

Templates I

CMSC 202

Warmup

- Define a class that represents an index out of bounds exception
 - Your class should have:
 - Data member that is the index requested
 - Data member that is the function name that throws the exception
 - Data member that is the vector/array that the index was out of bounds on

Recall...

- Polymorphism
 - "Many shapes"
- Types seen so far?
 - Ad-hoc
 - Functional overloading
 - Dynamic (true)
 - Virtual member functions, dynamic binding
- What's left?
 - Parameterized
 - Parameter-based (type based), static binding
 - Function & class-based templates

Problem?

- Common algorithms/actions for all/many types
 - Swap
 - findMax/Min/Worst/Better
 - Sort
 - search

Imagine...

```
float max ( const float a, const float b );
int max ( const int a, const int b );
Rational max ( const Rational& a, const Rational& b );
myType max ( const myType& a, const myType& b );
```

Code for each looks the same...

```
if ( a < b )
    return b;
else
    return a;
```

We want to reuse
this code for ALL
types!

Templates

Fundamental idea

- Write one implementation
- Use for any type
- Compiler generates appropriate code

Important!
Wherever you would
usually use the type
of the templating
object, you use T
instead!
T can be any identifier
you want

Syntax

```
template <class T>
retType funcName ( ..., T varName, ... )
{
    // some code...
}
```

Template Example

Function Template

```
template <class T>
T max ( const T& a, const T& b)
{
    if ( a < b )
        return b;
    else
        return a;
}
```

Notice how 'T' is mapped to 'int' everywhere in the function...

Compiler generates code based on the argument type
 cout << max(4, 7) << endl;

Generates the following:

```
int max ( const int& a, const int& b)
{
    if ( a < b )
        return b;
    else
        return a;
}
```

A Closer Look...

Function Template

```
template <class T>
T max ( const T& a, const T& b)
{
    if ( a < b )
        return b;
    else
        return a;
}
```

- Notice
 - Types that you want to use with this function must support the operator<
 - Compiler will give you an error if this operator is not supported

New variables of type T?

- Let's think about Swap()
 - There is a templated swap() already defined for your use...
- What might it look like?

```
template <class T>
void Swap ( T& a, T& b)
{
    T temp;
    temp = a;
    a = b;
    b = temp;
}
```

Assuming the code:
 double x = 7.0;
 double y = 5.4;
 Swap(x, y);

Compiler generates:
 void Swap (double & a, double & b)
 {
 double temp;
 temp = a;
 a = b;
 b = temp;
 }

What's wrong here?

```
template <class T>
T max ( const T& a, const T& b)
{
    if ( a < b )
        return b;
    else
        return a;
}
```

- Assume the code:

```
char* s1 = "hello";
char* s2 = "goodbye";
cout << max( s1, s2 );
```

Compiler generates:

```
char* max ( const char*& a,
            const char*& b)
{
    if ( a < b )
        return b;
    else
        return a;
}
```

Is this what we want?

How can we fix this?

- Create an explicit version of max to handle char*'s
 - Compiler will match this version and not use the template...

```
char* max(char *a, char *b)
{
    if (strcmp(a,b) < 0)
        return b;
    else
        return a;
}
```

Compiling Templates

- First trick...
 - Since compiler generates code based on function call...
 - If you don't actually CALL a templated function, it MIGHT not get compiled!
 - Or it might only get a general syntax check without strong type-checking...
- As you create templated functions...
 - Create a "dummy" main to call the function
 - Similarly with templated classes...

Practice

- Implement a templated function that
 - Searches a vector of some type
 - Finds the minimum element
 - You may assume the operator< is defined
 - Returns that element

Challenge

- Create a templated function
 - Sorts a vector of a templated type
 - Use any style of sort you like
 - Quicksort
 - Linear
 - Insertion
 - Merge
 - Bubble
 - Assume that operator> and operator< are overloaded
 - (so that you can use either...)
 - Try and do it in the fewest lines of code!
