

Intro, C Refresher

CSE 333 Spring 2021

Instructors: Travis McGaha, Justin Hsia

Teaching Assistants:

Atharva Deodhar

Callum Walker

Cosmo Wang

Dylan Hartono

Elizabeth Haker

Kyrie Dowling

Leo Liao

Markus Schiffer

Neha Nagvekar

Nonthakit Chaiwong

Ramya Challa

Introductions: Instructors (2 for 1!)

❖ Justin (he/him)

- From California (UC Berkeley and the Bay Area)
- I like: teaching, the outdoors, board games, and ultimate
- Taking care of newborn:

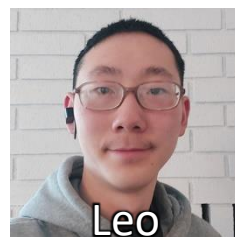
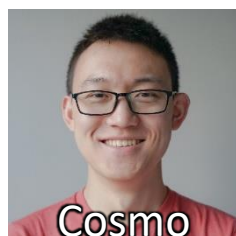
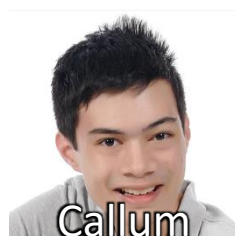
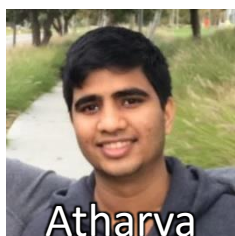


❖ Travis (he/him)

- Recent UW CSE BS/MS graduate
- I like: teaching, music, video games, video essays, and reading
- I'm on the job market!



Introductions: Teaching Assistants



- Available in section, office hours, and discussion group
- An invaluable source of information and help
- ❖ Get to know us (instructors + TAs)
 - We are here to help you succeed!



Introductions: Students

- ❖ ~190 students registered, split across two lectures
 - (Possibly) Largest offering ever!
 - There are no overload forms or waiting lists for CSE courses
 - Majors must add using the UW system as space becomes available
 - Non-majors should work with undergraduate advisors (in the Gates Center) to handle enrollment details

- ❖ Expected background
 - **Prereq:** CSE 351 – C, pointers, memory model, linker, system calls
 - **Indirect Prereq:** CSE 143 – Classes, Inheritance, Basic Data structures, and general good style practices
 - CSE 391 or Linux skills needed for CSE 351 assumed

Lecture Outline

❖ Course Policies

- <https://courses.cs.washington.edu/courses/cse333/21sp/syllabus.html>
- Digest here, but you *must* read the full details online

❖ Course Introduction

❖ C Reintroduction

Communication

- ❖ **Website:** <http://cs.uw.edu/333>
 - Schedule, policies, materials, assignments, etc.
- ❖ **Discussion:** <https://edstem.org/us/courses/4899/discussion/>
 - Announcements made here
 - Ask and answer questions – staff will monitor and contribute
- ❖ **Office Hours:** spread throughout the week
 - Can fill out Google Form to schedule individual 1-on-1 appointments
- ❖ **Anonymous feedback:**
 - Comments about anything related to the course where you would feel better not attaching your name

Course Components

- ❖ Lectures (28)
 - Introduce the concepts; take notes!!!
- ❖ Sections (10)
 - Applied concepts, important tools and skills for assignments, clarification of lectures, exam review and preparation
- ❖ Programming Exercises (12)
 - One due roughly every 4-5 days
 - We are checking for: **correctness, memory issues, code style/quality**
- ❖ Programming Projects (0+4)
 - Warm-up, then 4 “homework” that build on each other
- ❖ Takehome Exams (2)
 - **Midterm:** will be *around* May 7
 - **Final:** will be *around* June 9

Grading

- ❖ **Exercises:** 30% total
 - Submitted via GradeScope (under your UW email)
 - Graded on correctness and style by autograders and TAs
- ❖ **Projects:** 45% total
 - Submitted via GitLab; must tag commit that you want graded
 - Binaries provided if you didn't get previous part working
- ❖ **Exams:** Midterm (10%) and Final (10%)
 - Take-home; not traditional 333 exams
- ❖ **Course-wide Participation:** 5%
 - Many ways to earn credit here, relatively lenient on this
- ❖ **More details on course website**
 - You **must** read the syllabus there – you are responsible for it

Deadlines and Student Conduct

- ❖ Late policies
 - Exercises: no late submissions accepted, due 11 am
 - Homework: 4 late day “tokens” for quarter, max 2 per homework
 - Need to get things done on time – difficult to catch up!

