Lecture 4:
Beginning on Memory

UNIX, C, and Data Structures

October 10, 2019
Hank Childs, University of Oregon
What should you be doing?

• Due today: 1B
• Assigned today: 2B
• Reading the textbook, 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
  – will be an upcoming quiz on details in the book (things I have not lectured on will be fair game)
Quick Permissions Review / 1B
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)
Unix Script 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$
Project 1B

• Summary: write a script that will create a specific directory structure, with files in the directories, and specific permissions.
Project 1B

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

# Dir2
  Permissions: 775
  File2
    Permissions: 640

# Dir3
  Permissions: 000

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

Key

- Files:
  - Name of file Permissions

- Directories:
  - Name of directory Permissions:
Data Review (whirlwind)
Conventions Surrounding Binary Numbers

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

— Called “ASCII”

<table>
<thead>
<tr>
<th>Dec</th>
<th>HxOct</th>
<th>Char</th>
<th>Dec</th>
<th>HxOct</th>
<th>Html Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>NULL</td>
<td>32</td>
<td>20</td>
<td>#32: Space</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>SOH</td>
<td>33</td>
<td>21</td>
<td>#33: !</td>
</tr>
<tr>
<td>2</td>
<td>002</td>
<td>STX</td>
<td>34</td>
<td>22</td>
<td>#34: &quot;</td>
</tr>
<tr>
<td>3</td>
<td>003</td>
<td>ETX</td>
<td>35</td>
<td>23</td>
<td>#35: #</td>
</tr>
<tr>
<td>4</td>
<td>004</td>
<td>EOT</td>
<td>36</td>
<td>24</td>
<td>#36: $</td>
</tr>
<tr>
<td>5</td>
<td>005</td>
<td>ENQ</td>
<td>37</td>
<td>25</td>
<td>#37: %</td>
</tr>
<tr>
<td>6</td>
<td>006</td>
<td>ACK</td>
<td>38</td>
<td>26</td>
<td>#38: &amp;</td>
</tr>
<tr>
<td>7</td>
<td>007</td>
<td>BEL</td>
<td>39</td>
<td>27</td>
<td>#39: '</td>
</tr>
<tr>
<td>8</td>
<td>010</td>
<td>BS</td>
<td>40</td>
<td>28</td>
<td>#40: (</td>
</tr>
<tr>
<td>9</td>
<td>011</td>
<td>HT</td>
<td>41</td>
<td>29</td>
<td>#41: )</td>
</tr>
<tr>
<td>10</td>
<td>012</td>
<td>LF</td>
<td>42</td>
<td>30</td>
<td>#42: *</td>
</tr>
<tr>
<td>11</td>
<td>013</td>
<td>VT</td>
<td>43</td>
<td>31</td>
<td>#43: +</td>
</tr>
<tr>
<td>12</td>
<td>014</td>
<td>FF</td>
<td>44</td>
<td>32</td>
<td>#44: ,</td>
</tr>
<tr>
<td>13</td>
<td>015</td>
<td>CR</td>
<td>45</td>
<td>33</td>
<td>#45: -</td>
</tr>
<tr>
<td>14</td>
<td>016</td>
<td>SO</td>
<td>46</td>
<td>34</td>
<td>#46: .</td>
</tr>
<tr>
<td>15</td>
<td>017</td>
<td>SI</td>
<td>47</td>
<td>35</td>
<td>#47: /</td>
</tr>
<tr>
<td>16</td>
<td>020</td>
<td>DLE</td>
<td>48</td>
<td>36</td>
<td>#48: 0</td>
</tr>
<tr>
<td>17</td>
<td>021</td>
<td>DC1</td>
<td>49</td>
<td>37</td>
<td>#49: 1</td>
</tr>
<tr>
<td>18</td>
<td>022</td>
<td>DC2</td>
<td>50</td>
<td>38</td>
<td>#50: 2</td>
</tr>
<tr>
<td>19</td>
<td>023</td>
<td>DC3</td>
<td>51</td>
<td>39</td>
<td>#51: 3</td>
</tr>
<tr>
<td>20</td>
<td>024</td>
<td>DC4</td>
<td>52</td>
<td>40</td>
<td>#52: 4</td>
</tr>
<tr>
<td>21</td>
<td>025</td>
<td>NAK</td>
<td>53</td>
<td>41</td>
<td>#53: 5</td>
</tr>
<tr>
<td>22</td>
<td>026</td>
<td>SYN</td>
<td>54</td>
<td>42</td>
<td>#54: 6</td>
</tr>
<tr>
<td>23</td>
<td>027</td>
<td>ETB</td>
<td>55</td>
<td>43</td>
<td>#55: 7</td>
</tr>
<tr>
<td>24</td>
<td>030</td>
<td>CAN</td>
<td>56</td>
<td>44</td>
<td>#56: 8</td>
</tr>
<tr>
<td>25</td>
<td>031</td>
<td>EM</td>
<td>57</td>
<td>45</td>
<td>#57: 9</td>
</tr>
<tr>
<td>26</td>
<td>032</td>
<td>SUB</td>
<td>58</td>
<td>46</td>
<td>#58: 0</td>
</tr>
<tr>
<td>27</td>
<td>033</td>
<td>ESC</td>
<td>59</td>
<td>47</td>
<td>#59: 1</td>
</tr>
<tr>
<td>28</td>
<td>034</td>
<td>FS</td>
<td>60</td>
<td>48</td>
<td>#60: 2</td>
</tr>
<tr>
<td>29</td>
<td>035</td>
<td>GS</td>
<td>61</td>
<td>49</td>
<td>#61: 3</td>
</tr>
<tr>
<td>30</td>
<td>036</td>
<td>RS</td>
<td>62</td>
<td>50</td>
<td>#62: 4</td>
</tr>
<tr>
<td>31</td>
<td>037</td>
<td>US</td>
<td>63</td>
<td>51</td>
<td>#63: 5</td>
</tr>
</tbody>
</table>

