STL: Standard Template Library

- ANSI/ISO C++ Standard set of templates
- Developed by Stepanov, Lee, and Musser at HP (1994)

- Three key components of STL:
  - containers
  - iterators
  - algorithms

STL Containers

- STL Containers: Sequence Containers, Associative Containers, Container Adaptors

- Goal is to be flexible and very efficient
- Provides a set of standard operations with standard names and sematics
- Standard iterators

Sequence Containers

vector	rapid insertions and deletions at back direct access to any element
deque	rapid insertions and deletions at front or back direct access to any element
list	doubly-linked list rapid insertions and deletions anywhere

Associative Containers

set	rapid lookup, no duplicates allowed
multiset	rapid lookup, duplicates allowed
map	one-to-one mapping, no duplicates rapid key based lookup
multimap	one-to-many mapping, duplicates rapid key based lookup

Container Adapters

stack	last in first out (LIFO)
queue	first in first out (FIFO)
priority_queue	highest priority element is always first out

Containers

- These are referred to as first-class-containers
- Near-containers:
  - C-like array
  - **string**
  - **bitset**
  - **valarray**

- Exhibit similar capabilities to first class containers but do not support all the flexibility and capabilities as first-class containers

Common Operations for Containers

default constructor	A constructor to provide a default initialization
copy constructor	Copy existing container to another
destructor	Clean up
empty	Returns **true** if no elements otherwise **false**
max_size	Returns max number of elements for container
size	Returns the current number of elements
operator=	Assigns one container to another
operator<, operator<=	Obvious definitions (not defined for **priority_queue**)
operator>, operator>=	
operator==, operator!=	
swap	Swaps the elements of two containers

Common Operations for Containers

begin	Returns an iterator or a const_iterator that refers to the first element of the container
end	Returns an iterator or a const_iterator that refers to the next position **after the end** of the container
rbegin	Returns a reverse_iterator or a const_reverse_iterator that refers to the last element of the container
rend	Returns a reverse_iterator or a const_reverse_iterator that refers to the position before the first element of the container
erase	Erases one or more elements from the container
clear	Erases all elements from the container

Header files for STL Containers

<vector>	
<list >	
<deque >	
<queue>	both **queue** and **priority_queue**
<stack>	
<map>	both **map** and **multimap**
<set>	both **set** and **multiset**
<bitset>	
<iterator>	

Common **typedef**s for Containers

value_type	The element stored in the container.
reference	A reference to the type of element stored.
const_reference	A reference to the type of element stored.
pointer	A pointer to the type of element stored.
iterator	An iterator that points to the type of element.
const_iterator	
reverse_iterator	
const_reverse_iterator	
difference_type	The type of the result of subtracking tow iterators that refer to the same containter
size_type	The type used to count items in a container and index through a sequence container.

Iterator Types

Input	Used to read an element from a container. Input iterators support only one pass algorithms
Output	Used to write an element to a container. One pass only
Forward	Combines capabilities of input and output iterators
bi-directional	Combines forward with the ability to move backward. Support multi pass algorithms
random access	Combines bidirectional with ability to jump forward or backward by an arbitrary number of elements

Example - vector

```
std::vector<int> v;
std::cout << "The initial size of v is: " << v.size()
          << "\nThe initial capacity of v is: " << v.capacity();
v.push_back( 2 );  v.push_back( 3 );  v.push_back( 4 );
std::cout << "\nThe size of v is: " << v.size()
          << "\nThe capacity of v is: " << v.capacity();
std::cout << "\n\nContents of vector v a using array notation: ";
for (int i=0; i<v.size(); ++i)
        std::cout << v[i] << " ";
std::cout << "\nContents of vector v using iterator notation: ";
for (std::vector<int>::const_iterator p1 = v.begin();
                                      p1 != v.end(); p1++)
       std::cout << *p1 << " ";
std::cout << "\nReversed contents of vector v: ";
std::vector<int>::reverse_iterator p2;
for (p2 = v.rbegin(); p2 != v.rend(); ++p2)
       std::cout << *p2 << " ";
```

Output

```
The initial size of v is: 0
The initial capacity of v is: 0
The size of v is: 3
The capacity of v is: 4
Contents of vector v a using array notation: 2 3 4
Contents of vector v using iterator notation: 2 3 4
Reversed contents of vector v: 4 3 2
```

Example – list

```
std::list<int> lst;
lst.push_back(10); lst.push_back(20);
lst.push_back(30); lst.push_back(40);
for (std::list<int>::const_iterator i = lst.begin();
     i != lst.end();
     ++i) { std::cout << *i << " "; }
std::cout << std::endl;
std::list<int>::iterator ptr = lst.begin();
++ptr; ++ptr;
lst.insert(ptr, 100);
for (std::list<int>::const_iterator i = lst.begin();
     i != lst.end();
     ++i) { std::cout << *i << " "; }
std::cout << std::endl;
```

Output

```
10 20 30 40
10 20 100 30 40
```

Example – map

```
std::map<std::string, int> tbl;
tbl["joe"] += 1;
tbl["joe"] += 1;
tbl["sue"] += 1;
tbl["jon"] += 1;
tbl["sue"] += 1;
tbl["fred"] += 1;
for(std::map<std::string, int>::const_iterator i = tbl.begin();
    i != tbl.end(); ++i) {
        std::cout << i->first << " " << i->second << std::endl;
}
```

