Announcements

Canvas timed quiz 1 and programming assignment 1

pointers.c: A lab exercise for pointers, arrays, and memory

Lesson 1: What are pointers?
Lesson 2: Dereferencing pointers with *
Lesson 3: The integer datatype uses four bytes
Lesson 4: Printing each byte of an integer
Lesson 5: Pointers are just variables that live in memory
Lesson 6: Arrays are just places in memory
Lesson 7: Passing-by-value
Lesson 8: Passing-by-reference
Lesson 9: Passing an array leads to passing-by-reference
Lesson 10: How the stack works; recursion example
Quiz 1

1. Spanning Today Monday 1/30 - Friday 2/3.
2. 45 minutes.
3. Two tries.
4. Linux, some C.
5. Reviews recent concepts that would be fair game for exams.

Programming assignment 1

1. Due Friday 2/10.
2. Arrays, pointers, recursion, beginning data structures.
# Table of contents

**Announcements**

- Canvas timed quiz 1 and programming assignment 1

**pointers.c: A lab exercise for pointers, arrays, and memory**
- Lesson 1: What are pointers?
- Lesson 2: Dereferencing pointers with *
- Lesson 3: The integer datatype uses four bytes
- Lesson 4: Printing each byte of an integer
- Lesson 5: Pointers are just variables that live in memory
- Lesson 6: Arrays are just places in memory
- Lesson 7: Passing-by-value
- Lesson 8: Passing-by-reference
- Lesson 9: Passing an array leads to passing-by-reference
- Lesson 10: How the stack works; recursion example
From the folder 2023_0s_211, type: `git pull`
Why pointers?

Pointers underlie almost every programming language feature:

- arrays
- pass-by-reference
- data structures

Vital reason why C is a low-level, high-performance, systems-oriented programming language (why we use it for this class, computer architecture).
Lesson 1: What are pointers?

- Pointers are numbers
- The unary operator & gives the “address of a variable”.
- How big is a pointer? 32-bit or 64-bit machine?
- Pointers are typed
Lesson 2: Dereferencing pointers with *

*pointer: dereferencing operator: variable in that address
int* ptr and int *ptr

No difference between int* ptr and int *ptr

- int* ptr emphasizes that ptr is int* type
- int *ptr emphasizes that when you dereference ptr, you get a variable of type int
Lesson 3: The integer datatype uses four bytes

- Memory is an array of addressable bytes
- Variables are simply names for contiguous sequences of bytes
Lesson 4: Printing each byte of an integer

- Most significant byte (MSB) first → big endian
- Least significant byte (LSB) first → little endian
Which one is true for the ilab machine?
Lesson 5: Pointers are just variables that live in memory

- Pointers to pointer
Lesson 6: Arrays are just places in memory

- name of array points to first element
- `malloc()` and `free()`
- stack and heap
- using pointers instead of arrays
- pointer arithmetic
- `char* argv[]` and `char** argv` are the same thing
Using stack and heap picture, understand how pass by value and pass by reference are different.

- C functions are entirely pass-by-value
- `swap_pass_by_values()` doesn’t actually succeed in swapping two variables.
Lesson 8: Passing-by-reference

Using stack and heap picture, understand how pass by value and pass by reference are different.

- You can create the illusion of pass-by-reference by passing pointers
- `swap_pass_by_references()` does succeed in swapping two variables.
Lesson 9: Passing an array leads to passing-by-reference
Lesson 10: How the stack works; recursion example

<table>
<thead>
<tr>
<th>Low addresses</th>
<th>Global / static data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Heap grows downward</td>
</tr>
<tr>
<td></td>
<td>Dynamic memory allocation</td>
</tr>
<tr>
<td>High addresses</td>
<td>Stack grows upward</td>
</tr>
<tr>
<td></td>
<td>Local variables, parameters</td>
</tr>
</tbody>
</table>

Table: Memory structure