- ❖ Academic Integrity (**read** the full policy on the web)
 - I trust you implicitly and will follow up if that trust is violated
 - In short: don't attempt to gain credit for something you didn't do and don't help others do so either
 - This does **not** mean suffer in silence – learn from the course staff and peers, talk, share ideas; *but* don't share or copy work that is supposed to be yours

Hooked on Gadgets

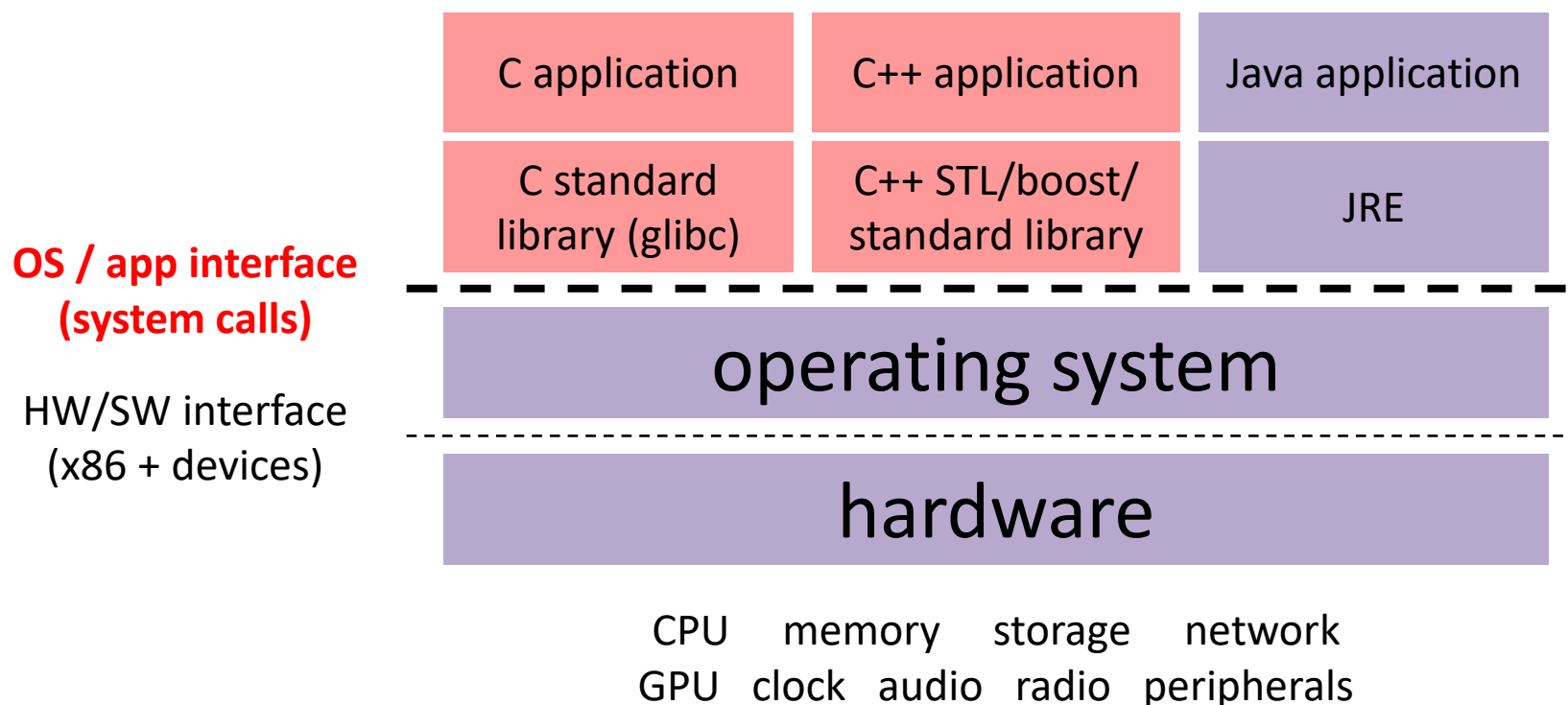
- ❖ Gadgets reduce focus and learning
 - Bursts of info (*e.g.*, emails, IMs, etc.) are *addictive*
 - Heavy multitaskers have more trouble focusing and shutting out irrelevant information
 - <http://www.npr.org/2016/04/17/474525392/attention-students-put-your-laptops-away>
 - Seriously, you will learn more if you use **paper** instead!!!

