CS 1704
Templates

Introduction to Data Structures and Software Engineering

What are Templates?

- A template is a pattern
  - Can apply to functions or classes
  - Expanded as needed to provide actual code
- Typedefs can provide a single type expansion
- Templates can provide multiple type expansions

Function Templates

- Allow you to perform similar operations on different types of data
- Supports code reuse since don't have to write the same routine for every datatype that you need operations for

Function Template Example

```
// Assumes << operator for Etype
template<typename Etype>
void PrintArray(const Etype array[], const int count)
{
    int i;          //array index
    for  ( i = 0; i < count; i++)
      cout << array[i] <<  " ";

    cout << endl;
}
```

Using template functions

```
void main ()
{
    int a[] = {1, 2, 3, 4, 5};
    float b[] = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7};
    char c[]="Hello";

    PrintArray(a, 5);
    PrintArray(b,7);
    PrintArray(c,6);
}
```

Issues in using templates

- Template may make assumptions about operations which are supported for the type (They should document these assumptions in the header comments)
- Some errors are found at the time the template is parsed (compile time), some at the time the template is instantiated (link time)

Another Example

```
// Assume assignment operator for Etype
template<typename Etype>
void SwapValues(Etype& a, Etype& b)
{
        Etype temp = a;
        a = b;
        b = temp;
        return;
}
```

Using Swap

```
void main ()
{
        int a=1, b=2, c=3, d=4;
        double e=5, f=6;
        char g='a',h='b';
        Circle c1(3,4,5,TColor::Black), c2(1,2,4,TColor::Red);

        SwapValues(a,b);      // ints
        SwapValues(c,d);      // ints
        SwapValues(a,e);      // Illegal - args must be same type
        SwapValues(c1,c2);    // Works for any type!
        SwapValues(g,h);      // char instantiation
        cout << a << b << c << d << e << f << g << h << endl;
        return;
}
```

Template Classes

- Same syntax used to specify template classes
- Template type can be used throughout the class in member functions, return types, member data, etc.
- Assumptions on type still need to be documented

Rules for Template Classes

- All operations defined outside of class declaration must be template functions
  - Each method needs its own template declaration
- Any use of template class name as type must be parameterized.
- Operations on a template class should be defined in the same file as the class declaration
  - interface and implementation are usually both put in the .h file

Template Class Example

```
template <typename Etype>
class Stack {
  public:
   Stack( );
   ~Stack( );
   const Stack &      operator=( const Stack & Rhs );
   void               Push( const Etype & X );
   void               Pop( );
   const Etype &      Top( ) const;
   bool               IsEmpty( ) const;
   bool               IsFull( ) const ;
   void               MakeEmpty( );
```

Example (cont)

```
private:
   int MaxSize;
  int TopOfStack;
  Etype *Array;
};
```

Member functions of template classes

```
template <typename Etype>
Stack<Etype> :: Stack( )
        : MaxSize( 12), TopOfStack( -1 )
{
   Array = new Etype[ MaxSize ];
}
```

Another Member Function

```
template <typename Etype>
const Etype & Stack<Etype>::Top( ) const
{
   return Array[ TopOfStack ];
}
```

Push and Pop Functions

```
template <typename Etype>
void Stack<Etype>::Push( const Etype & X )
{
   Array[ ++TopOfStack ] = X;
}

template <typename Etype>
void Stack<Etype>::Pop( )
{
   TopOfStack--;
}
```

IsEmpty and IsFull Functions

```
template <typename Etype>
bool Stack<Etype>::IsFull(void) const
{
   return ( TopOfStack + 1 == MaxSize );
}

template <typename Etype>
bool Stack<Etype>::IsEmpty(void) const
{
   return ( TopOfStack  == -1);
}
```

MakeEmpty Function

```
template <typename Etype>
void Stack<Etype> :: MakeEmpty( )
{
   TopOfStack = -1;
   delete Array;
}
```

Array-Based Stack Class Template Example

- Note how each of the rules is followed.
  - Stack.h contains all code, declarations and definitions for the Stack Class Template.
- Note on "in-line" functions
  - increases speed of execution.