Software Systems

Day 4 – Hex, Memory, and Pointers

Agenda

- Announcements
- Hex
- Memory
- Pointers

Announcements

- Lots of people seem to have issues building code.
- Much of this is likely due to being on Ubuntu 20 instead of Ubuntu 22.
- You can update in-place, but read up on the subtleties of it. (Things can still break because package names can change.)
- I'm investigating possible fixes.
  - Please post in Discord for help, and also include as much information about your setup as you can.
  - I'll try to fix issues for the release of Assignment 1.

Today: Hex, Memory, Pointers

- Systems is about understanding what a computer is doing under the hood.
- Compared to Python, C gives you a much closer view of what the machine is doing.
- There are still abstractions, but it's a lower-level view.

Today: Hex, Memory, Pointers

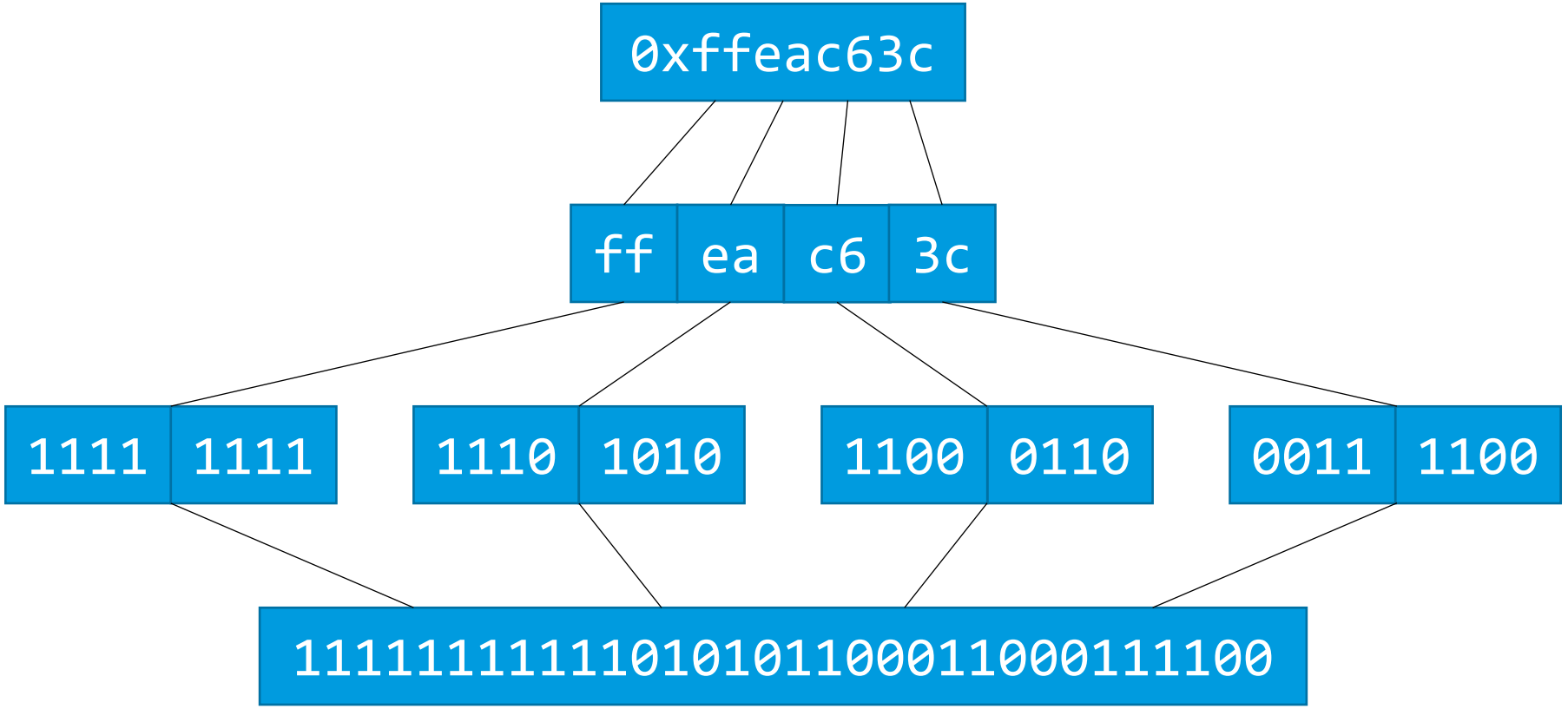
- With compilation, we saw that
  - Variable names don't get compiled into the program.
  - Things are instead identified by where in memory they live.
- Today, we'll learn:
  - Hexadecimal notation and why it is useful.
  - The memory layout and how it's used in a program.
  - Pointers and how they helps us work with memory in C.

