Today's Topics

• Logistics
  • Final Exam Friday, 8:30am. We will have some power strips, but make sure you have a full battery on your laptop!

• Where we have been
• Where you are going
• How to Prepare for CS107
Where We Have Been

CS 106B
Where We Have Been: Game of Life
Where We Have Been: Fractals

Order-3

... Order-6

Order-5 tree fractal
Where We Have Been: Backtracking

I crushed you, puny human!

**Human**: 5

- foil
- room
- tom
- roony

**Computer**: 17

- star
- fruit
- trim
- room
- mol
- iglu
- meerkat
- limy

```
FYCL
IOMG
ORIL
HJHU
```
<table>
<thead>
<tr>
<th>Type</th>
<th>Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero nodes</td>
<td>front /</td>
</tr>
<tr>
<td>one node</td>
<td>front --&gt; [?</td>
</tr>
<tr>
<td></td>
<td>+---+---+</td>
</tr>
<tr>
<td>N nodes</td>
<td>front --&gt; [?</td>
</tr>
<tr>
<td></td>
<td>+---+---+</td>
</tr>
<tr>
<td></td>
<td>+---+---+</td>
</tr>
<tr>
<td></td>
<td>+---+---+</td>
</tr>
</tbody>
</table>

![Binary Heap Diagram](image.png)
Where We Have Been: Binary Trees

Huffman!
Where We Have Been: Graphs
Where We Have Been: Sorting

So many ways to sort things!
We learned:
• Insertion sort
• Selection Sort
• Merge Sort
• Quicksort

Other sorts:
• Shell Sort
• Heap Sort
• Tim Sort
• Radix Sort
• Bubble Sort

Trying to achieve $O(n \log n)$
Where We Have Been: C++

For many of you, a new language!

Highlights:
• Object oriented language with classes
• Fast (except our wonky graphics...)
• Extremely robust (too much sometimes)
• Widely used in industry and for making games

Differences you probably saw from other languages:
• Mutable strings
• Input / Output streams
• Operator overloading
• Pointers
• Memory Management: new, delete
• Inheritance and Polymorphism
The Importance of Data Structures

Why Data Structures are Important

One reason we care about data structures is, quite simply, **time**. Let’s say we have a program that does the following (and times the results):

- Creates four “list-like” containers for data.
- Adds 100,000 elements to each container – specifically, the even integers between 0 and 198,998 (sound familiar?).
- Searches for 100,000 elements (all integers 0-100,000)
- Attempts to delete 100,000 elements (integers from 0-100,000)

What are the results?
# The Importance of Data Structures

<table>
<thead>
<tr>
<th>Structure</th>
<th>Overall(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted Vector</td>
<td></td>
</tr>
<tr>
<td>Linked List</td>
<td></td>
</tr>
<tr>
<td>Hash Table</td>
<td></td>
</tr>
<tr>
<td>Binary Tree</td>
<td></td>
</tr>
<tr>
<td>Sorted Vector</td>
<td></td>
</tr>
</tbody>
</table>
The Importance of Data Structures

Results:

<table>
<thead>
<tr>
<th>Structure</th>
<th>Overall(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted Vector</td>
<td>15.057</td>
</tr>
<tr>
<td>Linked List</td>
<td>92.202</td>
</tr>
<tr>
<td>Hash Table</td>
<td>0.145</td>
</tr>
<tr>
<td>Binary Tree</td>
<td>0.164</td>
</tr>
<tr>
<td>Sorted Vector</td>
<td>1.563</td>
</tr>
</tbody>
</table>

Processor: 2.8GHz Intel Core i7 (Macbook Pro)
Compiler: clang++

A factor of 103x
A factor of 636x!

Note: In general, for this test, we used optimized library data structures (from the "standard template library") where appropriate. The Stanford libraries are not optimized.

