Intro, C refresher
CSE 333 Fall 2022

Instructor: Hal Perkins

Teaching Assistants:

Nour Ayad	Frank Chen	Nick Durand
Dylan Hartono	Humza Lala	Kenzie Mihardja
Bennedict Soesanto	Chanh Truong	Justin Tysdal
Tanay Vakharia	Timmy Yang	

Lecture Outline

- ❖ **Course Introduction**

- ❖ Course Policies
  - ▪ https://courses.cs.washington.edu/courses/cse333/22au/syllabus.html

- ❖ C Intro

To get started…

❖ Welcome back. With luck this year will be a wholeeeee lot better than the last couple

❖ UW mask policy: optional but strongly recommended for at least the first couple of weeks
 ▪ No judgement: people can do what they will, and do not read anything into how others do or don't use masks
 ▪ Please be courteous and allow people distance or otherwise help them feel comfortable if asked
 ▪ Lectures: would like to be able work without if that's ok with everyone. Is OK?

❖ Stay healthy! If you do come down with something, please stay home until recovered and not contagious
 ▪ Lectures are on panopto if you do need to miss

It's almost normal – until it's not…

- ❖ This has been a strange world for the last few years and there's still a lot of stress for many people

- ❖ Please speak up if things aren't (or are!) going well
  - ▪ We can often help if we know about things, so stay in touch with TAs, instructor, advising, friends and peers, others
  - ▪ Don't try to "tough it out" or pretend it will get better if you just ignore it – speak up!

- ❖ We're all in this together but not all in the same way, so please show understanding and compassion for each other and help when you can – both in and outside of class

- ❖ But also, we've had a summer to reset, and with luck we'll have energy and focus to do great things this year!

Introductions: Course Staff

❖ Hal Perkins (instructor)
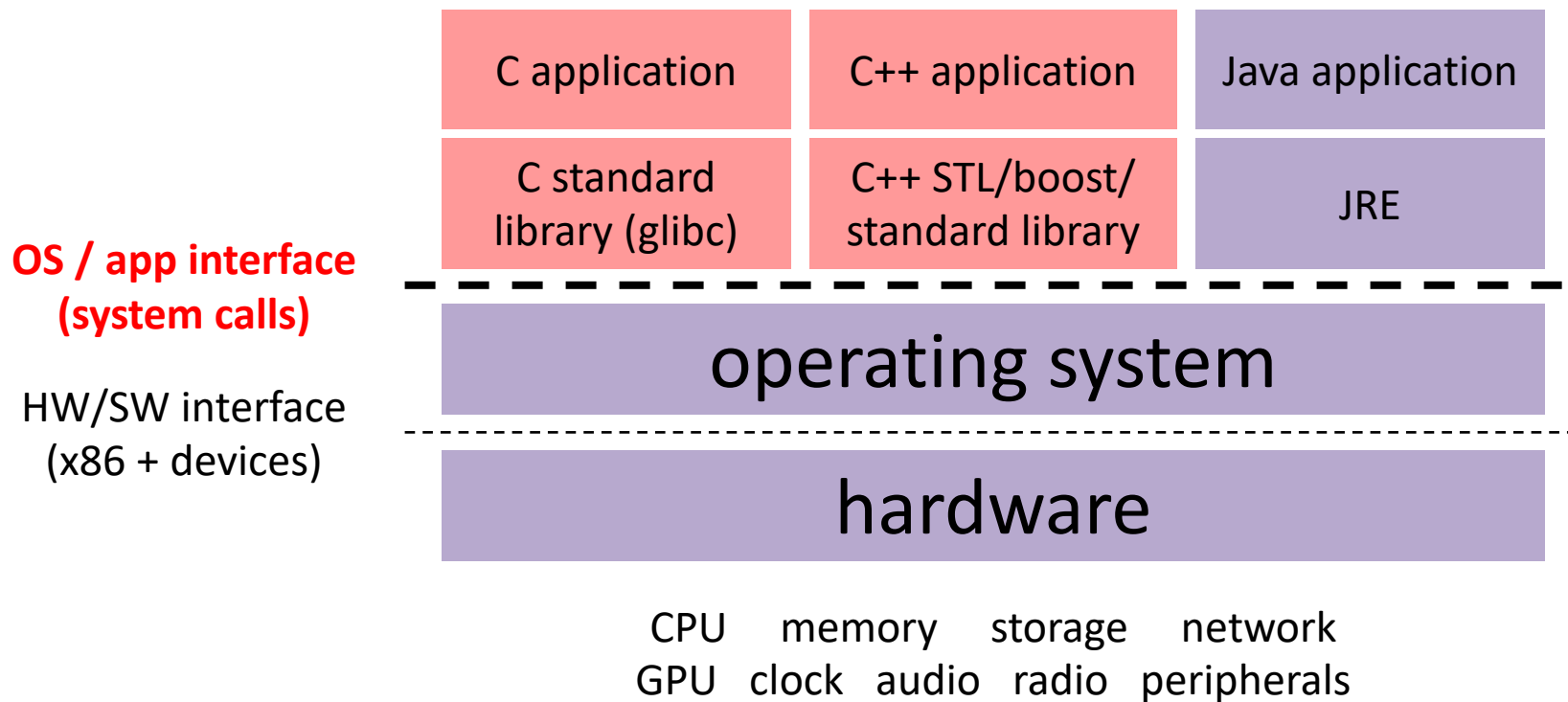
▪ Long-time CSE faculty member and CSE 333 veteran

