The logic of an algorithm is often applicable to many different data types. Consider:

```c
double max(double x, double y) {
    if(x >= y) {
        return x;
    }
    else return y;
}

int max(int x, int y) {
    if(x >= y) {
        return x;
    }
    else return y;
}

char max(char x, char y) {
    if(x >= y) {
        return x;
    }
    else return y;
}
```
C++ Function Templates

- C++ allows us to write functions that work on any data type
- Such functions are known as generic, or template functions

```cpp
template <class T>
T max(T x, T y) {
    if(x >= y) {
        return x;
    }
    else return y;
}
```

Instead of using a concrete type such as `int`, we use `T`: a placeholder that can be replaced with any type.
FUNCTION TEMPLATES

How is it going to compile?
Based on function calls:

- If $\text{max}(T,T)$ is never called, it will not be compiled at all.
- If $\text{max}(T,T)$ is called on two doubles, $T$ will be substituted with $\text{double}$, and $\text{max(double, double)}$ will be compiled.
- I.e., all the necessary “concrete” versions of the template function $\text{max}$ will be generated by the compiler automatically, based on the function calls encountered.
FUNCTION TEMPLATES

- double a = 37.2;
- double b = 20.25;
- cout << max(a, b) << endl;

- int c = 5;
- int d = 10;
- cout << max(c, d) << endl;

- char one = 'A';
- char two = 'Z';
- cout << max(one, two) << endl;

How many times will max compile?

3 times, once for each data type
FUNCTION TEMPLATES & CLASS TYPES

- Can we substitute a class type?
- I.e., can we do the following:
  - Cube a(2); // cube with side = 2
  - Cube b(3); // cube with side = 3
  - cout << max(a, b) << endl;
- Consider the template function again:

```cpp
template <class T>
T max(T x, T y)
{
    if(x >= y) {
        return x;
    }
    else return y;
}
```

What must a class implement to be able to use the generic function max?
FUNCTION TEMPLATES & CLASS TYPES

You can substitute any type – primitive type or class type, as long as all operators used in the template function are implemented for the type.

Alternative notation:

```
template <class T>
T max(T x, T y)
{
    if(x >= y) {
        return x;
    }
    else return y;
}
```

```
template <typename T>
T max(T x, T y)
{
    if(x >= y) {
        return x;
    }
    else return y;
}
```

Any other letter/word can be used in place of T.
FUNCTION TEMPLATES

- You can use more than one template type if you need:

```cpp
template <class M, class N>
void print(M var1, N var2)
{
    cout << var1 << " and " << var2;
    cout << " look good together " << endl;
}
```

- See example code
- Can return type be a template, too?
FUNCTION TEMPLATE NOTES

- Function templates can be overloaded
- Each template must have a unique parameter list

```
template <class T>
  T someFunction(T num) ...
```

```
template <class T1, class T2>
  T1 someFunction(T1 num1, T2 num2) ...
```

- All data types specified in template prefix must be used in template definition