Source: www.LookupTables.com
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
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?

1 0 0 0 1 0
What is the Data?
Mental Model For Memory

- 1 story hotel
- First room is 10000000
- Next room is 10000001
- And so on...
(Time to slow down the reviewing)
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
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")
Great Question from After the Last Lecture!

- `int X = 2^31-1;`
- `X += 1;`
- What happens?
- Answer: `X = -2^31`.
- This is called an overflow.
Overflow

1. Overflow Condition

- Arithmetic operations have a potential to run into a condition known as overflow.
- Overflow occurs with respect to the size of the data type that must accommodate the result.
- Overflow indicates that the result was too large or too small to fit in the original data type.
- When two signed 2's complement numbers are added, overflow is detected if:
  1. both operands are positive and the result is negative, or
  2. both operands are negative and the result is positive.
- When two unsigned numbers are added, overflow occurs if
  - there is a carry out of the leftmost bit.

Source: http://www.c-jump.com/CIS77/CPU/Overflow/lecture.html
## 2. Binary Arithmetic

- Computers don't know the difference between **signed** and **unsigned** binary numbers.
- This is a good thing, because it makes logic circuits fast.
- This is also a bad thing, because distinguishing between **signed** and **unsigned** is our responsibility.
- The distinction is very important when detecting an **overflow** after addition or subtraction.
- Correct approach to **detect the overflow** is to consider two separate cases:
  1. Overflow when adding **unsigned** numbers.
  1. Overflow when adding **signed** numbers.
Overflow: How to Detect?

```c
int addOvf(int a, int b) {
    int result = a + b;
    if (a > 0 && b > 0 && result < 0)
        return -1;
    if (a < 0 && b < 0 && result > 0)
        return -1;
    return 0;
}
```

Inspired by: https://www.geeksforgeeks.org/check-for-integer-overflow/
More Thoughts on Overflow

• Rarely comes up in practice
  – ... but sometimes the “rarely comes up in practice” are the worst bugs....
• Often programmers know that overflows are not possible & do not defend against it
• But it is good to be aware of
Overflows with unsigned char: let’s do an example
Hotel Rooms

- You must reserve rooms in blocks of 8 (8 bits = 1 byte) – 8, 16, 24, 32, ...
- The data is just data
- But there are conventions on how to interpret it

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Oct</th>
<th>Bin</th>
<th>ASCII</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
<td>0000</td>
<td>0000</td>
<td>null</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>01</td>
<td>0001</td>
<td>100</td>
<td>HT</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>02</td>
<td>0010</td>
<td>1001</td>
<td>space</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>03</td>
<td>0011</td>
<td>1001</td>
<td>#</td>
</tr>
<tr>
<td>4</td>
<td>0012</td>
<td>04</td>
<td>0012</td>
<td>1001</td>
<td>$</td>
</tr>
<tr>
<td>5</td>
<td>0013</td>
<td>05</td>
<td>0013</td>
<td>1001</td>
<td>&amp;</td>
</tr>
</tbody>
</table>

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);
}
```
Arrays

This is just 84 (4*21) bytes. We interpret these 84 bytes as 21 integers, following known conventions.

