

C Programming: Arrays, Functions

Yipeng Huang

Rutgers University

September 16, 2025

Table of contents

Announcements

Programming assignment 1

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

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

Programming assignment 1

Programming assignment 1

1. Due Friday 9/26.
2. Arrays, pointers, recursion, beginning data structures.

Table of contents

Announcements

Programming assignment 1

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

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

Lesson 5: Pointers are just variables that live in memory

- Pointers to pointer

Lesson 6: Arrays are just places in memory

- ▶ Three types of array in C: Fixed length, variable length, heap-allocated.
- ▶ name of array points to first element
- ▶ stack and heap
- ▶ `malloc()` and `free()`
- ▶ using pointers instead of arrays
- ▶ pointer arithmetic
- ▶ `char* argv[]` and `char** argv` are the same thing

Lesson 7: Passing-by-value

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

Low addresses		Global / static data
	Heap grows downward	Dynamic memory allocation
High addresses	Stack grows upward	Local variables, parameters

Table: Memory structure