Templates and Inheritance

Options:
- a template may be derived from another template.
- a non-template class may be derived from a template.
- a template may be derived from a non-template.
- templates may use multiple inheritance.

We will briefly examine the first two cases here; the remaining cases are left to the reader.

A Base Template...

Recall the queue template QueueT from earlier notes:

```cpp
class QueueT {private:
    Foo buffer[Size];
    int Head, Tail, Count;
public:
    QueueT();
    void Enqueue(Foo Item);
    Foo Dequeue();
    int getSize() const;
    bool isEmpty() const;
    bool isFull() const;
    ~QueueT();
};
```

We can derive an extended queue template that adds the ability to “peek” at the element at the front of the queue:

```cpp
template <class Foo>
class InspectableQueue : public QueueT {
public:
    InspectableQueue();
    Foo Inspect();
    ~InspectableQueue();
};
```

Using InspectableQueue Template

```cpp
InspectableQueue<Location> Path;
Location loc1(...);
Location loc2(...);
...
Path.Enqueue(loc1); // base class method
Path.Enqueue(loc2); // base class method
...
Location Front = Path.Dequeue(); // base class method
Location newFront = Path.Inspect(); // derived class method
```
Recalling the linked list template, `LinkListT`, discussed earlier, we can derive a `Polygon` class from it:

```cpp
class Polygon : public LinkListT<Location>
{
public:
    Polygon();
    void WidderShins(); // sort points
    void Print(ostream& Canvas); // draw itself
};
```

Note that the template `LinkListT` is elaborated with a specific type.

Using the `Polygon` Class

```cpp
Polygon Hex;
...
Hex.AppendNode(Location(20,20)); // inherited from
Hex.AppendNode(Location(30,30)); // LinkListT template
// insert other locations
Hex.Print(cout); // derived class method
```