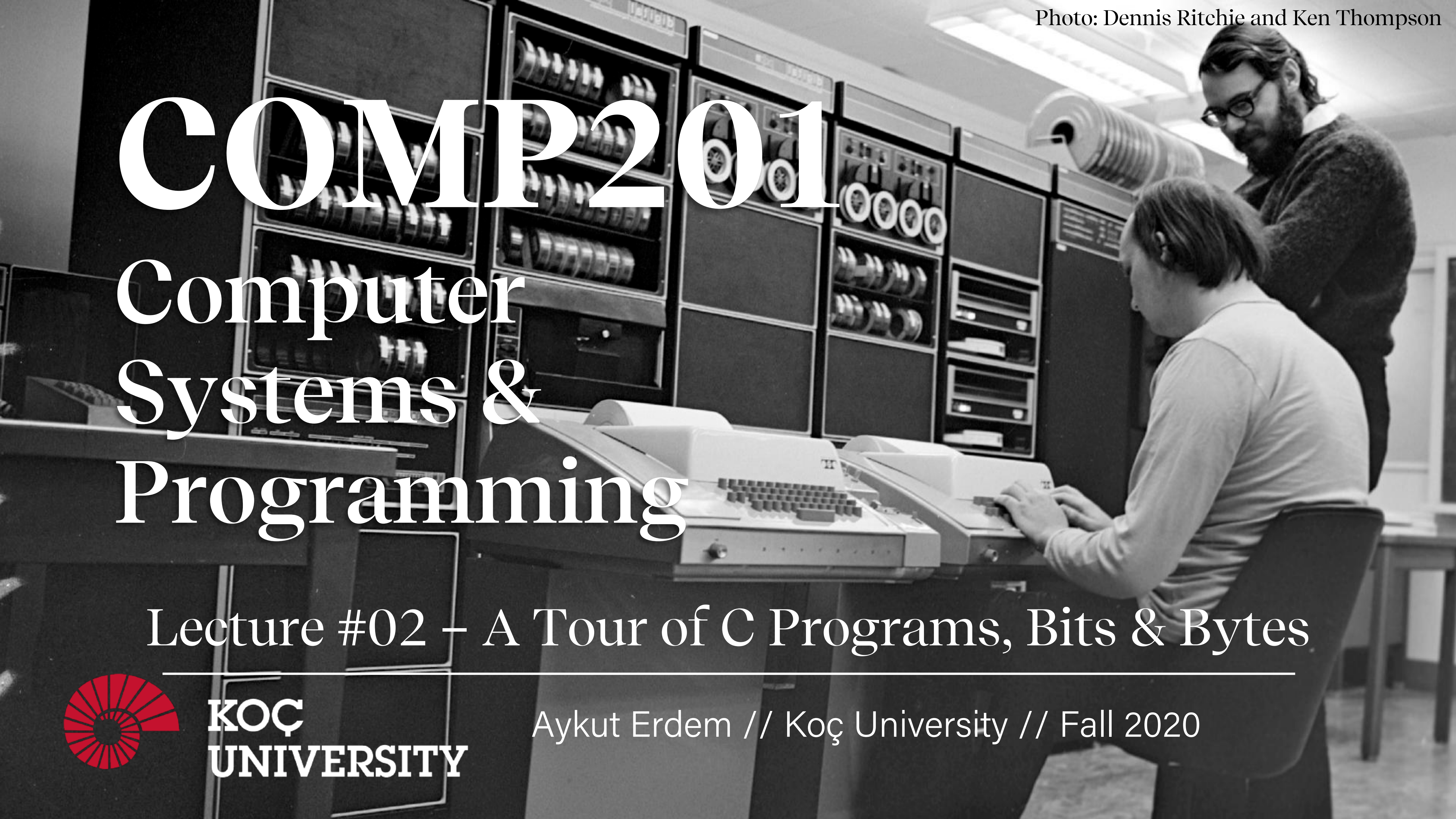


Photo: Dennis Ritchie and Ken Thompson

COMP201

Computer Systems & Programming

Lecture #02 – A Tour of C Programs, Bits & Bytes

KOÇ UNIVERSITY

Aykut Erdem // Koç University // Fall 2020

Plan For Today

- Getting Started With C
- Bits and Bytes
- Hexadecimal
- Integer Representations
- Unsigned Integers

Disclaimer: Slides for this lecture were borrowed from

—Nick Troccoli's Stanford CS107 class

Good news, everyone!

