CS 107
Lecture 1: Welcome

Monday, January 8, 2024

Computer Systems
Winter 2024
Stanford University
Computer Science Department

Lecturer: Chris Gregg

Reading: Course reader: Introduction, Number
Formats used in CS 107, Bits and Bytes

#include<stdio.h>
#include<stdlib.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}

(gdb)
(gdb) disassemble main
Dump of assembler code for function main:
0x000000000040052d <+0>:    push   %rbp
0x000000000040052e <+1>:    mov    %rsp,%rbp
0x0000000000400531 <+4>:    mov    $0x4005d4,%edi
0x0000000000400536 <+9>:    callq  0x400410 <puts@plt>
0x000000000040053b <+14>:   mov    $0x0,%eax
0x0000000000400540 <+19>:   pop    %rbp
End of assembler dump.
Today's Topics

• What is CS107?
• Who We Are
• Course Components and Overview
• The C Language
• Logistics
  • Exams
  • Labs
  • Assignment 0
  • Lab Signup
• Bits and Bytes
• 0b10100 Questions
What is CS107?

- The CS106 series teaches you how to solve problems as a programmer
- Many times CS106 instructors had to say “just don’t worry about that” or “it probably doesn’t make sense why that happens, but ignore it for now” or “just type this to fix it”
- **CS107 finally takes you behind the scenes**
- How do things really work in there?
  - It’s not quite down to hardware or physics/electromagnetism (those will have to stay even further behind the scenes for now!)
  - It’s how things work **inside Python/C++** (we will explore from C), and how your programs map onto the components of computer systems
CS107 Learning Goals

- The goals for CS107 are for students to gain **mastery** of
  - writing C programs with complex use of memory and pointers
  - an accurate model of the address space and compile/runtime behavior of C programs
- to achieve **competence** in
  - translating C to/from assembly
  - writing programs that respect the limitations of computer arithmetic
  - identifying bottlenecks and improving runtime performance
  - writing code that correctly ports to other architectures
- working effectively in UNIX development environment
- and have **exposure** to
  - a working understanding of the basics of computer architecture
Who We Are: Chris

Chris Gregg
https://web.stanford.edu/~cgregg/chris-gregg/
cgregg@stanford.edu

- B.S. Electrical Engineering, Johns Hopkins
- M.Ed, High School Physics Teaching, Harvard
- Ph.D., Computer Engineering, University of Virginia
- U.S. Navy, Active Duty (7 years), Reserves (15 years), Cryptologist, Retired Commander
- High School Teaching: Brooklinne High School, Pacific Collegiate School
- University Teaching: UVA, University of Maryland, Tufts University, Stanford
- Associate Professor (Teaching), Associate Chair for Education, CS Department
Who We Are: CAs

Theo Kanell
Rita Tlemcani
Sophie Andrews
Selaine Rodriguez
Ola Adekola
Melissa Lee
Trevor Carrell
Andreea Jitaru
Hari Vallabhaneni
Luke Babbitt
Abi Lopez
CS107A CA:
Arman Aydin
Advaya Gupta
Jessica Yu
Textbook: Bryant and O’Hallaron, 3rd Edition

You must get the 3rd Edition, as things have significantly changed since the previous editions.

- The suggested C reference is just one suggestion
  - You could do just as well with a different C book
  - You could do just as well with Google or websites like http://www.cplusplus.com/reference/clibrary/
    - Just need somewhere to turn when you have a question about C
There is a course reader, which condenses much of the material for the course:

