UNIX, C, AND DATA STRUCTURES

Lecture 3:
More Unix
What is Data?
Beginning on Memory

October 7, 2019  Hank Childs, University of Oregon
What should you be doing?

• Due today: 2A
• Assigned today: 1B
• Reading the stuff from lecture 2, if you have not done it already
  – Chapter 2.1, 2.2, 2.3, 2.4., 2.5, and 2.6.1
  – Chapter 4.1 (but not 4.1.2), 4.2-4.4
Plan for today

- Quick review of hello world in C
- Permissions
- Project 1B
- More Unix
- What is Data? (repeat)
- Beginning on Memory in C
Plan for today

- Quick review of hello world in C
- Permissions
- Project 1B
- More Unix
- What is Data? (repeat)
- Beginning on Memory in C
Hello World Example

```c
#include <stdio.h>
int main()
{
    printf("Hello world\n");
}
```

Hanks-iMac:Downloads hank$ gcc hello.c
.Hanks-iMac:Downloads hank$ ./a.out
Hello world
Notes on previous example

• Ignore for now: `#include <stdio.h>`
• ”main” is the first function called in C.
  – (Sort of. There are other things called before main to set up the program. But you won’t be mucking with those.)
• gcc is a C compiler. It is short for GNU Compiler Collection. GNU is an open source effort.
• The output is called “a.out”
• We have to add “./”, as it “./a.out”
  – “.” is the directory the shell is in. Therefore “./a.out” says run the program called a.out in the current directory.
More C: variable types

• When you declare variables, you have to declare their type.
  – We learned one type: int (integer)
  – There are more types (floating point, etc). We will discuss them later today.
More C: functions

- As you would expect, you can define your own functions and call them.

```c
#include <stdio.h>
int foo1() { return 3; }
int foo2() { return 4; }
int main()
{
    printf("%d\n", foo1()*foo2());
}
```

Hanks–iMac:Downloads hank$ gcc function.c
Hanks–iMac:Downloads hank$ ./a.out
12
Iteration is done with constructs “for” and “while”

```
#include <stdio.h>

int main()
{
    int X = 0;
    while (X < 83)
        X += 5;
    printf("X is %d\n", X);
}
```

```
hank$ gcc while.c
hank$ ./a.out
X is 85
```
For loop in practice
(with increment operator)

Hanks-iMac:Downloads hank$ cat sum.c
#include <stdio.h>
int main()
{
    int sum = 0;
    int i;
    for (i = 0 ; i < 10 ; i++)
    {
        sum = sum+i;
    }
    printf("Sum is %d\n", sum);
}
Hanks-iMac:Downloads hank$ gcc sum.c
Hanks-iMac:Downloads hank$ ./a.out
Sum is 45
Homework #2A: find numbers between 10 and 500 that are “odd” and “composite” (not-prime)

My program is creating this output these via printf() functions

Hanks-iMac:code hank$ ./a.out
15 is a composite number.
21 is a composite number.
25 is a composite number.
27 is a composite number.
33 is a composite number.
35 is a composite number.
39 is a composite number.
45 is a composite number.
49 is a composite number.
51 is a composite number.
55 is a composite number.
57 is a composite number.
63 is a composite number.
65 is a composite number.
69 is a composite number.
75 is a composite number.
77 is a composite number.
81 is a composite number.
85 is a composite number.
CIS 212: Project #2A
Assigned: October 4th, 2019
Due: October 8th, 2019
(which means submitted by 6am on October 9th, 2019)
Worth 6% of your grade

Assignment:
   1) Write a C program that finds the odd, composite numbers between 10 and 500. The name of the C program should be "project2A.c"
      a. Composite means it is not prime, i.e., "C" is composite if there exists an A and B such that A>1, B>1, and A*B = C.
      b. Odd means that it is not divisible by 2.
      c. Therefore "odd, composite", means both odd and composite
   2) Your program should have the exact same output as mine.
   3) You can confirm this:
      a. Download "proj2A_checker"
      b. Run your program as "sh ./proj2A_checker"
      c. You can also see the correct output as "proj2A_correct_output"

This project will be graded by:
   1) Running the proj2A_checker program
   2) Inspection of your code

If the diff program shows any differences whatsoever, you will get less than half credit. This includes formatting differences, not having all the numbers, etc.
Further, this will be checked on the CIS 212 Virtual Box image.

