Syllabus

CS 3410: Computer System Organization and Programming

[K. Bala, A. Bracy, E. Sirer, Z. Susag, H. Weatherspoon, and K. Laeufer]

https://www.cs.cornell.edu/courses/cs3410/2025fa/

CS 3410 - Fall 2025 taught by Giulia Guidi and Kevin Laeufer

Lecture: MoWe 1:25pm-2:40pm ET, Uris Hall G01	
Prelim: October 9, 2025 7:30 PM ET	**Makeup Prelim**: October 16, 2025 5:30 PM ET
Final: TBD	**Makeup Final**: TBD

Week	Date	Lecture	Lab	Assignment
1	Mon 8/25	• Intro (Slides / Notes) • 1+1=2 (Slides / Notes)	Lab 1: Nice to C You	• A1: Printf **(Due: Wed 9/3)** • Introductory Survey **due Friday 8/29** • Week 1 Topic Mastery Quiz **due Saturday 8/30** • E0 - E4: Course Policies, SSH, Unix & Git, C Compilation, Makefiles **due Sunday 9/7**
	Wed 8/27	• Numbers (Slides TBD / Notes) • C Intro (Slides TBD / Notes)		

https://www.cs.cornell.edu/courses/cs3410/2025fa/

CS 3410 - Fall 2025 taught by Giulia Guidi and Kevin Laeufer

Lecture: MoWe 1:25pm-2:40pm ET, Uris Hall G01	
Prelim: October 9, 2025 7:30 PM ET	**Makeup Prelim**: October 16, 2025 5:30 PM ET
Final: TBD	**Makeup Final**: TBD

Week	Date	Lecture	Lab	Assignment
1	Mon 8/25	• Intro (Slides / Notes) • 1+1=2 (Slides / Notes)	Lab 1: Nice to C You	• A1: Printf **(Due: Wed 9/3)** • Introductory Survey **due Friday 8/29** • Week 1 Topic Mastery Quiz **due Saturday 8/30** • E0 - E4: Course Policies, SSH, Unix & Git, C Compilation, Makefiles **due Sunday 9/7**
	Wed 8/27	• Numbers (Slides TBD / Notes) • C Intro (Slides TBD / Notes)		

https://www.cs.cornell.edu/courses/cs3410/2025fa/

CS 3410 - Fall 2025 taught by Giulia Guidi and Kevin Laeufer

Lecture: MoWe 1:25pm-2:40pm ET, Uris Hall G01	
Prelim: October 9, 2025 7:30 PM ET	**Makeup Prelim**: October 16, 2025 5:30 PM ET
Final: TBD	**Makeup Final**: TBD

Week	Date	Lecture	Lab	Assignment
1	Mon 8/25	• Intro (Slides / Notes) • 1+1=2 (Slides / Notes)	Lab 1: Nice to C You	• A1: Printf **(Due: Wed 9/3)** • Introductory Survey **due Friday 8/29** • Week 1 Topic Mastery Quiz **due Saturday 8/30** • E0 - E4: Course Policies, SSH, Unix & Git, C Compilation, Makefiles **due Sunday 9/7**
	Wed 8/27	• Numbers (Slides TBD / Notes) • C Intro (Slides TBD / Notes)		

https://www.cs.cornell.edu/courses/cs3410/2025fa/

CS 3410 - Fall 2025 taught by Giulia Guidi and Kevin Laeufer

Lecture: MoWe 1:25pm-2:40pm ET, Uris Hall G01	
Prelim: October 9, 2025 7:30 PM ET	**Makeup Prelim**: October 16, 2025 5:30 PM ET
Final: TBD	**Makeup Final**: TBD

Week	Date	Lecture	Lab	Assignment
1	Mon 8/25	• Intro (Slides / Notes) • 1+1=2 (Slides / Notes)	Lab 1: Nice to C You	• A1: Printf **(Due: Wed 9/3)** • Introductory Survey **due Friday 8/29** • Week 1 Topic Mastery Quiz **due Saturday 8/30** • E0 - E4: Course Policies, SSH, Unix & Git, C Compilation, Makefiles **due Sunday 9/7**
	Wed 8/27	• Numbers (Slides TBD / Notes) • C Intro (Slides TBD / Notes)		

https://www.cs.cornell.edu/courses/cs3410/2025fa/

CS 3410 - Fall 2025 taught by Giulia Guidi and Kevin Laeufer

Lecture: MoWe 1:25pm-2:40pm ET, Uris Hall G01	
Prelim: October 9, 2025 7:30 PM ET	**Makeup Prelim**: October 16, 2025 5:30 PM ET
Final: TBD	**Makeup Final**: TBD

Week	Date	Lecture	Lab	Assignment
1	Mon 8/25	• Intro (Slides / Notes) • 1+1=2 (Slides / Notes)	Lab 1: Nice to C You	• A1: Printf **(Due: Wed 9/3)** • Introductory Survey **due Friday 8/29** • Week 1 Topic Mastery Quiz **due Saturday 8/30** • E0 - E4: Course Policies, SSH, Unix & Git, C Compilation, Makefiles **due Sunday 9/7**
	Wed 8/27	• Numbers (Slides TBD / Notes) • C Intro (Slides TBD / Notes)		

https://www.cs.cornell.edu/courses/cs3410/2025fa/course/syllabus.html

TL;DR

- Course communication will happen on Ed.
  - Log in with your `netid@cornell.edu` email address. You should already have access.
  - You're responsible for knowing everything that we post as announcements there. Ignore announcements at your own risk.
- Homework hand-in and grading happens on Gradescope.
- There are 11 assignments.
  - The deadline is usually Wednesday night at 11:59pm. See the schedule for details.
  - You have 12 total "slip days" you can use throughout the semester. You may use up to 3 for a given assignment.
  - Your lowest homework score will be dropped.
- There is one prelim and a final exam. For the prelim, there is only one make-up exam the following week with partial weight transfer (read the syllabus carefully, no exceptions).

https://www.cs.cornell.edu/courses/cs3410/2025fa/course/syllabus.html

On **assignments**, everything you turn in must be 100% completely your own work. You may discuss the work in generalities with other students using natural language, but *you may not show anyone else your code or look at anyone else's code*. Specifically:

- Do not show any (partial or complete) solution to another student.
- Do not look at any (partial or complete) solution written by another student.
- Do not post solutions on Ed, except in private threads with course staff.
- *Do* ask someone if you're confused about what the assignment is asking for.
- *Definitely* ask the course staff if you're not sure whether or not something is OK.

https://www.cs.cornell.edu/courses/cs3410/2025fa/course/syllabus.html

Generative AI

You can use **Copilot AI through Cornell** to support your learning in CS3410. However, the use of these tools must follow strict guidelines to ensure academic integrity and independent understanding.

The work you submit must be 100% handwritten by you. This means that every line of code, every explanation, and every answer must be written by you, and based on your own understanding. This also applies to Stack Overflow, GitHub, or Google. You may **not** copy and paste code, assignment instructions, or links to assignment materials into GenAI tools (e.g., GitHub repositories, CMS pages, shared documents). You may **not** copy code, instructions, or links generated by GenAI tools into your assignment submission.

You must disclose the use of GenAI. Each assignment includes a GenAI Usage Quiz. You must complete this quiz honestly to report how you have used the GenAI tools.