- ❖ Suggestions to try
 - Disable notifications on your primary machine
 - Do Not Disturb mode on your smartphone
 - Close (not minimize) your non-class browser windows & tabs

Lecture Outline

- ❖ Course Policies
 - <https://courses.cs.washington.edu/courses/cse333/21sp/syllabus/>
 - Summary here, but you *must* read the full details online
- ❖ **Course Introduction**
- ❖ C Reintroduction

Course Map: 100,000 foot view



Systems Programming

- ❖ The programming skills, engineering discipline, and knowledge you need to build a system
 - **Programming:** C / C++
 - **Discipline:** testing, debugging, performance analysis
 - **Knowledge:** long list of interesting topics
 - Concurrency, OS interfaces and semantics, techniques for consistent data management, distributed systems algorithms, ...
 - Most important: a deep(er) understanding of the “layer below”



Discipline?!?

- ❖ Cultivate good habits, encourage clean code
 - Coding style conventions
 - Unit testing, code coverage testing, regression testing
 - Documentation (code comments, design docs)
 - Code reviews

- ❖ Will take you a lifetime to learn, but oh-so-important, especially for systems code
 - Avoid write-once, read-never code
 - Treat assignment submissions in this class as production code
 - Comments must be updated, no commented-out code, no extra (debugging) output

Style Grading in 333

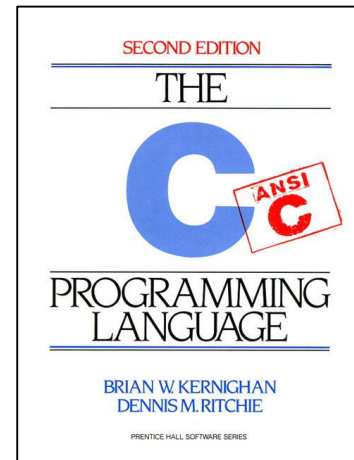
- ❖ A **style guide** is a “set of standards for the writing, formatting, and design of documents” – in this case, code
- ❖ No style guide is perfect
 - Inherently limiting to coding as a form of expression/art
 - Rules should be motivated (*e.g.*, consistency, performance, safety, readability), even if not everyone agrees
- ❖ In 333, we will use a subset of the Google C++ Style Guide
 - Want you to experience adhering to a style guide
 - Hope you view these more as *design decisions* to be considered rather than rules to follow to get a grade
 - We acknowledge that judgments of language implicitly encode certain values and not others

