Lecture 19: Finals Review
Plan For The Rest Of The Term (1/2)

• 3D: assigned over break
  – Major project, worth 8%
  – Grade options for partial credit

• No projects after 3D
Plan For The Rest Of The Term (2/2)

• Lecture 18: November 27, more data structures & sorting
• Lecture 19: November 29, finals review & Hank’s unsolicited advice
• Final: Monday Dec 3rd, 8am, in this room
Oversleeping for the final

• From class syllabus:
  • “If you miss the Final exam period, you may take the Final afterwards for half-credit. Exceptions for full credit will be granted in appropriate situations. These situations include medical emergencies, etc, and do not include oversleeping, forgetting the day of the final, etc.”
Revising syllabus

• If you run out of late passes, then you may continue to earn half credit on any project up until we take the Final. Once we get to the Final (Monday December 3rd, 2018 at 8am), I will no longer accept homeworks.

• If you run out of late passes, then you may continue to earn half credit on any project up until we take the Final. Once we get to Wednesday December 5th (meaning: Thursday December 6th, 6am) I will no longer accept homeworks.

  – Note: if you misread this statement and think you have until Thursday, there will be no sympathy (don’t ask).
Lab This Week

• Lab this week is important
• Looking at some usage of memory
• The sort of concepts I want you to know for the final
Extra OH this week

• When?
YouTube Video Ideas?

• ???
3D: questions
3D – skip to slide 19 unless following slides are needed for questions
Maps

- A map stores tuples: (key, value)
- You store both key and value
- You retrieve with just the key (and it gives the value)
- If the key is not in the map, then some appropriate value is returned
Python Equivalent: Dictionaries

```python
>>> dict = {
...    "951301000": "Hank Childs",
...    "951302000": "Zayd Hamoudeh",
...    "951303000": "Priya Kudva",
...    "951304000": "Viet Lai"
... }

>>> dict["951301000"]
'Hank Childs'
>>> dict["951301001"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: '951301001'
```
Functions for the Map Data Structure

• Store
  – Textbook: put

• Fetch
  – Textbook: get

• Other noteworthy functions from textbook:
  – containsKey
  – putUnique
  – isEmpty
  – remove
How To Implement a Map

• Idea #1: Linked Lists
• Idea #2: Hash Tables
How To Implement a Map: Linked Lists

- Make a tuple type (key, value)
- Make a linked list of that tuple type
- Store: insert the tuple into the list
- Fetch: iterate over the linked list, comparing the desired key with the key element of each tuple in the linked list

```c
struct ID_Name_Tuple
{
    int UO_id;  /* NOTE: Python example used string, not int */
    char *student_name;
};
```
How To Implement a Map: Hash Tables

• Make a hash function to convert keys to indices
• Decide on hash table details (open addressing, chaining)
• Store: hash key, insert tuple into hash table
• Fetch: hash desired key, compare with key in tuple (repeat as appropriate)
3D

• Assigned tomorrow, due Friday November 30th
  – Note: full credit up to Weds December 5th
  – 0% credit as of 6am Thurs Dec 6th
  – (As before, no sympathy for misreading this)
3D

• Implement two types of maps
  – Linked list
  – Hash table

• Analyze performance and write very short report (one paragraph)
New material
Your Grade

• 70% projects
• 30% final
• If you have 70% right now, you have already passed the class.
  – Although if you bomb the final, I may meet with you and ask you to code
Late Passes

• You have 3 "late passes."

• Late passes allow you to turn in your project (specifically a sub-project, i.e., project 1A) two days after the due date for full credit.

• You may also use two late passes on one assignment and get a four day extension.
  – For example, you submit a project that was due on a Wednesday on Friday (i.e., two days later) and get full credit if you use one late pass.
Late Passes

• I will ask you how to apply your late passes on the final.

• Work it out ahead of time!
Late Pass Decisions

• There will be a spot on the final

Name: ________________________________

Answer the questions in the spaces provided below the questions. If you run out of room for an answer, continue on the back of the page.

Please write your name on every page!

This is a closed-book test. No notes, no calculators, no computers.

Apply Late Pass #1 To:
Apply Late Pass #2 To:

List Projects That You Feel Have Been Improperly Graded:
Notes

• No notes
• Closed book
Will the Final Be Hard?

