

# C Introduction

---

## CS 2130: Computer Systems and Organization 1

**Xinyao Yi** Ph.D.  
Assistant Professor

## Announcements

---

- Homework 6 due Monday at 11:59pm
- Midterm 2 is 2 weeks away (April 3)
  - Similar format to midterm 1
  - Schedule with SDAC early if needed

## Compilation Pipeline

---

Turning our code into something that runs

- **Pipeline** - a sequence of steps in which each builds off the last

# C

---

C is a thin wrapper around assembly

- This is by design!
- Invented to write an operating system
  - Can write inline assembly in C
- Many other languages decided to look like C

## Simple C Example

---

```
int main() {  
    int y = 5;  
    return 0;  
}
```

## Compilation Pipeline

---

Earlier, we saw:

- C files (.c) compiled to assembly (.s)
- Assembly (.s) assembled into object files (.o)
- Object files (.o) linked into a program / executable

## Compiling C to Assembly

---

Multiple stages to compile C to assembly

- Preprocess - produces C
  - C is actually implemented as 2 languages:  
C preprocessor language, C language
  - Removes comments, handles preprocessor directives (#)
  - `#include`, `#define`, `#if`, `#else`, ...
- Lex - breaks input into individual tokens
- Parse - assembles tokens into intended meaning (parse tree)
- Type check - ensures types match, adds casting as needed
- Code generation - creates assembly from parse tree

## Compiling C to Assembly

---

## Compiling C to Assembly

---

## Errors

---

### Compile-time errors

- Errors we can catch during compilation (this process)
- **Before** running our program

### Runtime errors

- Errors that occur when running our programs

## 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`

## Data Types in C

---

Integer data types

Data type	Size
char	
short	
int	
long	
long long	

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

## Data Types in C

---

### Helpful Resources

- Wikipedia
- Our Reference and Summary

`sizeof()` - returns size in bytes

- `sizeof(int)` returns 4

## 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

## 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

```

# Using Compilers

## 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