

C Introduction

CS 2130: Computer Systems and Organization 1

March 27, 2023

Announcements

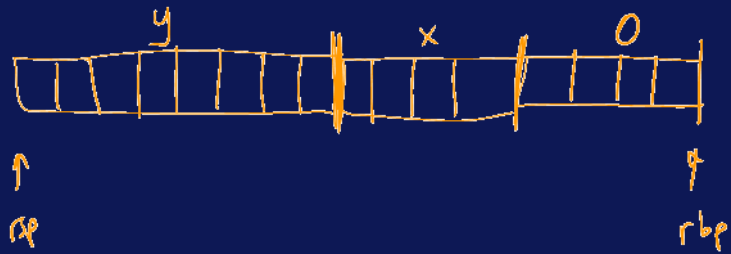
- Homework 6 Escape Room due tonight at 11pm
- If you are having **git** issues, please come to office hours!
- Exam 2 next Friday

Simple C Example

```
int main() {  
    return 0;  
}
```

The `main` function

- Start running the `main()` function
- `main` must return an integer - **exit code**
 - 0 = everything went okay
 - Anything else = something went wrong
- There *should* be arguments to main



Examples

Helpful Resources

- Wikipedia
- Our Reference and Summary

`sizeof()` - returns size in bytes

- `sizeof(int)` returns 4

Data Types in C

Integer data types

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

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

unsigned long long x = 0;

Data Types in C

Floating point

- float
- double

Data Types in C

Data Types in C

Pointers - how C uses addresses!

Data Types in C

Pointers - how C uses addresses!

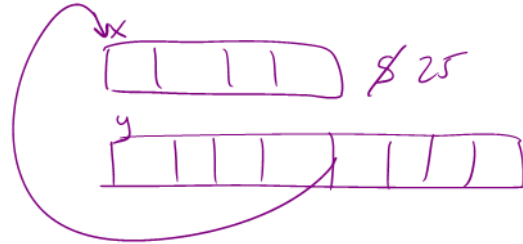
- Hold the address of a position in memory
- Need to know the kind of information stored at that location

```
int x = 5;
```

```
int *y;
```

```
y = &x;
```

```
*y = 5;
```



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
```

Arrays

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

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

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

Pointers and Arrays

`*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)`
 - 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

Pointers and Arrays

Consider: `int **a`

Example

Swap Example

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


Pointers

- All pointers are the same size: address size in underlying ISA
- Two special int types (defined using typedef)
 - `size_t` - integer the size of a pointer (unsigned)
 - `ssize_t` - integer the size of a pointer (signed)
 - With our compiler and ISA, these are both variants of `long`

Pointers

Consider the following code:

```
int x = 10;  
int *y = &x;  
int *z = y + 2;  
long w = ((long)z) - ((long)y);
```

Why is $w = 8$?

Other Types and Values

- Literal values - integer literals are implicitly cast
 - `unsigned long very_big = 9223372036854775808uL`
 - u for unsigned, L for long
- **enum** - named integer constants (in ascending order)
 - `enum { a, b, c, d=100, e };`
 - `int foo = e;`
- **void** - a byte with no meaning or "nothing"
 - Pointers: `void *p`
 - Return values: `void myfunction();`
- Casting - changing type, converting
 - Integer: zero- or sign-extend or truncate to space
 - Int to float: convert to nearby representable value
 - Float to int: truncate remainder (no rounding)

Structures

`struct` - Structures in C

- Act like Java classes, but no methods and all public fields
- Stores fields adjacently in memory (but may have padding)
- Compiler determines padding, use `sizeof()` to get size
- Name of the resulting type includes word `struct`

```
struct foo {  
    long a;  
    int b;  
    short c;  
    char d;  
};
```

```
struct foo x;  
x.b = 123;  
x.c = 4;
```

Structure Literals

```
struct a {  
    int b;  
    double c;  
};
```

```
/* Both of the following initialize b to 0 and c to 1.0 */  
struct a x = { 0, 1.0 };  
struct a y = { .b = 0, .c = 1.0 };
```

typedef

`typedef` - give new names to any type!

- Fairly common to see several names for same data type to convey intent
- Ex: `unsigned long` may be `size_t` when used in sizes

- Examples:

```
typedef int Integer;
```

```
Integer x = 4;
```

```
typedef double ** dpp;
```

- Used with *anonymous structs*:

```
typedef struct { int x; double y; } foo;
```

```
foo z = { 42, 17.4 };
```

Struct Example