• I used to say “no”...
  but now I say “yes”...

I want to understand what you have learned about C/Unix/data structures

Everything in lecture is fair game
But: almost all parts of lecture are reinforced in the homeworks
Is This Review Exhaustive?

• No.

• Things not in the review are fair game
Directories are hierarchical

• Directories can be placed within other directories

• “/” -- The root directory
  – Note “/”, where Windows uses “\”

• “/dir1/dir2/file1”
  – What does this mean?

File file1 is contained in directory dir2, which is contained in directory dir1, which is in the root directory
Home directory

- Unix supports multiple users
- Each user has their own directory that they control
- Location varies over Unix implementation, but typically something like “/home/username”
- Stored in environment variables

```
fawcett:~ childs$ echo $HOME
/Users/childs
```
File manipulation

New commands: mkdir, cd, touch, ls, rmdir, rm
There are 9 file permission attributes

- Can user read?
- Can user write?
- Can user execute?
- Can group read?
- Can group write?
- Can group execute?
- Can other read?
- Can other write?
- Can other execute?

A bunch of bits ... we could represent this with binary

User = “owner”
Other = “not owner, not group”
Translating R/W/E permissions to binary

<table>
<thead>
<tr>
<th>#</th>
<th>Permission</th>
<th>rwx</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>full</td>
<td>111</td>
</tr>
<tr>
<td>6</td>
<td>read and write</td>
<td>110</td>
</tr>
<tr>
<td>5</td>
<td>read and execute</td>
<td>101</td>
</tr>
<tr>
<td>4</td>
<td>read only</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>write and execute</td>
<td>011</td>
</tr>
<tr>
<td>2</td>
<td>write only</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td>execute only</td>
<td>001</td>
</tr>
<tr>
<td>0</td>
<td>none</td>
<td>000</td>
</tr>
</tbody>
</table>

Which of these modes make sense? Which don’t?

We can have separate values (0-7) for user, group, and other.
Unix command: chmod

• chmod: change file mode

• chmod 750 <filename>
  – User gets 7 (rwx)
  – Group gets 5 (rx)
  – Other gets 0 (no access)

Lots of options to chmod (usage shown here is most common)
Directories are hierarchical

• Directories can be placed within other directories

• “/” -- The root directory
  – Note “/”, where Windows uses “\”

• “/dir1/dir2/file1”
  – What does this mean?

File file1 is contained in directory dir2, which is contained in directory dir1, which is in the root directory
Home directory

- Unix supports multiple users
- Each user has their own directory that they control
- Location varies over Unix implementation, but typically something like “/home/username”
- Stored in environment variables

```bash
fawcett:~ childs$ echo $HOME
/Users/childs
```
File manipulation

New commands: mkdir, cd, touch, ls, rmdir, rm
There are 9 file permission attributes

- Can user read?
- Can user write?
- Can user execute?
- Can group read?
- Can group write?
- Can group execute?
- Can other read?
- Can other write?
- Can other execute?

A bunch of bits ... we could represent this with binary

User = “owner”
Other = “not owner, not group”
Translating R/W/E permissions to binary

<table>
<thead>
<tr>
<th>#</th>
<th>Permission</th>
<th>rwx</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>full</td>
<td>111</td>
</tr>
<tr>
<td>6</td>
<td>read and write</td>
<td>110</td>
</tr>
<tr>
<td>5</td>
<td>read and execute</td>
<td>101</td>
</tr>
<tr>
<td>4</td>
<td>read only</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>write and execute</td>
<td>011</td>
</tr>
<tr>
<td>2</td>
<td>write only</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td>execute only</td>
<td>001</td>
</tr>
<tr>
<td>0</td>
<td>none</td>
<td>000</td>
</tr>
</tbody>
</table>

Which of these modes make sense? Which don’t?