❖ TAs:

▪ Nour Ayad, Frank Chen, Nick Durand, Dylan Hartono, Humza Lala, Kenzie Mihardja, Bennedict Soesanto, Chanh Truong, Justin Tysdal, Tanay Vakharia, and Timmy Yang

▪ Available in section, office hours, and discussion group

▪ An invaluable source of information and help

❖ Get to know us

▪ We are here to help you succeed!

Introductions: Students

- ❖ ~165 students this quarter

- ❖ Expected background
  - ▪ **Prereq:** CSE 351 – C, pointers, memory model, linker, system calls
  - ▪ CSE 391 or Linux skills needed for CSE 351 assumed

Course Map: 100,000 foot view

C application	C++ application	Java application
C standard library (glibc)	C++ STL/boost/ standard library	JRE

OS / app interface (system calls)

- -

operating system

HW/SW interface (x86 + devices)

- -

hardware

CPU   memory   storage   network
GPU clock audio radio peripherals

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

Lecture Outline

❖ Course Introduction

❖ **Course Policies**

  ▪ https://courses.cs.washington.edu/courses/cse333/22au/syllabus.html

  ▪ Summary/highlights here, but you *must* read the full details online

❖ C Intro

Communication

- ❖ Website: http://cs.uw.edu/333
  - Schedule, policies, materials, assignments, etc.
- ❖ Discussion: Ed group linked to course home page
  - Log in using your **@uw.edu** Google identity (not cse)
  - Ask and answer questions – staff will monitor and contribute
  - Use private messages for questions about detailed code, etc.
- ❖ Messages to staff: for things not suitable for ed chat messages, please send email to cse333-staff@cs.uw.edu. Reaches all staff so the right person can help out quickly, helps follow up until resolved
  - (***don't*** email to instructor or individual TAs if possible – we can get quick answers for you and coordinate better if it goes to the staff)
- ❖ Announcements: will use broadcast Ed messages to send "things everyone must read and know"
- ❖ Office Hours: spread throughout the week
  - Schedule posted shortly and will start today!

Course Components

- ❖ Lectures (~30)
  - ▪ Introduce the concepts; take notes!!!

- ❖ Sections (10)
  - ▪ Applied concepts, important tools and skills for assignments, clarification of lectures, exam review and preparation

- ❖ Programming Exercises (~18)
  - ▪ Roughly one per lecture, due the morning before the next lecture
  - ▪ Coarse-grained grading (check plus/check/check minus = 0, 1, 2, or 3)

- ❖ Programming Projects (0+4)
  - ▪ Warm-up, then 4 "homeworks" that build on each other

- ❖ Midterm (in-class) and final exam
  - ▪ Goal is to revisit and internalize concepts

Grading (tentative)

- ❖ **Exercises:** ~30%
    - Submitted via GradeScope (account info mailed earlier today)
    - Evaluated on correctness and code quality
- ❖ **Projects:** ~45% total
    - Submitted via GitLab; must tag commit that you want graded
    - "does it work" and code quality both matter
    - Binaries provided if you didn't get previous part working or prefer to start with a known good solution to previous parts
- ❖ **Exams:** Midterm: ~10%, Final: ~15%

- ❖ More details on course website
    - You **must** read the syllabus there – you are responsible for it

Deadlines and Student Conduct

❖ Late policies

  ▪ <u>Exercises</u>: no late submissions accepted, due 10 am before class

  ▪ <u>Projects</u>: 4 late days for entire quarter, max 2 per project

  ▪ Need to get things done on time – difficult to catch up!

