Announcements

- Reading: Chapter 3 due Monday
  - Quiz before class
- Recitations
  - this week: make
  - Next week: optional!
    - Remote access from Mac OS, Windows & Linux
    - With X11 support
- Programming Assignment #2 is due Wednesday

PA1 Discussion

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

What about Hidden Arguments?

- Every method has a hidden argument that is the object being operated on
- By default, hidden arguments are pass by reference
- What if a method doesn’t change the object it is called on?

```cpp
Int Herd::Size() const;
```

Declares the hidden argument constant

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
Accessors vs Mutators

- A method with a constant hidden argument is called an **accessor**
  - Because it doesn't change the object it accesses
- A method without a non-constant hidden argument is called a **mutator**
  - Because it can and presumably does change the object

New Term: Scope

- The scope of a variable refers to where it can be accessed.
- Local variables
  - enter scope when they are declared
  - Leave scope when the block they were declared in ends
    - At the `}
- Arguments
  - Enter scope when the method/function is called
  - Leave scope when the method/function returns

Why Scope Matters

- Constructors are called when objects enter scope
- Destructors are called when they leave scope
- This is why you rarely need to invoke a destructor on a local variable
  - It happens automatically on return
- This is why `}
  - Error in a destructor

Scoping & Parameter Passing

- Pass by reference parameters
  - Do not create new objects
  - Do not enter or leave scope
  - Do not trigger constructors
  - Do not trigger destructors
- Pass by value & Pass by reference
  - Different in terms of side effects
  - Different in terms of copying, which implies...
  - Different in terms of constructors/destructors
- Pass by constant reference is still by reference

Special Case: Return Values

- Return values:
  - Are copied back to the calling function directly
  - Do not trigger destructors in the returning function
  - Do not trigger constructors in the calling function
- All **other** local objects trigger destructors when the function returns.

Return Value Example

- How many constructors are called, and where?
  ```cpp
  Quagga MakeQuagga()
  {    
  Quagga novel_quagga = Quagga(100);    
  return novel_quagga;    }
  int main(int argc, char* argv[])    
  {    
  std::cout << MakeQuagga() << endl;    }
  ```