Module 21: Overloading Functions

Intro to Computer Science 1 - C++

Professor Scott Frees

Textbook

The following topic is covered in section 6.7 in the textbook

Function names v.s. signatures

You may only have one variable named “x” in your program at the same *scope*

This is because the compiler must know **which** variable you are referring to at all times

Recall - you may have several “x” variables in a program - but they’d need to be in different functions, in different parts of if/else statements, or in different loops

Function names v.s. signatures

Likewise, whenever you call a function, the compiler must be able to tell precisely which function you are talking about…

However - it has more to go on than just the name, it can also use the parameters…

```
cout << max (3, 4) << endl;
cout << max (3.5, 4.5) << endl;
```

Function signatures

A function's signature includes:

1. The return type
2. The name
3. The parameters types **(not parameter names)**

Two functions may have the same name, as long as they have either:

1. A different number of parameters
2. A different data type for at least one parameter

Determining which function

The rule is that the compiler must be able to tell which function you are using

```
double maxNumber (int x, double y) {
    ...
}
double maxNumber(double x, int y) {
  ...
}
int main() {
    cout << maxNumber(5, 6) << endl;
}
```

- Neither match identically, but the compiler can automatically turn integers into doubles.
- However, the compiler won't know which to choose -
- Should it turn the first parameter (5.0) into a double and match the second maxNumber function?
- Or should it turn the second parameter into a double (6.0) and match the top function?
- Result: compiler error

What about parameter names?

```
int something(int x, int y) {
    …
}
int something(int a, int b) {
    …
}
int main() {
    something(2, 3);
}
```

When called, we do not make any reference to parameter *names*, which is why this will always be a compiler error!

What about return types?

```
int something(int x, int y) {
    …
}
double something(int x, int y) {
    …
}
int main() {
    something(3, 4);
    cout << something(3, 4) << endl;
}
```

The return data will not necessarily help the compiler determine what you mean… thus compiler error.