COMP201
Computer Systems & Programming

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

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

TIOBE Programming Community Index
Source: www.tiobe.com

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

Program comments
You can write block or inline comments.
/*
 * 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

/*
 * 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

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

for (int i = 0; i < 10; i++) {
    if (i % 2 == 0) {
        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`:

```c
#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 \texttt{true}, and a zero value as \texttt{false}:

```c
#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(\textit{text}, \textit{arg1}, \textit{arg2}, \textit{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 placeholder\textit{s} in your printed text, \texttt{printf} will replace each placeholder \textit{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?
US Air, Comair Scramble To Get Back to Normal

A Wall Street Journal NEWS ROUNDUP
Dec. 27, 2004 12:01 am ET

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
Plan For Today

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

\[\begin{align*}
5 & \times 1000 + 9 & \times 100 + 3 & \times 10 + 4 & \times 1 \\
\end{align*}\]
Base 10

5 9 3 4

$10^3 \uparrow$  $10^2 \uparrow$  $10^1 \uparrow$  $10^0 \uparrow$
Base 10

5 9 3 4

10^x:  3  2  1  0
Base 2

\[ 1 \quad 0 \quad 1 \quad 1 \]

\[ 2^x: \quad 3 \quad 2 \quad 1 \quad 0 \]

Digits 0-1 (0 to \textit{base-1})
Base 2

1 0 1 1

$2^3 \quad 2^2 \quad 2^1 \quad 2^0$
Base 2

1 0 1 1

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

Most significant bit (MSB)

Least significant bit (LSB)
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? $2^2 = 4$
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?
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 – $2^2$? $2^1 = 2$

$$
\begin{array}{cccc}
0 & 1 & 1 \\
2^3 & 2^2 & 2^1 & 2^0
\end{array}
$$
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? \(2^2 = 4\)
  • Now, what is the largest power of 2 ≤ 6 – \(2^2\)? \(2^1 = 2\)
  • \(6 - 2^2 - 2^1 = 0\)!

\[\begin{array}{cccc}
0 & 1 & 1 & 0 \\
\hline
2^3 & 2^2 & 2^1 & 2^0 \\
\end{array}\]
Question: What is 6 in base 2?

- Strategy:
  - What is the largest power of $2 \leq 6$? $2^2 = 4$
  - Now, what is the largest power of $2 \leq 6 - 2^2$? $2^1 = 2$
  - $6 - 2^2 - 2^1 = 0$!

$$0 \quad 1 \quad 1 \quad 0$$

$$2^3 \quad 2^2 \quad 2^1 \quad 2^0$$

$$= 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 = 6$$
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 = ?
Byte Values

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

• Strategy 1: \[1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 255\]
Byte Values

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

• Strategy 1: $1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 255$

• Strategy 2: $2^8 - 1 = 255$
Multiplying by Base

1450 \times 10 = 14500

1100_2 \times 2 = 11000_2

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

1450 / 10 = 145

1100₂ / 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.
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 \textit{base-16}, so we need digits for 1-15. How do we do this?
# Hexadecimal

<table>
<thead>
<tr>
<th>Hex digit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Binary</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hex digit</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>Binary</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
</tr>
</tbody>
</table>
Hexadecimal

• We distinguish hexadecimal numbers by prefixing them with `0x`, and binary numbers with `0b`.
• E.g. `0xf5` is `0b1110101`
Practice: Hexadecimal to Binary

What is $0x173A$ in binary?

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>1</th>
<th>7</th>
<th>3</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>0001</td>
<td>0111</td>
<td>0011</td>
<td>1010</td>
</tr>
</tbody>
</table>
Practice: Hexadecimal to Binary

What is 0b111001010 in hexadecimal? (*Hint: start from the right*)

<table>
<thead>
<tr>
<th>Binary</th>
<th>11</th>
<th>1100</th>
<th>1010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hexadecimal</td>
<td>3</td>
<td>C</td>
<td>A</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>C Declaration</th>
<th>Size (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>char</td>
<td>1</td>
</tr>
<tr>
<td>char *</td>
<td>8</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
</tr>
</tbody>
</table>
In The Days Of Yore...

<table>
<thead>
<tr>
<th>C Declaration</th>
<th>Size (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>char</td>
<td>1</td>
</tr>
<tr>
<td>char *</td>
<td>4</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
</tr>
</tbody>
</table>
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:

  
  \[
  \begin{align*}
  0b0001 &= 1 \\
  0b0101 &= 5 \\
  0b1011 &= 11 \\
  0b1111 &= 15
  \end{align*}
  \]

• The range of an unsigned number is \(0 \rightarrow 2^w - 1\), 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

4-bit unsigned integer representation
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?