Lecture 23: Intro to STL containers
STL stands for Standard Template Library

There are many data structures that share similar features

These data structures can be manipulated by a common set of algorithms

STL is the C++ library that contains these data structures and algorithms

You already know one STL container!!
Different types of STL containers

Fundamental containers (sequential containers)
• Vector
• List
• Deque

Adapters (Built using the fundamental containers)
• Queue
• Stack
• Priority queue

Associative containers (elements ordered for fast retrieval)
• Set
• Map
In a linked list objects are strung together by "arrows"

Every object has an "arrow" to a previous and next objects.
Lists

Advantages

• Speedy insertion and deletion compared to vectors

Disadvantage

• No indexing
Example

```cpp
list<int> L;
L.push_back(3);
L.push_back(5);
L.push_front(0);
L.push_back(6);

What does L look like?
```
Iterators

How can we access elements in a data structure that has no indexes?

Some data structures elements are not even ordered in any particular way.

All STL containers have iterators that can be used to step through the container.

Iterator is an object that is storing a location of an element in the container and it can be incremented to find the next iterator.

See example for lists.
list<int>::iterator i; // Declare an iterator

i = L.begin(); // Set the iterator to the beginning of L

++i; // With iterators it is better to get into the habit of using pre increment.
++i;

L.insert(i,2);

For( i = L.begin(); i != L.end(); ++i )
    cout << *i << " ";

cout << endl;
Iterators can be used to step through containers:

```cpp
for( i = L.begin(); i != L.end(); ++i )
    cout << *i << " ";

cout << endl;

*i is used to access the object that i is associated with
```
list<int> one;
list<int> two;

//both lists are filled
list<int>::iterator p = one.begin();

while(p != two.end())
{
  //stuff

  //stuff
  ++p;
}
Note that for all the containers

c.end()

Is an iterator, but it is actually out of bounds.

It is the first iterator that is out of bounds
and it should not be accessed using the *operator.

When we think of two iterators representing a range the first iterator corresponds to an object at the beginning of the range, however,
The second iterator only marks the end object and is not considered Part of the range.
list<string> words;
// initialize words list

// Find the first occurrence of the string "ant"
list<string>::iterator pos = find(words.begin(),
words.end(), "ant");
if (pos != words.end())
{
    list<string>::iterator end = find(pos,
words.end(), "bee");
    while (pos != end)
    {
        cout << "Word is " << *pos << "\n";
    }
}
Queues

Queue is a data structure implementing First in First Out (FIFO)

Many situations in programming are best implemented as a queue. Think print requests sent to a printer.

See example.
Stacks

Stacks implement Last in First Out (LIFO).

Think of a stack of books (or people hired to a business.)

Stacks also have a push and pop

Instead of front stack has top

See example
Sets

Sets do not keep things in the order you inserted them

Sets reject duplicates

Major advantage of a set is that an item can be found quickly

Basic operations for a set are:

- Adding elements
- Removing elements
- Finding elements
- Traversing elements

Like lists, sets have iterators

See example
Similarities between the containers

All STL containers are templated

vector<string> words;
vector<Point> Path;

list<int> numbers;
stack<vector<int>> NumberLists;
All STL containers have iterators

Syntax:

container_type<elements_data_type>::iterator i;

Example:

list<string>::iterator i;
Similarities between the containers

You can use the following member functions with any STL container C:

C.size()

Returns number of items in the container (int).

C.begin()

Returns iterator to first element.

C.end()
Returns iterator to "sentinel" node at end.
You can step through a container using iterators

```cpp
list<string>::iterator iter = myList.begin();

while (iter != myList.end())
{
    cout << *iter;
    ++iter;
}
```
Differences between STLs

Many containers have their own member functions.

For example:

```cpp
template<typename T, size_t N> vector<T> vector<T>(N);
```

```cpp
vector<string> words(10);
words[0] = "Homer";
```

makes sense, but

```cpp
list<int> numbers;
numbers[0] = 7;
```

Does not.
Differences between STLs

Push and pop make sense for queues and stacks, but there is no push and pop for sets.

```cpp
stack<int> numbers;
numbers.push(2);

set<int> numbers;
numbers.push(2); // error
```
Recall the `<algorithm>` library we used with vectors.

```cpp
vector<int> numbers(10);
// fill the vector
numbers.sort(numbers.begin(), numbers.end());
```

Same technique can be used to sort a deque or even an ordinary array.

Sets and maps are already sorted.

Lists have their own sort member function.
Common algorithms

These are for vectors and deques

vector<int> a(10);

Fill a vector with 1s
fill (a.begin(), a.end(), 1);

Fill next 5 positions with 1s
fill_n(a.begin(), a.end(), 5, 1);

vector<int>b(20);
Initialize first 10 positions of b
copy((a.begin(), a.end(), b.begin());
sort(a.begin(), a.end());
reverse(a.begin(), a.end());
random_shuffle(a.begin(), a.end());
rotate(a.begin(), a.end());
There are many algorithms that can take a function as part of the input!

```cpp
bool is_even(int n)
{
    return !(n%2);
}
```

First even number in a list

```cpp
list<int>::iterator = find_if(a.begin(),a.end(), is_even)
```
int random_100()
{
    return 1 + rand() % 100;
}

generate(a.begin(), a.end(), random_100);
Using functions as arguments

```cpp
partition(a.begin(), a.end(), is_even);
```
Using functions as arguments

```cpp
int square(int n)
{
    return n*n;
}

transform(a.begin(),a.end(),b.begin, square);
```