    • But we will work with you if unusual circumstances / problems

❖ Academic Integrity (**read** the full policy on the web)

  ▪ We 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

14

And off we go…

❖ Goal is to figure out setup and computing infrastructure right away so we don't put that off and then have a crunch later in the quarter

❖ So:

- First exercise out today, due Friday morning **10 am** before class
- Warmup/logistics for larger projects in sections Thursday
  - HW0 (the warmup project) published this afternoon and gitlab repos created then. Feel free to ignore until sections tomorrow and we'll walk through the whole thing.
  - Bring a laptop to sections tomorrow! We should have time to go through some of the initial configuration parts for hw0.

Gadgets (1)

- ❖ Gadgets reduce focus and learning
  - Bursts of info (*e.g.* emails, IMs, notifications, 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!!!
    - (even compared to note-taking on a tablet, although that is better than a keyboard, and that is way better than just "watching the show")

Gadgets (2)

❖ So how should we deal with laptops/phones/etc.?

- Just say no!

- No open gadgets during class (really!)
  - Unless you're actually using a tablet or something to take notes

- Urge to search? – ask a question! Everyone benefits!!

- You may close/turn off non-notetaking electronic devices now

- Pull out a piece of paper and pen/pencil instead ☺

Deep Breath….

❖ Any questions, comments, observations, before we go on to, uh, some technical stuff?

Lecture Outline

- ❖ Course Introduction

- ❖ Course Policies
  - ▪ https://courses.cs.washington.edu/courses/cse333/22au/syllabus.html

- ❖ **C Intro**
  - ▪ **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) and 2017 (C17)

- ❖ Characteristics
  - "Low-level" language that allows us to exploit underlying features of the architecture – but easy to fail spectacularly (!)
  - Procedural (not object-oriented)
  - Typed but unsafe (possible to bypass the type system)
  - Small, basic library compared to Java, C++, most others….
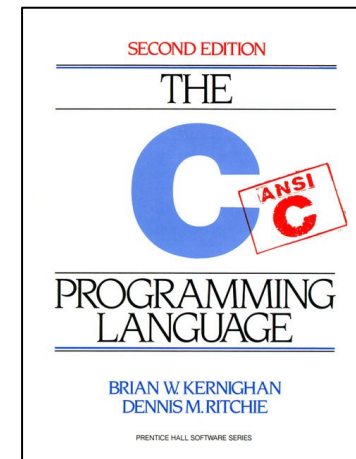
Generic C Program Layout