Homework

- Read the syllabus.
- Introductory survey on Gradescope (**due on Friday**)
- Set up infrastructure **before lab 1**
- Assignment 1 will be released **tomorrow**

Two's Complement Addition and Overflow

CS 3410: Computer System Organization and Programming

[K. Bala, A. Bracy, E. Sirer, Z. Susag, H. Weatherspoon, and K. Laeufer]

Two's Complement

Non-negatives unchanged:

- +0 = 0000
- +1 = 0001
- +2 = 0010
- +3 = 0011
- +4 = 0100
- +5 = 0101
- +6 = 0110
- +7 = 0111
- +8 = 1000

How do you express this magnitude in the negative?

	flip	Negatives then add 1
$\overline{0}$	1111	-0 = 0000
$\overline{1}$	1110	-1 = 1111
$\overline{2}$	1101	-2 = 1110
$\overline{3}$	1100	-3 = 1101
$\overline{4}$	1011	-4 = 1100
$\overline{5}$	1010	-5 = 1011
$\overline{6}$	1001	-6 = 1010
$\overline{7}$	1000	-7 = 1001
$\overline{8}$	0111	-8 = 1000

(8 no longer expressible in 4 bits when moving to signed)

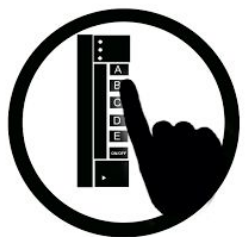
PollEV Question #1:

What is the value (expressed in decimal) of the 2s complement number

`11010`

A. 26
B. 6
C. -6
D. -10
E. -26

PollEv.com/cs3410

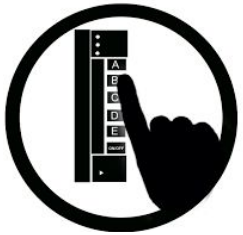


PollEV Question #2:

Suppose I want to express the 2s complement number `1010` in 5 bits instead of 4 bits. What number should I use?

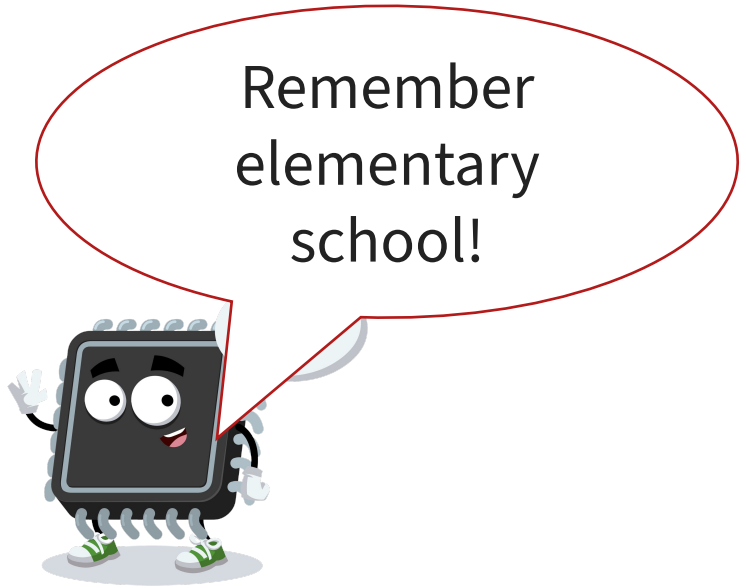
A. `01010`
B. `11010`
C. `10101`
D. `10100`
E. Sorry, it is not possible.

PollEv.com/cs3410



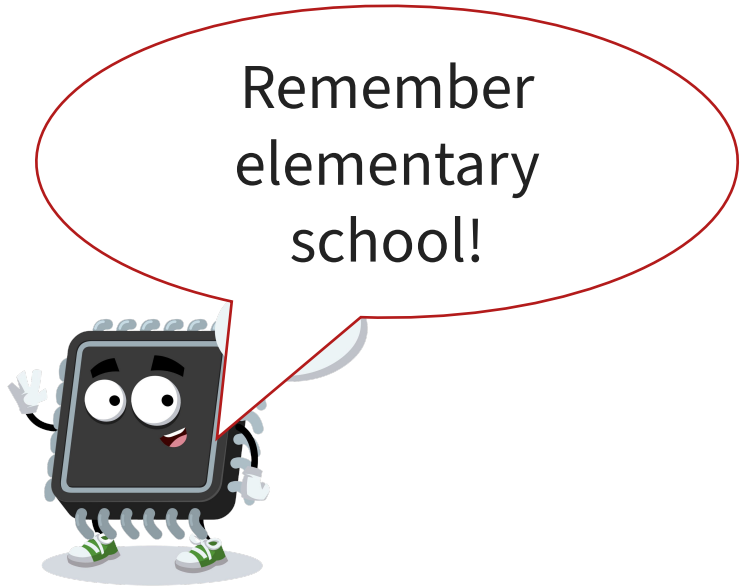
Adding Binary Numbers

$$\begin{array}{r} 24_{10} \\ +\ 17_{10} \\ \quad +1 \\ \hline 41_{10} \end{array} \qquad \begin{array}{r} 11000_{2} \\ +\ 10001_{2} \\ \quad +1 \\ \hline 101001_{2} \end{array}$$



Adding Negative Binary Numbers

$$
\begin{array}{r}
-\ 24_{10} \\
+\ 17_{10} \\
\hline
-\ 7_{10}
\end{array}
\qquad\qquad
\begin{array}{r}
+\ 010001_{2} \\
\hline
\end{array}
$$