We can have separate values (0-7) for user, group, and other.
Unix command: chmod

- chmod: change file mode

- chmod 750 <filename>
  - User gets 7 (rwx)
  - Group gets 5 (rx)
  - Other gets 0 (no access)

Lots of options to chmod
(usage shown here is most common)
Character strings example

```c
128-223-223-72-wireless:330 hank$ cat string.c
#include <stdio.h>

int main()
{
    char str[12] = "hello world";
    char *str2 = str+6;

    printf("str is \"%s\" and str2 is \"%s\"\n", str, str2);

    str[5] = '\0';

    printf("Now str is \"%s\" and str2 is \"%s\"\n", str, str2);
}
128-223-223-72-wireless:330 hank$ gcc string.c
128-223-223-72-wireless:330 hank$ ./a.out
str is "hello world" and str2 is "world"
Now str is "hello" and str2 is "world"
```
Enums

• Enums make your own type
  – Type is “list of key words”

• Enums are useful for code clarity
  – Always possible to do the same thing with integers

• Be careful with enums
  – ... you can “contaminate” a bunch of useful words
enum example

C keyword "enum" – means enum definition is coming

```c
enum StudentType {
    HighSchool, Freshman, Sophomore, Junior, Senior, GradStudent
};
```

This enum contains 6 different student types

semi-colon!!!
Structs: a complex data type

• Construct that defines a group of variables
  – Variables must be grouped together in contiguous memory

• Also makes accessing variables easier ... they are all part of the same grouping (the struct)
C keyword "struct" – means struct definition is coming

```c
struct Ray
{
    double origin[3];
    double direction[3];
};
```

This struct contains 6 doubles, meaning it is 48 bytes

Declaring an instance

```
int main()
{
    struct Ray r;
    r.origin[0] = 0;
    r.origin[1] = 0;
    r.origin[2] = 0;
    r.direction[0] = 1;
    r.direction[1] = 0;
    r.direction[2] = 0;
}
```

"." accesses data members for a struct
Function Pointer Example

128-223-223-72-wireless:cli hank$ cat function_ptr.c
#include <stdio.h>
int doubler(int x) { return 2*x; }
int tripler(int x) { return 3*x; }
int main()
{
    int (*multiplier)(int);
    multiplier = doubler;
    printf("Multiplier of 3 = %d\n", multiplier(3));
    multiplier = tripler;
    printf("Multiplier of 3 = %d\n", multiplier(3));
}

128-223-223-72-wireless:cli hank$ gcc function_ptr.c
128-223-223-72-wireless:cli hank$ ./a.out
Multiplier of 3 = 6
Multiplier of 3 = 9
What is an array?

• Block of contiguous memory
• If elements each have size N bytes and there are M elements, then N*M contiguous bytes
• Let A be address of the beginning of the array
• Then A[0] is at “A”
• And A[1] is at “A+N”
• and so on...
Dynamic memory works differently

• You allocate it, it stays around until you de-allocate it or the program ends
• Important: you need a way to keep track of memory
  – If not, the memory will be “leaked.”
• So we need a way of managing dynamic memory.
• The concept for doing this in C is **POINTERS**
Pointers

• Pointer: points to memory location
  – Denoted with ‘*’
  – Example: “int *p”
    • pointer to an integer
  – You need pointers to get to dynamic memory

• Address of: gets the address of memory
  – Operator: ‘&’
  – Example:
    
    int x;
    int *y = &x;  ← this example is pointing to an automatic variable, not a dynamic variable
free/malloc example

```
#include <stdlib.h>
int main()
{
    /* allocates memory */
    int *ptr = malloc(2*sizeof(int));

    /* deallocates memory */
    free(ptr);
}
```

Enables compiler to see functions that aren’t in this file. More on this next week.

sizeof is a built in function in C. It returns the number of bytes for a type (4 bytes for int).
don’t have to say how many bytes to free … the OS knows
character strings

• A character “string” is:
  – an array of type “char”
  – that is terminated by the NULL character

• The C library has multiple functions for handling strings
```c
#include <stdio.h>

int main()
{
    char H[12] = "hello world";
    printf("H is %s\n", H);
}
```

```
H is hello world!
```
Specifics about the final

• Will be:
  – A question where you write a script in Unix
    • (Simple commands, not like 1C)
  – A question like 2D
  – One or more questions where I write code and you tell me what the program does
  – One or more questions where you write code
  – One or more questions where I infer your knowledge about data structures. Perhaps a simple implementation.
  – A true/false (or multiple choice) section
  – Almost certain to appear:
    • Character strings and their conventions (char * / ‘\0’)
    • Structs
    • Malloc / free / understanding of memory / pointers
What data structures do I expect you to know?

