TEMPLATES

ACKNOWLEDGEMENT: THE SLIDES ARE PREPARED FROM SLIDES PROVIDED BY NANCY M. AMATO AND JORY DENNY
- Programming/developing algorithms with the abstraction of types
- Algorithms/data is expressed “without type”
- The uses of the abstract type define the necessary operations needed when instantiation of the algorithm/data occurs

```cpp
template<typename T>
T Add(const T& t1, const T& t2){
    return t1 + t2;
}
```
Templates are not types, but rather they are a placeholder for a type.

At compile time, the compiler automatically “replaces” the placeholders with the concrete type:
- Closer to reality – the compiler makes a copy of the template, fills in the placeholders, and compiles the code.

C++ templates come in two flavors:
- Functions templates
- Class templates

Similar to Java’s Generics.
FUNCTION TEMPLATES

- used to define generic algorithms
  ```cpp
  int max(int x, int y){
      return x < y ? y : x;
  }
  ```

- While this is useful, it only works for integers.

- A better solution is to define a function template for `max`
  ```cpp
  template<class T>
  T max(T x, T y){
      return x < y ? y : x;
  }
  ```
Nothing special has to be done to use a function template

```c
int main(int argc, char* argv[]) {
    int a = 3, b = 7;
    double x = 3.14, y = 2.71;

    cout << max(a, b) << endl; // Instantiated with type int
    cout << max(x, y) << endl; // Instantiated with type double
    cout << max(a, x) << endl; // ERROR: types do not match
}
```

Note: all that is required of the type passed to max is the comparison operator, operator<.
- Class Templates
  - May contain data member objects of any type.

- Then instantiate the same container with objects of different types:

```cpp
template <class T>  
class myarray {  
public:  
  T* v;  
  size_t sz;  
  myarray(size_t s) { v = new T [sz = s]; }  
~myarray() { delete[] v; }  
T& operator[](size_t i) { return v[i]; }  
void set(size_t i, T val) { v[i] = val; }  
int size() { return sz; }  
};

myarray<int> iarr(10);  
myarray<shape> sarr(10);  
```
THE SKIES ARE THE LIMIT!

- Templates have an extreme amount of uses
- Can many template parameters
- Specialized templates for specific types
- Variadics