Overall, the Hash Table "won" — but (as we shall see!) while this is generally a great data structure, there are trade-offs to using it.
## Full Results

<table>
<thead>
<tr>
<th>Structure</th>
<th>Overall(s)</th>
<th>Insert(s)</th>
<th>Search(s)</th>
<th>Delete(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted Vector</td>
<td>15.057</td>
<td>0.007</td>
<td>10.307</td>
<td>4.740</td>
</tr>
<tr>
<td>Linked List</td>
<td>92.202</td>
<td>0.025</td>
<td>46.436</td>
<td>45.729</td>
</tr>
<tr>
<td>Hash Table</td>
<td>0.145</td>
<td>0.135</td>
<td>0.002</td>
<td>0.008</td>
</tr>
<tr>
<td>Binary Tree</td>
<td>0.164</td>
<td>0.133</td>
<td>0.010</td>
<td>0.0208</td>
</tr>
<tr>
<td>Sorted Vector</td>
<td>1.563</td>
<td>0.024</td>
<td>0.006</td>
<td>1.534</td>
</tr>
</tbody>
</table>

Why are there such discrepancies??

**Bottom line:**
- Some structures carry more *information* simply because of their design.
- Manipulating structures takes time
Where to from here?
CS Core

- CS 106B: Programming Abstractions
- CS 107: Computer Organization and Systems
- CS 110: Principles of Computer Systems
- CS 103: Mathematical Foundations of Computing
- CS 109: Introduction to Probability for Computer Scientists
- CS 161: Design and Analysis of Algorithms
CS Core

- **CS 106B**: Programming Abstractions
- **CS 107**: Computer Organization and Systems
- **CS 110**: Principles of Computer Systems
- **CS 103**: Mathematical Foundations of Computing
- **CS 109**: Introduction to Probability for Computer Scientists
- **CS 161**: Design and Analysis of Algorithms
CS Core

- CS 106B: Programming Abstractions
- CS 107: Computer Organization and Systems
- CS 110: Principles of Computer Systems
- CS 103: Mathematical Foundations of Computing
- CS 109: Introduction to Probability for Computer Scientists
- CS 161: Design and Analysis of Algorithms
Can computers solve all problems?

Why are some problems harder than others?

We can do find in an unsorted array in $O(N)$, and we can sort an unsorted array in $O(N\log N)$. Is sorting just inherently a harder problem, or are there better $O(N)$ sorting algorithm yet to be discovered?

How can we be certain about this?
How do we encode text, numbers, programs, etc. using just 0s and 1s?

Where does memory come from? How is it managed?

How do compilers, debuggers, etc. work?
CS107 is *not* a litmus test for whether you can be a computer scientist.

- You can be a *great* computer scientist without enjoying low-level systems programming.

CS107 is *not* indicative of what programming is “really like.”

- CS107 does a lot of low-level programming. You don’t have to do low-level programming to be a good computer scientist.
How to Prepare for cs107

- You are prepared! cs106b is a great preparation!
- However...you may want to learn a few things before class
  - Terminal

```bash
/chrisgregg-imac:/Downloads chrisgregg 2017-06-07 11:16:09
$ ls *.cpp
atoi_ex.cpp knights.cpp test.cpp
bufferTest.cpp ngrams.cpp test2.cpp
good-trailblazer.cpp permute.cpp testVec.cpp
hello.cpp r_williams.cpp testptr.cpp
hello2.cpp st100.cpp traveling_salesman.cpp
helloOrig.cpp st100Orig.cpp
/chrisgregg-imac:/Downloads chrisgregg 2017-06-07 11:16:11
$ whoami
chrisgregg
/chrisgregg-imac:/Downloads chrisgregg 2017-06-07 11:16:13
$ uptime
11:16 up 13 days, 2:27, 11 users, load averages: 1.60 1.75 1.73
/chrisgregg-imac:/Downloads chrisgregg 2017-06-07 11:16:17
$ 

/afs/ir/users/c/g/cgregg/cowsay/bin/cowsay Moooo
ls /afs/ir/users/c/g/cgregg/cowsay/share/cows
/afs/ir/users/c/g/cgregg/cowsay/bin/cowsay -f stegosaurus rowr
• How to log into Myth: Open Terminal (on Mac / Linux), use MobaXTerm on Windows. Then:

```
chrisgregg@chrisgregg-xubuntu:~$ ssh cgregg@myth.stanford.edu
The authenticity of host 'myth.stanford.edu (171.64.15.185)' can't be established.
ECDSA key fingerprint is SHA256:Tu94V/MyK4eRL3qckCSfv1UtpKHLjW2BFZpgwrols8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'myth.stanford.edu,171.64.15.185' (ECDSA) to the list of known hosts.
cgregg@myth.stanford.edu's password:
=================================================================
myth4.stanford.edu
Ubuntu 14.04 (Linux 3.13.0-106-generic amd64)
2 x Intel(R) Core(TM)2 Duo CPU E8500 @ 3.16GHz, 7.73 GB RAM, 3.81 GB swap
=================================================================
-> For help with your SUNetID, AFS, or class dirs call 725-HELP, or visit http://helpsu.stanford.edu
-> For problems with hardware, local software, or facilities email action@soe.stanford.edu
-> The myths are not for CPU-intensive workloads. For alternative computers see http://farmshare.stanford.edu
-> To logout of the console, click in the background then hit ctrl-alt-del.
```
How to Prepare for cs107

• How to log into Myth: Open Terminal (on Mac / Linux), use MobaXTerm on Windows. Then:

```bash
chrisgregg@chrisgregg-xubuntu:~$ ssh cgregg@myth.stanford.edu
The authenticity of host 'myth.stanford.edu (171.64.15.185)' can't be established.
ECDSA key fingerprint is SHA256:Tu94V/MyK4eRL3qckCSfv1UtpKHLjW2BFZzpwrols8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'myth.stanford.edu,171.64.15.185' (ECDSA) to the list of known hosts.
chrisgregg@myth.stanford.edu's password:

myth4.stanford.edu
Ubuntu 14.04 (Linux 3.13.0-106-generic amd64)
2 x Intel(R) Core(TM)2 Duo CPU E8500 @ 3.16GHz, 7.73 GB RAM, 3.81 GB swap

--> For help with your SUNetID, AFS, or class dirs call 725-HELP, or visit http://helpsu.stanford.edu
--> For problems with hardware, local software, or facilities email action@soe.stanford.edu
--> The myths are not for CPU-intensive workloads. For alternative computers see http://farmshare.stanford.edu
--> To logout of the console, click in the background then hit ctrl-alt-del.

chrisgregg@myth4:~$ "
```

Only need to answer this once.
How to Prepare for cs107

• How to log into Myth: Open Terminal (on Mac / Linux), use MobaXTerm on Windows. Then:

```
$ ssh cgregg@myth.stanford.edu
The authenticity of host 'myth.stanford.edu (171.64.15.185)' can't be established.
ECSA key fingerprint is SHA256:Tu94V/MyK4eRL3qckCSfv1UtpKHLjW2BFZpgwrols8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'myth.stanford.edu,171.64.15.185' (ECSA) to the list of
known hosts.

Password won't show up when you type it.
```
How to Prepare for cs107

• How to log into Myth: Open Terminal (on Mac / Linux), use MobaXTerm on Windows. Then:

```
ls command "lists files"
```
How to Prepare for cs107

- How to log into Myth: Open Terminal (on Mac / Linux), use MobaXTerm on Windows. Then:
  
  ```bash
  whoami
cgregg
cgregg@myth4:~$ whoami
  axyl1 pts/19 2017-06-07 04:39 (c-73-70-10-134.hsd1.ca.comcast.net)
  rcai8 pts/20 2017-06-07 07:48 (dn51q836.sunet)
  cgregg pts/24 2017-06-07 12:03 (chrisgregg-xubuntu.stanford.edu)
  hungn2 pts/26 2017-06-07 08:54 (eluicipi.stanford.edu)
  song22l pts/28 2017-06-07 10:26 (dn51t421.sunet)
  psprads pts/30 2017-06-07 10:46 (dn0a632a79.sunet)
  bzhou2 pts/31 2017-06-07 10:51 (76-218-107-95.lightspeed.sntcca.sbcglobal)
  l.net)  
zibo pts/32 2017-06-07 02:25 (rescomp-16-336632.stanford.edu)
  bzhou2 pts/33 2017-06-07 10:51 (76-218-107-95.lightspeed.sntcca.sbcglobal)
  l.net)
bzhou2 pts/36 2017-06-07 10:52 (76-218-107-95.lightspeed.sntcca.sbcglobal)
  l.net)
zhenglyu pts/35 2017-06-07 11:15 (c-73-170-32-32.hsd1.ca.comcast.net)
  hungn2 pts/12 2017-06-06 06:02 (eluicipi.stanford.edu)
  abaghel pts/40 2017-06-07 11:43 (dn800c1309.stanford.edu)
  rcai8 pts/7 2017-06-07 07:48 (dn51q836.sunet)
  shemmati pts/37 2017-06-07 11:28 (68.65.174.192)
  cwalker4 pts/38 2017-06-07 11:44 (dn51vc2d.sunet)
  abaghel pts/39 2017-06-07 11:43 (dn800c1309.stanford.edu)
cgregg@myth4:~$ whoami
  ```

Some other commands
How to Prepare for cs107

- Your first C program!

```bash
$ cat > test.c
#include <stdlib.h>
#include <stdio.h>
int main() {
    printf("Hello, World!\n");
    return 0;
}
$ make
$ ./test
Hello, World!
```

But what if you want an actual editor...

Type "ctrl-D" to exit

`make` compiles your program

`./test` runs your program
How to Prepare for cs107

• Lots of choices:
  • vi / vim
  • emacs
  • nano
  • Sublime Text from your own computer (slower, must transfer files)
How to Prepare for cs107

Editor war

From Wikipedia, the free encyclopedia

For a type of conflict between Wikipedia editors, see Wikipedia:Edit war

Editor war is the common name for the rivalry between users of the Emacs and vi (Vim) text editors. The rivalry has become a lasting part of hacker culture and the free software community.

The Emacs vs vi debate was one of the original "holy wars" conducted on Usenet groups,[1] with many flame wars fought between those insisting that their editor of choice is the paragon of editing perfection, and insulting the other, since at least 1985.[2] Related battles have been fought over operating systems, programming languages, version control systems, and even source code indent style.[3][4][5] Notably, unlike other wars (ie, UNIX vs ITS vs VMS, C vs Pascal vs Fortran), the editor war has yet to be resolved with a clear winner, and the hacker community remains split roughly 50/50.[6]

### Contents

1 Differences between vi and Emacs
   1.1 Benefits of Emacs
   1.2 Benefits of vi-like editors
2 Humor
3 Today
4 See also
5 Notes
6 References
7 External links

Differences between vi and Emacs [edit]

The most important differences between vi and Emacs are presented in the following table:
How to Prepare for cs107

• Vim Adventures: https://vim-adventures.com/
How to Prepare for cs107

- You might want to learn more about pointers, and pointers-to-pointers... https://www.tutorialspoint.com/cprogramming/c_pointer_to_pointer.htm

```c
#include <stdio.h>

int main () {
    int var;
    int *ptr;
    int **pptr;

    var = 3000;

    /* take the address of var */
    ptr = &var;

    /* take the address of ptr using address of operator & */
    pptr = &ptr;

