Standard Template Library

Containers

- Sequence (vector, list, deque)
- Associative (set, multiset, map, multimap)
- Container Adapters (stack, queue, priority_queue)

Iterators

Algorithm Library

STL Containers

Sequence containers have a concept of position with a technique to visit elements in order

Each is optimized for some combination of insert/removal

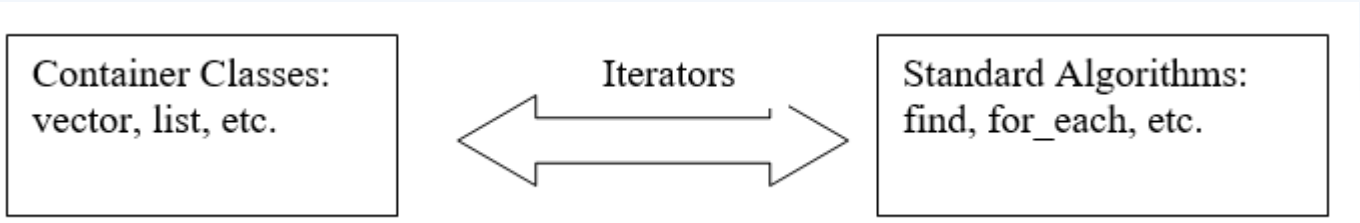
vector	rapid insert/remove at one end
	direct access to any position (array-based)
list	rapid insertion anywhere given the position
	doubly linked list (travel forward or backward)
deque	rapid insert/remove at either end
	direct access to any position (array/list based)

C++ 11:

array	wrapper class around static primitive array
forward_list	singly linked list (only travel forward)

Member function	Description
default constructor	A constructor that *initializes an empty container*. Normally, each container has several constructors that provide different ways to initialize the container.
copy constructor	A constructor that initializes the container to be a *copy* of an existing container of the same type.
move constructor	A move constructor (added in C++11 and discussed in Chapter 24) moves the contents of an existing container into a new container of the same type—the old container no longer contains the data. This avoids the overhead of copying each element of the argument container.
destructor	Destructor function for cleanup after a container is no longer needed.
`empty`	Returns `true` if there are *no* elements in the container; otherwise, returns `false`.
`insert`	Inserts an item in the container.
`size`	Returns the number of elements currently in the container.
`operator==`	Returns `true` if the contents of the first container are *equal to* the contents of the second; otherwise, returns `false`.
`operator!=`	Returns `true` if the contents of the first container are *not equal to* the contents of the second; otherwise, returns `false`.
`erase`	Removes *one or more* elements from the container.
`clear`	Removes *all* elements from the container.

STL Iterators



vector:

list:

find algorithm (sequential search) can be applied to vector or list

How? They have different implementations

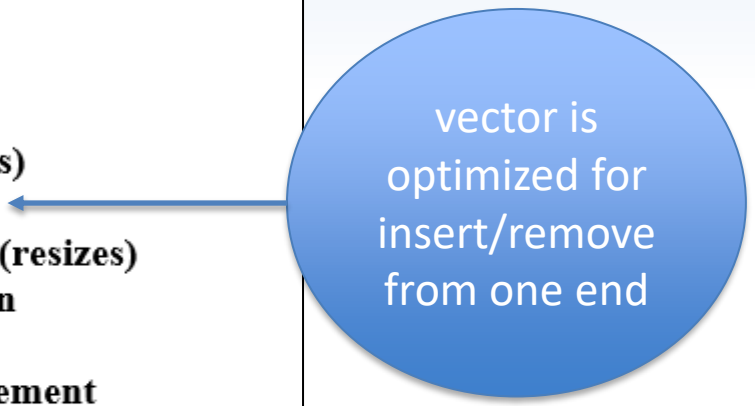
An iterator class wraps around a pointer