- Lab preference submissions are open! You may submit your preferences anytime until ~~Friday 10/9 at 5PM~~ Thursday 10/9 at 11:59PM.
- Assg0 will be out today (due Oct 16)
- C bootcamp (today & tomorrow)

Plan For Today

- Getting Started With C
- Bits and Bytes
- Hexadecimal
- Integer Representations
- Unsigned Integers

The C Language

C was created around 1970 to make writing Unix and Unix tools easier.

- Part of the C/C++/Java family of languages (C++ and Java were created later)
- Design principles:
  - Small, simple abstractions of hardware
  - Minimalist aesthetic
  - Prioritizes efficiency and minimalism over safety and high-level abstractions

C vs. C++ and Java

They all share:

- Syntax
- Basic data types
- Arithmetic, relational, and logical operators

C doesn’t have:

- More advanced features like operator overloading, default arguments, pass by reference, classes and objects, ADTs, etc.
- Extensive libraries (no graphics, networking, etc.) – this means not much to learn C!
- many compiler and runtime checks (this may cause security vulnerabilities!)

Programming Language Philosophies

- **C is procedural:** you write functions, rather than define new variable types with classes and call methods on objects. C is small, fast and efficient.

- **C++ is procedural, with objects:** you write functions, and define new variable types with classes, and call methods on objects.

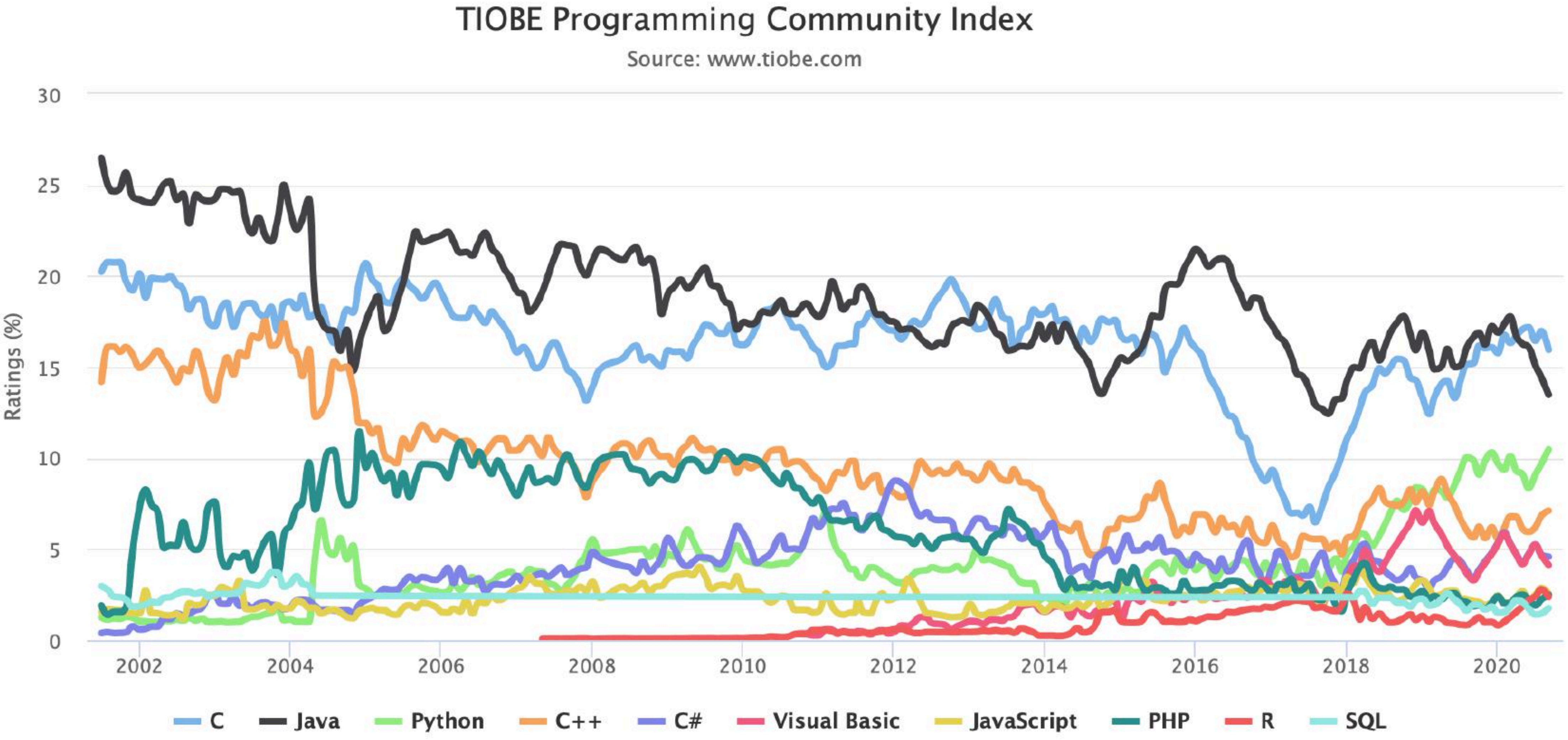
- **Python is also procedural, but dynamically typed:** you still write functions and call methods on objects, but the development process is very different.

