CSE 303
Concepts and Tools for Software Development

Magdalena Balazinska
Winter 2010
Lecture 20 – C++: Templates and STL

Where We Are

- We are almost done talking about C++
  - Still need to talk about templates and STL

- So what are we going to do for the rest of the quarter?
  - Software engineering basics
    - Unit testing, stubs, specifications
    - Writing robust and readable code
  - Societal implications
  - A few extra things: threads and (maybe) profilers

Introduction to Templates

- Motivation: often want to perform the same operations on different data types
- Example: storing data in a linked list
  - Solution 1: Create a new list class for each data type we want to store in a list
  - Solution 2: Force all data types to have a common ancestor X and create a list of X (Java solution)
  - Solution 3: Create a generic list class, and have the compiler use that generic class as a *template* to generate code for all the list classes we need
    - Note: this is DIFFERENT from Java generics

C++ Templates Basic Idea

- With a single code segment, define a whole group of related *functions* or *classes*
- From the template, the compiler **generates** the code for all actual functions or classes
  - C++ templates are said to be implemented "by expansion"
- The generated code is then compiled

Syntax for Class Templates

- Class definition in `.h` file

```
template < class T >
class MyClass {
  // Here use T like ordinary type
  bool test(T item);
};
```

- Function definitions in the `.cc` file

```
template < class T >
bool MyClass<T>::test(T item) {
  // here use T like ordinary type
};
```

Syntax for Using Class Templates

```
MyClass<int> example1;
example1.test(3);

MyClass<char> example2;
example2.test('b');

...
```

- Full example in file `template.cc`

Standard Template Library

- C++ library of:
  - Basic data structures (i.e., container classes)
    - Lists, Maps, Sets, etc.
  - Iterators for traversing these containers
    - Iterators are a generalization of pointers
  - And basic algorithms to operate over various containers: sort, reverse, etc.
    - Algorithms are decoupled from specific containers
    - They are templates parameterized by the type of iterator
- We will only consider two concrete examples
  - `list` in lecture and `map` in assignment

Example: List of Integers

```
#include <list>
[...]

  list<int> my_list;
  for ( int i = 0; i < 10; i++) {
      my_list.push_back(i);
  }

  list<int>::const_iterator i;
  for ( i = my_list.begin();
        i != my_list.end(); ++i ) {
      cout << "Element is " << (*i) << endl;
  }
```

- Other example in file `main.cc`

Java Generics

- Very different from C++ templates and STL
  - Ex: generic collections classes are based on std Java collections classes where everything is a container of Objects
- Java generics are implemented by "type erasure"
  - Compiler reads type information
  - Compiler performs type checks
  - Compiler automatically generates type casts
  - Compiler erases any type information
  - So the resulting bytecode is the same as without using generics, but traditional collections classes
- Goal in Java was backward compatibility

No Templates nor STL on Final

- Templates and STL are an advanced topic
- We overview them briefly because they are very frequently used in C++
- But there will be no question about templates nor STL on the final

Readings

- Carefully study the code that accompanies today's lecture

- Standard Template Library Reference
  - http://www.sgi.com/tech/stl/