Lecture Outline

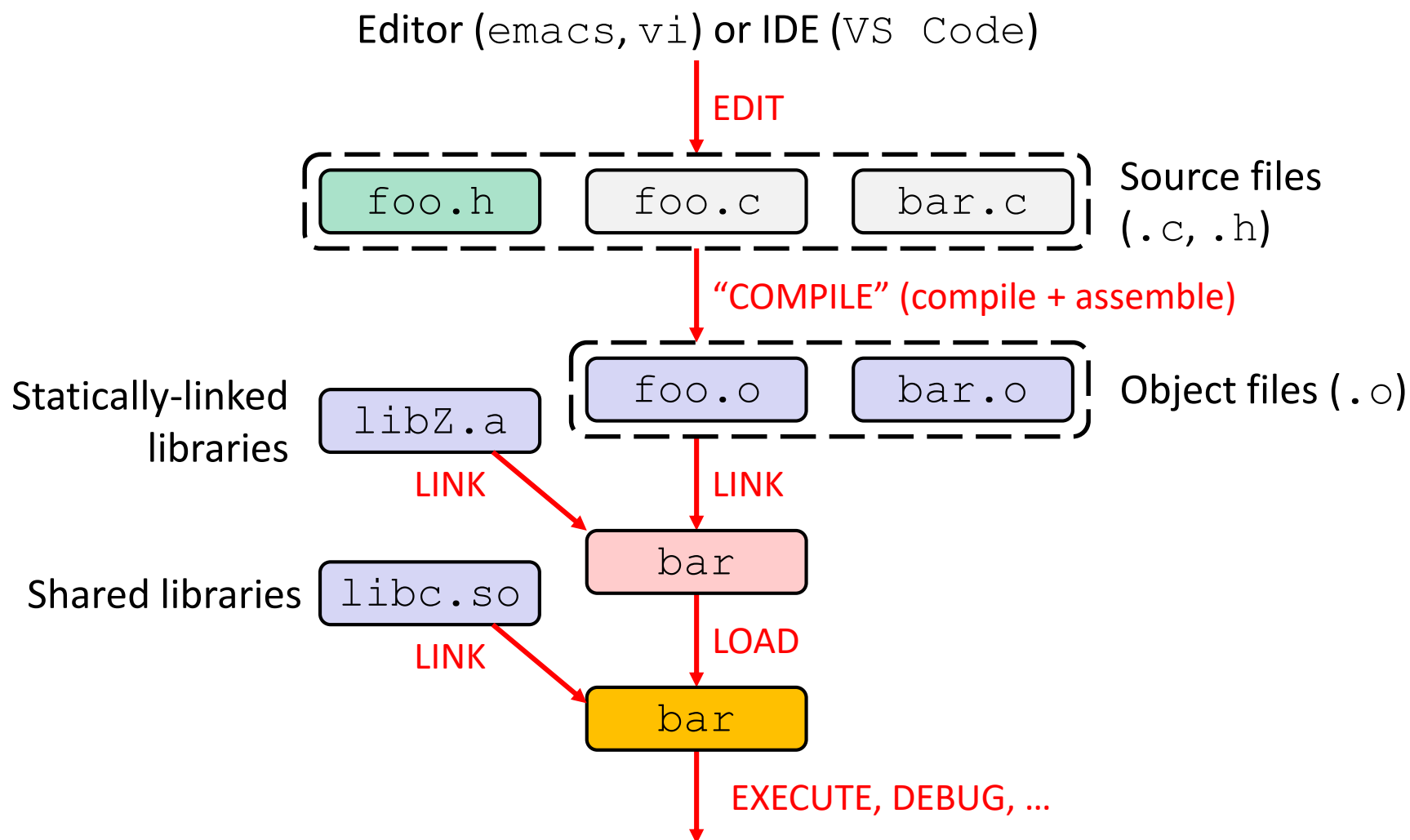
- ❖ Course Policies
 - <https://courses.cs.washington.edu/courses/cse333/21sp/syllabus/>
 - Summary here, but you *must* read the full details online
- ❖ Course Introduction
- ❖ **C Reintroduction**
 - **Workflow, Variables, Functions**