- **Java is object-oriented:** virtually everything is an object, and everything you write needs to conform to the object-oriented design pattern.

Why C?

- Many tools (and even other languages, like Python!) are built with C.
- C is the language of choice for fast, highly efficient programs.
- C is popular for systems programming (operating systems, networking, etc.)
- C lets you work at a lower level to manipulate and understand the underlying system.

Programming Language Popularity



https://www.tiobe.com/tiobe-index/

Our First C Program

```c
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>   // for printf

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    return 0;
}
```

Our First C Program

```c
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>   // for printf

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    return 0;
}
```

Program comments
You can write block or inline comments.

Our First C Program

```c
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>  // for printf

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    return 0;
}
```

Import statements
C libraries are written with angle brackets.
Local libraries have quotes:
#include "lib.h"

Our First C Program

```
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>   // for printf

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    return 0;
}
```

main function – entry point for the program
Should always return an integer (0 = success)

Our First C Program

```
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>   // for printf

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    return 0;
}
```

Main parameters – main takes two parameters, both relating to the command line arguments used to execute the program.

argc is the number of arguments in **argv**
argv is an array of arguments **(char * is C string)**

Our First C Program

```c
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>   // for printf

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    return 0;
}
```

printf – prints output to the screen

Familiar Syntax

```
int x = 42 + 7 * -5;                  // variables, types
double pi = 3.14159;
char c = 'Q';                         /* two comment styles */


for (int i = 0; i < 10; i++) {        // for loops
    if (i % 2 == 0) {                 // if statements
        x += i;
    }
}

while (x > 0 && c == 'Q' || b) {      // while loops, logic
    x = x / 2;
    if (x == 42) { return 0; }
}

binky(x, 17, c);                     // function call
```

Boolean Variables

To declare Booleans, (e.g. **`bool b = ____`**), you must include **`stdbool.h`**:

```
#include <stdio.h>      // for printf
#include <stdbool.h>    // for bool

int main(int argc, char *argv[]) {
    bool x = 5 > 2 && binky(argc) > 0;
    if (x) {
        printf("Hello, world!\n");
    } else {
        printf("Howdy, world!\n");
    }
    return 0;
}
```

Boolean Expressions

C treats a nonzero value as <u>true</u>, and a zero value as <u>false</u>:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    int x = 5;
    if (x) {   // true
        printf("Hello, world!\n");
    } else {
        printf("Howdy, world!\n");
    }
    return 0;
}
```

Console Output: printf

