Templates in C++

Templated Functions

- Ever feel you're coding the same function over and over again?
- For example:

```
int incMe(int &val)
{
  val+=50;
  return val;
}
```

```
double incMe(double &val)
{
  val+=50;
  return val;
}
```

```
String incMe(String &val)
{
  val+=50;
  return val;
}
```

- Note that the body is the same for each!

Templated Functions

- C++ lets you write a "template" equivalent:

```
template <class T>
T incMe(T &val)
{
  val+=50;
  return val;
}
```

the following definition is "generic", using type T

This method returns a value of type T

This method takes a parameter of type T

- Can be invoked in multiple ways:

```
int x=4;
x = incMe<int>(x);
```

```
string s = "hello";
s = incMe<string>(s);
```

```
double d = 3.14;
d = incMe<double>(d);
```

- Note that "+" must be defined for the type in <>'s !
- C++ usually lets you skip the <type> part for method calls.

Templated Classes

- C++ lets you write a “template” class as well. Consider:

```
template <class T>
class PFA
{
private:
        T *_array;
        int _currSize;
public:
        PFA(int maxSize) {_array = new T[maxSize];_currSize=0;}

        void append(const T &newValue) {_array[_currSize++] = newValue;}
        std::ostream &print (std::ostream &os)
        {
          for(int index=0; index<_currSize; index++)
                os << _array[index] << " ";
          return os;
        }
};
```

- Can now build objects of type PFA:
  - `PFA<int> intArray(250); // partially filled array of ints (250 max)`
  - `PFA<string> strs(1000); // partially filled array if strings (1000 max)`

Templates and Compilation

- Unfortunately, template class files cannot be compiled into object files
  - cannot know how templates will be used while compiling the template class itself
- Recommendation:
  - save template class **implementations** in a ".cct" file
  - #include the .cct file in order to use the template