Templates

Rob Hackman

Winter 2025

University of Alberta

Table of Contents

The `max` function

When we first learned about C++ we talked about how we could overload functions. Our example was the following

```
int max(int a, int b) {
  return a > b ? a : b;
}
float max(float a, float b) {
  return a > b ? a : b;
}
```

What do we notice about these functions, however?

The `max` function

When we first learned about C++ we talked about how we could overload functions. Our example was the following

```
int max(int a, int b) {
  return a > b ? a : b;
}
float max(float a, float b) {
  return a > b ? a : b;
}
```

What do we notice about these functions, however?

The code is exactly the same, the only difference is the return type and parameter types!

Writing `max` for characters

What if we wanted to add a similar function that checked the values of characters? We'd have to write a whole new function, again the code being exactly the same.

```
char max(char a, char b) {
  return a > b ? a : b;
}
```

Seems silly. What else can we do though?

Introducing C++ Templates

C++ has support for type generics in the form of the language feature *templates*. Templates allow you to write a function (or class!) using a place holder name for the type your function (or class) uses. When the client programmer uses your template they specify which type they are using it for and the compiler generates the code for the appropriate function. Consider the following:

```
template <typename T>
T max(T a, T b) {
  return a > b ? a : b;
}
```

This declares a template function `max`.

Template syntax

- The keyword `template` specifies that what we are about to define is a template
- The syntax `<typename T>` are the parameters of the template. This template has one parameter the type `T`.
- A template can work on multiple parameters for example `<typename T, typename Q>`
- After that continue defining the function you desire using your template parameters to specify the types of your return values, parameters, or data as necessary.

Calling a template function — explicit paramterization

To call a template function we can explicitly instantiate the template, like so:

```
int main() {
  int a = 5, b = 10;
  int a = max<int>(a, b);
}
```

In the call the syntax `max<int>` is parameterizing the template, specifiying that `int` should be substituted for `T` in the template, the rest is a function call as before.

Calling a template function — implicit parameterization

When all of the template's parameters are used to declare the types of the functions parameters then the compiler can also *infer* the template type from the provided function arguments.

In our function `max` all of the template parameters (there is only one, `T`) are used to specify function paramaters (T is used for both a and b). So the compiler can infer from the types of arguments we supply what the template type is.

```
int main() {
  int a = 5, b = 10;
  float x = 33.3, y=203.1;
  // Compiler infers this is max<int>
  int c = max(a, b);
  // Compiler infers this is max<float>
  float z = max(x, y);
}
```

What types can you substitute for a template parameter?

Which types can we template `max` function with? Or more generally which types can be substituted for a template type?

What types can you substitute for a template parameter?

Which types can we template `max` function with? Or more generally which types can be substituted for a template type?

Depends on the template! Templates use *duck typing*.

What types can you substitute for a template parameter?

Which types can we template `max` function with? Or more generally which types can be substituted for a template type?

Depends on the template! Templates use *duck typing*.

Duck typing is the concept that "if it walks like a duck, and it quacks like a duck, then it must be a duck".

This means you can substitute any type that supports the behaviours used by your template function. In the case of `max` this means any type for which `operator>` is defined and the type is able to be returned by value (must have a public copy or move constructor, or be plain-old-data).

What types can you substitute for a template parameter?

Which types can we template `max` function with? Or more generally which types can be substituted for a template type?

What types can you substitute for a template parameter?

Which types can we template `max` function with? Or more generally which types can be substituted for a template type?

Depends on the template! Templates use *duck typing*.

What types can you substitute for a template parameter?

Which types can we template `max` function with? Or more generally which types can be substituted for a template type?

Depends on the template! Templates use *duck typing*.

Duck typing is the concept that "if it walks like a duck, and it quacks like a duck, then it must be a duck".

This means you can substitute any type that supports the behaviours used by your template function. In the case of `max` this means any type for which `operator>` is defined and the type is able to be returned by value (must have a public copy or move constructor, or be plain-old-data).