```
printf(text, arg1, arg2, arg3);
```

```
    // Example
    char *classPrefix = "COMP";
    int classNumber = 201;
    printf("You are in %s%d", classPrefix, classNumber);    // You are in COMP201
```

`printf` makes it easy to print out the values of variables or expressions.

If you include *placeholders* in your printed text, `printf` will replace each placeholder *in order* with the values of the parameters passed after the text.

`%s (string)`   `%d (integer)`   `%f (double)`

Question Break!

Writing, Debugging and Compiling

We will use:

- the **vi/emacs** text editor to write our C programs
- the **make** tool to compile our C programs
- the **gdb** debugger to debug our programs
- the **valgrind** tools to debug memory errors and measure program efficiency

Demo: Compiling And Running A C Program

Working On C Programs Recap

- **ssh** – remotely log in to `linuxpool` computers (*later*)
- **Vi/Emacs** – text editor to write and edit C programs
  - Use the mouse to position cursor, scroll, and highlight text
  - :w / Ctl-x Ctl-s to save, :q / Ctl-x Ctl-c to quit
- **make** – compile program using provided Makefile
- **./myprogram** – run executable program (optionally with arguments)
- **make clean** – remove executables and other compiler files
- Lecture codes are accessible at Blackboard

COMP201 Topic 1: How can a computer represent integer numbers?

THE WALL STREET JOURNAL.

SHARE

US Air, Comair Scramble To Get Back to Normal

A Wall Street Journal NEWS ROUNDUP

Dec. 27, 2004 12:01 am ET

SAVE PRINT TEXT

Air travelers on two airlines continued to face canceled flights and lost luggage after weather, worker absences and computer glitches left thousands stranded in airports over the holiday weekend. Comair Inc., a regional carrier owned by Delta Air Lines that canceled its entire schedule Saturday, resumed limited flights yesterday but said it wouldn't return to normal until midweek.

US Airways Group Inc. blamed more than 400 canceled flights and thousands of pieces of stranded luggage on large numbers of workers who called in sick, as well as on a heavy winter storm. A spokesman said the carrier had no evidence of a concerted job action, but the troubles underscore the problems low morale could cause the carrier as it struggles to emerge from bankruptcy-court protection.