- Stack
- Queue
- Array
- Linked list / doubly linked list
- Deques
- Hash table
- Map
- + a little about sorting
Hank’s Unsolicited Advice
Internships

• Great opportunity
  – Learn what you like and don’t like
  – Networking
  – Helps bolster your resume
Things you should be doing during your undergrad

• Personal projects?
• Good grades
• Investing in yourself, esp. coding
  – It is very likely you will have an interview where you are asked to code. Practice early and often.
Grad School?

• MS: possibly better job than BS
  – Possibly not

• PhD: opens the door to research (being at the leading edge of computing)
  – Closes other doors
Big Analogy #1: GREs
Big Analogy #2:
Training for a Marathon
This class

• This class was challenging
• Many of you stayed on the “Oregon Trail”
  – You should be very proud ... you have done something very hard
• If you keep “running,” the “marathon” is well within your reach
Bonus Slides
How a Computer is **Not** Like Monopoly

- **Actions during cycle/turn:**
  - Monopoly: roll, buy, build, trade
  - Computer: other

- **Who’s turn?**
  - Monopoly: passes between players
  - Computer: always the computer’s turn

- **Time spent per cycle/turn:**
  - Monopoly: variable
  - Computer: fixed

- **Duration of turn:**
  - Monopoly: sometimes over a minute
  - Computer: much faster!!!
“First” computer: ENIAC

- Year: 1946
- Location: Pennsylvania
- Purpose: military
- Cost: $487K
  - ($6.9M today)
- Technology:
  - very different than today
  - ... but still the same
Vacuum Tubes

• Vacuum tubes:
  – Glass tubes with no gas
  – Used to control electron flow in early computers

• Occasionally, a bug would get stuck in the tube and cause the program to malfunction

• We no longer have vacuum tubes, but the term bug has remained with us...
An ENIAC Computation

- Used for military calculations:
  - A-bomb design
  - Missile delivery
- ENIAC could do ~5000 calculations in one minute
- In one case:
  - ENIAC did a calculation in 30 seconds
  - Human being took 20 hours
  - 2400x increase in speed

source: wikipedia
Hertz (Hz) = unit of measurement for how fast you do something

- 1 Hertz = do something once per second
- KHz = 1024 Hz
- MHz = 1024 KHz
- GHz = 1024 MHz

- The ENIAC machine ran at 5000 Hertz, or about 5 KHz.
  - Vocab term: “clock speed” → the number of cycles per second
    - the clock speed of the ENIAC was 5 KHz
Today’s Desktop Computers Are Fast!

- Most computers run at ~1-3 GHz
- i.e., operates billions of instructions each second

- This is about one million times faster than the ENIAC
  - ... and the ENIAC was 2400X faster than humans
  - (at least at tasks computers are good at)
What does a million-fold increase mean?

Distance: a 2” map of Oregon is 1:1,000,000 scale

Time: 1 second to 277 hours is 1:1,000,000 scale

Time: 1 minute to 694 days is 1:1,000,000 scale

Time: 1 hour to 114 years is 1:1,000,000 scale

Time: 1 day to 2738 years is 1:1,000,000 scale
1 million-fold increase!
How does this happen?

• Moore’s Law (old timer’s version)
  – Clock speed doubles every 18 months

• Moore’s Law (newer version but still for old timers)
  – Clock speed doubles every 24 months
Moore’s Law

• Moore’s Law (actual version)
  – Number of transistors doubles every 24 months
  – And clock speed is a reflection of number of transistors

• So what is a transistor?
  – Semiconductor device for amplifying or switching electronic signals/power
  – Fundamental building block of modern electronics
  – Replacement for vacuum tube
Moore’s Law – The number of transistors on integrated circuit chips (1971-2016)

Moore’s law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore’s law.

The data visualization is available at OurWorldInData.org. There you find more visualizations and research on this topic. Licensed under CC-BY-SA by the author Max Roser.
But actually...