https://stanford.edu/~cgregg/cgi-bin/107-reader

• If you find typos, let me know!
## Course Components and Overview

<table>
<thead>
<tr>
<th>Week</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Admin, UNIX environment, Integer representation</td>
</tr>
<tr>
<td>2</td>
<td>Bits/bitwise ops, computer arithmetic, C pointers/arrays</td>
</tr>
<tr>
<td>3</td>
<td>C-strings, C stdlib, dynamic allocation</td>
</tr>
<tr>
<td>4</td>
<td>C generics, void *, function pointers</td>
</tr>
<tr>
<td>5</td>
<td>Floating point representation, intro to assembly</td>
</tr>
<tr>
<td>6</td>
<td>x86-64: addressing, ALU ops  <strong>Midterm: Thu Feb 8th, Evening</strong></td>
</tr>
<tr>
<td>7</td>
<td>x86-64: control, function calls, runtime stack</td>
</tr>
<tr>
<td>8</td>
<td>Address space, dynamic memory management</td>
</tr>
<tr>
<td>9</td>
<td>Performance / Optimization</td>
</tr>
<tr>
<td>10</td>
<td>Advanced topics, wrap/review</td>
</tr>
<tr>
<td>11</td>
<td><strong>Final: Monday Mar 18th 3:30PM-6:30PM</strong></td>
</tr>
</tbody>
</table>
A Bit About the Honor Code

The Basics:

- **Read:**
  - the honor code web page
  - Handout - Honor Code Policy
  - Handout - Tutoring Policy
  - Yes, it's a lot of reading
- We will run software that checks for students working with each others' code. **The software is very good.**
- We have methods for determining if you used ChatGPT. I've been working on it for a year. **The methods are very good.**
- If I catch you violating the honor code:
  - I will forward the information to the Office of Community Standards
  - Once that process completes, you will **fail the class**.
    - This is true even if you copy / collaborate / use ChatGPT for a single assignment.
- Why do I have this policy? So **honest students are not harmed by dishonest students when it comes to the final grades for the class.**
The C Language: History and Background

- Birthdate around 1970
- Created to make writing Unix (the OS itself) and tools for Unix easier
- Part of the C/C++/Java family of languages
  › (with C++ and Java coming later)
- Design principles:
  › Small, simple abstractions of hardware
  › Minimalist aesthetic
  › C is much more concerned with efficiency and minimalism than safety
    (Java/Python) or convenient high-level services and abstractions (Java, Python, C++)
The C Language: Comparison of C and C++

- **Some things will be very familiar:**
  - Syntax
  - Basic data types
  - Arithmetic, relational, and logical operators
- **You may be sad about what’s missing:**
  - No power features of C++ (overloading operators, default arguments, pass by reference, classes/objects, fancy ADTs)
  - Thin standard libraries (no graphics, networking, etc)
  - Weak compiler checks, almost no runtime checks
- **Benefits:**
  - Small language footprint (not much to learn)
- **Philosophical difference:**
  - Procedural (C)
  - Procedural + Objects (C++)
The C Language: Hello, World!

(lecture code)
Logistics

See the Course Handout for details (link)

Web site: https://cs107.stanford.edu

Class time: 10:30AM-11:20AM, M/W/F, NVIDIA Auditorium

Labs: Various Times We/Th

Exams: Midterm, Thursday, February 8th, Time TBD (evening)
       Final Exam: Monday, March 18th, 3:30pm-6:30pm
(Note: there are no alternate final exam times)
Assignment 0: Unix!

Assignment page: https://web.stanford.edu/class/cs107/assign0/

Assignment already released, due Monday, 1/15

Six parts:
1. Read / View Unix Overview Documents / Videos
2. "Clone" Assignment 0 starter code
3. Answer Questions in readme.txt
4. Honor Code Quiz
5. Run **make** to compile a program, and make minor modifications
6. Submit the assignment
Lab Signup

Online:

https://cs107.stanford.edu/labs

The signup will be available Tuesday, January 9, 10:00am.

Labs will be weekly, starting during **week 2**.

Lab signup will not be first-come, first-served (you'll put in preferences), and the labs are held on Wednesdays, Thursdays
Bits and Bytes Introduction
1
Computers are good at detecting "off" or "on"

We have lots of ways to tell the difference between two different states:

- Clockwise / Counterclockwise
- Lightbulb off / on
- Punchcard hole / no hole
Computers are good at detecting "off" or "on"

Electronic computers are built using transistors

A transistor can be set up to either be "off" or "on" -- this gives us our 0 and 1!
One bit doesn't do much for us!