Adding Negative Binary Numbers

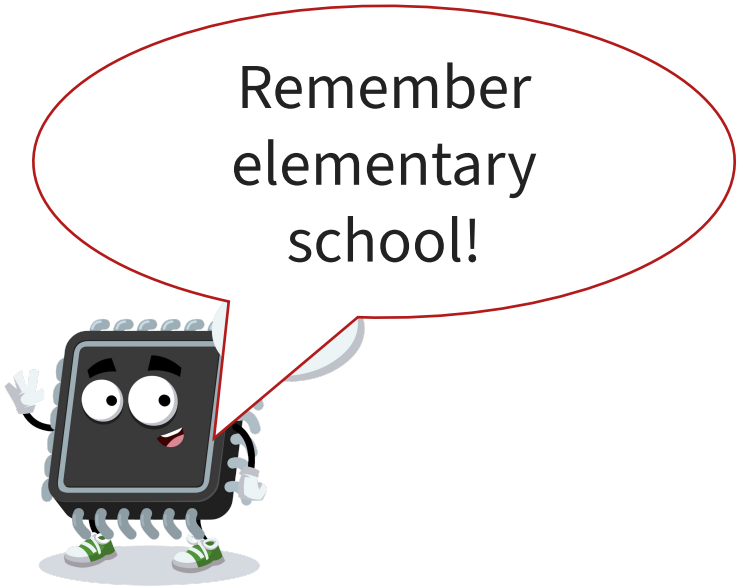
$$011000_2$$

$$-\ 24_{10}$$

$$+\ 17_{10}$$

$$-\ 7_{10}$$

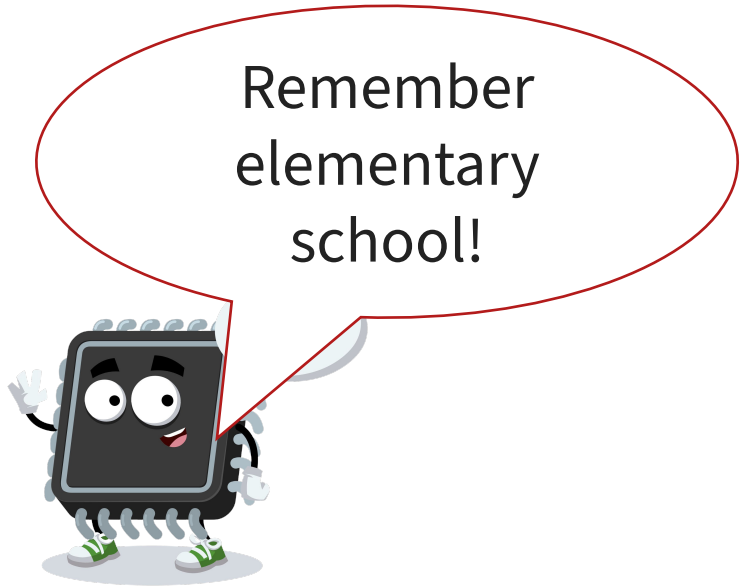
$$+\ 010001_2$$



Adding Negative Binary Numbers

$$\sim 011000_2 + 1 =$$

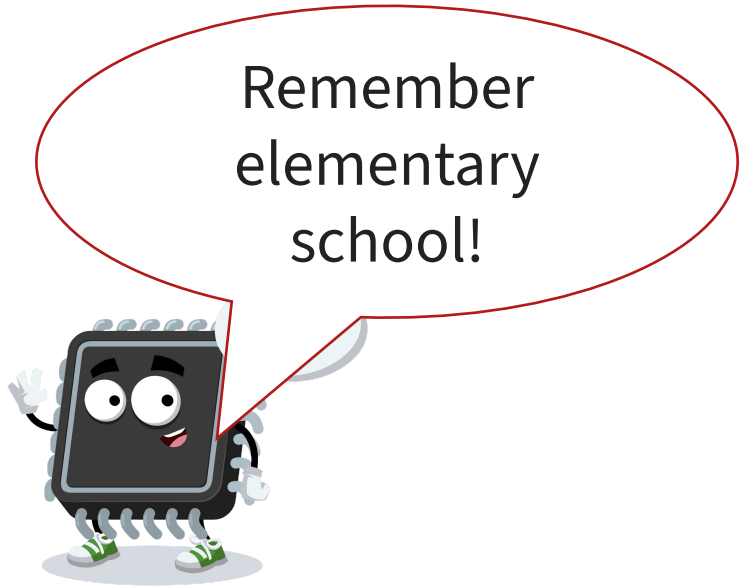
$$\begin{array}{rr} -\ 24_{10} & 101000_2 \\ +\ 17_{10} & +\ 010001_2 \\ \hline -\ 7_{10} & \end{array}$$



Adding Negative Binary Numbers

$$\sim 011000_2 + 1 =$$

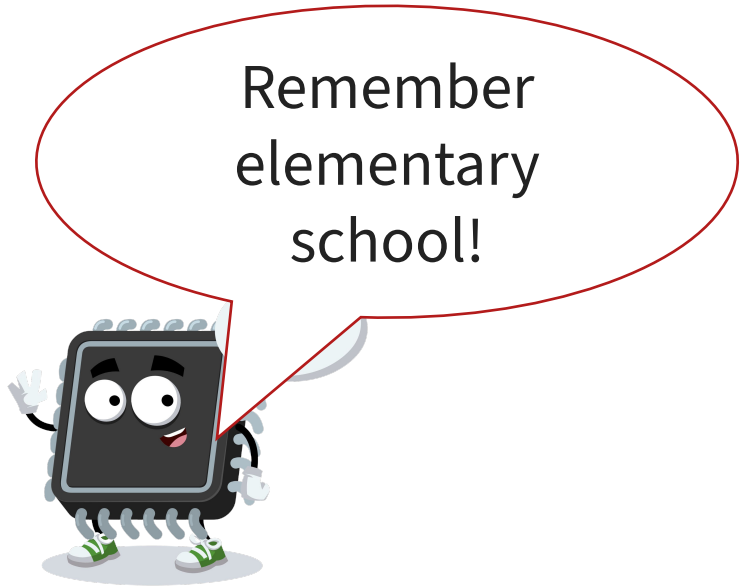
$-\ 24_{10}$	101000_2
$+\ 17_{10}$	$+\ 010001_2$
$-\ 7_{10}$	111001_2



Adding Negative Binary Numbers

$$\sim 011000_2 + 1 =$$

$-\ 24_{10}$	101000_2
$+\ 17_{10}$	$+\ 010001_2$
-7_{10}	111001_2

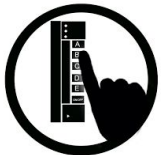


$$\sim 111001_2 + 1 = 111_2 = 7_{10}$$

Two's Complement Addition

Which of the following has problems (assuming a four-bit number)?

a) 7 + 1
b) -7 + -3
c) -7 + -1
d) Only A & B have problems
e) They all have problems.



PollEv.com/cs3410

-1 =	1111	= 15
-2 =	1110	= 14
-3 =	1101	= 13
-4 =	1100	= 12
-5 =	1011	= 11
-6 =	1010	= 10
-7 =	1001	= 9
-8 =	1000	= 8
+7 =	0111	= 7
+6 =	0110	= 6
+5 =	0101	= 5
+4 =	0100	= 4
+3 =	0011	= 3
+2 =	0010	= 2
+1 =	0001	= 1
0 =	0000	= 0

When can **overflow** occur?

- adding a negative and a positive?
  - Overflow *cannot occur*
- adding two positives?
  - Overflow *can occur*
- adding two negatives?
  - Overflow *can occur*
  -

YouTube
Shared publicly - Dec 1, 2014

We never thought a video would be watched in numbers greater than a 32-bit integer (=2,147,483,647 views), but that was before we met PSY. "Gangnam Style" has been viewed so many times we had to upgrade to a 64-bit integer (9,223,372,036,854,775,808)!

Hover over the counter in PSY's video to see a little math magic and stay tuned for bigger and bigger numbers on YouTube.

Programming in C

CS 3410: Computer System Organization and Programming

[K. Bala, A. Bracy, E. Sirer, Z. Susag, H. Weatherspoon, D. Garcia, and K. Laeufer]

Goals for today

- Minimal C basics
  - Minimal C program highlighting C basics
- How to print!
  - Print numbers
  - Print negative numbers
  - Overflow
- More C basics
  - Prototypes, Headers, Libraries
  - Compiling and Linking

Why C?

"C is a horrible, horrible programming language."

- David Gries (Cornell)

Why C?

"C is a horrible, horrible programming language."

- David Gries (Cornell)

<expletives deleted>

- Michael Clarkson (Cornell)

Why C?

"C is a horrible, horrible programming language."
- David Gries (Cornell)

<expletives deleted>
- Michael Clarkson (Cornell)

"C had a good run."
– Andrew Myers (Cornell)

Why C?

"C is a horrible, horrible programming language."
- David Gries (Cornell)

<expletives deleted>
- Michael Clarkson (Cornell)

"C had a good run."
– Andrew Myers (Cornell)

"C is quirky, flawed, and an enormous success."
- Dennis Ritchie (creator of C)

Why C?

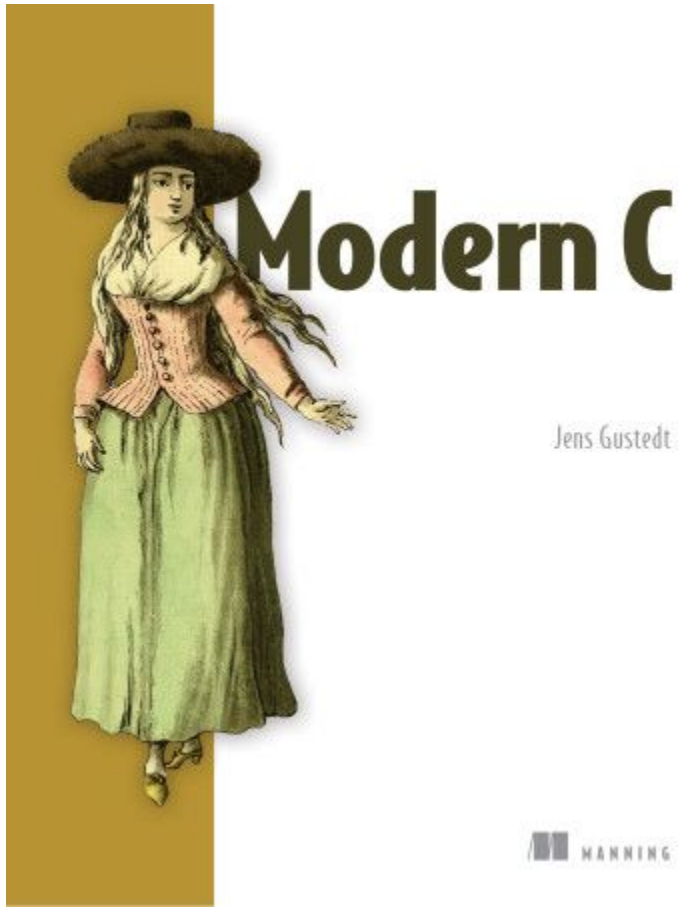
- co-evolved with the Unix operating system
- forces you to manage memory on your own

Disclaimers

- Our lectures only teach key concepts.

