C Programming: I/O, files, pointers

Yipeng Huang

Rutgers University

September 9, 2025

Table of contents

Class resources

- You should notice now that these slides are not comprehensive.
- Supplemental reading and recitation slides on Canvas.
- Sequence of recitations this afternoon.
- Programming assignment 0 progress?
- Where have you found help?
- Piazza.

Quiz 1

1. Starting this week; open Wednesday 9/10, due Wednesday 9/17.
2. 60 minutes.
3. Two tries.
4. Linux, some C.
5. Reviews recent concepts that would be fair game for exams.

Table of contents

`rootFinder`: A program that prints square roots if integer

- Headers
- Command line arguments
- Opening files
- Reading from files
- `printf` and format specifiers
- `EXIT_SUCCESS`

Command line arguments: First encounter with pointers

What is `char* argv[]`

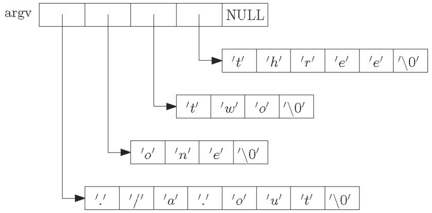

Figure: Image credit: `http://www.csc.villanova.edu/~mdamian`

In C, Strings, `char*`, and `char[]` are all the same

- `char greeting[6] = {'H','e','l','l','o','\0'};`
- `char greeting[] = "Hello";`

Compiling and running your program

How does a program end up on your computer?

```
gcc -Wall -Werror -fsanitize=address -std=c99 -o
rootFinder rootFinder.c -lm
```

- `gcc`: GNU C Compiler
- `-Wall -Werror`: Enable helpful warnings.
- `-fsanitize=address`: Enable memory checking.
- `-std=c99`: Set C standard version number.
- `-o rootFinder`: Output binary.
- `rootFinder.c`: Source file.
- `-lm`: Link the math library implementation.

Compiling and running your program

How does a program end up on your computer?

How a Makefile works

- `$<`: first prerequisite
- `$^`: all prerequisites
- `$@`: target file name

Assignment infrastructure for this course

Navigating the 2025_1f_211/ assignments directory

- `autograder.py`
- `tests/`: test cases
- `answers/`: expected answers
- Every assignment part has several fixed test cases for development, several randomized test cases for validataion.
- `assignment_autograder.py`
- `tar cvf pa0.tar .`

Table of contents

git pull

From the folder 2025_1f_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 $\rightarrow$ 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