```c
#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]);
}
```
Let’s pause and do some examples with arrays (and film it)
CIS 212: Project #2B
Assigned: October 10, 2019
Due: October 16, 2019
(which means submitted by 6am on October 17, 2019)
Worth 4% of your grade

Assignment:
1) Write a C program that sorts 100 numbers in an array. The name of the C program should be “project2B.c”
2) You can sort however you want.
   a. https://en.wikipedia.org/wiki/Bubble_sort if you need ideas. Also see the Python code below.
   b. You should not use any subroutines from the C library. (Don’t use qsort, for example)
3) Your program should have the exact same output as mine.
   a. 10 numbers per row, 10 rows
   b. Note I used “tab” to do whitespaces. That makes it pretty. You will need to use tabs too.
   c. You can also see the correct output as “proj2B_correct_output”. Make sure to download this by right clicking and “save link as.” Otherwise, tabs can get converted to spaces.
4) You can confirm this:
   a. Download “proj2B_checker”.
   b. Do a chmod: “chmod 755 proj2B_checker”
   c. Run your program as “./proj2B_checker”

This project will be graded by:
1) Running proj2B_checker on your code and confirming it is correct
2) Inspection of your code
Correct output

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>40</td>
<td>45</td>
<td>66</td>
<td>70</td>
<td>77</td>
<td>95</td>
<td>133</td>
<td>134</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>159</td>
<td>166</td>
<td>166</td>
<td>181</td>
<td>190</td>
<td>202</td>
<td>214</td>
<td>216</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>237</td>
<td>243</td>
<td>252</td>
<td>267</td>
<td>268</td>
<td>285</td>
<td>314</td>
<td>325</td>
<td>325</td>
<td>369</td>
<td></td>
</tr>
<tr>
<td>385</td>
<td>388</td>
<td>391</td>
<td>395</td>
<td>402</td>
<td>414</td>
<td>428</td>
<td>434</td>
<td>437</td>
<td>443</td>
<td></td>
</tr>
<tr>
<td>446</td>
<td>466</td>
<td>474</td>
<td>477</td>
<td>508</td>
<td>517</td>
<td>552</td>
<td>577</td>
<td>577</td>
<td>599</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>603</td>
<td>625</td>
<td>634</td>
<td>635</td>
<td>637</td>
<td>647</td>
<td>649</td>
<td>657</td>
<td>673</td>
<td></td>
</tr>
<tr>
<td>673</td>
<td>679</td>
<td>682</td>
<td>682</td>
<td>687</td>
<td>697</td>
<td>705</td>
<td>712</td>
<td>717</td>
<td>729</td>
<td></td>
</tr>
<tr>
<td>737</td>
<td>752</td>
<td>755</td>
<td>760</td>
<td>765</td>
<td>765</td>
<td>780</td>
<td>794</td>
<td>801</td>
<td>804</td>
<td></td>
</tr>
<tr>
<td>805</td>
<td>821</td>
<td>855</td>
<td>858</td>
<td>876</td>
<td>899</td>
<td>902</td>
<td>904</td>
<td>916</td>
<td>920</td>
<td></td>
</tr>
<tr>
<td>927</td>
<td>941</td>
<td>949</td>
<td>950</td>
<td>953</td>
<td>954</td>
<td>963</td>
<td>964</td>
<td>968</td>
<td>987</td>
<td></td>
</tr>
</tbody>
</table>

Used tab ("\t") to make the columns so organized
#include <stdio.h>

int main()
{
    /* YOUR CODE GOES HERE */

    /* SORT IT */

    /* PRINT IT */

    /* HINT: WRITE THE PRINT FUNCTION FIRST */
}

Important Memory Concepts in C: 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 allocate variables that live for the whole program (or until you delete them)
  – Called dynamic/heap variables (will talk more about this later as well)
#include <stdio.h>

void function1()
{
    int X = 4;
    printf("X is %d\n", X);
    /* X goes out of scope and the end of this function and thus
     * *automatically* disappears and is no longer accessible */
}

int main()
{
    int Z = 6;
    function1();

    /* NEW SCOPE */
    {
        int Y = 5;
    }
    /* Y no longer exists. Automatically allocated and de-allocated */

    /* only valid variable here is Z ... not X, not Y. */
    printf("Z is %d\n", Z);

    /* Now Z goes away. */
}
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**
Important Memory Concepts in C: Hexadecimal address

• Memory addresses are in hexadecimal
  – Hexadecimal: 16 options for each digit
  – 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

• House address: 123 A Street

• Memory address: 0x7FFFFFFF3AB
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:
    ```c
    int x;
    int *y = &x;  // this example is pointing to an automatic variable, not a dynamic variable
    ```
Let’s pause and do some more examples with arrays (and film it too)
Dynamic 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
Automatic vs Dynamic

- Automatic memory lives only for its current scope
- Dynamic memory lives until you free it, or until the program ends
This is just fine...

Hanks-iMac:Downloads hank$ cat scope.c
#include <stdio.h>
#include <stdlib.h>

int *foo1()
{
    int *X = malloc(sizeof(int)*2);
    X[0] = 1;
    X[1] = 2;
    return X;
}

int main()
{
    int *Y = foo1();
    printf("Y[0] is %d and Y[1] is %d\n", Y[0], Y[1]);
}

Hanks-iMac:Downloads hank$ gcc scope.c
Hanks-iMac:Downloads hank$ ./a.out
Y[0] is 1 and Y[1] is 2
This is not fine...