    /* take the value using pptr */
    printf("Value of var = %d\n", var);
    printf("Value available at *ptr = %d\n", *ptr);
    printf("Value available at **pptr = %d\n", **pptr);

    return 0;
}
```
• What is this "printf()" function??
• It is the "cout" for C

Let's talk about some C differences...
How to Prepare for cs107

Going from C++ to C: “C has the speed and efficiency of assembly language combined with readability of assembly language.”

- You will find C to be similar in feel to C++ (C++ is based on C, after all)
- Things that are the same:
  - built-in data types: int, char, float, double, long, unsigned int
  - array indexing (using brackets, e.g., v[4])
  - function declarations
  - modularity between .h and .c files
  - flow: while, if, for, etc.
How to Prepare for cs107

Going from C++ to C

• Things that are different:
  • There isn’t a “string” type. Strings are null terminated arrays of chars.
  • There are no classes, no objects, and no constructors/destructors.
  • Memory management is more involved than in C++
    • To allocate memory, C uses “malloc()” (so can C++, but normally we use new())
C strings: null terminated char arrays

• There is no string class in C. Let me repeat: there is no string class in C.
• Strings are simply arrays of characters, with the final character of the array set aside for the NULL character.
• You must be careful to avoid buffer overflows when dealing with char strings.
How to Prepare for cs107

Going from C++ to C

• Things that are the same but that you might not have learned about yet:
  • There are entities called “void pointers” which can point to any type
  • You often need to cast variables to different types
  • “Pointer arithmetic” is often used instead of bracket notation
How to Prepare for cs107

Going from C++ to C

- `malloc()` and `free()`, and `sizeof()`: used for memory management instead of new and delete

- `malloc()` is used to reserve memory from the heap. You must pass it the size in bytes of the amount of memory you want. How do you know the size in bytes? You use `sizeof()`:

- `sizeof(int)` returns the size of an integer. On our machines, this is 4 bytes, or 32-bits.
Going from C++ to C

• `malloc()` returns a pointer to memory, e.g.,

```c
/* an array of 10 ints from the heap */
int *int_ptr;
int_ptr = malloc(sizeof(int)*10);
```

• `malloc()` returns `NULL` if the system runs out of memory.
How to Prepare for cs107

Going from C++ to C

• `free()` simply frees the previously allocated memory:

```c
/* an array of 10 ints from the heap */
int *int_ptr;
int_ptr = malloc(sizeof(int)*10);
free(int_ptr);
```
Going from C++ to C

- Pointer arithmetic is really important in C, and you have to understand it.

```c
int main() {
    /* an array of 10 ints */
    int *int_ptr;
    int_ptr = (int *)malloc(sizeof(int)*10);

    int i;
    for(i=0;i<10;i++) {
        *(int_ptr+i)=i*10;
    }
    for(i=0;i<5;i++) {
        printf("%d\n",*(int_ptr+i*2));
    }
    free(int_ptr);
    return 0;
}
```

What will this print out?
Oops — we’d better take a detour into printf() land.
**printf()** — *the way to print to stdout in C*

- **printf()** writes a formatted string to the standard output. The format can include *format specifiers* that begin with % and additional arguments are inserted into the string, replacing their formatters.

- Example: print the int variable, d:
  ```c
  int i = 22;
  printf("This is i: %d\n",i);
  