A vector iterator uses pointer increment to advance to next array element
A list iterator uses pointer dereference to advance to the next node

find uses the iterator for the class it is passed

Some vector Functions

```
v.capacity ( )                  // return the number of positions available in v
v.size ( )                      // return the number of values currently held in v
v.empty ( )                     // return true if v.size == 0
v.reserve ( n );                // increase the capacity of v to n

v.push_back (value)             // append value to end of v (resizes)
v.pop_back ( )                  // remove value at v's end
v.insert (position, value)      // insert value at iterator position (resizes)
v.erase (position)              // remove value at iterator position

v.front ( )                     // return a reference to v's first element
v.back ( )                      // return a reference to v's last element

v[ i ]                          // access the element of v at index i
v.at ( i )                      // access the element of v at index i

v1 = v2;                        // v1 has the same size and elements as v2

v1 == v2                        // return true if vectors contain the same values
```



vector Constructors

```
#include <vector>
#include <iostream>
#include <string>
using namespace std;

int main()
{   vector <int> iv;                    // int vector capacity 0, size 0
    cout << "iv capacity: " << iv.capacity() << endl;
    cout << "iv size: " << iv.size() << endl;

    vector <string> sv(5);              // string vector capacity 5, size 5, filled with ""
    cout << "sv capacity: " << sv.capacity() << endl;
    cout << "sv size: " << sv.size() << endl;

    vector <double> dv(5, 3.5);         // double vector capacity 5, size 5, filled with 3.5
    cout << "dv capacity: " << dv.capacity() << endl;
    cout << "dv size: " << dv.size() << endl;

    for (double d: dv)
        cout << d << " ";
    return 0;
}
```

```
iv capacity: 0
iv size: 0
sv capacity: 5
sv size: 5
dv capacity: 5
dv size: 5
3.5 3.5 3.5 3.5 3.5
```

vector Example

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

int main()
{   vector <int> v;
    int number;
    cout << "Enter integers (0 to quit): ";
    cin >> number;
    while (number != 0)
    {   v.push_back(number);
        cin >> number;
    }
    cout << "Size is: " << v.size() << endl;
    cout << "Removing last number: " << v.back() << endl;
    v.pop_back();
    cout << "Size now is: " << v.size() << endl;
    v[0] = 100;
    v.at(1) = 200;
    for (int k=0; k<v.size(); k++)
        cout << v[k] << " ";
    return 0;
}
```

```
Enter integers (0 to quit): 10 20 30 40 0
Size is: 4
Removing last number: 40
Size now is: 3
100 200 30
```

vector Iterator

A vector iterator is a random access iterator

It can advance forward or backward by any number of units

Iterator objects support:

++	advance to next item
+ n	advance n times
--	backup to last item
- n	backup n times
*	dereference to get access to item

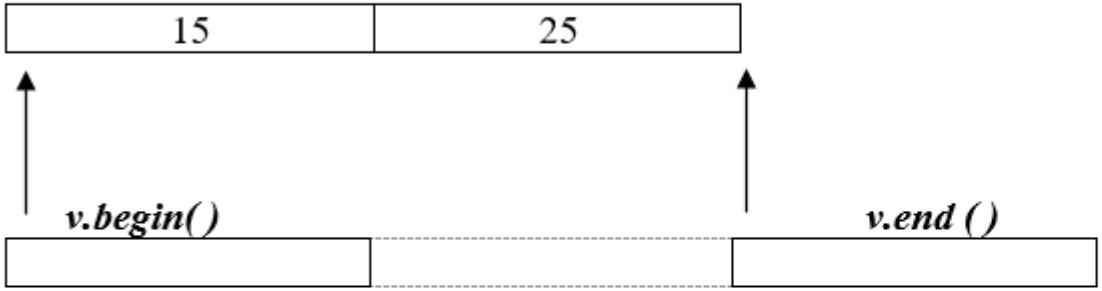
vector function begin( ) returns an iterator to the first item

vector function end ( ) returns an iterator to the "end"

Vector Iterator

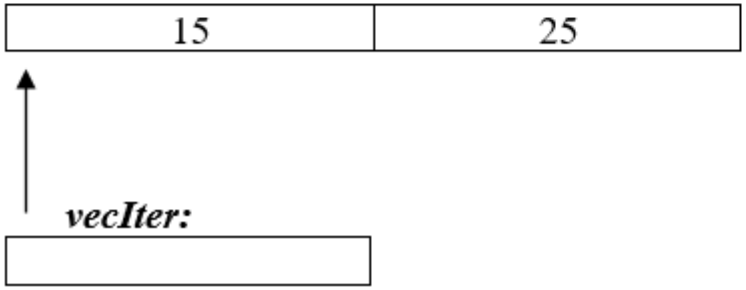
```
vector  <int> v;
v.push_back (15);
v.push_back (25);
```

v:



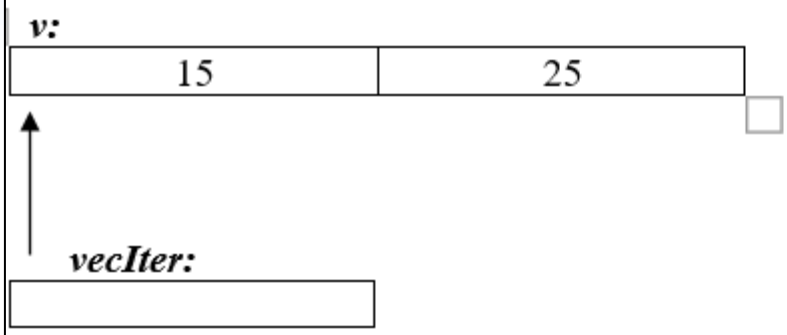