- We call a single on/off representation a 'bit'.
- But having one bit isn't particularly helpful!
- We only have two states we can represent with one bit!

- If we want more states, we simply combine bits together, much like we do with base 10 representation.
- If we want to combine more than ten states with base 10, we add another digit.

- Base 10 has ten digits: 0 1 2 3 4 5 6 7 8 9
  - We can represent up to ten numbers with one digit in base 10
  - If we want to represent more numbers, we add more digits: 10 11 12 13 14 ...

- Base 2 is the same. We can represent two numbers with one digit: 0 or 1
  - To represent more numbers, we add more digits! 10 11 100 101 110 ...
Combinations of bits can represent everything. We can encode anything we want with bits. E.g., the ASCII character set.

<table>
<thead>
<tr>
<th>ASCII Code: Character to Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0011 0000 O 1000 1111 m 1101 1101</td>
</tr>
<tr>
<td>1 0011 0001 P 101 0000 n 110 1110</td>
</tr>
<tr>
<td>2 0011 0010 Q 101 0001 o 110 1111</td>
</tr>
<tr>
<td>3 0011 0011 R 101 0010 p 111 0000</td>
</tr>
<tr>
<td>4 0011 0100 S 101 0011 q 111 0001</td>
</tr>
<tr>
<td>5 0011 0101 T 101 0100 r 111 0010</td>
</tr>
<tr>
<td>6 0011 0110 U 101 0101 s 111 0011</td>
</tr>
<tr>
<td>7 0011 0111 V 101 0110 t 111 0100</td>
</tr>
<tr>
<td>8 0011 1000 W 101 0111 u 111 0101</td>
</tr>
<tr>
<td>9 0011 1001 X 101 1000 v 111 0110</td>
</tr>
<tr>
<td>A 0100 0001 Y 101 1001 w 111 0111</td>
</tr>
<tr>
<td>B 0100 0010 Z 101 1010 x 111 1000</td>
</tr>
<tr>
<td>C 0100 0011 A 110 0001 y 111 1001</td>
</tr>
<tr>
<td>D 0100 0100 B 110 0010 z 111 1010</td>
</tr>
<tr>
<td>E 0100 0110 C 110 0011 0 010 1110</td>
</tr>
<tr>
<td>F 0100 0111 D 110 0100 1 010 1111</td>
</tr>
<tr>
<td>G 0100 1011 E 110 0101 i 010 1110</td>
</tr>
<tr>
<td>H 0100 1000 F 110 0110 j 010 1111</td>
</tr>
<tr>
<td>I 0100 1001 G 110 0111 ? 011 1111</td>
</tr>
<tr>
<td>J 0100 1010 H 110 1000 l 010 0001</td>
</tr>
<tr>
<td>K 0100 1101 I 110 1001 ' 010 1100</td>
</tr>
<tr>
<td>L 0100 1100 J 110 1010 &quot; 010 0010</td>
</tr>
<tr>
<td>M 0100 1111 K 110 1111 ( 010 1000</td>
</tr>
<tr>
<td>N 0100 1110 L 111 1100 ) 010 1001</td>
</tr>
</tbody>
</table>

space 0010 0000
CS107: Three Number Representations

**Unsigned Integers**: positive integers and zero only
Ex. 0, 1, 2, ..., 74629, 99999999

**Signed Integers**: negative, positive, and zero integers only
Ex. 0, 1, 2, ..., 74629, 99999999
(represented in "two's complement")

**Floating Point Numbers**: a base-2 representation of scientific notation, for real numbers
Ex. 0.0, 0.1, -12.2, 4.87563 \times 10^3, -1.00005 \times 10^{-12}
Computers use a limited number of bits for numbers

Let's write a little program...
Computers use a limited number of bits for numbers.

