

# C

---

Monday!

CS 2130: Computer Systems and Organization 1

November 7, 2022

# Announcements

- Homework 8 out Wednesday, due Wednesday at 11pm
- Homework 2 graded, please review
- **No lab** tomorrow (election day)
- No lab Thanksgiving week

```
int z = (int) x;
```

```
if (x == 0) {  
  
}
```

```
if (!x) {  
  
}
```



# Calling Functions

The C code

```
long a = f(23, "yes", 34uL);
```

compiles to

```
movl $23, %edi  
leaq label_of_yes_string, %rsi  
movq $34, %rdx  
callq f  
# %rax is "long a" here
```

without respect to how `f` was defined. It is the calling convention, not the type declaration of `f`, that controls this.

# Calling Functions

But, if the C code has access to the type declaration of `f`, then it might perform some implicit casting first; for example, if we declared

```
long f(double a, const char *b, double c);
```

```
long a = f(23, "yes", 34uL);
```

then the call would be interpreted by C as having implicit casts in it:

```
long a = f((double)23, "yes", (double)34uL);
```

# Calling Functions

and the arguments would be passed in floating-point registers, like so:

```
movl $23, %eax
cvtsi2sd %eax, %xmm0           # first floating-point argument

leaq label_of_yes_string, %rdi # first integer/pointer argument

movl $34, %eax
cvtsi2sd %eax, %xmm1           # second floating-point argument

callq f
# %rax is "long a" here
```

# Function Declaration

```
int f(int x);
```

- Declaration of the function
- Function header
- Function signature
- Function prototype

We want this in every file that invokes `f()`

# Function Definition

```
int f(int x) {  
    return 2130 * x;  
}
```

- Definition of the function

We only want this in **one** **.c** file

- Do not want 2 definitions
- Which one should the linker choose?

# Header Files

C header files: `.h` files

- Written in C, so look like C
- Only put header information in them
  - Function headers
  - Macros
  - `typedefs`
  - `struct` definitions
- Essentially: information for the **type checker** that does not produce any actual binary
- `#include` the header files in our `.c` files

# Big Picture

## Header files

- Things that tell the type checker how to work
- Do not generate any actual binary

## C files

- Function definitions and implementation
- Include the header files

# Including Headers

```
#include "myfile.h"
```

- Quotes: look for a file where I'm writing code
- Our header files

```
#include <string.h>
```

- Angle brackets: look in the standard place for includes
- Code that came with the compiler
- Likely in `/usr/include`

`string.h`