CS 368 Announcements
Wednesday, November 27, 2013

Program p4 - due today, 11/27, at 10:00 pm

Program p5 – due Wednesday, 12/11, at 10:00 pm – 10%

Last Time
- string class
- C strings
- C I/O
- manipulators

Today
- start Ch. 7 (Templates)
- templated functions
- templated classes
- compiling with templates
- more template features

Next Time
- start Ch. 10 (Collections: The STL)
- STL (containers)

Templated Functions

A function template is not a function but a pattern for what could become a function

How to write a function template:

```
template < generic_type_list >
// rest of function definition using types
// listed in the generic_type_list
```

where *`generic_type_list`* is a comma-separated list of

```
class name              or
typename name
```

Examples of Templated Functions

Example: `minimum`

```
int minimum( int x, int y ) {
    return (x < y) ? x : y;
}
```

Example: `swapIt`

Using Templated Functions

```
int x1 = 5, y1 = 9;
double x2 = 3.2, y2 = 9.7;
string s1("hello"), s2("goodbye");

cout << minimum(x1, y1) << endl;

cout << minimum<double>(x2, y2) << endl;

cout << minimum(s1, s2) << endl;


swapIt(x1, y1);

swapIt(x2, y2);

swapIt(s1, s2);

cout << "After swapIt, x1 = " << x1
     << ", y1 = " << y1 << endl;
cout << "After swapIt, x2 = " << x2
     << ", y2 = " << y2 << endl;
cout << "After swapIt, s1 = " << s1
     << ", s2 = " << s2 << endl;
```

Templated Classes

```
template <typename Object>
class ObjectWrapper {

  public:

    ObjectWrapper(const Object & initValue = Object() ) :
      value(initValue) { }

    const Object & getValue() const {
        return value;
    }

    void setValue( const Object & newValue );

  private:

    Object value;
};


template <typename Object>
void ObjectWrapper<Object>::setValue(
                              const Object & newValue ) {
    value = newValue;
}
```

Compiling Templated Functions and Classes

What happens at compile time:

Compiling Templated Functions and Classes (cont.)

Separate compilation

Inclusion model

Special Template Features

Multiple template parameters

```
template <typename KeyType, typename ValueType>
class Map {
    ...
};
```

Specialized templates

```
template <typename T>
const T & minimum(const T & x, const T & y) {
    return (x < y) ? x : y;
}
```

Template non-type parameters

```
template <typename Object, int size>
class Buffer {
    ...
  private:
    Object buf[size];
};
```