Disclaimers

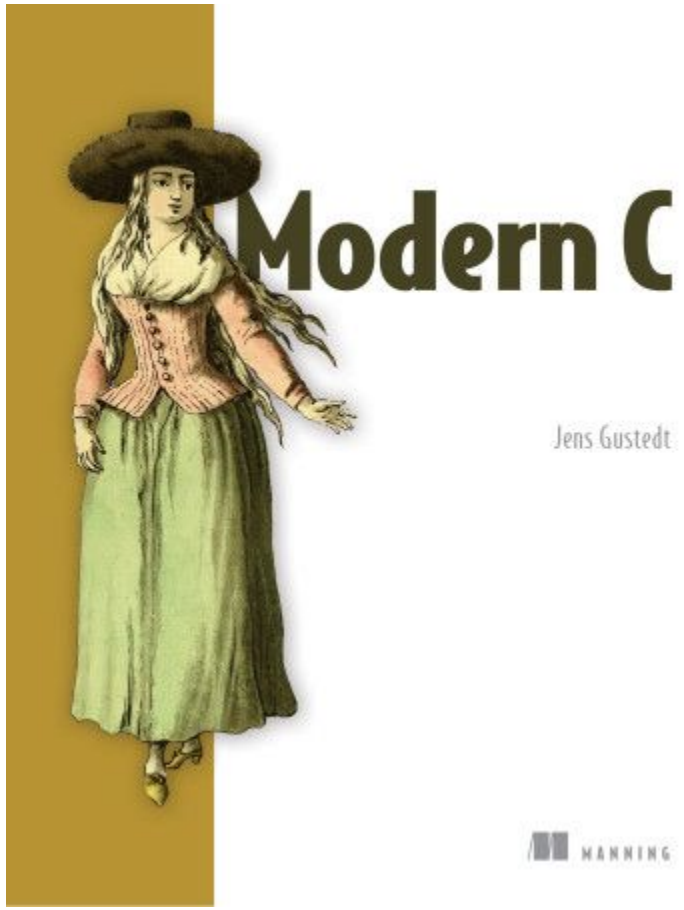
- Our lectures only teach key concepts.
- Most books written about C are wrong.



One of the good C book.

Disclaimers

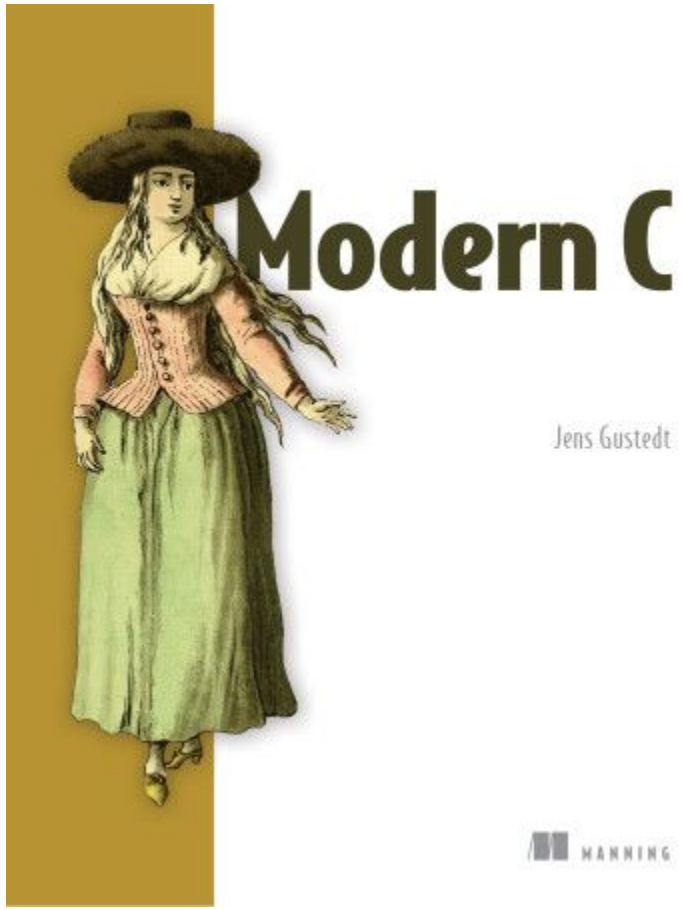
- Our lectures only teach key concepts.
- Most books written about C are wrong.
- C is useful for learning low-level programming



One of the good C book.

Disclaimers

- Our lectures only teach key concepts.
- Most books written about C are wrong.
- C is useful for learning low-level programming
- Do not start any new project in C; use Rust or Go.



One of the good C book.

Hello CS3410 in C

```c
#include <stdio.h>                      // Import library using #include

int main(int argc, char *argv[]) {      // Main Function
    printf("Hello CS3410!\n");          // Print

    return 0;                           // main returns an integer
}
```

Hello CS3410 in C

```c
#include <stdio.h>                  // Import library using #include

int main() {                        // Main Function
    printf("Hello CS3410!\n");      // Print

    return 0;                       // main returns an integer
}
```

How can we run our program?

Compile

```
$ gcc hello.c
```

How can we run our program?

Compile

```
$ gcc hello.c
```

Execute

```
$ ./a.out
```

How can we run our program?

Compile

```
$ gcc hello.c
```

Execute

```
$ ./a.out
  Hello CS3410!
```

How can we run our program with the class infrastructure?

Compile

```
$ rv gcc hello.c
```

Execute

```
$ rv qemu ./a.out
  Hello CS3410!
```

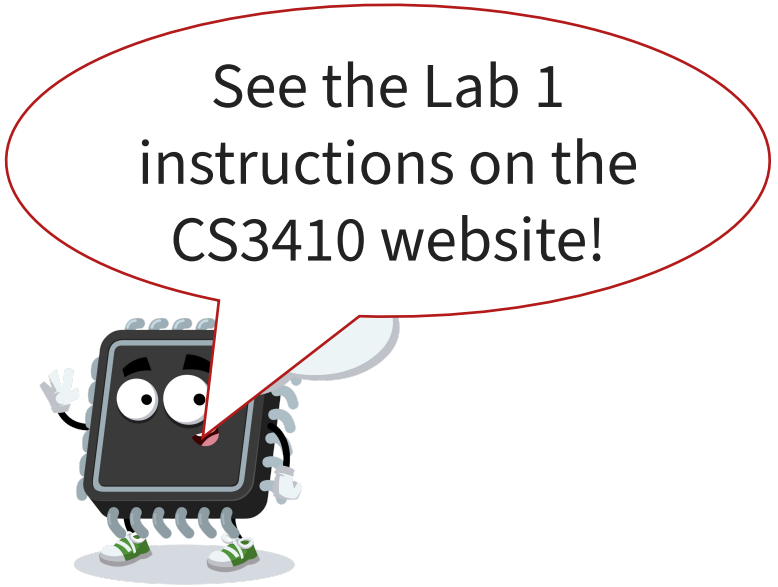
How can we run our program with the class infrastructure?

