Standard Template Library II

Logistics

- Project
  - Part 1 (clock and design) due Sunday, Sept 25th
  - Part 2 (water) due Sunday, Oct 16th
  - Partners for Parts 2-3
- Questions?

Logistics

- Important date:
  - THURSDAY is Exam 1
- Will cover:
  - C++ environment / architecture
  - C++ variables, pointers, references
  - Aggregates (Arrays, struct) static, const
  - Makefiles
- Will not cover
  - Classes
  - Operator overloading
  - Constructors, Destructors, operator=
  - Templates

Exam 1

- Review session
  - Wednesday, Sept 28th
  - 9-10am / 8-9pm
  - 70-3435

Before we begin

- Any questions?

The Plan

- Today: STL 1
- Wednesday: STL 2
- Thursday: Exam 1
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
• 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
• Function

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

```cpp
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 5 6 6 7 7 8 8 8 9 9 10 10 12 12 14 14 16 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

typedef
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));

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)

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)

Maps

• Of course, one can always use the [] operator to make life easier…

Maps

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

mymap ["January"] = 31;
mymap ["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

```
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;
```

Other containers

- There are others:
  - See online docs for full list

- Questions?

Basic_string

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

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;
}

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
  • Takes predicate as argument! FUNCTOR
• 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
  • Takes operation as argument! FUNCTOR!
• Numeric Algorithms
  – Accumulate, partial_sum

Functors
• Function Objects (or Functors)
  – Objects that represent functions
  – Overrides operator()
  – Functors are Templated
  • But why bother when you have functions?
    • Remember maps
      map<string, int, less<string> > mymap;
Functors

```cpp
template <class T> struct less : public
    binary_function<T, T, bool> {
    bool operator() (const T &x, const T &y) const
    { return x < y; }
}
```

Note: will only work if < is defined for T.

Functors

### STL Predicates (returns bool)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equal_to</td>
<td>arg1 == arg2</td>
</tr>
<tr>
<td>not_equal_to</td>
<td>arg1 != arg2</td>
</tr>
<tr>
<td>greater</td>
<td>arg1 &gt; arg2</td>
</tr>
<tr>
<td>less</td>
<td>arg1 &lt; arg2</td>
</tr>
<tr>
<td>greater_equal</td>
<td>arg1 &gt;= arg2</td>
</tr>
<tr>
<td>less_equal</td>
<td>arg1 &lt;= arg2</td>
</tr>
<tr>
<td>logical_and</td>
<td>arg1 &amp;&amp; arg2</td>
</tr>
<tr>
<td>logical_or</td>
<td>arg1</td>
</tr>
<tr>
<td>logical_not</td>
<td>!arg1</td>
</tr>
</tbody>
</table>

Functors

### STL Arithmetic functors

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plus</td>
<td>arg1 + arg2</td>
</tr>
<tr>
<td>minus</td>
<td>arg1 - arg2</td>
</tr>
<tr>
<td>multiplies</td>
<td>arg1 * arg2</td>
</tr>
<tr>
<td>divides</td>
<td>arg1 / arg2</td>
</tr>
<tr>
<td>modulus</td>
<td>arg1 % arg2</td>
</tr>
<tr>
<td>negate</td>
<td>- arg1</td>
</tr>
</tbody>
</table>

Functors and Algorithms

- Note that generic algorithms do not care if the operation argument is
  - A function
  - A pointer to a function
  - Functor

Questions

- But why use STL?
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
- Functors

- Questions?
- Thursday: Exam 1