Hexadecimal Notation (Hex)

- Everything that a computer works with is ultimately in bits.
- Bits are written as binary digits - 1s and 0s.
  - The number 42 is `101010`.
  - The string "sus" in C is `01110011011110101011001100000000`.
- But this is tedious to read.
- Hexadecimal works in powers of 16 rather than powers of 2.
  - Group 4 bits at a time.
  - Still small enough to work with, but much more compact.

Hexadecimal Notation (Hex)

- 32-bit: 2^{32} addresses, 64-bit: 2^{64} addresses.



Hexadecimal Notation (Hex)

- Hex is often written with the prefix `0x` to distinguish it from other notations.
- Usually written as an even number of digits, with leading 0s if necessary (e.g., `0x0f` instead of `0xf`).

Hexadecimal Notation (Hex)

- Let's do an exercise with hex. A hexercise, if you will.
- What is `0xb` in binary? In decimal?
- What is `0xbeefface` in decimal?
- What is `0xcafe` in binary?
- What is the string "IYKYK" in binary? In hex?
  - Look up an ASCII table for how to encode characters.
- What is the longest word you can spell with only hexadecimal digits?

Memory

- Head First C Chapter 2 talks about the memory layout.
- From top to bottom:
  - Stack (functions and local variables)
  - Heap (dynamically allocated memory)
  - Globals (variables accessible to everyone)
  - Constants (values that can't be changed)
  - Code (the compiled/linked bytes of the program)

