

[Skip to page content](#)

# Lab 1a: Pointers in C

**Assigned:** Friday, March 29, 2024

**Due Date:** Monday, April 8, 2024 at 11:59 pm

**Video(s):** [Getting Started](#) and [Checking Your Work](#)

## Overview

### Learning Objectives

- Describe what a pointer is and use them to manipulate data.
- Use pointer arithmetic to perform address manipulations and access data.
- Describe the relationship between pointers and arrays.
- Use C casting to switch between different pointer types to make use of the different data type sizes.

Pointers are a critical part of C and necessary for understanding assembly code (Lab 2-3) and memory allocation (Lab 5).

## Code for this lab

**Browser:** [Download here](#)

**Terminal:** `wget`

`https://courses.cs.washington.edu/courses/cse351/24sp/labs/lab1a.tar.gz`

**Unzip:** Running `tar zxvf lab1a.tar.gz` from the terminal will extract the lab files to a directory called `lab1a`.

## Lab 1a Instructions

## Lab Format

`pointer.c` contains a skeleton for some programming puzzles, along with a comment block that describes exactly what the functions must do and what restrictions there are on their implementation. Your assignment is to complete each function skeleton according to the following rules:

- Only straightline code (*i.e.*, no loops or conditionals) unless otherwise stated. Look for "Control Constructs" under `ALLOWED` in `pointer.c` comments.
- A limited number of C arithmetic and logical operators (described in `pointer.c` comments).
- The C bit shift operators ( `<<` and `>>` ) may be needed for some problems but will not be covered in lecture until next Wednesday.
- No constants larger than 8 bits (*i.e.*, 0 - 255 inclusive) are allowed.
- Feel free to use " ( " , " ) ", and " = " as much as you want.
- ***You are permitted to use casts for these functions.***
- We will not grade comments, but commenting your code is good practice and helpful for debugging.

You will start working with basic pointers and use them to compute the size of different data items in memory, modify the contents of an array, and complete a couple of pointer "puzzles" that deal with alignment and array addresses.

---

## Using Pointers

This section describes the functions you will be completing in `pointer.c` found in the `lab1a` folder you downloaded. Refer to the file `pointer.c` itself for more complete details.

### Pointer Arithmetic

1. The first three functions ask you to compute the size (how much memory a single one takes up, in bytes) of various data elements (ints, doubles, and pointers). You will accomplish this by noting that arrays of these data elements allocate contiguous space in memory so that one element follows the next.

### Manipulating Data Using Pointers

The next two functions challenge you to manipulate data in new ways with your new knowledge of pointers:

4. The `swap_ints` function asks you to swap the values that two given pointers point to, without changing the pointers themselves (*i.e.*, they should still point to the same memory addresses).
5. The `change_value` function asks you to change the value of an element of an array using only the starting address of the array. You will add the appropriate value to the pointer to

create a new pointer to the data element to be modified. ***You are not permitted to use [] syntax to access or change elements in the array anywhere in the `pointer.c` file.***

## Pointers and Address Ranges

These two problems are particularly confusing, so please post on the discussion board if you are having trouble understanding any of these examples!

6. The `within_same_block` function asks you to determine if the addresses stored by two pointers lie within the same block of 64-byte aligned memory. The following are some examples of parameters and returns for calls to this function:

- `ptr1: 0x0`  
`ptr2: 0x3F`  
`return: 1`
- `ptr1: 0x0`  
`ptr2: 0x40`  
`return: 0`
- `ptr1: 0x3F`  
`ptr2: 0x40`  
`return: 0`
- `ptr1: 0x3CE`  
`ptr2: 0x3EF`  
`return: 1`
- `ptr1: 0x3CE`  
`ptr2: 0x404`  
`return: 0`

7. The `within_array` function ask you to determine if the address stored in `ptr` is pointing to a byte that makes up some part of an array element for the passed array. The byte does not need to be the first byte of the array element that it is pointing to. That description is a bit wordy, so here are some examples.