If you write your code on your own machine (not on the VirtualBox): if your program produces the wrong output on that machine, then it will get less than half credit. **Explicitly:** it may run fine on your computer, but not on the VirtualBox. If this happens, you will receive less than half credit. Therefore, when you believe your program works, you should transfer it to the VirtualBox and test it there. This will be the case for all programs for the rest of this term, whether or not I include this information in the prompt.

**What should you upload?:** Just a single file, which is your C source code.

**Note:** I will not award much credit to programs that “cheat.” I.e., if you work out all of the odd, composite numbers ahead of time and put them into a list, and then use that list of numbers, then you will not receive credit.
Plan for today

• Quick review of hello world in C
• Permissions
• Project 1B
• More Unix
• What is Data? (repeat)
• Beginning on Memory in C
Files

- Unix maintains a file system
  - File system controls how data is stored and retrieved
- Primary abstractions:
  - Directories
  - Files
- Files are contained within directories
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
Permissions: System Calls

• System calls: a request from a program to the OS to do something on its behalf
  – ... including accessing files and directories

• System calls:
  – Typically exposed through functions in C library
  – Unix utilities (mkdir, ls, touch) are programs that call these functions

Permissions in Unix are enforced via system calls.
Permissions: Unix Groups

• Groups are a mechanism for saying that a subset of Unix users are related
  – I could make a “212_F19” unix group

• Members:
  • Me
  • 2 TAs

CIS uses “groupctl”

The commands for creating a group tend to vary, and are often done by a system administrator
Permissions

• Permissions are properties associated with files and directories
  – System calls have built-in checks to permissions
    • Only succeed if proper permissions are in place
• Three classes of permissions:
  – User: access for whoever owns the file
    • You can prevent yourself from accessing a file!
      – (But you can always change it back)
  – Group: allow a Unix group to access a file
  – Other: allow anyone on the system to access a file
Three types of permissions

- Read
- Write
- Execute (see next slide)
Executable files

• An executable file: a file that you can invoke from the command line
  – Scripts
  – Binary programs

• The concept of whether a file is executable is linked with file permissions
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)
Manpage for chmod

• “man chmod”

```
CHMOD(1) 
BSD General Commands Manual 
CHMOD(1)

NAME
  chmod -- change file modes or Access Control Lists

SYNOPSIS
  chmod [-fv] [-R [-H | -L | -P]] mode file ...
  chmod [-fv] [-R [-H | -L | -P]] [-a | +a | =a] ACE file ...
  chmod [-fhv] [-R [-H | -L | -P]] [-E] file ...
  chmod [-fhv] [-R [-H | -L | -P]] [-C] file ...
  chmod [-fhv] [-R [-H | -L | -P]] [-N] file ...

DESCRIPTION
  The chmod utility modifies the file mode bits of the listed files as
  specified by the mode operand. It may also be used to modify the Access
  Control Lists (ACLs) associated with the listed files.

  The generic options are as follows:

  -f       Do not display a diagnostic message if chmod could not modify the
           mode for file.
```
Unix commands for groups

• chgrp: changes the group for a file or directory
  – chgrp <group> <filename>

• groups: lists groups you are in
ls -l

- Long listing of files

```
Last login: Thu Apr  3 08:09:23 on ttys007
C02LN00GFD58:~ hank$ mkdir CIS330
C02LN00GFD58:~ hank$ cd CIS330
C02LN00GFD58:CIS330 hank$ touch a
C02LN00GFD58:CIS330 hank$ ls -l

total 0
-rw-r--r-- 1 hank staff 0 Apr  3 08:14 a
```

How to interpret this?
Permissions and Directories

• You can only enter a directory if you have “execute” permissions to the directory

• Quiz: a directory has permissions “400”. What can you do with this directory?

