ECE2036: Week 12 - Class and Function Templates

Function and class templates can be used to create a generic set of data structures and functions that can have great flexibility in their use. One of the key programming concepts with generic programming is to ***maximize code reuse***.

Basic Syntax for Function Templates

```
template <typename T>

functiontype functionName (T item)
{

//The functionType is void, int, etc...
// This is the body of the function.  The
//idea is that this template function is useful
//when the functionality remains the same,
// BUT the datatypes can change.

}
```

Basic Syntax for Class Templates

```
template <typename T>
class ClassName
{

//T is a type that is used as declarations
// for data members and member functions

};
```

Basic Syntax to Instantiate Class Object with Templates

- className <type> myObject

```
//------ main function in main.cc

#include <iostream>
#include "stack.h"
using namespace std;

int main()
{
  Stack <double> doubleStack(5);
  double doubleValue = 3.14159;

  while (doubleStack.push(doubleValue))
  {   cout << "push" << endl;
     doubleValue += 3.14159; }

  while (doubleStack.pop(doubleValue))
  { cout << doubleValue << " "; }

  cout << endl;

}// end main
```

ECE2036: Week 12 - Class and Function Templates

```
//--- class template and member function BOTH in stack.h file!!

template <typename T>
class Stack
{
  public:
  Stack(int =10); //default constructor
  ~Stack() {  delete [] stackPtr; }
  bool push(const T &);

  bool pop (T & );

  bool isEmpty() const
  { return (top == -1); } //This is condition for being empty

  bool isFull() const
  { return (top == size-1);} //stack

  private:
  int size;
  int top;
  T * stackPtr;
}; // end class Template


//---------------------- If the functions are outside the
//---------------------- class interface they must have template
template <typename T>
Stack<T>::Stack (int s): size(s>0?s:10), top(-1), stackPtr(new T[size]) {}

//----------------------
template <typename T>
bool Stack<T>:: push (const T & pushValue)
{
  if (!isFull())
  {
    stackPtr[++top] = pushValue; //place on stack
    return true; //push succesfull
  }
  return false;
} //end push
//----------------------

template <typename T>
bool Stack<T>::pop (T & popValue)
{
  if (!isEmpty())
  {
    popValue = stackPtr[top--]; //remove from stack;
    return true; //successfully popped!
  }
  return false;
}//end of pop
```