { 4, 2, 1, 3, 8 };
list<int>  l1(arr1, arr1+5); // initialize l1 with arr1
list<int>  l2(arr2, arr2+5); // initialize l2 with arr2
l1.sort();  // l1 = {1, 4, 6, 7, 9}
l2.sort(); // l2= {1, 2, 3, 4, 8 }
l1.merge(l2);  // merges l2 into l1
// l1 = { 1, 1, 2, 3, 4, 4, 6, 7, 8, 9},  l2= {}
```

Functions Objects

- Some algorithms like sort, merge, accumulate can take a function object as argument.
- A function object is an object of a template class that has a single member function : the overloaded operator ()
- It is also possible to use user-written functions in place of pre-defined function objects

```
#include <list>
#include <functional>
int arr1[]= { 6, 4, 9, 1, 7 };
list<int>  l1(arr1, arr1+5); // initialize l1 with arr1
l1.sort(greater<int>());  // uses function object greater<int>
// for sorting in reverse order l1 = { 9, 7, 6, 4, 1 }
```

Function Objects

- The accumulate algorithm accumulates data over the elements of the containing, for example computing the sum of elements

```
#include <list>

#include <functional>

#include <numeric>

int arr1[]= { 6, 4, 9, 1, 7 };
list<int>  l1(arr1, arr1+5); // initialize l1 with arr1

int sum = accumulate(l1.begin(), l1.end() , 0, plus<int>());

int sum = accumulate(l1.begin(), l1.end(),0);  // equivalent

int fac = accumulate(l1.begin(), l1.end() , 0, times<int>());
```

User Defined Function Objects

```
class squared _sum  // user-defined function object
{
   public:
      int operator()(int n1, int n2) { return n1+n2*n2; }
};
int sq = accumulate(l1.begin(), l1.end() , 0,
    squared_sum() );
// computes the sum of squares
```

So long and thanks for all the attention ☺

THANK YOU.!!!

?