  Answer: it depends on what permissions a system call requires.
Directories with read, but no execute

```
Last login: Thu Apr  3 08:14:33 on ttys007
C02LN00GFD58:~ hank$ mkdir CIS330
C02LN00GFD58:~ hank$ touch CIS330/a
C02LN00GFD58:~ hank$ chmod 400 CIS330
C02LN00GFD58:~ hank$ ls CIS330
a
C02LN00GFD58:~ hank$ cd CIS330
-bash: cd: CIS330: Permission denied
C02LN00GFD58:~ hank$ cat CIS330/a
cat: CIS330/a: Permission denied
```
Plan for today

• Quick review of hello world in C
• Permissions
• Project 1B
• More Unix
• What is Data? (repeat)
• Beginning on Memory in C
Unix scripts

• Scripts
  – Use an editor (vi/emacs/other) to create a file that contains a bunch of Unix commands
  – Give the file execute permissions
  – Run it like you would any program!!
Example

Hanks-iMac:212 hank$ cat my_first_script
echo "Hello CIS 212, from my script"
echo "This is the ls -l for the script"
ls -l my_first_script
Hanks-iMac:212 hank$ ./my_first_script
-bash: ./my_first_script: Permission denied
Hanks-iMac:212 hank$ chmod 755 my_first_script
Hanks-iMac:212 hank$ ./my_first_script
Hello CIS 212, from my script
This is the ls -l for the script
-rwxr-xr-x  1 hank  staff  99 Oct  7 18:58 my_first_script
Hanks-iMac:212 hank$

Why "./my_first_script"?
Why do we need ".//"?
(answer coming in a bit)
Unix scripts

• Arguments
  – Assume you have a script named “myscript”
  – If you invoke it as “myscript foo bar”
  – Then
    • $# == 2
    • $1 == foo
    • $2 == bar
Project 1B

• Summary: write a script that will create a specific directory structure, with files in the directories, and specific permissions.
Assignment: Create a shell script that will create a directory structure and files within that directory structure, all with the specified file permissions. The script should be named “proj1b.sh”. (A consistent name will help with grading.)

Note: you are only allowed to use the following commands: mkdir, touch, cd, chmod, mv, cp, rm, rmdir. (You do not need to use all of these commands to successfully complete the assignment.)
Project 1B

The directory structure should be:

```
Root dir

Dir1
  Permissions: 770
  File1
    Permissions: 400

Dir3
  Permissions: 000

Dir2
  Permissions: 775
  File2
    Permissions: 640

Dir4
  Permissions: 750
  File3
    Permissions: 200
  File4
    Permissions: 666
```

Key

Files:
- Name of file
  - Permissions

Directories:
- Name of directory
  - Permissions
Plan for today

- Quick review of hello world in C
- Permissions
- Project 1B
- More Unix
- What is Data? (repeat)
- Beginning on Memory in C
“.” and “..”

- Unix convention:
  - “.” : the current directory
  - “..” : the parent directory

Quiz: you in `/path/to/dir` and issue “cd ./.././../”. Where do you end up?

Answer: “/path”
pwd and $PXD

• `pwd`: Unix command that returns the “present working directory”
• `$PXD`: Environment variable that contains the present working directory
• `$OLDPXD`: Environment variable that contains the previous present working directory
• “-“: Shortcut for the previous PWD
PATH environment variable

When the shell wants to invoke a command, it searches for the command in the path.

```
128-223-223-72-wireless:Documents hank$ echo $PATH
/opt/local/bin:/opt/local/sbin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin:/opt/X11/bin:/usr/texbin
```

```
128-223-223-72-wireless:Documents hank$ echo $PATH | tr : '\\n'
```

“tr”: Unix command for replacing characters (translating characters).
which

which: tells you the directory the shell is finding a command in.
Invoking programs in current directory

shell works with ./prog_name since it views this as a path. Hence $PATH is ignored.
Invoking programs in current directory

```
C02LN00GFD58:330 hank$ echo "echo hello world" > my_script
C02LN00GFD58:330 hank$ chmod 755 my_script
C02LN00GFD58:330 hank$ my_script
-bash: my_script: command not found
C02LN00GFD58:330 hank$ ./my_script
hello world
C02LN00GFD58:330 hank$ export PATH=$PATH:.
C02LN00GFD58:330 hank$ my_script
hello world
C02LN00GFD58:330 hank$
```
Trojan Horse Attack

- export PATH=..:$PATH
  - why is this a terrible idea?

```
C02LN00GFD58:330 hank$ echo "rm -Rf ~" > ls
C02LN00GFD58:330 hank$ export PATH=..:$PATH
C02LN00GFD58:330 hank$ chmod 755 ls
C02LN00GFD58:330 hank$ ls # this would be bad...
```
Wild Cards

