C Programming: Pointers, Arrays, Memory

Yipeng Huang

Rutgers University

January 27, 2022

Table of contents

Reminders about return to in-person

Lecture

- Hill 114
- Hybrid: will also livestream & record
- Masks required, sit apart. If feeling sick, stay home and watch livestream.

Recitation

- Because students are free to attend any recitation session, rooms may overflow
- If room is overcrowded, you will have to go somewhere else for livestream, watch recording, or go to different session.

Good luck and stay safe with moving weekend and return to classroom plans.

Assignments, lectures, and quizzes

Assignments

1. PA0 has been out, due Tuesday Feb. 1. Currently 50% have submitted.
2. PA1 has been out, due Tuesday Feb. 8.

Lectures

1. Today: pointers
2. Tuesday, Feb 1: Arrays, relating to PA1 greedyScheduling
3. Thursday, Feb 3: Matrices, recursion, dynamic programming, relating to PA1 matChainMul

Quiz next week

1. Spanning Tuesday 2/1 - Thursday 2/3. Linux, some C, stuff covered up to today.

Table of contents

Any missing pieces for `collatz.c`

Table of contents

git pull

From the folder 2022_0s_211, type: `git pull`

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 $\rightarrow$ big endian
- Least significant byte (LSB) first $\rightarrow$ 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

Lesson 7: Passing-by-value

- C functions are entirely pass-by-value

Lesson 8: Passing-by-reference

- You can create the illusion of pass-by-reference by passing pointers

Lesson 9: Passing an array leads to passing-by-reference