Memory

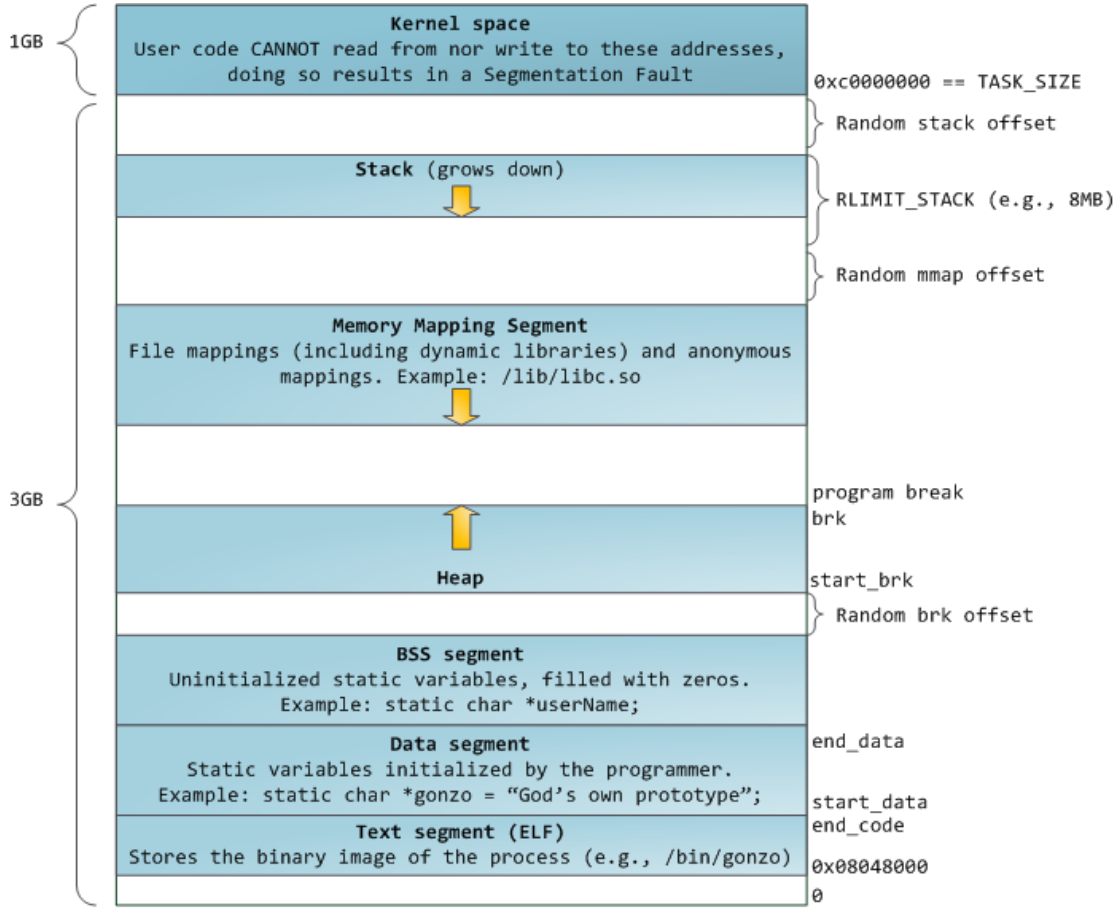
- In reality, this isn't quite accurate - the kernel takes part of memory.
- Specifically, the upper half.
- But actually, this isn't quite accurate either.
- 64 bits of address space is a lot of memory.
  - About 16 million TB can be referenced with 64 bits.
- These are not the real addresses in memory.
  - There's a translation step (for next time).

Memory

- To save on memory translation costs, the upper 17 bits must be identical for all "canonical" (valid) addresses in memory.
- So only the lower 48 bits are used as addresses.
- Exercise:
  - What is the lowest "higher half" address?
  - What is the highest "lower half" address?
  - How much total memory can be expressed with 48 bits?

Memory

- In practice, the layout is a bit more complicated than what Head First C shows.
- The stack grows down, and the heap grows up.



Pointers

- Pointers are how we refer to, and access, locations in memory.
- Declare pointers using *:
  - `int* p;`
- Dereference pointers also using *:
  - `int x = *p;`
- Find the location of a variable using &:
  - `int* p = &x;`

Pointers

- You can also combine dereference and assignment:
  - `*p = 42;`
- And `a[3]` is just "syntactic sugar" for `*(a + 3)`.
- This means that `3[a]` is also completely valid (though cursed).

