CS2141 – Software Development using C/C++

Templates
Defining Templates

- A *template* allows a class or function to be parameterized by one or more data types
  - All templates start with the keyword `template` followed by a list of template parameters
  - Each template parameter is preceded by the keywords `class` or `typename`

```plaintext
    template <typename T>
    or
    template <class DType>
    or
    template <typename V, typename B>
```
Defining Templates cont.

• A template parameter is then used as a data type through the rest of the template definition:

```cpp
template <typename T> class Box {
  public:
  Box( T v ) : val( v ) { }
  Box( Box<T> & b ) : val( b.val ) { }
  T value( ) const { return val; }

  void operator=( T r ) { val = r; }
  void operator=( Box<T> & r ) { val = r.val; }

  private:
  T val;
};
```
Using a Template Class

- To create instances of a template class, data types must be passed as parameters:

```cpp
Box<int> intBox;
Box<double> dblBox;
...
Box<int> * intBoxPtr = new Box<int>();
```

- After that, an object created from a template class works like any other object:

```cpp
*intBoxPtr = 7;
dblBox = 3.45;
int num = intBox.value();
```
Using a Template Class cont.

- Any data type can be used, including user-defined types:

```
// Suppose there are classes Duck and Cow...
Duck daffy;

Box<Cow> cowBox;
Box<Duck> duckBox( daffy );

Cow betty;
cowBox = betty;
Box<Cow> cowBoxTwo( cowBox );
```
Advantage of Templates

- Templates are put together at compile time
  - This allows static type checking:
    ```
    // Both cause compile errors
    duckBox = betty;
    Cow dorthy = intBox.value();
    // Converting to int without cast
    // causes compiler warning
    int val = dblBox.value();
    ```

- Templates avoid all the dynamic casting that would be involved with using polymorphism
Template Methods

- Methods defined outside a template class definition also need template parameters

- A separate implementation of the `value` method from the example `Box` class looks like this:

  ```cpp
  template <typename T>
  T Box<T>::value() const {
    return val;
  }
  ```

- Template methods should be implemented in the header file rather than in a separate file
Template Functions

- Functions can also be written in template form:

  ```cpp
  template <class T>
  void swap( T & left, T & right ) {
    T tmp = left;
    left = right;
    right = tmp;
  }
  ```

- The data type is inferred from the parameters:

  ```cpp
  int a = 1, b = 4;
  swap( a, b );

  Duck daffy, donald;
  swap( daffy, donald );
  ```