

Section 2.3

Pointers

1. Overview
2. Using pointers
3. Memory map
4. Compound data types
5. Pointer arithmetic
6. Parameter passing
7. Command line arguments

2.3.1 Overview

- What is a *pointer*?
 - it's a **variable** that stores a memory address as its **value**
 - a pointer variable can be used to store:
 - the memory address of another variable
 - the memory address of a block of *dynamically allocated memory*
 - ... more on this later ...



Overview (cont.)

- Why use pointers?
 - they are a small, fixed size variable
 - a pointer is usually smaller in size than the variable it points to
 - they allow us to pass parameters by reference
 - so we can make changes to areas of memory that are out of scope
 - example: modifying variables in a calling function
 - they allow us to avoid copying data
 - we can avoid making multiple copies of the same thing
 - we can avoid making temporary copies
 - making copies is generally bad:
 - it wastes processing time
 - it can lead to inconsistencies in the data

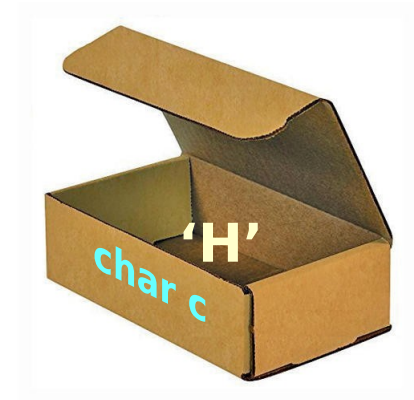
Pointer Variables

- All variables have:
 - a name
 - a data type
 - a value
 - a location in memory == an address
 - a scope and a storage class
 - ... more on this later ...

Pointer Variables (cont.)

- Pointer variables have:
 - a name
 - a data type
 - the data type that they point to
 - the * symbol
 - a value
 - this value is an address in memory
 - it may be the address of another variable (the *pointee*)
 - the pointee must of the **same** data type as the pointer declaration
 - a location in memory == an address


Pointer Variables (cont.)



Name	Type	Value	Location



Pointer Variables (cont.)

- Characteristics of pointers
 - in a 64-bit architecture: pointer variables occupy 8 bytes
 - pointers may point to variables of:
 - any data type
 - any size
- coding example <p1>
- Interesting thoughts 
 - a pointer can point to only *one* variable
 - a variable may have *multiple pointers* to it

2.3.2 Using Pointers

- Declaring pointers
- Assigning values to pointers
- Accessing pointer values

Declaring Pointers

- What is a variable declaration?
 - a statement specifying a variable's name and data type
 - ... and more stuff we'll talk about later ...
- A pointer declaration has three components:
 - the type of data that the pointer will point to
 - the symbol `*` to indicate that the variable declared is a pointer
 - the pointer variable name

Assigning Values to Pointers

- Pointers are assigned values
 - from the memory address of another variable
 - from a system call requesting a new block of memory
 - ... more on this later ...
- Important symbol: **&**
 - this is the **address-of** operator
 - it returns the memory location of the operand variable
 - the address in memory where the variable is located

Accessing Pointer Values

- Important symbol: ***** has **three** roles
 - in a variable declaration: indicates a pointer data type
 - as a *binary* operator: it's the multiplication operator
 - as a *unary* operator: it's the **dereferencing** operator
 - it returns the value stored at the operand memory address
- Common problem: NULL pointer exception
 - this happens when we dereference a pointer that is set to NULL
 - the pointer value is zero, so we try to access memory address zero
 - good habit: always check that your pointers are **not** NULL

2.3.3 Memory Map

- What is a memory map?
 - a tool for programmers to visualize the memory layout of program
- Characteristics
 - it is represented as a table listing all the variables in a program
 - it includes:
 - variable names
 - their values
 - their memory addresses

Memory Map (cont.)

- Each memory address:
 - points to one byte, either:
 - the first byte of a multi-byte data type, or
 - the first byte of a block of dynamically allocated memory
 - is usually represented in hexadecimal
 - each hexadecimal digit is one nibble
 - two nibbles == one byte
 - example: `0x00007ffd765afb9b`

Memory Map (cont.)

- Purpose of a memory map
 - it helps us track the values of our variables
 - some debuggers allow you to view the program's raw memory
 - it helps us avoid logic errors
 - make sure your program is doing what you want
 - it helps us keep our sanity
 - ... wait until we talk about *double pointers* ...

2.3.4 Compound Data Types

- Using pointers with arrays
 - the name of an array is used interchangeably with its address
 - the array name is a pointer
 - the array name points to the first element of the array
 - a pointer can point to any element

Compound Data Types (cont.)

- Using pointers with structures
 - they are used in the same way as with any data type
 - accessing fields from a *structure variable*
 - use the dot operator
 - example: `gertrude.age = 99;`
 - accessing fields from a *pointer to a structure*
 - option #1: dereference the pointer, then use the dot operator
 - kind of a newbie move
 - option #2: use the *arrow operator*
 - example: `stuPtr->age = 99;`
- coding example <p2>



2.3.5 Pointer Arithmetic

- We can perform addition and subtraction on pointers !
 - this **only** makes sense when used with arrays
- Arrays and pointers
 - use pointer arithmetic on an array name to access its elements
 - + and – operators
 - the arithmetic unit is the number of bytes in each **element**
- coding example <p3>

2.3.6 Parameter Passing

- There are two ways to pass parameters to a function
 - pass by value
 - the value is copied into the called function
 - pass by reference
 - the address is copied into the called function

Parameter Passing (cont.)

- What happens in *pass by value*?
 - the **value** of a variable:
 - is copied
 - is passed in to the called function
 - all changes to the variable are to the *local copy only*
 - the changes disappear once the called function returns
 - called function **cannot** change the values in the calling function

Parameter Passing (cont.)

- Pass by reference
 - the **address** of a variable is passed in to the called function
 - the address is passed in as a pointer data type
 - changes to the variable are **not** local
 - there is *no local copy*!
 - the changes are made to the variable inside the calling function
 - this is used to “return” multiple values to the calling function
 - the function return value is usually returns **only** success or failure
 - changes to data are made using parameters passed by reference
- coding example <p4>

2.3.7 Command Line Arguments

- What are command line arguments?
 - everything typed at the command line to launch the program
 - every space-delimited word typed in at the shell prompt
 - the *command line* is another term for the shell prompt
- Characteristics
 - every word on the command line is part of a parameter to **main()**
 - this includes the executable name
- **coding example <p5>**

Command Line Arguments (cont.)

- Purpose of command line arguments
 - fun fact: re-compiling can be slow and error-prone
 - professionally built software avoids re-compiling at all cost
 - re-linking is fine, because it's much faster
 - command line arguments can configure the program behaviour
 - a program can exhibit different behaviour without re-compiling
 - program can change its behaviour depending on argument value
 - every program execution may be different