C

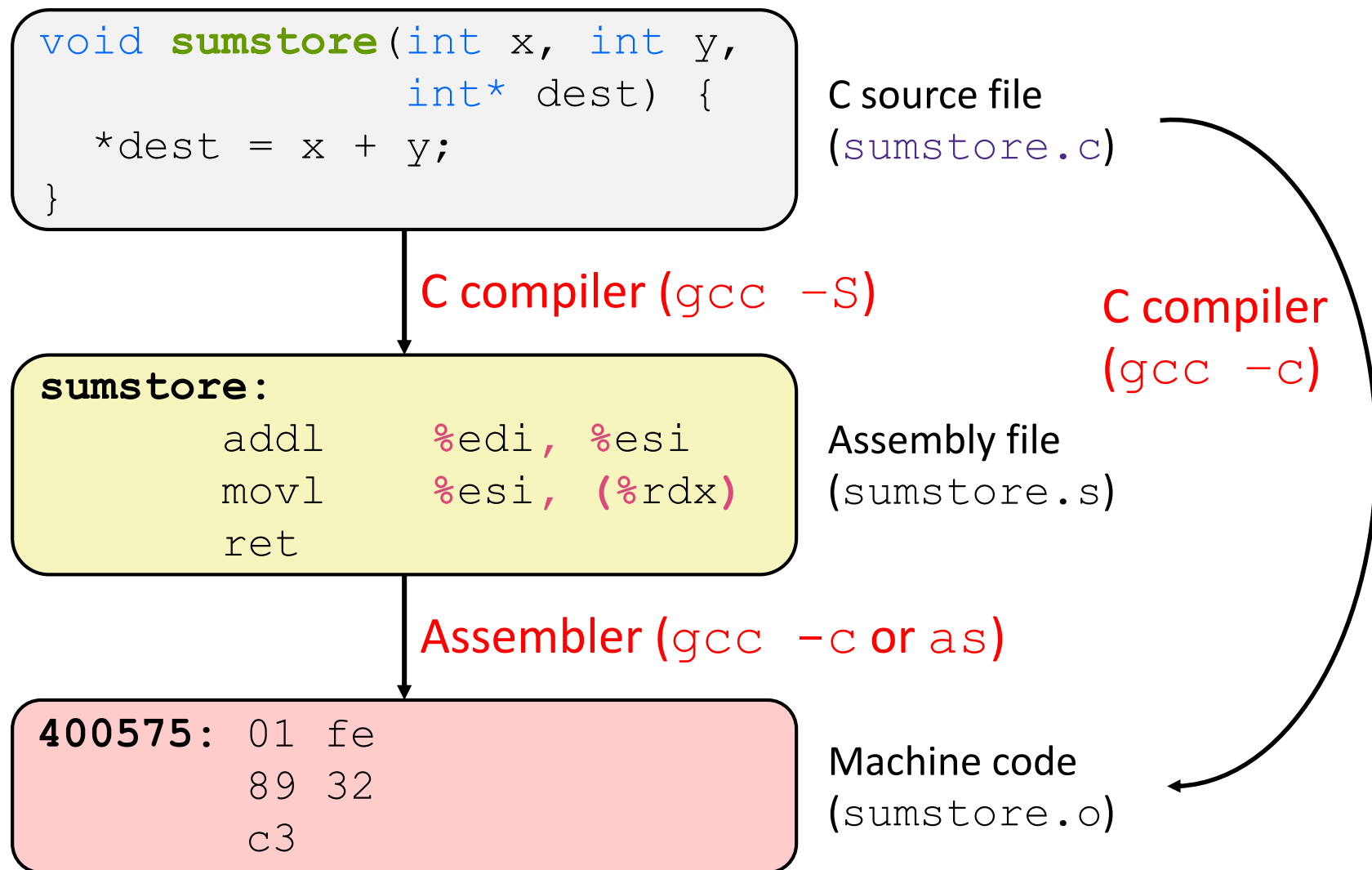
- ❖ Created in 1972 by Dennis Ritchie
 - Designed for creating system software
 - Portable across machine architectures
 - Most recently updated in 1999 (C99) and 2011 (C11)
 - There's also C17, which is a bug-fix version of C11.
- ❖ Characteristics
 - “Low-level” language that allows us to exploit underlying features of the architecture – **but easy to fail spectacularly (!)**
 - Procedural (not object-oriented)
 - “Weakly-typed” or “type-unsafe”
 - Small, basic library compared to Java, C++, most others....



C Workflow



C to Machine Code





Generic C Program Layout

```
#include <system_files>
#include "local_files"

#define macro_name macro_expr

/* declare functions */
/* declare external variables & structs */

int main(int argc, char* argv[]) {
    /* the innards */
}

/* define other functions */
```

C Syntax: `main`

- ❖ To get command-line arguments in `main`, use:

```
int main(int argc, char* argv[])
```

- ❖ What does this mean?

- `argc` contains the number of strings on the command line (the executable name counts as one, plus one for each argument).
- `argv` is an array containing *pointers* to the arguments as strings (more on pointers later)

- ❖ Example: `$ foo hello 87`

- `argc = 3`
- `argv[0] = "foo", argv[1] = "hello", argv[2] = "87"`



When Things Go South...

❖ Errors and Exceptions

- C does not have exception handling (no `try/catch`)
- Errors are returned as integer error codes from functions
 - Standard codes found in `stdlib.h`:
`EXIT_SUCCESS` (usually 0) and `EXIT_FAILURE` (non-zero)
 - Return value from `main` is a status code
- Because of this, error handling is ugly and inelegant

❖ Crashes

- If you do something bad, you hope to get a “segmentation fault” (believe it or not, this is the “good” option)

Java vs. C (351 refresher)

- ❖ Are Java and C mostly similar (S) or significantly different (D) in the following categories?
 - List any differences you can recall (even if you put 'S')

Language Feature	S/D	Differences in C
Control structures		
Primitive datatypes		
Operators		
Casting		
Arrays		
Memory management		

Primitive Types in C

❖ Integer types

- `char`, `int`

❖ Floating point

- `float`, `double`

❖ Modifiers

- `short` [int]
- `long` [int, double]
- `signed` [char, int]
- `unsigned` [char, int]

C Data Type	32-bit	64-bit	printf
char	1	1	%c
short int	2	2	%hd
unsigned short int	2	2	%hu
int	4	4	%d / %i
unsigned int	4	4	%u
long int	4	8	%ld
long long int	8	8	%lld
float	4	4	%f
double	8	8	%lf
long double	12	16	%Lf
pointer	4	8	%p

