CS 240
Computer Systems

#1: Introduction and C Programming
CS 240, Spring 2021 - Week 1
Wade Fagen-Ulmschneider

Welcome to CS 240: Introduction to Computer Systems

Course Website: https://courses.engr.illinois.edu/cs240/

Description: Basics of computer systems. Number representations, assembly/machine language, abstract models of processors (fetch/execute, memory hierarchy), processes/process control, simple memory management, file I/O and directories, network programming, usage of cloud services. 3 credit hours.

Instructors:

Prof. Wade Fagen-Ulmschneider <waf@>, Course Instructor
Teaching Associate Professor of Computer Science, Grainger College of Engineering

Natalia Ozymko <nozymko2@>, Lead Course Assistant
Computer Science, Grainger College of Engineering

Coursework and Grading

A total of 1,000 points are available in CS 240, along with many opportunities to earn extra credit. The points are broken down in the following way:

- **200 points**: Homeworks (10 × 20 points)
  - Points over 200 are extra credit!
  - Usually on PrairieLearn, but occasionally another platform
- **400 points**: Open-book Midterm Exams (2 × 200 points)
  - Midterm 1 Exam: Thursday, October 8, 2020
  - Midterm 2 Exam: Thursday, November 19, 2020
- **250 points**: Machine Projects (10 weeks × 25 points)
  - 7-8 MPs, including short (1-week) and long (2-week) MPs
  - Long MPs are worth 50 points, short MPs are worth 25 points
- **100 points**: Final Project
  - Multi-week Final Project, presented during finals weeks instead of a final exam (no final exam!)
- **50 points**: Participation
  - If you regularly engage with the course, you'll receive the full points. I really want your feedback on how to build CS 240 to be the best course possible and you to enjoy it! :)

Final Course Grades

Your final course grade is determined by the number of points you earned during the semester:

Points	Grade	Points	Grade	Points	Grade
[1070, ∞)	A+	[930, 1070)	A	[900, 930)	A-
[870, 900)	B+	[830, 870)	B	[800, 830)	B-
[770, 800)	C+	[730, 770)	C	[700, 730)	C-
[670, 700)	D+	[630, 670)	D	[600, 630)	D-
		(600, 0]	F		

We never curve individual exam or assignment scores; instead, if necessary, we may lower the points required for each grade cutoff to be lower than the stated cutoff. In no case will we raise the cutoff.

Foundations of Computer Systems

There are six major components to a computer, which we will refer to as the "foundations" of a computer system:

[1]:

[2]:

[3]:

[4]:

[5]:

[6]:

System-level Abstractions

After covering the "foundations", we will begin to abstract the entire system as **node** and explore more complex topics:

[1]:

[2]:

[3]:

Representing Data: Binary

All data within a computer is _______________; either **`0`** or **`1`**.

Converting between base-2 and base-10:

1_2: ______________ $_{10}$

10_2: ______________ $_{10}$

11_2: ______________ $_{10}$

100_2: ______________ $_{10}$

$101\ 1000_2$: ______________ $_{10}$

Any value can be represented in binary by writing it in base-2:

4_{10}: ______________ $_2$

7_{10}: ______________ $_2$

18_{10}: ______________ $_2$

In C/C++, you can write a number in binary by prefixing the number with **`0b`**:

11_{10}: ______________

33_{10}: ______________

Bit Manipulation:

We can manipulate bits by binary operations:

`AND`, **`&`** operator:

`OR`, **`|`** operator:

`XOR`, **`^`** operator:

`NOT`, **`!`** or **`~`** operator:

Bit Manipulation:

A	B	A & B	A | B	A ^ B	!A
`1100`	`1010`				
`110011`	`11`				
`101`	`010`				

Representing Data: Hexadecimal

Binary data gets really long, really fast! The number of students enrolled at University of Illinois is `0b1100011111111100` (!!).

- To represent binary data in a compact way, we often will use **hexadecimal** -- or "base-16 -- denoted by the prefix `0x`.

Hexadecimal Digits:

11_{10}: `0x`_____________ 87_{10}: `0x`_____________

34_{10}: `0x`_____________

Hexadecimal is particularly useful as it ______________________**:**

University of Illinois student population in Fall 2019 (51,196):				
`0b`	`1100`	`0111`	`1111`	`1100`
`0x`				

Number of people following Taylor Swift on Twitter (87,042,176):	
`0b`	`101 0011 0000 0010 1000 1000 0000`
`0x`	

Orders of Magnitude

Bits are organized into 8-bit chunks called _______.

Bytes are organized into by orders of magnitude.

1. Historical Use of $\mathbf{10^x}$:

   4 KB on disk == _________ B

2. Historical Use of $\mathbf{2^x}$:

   4 KB in RAM == _________ B

Prefix	Magnitude	Prefix	Magnitude
kilo-, K-	10^3	**kibi-, Ki-**	
mega-, M-		**mebi, Mi-**	
giga-, G-		**gibi-, Gi-**	2^{30}
tera-, T-	10^{12}	**tebi-, Ti-**	2^{40}

Example: Downloading a 1 GiB file on a "1 gig" connection:

Representing Letters: ASCII

Representing numbers is great -- but what about words? Can we make sentences with binary data?

- **Key Idea:** Every letter is ______ binary bits.
  (This means that every letter is ______ hex digits.)

- Global standard called the **American Standard Code for Information Interchange (ASCII)** is a ____________ ________________ for translating numbers to characters.

ASCII Character Encoding Examples:

Binary	Hex	Char.	Binary	Hex	Char.
0b0100 0001	0x41	A	0b0110 0001	0x61	a
0b0100 0010	0x42	B	0b0110 0010	0x62	b
		C			c
		D			d
0b0010 0100	0x24	$	0b0111 1011	0x7b	{

...and now we can form sentences!

Q: Are there going to be any issues with ASCII?

Representing Letters: Other Character Encodings

Since ASCII uses only 8 bits, we are limited to only 256 unique characters. There's far more than 256 characters -- and what about EMOJIs?? 🎉

- **Many** other character encodings exist other than ASCII.
- The most widely used character encoding is known as **Unicode Transformation Format (8-bit)** or _______.

UTF-8 uses a ____________-bit design where each character by be any of the following:

Specifically the first four bits tell us about the number of bytes used to encode the character:

Length	Byte #1	Byte #2	Byte #3	Byte #4
1-byte	`0___ ____`			
2-bytes:	`110_ ____`	`10__ ____`		
3-bytes:	`1110 ____`	`10__ ____`	`10__ ____`	
4-bytes:	`1111 0___`	`10__ ____`	`10__ ____`	`10__ ____`

For all single-byte characters, the ASCII character encoding is used. This means **`'a'`** is still **`0x61`**. Unicode characters are represented by **`U+`** followed by the hex value, like **`U+61`**.

Example: ε (epsilon) is defined as U+03B5. How do we encode this?

Example: I received the following binary message encoded in UTF-8:

`0100 1000 0110 1001 1111 0000 1001 1111 1000 1110 1000 1001`

1. What is the hexadecimal representation of this message?

2. What is the character length of this message?

3. What does the message say?

...what technique did we just apply to find the Unicode character code?

Finally, UTF-8 has seen universal design due to several brilliant features: