CS 368 Announcements
Wednesday, May 1, 2013

Program p5 – due Wednesday, 5/8, at 10:00 pm – 10%

Last Time
- finish Ch. 9
- manipulators
- start Ch. 7 (Templates)
- templated functions
- templated classes

Today
- finish Ch. 7, start Ch. 10 (Collections: The STL)
- compiling with templates
- more template features
- STL overview
- containers
- iterators intro

Next Time
- cont. Ch. 10
- iterators
- function objects (functors)
- algorithms

Compiling Templated Functions and Classes

What happens at compile time:

Compiling Templated Functions and Classes (cont.)

Separate compilation

Inclusion model

Special Template Features

Multiple template parameters

```
template <typename KeyType, typename ValueType>
class Map {
    ...
};
```

Specialized templates

```
template <typename T>
const T & minimum(const T & x, const T & y) {
    return (x < y) ? x : y;
}
```

Template non-type parameters

```
template <typename Object, int size>
class Buffer {
    ...
  private:
    Object buf[size];
};
```

Standard Template Library (STL)

reference & download: **`http://www.sgi.com/tech/stl/`**

STL is a library of:

- containers
- algorithms
- iterators

Find Algorithm:

```
template <class InputIterator, class T>
InputIterator find(InputIterator first, InputIterator last,
                                              const T& value) {
    while (first != last && *first != value) ++first;
    return first;
}
```

STL makes heavy use of:

- templates
- operator overloading
- iterators

Some STL Containers

Sequence Containers

- vector
- list
- deque

Associative Containers

- set
- multiset

- map
- multimap

- + hash versions

Container Adaptors

- stack
- queue
- priority_queue

Container Operations

Operations all containers support:

- int size() const
- void clear()
- bool empty() const
- some kind of add op

Sequence container operations:

Iterators

Each container defines these iterator member functions:

```
iterator begin( );
const_iterator begin( ) const;
iterator end( );
const_iterator end( ) const;
```

Using Iterators

```
list<double> L;
L.push_back(1.2);
L.push_front(3.4);
L.insert(L.begin(), 5.6);
L.insert(L.end(), 7.8);



list<double>::const_iterator iter;
for (iter = L.begin(); iter != L.end(); ++iter)
    cout << *iter << " ";
cout << endl;
```