Parameters

Class 3

Items from Assignment 113

- to indicate a range like “lines 11 – 13”, use an n-dash, not a hyphen
  here is a hyphen: lines 11 - 13
- math modes inline and centered

- space after control constructs, not after function name
- line length
- tabs
- newline at the end of a file

References

- in addition to pointers, C++11 has references
- identical to Java object references when referring to objects
- can also refer to primitives as well as objects
- a reference variable is declared with an ampersand

```
int foo = 5;
int& fooref = foo; // no address-of operator for reference
int* fooptr = &foo; // must use address-of for pointer
```

- reference variables "point" to another location just like pointers
- reference variables are not dereferenced when used
- this allows simpler syntax sometimes, like for loops
- they are critical for parameter passing

example forref.cpp

Variables

so now there are three kinds of variables

1. immediate — "normal", values literally in stack frame
2. pointer — indirect, point to somewhere else, must dereference to use
3. reference — indirect, refer to somewhere else, no dereference to use

Rvalues and Lvalues

- actually, the situation is even more complicated
- C++11 and later have a fourth kind of variable
- what we have been calling a reference variable is really an lvalue reference
- Java's references are strictly lvalues
- C++11 also defines rvalue references denoted by a double ampersand: `int&& fooref = foo;`
- in this course, we will not discuss rvalues
- if you really want to be an awesome programmer who knows everything, learn about rvalues and rvalue references and brag that you're a pro

Function and Method Parameters

- under the hood, C++ (and Java) has only one parameter-passing mode: pass by copy
- the value of *something* is copied from the calling scope to the called scope, on the stack
- in practice, this leads to four different parameter- (and return-) passing modes (but we'll only use 1, 3, and 4)
  1. pass by value: copy the value itself, bit by bit, strictly for primitives, never for objects
  2. pass by pointer: old fashioned C-style for objects and for primitives that will be changed (do not use in this class!)
  3. pass by reference: the modern C++ way to pass objects and primitives that will be changed by the function
  4. pass by constant reference: the way to pass objects that the function guarantees not to change

example programs swap.cpp and findmax.cpp