Standard Template Library II

Reminder

- Final exam
  - The date for the Final has been decided:
  - Saturday, November 16th
  - 8am – 10am
  - 01-2000

Announcement

- Exam 2
  - Has been moved to Monday October 28th

Project

- Questions?
- Farmer Problem: due Oct 30th

New plan

- Today: STL 2
- Tuesday: IOStreams 1
- Thursday: IOStreams 2
- Monday: Exam 2

The Standard Template Library

- A general-purpose C++ library of algorithms and data structures
- Based on a concept known as **generic programming**
- Implemented by means of the C++ **template** mechanism
- Part of the standard ANSI C++ library
- util package for C++

STL Components

- containers
  - classes that hold stuff
- iterators
  - Used to iterate through containers
  - Generalization of C++ pointers
- generic algorithms
  - Templated functions

STL Components

- function objects (Functors)
  - Objects that overload operator();
  - Substitute for pointers to functions
  - Beyond the scope of this course
- adaptors
  - adapt other components to special purposes.
  - Queues and stacks are adaptors
- Allocators
  - encapsulate a memory model.
  - decouple the algorithms from assumptions about a particular model.

Plan for Today

- More complex containers
- Algorithms
- Why use STL?
- Reading the docs

Sorted Containers

- Objects are maintained in sorted order
- Requires Comparator function
- Examples
  - Set – Collection of unique values
  - Multiset – Collection of non-unique values

Sorted Containers

```
set<int, less<int> > s;
multiset<int, less<int> > ms;

for (int i = 0; i < 10; i++) {
           s.insert(i); s.insert(i * 2);
           ms.insert(i); ms.insert(i *
  2);
}
```

s = 0 1 2 3 4 5 6 7 8 9 10 12 14 16 18
ms = 0 0 1 2 2 3 4 4 5 6 6 7 8 8 9 10 12 14 16 18

Associative Containers

- Associates a key object with a value object (Dictionary)
- Container holds a `pair` of objects
  - Accessed via predefined `value_type`
- Examples
  - map
  - hash_map

Maps

```
map<string, int, less<string> > mymap;

mymap.insert(value_type(string("January"), 31));
mymap.insert(value_type(string("February"), 28));

map<string, int, less<string>>::iterator it =
    mymap.find (string "January");
map<string, int, less<string>>::value_type V = (*it);

cout << V.first(); // prints out key (January)
cout << V.second(); //prints out value (31)
```

typedef

- Means to define a new typename
- Makes Template types more manageable

```
typedef definition typename
```

Maps

```
typedef map<string, int, less<string> > monthmap;
monthmap mymap;

mymap.insert(value_type(string("January"), 31));
mymap.insert(value_type(string("February"), 28));

monthmap::iterator it = mymap.find (string "January");
monthmap:: value_type V = (*it);

cout << V.first(); // prints out key (January)
cout << V.second(); //prints out value (31)
```

Bitsets

- space-efficient support for sets of bits.
- operator[] overloaded to provide access to individual bits
- NOT the same as a vector of `bool`

Bitsets

```
bitset<16> b1("1011011110001011");
bitset<16> b2;

b2 = ~b1;

for (int i = b2.size() - 1; i >= 0; i--)

     cout << b2[i];
```

strings

- `strings` are actually the char instantiation of the STL `basic_string` template (which is a container class).

```
template <class charT,
    class traits = char_traits<charT>,
    class Allocator allocator<charT> >
class basic_string;

typedef basic_string <char> string;
```

basic_string

- Provides capabilities:
  - Compare
  - Append
  - Assign
  - Insert
  - Remove
  - Replace
  - various searches
  - Iterator access

Other containers

- There are others:
  - See SGI docs for full list
- Questions?

Algorithms

- A set of commonly used templated functions.
- Many work on container objects
  - Iterators passed in to indicate positions within containers

Algorithms

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

Algorithms

```
list<int> nums;
list<int>::iterator nums_iter;

nums.push_back (3);
nums.push_back (7);
nums.push_front (10);

// Search the list
nums_iter = find(nums.begin(), nums.end(), 3);

if (nums_iter != nums.end()) { /* found */ }
else { /* not found */ }
```

Abridged Catalogue of algorithms

- **Filling & generating**
  - Fills or a range with a particular value (constant or generated)
  - `fill, fill_n, generate, generate_n`
- **Counting**
  - `count, count_if` **(counts elements w/a given value)**
- **Manipulating sequences**
  - `copy, reverse, swap, random_shuffle`

Abridged Catalogue of algorithms

- **Searching & replacing**
  - `find, find_if, find_first_of, replace, replace_if`
  - `max_element, min_element`
  - `search` **(range searches)**
- **Comparing ranges**
  - `equal, mismatch`
- **Removing elements**
  - `remove, unique` (removes duplicates)

Abridged Catalogue of algorithms

- **Sorting**
  - `sort, partial_sort, nth_element`
  - `binary_search, lower_bound, upper_bound`
  - `merge`
  - `set_union, set_difference, set_intersection`
- **Applying an operation to each element**
  - `for_each, transform`
- **Numeric Algorithms**
  - `Accumulate, partial_sum`

Functors

```
// Set up a vector
vector<int> v;

// Setup a function object
out_times_x<int> f2(2);

for_each(v.begin(),v.end(),f2); // Apply function
```

Questions

- But why use STL?

Top 5 Reasons to use STL

5. Source, 2K / Executable 1.5M

4. Who needs understandable compiler errors?

Top 5 Reasons to use STL

```
"/opt/SUNWspro/SC5.0/include/CC/./algorithm.cc", line
  1015: Error: The operation "std::list<int,
  std::allocator<int>>::iterator - std::list<int,
  std::allocator<int>>::iterator" is illegal.
"/opt/SUNWspro/SC5.0/include/CC/./algorithm", line 776:
  Where: While instantiating
  "std::__final_insertion_sort<std::list<int,std::alloc
  ator<int>>::iterator>(std::list<int,
  std::allocator<int>>::iterator, std::list<int,
  std::allocator<int>>::iterator)".
"/opt/SUNWspro/SC5.0/include/CC/./algorithm", line 776:
  Where: Instantiated from non-template code.
```

Top 5 Reasons to use STL

5. Source, 2K / Executable 1.5M

4. Who needs understandable compiler errors?

3. Who needs understandable linker errors?

2. Why make your program look overly complicated when STL can do it for you?

1. Now there's a ***standard*** way to access elements beyond the bounds of an array!

But seriously…

- Lots of functionality
- Efficiency
  - very good performance at low run-time space cost
  - Generalized algorithms are only provided when their efficiency is good.
  - The implementation of the containers and algorithms of the STL is not specified in the standard, but the efficiency of each algorithm is.

Reading the docs

- There's no standard STL docs
  - No `man` for STL
  - Except, perhaps the ANSI spec
  - Docs on webpage
    - From STL vendors
      - Roguewave
      - SGI – categorizes templates

Summary

- Advanced STL Containers
- Algorithms
- Top 5
- Reading docs

- Questions?
- Tomorrow: IOStreams