```
Hanks-iMac:Downloads hank$ cat bad_scope.c
#include <stdio.h>
#include <stdlib.h>

int *foo1()
{
    int X[2];
    X[0] = 1;
    X[1] = 2;
    return X;
}

int main()
{
    int *Y = foo1();
    printf("Y[0] is %d and Y[1] is %d\n", Y[0], Y[1]);
}
```

Hanks-iMac:Downloads hank$ gcc bad_scope.c
bad_scope.c:9:12: warning: address of stack memory associated with local variable 'X' returned [-Wreturn-stack-address]
    return X;

1 warning generated.
Hanks-iMac:Downloads hank$ ./a.out
Y[0] is 1 and Y[1] is 2
```
int *foo1()
{
    int X[2];
    X[0] = 1;
    X[1] = 2;
    return X;
}

int *foo2()
{
    int A[2];
    A[0] = 3;
    A[1] = 4;
    return A;
}

int main()
{
    int *Y = foo1();
    int *B = foo2();
    printf("Y[0] is %d and Y[1] is %d\n", Y[0], Y[1]);
    printf("B[0] is %d and B[1] is %d\n", B[0], B[1]);
}

Hanks-iMac:Downloads hank$ gcc bad_scope.c
bad_scope.c:9:12: warning: address of stack memory associated with local variable 'X' returned [-Wreturn-stack-address]
    return X;
    ^
bad_scope.c:17:12: warning: address of stack memory associated with local variable 'A' returned [-Wreturn-stack-address]
    return A;
    ^
2 warnings generated.
Hanks-iMac:Downloads hank$ ./a.out
Y[0] is 3 and Y[1] is 4
B[0] is 0 and B[1] is 0
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...

The part about clock speed increasing with the number of transistors stopped about fifteen years ago.

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 $18M$
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>CPU Core Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORES</td>
<td>GHZ</td>
<td>MEMORY</td>
</tr>
<tr>
<td>7290¹</td>
<td>72</td>
<td>1.5</td>
</tr>
<tr>
<td>Best Performance/Node</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7250</td>
<td>68</td>
<td>1.4</td>
</tr>
<tr>
<td>Best Performance/Watt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7230</td>
<td>64</td>
<td>1.3</td>
</tr>
<tr>
<td>Best Memory Bandwidth/Core</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7210</td>
<td>64</td>
<td>1.3</td>
</tr>
<tr>
<td>Best Value</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.
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 (graph cards)

• Historical:
  – Introduced to accelerate graphics (gaming!)
  – Boom with desktop PCs late 90s onward
  – Mid-2000’s: people started hacking 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?

• NVIDIA: company that makes GPUs
• NVIDIA Volta: latest type of NVIDIA GPU
• Volta facts:
  – 5120 cores
  – 1200MHz clock speed
  – can do 2000X more operations than my laptop
  – Suggested MSRP: $2,999.00

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