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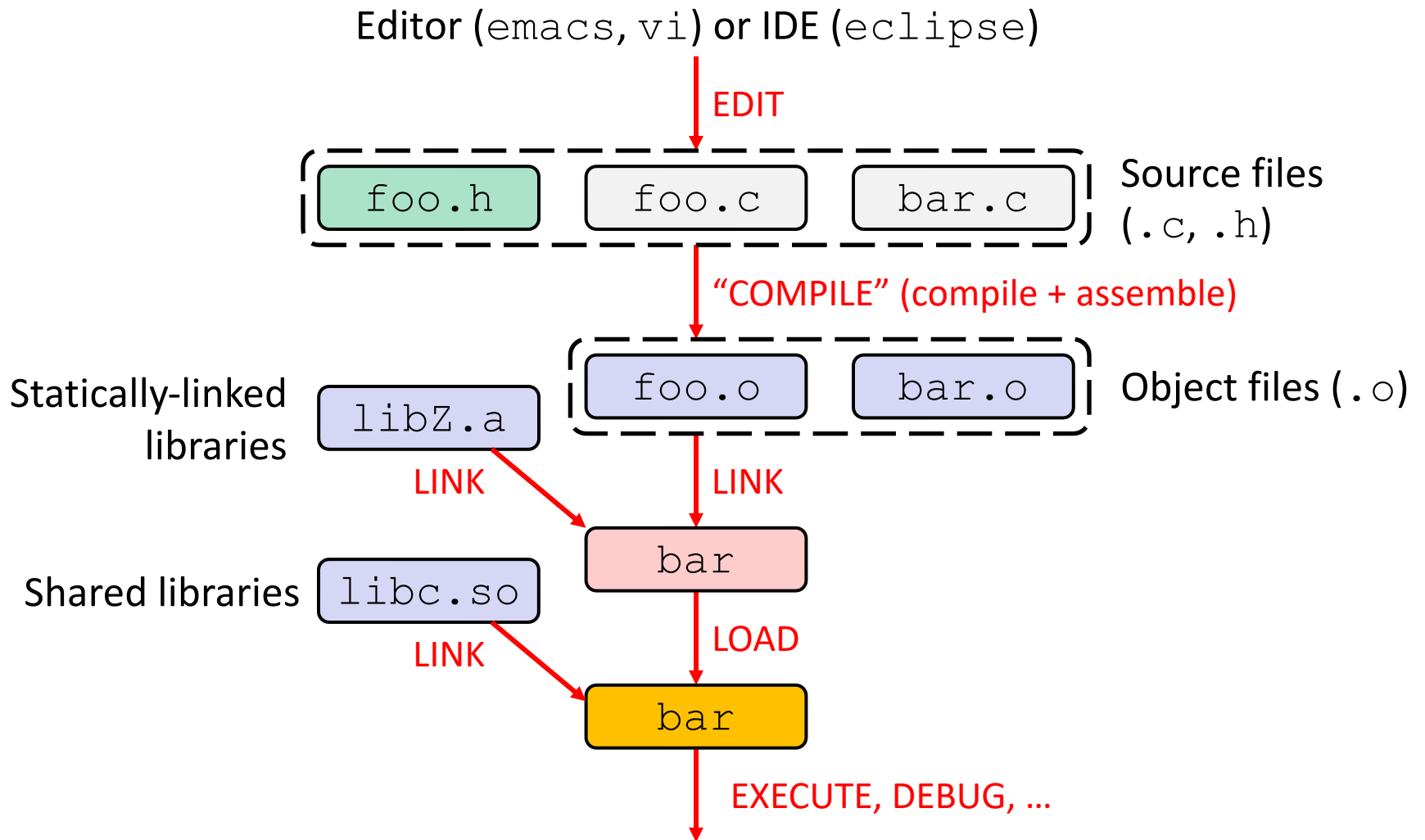
```
int main(int argc, char* argv[])
```

❖ 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).