It was unclear how many holiday travelers were affected, though the major disruptions appeared to be limited to US Airways and Comair. UAL Corp.'s United Airlines and Northwest Airlines reported weather difficulties in Chicago and Detroit, respectively. AMR Corp.'s American Airlines said it experienced problems due to unusual snowfall at its Dallas-Fort Worth hub over the weekend

Demo: Unexpected Behavior

`airline.c`

Plan For Today

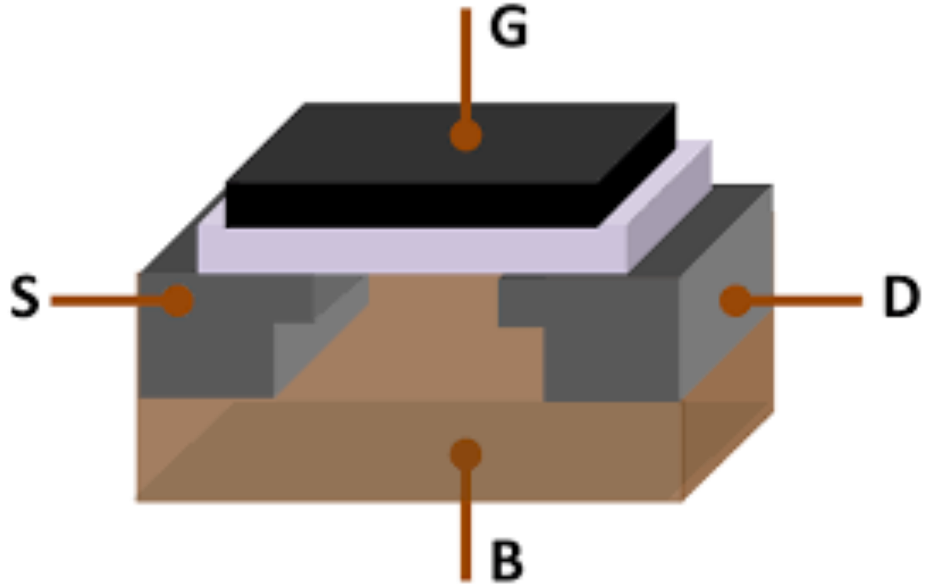
- Getting Started With C
- Bits and Bytes
- Hexadecimal
- Integer Representations
- Unsigned Integers

0

1

Bits

- Computers are built around the idea of two states: “on” and “off”. Transistors represent this in hardware, and bits represent this in software!



One Bit At A Time

- We can combine bits, like with base-10 numbers, to represent more data. **8 bits = 1 byte**.
- Computer memory is just a large array of bytes! It is *byte-addressable*; you can't address (store location of) a bit; only a byte.
- Computers still fundamentally operate on bits; we have just gotten more creative about how to represent different data as bits!
    - Images
    - Audio
    - Video
    - Text
    - And more...

Base 10

5 9 3 4

Digits 0-9 (0 to base-1)

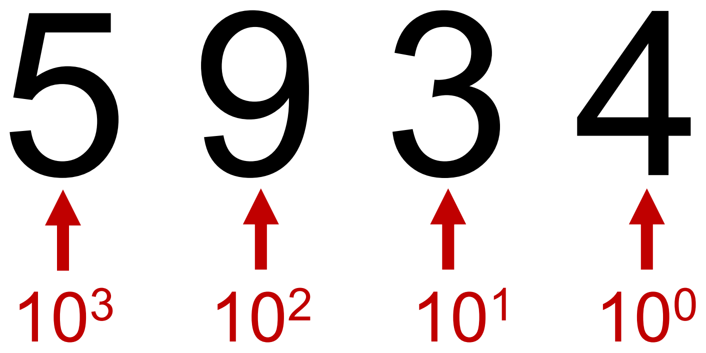
Base 10

5 9 3 4

thousands hundreds tens ones

$$= \mathbf{5}*1000 + \mathbf{9}*100 + \mathbf{3}*10 + \mathbf{4}*1$$

Base 10



Base 10

5 9 3 4

10^X: 3 2 1 0

Base 2

1 0 1 1

2^x: 3 2 1 0

Digits 0-1 (*0* to *base-1*)

Base 2

1 0 1 1

2^3 2^2 2^1 2^0

Base 2

Most significant bit (MSB)

Least significant bit (LSB)

1 0 1 1

eights fours twos ones

$$= \mathbf{1}*8 + \mathbf{0}*4 + \mathbf{1}*2 + \mathbf{1}*1 = 11_{10}$$

Base 10 to Base 2

Question: What is 6 in base 2?

- Strategy:
  - What is the largest power of 2 $\leq$ 6?

Base 10 to Base 2

Question: What is 6 in base 2?

- Strategy:
  - What is the largest power of 2 ≤ 6? **2^2=4**

$$\underset{2^3}{\underline{0}} \quad \underset{2^2}{\underline{1}} \quad \underset{2^1}{\underline{}} \quad \underset{2^0}{\underline{}}$$

Base 10 to Base 2

Question: What is 6 in base 2?

- Strategy:
  - What is the largest power of 2 ≤ 6? **2^2=4**
  - Now, what is the largest power of 2 ≤ 6 – 2^2?

0	1	_	_
2^3	2^2	2^1	2^0

Base 10 to Base 2

Question: What is 6 in base 2?

- Strategy:
  - What is the largest power of 2 ≤ 6? **2^2=4**
  - Now, what is the largest power of 2 ≤ 6 – 2^2? **2^1=2**

0	1	1	_
2^3	2^2	2^1	2^0

Base 10 to Base 2

Question: What is 6 in base 2?

- Strategy:
  - What is the largest power of 2 ≤ 6? **2^2=4**
  - Now, what is the largest power of 2 ≤ 6 – 2^2? **2^1=2**
  - 6 – 2^2 – 2^1 = 0!

0	1	1	_
2^3	2^2	2^1	2^0

Base 10 to Base 2

Question: What is 6 in base 2?

- Strategy:
  - What is the largest power of 2 ≤ 6? $\mathbf{2^2=4}$
  - Now, what is the largest power of 2 ≤ 6 – 2^2? $\mathbf{2^1=2}$
  - $6 - 2^2 - 2^1 = 0$!

0	1	1	0
2^3	2^2	2^1	2^0

Base 10 to Base 2

Question: What is 6 in base 2?

- Strategy:
  - What is the largest power of 2 ≤ 6? **2^2=4**
  - Now, what is the largest power of 2 ≤ 6 – 2^2? **2^1=2**
  - 6 – 2^2 – 2^1 = 0!

0	1	1	0
2^3	2^2	2^1	2^0

= **0***8 + **1***4 + **1***2 + **0***1 = 6

Practice: Base 2 to Base 10

What is the base-2 value 1010 in base-10?

a) 20
b) 101
c) 10
d) 5
e) Other

Practice: Base 10 to Base 2

What is the base-10 value 14 in base 2?

a) 1111
b) 1110
c) 1010
d) Other

Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store?

Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store? **minimum = 0** **maximum = ?**

Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store? **minimum = 0** **maximum = ?**

11111111

2^x: 7 6 5 4 3 2 1 0

Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store? **minimum = 0** **maximum = ?**

11111111

2^x: 7 6 5 4 3 2 1 0

- **Strategy 1:** $1*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 255$

Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store? **minimum = 0** **maximum = 255**

11111111

2^x: 7 6 5 4 3 2 1 0

- **Strategy 1:** $1*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 255$
- **Strategy 2:** $2^8 - 1 = 255$

Multiplying by Base

1450 x 10 = 1450<u>0</u>

1100_2 x 2 = 1100<u>0</u>

Key Idea: inserting 0 at the end multiplies by the base!

Dividing by Base

$$1450 / 10 = 145$$

$$1100_2 / 2 = 110$$

Key Idea: removing 0 at the end divides by the base!

