Lecture 03
Const
February 3rd, 2016

Announcements
• Reading: Chapter 3 due Monday
  – Quiz before class
• Recitations this week: make
• Programming Assignment #1 is due today
• Programming Assignment #2 is one the web
  – Due next week (Wed., Feb 10)
• Questions from Piazza:
  – Blank lines
  – Are there other questions I should address?
• Winward Code Wars
  – When: Saturday, 9-4.
  – Register: codewar.winward.net
  – Info: Alexander Arbuckle alxndr@rams.colostate.edu

Once Again: An Example Class

```cpp
class Quagga {
  public:
    Quagga(double s) size(s) {};
    inline void Grow() {size *= 1.1;}
    double Size() {return size;}
    double size;
};
```

What does this print?

```cpp
#include <iostream>
#include <Quagga.h>
int main(int argc, char* argv[]) {
    Quagga quip(100);
    Quagga quak(100);
    quak.Grow();
    std::cout << quip.Size() << " " << quak.Size() << std::endl;
    return 0;
}
```

But how does it print that?

• Every method has a hidden argument
  – An instance of its object class is passed as an argument
  – Unless the method is declared static
  – The hidden argument is used to access object fields
• The following are equivalent
  – void Quagga::Grow() {size *= 1.1;}
  – void Grow(Quagga& q) {q.size *= 1.1;}

CS270 refresher

• Every function/method has a stack frame
  – Except (sometimes) inline functions
• The stack frame holds:
  – Internal data (e.g. the function pointer)
  – Memory for local variables
• The stack frame is pushed onto the stack when a function is called
• The stack frame is popped off the stack when it returns
How are arguments passed?

• Two methods:
  – Call by value
    • The value in the calling function is copied to the stack frame of the called function
  – Call by reference
    • The value in the calling function is the value in the called function
    • More formally, the argument is a reference to a value in the calling function

Rules of Thumb

• If a method does not change an argument:
  – Pass it by value
    • So readers know it doesn’t change
    • So the compiler knows it doesn’t change
  • Otherwise, pass by reference
    – Signaling the side effect
  • But wait!
    – What about efficiency?
    – What if the argument is large?

Constant Reference

• Pass by constant reference
  
  void Herd::Join(const Quagga& q)

  Promise to readers and the compiler not to change the argument

3 ways to pass arguments

• Pass by value: void Herd::Join(Quagga q);
  – Join is given copy of q
• Pass by ref: void Herd::Join(Quagga& q);
  – Join is given access to q in calling function
  – Assumption is that it modifies q
• Pass by const ref:
  void Herd::Join(const Quagga& q);
  – Join is given access to q in calling function
  – Join is prevented from modifying q

Argument Passing

• In complex code
  – Objects are passed as arguments more than primitives
  – Side effects of arguments are rare
    • Better avoided when possible
• Therefore

Most arguments are passed by constant reference