An Application of Classes

A Stack Class

Problem: Create an integer stack class

- The stack class will implement the functions of the stack data structure
  - push( )
  - pop( )

Solution: an integer Stack

- public interface methods:
    - initialize a Stack object
    - check whether a Stack is empty or full
    - to push integers onto a Stack
    - to pop integers from a Stack

```
class Stack{
public:
  enum {MaxStack = 5};
  void init( ) {top = -1;}
  void push( int n ){
    if ( isFull( ) ) {
      cerr << "Full Stack. DON'T PUSH\n";
      return; }
    else {
      arr[ ++top ] = n;
      return;}
  }
  int pop() {
    if (isEmpty( ) ) {
      cerr << "\tEmpty Stack. Don't Pop\n\n";
      return 1;
    }
    else
      return arr[top--]; }
  bool isEmpty() {return top < 0 ? top : -1;}
  bool isFull() {return top >= MaxStack -1 ? top : 0;}
  void dump_stack() {
  cout << "The Stack contents, from top to bottom, from a stack dump are: " <<
endl;
    for (int i = top; i >= 0; i--)
      cout << "\t\t" << arr[i] << endl;
  }
private:
  int top;
  int arr[MaxStack];
};                    //End class
```

Class Templates

- In object-oriented programming, *inheritance* and *encapsulation* are methods to reuse code.
    - However, we have another method of code reuse: *Templates*
- We just looked at the Stack class, from Stack.cpp
    - This is an example of a *container class*, that is a class designed to hold other data types or objects.

Class Templates

- From your earlier studies, you defined functions and created header files.

✓ This is good programming practice because it promotes code reuse!

  ✓ But there is a drawback: You have to edit the header file every time you have to change the data type.

Class Templates

- So, to get around this, use a C++ Template (or class template)
  - Using our Stack.cpp program, we will change the program to use a class template

Class Templates

Using the keyword template:

```
template <typename  Type>
int arr[MaxStack];
```

now becomes

```
Type arr[MaxStack];
```

However, similarly we can replace the class methods of stack class with template member functions:

```
template <class Type>
```

Class Templates

```
template <class Type>  // class template

class Stack{
public:
  enum {MaxStack = 5};
  void init() {top = -1;}
  void push( Type n ){    // Notice the parameter Type
    if ( isFull() ) {
      cerr << "Full Stack. DON'T PUSH\n";
      return;
    }
    else {
      arr[ ++top ] = n;
      return;}
  }
  int pop() {
    if (isEmpty() ) {
      cerr << "\tEmpty Stack. Don't Pop\n\n";
      return 1;
    }
    else
      return arr[top--];
  }
  bool isEmpty() {return top < 0 ? top : -1;}
  bool isFull() {return top >= MaxStack -1 ? top : 0;}
  void dump_stack() {
    cout << "The Stack contents, from top to bottom, from a stack dump are: " << endl;
    for (int i = top; i >= 0; i--)
      cout << "\t\t" << arr[i] << endl;
  }
private:
  int top;
  Type arr[MaxStack];    // class Type
};
```

Now for the Driver

```
int main()
{ Stack<int> a_stack; //Note the template argument
  a_stack.init();
  a_stack.push(4);
  a_stack.push(3);
  a_stack.pop();
  a_stack.dump_stack();

  Stack<char> b_stack;  //And here
  b_stack.init();
  b_stack.push('g');
  b_stack.dump_stack();

return 0;
}
```