Compile

```
$ rv gcc hello.c
```

Execute

```
$ rv qemu ./a.out
  Hello CS3410!
```



C vs. Java — Philosophies

	C	Java
Type of Language	Procedural	Object Oriented
Programming Unit	Function	Class
Compilation	`gcc hello.c`	`javac Hello.java`
Execution	`./a.out`	`java Hello`
Storage	Manual (More on this next lecture!)	Automatic

C vs. Java — Syntax

	C	Java
Libraries	`#include <stdio.h>`	`import java.io.File`
Comments	`/* block comment */` or `// line comment`	
Variables	`int x = 5;`	
Operators	Arithmetic, Assignment: `+, -, *, /, %, =, +=, -=, …` Comparison: `==, !=, <=, >=, <, >` Increment, Decrement: `++, --` Bitwise: `<<, >>, ~, &, |, ^`	
Constants	`#define` or `const`	`final`
Naming Conventions	`snake_case`	`camelCase`

Functions

```
int main(int argc, char* argv[]) {
    printf("Hello World!\n");
    return 0;
}
void foo() { printf(“foo\n”); }
```

Functions


```
int main(int argc, char* argv[]) {
    printf("Hello World!\n");
    return 0;
}
void foo() { printf(“foo\n”); }
```

Functions


```
int main(int argc, char* argv[]) {
    printf("Hello World!\n");
    return 0;
}
void foo() { printf(“foo\n”); }
```

Functions


```
int main(int argc, char* argv[]) {
    printf("Hello World!\n");
    return 0;
}
void foo() { printf(“foo\n”); }
```

Control Flow

```
while (expression) {
   statements;
}
```

Control Flow

```
while (expression) {
    statements;
}
```

```
for (init-statement; condition; inc-expression) {
    statements;
}
```

Control Flow

```
while (expression) {
   statements;
}
```

```
for (init-statement; condition; inc-expression) {
   statements;
}
```

```
if (condition) {
   statements;
} else if (condition) {
   statements;
} else {
    statements;
}
```

Control Flow and Curly Braces

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0) {
  goto fail;
}
```

Control Flow and Curly Braces

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
  goto fail;
```

Control Flow and Curly Braces

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
  goto fail;
  goto fail;
```

Control Flow and Curly Braces

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0) {
  goto fail;
  goto fail;
}
```

Control Flow and Curly Braces

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0) {
  goto fail;
}
goto fail;
```

Control Flow and Curly Braces

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0) {
  goto fail;
}
goto fail;
```

Gets executed unconditionally

Control Flow and Curly Braces

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0) {
  goto fail;
}
goto fail;
```

Gets executed unconditionally

Security Vulnerability in Apple's SSL/TLS Implementation: CVE-2014-1266

What is argv?

```
#include <stdio.h>
int main(int argc, char* argv[]) {


    return 0;
}
```

What is argv?

```
#include <stdio.h>
int main(int argc, char* argv[]) {


    return 0;
}
```

What is argv?

```
#include <stdio.h>
int main(int argc, char* argv[]) {


    return 0;
}
```

an array of strings

Similar to `String[]` in Java

What is argv?

```
#include <stdio.h>
int main(int argc, char* argv[]) {

    return 0;
}
```

Similar to `String[]` in Java

However, no `argv.length`.

What is argv?

```c
#include <stdio.h>
int main(int argc, char* argv[]) {

    return 0;
}
```

an array of strings

Similar to `String[]` in Java

However, no `argv.length`.

Length is passed separately.

What is argv?

```
#include <stdio.h>
int main(int argc, char* argv[]) {

      printf("%s\n", argv[0]);

    return 0;
}
```

What is argv?

```c
#include <stdio.h>
int main(int argc, char* argv[]) {

    printf("%s\n", argv[0]);

    return 0;
}
```

Compile

```
$ rv gcc argv1.c
```

Execute

```
$ rv qemu ./a.out
```

What is argv?

```c
#include <stdio.h>
int main(int argc, char* argv[]) {

    printf("%s\n", argv[0]);

    return 0;
}
```

Compile

```
$ rv gcc argv1.c
```

Execute

```
$ rv qemu ./a.out
```

```
./a.out
```

What is argv?

```c
#include <stdio.h>
int main(int argc, char* argv[]) {

    printf("%s\n", argv[1]);

    return 0;
}
```

Compile

```
$ rv gcc argv2.c
```

Execute

```
$ rv qemu ./a.out
```

What is argv?

```c
#include <stdio.h>
int main(int argc, char* argv[]) {

    printf("%s\n", argv[1]);

    return 0;
}
```

Compile

```
$ rv gcc argv2.c
```

Execute

```
$ rv qemu ./a.out
```

```
Job 1, './a.out' terminated by signal
SIGSEGV (Address boundary error)
```

What is argv?

```c
#include <stdio.h>
int main(int argc, char* argv[]) {

    printf("%s\n", argv[1]);

    return 0;
}
```

Compile

```
$ rv gcc argv2.c
```

Execute

```
$ rv qemu ./a.out CS3410
CS3410
```

What is argv?

```c
#include <stdio.h>
int main(int argc, char* argv[]) {

    printf("%s\n", argv[1]);

    return 0;
}
```

Compile

```
$ rv gcc argv2.c
```

Execute

```
$ rv qemu ./a.out
```

```
Job 1, './a.out' terminated by signal
SIGSEGV (Address boundary error)
```

What is argv?

```c
#include <stdio.h>
int main(int argc, char* argv[]) {

    printf("%s\n", argv[1]);

    return 0;
}
```

Compile

```
$ rv gcc -fsanitize=address argv2.c
```

Execute

```
$ rv qemu ./a.out
```

```
SEGV on unknown address 0x000000000000
/home/kevin/d/cs3410/lec2/argv2.c:4 in main
```

What is argv?

```
#include <stdio.h>
int main(int argc, char* argv[]) {

      printf("%s\n", argv[1]);

    return 0;
}
```

Can you write a program that prints all arguments?
You will need to use `argv` and `argc`!

What is argv?

```
#include <stdio.h>
int main(int argc, char* argv[]) {
  for (          ;          ;       ) {
     printf("%s\n", argv[  ]);
  }
    return 0;
}
```

Can you write a program that prints all arguments?
You will need to use `argv` and `argc`!

What is argv?

```
#include <stdio.h>
int main(int argc, char* argv[]) {
  for (int i = 0;          ;       ) {
     printf("%s\n", argv[  ]);
  }
    return 0;
}
```

Can you write a program that prints all arguments?
You will need to use `argv` and `argc`!

What is argv?

```
#include <stdio.h>
int main(int argc, char* argv[]) {
  for (int i = 0;          ; i += 1) {
     printf("%s\n", argv[  ]);
  }
    return 0;
}
```

Can you write a program that prints all arguments?
You will need to use `argv` and `argc`!

What is argv?

```
#include <stdio.h>
int main(int argc, char* argv[]) {
  for (int i = 0; i < argc; i += 1) {
     printf("%s\n", argv[  ]);
  }
    return 0;
}
```

Can you write a program that prints all arguments?
You will need to use `argv` and `argc`!

What is argv?

```
#include <stdio.h>
int main(int argc, char* argv[]) {
  for (int i = 0; i < argc; i += 1) {
     printf("%s\n", argv[i]);
  }
    return 0;
}
```

Can you write a program that prints all arguments?
You will need to use `argv` and `argc`!

What is argv?