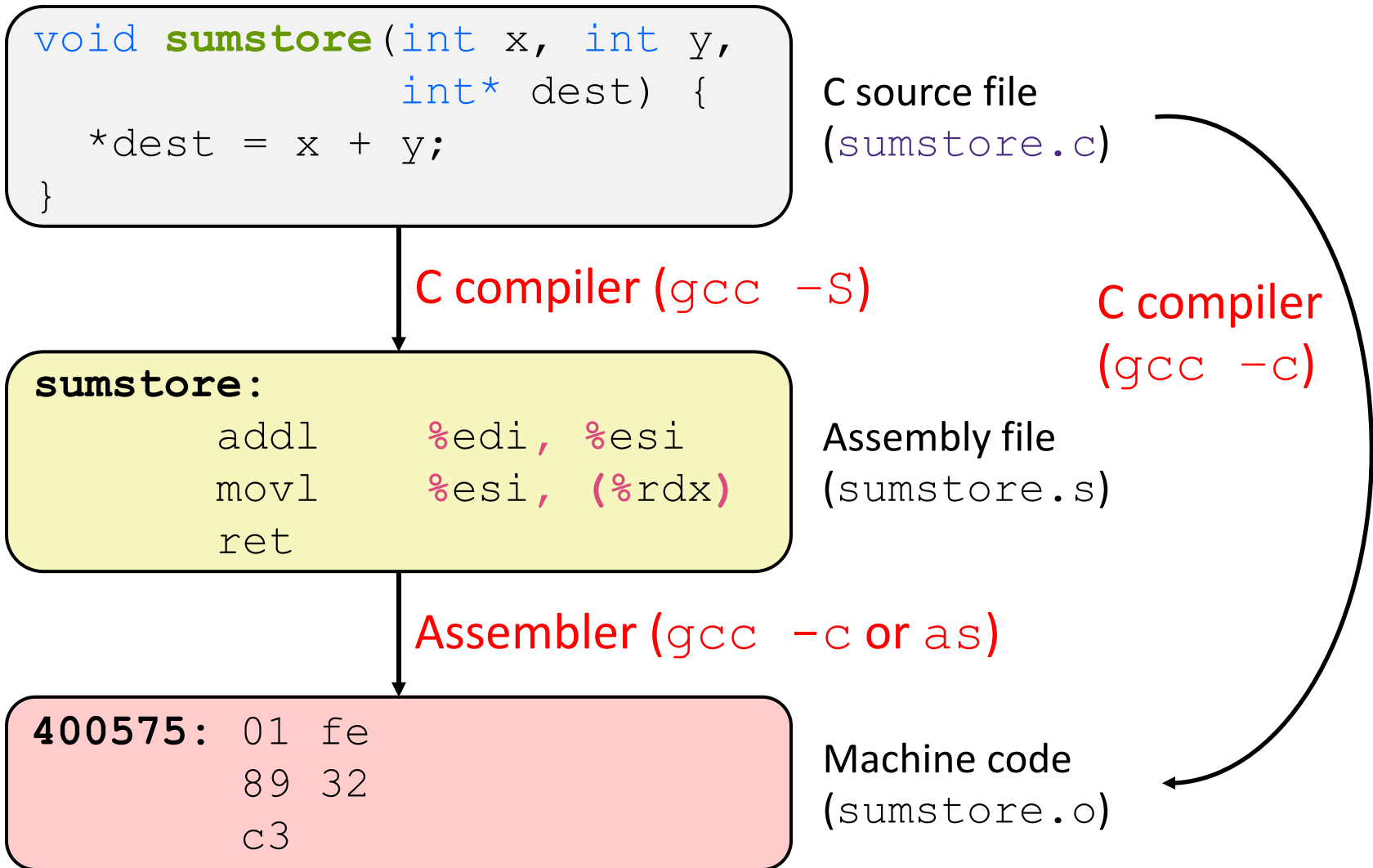- ▪ `argv` is an array containing *pointers* to the arguments as strings (more on pointers later)

❖ <u>Example</u>: `$ foo hello 87`

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

C Workflow

Editor (`emacs`, `vi`) or IDE (`eclipse`)

↓ EDIT

| `foo.h` | `foo.c` | `bar.c` |

Source files (`.c`, `.h`)

↓ "COMPILE" (compile + assemble)

Statically-linked libraries `libZ.a`

| `foo.o` | `bar.o` |

Object files (`.o`)

LINK   LINK

`bar`

Shared libraries `libc.so`

LINK

`bar`

↓ LOAD

EXECUTE, DEBUG, …

C to Machine Code

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

C source file
(`sumstore.c`)

C compiler (`gcc -S`)

C compiler
(`gcc -c`)

```
sumstore:
        addl    %edi, %esi
        movl    %esi, (%rdx)
        ret
```

Assembly file
(`sumstore.s`)

Assembler (`gcc -c` or `as`)

```
400575: 01 fe
        89 32
        c3
```

Machine code
(`sumstore.o`)

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

  ▪ Because of this, error handling is ugly and inelegant

❖ Processes return an "exit code" when they terminate

  ▪ Can be read and used by parent process (shell or other)

    • In main: return EXIT_SUCCESS; or return EXIT_FAILURE; (e.g., 0 or 1)

❖ 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?

Language Feature	S/D	Differences in C
Control structures	S	
Primitive datatypes	S/D	Similar but sizes can differ (char, esp.), unsigned, no boolean, uninitialized data, …
Operators	S	Java has >>>, C has ->
Casting	D	Java enforces type safety, C does not
Arrays	D	Not objects, don't know their own length, no bounds checking
Memory management	D	Manual (malloc/free), no garbage collection

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`?"

```c
#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
  ...
}
```

Use extended types in most cse333 code

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

But int is usually fine for simple ints

```c
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";`   x | h | e | l | l | o | \n | \0 |

❖ **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
int sumTo(int max) {
  int 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

sum_badorder.c

```c
#include <stdio.h>

int main(int argc, char** argv) {
  printf("sumTo(5) is: %d\n", sumTo(5));
  return 0;
}

// sum of integers from 1 to max
int sumTo(int max) {
  int 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

```c
#include <stdio.h>

// sum of integers from 1 to max
int sumTo(int max) {
  int 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 0;
}
```