- ‘*’ (asterisk) serves as a wild card that does pattern matching
Wild Cards

- You can use multiple asterisks for complex patterns

```
C02LN00GFD58:~ hank$ ls -1 */*.C
330/binary.C
330/cis330.C
Downloads/avtConnComponentsExpression.C
```
Plan for today

• Quick review of hello world in C
• Permissions
• Project 1B
• More Unix
• What is Data? (repeat)
• Beginning on Memory in C
To a Computer, Data Means Lots of Bits. So What is a Bit?

• Short for “binary digit”
• 2 states:
  – on (1)
  – off (0)
• Like a light switch
• Computers can turn their bits (light switches) on and off
• Computers implement this using electricity in a “capacitor”:
  – has electrical charge: on / 1
  – no electrical charge: off / 0
What is a Byte?

• A group of 8 bits

• We could write the values of the 8 bits in a row.
  • Example: 01000001
    – This is a binary number
Binary Numbers

• Normal system (called decimal):
  – Values for any digit are: 0,1,2,3,4,5,6,7,8,9
  – What we use every day

• Binary:
  – Values for any digit are: 0,1
  – What computers use
Conventions Surrounding Binary Numbers

- 01000001 is 65 in decimal
- By convention, 65 represents ‘A’

— Called “ASCII”

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>000</td>
<td>NULL</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>001</td>
<td>SOH</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>010</td>
<td>STX</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>011</td>
<td>ETX</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>100</td>
<td>EOT</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>101</td>
<td>ENQ</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>110</td>
<td>ACK</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>111</td>
<td>BEL</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>1000</td>
<td>HT</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>1001</td>
<td>LF</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>1010</td>
<td>VT</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>1011</td>
<td>FF</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>1100</td>
<td>CR</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>1101</td>
<td>SUB</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>1110</td>
<td>ESC</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>1111</td>
<td>crlf</td>
</tr>
</tbody>
</table>

Also conventions for supporting decimal numbers (3.456)...

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>037</td>
<td>037</td>
<td>ETB</td>
</tr>
<tr>
<td>24</td>
<td>038</td>
<td>038</td>
<td>CAN</td>
</tr>
<tr>
<td>25</td>
<td>039</td>
<td>039</td>
<td>EM</td>
</tr>
<tr>
<td>26</td>
<td>040</td>
<td>040</td>
<td>SUB</td>
</tr>
<tr>
<td>27</td>
<td>043</td>
<td>043</td>
<td>ESC</td>
</tr>
<tr>
<td>28</td>
<td>044</td>
<td>044</td>
<td>FS</td>
</tr>
<tr>
<td>29</td>
<td>045</td>
<td>045</td>
<td>GS</td>
</tr>
<tr>
<td>30</td>
<td>046</td>
<td>046</td>
<td>RS</td>
</tr>
<tr>
<td>31</td>
<td>047</td>
<td>047</td>
<td>US</td>
</tr>
</tbody>
</table>

Source: www.LookupTables.com
• There are 10 types of people in the world – those that know binary and those that don’t
How do Bytes relate to computers?

• Computers have “memory”
  – Places to store data and then retrieve it later
• The memory is made up of many, many Bytes
Von Neumann Architecture

Input Device

Central Processing Unit

Control Unit

Arithmetic/Logic Unit

Memory Unit

Output Device

Image source: wikipedia
There is so much data that we need lots of prefixes...

- 1 KiloByte (KB) = 1024 Bytes
- 1 MegaByte (MB) = 1024 KB = ~1million Bytes
- 1 GigaByte (GB) = 1024 MB = ~1billion Bytes
- 1 TeraByte (TB) = 1024 GB = ~1trillion Bytes
- 1 PetaBytes (PB) = 1024 TB = ~1quadrillion Bytes
- 1 ExaByte = 1024 PB = ~1 quintillion Bytes
  = 1 billion billion Bytes
Analogy

- 1 story hotel
- 6 rooms, side by side
  - Room 1, Room 2, Room 3, Room 4, Room 5, Room 6
- Light left on == 1
- Light left off == 0
What is the Data?
What is the Data?
Mental Model For Memory

• 1 story hotel
• First room is 10000000
• Next room is 10000001
• And so on...
Reserving Memory

• You go to the hotel front desk and ask for a room
• When you are done, you check out and tell them you have left the room
Plan for today

• Quick review of hello world in C
• Permissions
• Project 1B
• More Unix
• What is Data? (repeat)
• Beginning on Memory in C
Why C?

