C Introduction

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Announcements

- Homework 6 due tonight at 11:59pm
- Homework 7 available soon, due next Monday on Gradescope
- Midterm 2 is on April 3
  - Similar format to midterm 1
  - Schedule with SDAC early if needed

Data Types in C

Integer data types

Data type	Size (bits)	Size (bytes)
char	8	1
short	16	2
int	32	4
long	64	8
long long	64	8

Each has 2 versions: *signed* and *unsigned*

by default, short, int, long, long long are signed.

char is implementation-defined. { char - depends on compiler/platform
signed char
unsigned char

Example of creating a variable: unsigned long long x = 5;

Data Types in C

Helpful Resources

- Wikipedia
- Our Reference and Summary

`sizeof()` - returns size in bytes

- `sizeof(int)` returns 4

Data Types in C (check "Readings → C Reference" for exponent bits and fraction bits)

Floating point

- float
- double

Data Types in C

We stored things like arrays of memory addresses.

We called functions by giving it the memory address.

.....

Data Types in C

Pointers - how C uses addresses!

Data Types in C

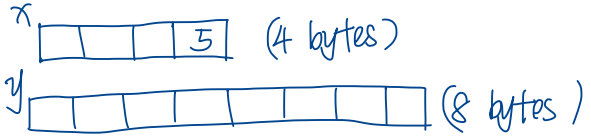
Pointers - how C uses addresses!

- Hold the address of a position in memory
- Need to know the kind of information stored at that location

The size of a pointer? How many bytes? — 64 bits (8 bytes) (Registers are 64 bits because we had 64-bit memory addresses)

If I want to have a pointer to an int: `int *y;`
→ the star references the fact that this is not an integer, it's a pointer.


```
int x=5;
int *y;
y=x; → doesn't work
```



`y=&x;` It works!
`*y=25;` will change x to 25.

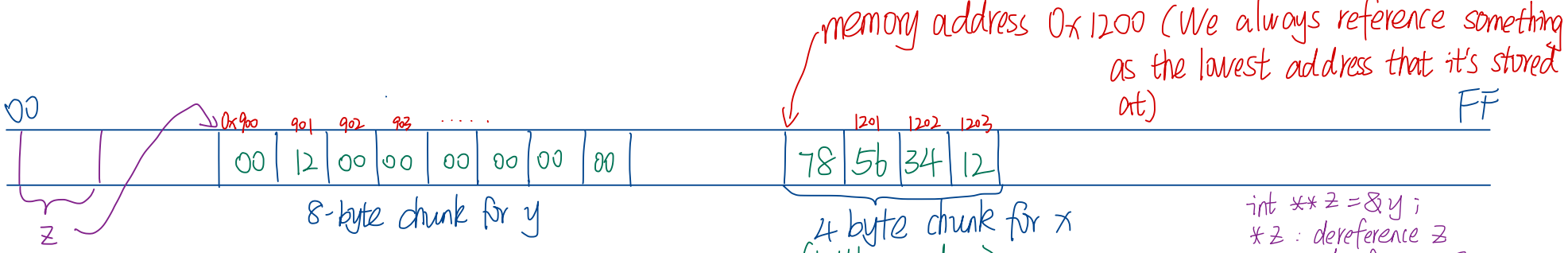
int ***z = &y; z is a pointer to a pointer to an int.
z has the address of y.

Data Types in C

later, if I say "*z", that means y.
"**z": the value of x.

Pointers - how C uses addresses!

- Hold the address of a position in memory
- Need to know the kind of information stored at that location



int x = 0x12345678

int *y;

y = &x, (I want y point to x, which means store the address of x in y).

int **z = &y;
*z : dereference z
**z: dereference *z
**z = 0x00000000;
(set x to 0x00000000)
*z = 25; (Warning: you're trying to assign an integer to a pointer)
(But later, when you use it as an address, may seg Fault).

y = 0xABCDEF01; (I'm changing the pointer) → I may don't want to do this. Sometimes seg fault?

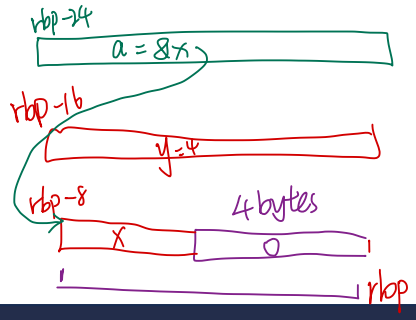
*y = 25; (When I use an asterisk, it will say is follow the pointer on y, follow the address to the place in memory that y points to and change that value)

Example

```
int main() {
    int x = 3;
    long y = 4;
    int *a = &x;
    long *b = &y;
    long z = *a;
    int w = *b;
    return 0;
}
```