Plan For Today

- Getting Started With C
- Bits and Bytes
- Hexadecimal
- Integer Representations
- Unsigned Integers

Hexadecimal

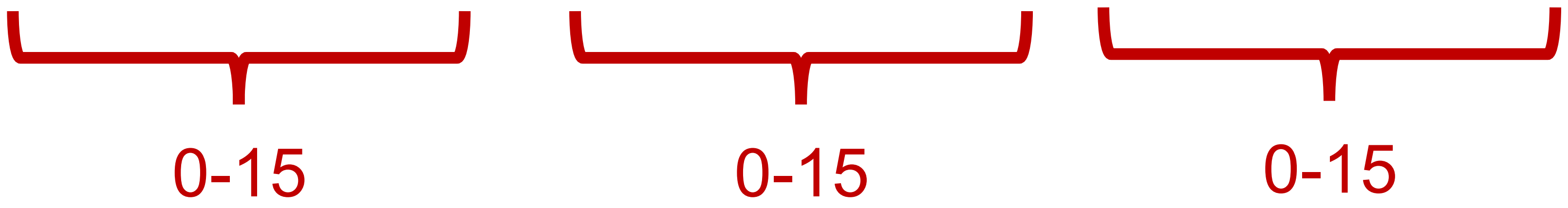
- When working with bits, oftentimes we have large numbers with 32 or 64 bits.
- Instead, we'll represent bits in *base-16 instead;* this is called **hexadecimal**.

0110 1010 0011

0-15 0-15 0-15

Hexadecimal

- When working with bits, oftentimes we have large numbers with 32 or 64 bits.
- Instead, we'll represent bits in *base-16 instead;* this is called **hexadecimal**.



Each is a base-16 digit!

Hexadecimal

- Hexadecimal is *base-16*, so we need digits for 1-15. How do we do this?

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
										10	11	12	13	14	15

Hexadecimal

Hex digit	0	1	2	3	4	5	6	7
Decimal value	0	1	2	3	4	5	6	7
Binary value	0000	0001	0010	0011	0100	0101	0110	0111

Hex digit	8	9	A	B	C	D	E	F
Decimal value	8	9	10	11	12	13	14	15
Binary value	1000	1001	1010	1011	1100	1101	1110	1111

Hexadecimal

- We distinguish hexadecimal numbers by prefixing them with **`0x`**, and binary numbers with **`0b`**.
- E.g. **`0xf5`** is **`0b11110101`**

0x f 5

1111 0101

Practice: Hexadecimal to Binary

What is **`0x173A`** in binary?

Hexadecimal	**1**	**7**	**3**	**A**
Binary	**0001**	**0111**	**0011**	**1010**

Practice: Hexadecimal to Binary

What is **`0b1111001010`** in hexadecimal? (*Hint: start from the right*)

Binary	**11**	**1100**	**1010**
Hexadecimal	**3**	**C**	**A**

Question Break!

Plan For Today

- Getting Started With C
- Bits and Bytes
- Hexadecimal
- Integer Representations
- Unsigned Integers

Number Representations

- **Unsigned Integers:** positive and 0 integers. (e.g. 0, 1, 2, … 99999…
- **Signed Integers:** negative, positive and 0 integers. (e.g. …-2, -1, 0, 1,… 9999…)

- **Floating Point Numbers:** real numbers. (e,g. 0.1, -12.2, $1.5x10^{12}$)

Number Representations

- **Unsigned Integers:** positive and 0 integers. (e.g. 0, 1, 2, ... 99999...
- **Signed Integers:** negative, positive and 0 integers. (e.g. ...-2, -1, 0, 1,... 9999...)

- **Floating Point Numbers:** real numbers. (e,g. 0.1, -12.2, $1.5x10^{12}$)

→ More on this next week!

Number Representations

C Declaration	Size (Bytes)