  output:
  This is i: 22
  ```
# How to Prepare for cs107

**printf()** — *the way to print to stdout in C*  
- lots of specifiers

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Description</th>
<th>Example</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>Unsigned octal</td>
<td>610</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>Unsigned hexadecimal integer</td>
<td>7fa</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>Unsigned hexadecimal integer (uppercase)</td>
<td>7FA</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>Decimal floating point, lowercase</td>
<td>392.65</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>Decimal floating point, uppercase</td>
<td>392.65</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>Scientific notation (mantissa/exponent), lowercase</td>
<td>3.9265e+2</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Scientific notation (mantissa/exponent), uppercase</td>
<td>3.9265E+2</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>Use the shortest representation: %e or %f</td>
<td>392.65</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>Use the shortest representation: %E or %F</td>
<td>392.65</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>Hexadecimal floating point, lowercase</td>
<td>-0xc.90fe</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>Hexadecimal floating point, uppercase</td>
<td>-0XC.90FE</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>Character</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>String of characters</td>
<td>sample</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>Pointer address</td>
<td>b8000000</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>A % followed by another % character will write a single % to the stream.</td>
<td>%</td>
<td></td>
</tr>
</tbody>
</table>
#include <stdio.h>

int main()
{
    printf("Characters: %c %c \n", 'a', 65);  // Characters: a A
    printf("Decimals: %d %ld \n", 1977, 650000L);  // Decimals: 1977 650000
    printf("Preceding with blanks: %10d \n", 1977);  // Preceding with blanks: 1977
    printf("Preceding with zeros: %010d \n", 1977);  // Preceding with zeros: 0000001977
    printf("Radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);  // Radices: 100 64 144 0x64 0144
    printf("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);  // floats: 3.14 +3e+000 3.141600E+000
    printf("Width trick: %*d \n", 5, 10);  // Width trick: 10
    printf("%s \n", "A string");  // A string
    return 0;
}
How to Prepare for cs107

Going from C++ to C (back to our example)

- **Pointer arithmetic is really important in C, and you have to understand it.**

```c
int main() {
    /* an array of 10 ints */
    int *int_ptr;
    int_ptr = (int *)malloc(sizeof(int)*10);

    int i;
    for(i=0;i<10;i++) {
        *(int_ptr+i)=i*10;
    }
    for(i=0;i<5;i++) {
        printf("%d\n",*(int_ptr+i*2));
    }
    free(int_ptr);
    return 0;
}
```

What will this print out?

```
0
20
40
60
80
```
Foundations of probability

Narrative driven

Intro to Machine Learning

CS109
Computer Science Affects Every Field
Courses Aren't Necessary!

Things to learn on your own:

- **A new language. Good candidates?**
  - Python: used everywhere, easy to learn, easy to write quick programs. Best online resource: https://www.reddit.com/r/Python/ (see right side-bar)
  - Haskell: a "functional" programming language. Best online resource: Learn You a Haskell for Great Good

- **iOS / Android Programming: Why not learn how to program your phone?**
  - Best iOS resource: https://www.raywenderlich.com
  - Good tutorials link: http://equallysimple.com/best-android-development-video-tutorials/
  - Want to code for all phones (and the web, and the desktop?) Check out React Native: https://facebook.github.io/react-native/

- **Hardware: Raspberry Pi, Arduino, FPGA: Hardware is awesome!**
  - Raspberry Pi resources: https://www.reddit.com/r/raspberry_pi/
  - Arduino Resources: https://www.reddit.com/r/arduino/

- **GPU and Multicore Programming: hard, but your code can fly**
  - Your GPU might have hundreds of individual processors. Resources: http://gpgpu.org
Python

I FIGURED OUT HOW TO TURN ON MY MICROWAVE

USING PYTHON

STUDYWEBDEVELOPMENT.COM
It is the Time and Place for CS
Thank You
Congrats (in advance)
References and Advanced Reading

**References:**
- [https://explorecourses.stanford.edu/search;jsessionid=1xnekjf0f94bc1gql4to9qrgzp?view=catalog&academicYear=&page=0&q=CS&filter-departmentcode-CS=on&filter-coursestatus-Active=on&filter-term-Autumn=on](https://explorecourses.stanford.edu/search;jsessionid=1xnekjf0f94bc1gql4to9qrgzp?view=catalog&academicYear=&page=0&q=CS&filter-departmentcode-CS=on&filter-coursestatus-Active=on&filter-term-Autumn=on)
- [https://www.theguardian.com/technology/self-driving-cars](https://www.theguardian.com/technology/self-driving-cars)