• You can control the memory
• That helps get good performance

• If you don’t control the memory (like in other programming languages), you are likely to get poor performance

• ... so let’s talk about memory
Automatic vs Dynamic Memory

• In C, there are two types of variables
  – Actually three, but ignoring “static” for now
• Automatic: taken care of for you automatically
• Dynamic: you manage it

• So, of course, everyone wants automatic, right?
Minimum Reservation

• When reserving memory, the minimum request is 1 byte
  – You can access individual bits, but the hotel front desk will only let you reserve 1 byte at a time
  – So, minimum reservation is a block of 8 rooms
Types in C

- **char**: ASCII character, 1 byte
  - Values: -128 to 127
- **unsigned char**: 1 byte
  - Values: 0 to 255

Both char and unsigned char treat this as 127

Unsigned char: 128
Char: -128 ("two’s complement")
Hotel Rooms

- You must reserve rooms in blocks of 8 – 8, 16, 24, 32, ...
- The data is just data
- But there are conventions on how to interpret it
Known types in C

- char – 1 byte
- unsigned char – 1 byte
- int – 4 bytes
- float – 4 bytes
- double – 8 bytes
- more...
unsigned int

- 4 bytes -> 32 bits
- Values: 0 to $2^{32}-1$
int

• 4 bytes -> 32 bits
• Values: \(-2^{31} \) to \(2^{31}-1\)
- Essentially like scientific notation
- $-534563E-6$
- Store 4 things:
  - Sign of number (1 bit)
  - "Mantissa" (534563)
  - Sign of exponent (1 bit)
  - Value of exponent (6)
- Inexact!!
Requesting automatic memory

- This is done “automatically”...

```c
#include <stdio.h>

int main()
{
    int A;
    A = 4;
    printf("Value of A is %d\n", A);
}
```
#include <stdio.h>

int main()
{
    int A[21];
    int i;
    for (i = 0 ; i < 21 ; i++)
        A[i] = 3;
    printf("Value of A[10] is %d\n", A[10]);
}

This is just 84 (4*21) bytes. We interpret these 84 bytes as 21 integers, following known conventions.
Important Memory Concepts in C (1/9): Automatic Vs Dynamic

• You can allocate variables that only live for the invocation of your function
  – Called automatic/stack variables (will talk more about this later)

• You can allocated variables that live for the whole program (or until you delete them)
  – Called dynamic/heap variables (will talk more about this later as well)
Important Memory Concepts in C (2/9): Pointers

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

• Address of: gets the address of memory
  – Operator: ‘&’
  – Example:
    ```
    int x;
    int *y = &x;
    ```
Important Memory Concepts in C (3/9): Memory allocation

• Special built-in function to allocate memory from heap: `malloc`
  – Interacts with Operating System
  – Argument for `malloc` is how many bytes you want

• Also built-in function to deallocate memory: `free`
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
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...

Vacuum tubes in ENIAC
Image source: wikipedia
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 * 1 year → ~$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 $18$M
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>CORES</th>
<th>GHZ</th>
<th>MEMORY</th>
<th>FABRIC</th>
<th>DDR4</th>
<th>POWER</th>
<th>RECOMMENDED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CORES</td>
<td>GHZ</td>
<td>MEMORY</td>
<td>FABRIC</td>
<td>DDR4</td>
<td>POWER</td>
<td>RECOMMENDED</td>
</tr>
<tr>
<td>7290¹</td>
<td>72</td>
<td>1.5</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>
<td></td>
</tr>
<tr>
<td>7250</td>
<td>68</td>
<td>1.4</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>
<td></td>
</tr>
<tr>
<td>7230</td>
<td>64</td>
<td>1.3</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>
<td></td>
</tr>
<tr>
<td>7210</td>
<td>64</td>
<td>1.3</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>
<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.

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
  - Mid-2000’s: people start hacking interface to program a GPU to make it do 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