Solution 2: Function Declaration

❖ Teaches the compiler arguments and return types; function definitions can then be in a logical order

sum_declared.c

Hint: code examples from slides are on the course web for you to experiment with

```c
#include <stdio.h>

int sumTo(int);   // func prototype

int main(int argc, char** argv) {
  printf("sumTo(5) is: %d\n", sumTo(5));
  return 0;
}

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

34

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

definition

C source file 1
(sumstore.c)

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

C source file 2
(sumnum.c)

declaration

```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);
    printf("%d + %d = %d\n",x,y,z);
    return 0;
}
```
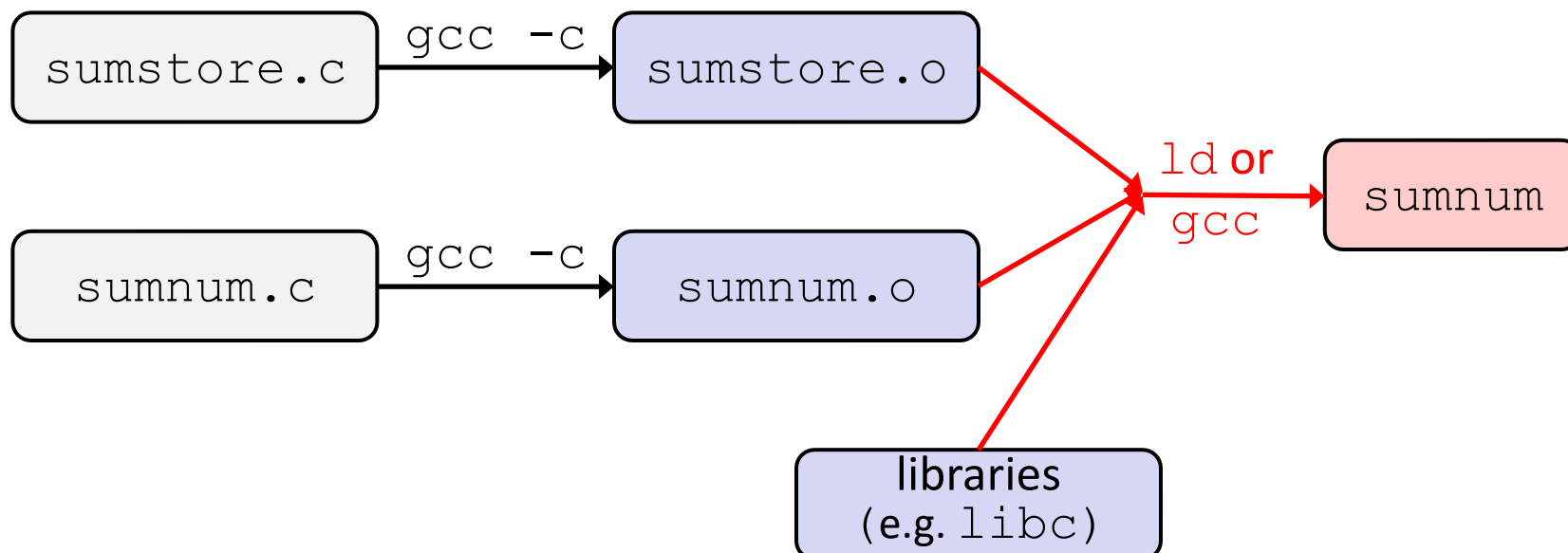
Compile together:
```
$ gcc -o sumnum sumnum.c sumstore.c
```

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

To-do List

- ❖ Explore the website *thoroughly*: http://cs.uw.edu/333
- ❖ Computer setup: CSE labs, attu, or CSE Linux VM
  - ▪ Late upgrades today: get 22au VM when available if you use it
- ❖ Exercise 0 is due 10 am Friday before class
  - ▪ Find exercise spec on website, submit via Gradescope
  - ▪ Sample solution will be posted Wednesday after class
  - ▪ Give it your best shot to get it done on time
- ❖ Gradescope accounts created before class today
  - ▪ Userid is your uw.edu email address
  - ▪ Exercise submission: find CSE 333 22au, click on the exercise, drag-n-drop file(s)! That's it!!
- ❖ Project repos created and hw0 out later today
  - ▪ All will become clear in sections tomorrow! Bring your laptop!! ☺