CPU Speed Overclocked (MHz)

Source: maximumpc.com
The reason is power

• Desktop computer takes ~200W
  – There are multiple components that consume the power:
    • CPU
    • Monitor
    • Disk
    • Memory

• $200W \times 1 \text{ year} \rightarrow \sim$70
Relationship Between Power and Clock Speed

- Clock goes twice as fast $\rightarrow$ Power goes up by factor of 8
  - (Increase of $X$ in clock speed $\rightarrow$ Increase of $X^3$ in power)
- Clock speeds haven’t changed in 12 years
- What if they had doubled every 2 years?
- Then 64X faster
  - $\rightarrow$ 262144X more power (for the CPU)
  - $\rightarrow$ power bill now $\$18M$
New vocab term: "core"

What Changed?

• We are getting double the transistors every two years
• … but clock speed is the same
• … so what is changing?

**Choose Your Optimization Point**

<table>
<thead>
<tr>
<th></th>
<th>Integrated</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CORES</td>
<td>GHZ</td>
<td>MEMORY</td>
<td>FABRIC</td>
<td>DDR4</td>
<td>POWER²</td>
<td>RECOMMENDED CUSTOMER PRICING³</td>
</tr>
<tr>
<td>7290¹</td>
<td>72</td>
<td>16GB 7.2 GT/s</td>
<td>Yes</td>
<td>384GB 2400 MHz</td>
<td>245W</td>
<td>$6254</td>
</tr>
<tr>
<td>Best Performance/Node</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7250</td>
<td>68</td>
<td>16GB 7.2 GT/s</td>
<td>Yes</td>
<td>384GB 2400 MHz</td>
<td>215W</td>
<td>$4876</td>
</tr>
<tr>
<td>Best Performance/Watt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7230</td>
<td>64</td>
<td>16GB 7.2 GT/s</td>
<td>Yes</td>
<td>384GB 2400 MHz</td>
<td>215W</td>
<td>$3710</td>
</tr>
<tr>
<td>Best Memory Bandwidth/Core</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7210</td>
<td>64</td>
<td>16GB 6.4 GT/s</td>
<td>Yes</td>
<td>384GB 2133 MHz</td>
<td>215W</td>
<td>$2438</td>
</tr>
<tr>
<td>Best Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

¹Available beginning in September ²Plus 15W for integrated fabric
³Pricing shown is for parts without integrated fabric. Add additional $278 for integrated fabric versions of these parts. Integrated fabric parts available in October.

---

Intel Xeon Silver  🟧 Intel Xeon Gold  🟦 Intel Xeon Platinum

M - an optional SKU is available with support for up to 1.5TB memory per CPU socket
F - an optional SKU is available with integrated 100Gbps Intel Omni-Path fabric
How To Use Multiple Cores?

• Answer: parallel programming
  – Write computer programs that use all the cores
  – Ideally the coordination between the cores is minimal
Parallel Programming Concepts

• Usual goal:
  – if program takes N seconds to run with one core
  – then take N/2 seconds to run with two cores
  – and N/M seconds to run with M cores

Let’s consider an example outside of computers
Example: paint a house

• One person: 6 days (1 day = 10 hours)
• Two people: 3 days
• Three people: 2 days
• Six people: 1 day

• Sixty people: 1 hour?
• Six hundred people: 6 minutes?
Example: paint a house, plan #2

- One person: paint one house in 6 days
- Two people: paint two houses in 6 days
- Three people: paint three houses in 6 days
- One thousand people: paint 1000 houses in 6 days?

Parallel programming is hard, and smart people spend their whole careers figuring out how to make parallel programs be efficient.
GPUs: Graphical Processing Units (graphics cards)

• Historical:
  – Introduced to accelerate graphics (gaming!)
  – Boom with desktop PCs in late 90s onward
  – Mid-2000’s: people start hacking the interface to program a GPU to make things besides graphics
  – Late 2000’s: GPU makers jump on board and start encouraging folks to program GPUs directly

• GPGPU: General-purpose GPU programming
  – Mid 2010’s: GPUs used for *lots* of computing problems.
  – Machine learning workhorse!
Why Are GPUs So Good?

Market summary > NVIDIA Corporation
NASDAQ: NVDA - Mar 5, 7:59 PM EST

235.65 USD ↓ 0.89 (0.38%)
After-hours: 236.50 ↑ 0.36%

1 day | 5 day | 1 month | 3 month | 1 year | 5 year | max

12.82 Mar 8, 2013

This level of increase in computation is not just a quantitative change, it is a qualitative one too.