Typical sizes – see `sizeofs.c`



C99 Extended Integer Types

- ❖ Solves the conundrum of “how big is an `long int`?”

```
#include <stdint.h>

void foo(void) {
    int8_t  a; // exactly 8 bits, signed
    int16_t b; // exactly 16 bits, signed
    int32_t c; // exactly 32 bits, signed
    int64_t d; // exactly 64 bits, signed
    uint8_t w; // exactly 8 bits, unsigned
    ...
}
```

```
void sumstore(int x, int y, int* dest) {
```



```
void sumstore(int32_t x, int32_t y, int32_t* dest) {
```

Basic Data Structures

- ❖ C does not support objects!!!
- ❖ **Arrays** are contiguous chunks of memory
 - Arrays have no methods and do not know their own length
 - Can easily run off ends of arrays in C – **security bugs!!!**
- ❖ **Strings** are null-terminated char arrays
 - Strings have no methods, but `string.h` has helpful utilities

```
char* x = "hello\n";
```



- ❖ **Structs** are the most object-like feature, but are just collections of fields – no “methods” or functions

Function Definitions

❖ Generic format:

```
returnType fname(type param1, ..., type paramN) {  
    // statements  
}
```

```
// sum of integers from 1 to max  
int32_t sumTo(int32_t max) {  
    int32_t i, sum = 0;  
  
    for (i = 1; i <= max; i++) {  
        sum += i;  
    }  
  
    return sum;  
}
```

Function Ordering

- ❖ You *shouldn't* call a function that hasn't been declared yet

Note: code examples from slides are posted on the course website for you to experiment with!

sum_badorder.c

```
int main(int argc, char** argv) {
    printf("sumTo(5) is: %d\n", sumTo(5));
    return EXIT_SUCCESS;
}

// sum of integers from 1 to max
int32_t sumTo(int32_t max) {
    int32_t i, sum = 0;

    for (i = 1; i <= max; i++) {
        sum += i;
    }
    return sum;
}
```

Solution 1: Reverse Ordering

- ❖ Simple solution; however, imposes ordering restriction on writing functions (who-calls-what?)

sum_betterorder.c

```
// sum of integers from 1 to max
int32_t sumTo(int32_t max) {
    int32_t i, sum = 0;

    for (i = 1; i <= max; i++) {
        sum += i;
    }
    return sum;
}

int main(int argc, char** argv) {
    printf("sumTo(5) is: %d\n", sumTo(5));
    return EXIT_SUCCESS;
}
```



Solution 2: Function Declaration

- ❖ Teaches the compiler arguments and return types; function definitions can then be in a logical order
 - Function comment usually by the *prototype*

sum_declared.c

```
// sum of integers from 1 to max
int32_t sumTo(int32_t); // func prototype

int main(int argc, char** argv) {
    printf("sumTo(5) is: %d\n", sumTo(5));
    return EXIT_SUCCESS;
}

int32_t sumTo(int32_t max) {
    int32_t i, sum = 0;
    for (i = 1; i <= max; i++) {
        sum += i;
    }
    return sum;
}
```

Function Declaration vs. Definition

- ❖ C/C++ make a careful distinction between these two
- ❖ **Definition:** the thing itself
 - *e.g.* code for function, variable definition that creates storage
 - Must be **exactly one** definition of each thing (no duplicates)
- ❖ **Declaration:** description of a thing
 - *e.g.* function prototype, external variable declaration
 - Often in header files and incorporated via `#include`
 - Should also `#include` declaration in the file with the actual definition to check for consistency
 - Needs to appear in **all files** that use that thing
 - Should appear before first use

Multi-file C Programs

C source file 1
(sumstore.c)

```
void sumstore(int x, int y, int* dest) {  
    *dest = x + y;  
}
```

C source file 2
(sumnum.c)

```
#include <stdio.h>  
  
void sumstore(int x, int y, int* dest);  
  
int main(int argc, char** argv) {  
    int z, x = 351, y = 333;  
    sumstore(x, y, &z); <- used  
    printf("%d + %d = %d\n", x, y, z);  
    return 0;  
}
```

