Intro, C Refresher
CSE 333 Fall 2023

Instructor: Chris Thachuk

Teaching Assistants:

Ann Baturytski  Humza Lala
Yuquan Deng    Alan Li
Noa Ferman     Leanna Mi Nguyen
James Froelich Chanh Truong
Hannah Jiang   Jennifer Xu
Yegor Kuznetsov
Introductions: Instructor

❖ Chris (he/him)

From Canada (with lots of moving around)
• Windsor (CA) → Toronto (CA) → Vancouver (CA) → Mexico City (MX) → Vancouver (CA) → Oxford (UK) → Pasadena (USA) → Seattle (USA)

I like: research, teaching, training, hiking, sci-fi

As a high school student (many years ago) I won a contest and was gifted a copy of “Visual Studio C++” and have been programming in C/C++ ever since

I research systems programming of molecules such as DNA!

```c
int main(int argc, char** argv) {
    make_triangle_from_DNA();
    return EXIT_SUCCESS;
}
```
Introductions: Teaching Assistants

Available in section, office hours, and discussion board

❖ More than anything, we want you to feel...
  Comfortable and welcome in this space
  Able to learn and succeed in this course
  Comfortable reaching out if you need help or want change
Introductions: Students

❖ ~170 students registered

There are no overload forms or waiting lists for CSE courses
• Majors must add using the UW system as space becomes available
• Non-majors would have already needed to petition before today

❖ 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
Introductions: Students

- Get to know each other! Help each other out!
  Working well with others is a valuable life skill
  Helping others is rewarded in this course
  Take advantage of partner work, where permissible, to
  learn, not just get a grade
    • Good chance to learn collaboration tools and tricks
Lecture Outline

❖ **Course Policies**


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/47406/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**
Course Components

❖ Lectures (26+1)
   Introduce the concepts; take notes!!!

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

❖ Programming Exercises (12-15)
   One due roughly every 2-4 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
   Homework 2, 3 and 4 can be completed with a partner

❖ In-Person Exams (2)
   Midterm (TBD)
   Final (Dec. 13)
Grading

❖ **Exercises:** 30% total
  Submitted via GradeScope (under your UW email)
  Graded on correctness and style by autograders and TAs

❖ **Projects:** 43% total
  Submitted via GitLab; must tag commit that you want graded
  Binaries provided if you didn’t get previous part working
  Graded on test suite, manual tests, and style

❖ **Exams:** Midterm (12%) and Final (12%)
  In-person; questions to validate concepts learned

❖ **Effort, Participation, Altruism:** 3%
  Many ways to earn credit here, relatively lenient on this
Student Conduct

❖ 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

❖ If you find yourself in a situation where you are tempted
  to perform academic misconduct, please reach out to
  Chris to explain your situation instead

  See the Extenuating Circumstances section of the syllabus
Deadlines (flexibility policy)

❖ Exercises
  ❖ Must be submitted via Gradescope by 10pm on due date
  ❖ Submissions will be accepted until 10am the next morning, *without penalty*
  ❖ You will be awarded a ‘promptness bonus’ for consistently meeting the original deadline

❖ Homework
  ❖ Must be submitted & tagged on Gitlab by 10pm on due date
  ❖ Submissions will be accepted up to two days later, *without penalty (weekends count as one day)*
  ❖ You will be awarded a ‘promptness bonus’ for consistently meeting the original deadline
Lecture Outline

❖ Course Policies
   https://courses.cs.washington.edu/courses/cse333/23au/syllabus/
   Summary here, but you must read the full details online

❖ Course Introduction

❖ C Reintroduction
Course Map: 100,000 foot view

OS / app interface (system calls)

HW/SW interface (x86 + devices)

operating system

hardware

C application

C++ application

Java application

C standard library (glibc)

C++ STL/boost/standard library

JRE

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
  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/cse333/23au/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

**Editor** (emacs, vi) or **IDE** (VS Code)

- Source files (.c, .h)
- Object files (.o)

**Statically-linked libraries**
- libZ.a
  - LINK

**Shared libraries**
- libc.so
  - LINK

**“COMPILE” (compile + assemble)**
- foo.o
- bar.o

**LINK**
- bar

**LOAD**
- bar

**EDIT**

**EXECUTE, DEBUG, …**
C to Machine Code

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

**C source file**
```
(sumstore.c)
```

**C compiler**
```
gcc –S
```

**C compiler**
```
gcc –c
```

**Assembly file**
```
(sumstore.s)
```

**Assembler**
```
gcc –c or as
```

**Machine code**
```
(sumstore.o)
```

400575: 01 fe 89 32 c3

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

❖ **Example:**

```
$ foo hello 87
```

- `argc = 3`
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’)

<table>
<thead>
<tr>
<th>Language Feature</th>
<th>S/D</th>
<th>Differences in C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control structures</td>
<td>S</td>
<td>if-else if-else, switch, while, for are all the same.</td>
</tr>
<tr>
<td>Primitive datatypes</td>
<td>S/D</td>
<td>S: same/similar names</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D: char (ASCII, 1 byte), machine-dependent sizes, no built-in boolean type, not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>initialized.Modifiers.</td>
</tr>
<tr>
<td>Operators</td>
<td>S</td>
<td>Almost all match. One notable difference is no &gt;&gt;&gt;&gt; for logical shift.</td>
</tr>
<tr>
<td>Casting</td>
<td>D</td>
<td>Java has type-safe casting, while C does not.</td>
</tr>
<tr>
<td>Arrays</td>
<td>D</td>
<td>Not objects; don’t know own length.</td>
</tr>
<tr>
<td>Memory management</td>
<td>D</td>
<td>Explicit memory management (malloc/free). No automatic garbage collection.</td>
</tr>
</tbody>
</table>
Primitive Types in C

- **Integer types**
  - `char, int`

- **Floating point**
  - `float, double`

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

---

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

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
    ...
}

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

```c
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:**

```c
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!

```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?)

```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*

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

```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;
}
```
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)

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

C source file 2
(sumnum.c)

```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 later lecture

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

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

ld or gcc   sumnum

libraries (e.g. libc)
```
Polling Question

Discuss on Ed!
PollEnv survey will posted to Ed after lecture.

Which of the following statements is FALSE?

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 out later today, due 10 pm on Friday Monday
   Find exercise spec on website, submit via Gradescope
   • Course “CSE 333” under “Fall 2023”, Assignment “Exercise 1”, then drag-n-drop file(s)!
   Sample solution will be posted Saturday Tuesday afternoon
   Hint: look at documentation for stdlib.h, string.h, and inttypes.h
❖ Homework 0 is out later today