```
vector <int>::iterator  vecIter;          // object vecIter is an iterator to an int vector
vecIter = v.begin( );                     // vecIter is assigned the value of the  iterator object
                                          // returned from call to v.begin ( )
```

v:

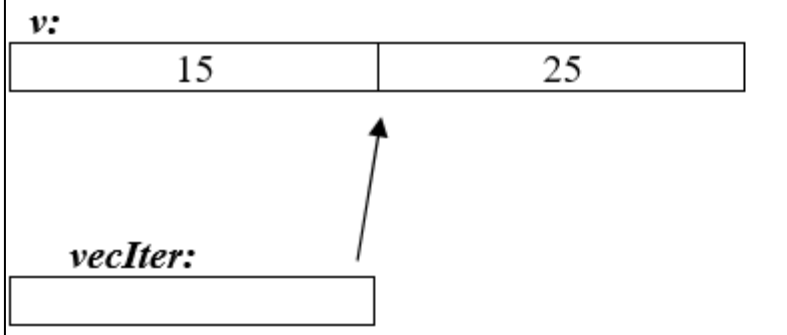


Vector Iterator

```
vector <int>::iterator  vecIter;        // object vecIter is an iterator to an int vector
vecIter = v.begin( );                   // vecIter is assigned the value of the  iterator object
                                        // returned from call to v.begin ( )
```




```
vecIter++;
```



Using a Vector Iterator

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

void display (vector<int> v);

int main()
{   vector <int> v;
    v.push_back (15);
    v.push_back (25);
    v.push_back (35);
    v.push_back (45);
    v.push_back (55);
    display (v);
    v.erase(v.begin() + 1);                 // remove from second position (25)
    display (v);
    v.insert (v.begin() + 2, 100);          // insert at third position
    display (v);
    return 0;
}
void display(vector<int> v)
{   vector<int>::iterator iter = v.begin();
    while (iter != v.end())
    {   cout << *iter << " ";
        iter++;
    }
    cout << endl;
}
```

```
15 25 35 45 55
15 35 45 55
15 35 100 45 55
```

```
for (int k=0; k<v.size(); k++)
        cout << v[k] << " ";
cout << endl;
```

STL Containers

Sequence containers have a concept of position with a technique to visit elements in order

Each is optimized for some combination of insert/removal

vector	rapid insert/remove at one end
	direct access to any position (array-based)
list	rapid insertion anywhere given the position
	doubly linked list (travel forward or backward)
deque	rapid insert/remove at either end
	direct access to any position (array/list based)

STL Containers

Associative containers store data by key

The following containers are ordered

Iteration will report keys in natural order defined by <

set	group of keys with no duplicates
multiset	permits duplicates
map	group of key:value pairs with no duplicates
multimap	permits duplicates (ex. key is ID number, value is name of person)

STL Containers

Container Adapters are constructed from sequence containers and do not permit iteration

They have constrained access

stack	LIFO access
queue	FIFO access
priority_queue	insert by priority
	removal of item with highest priority

Algorithm Library

Use the algorithm library with primitive arrays

Pass in the start and end addresses of the array

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

int main()
{   int a[] = {50, 10, 30};
    sort(a, a + 3);
    for (int k = 0; k < 3; k++)
        cout << a[k] << " ";
    cout << endl;
    return 0;
}
```

```
10 30 50
```

Algorithm Library

With STL containers, functions are passed iterators

Pass in start and end iterators of the container

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{   vector<int> v;
    v.push_back(50);
    v.push_back(10);
    v.push_back(30);
    sort(v.begin(), v.end());
    for (int k = 0; k < 3; k++)
        cout << v[k] << " ";
    cout << endl;
    return 0;
}
```

```
10 30 50
```