Pointers

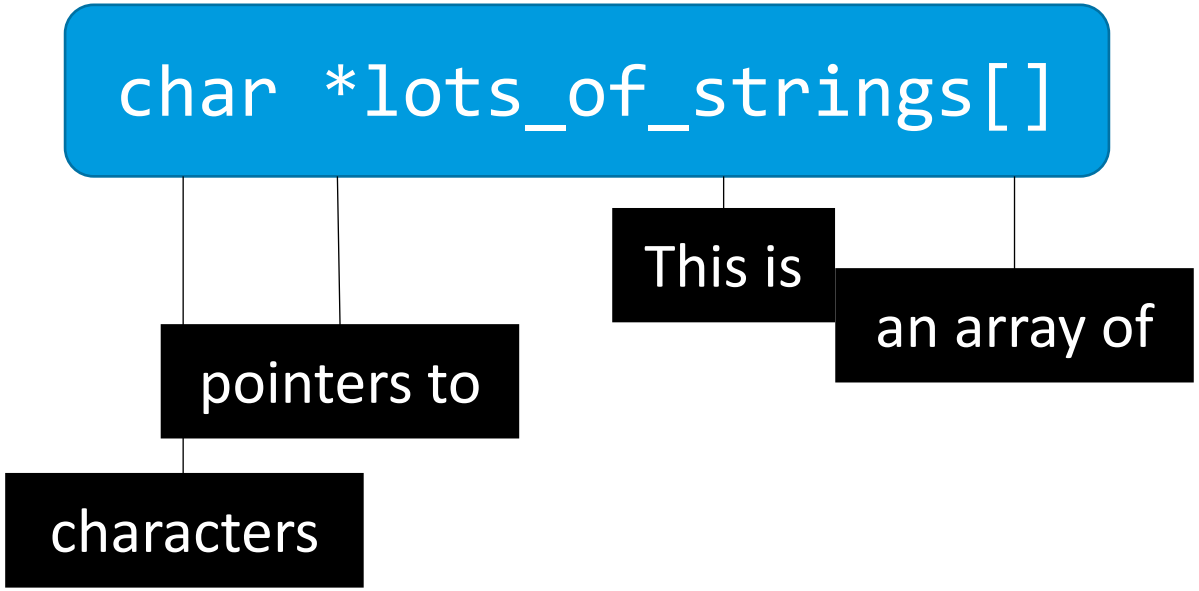
- To print a pointer, use %p:
```
int x = 42;
printf("%p\n", &x);
```
- Exercise: try allocating pointers in different parts of memory, and printing them.
  - To allocate memory on the heap:
```
#include <stdlib.h>
// Do stuff here
int* p = malloc(sizeof(int));
// Do stuff here
free(p);
```
  - What ranges of addresses do you see?

Pointers and Types

- Why does C care about what kind of data it points to?
- Dereferencing a pointer "removes" the *:
```
int x = 42;
int* p = &x;
int** pp = &p;
int* y = *pp;
```
- C needs to know what it can do with the data after dereferencing.
- For arrays, the pointer arithmetic needs to work out:
```
int numbers[3] = {3, 1, 4};  // Usually 12 bytes
int a = numbers[1];  // 4 bytes from numbers
```
- You can also have a generic pointer: `void*` (but don't use it now).

Pointers and Types

- So how do you know what type `char *lots_of_strings[]` is?
- Use the right-to-left rule:
  - Start from the variable name and read right.
  - Then go back to the variable name and read left.



Pointers and Types

- Exercise: suppose that you have:

```
int* x = malloc(sizeof(int));
int a[3] = {3, 1, 4};
const char* name = "Steve";
```

- What type is each expression?
  - `&x - int**`
  - `a[1] - int`
  - `*x - int`
  - `*name - (const char)`
  - `name + 1 - const char*`

Pointers: Passing to Functions

- Everything in C is what we call pass by value:
  - If you pass something to a function, it makes a copy of the value.
- What does the following program print?

```
#include <stdio.h>

void foo(int x) {
  x = 42;
}

int main(void) {
  int x = 0;
  foo(x);
  printf("%d\n", x);
  return 0;
}
```

Pointers: Passing to Functions

- If you want to let other functions change a variable, you can provide a pointer.
- What does the following program print?

```
#include <stdio.h>

void foo(int* p) {
  *p = 42;
}

int main(void) {
  int x = 0;
  foo(&x);
  printf("%d\n", x);
  return 0;
}
```

Pointers: Passing to Functions

- Be aware that Python always passes by pointer under the hood.
- So this will print [1, 3, 4]:

```
def sort(L):
    L.sort()


my_list = [3, 1, 4]
sort(my_list)
```

Pointers

- https://www.youtube.com/watch?v=mnXkiAKbUPg
- https://www.youtube.com/watch?v=G7LJC9vJluU