- `int_array: 0x0`  
`size: 4`  
`ptr: 0x0`  
`return: 1`
- `int_array: 0x0`  
`size: 4`  
`ptr: 0xF`  
`return: 1`
- `int_array: 0x0`  
`size: 4`

```

    ptr: 0x10
    return: 0
◦ int_array: 0x100
    size: 30
    ptr: 0x12A
    return: 1
◦ int_array: 0x100
    size: 30
    ptr: 0x50
    return: 0
◦ int_array: 0x100
    size: 30
    ptr: 0x18C
    return: 0

```

## Byte Traversal

8. The `string_length` function has you return the length of a string, given a pointer to its beginning. Recall that C-strings do not know how long they are, so we must calculate it for ourselves. Note that you **are** allowed to use loops and that the null character does **NOT** count as part of the string length.
9. The `endian_experiment` function has you set the value a pointer points to to the number 351351. Remember that we work with little endian data storage, and what that means.

## Selection Sort

10. The final part of the lab has you implement `selection_sort`. Selection sort works by effectively partitioning an array into a sorted section, followed by an unsorted section. It repeatedly finds (and selects) the minimum element in the unsorted section, and moves it to the end of the sorted section ( `swap_ints` might be useful for this). The pseudo code might look something like this:

```

// "arr" is an array
// "n" is the length of arr
for i = 0 to n - 1
    minIndex = i
    for j = i + 1 to n
        if arr[minIndex] > arr[j]
            minIndex = j
        end if
    end for
    Swap(arr[i], arr[minIndex])
end for

```

Note that you **are** allowed to use loops and if statements in this one.

## Checking Your Work

Make sure that you remove all `printf` statements from your code before you submit!

We have included the following tools to help you check the correctness of your work:

- `pctest.c` is a test harness for `pointer.c`. You can test your solutions like this:

```
$ make
$ ./pctest
```

This only checks if your solutions return the expected result. *We may test your solution on inputs that `pctest` does not check by default and we will review your solutions to ensure they meet the restrictions of the assignment.*

- `dlc.py` is a Python script that will check for compliance with the coding rules. The usage is:

```
$ ./dlc.py
```

When no issues are found, the output will be `[]`, otherwise, the output inside the brackets will tell you (using shorthand notation) what issues were found function-by-function.

- The `dlc` program enforces a stricter form of C declarations than is the case for C++ or that is enforced by `gcc`. In particular, in a block (what you enclose in curly braces) all your variable declarations must appear before any statement that is not a declaration. For example, `dlc` will complain about the following code:

```
int foo(int x) {
    int a = x;
    a *= 3;      /* Statement that is not a declaration */
    int b = a;   /* ERROR: Declaration not allowed here */
}
```

Instead, you must declare all your variables first, like this:

```
int foo(int x) {
    int a = x;
    int b;
    a *= 3;
    b = a;
}
```

- The `dlc` program will also complain about binary constants such as `0b10001000`, so avoid using them.

# Lab 1a Synthesis Questions

Make sure your answers to these questions are included in the file `lab1Asynthesis.txt` !

In both `lab0.c` and `pointer.c`, we saw the effects of pointers and pointer arithmetic:

1. Considering how you calculated the actual difference (in bytes) between two addresses in C *without any compiler warnings*, briefly explain why *each step* was necessary. [3 pt]
2. Explain why the parameters to the function `swap_ints` must both be pointers. What would happen if the parameters were integers? [3 pt]
3. △ *The following question is open-ended! This question is less about "right" or "wrong" and more about you giving us your personal reflection.*

The Intro series at the Allen School uses Java, which leaves out the concept of pointers among other "low-level" language features. How do you think this has impacted your entry into computer science and programming? Name *at least one positive impact and at least one drawback/limitation* and justify all of your answers. [3 pt]


## Submission

You will submit: `pointer.c` and `lab1Asynthesis.txt`.

Be sure to run `make` and `./dlc.py` on your code before submitting! Submissions that don't compile or can't be parsed by `dlc` will automatically receive a score of ZERO.

After submitting, please wait until the autograder is done running and double-check that you passed the "File Check" and "Compilation and Execution Issues" tests. If either test returns a score of -1, be sure to read the output and fix any problems before resubmitting. Failure to do so will result in a programming score of ZERO for the lab.

Make sure that you remove all `printf` statements from your code before you submit!

Submit your files to the "Lab 1a" assignment on  Gradescope. Don't forget to add your partner, if you have one.

It is fine to submit multiple times up until the deadline; we will only grade your last submission. If you do re-submit, you must re-submit BOTH files again AND add your partner again to the new submission.

**W** PAUL G. ALLEN SCHOOL  
OF COMPUTER SCIENCE & ENGINEERING  
(<https://cs.uw.edu>)

UW Site Use Agreement ([//www.washington.edu/online/terms/](https://www.washington.edu/online/terms/))