Example

```
int main() {
    int x = 3;
    long y = 4;
    int *a = &x;
    long *b = &y;
    long z = *a;
    int w = *b;
    return 0;
}
```




```
0000000000000000 <main>:
   0:   55                          push   %rbp
   1:   48 89 e5                    mov    %rsp,%rbp
   4:   31 c0                       xor    %eax,%eax
   6:   c7 45 fc 00 00 00 00        movl   $0x0,-0x4(%rbp)
   d:   c7 45 f8 03 00 00 00        movl   $0x3,-0x8(%rbp)
  14:   48 c7 45 f0 04 00 00        movq   $0x4,-0x10(%rbp)
  1b:   00
  1c:   48 8d 4d f8                 lea    -0x8(%rbp),%rcx
  20:   48 89 4d e8                 mov    %rcx,-0x18(%rbp)
  24:   48 8d 4d f0                 lea    -0x10(%rbp),%rcx
  28:   48 89 4d e0                 mov    %rcx,-0x20(%rbp)
  2c:   48 8b 4d e8                 mov    -0x18(%rbp),%rcx
  30:   48 63 09                    movslq (%rcx),%rcx
  33:   48 89 4d d8                 mov    %rcx,-0x28(%rbp)
  37:   48 8b 4d e0                 mov    -0x20(%rbp),%rcx
  3b:   48 8b 09                    mov    (%rcx),%rcx
  3e:   89 4d d4                    mov    %ecx,-0x2c(%rbp)
  41:   5d                          pop    %rbp
  42:   c3                          retq
```

(compiler knows we are working on a 64-bit machine. I'll probably try to line everything to 8 bytes if I can. to make things faster for you)

loading the addr of x into rcx

y is treated as the same way.

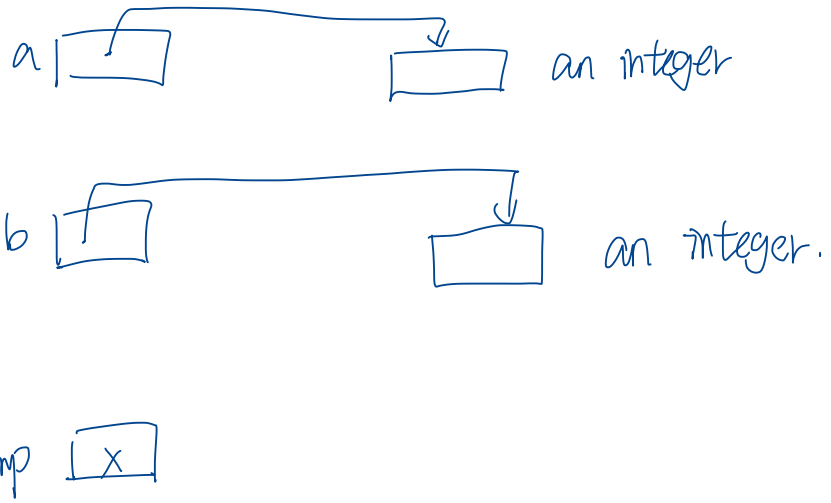
signed extend I to q, long to quad

b going from long to int. move the value of y into rcx, but I will just read out the value of ecx.

Example

Swap Example

```
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```



Using Compilers

Arrays

(We pick a spot in memory, and the next so many spots are our array)

Array: 0 or more values of same type stored contiguously in memory

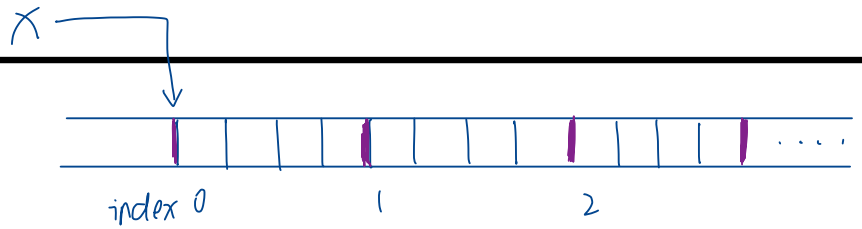
- Declare as you would use: `int myarr[100];` (100 integers in my array)
- `sizeof(myarr)` = 400 — 100 4-byte integers (How big is my array in bytes?)
- `myarr` treated as pointer to first element
- Can declare array literals:

```
int y[5] = {1, 1, 2, 3, 5}
```

(optional)

When I create an array, it actually create a pointer to the first thing in that list.

Pointers and Arrays



(dereference x will be the first element of array x)

`*x` and `x[0]` are equivalent

- Pointer to single value and pointer to first value in array
- Treat array as pointer to the first value (lowest address)
- Indexing into array: `x[n]` and `*(x+n)` → pointer arithmetic
  - If `x` is an `int *`, then `x+1` points to **next int** in memory
  - Adding 1 to pointer adds `sizeof()` the type we're pointing to

Not skip 5 bytes in memory

I know this pointer is a pointer to an integer, so I plus 5 * sizeof(int)