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

int main() {
    int a = 200;
    int b = 300;
    int c = 400;
    int d = 500;

    int answer = a * b * c * d;
    printf("%d\n",answer);
    return 0;
}
```

```
$ gcc -g -O0 mult-test.c -o mult-test
$ ./mult-test
-884901888
$
```
Computers use a limited number of bits for numbers

Recall that in base 10, you can represent:
- 10 numbers with one digit (0 - 9),
- 100 numbers with two digits (00 - 99),
- 1000 numbers with three digits (000 - 999)

I.e., with \( n \) digits, you can represent up to \( 10^n \) numbers.

In base 2, you can represent:
- 2 numbers with one digit (0 - 1)
- 4 numbers with two digits (00 - 11)
- 8 numbers with three digits (000 - 111)

I.e., with \( n \) digits, you can represent up to \( 2^n \) numbers

The \texttt{C int} type is a "32-bit" number, meaning it uses 32 digits. That means we can represent up to \( 2^{32} \) numbers.
Computers use a limited number of bits for numbers.

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

int main() {
    int a = 200;
    int b = 300;
    int c = 400;
    int d = 500;

    int answer = a * b * c * d;
    printf("%d\n",answer);
    return 0;
}
```

$ gcc -g -O0 mult-test.c -o mult-test
$ ./mult-test
-884901888
$

$2^{32} = 4,294,967,296$

$200 * 300 * 400 * 500 = 12,000,000,000$

Problem?

Turns out it is worse -- ints are signed, meaning that the largest positive number is

$\left(2^{32} / 2\right) - 1 = 2^{31} - 1 = 2,147,483,647$
Computers use a limited number of bits for numbers.

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

int main() {
    int a = 200;
    int b = 300;
    int c = 400;
    int d = 500;

    int answer = a * b * c * d;
    printf("%d\n",answer);
    return 0;
}
```

The good news: all of the following produce the same (wrong) answer:

```
(500 * 400) * (300 * 200)
((500 * 400) * 300) * 200
((200 * 500) * 300) * 400
400 * (200 * (300 * 500))
```

$ gcc -g -O0 mult-test.c -o mult-test
$ ./mult-test
-884901888
$
Let's look at a different program

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

int main() {
    float a = 3.14;
    float b = 1e20;

    printf("(3.14 + 1e20) - 1e20 = %f\n", (a + b) - b);
    printf("3.14 + (1e20 - 1e20) = %f\n", a + (b - b));

    return 0;
}
```

$ gcc -g -Og -std=gnu99 float-mult-test.c -o float-mult-test

$ ./float-mult-test.c
(3.14 + 1e20) - 1e20 = 0.000000
3.14 + (1e20 - 1e20) = 3.140000
$
Let's look at a different program

```
$ gcc -g -O0 mult-test.c -o mult-test
$ ./mult-test
-884901888
$

$ gcc -g -Og -std=gnu99 float-mult-test.c -o float-mult-test
$ ./float-mult-test.c
(3.14 + 1e20) - 1e20 = 0.000000
3.14 + (1e20 - 1e20) = 3.140000
$`

Both C and C++ have specific representations of numbers that allow for these kinds of bugs.
In the last few minutes of class, you get to ask me 20 questions
• The questions can be about the class, about me, about computing, about philosophy, etc.
• I do reserve the right to not answer something too personal. :)}
References and Advanced Reading

• References:
  • Tiobe Programming Index: https://www.tiobe.com/tiobe-index/
  • The C Language: https://en.wikipedia.org/wiki/C_(programming_language)
  • Kernighan and Ritchie (K&R) C: https://www.youtube.com/watch?v=de2Hsvxaf8M
  • C Standard Library: http://www.cplusplus.com/reference/clibrary/

• Advanced Reading:
  • After All These Years, the World is Still Powered by C Programming
  • Is C Still Relevant in the 21st Century?
  • Why Every Programmer Should Learn C