```c
#include <stdio.h>
int main(int argc, char* argv[]) {
  for (int i = 0; i < argc; i += 1) {
    printf("%s\n", argv[i]);
  }
  return 0;
}
```

Compile

```
$ rv gcc -fsanitize=address argv3.c
```

Execute

```
$ rv qemu ./a.out
```

./a.out

What is argv?

```c
#include <stdio.h>
int main(int argc, char* argv[]) {
  for (int i = 0; i < argc; i += 1) {
    printf("%s\n", argv[i]);
  }
  return 0;
}
```

Compile

```
$ rv gcc -fsanitize=address argv3.c
```

Execute

```
$ rv qemu ./a.out 1 2
```

What is argv?

```c
#include <stdio.h>
int main(int argc, char* argv[]) {
  for (int i = 0; i < argc; i += 1) {
     printf("%s\n", argv[i]);
  }
    return 0;
}
```

Compile

```
$ rv gcc -fsanitize=address argv3.c
```

Execute

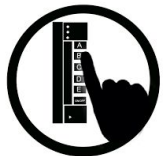
```
$ rv qemu ./a.out 1 2
  ./a.out
  1
  2
```

Variable Initialization

```c
#include <stdio.h>                      // Import library using #include

int main(int argc, char *argv[]) {      // Main Function
    int x;                              // Define x
    printf("%d\n", x);                  // Print x
    return 0;                           // main returns an integer
}
```

What gets printed?



A: Nothing: The compiler catches the uninitialized access and errors
B: Nothing: When running the code, an error occurs
C: 0
D: A random number that changes depending on how you run the code
E: It is impossible to predict; anything could happen.

PollEv.com/cs3410

Variable Initialization

```c
#include <stdio.h>                      // Import library using #include

int main(int argc, char *argv[]) {      // Main Function
    int x;                              // Define x
    printf("%d\n", x);                  // Print x
    return 0;                           // main returns an integer
}
```

Accessing uninitialized variables triggers **undefined behavior**.

- C compiler is allowed to assume that your program is correct
- If you are using C *incorrectly*, all bets are off.

Variable Initialization

```c
#include <stdio.h>                      // Import library using #include

int main(int argc, char *argv[]) {      // Main Function
    int x;                              // Define x
    printf("%d\n", x);                  // Print x
    return 0;                           // main returns an integer
}
```

Compile

```
$ rv gcc -Wall -Wextra -Wpedantic -Wshadow -Wformat=2 -std=c23 -fsanitize=address,undefined uninit.c
```

Variable Initialization

```c
#include <stdio.h>                      // Import library using #include

int main(int argc, char *argv[]) {      // Main Function
    int x;                              // Define x
    printf("%d\n", x);                  // Print x
    return 0;                           // main returns an integer
}
```

Compile

```
$ rv gcc -Wall -Wextra -Wpedantic -Wshadow -Wformat=2 -std=c23 -fsanitize=address,undefined uninit.c

uninit.c:5:5: warning: 'x' is used uninitialized [-Wuninitialized]
    5 |     printf("%d\n", x);             // Print x
      |     ^~~~~~~~~~~~~~~~
uninit.c:4:9: note: 'x' was declared here
    4 |     int x;                         // Define x
```

C Types

Type	Description	Example	Library
`int`	Signed integers	`1, -2, 0xFF`	C
`unsigned int`	Unsigned integers	`1, 9, 15`	C
`float`	Floating point decimal	`0.0, 3.14`	C
`double`	Higher precision floating point	`3.1415926`	C
`char`	Character	`'a'`	C
`long`	Longer integer	`1234567898765`	C
`long long`	Even longer integer	`1234...6061`	C
`int32_t`	32-bit signed integer	`-61`	stdint.h
`uint32_t`	32-bit unsigned integer	`61`	stdint.h
`bool`	boolean value	`true, false`	C23

C Types: How big are they, really?

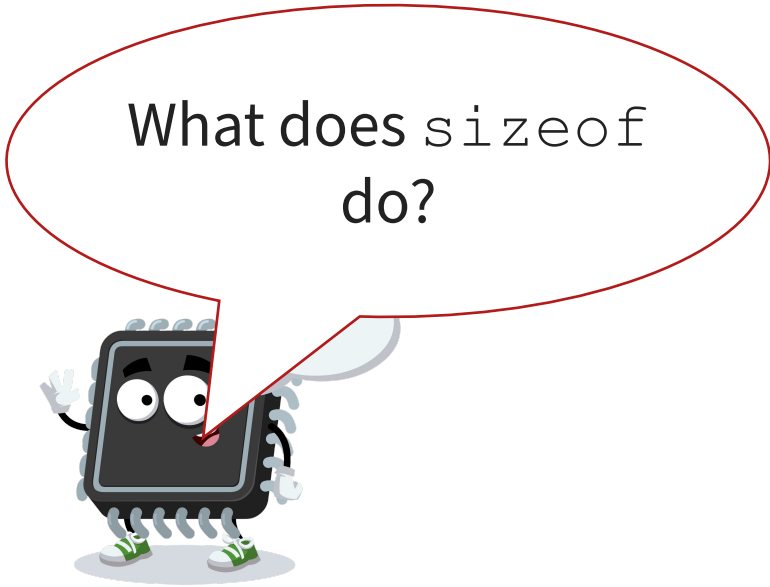
- Note: the number of bytes in an `int` depends on the computer.
- The C standard guarantees the following:

```
sizeof(int) <= sizeof(long) <= sizeof(long long)
```

C Types: How big are they, really?

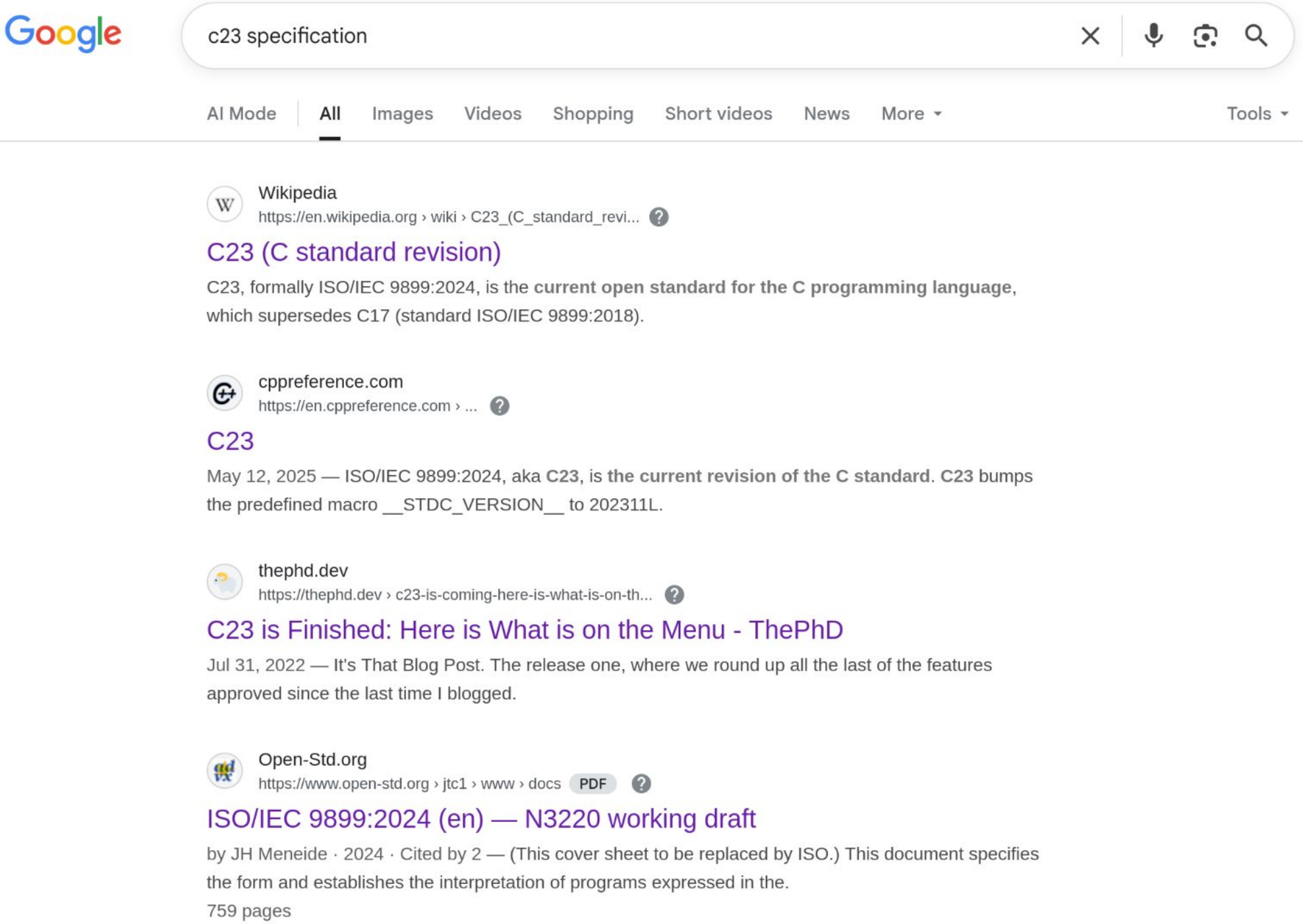
- Note: the number of bytes in an `int` depends on the computer.
- The C standard guarantees the following:

```
sizeof(int) <= sizeof(long) <= sizeof(long long)
```



C Typ

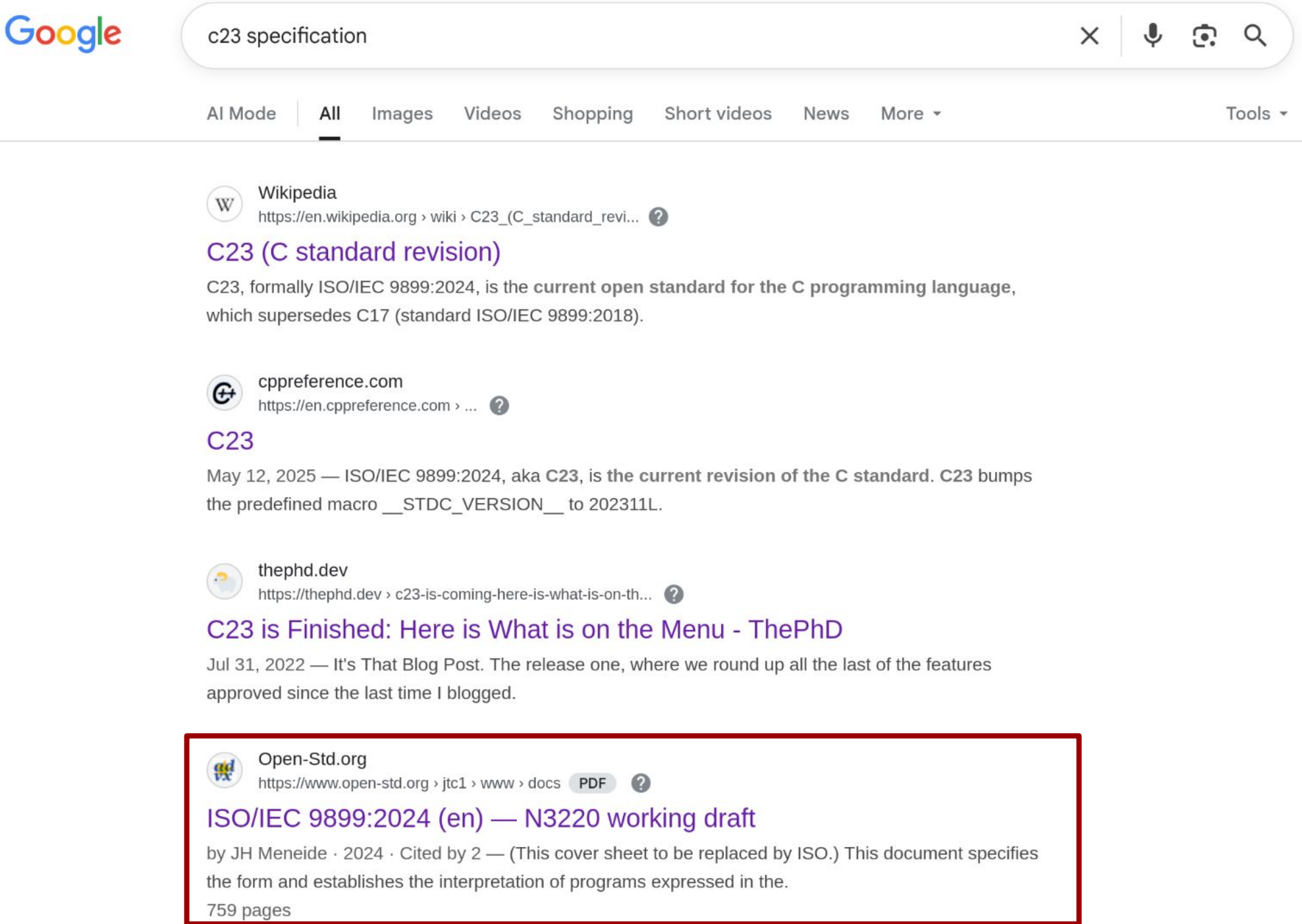
- Nc
- Th



zeof

C Typ

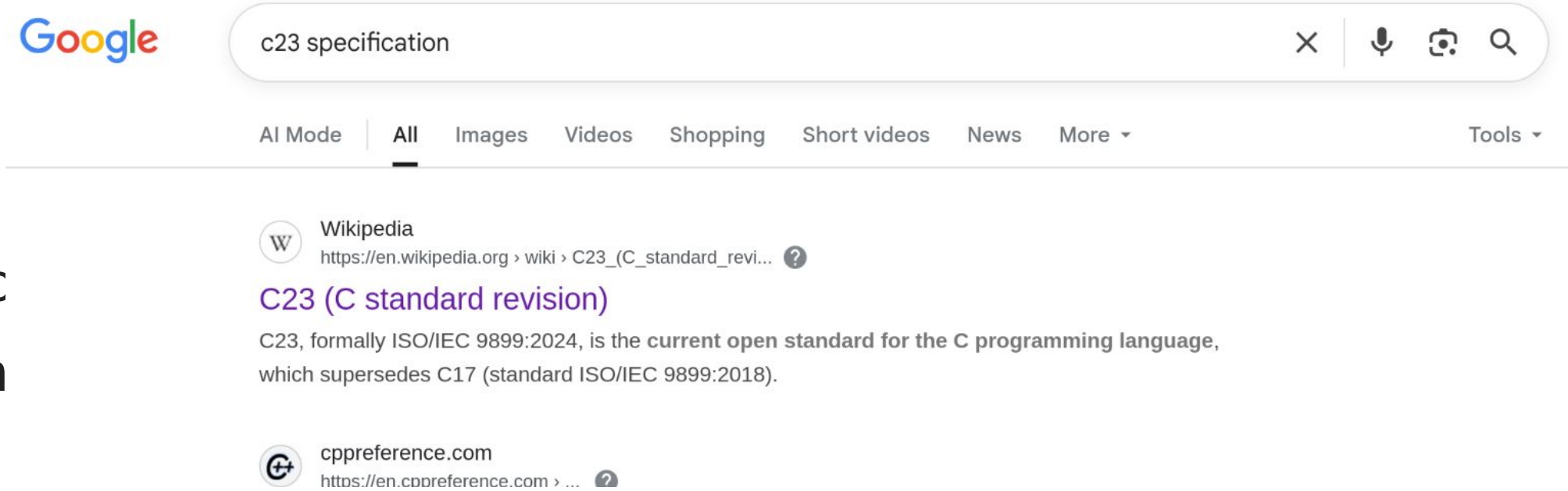
- Nc
- Th



zeof

C Typ

- Nc
- Th



Semantics

2 The `sizeof` operator yields the size (in bytes) of its operand, which may be an expression or the parenthesized name of a type. The size is determined from the type of the operand. The result is an integer. If the type of the operand is a variable length array type, the operand is evaluated; otherwise, the operand is not evaluated and the result is an integer constant.

Jul 31, 2022 — It's That Blog Post. The release one, where we round up all the last of the features approved since the last time I blogged.

Open-Std.org
https://www.open-std.org › jtc1 › www › docs PDF
ISO/IEC 9899:2024 (en) — N3220 working draft
by JH Meneide · 2024 · Cited by 2 — (This cover sheet to be replaced by ISO.) This document specifies the form and establishes the interpretation of programs expressed in the.
759 pages

C Types: How big are they, really?

- Note: the number of bytes in an `int` depends on the computer.
- The C standard guarantees the following:

