Unix Tutorial Slides - Templates

CSE 326 Quiz Section
April 4, 2002

With much thanks to the UW ACM

C++ Templates - Introduction

- Templates are cookie-cutters with which the compiler generates real C++ code. *Templates themselves do not exist.*
- When a template is used, (that is, specialized for a specific type), it get *instantiated*. This is when actual machine code is generated.
- The instantiation creates a version of the template where each placeholder is replaced by its specialization. At this point, the specific version of the template comes into existence and can be compiled. *It does not exist otherwise!*
- In a very real way, a template just does a search and replace for each type you *specialize* the template for. It's just done for you, behind your back.

These template slides are freely stolen from Albert Wong (awong@cs) http://www.cs.washington.edu/orgs/acm

C++ Templates - Problems

- Problem:
  - Because templates do not really exist, *they don't exist to the compiler until they are instantiated!*
- Effects:
  - Template code will not get compiled until it is used (ie, instantiated). Thus, *the compiler will not catch syntax errors until the template is used!*
  - A specialization (a place where the template is actually used) instantiates all *relevant* templated code before it.
    - If templated code *occurs in a different file*, it will not get instantiated by that specialization.
    - If templated code *occurs later in the same file*, it will not get instantiated by that specialization.
  - Worse yet: implicit template instantiation
    - Only *explicitly used* templated code will be instantiated.
    - Thus, if templated code *occurs in the same file and before the template specialization*, it still will not get instantiated by that implicit specialization!

C++ Templates - Possible Problem I

Will this code compile?

```
/* Array.hh */
#ifndef ARRAY_HH
#define ARRAY_HH

template <typename T>
class Array {
   Array( int i ) {
      b = "Hello Mom!";
   }
};

#endif /* ARRAY_HH */
```

- Unfortunately, yes. Although `b` is undeclared, no warnings or errors will be generated.
- It appears to "compile" because nothing actually instantiates the template, so the compiler never sees the template code.

C++ Templates - Possible Problem II

Will this program link?

```
/* main.cc */
#include <iostream>
using namespace std;

#include "Array.hh"

int main(void) {
   Array<int> a(10);

   cout << a.GetCapacity()
        << endl;

   return 0;
}
```

The link error happens at `a.GetCapacity()`

- **Nothing** from a template gets instantiated until it is either used or explicitly instantiated.
- `Array<int>::GetCapacity()` is used at `a.GetCapacity()`, but the function definition is not in this file.
- The definition of the function is in Array.cc, but it is never used there.

Thus the definition of `Array<int>::GetCapacity()` never gets instantiated and compiled to object code.

C++ Templates - Avoiding the Problems

- There are 3 conventions to avoiding template problems:
  - Write all the code inline in the .h file
  - Write the code in two files, but #include the *implementation file* at the bottom of the .h (essentially the same as above)
  - Write the templated class as you would with a normal class (using a header and implementation file)
    - Create a new source file and #include the *implementation* file there.
    - Inside this new file, explicitly instantiate all your templates.
    - This new instantiation file is the one that you compile, *not* the implementation file.
- Advantages of the third method:
  - It is not necessary to recompile *all* the code that uses the template (just the template itself).
  - The third method instantiates the entire templated class all at once, removing potential link problems.

C++ Templates - The Safe Way

The proper procedure

- Write the template, separated into a header and an implementation file
- Create an instantiation file for the template which includes the implementation file.
- Compile the instantiation file and *not* the template implementation file.
- The instantiation file generates the object code for the template.

This line forces the instantiation of the Array class template (and all its member functions), for `int`s.

```
/* ArrayInst.cc */
#include "Array.cc"

template Array<int>;
template Array<double>;
```

- To make the previously broken program link properly, explicitly instantiate an integer version of the Array template.
- Remember, each specialization will require their own line in the instantiation file
- Compile line:

```
g++ -Wall -ansi main.cc ArrayInst.cc
```