`int`	**4**
`double`	**8**
`float`	**4**
`char`	**1**
`char *`	**8**
`short`	**2**
`long`	**8**

In The Days Of Yore…

C Declaration	Size (Bytes)
`int`	4
`double`	8
`float`	4
`char`	1
`char *`	4
`short`	2
`long`	4

Transitioning To Larger Datatypes

- **Early 2000s:** most computers were **32-bit.** This means that pointers were **4 bytes (32 bits).**
- 32-bit pointers store a memory address from 0 to 2^{32}-1, equaling **2^{32} bytes of addressable memory**. This equals **4 Gigabytes**, meaning that 32-bit computers could have at most **4GB** of memory (RAM)!
- Because of this, computers transitioned to **64-bit.** This means that datatypes were enlarged; pointers in programs were now **64 bits.**
- 64-bit pointers store a memory address from 0 to 2^{64}-1, equaling **2^{64} bytes of addressable memory.** This equals **16 Exabytes**, meaning that 64-bit computers could have at most **1024*1024*1024 GB** of memory (RAM)!

Lecture Plan

- Getting Started With C
- Bits and Bytes
- Hexadecimal
- Integer Representations
- Unsigned Integers

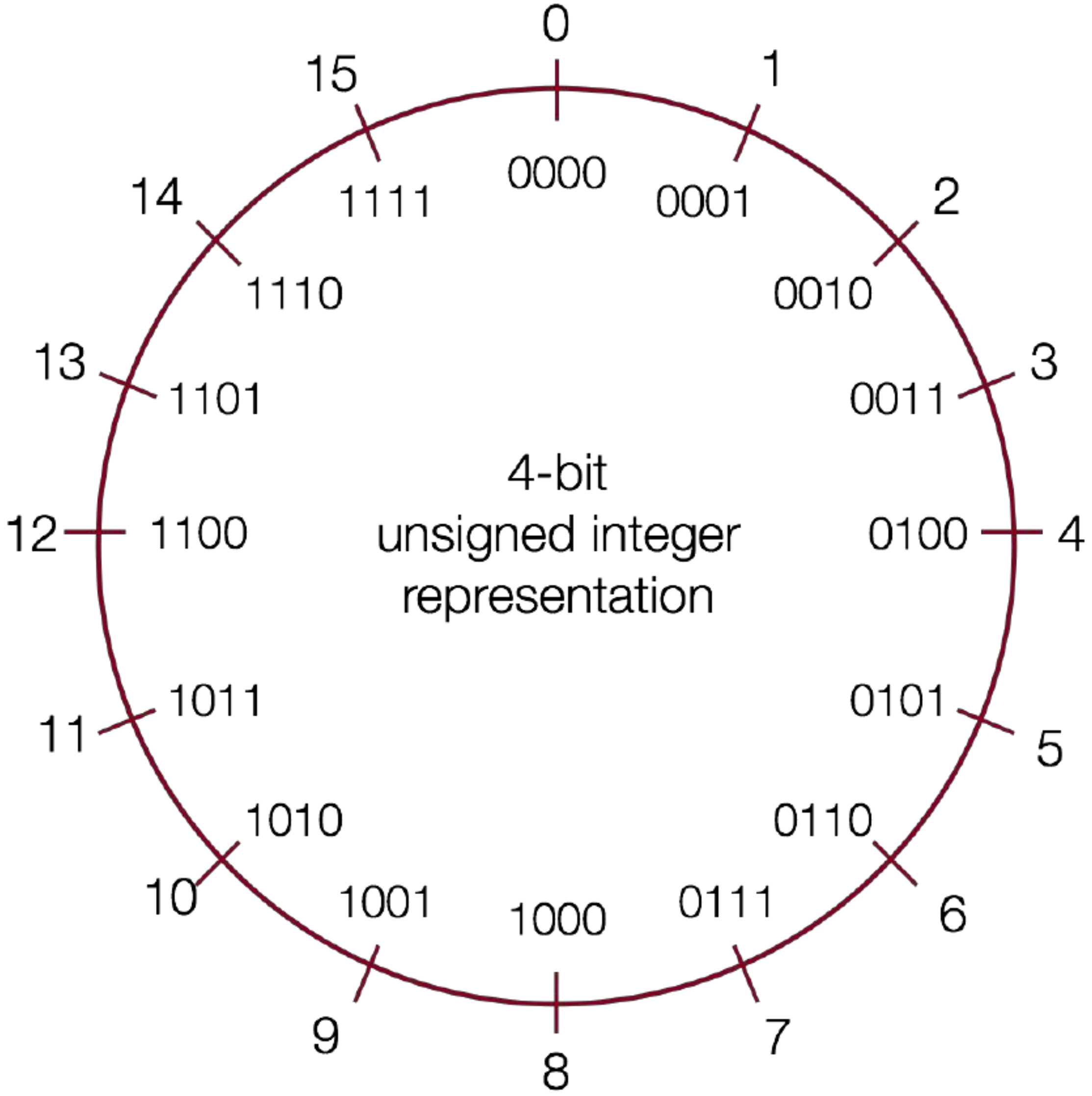
Unsigned Integers

- An **unsigned** integer is 0 or a positive integer (no negatives).
- We have already discussed converting between decimal and binary, which is a nice 1:1 relationship. Examples:

```
0b0001 = 1
0b0101 = 5
0b1011 = 11
0b1111 = 15
```

- The range of an unsigned number is 0 → 2^w - 1, where *w* is the number of bits. E.g. a 32-bit integer can represent 0 to 2^{32} – 1 (4,294,967,295).

Unsigned Integers



Let's Think Now

To ponder till Friday:

A **signed** integer is a negative, 0, or positive integer. How can we represent both negative *and* positive numbers in binary?

Recap

- Getting Started With C
- Bits and Bytes
- Hexadecimal
- Integer Representations
- Unsigned Integers

Next Time on COMP201

- Make sure to reboot Boeing Dreamliners every 248 days
- Comair/Delta airline had to cancel thousands of flights days before Christmas
- Many operating systems may have issues storing timestamp values beginning on Jan 19, 2038
- Reported vulnerability CVE-2019-3857 in libssh2 may allow a hacker to remotely execute code

Next time: *More on how a computer represents integer numbers? What are the limitations?*