

Functions: Advanced Parameter Passing

CS 1044

Parameter = Argument

I'll use these terms
interchangeably a lot.

They mean the same thing.

Formal Arguments

- ✦ When you declare/define a function, the arguments in the function header are its **formal arguments**
- ✦ Formal arguments are **placeholders** for values that will be passed into the function elsewhere

Actual Arguments

- ✦ When you call a function, the arguments passed to the function call are its **actual arguments**
- ✦ The **same number** of actual arguments must be passed to a function as it has formal arguments in its definition
- ✦ The **types** of the formal and actual arguments must also **match**, with exceptions made for default conversions (like between **int** and **double**)

Formal vs. Actual

Formal arguments

```
void add_person(string name, int age);
```

Actual arguments

```
int main()
{
    add_person("Joe Hokie", 19);

    // more code...
}
```


Concerns

- ✦ What if I want to **return more than one thing** from a function?
 - ✦ Example: A function that computes division and remainder at once
- ✦ What if I need to be able to **modify the input parameters** to a function?

Kinds of Parameter Passing

- ✦ **Parameter passing modes** determine how the formal and actual arguments are **related** or **connected**
- ✦ In C++, we can talk about three modes:
 - ✦ Pass by value
 - ✦ Pass by reference
 - ✦ Pass by constant reference

Pass by Value

- ✦ Pass by value is the default
- ✦ The formal arguments get **copies** of the actual arguments
- ✦ Inside the function, you can modify the formal arguments, but this **does not affect or change** the actual arguments

Pass by Value

```
int foo(int a, int b)
{
    a = 12;
    b = 19;
}
```

```
int main()
{
    int x = 5;
    int y = 10;
    foo(x, y);
}
```

Name	Value
a	12
b	19

Name	Value
x	5
y	10

Pass by Value

- ✦ With pass by value, the actual argument can be...
 - ✦ a literal constant
 - ✦ a variable
 - ✦ a larger expression

```
int main()  
{  
    int x = 5;  
    int y = 10;  
  
    foo(5, 10);  
    foo(x, y);  
    foo(x + y / 2, x - 1);  
}
```


Pass by Reference

- ✦ Pass by reference **links** the formal arguments to the same slots as the actual arguments
- ✦ Modifying the formal arguments **will change** the actual arguments
- ✦ Mostly used to **pass multiple values out** of a function
- ✦ To pass an argument by reference, put an ampersand (**&**) after its type in the **formal** argument list

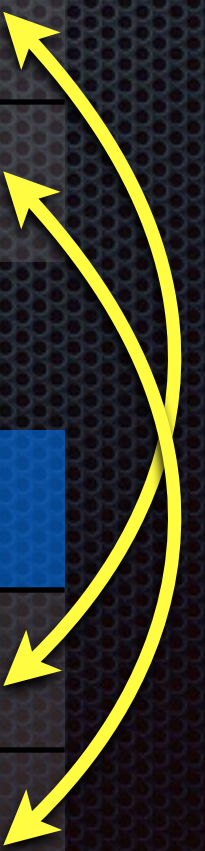
Pass by Reference

```
int foo(int& a, int& b)
{
    a = 12;
    b = 19;
}
```

```
int main()
{
    int x = 5;
    int y = 10;
    foo(x, y);
}
```

Name	Value
a	12
b	19

Name	Value
x	5 12
y	10 19



Pass by Reference Limitations

- ✦ The actual argument to a reference argument must be something that can go on the left-hand side of an assignment statement (like a variable)

```
int foo(int& a, int& b)
{
    a = 12;
    b = 19;
}

int main()
{
    foo(5, 10);
}
```

Error: foo would try to change 5 and 10; doesn't make sense

Pass by Constant Reference

- ✦ Like pass by reference, this **links** the formal argument to the same slot as the actual argument
- ✦ But, the formal argument **can't be changed**, because it's treated as constant
- ✦ Most useful when the argument is **large** (a long string or collection of data, discussed later), and copying it via pass by value would be slow/wasteful
- ✦ Do this by putting **const** before a reference argument

Pass by Constant Reference

```
int foo(const int& a, const int& b)
{
    a = 12;
    b = 19;
}
```

Compiler error: Can't modify a constant

```
int main()
{
    int x = 5;
    int y = 10;
    foo(x, y);
}
```


Mixing Parameter Passing Techniques

- ✦ All the above examples have shown all the arguments to a function passed using a certain mode
- ✦ You can, of course, **mix-and-match** by-value, by-reference, and by-constant-reference, all in the same function
- ✦ Pick the right mode for each argument, depending on its usage

References as Variables

- ✦ References can be used for more than just function parameters
- ✦ Regular variables can be declared as references
- ✦ Links the variables together in the same way; changing one changes the other
- ✦ Note: Variables must be the **same type**, and the reference **must be initialized immediately** when declared

References as Variables

```
double value = 5.09;
```

Error: Reference type must be same as variable type

```
int& valueRef = value;
```

Error: Reference must point to something immediately

```
double& valueRef;
```

```
double& valueRef = value;  
valueRef = 1.23;
```

OK – now **both** value and valueRef == 1.23