Lecture 04
Arrays & C-Style Strings
February 12th, 2016

Announcements
• Quiz #3: 1 version of 1 question was wrong
  – If the grade in Canvas hasn’t changed, email me
• PA3 is due on Wednesday
  – Questions?
• No Quiz #4 for Monday
  – We are running a little behind…
  – It might be due on Wednesday or Friday
• Next week’s recitation: Mandatory
  – gdb

Memory Management Strategy #0
• Use local variables, avoid pointers
  – No memory leaks
  – (Almost) no invalid pointers or references
• Limits to local variables
  – Doesn’t apply to persistent data
  – Doesn’t apply to dynamic data
    • Size of stack frame must be known at compile time

The Heap
• The heap is memory under programmer control
  – Starts at the other end of VM from the stack
  – The ‘new’ operator:
    • Allocates memory on the heap
    • For objects, calls the class constructor to initialize it
  – The ‘delete’ operator:
    • Takes a pointer to an object as a parameter
    • Calls the destructor (if an object)
    • Deallocates the memory
      – Available for use by ‘new’ again
• Memory Management: every ‘new’ needs a matching ‘delete’

Pointers
• A pointer is a typed VM address
  – As such, it is its own entity (datum)
  – It is not the same thing as the object it points to
  – It can point into the heap or the stack
  – It can point to uninitialized or unallocated memory
    • But this is almost always a bug (or malware)
  – It can point into the OS’s half of memory
    • But most attempts to use such an address will flag an error (core dump)

Operations on pointers
• ‘new’ allocates a pointer (from the heap)
• ‘delete’ de-allocates a pointer
  – If the pointer does not point to the heap, your code will crash
  – Delete does not change the value of the pointer
    • Setting it to NULL after delete is good style
    • Deleting a NULL pointer is a no-op
• If *q is an instance of Quagga:
  – &*q returns a pointer to q
• If *p is a pointer to a Quagga:
  – *p returns the quagga pointed to by *p
  – p->foo() call the foo method of *p
  – p->bar returns the bar field of *p
  – p->bar is same as (*p)->bar
Why doesn’t Java have Pointers?

• Hint: it’s related to why it doesn’t have destructors…

Objects vs Variables

• Imagine the following line of code:
  Quagga* qptr = new Quagga(10);

• Two simple questions:
  – What is the data type of qptr?
    • Quagga* (pointer to Quagga)
  – What is the variable name of the Quagga that was allocated?
    • It doesn’t have one!
    • The Quagga is an object, but not a (local) variable
    • qptr is a variable of type Quagga*

Warnings About Pointers

1. After you delete a pointer, it still has a value
   – Good style to set it to NULL
2. If you have 2 pointers to 1 object and delete 1, the other becomes invalid
   – Having 2 pointers to 1 object is called aliasing
3. If the only pointer to a heap object falls out of scope, you have a memory leak
4. Pointers to local variables become invalid when they fall out of scope
5. Deleting a pointer to a local variable will (generally) crash the machine

Arrays

• An array is
  – A contiguous block of memory
  – Containing instances of 1 type of data
• The data type of an array is a pointer
  – Quagga* and Quagga[] are the same data type
• The Brackets [] are used to offset into arrays
  – Arr[0]: 1st element
  – Arr[1]: 2nd element
  – Arr[n-1]: Nth element