```
sizeof(int) <= sizeof(long) <= sizeof(long long)
```

- We recommend that you use the `intN_t` and `uintN_t` types!

Adding your own Types: `typedef` and `struct`

- `typedef` allows you to define new names for existing types:
  - `typedef uint8_t BYTE;`
- `struct` allows you to define a structured group of variables

```
typedef enum cardsuit{DIAMONDS, SPADES, HEARTS, CLUBS} suit_t;
typedef struct cardstruct {
    int rank;
    suit_t suit;
} card_t;

card_t card;
card.rank = 1;
card.suit = SPADES;
```

printf

```c
#include <stdio.h>

int main() {
   int n = 3410;
   printf("Decimal: %d\n", n);
   printf("Binary: %b\n", n);
   printf("Hexadecimal: %x\n", n);

   return 0;
}
```

printf

```c
#include <stdio.h>
```

Needed to use printf

```c
int main() {
   int n = 3410;
   printf("Decimal: %d\n", n);
   printf("Binary: %b\n", n);
   printf("Hexadecimal: %x\n", n);

   return 0;
}
```

printf

```c
#include <stdio.h>

int main() {
   int n = 3410;
   printf("Decimal: %d\n", n);
   printf("Binary: %b\n", n);
   printf("Hexadecimal: %x\n", n);

   return 0;
}
```

Needed to use printf

Format specifiers

printf

```c
#include <stdio.h>

int main() {
    long int n = 3410;
    printf("Decimal: %ld\n", n);
    printf("Binary: %lb\n", n);
    printf("Hexadecimal: %lx\n", n);

    return 0;
}
```

Needed to use printf

Format specifiers

printf

```c
#include <stdio.h>

int main() {
    long int n = 3410;
    printf("Decimal: %ld\n", n);
    printf("Binary: %lb\n", n);
    printf("Hexadecimal: %lx\n", n);

    return 0;
}
```

Needed to use printf

Format specifiers

`l` is a length modifier

printf

```c
#include <stdio.h>

int main() {
    long int n = 3410;
    printf("Decimal: %ld\n", n);
    printf("Binary: %lb\n", n);
    printf("Hexadecimal: %lx\n", n);

    return 0;
}
```

Needed to use printf

Format specifiers

`l` is a length modifier

`d`, `b`, and `x` are conversion specifiers

printf

```c
#include <stdio.h>
#include <stdint.h>
int main() {
    int16_t n = 3410;
    printf("Decimal: %w16d\n", n);
    printf("Binary: %w16b\n", n);
    printf("Hexadecimal: %w16x\n", n);

    return 0;
}
```

Needed to use printf

Format specifiers

`l` is a length modifier

printf

```c
#include <stdio.h>
#include <stdint.h>
int main() {
    int16_t n = 3410;
    printf("Decimal: %w16d\n", n);
    printf("Binary: %w16b\n", n);
    printf("Hexadecimal: %w16x\n", n);

    return 0;
}
```

Needed to use printf

Format specifiers

`l` is a length modifier for `long int`

`w16` is a length modifier for `int16_t`

7.23.6 Formatted input/output functions

1 The formatted input/output functions shall behave as if there is a sequence point after the actions associated with each specifier.[320)]

7.23.6.1 The `fprintf` function

Synopsis

1
```c
#include <stdio.h>
int fprintf(FILE * restrict stream, const char * restrict format, ...);
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}

int main() {
 greet("3410");
}
```

Compile

```
$ rv gcc lib1.c
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}

int main() {
 greet("3410");
}
```

Compile

```
$ rv gcc lib1.c
```

Execute

```
$ rv qemu ./a.out
   Hello, 3410!
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Compile

```
$ rv gcc lib2.c
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Compile

```
$ rv gcc lib2.c
```

```
lib2.c:3:2: error: implicit declaration of
function ‘greet’
   3 | greet("3410");
     | ^~~~~
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Compile

```
$ rv gcc lib2.c
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
void greet(const char* name);
int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Compile

```
$ rv gcc lib3.c
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
void greet(const char* name);
int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Compile

```
$ rv gcc lib3.c
```

Execute

```
$ rv qemu ./a.out
   Hello, 3410!
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
void greet(const char* name);
int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Exact name does not matter!

Compile

```
$ rv gcc lib3.c
```

Execute

```
$ rv qemu ./a.out
   Hello, 3410!
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
void greet(const char* name);
int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Function declaration.

Exact name does not matter!

Function definition.

Header file: greet.h

```c
void greet(const char* name);
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
#include "greet.h"


int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Separating files: greet.c

```c
#include <stdio.h>
#include "greet.h"
```

```c
void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Separating files: main.c

```c
#include <stdio.h>
#include "greet.h"


int main() {
 greet("3410");
}
```

Compile and link

Compile **.c** files together

Compile

```
$ rv gcc main.c greet.c
```

Execute

```
$ rv qemu ./a.out
```

Compile and link

Or,

Compile **.c** files ***separately***, then ***link*** them together

Compile [on 3410 infrastructure]

```
$ rv gcc –c main.c -o main.o
$ rv gcc –c greet.c -o greet.o
$ rv gcc main.o greet.o
```

Execute [on 3410 infrastructure]

```
$ rv qemu main
```

https://www.cs.cornell.edu/courses/cs3410/2025fa/

CS 3410 - Fall 2025 taught by Giulia Guidi and Kevin Laeufer

Lecture: MoWe 1:25pm-2:40pm ET, Uris Hall G01	
Prelim: October 9, 2025 7:30 PM ET	**Makeup Prelim**: October 16, 2025 5:30 PM ET
Final: TBD	**Makeup Final**: TBD

Week	Date	Lecture	Lab	Assignment
1	Mon 8/25	• Intro (Slides / Notes) • 1+1=2 (Slides / Notes)	Lab 1: Nice to C You	• A1: Printf **(Due: Wed 9/3)** • Introductory Survey **due Friday 8/29** • Week 1 Topic Mastery Quiz **due Saturday 8/30** • E0 - E4: Course Policies, SSH, Unix & Git, C Compilation, Makefiles **due Sunday 9/7**
	Wed 8/27	• Numbers (Slides TBD / Notes) • C Intro (Slides TBD / Notes)		