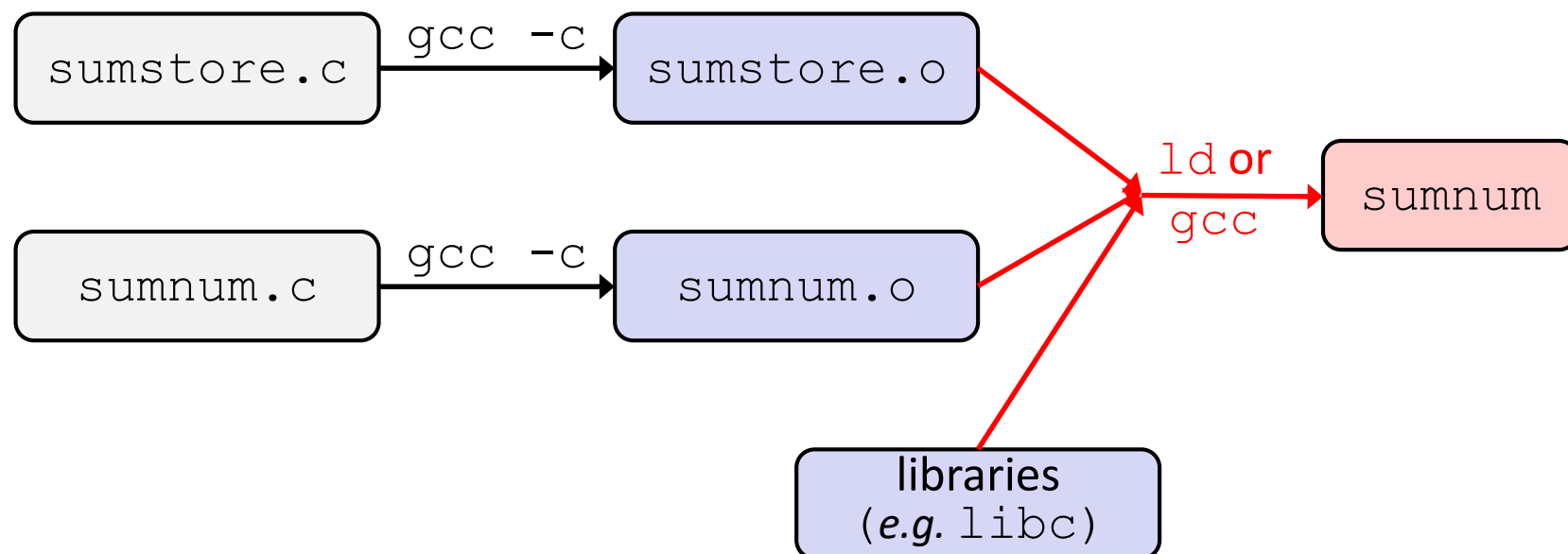
Note: not good
style. More on
multiple files in
lecture 5

Compile together:

```
$ gcc -o sumnum sumnum.c sumstore.c
```

Compiling Multi-file Programs

- ❖ The **linker** combines multiple object files plus statically-linked libraries to produce an executable
 - Includes many standard libraries (*e.g.* `libc`, `crt1`)
 - A *library* is just a pre-assembled collection of `.o` files



Polling Question

- ❖ Which of the following statements is FALSE?
 - Vote at <http://PollEv.com/cse333>
 - A. **With the standard `main()` syntax, it is always safe to use `argv[0]`.**
 - B. **We can't use `uint64_t` on a 32-bit machine because there isn't a C integer primitive of that length.**
 - C. **Using function declarations is beneficial to both single- and multi-file C programs.**
 - D. **When compiling multi-file programs, not all linking is done by the Linker.**
 - E. **We're lost...**

To-do List

- ❖ Make sure you're registered on Canvas, Ed Discussion, Gradescope, and Poll Everywhere
 - All user IDs should be your **uw.edu** email address
- ❖ Explore the website *thoroughly*: <http://cs.uw.edu/333>
- ❖ Computer setup: CSE lab, attu, or CSE Linux VM
- ❖ **Exercise 1 is due 11 am on Friday**
 - Find exercise spec on website, submit via Gradescope
 - Course "CSE 333" under "Spring 2021", Assignment "Exercise 1", then drag-n-drop file(s)!
 - Sample solution will be posted Friday afternoon
 - **Hint:** look at documentation for [stdlib.h](#), [string.h](#), and [inttypes.h](#)
- ❖ Pre-Quarter survey up on canvas. Due Friday @ 11:59 pm
 - Answers are anonymous.