Output:

```
fred 1
joe 2
jon 1
sue 2
```

Output

```
10 20 30 40
10 20 100 30 40
```

Example - set

```
typedef std::set<double, std::less<double>> double_set;
const int SIZE = 5;
double a[ SIZE ] = { 2.1, 4.2, 9.5, 2.1, 3.7 };
double_set doubleSet( a, a + SIZE );;
std::ostream_iterator<double> output( std::cout, " " );
std::cout << "doubleSet contains: ";
std::copy( doubleSet.begin(), doubleSet.end(), output );
std::pair<double_set::const_iterator, bool> p;
p = doubleSet.insert( 13.8 ); // value not in set
std::cout << '\n' << *( p.first )
     << ( p.second ? " was" : " was not" ) << " inserted";
std::cout << "\ndoubleSet contains: ";
std::copy(doubleSet.begin(), doubleSet.end(), output);
p = doubleSet.insert( 9.5 );  // value already in set
std::cout << '\n' << *( p.first )
        << ( p.second ? " was" : " was not" ) << " inserted";
std::cout << "\ndoubleSet contains: ";
std::copy(doubleSet.begin(), doubleSet.end(), output);
```

Output - set

```
Output:

doubleSet contains: 2.1 3.7 4.2 9.5
13.8 was inserted
doubleSet contains: 2.1 3.7 4.2 9.5 13.8
9.5 was not inserted
doubleSet contains: 2.1 3.7 4.2 9.5 13.8
```

- ostream_iterator declares an iterator on ostream that is a type safe output mechanism that will only output values of type double.
- Typdef creates a new type for a set of double values oreder in ascending order using the function object less<double>
- pair defines a type with two values. In this case an iterator and a bool. Insert returns a pair.

Algorithms

- A set of algorithms that can be used generically across a variety of containers.
- Around 60 standard algorithms.
- begin() returns an iterator to the first element of a container.
- end() returns an iterator to the first position past the last element of a container.
- Algorithms often return iterators.

- find() locates a particular element and returns an iterator to that element. If the element is not found it returns end().

Algorithms (modifying)

copy()

copy_backward()

fill()

fill_n()

generate()

generate_n()

partition()

random_shuffle()

remove()

remove_copy()

remove_copy_if()

remove_if()

replace_copy()

replace_copy_if()

replace_if()

reverse()

reverse_copy()

rotate()

swap()

transform()

Algorithms (non-modifying)

find()

find_if()

count()

mismatch()

search()

for_each()

adjacent_find()

count_if()

equal()

find_end()

count()

- The count() and count_if() algorithms count occurrences of a value in a sequence

```
int occurs(const char *p, int size) {
  int n = count(p, p+size, ‘e’);
  return n;
}
```

Algorithms (Sorted Sequences)

sort()
partial_sort()
nth_element()
lower_bound()
upper_bound()
stable_sort()
binary_search()
merge()
partition()
stable_partition()

Set Algorithms

set_union()
set_intersection()
set_difference()

Algorithms (Sorted Sequences)

```
bool greater10( int value );

int main() {
   const int SIZE = 10;
   int a[ SIZE ] = { 10, 2, 17, 5, 16, 8, 13, 11, 20, 7 };
   vector< int > v( a, a + SIZE );
   ostream_iterator< int > output( cout, " " );

   cout << "Vector v contains: ";
   copy( v.begin(), v.end(), output );
```

Algorithms (Sorted Sequences)

```
vector< int >::iterator location;
location = find( v.begin(), v.end(), 16 );

if ( location != v.end() )
   cout << "\n\nFound 16 at location "
        << ( location - v.begin() );
else
   cout << "\n\n16 not found";
location = find( v.begin(), v.end(), 100 );
if ( location != v.end() )
   cout << "\nFound 100 at location "
        << ( location - v.begin() );
else
   cout << "\n100 not found";
```

Algorithms (Sorted Sequences)

```
   location = find_if( v.begin(), v.end(), greater10 );

   if ( location != v.end() )
      cout << "\n\nThe first value greater than 10 is "
           << *location << "\nfound at location "
           << ( location - v.begin() );
   else
      cout << "\n\nNo values greater than 10 were found";
```

```
bool greater10( int value ) { return value > 10; }
```

Algorithms (Sorted Sequences)

```
   sort( v.begin(), v.end() );
   cout << "\n\nVector v after sort: ";
   copy( v.begin(), v.end(), output );
   if ( binary_search( v.begin(), v.end(), 13 ) )
      cout << "\n\n13 was found in v";
   else
      cout << "\n\n13 was not found in v";

   if ( binary_search( v.begin(), v.end(), 100 ) )
      cout << "\n100 was found in v";
   else
      cout << "\n100 was not found in v";

   cout << endl;
   return 0;
}
```

Searching/sorting

- Find_if - looks at user defined conditional
- sort() - ascending order
- binary_search() - sequence must be sorted in ascending order.
- Output:

```
Vector v contains:  10 2 17 5 16 8 13 11 20 7
Found 16 at location 4
100 not found
The first value greater than 10 is 17
found at location 2
Vector v after sort: 2 5 7 8 10 11 13 16 17 20
13 was found in v
100 was not found
```