Lecture 4:
Beginning on Memory
Hank’s OH

• OH today: 1230-130, 301 Deschutes
• And: telecon today 1130-1230
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
      ├── File1
      │    ┌── Dir2
      │    │    └── File2
      │    └── Dir3
      │         └── Dir4
      │             └── File4
      └── File3
```

Key
- Files: Name of file Permissions
- Directories: Name of directory Permissions:

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

Note 2: Every file should contain the contents "This is file <name>". For example, the contents of File1 should be "This is File1".
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 Code</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>NUL (null)</td>
</tr>
<tr>
<td>1</td>
<td>0010</td>
<td>SOH (start of heading)</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>STX (start of text)</td>
</tr>
<tr>
<td>3</td>
<td>0010</td>
<td>ETX (end of text)</td>
</tr>
<tr>
<td>4</td>
<td>0011</td>
<td>EOT (end of transmission)</td>
</tr>
<tr>
<td>5</td>
<td>0100</td>
<td>ENQ (enquiry)</td>
</tr>
<tr>
<td>6</td>
<td>0100</td>
<td>ACK (acknowledge)</td>
</tr>
<tr>
<td>7</td>
<td>0101</td>
<td>BEL (bell)</td>
</tr>
<tr>
<td>8</td>
<td>0110</td>
<td>BS (backspace)</td>
</tr>
<tr>
<td>9</td>
<td>0111</td>
<td>HT (horizontal tab)</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>LF (line feed, new line)</td>
</tr>
<tr>
<td>11</td>
<td>1001</td>
<td>VT (vertical tab)</td>
</tr>
<tr>
<td>12</td>
<td>1010</td>
<td>FF (form feed, new page)</td>
</tr>
<tr>
<td>13</td>
<td>1011</td>
<td>CR (carriage return)</td>
</tr>
<tr>
<td>14</td>
<td>1100</td>
<td>SO (shift out)</td>
</tr>
<tr>
<td>15</td>
<td>1101</td>
<td>SI (shift in)</td>
</tr>
<tr>
<td>16</td>
<td>1110</td>
<td>DLE (data link escape)</td>
</tr>
<tr>
<td>17</td>
<td>1111</td>
<td>DC1 (device control 1)</td>
</tr>
<tr>
<td>18</td>
<td>0000</td>
<td>DC2 (device control 2)</td>
</tr>
<tr>
<td>19</td>
<td>0001</td>
<td>DC3 (device control 3)</td>
</tr>
<tr>
<td>20</td>
<td>0010</td>
<td>DC4 (device control 4)</td>
</tr>
<tr>
<td>21</td>
<td>0011</td>
<td>NAK (negative acknowledge)</td>
</tr>
<tr>
<td>22</td>
<td>0100</td>
<td>SYN (synchronous idle)</td>
</tr>
<tr>
<td>23</td>
<td>0101</td>
<td>ETB (end of trans. block)</td>
</tr>
<tr>
<td>24</td>
<td>0110</td>
<td>CAN (cancel)</td>
</tr>
<tr>
<td>25</td>
<td>0111</td>
<td>EM (end of medium)</td>
</tr>
<tr>
<td>26</td>
<td>1000</td>
<td>SUB (substitute)</td>
</tr>
<tr>
<td>27</td>
<td>1001</td>
<td>ESC (escape)</td>
</tr>
<tr>
<td>28</td>
<td>1010</td>
<td>FS (file separator)</td>
</tr>
<tr>
<td>29</td>
<td>1011</td>
<td>GS (group separator)</td>
</tr>
<tr>
<td>30</td>
<td>1100</td>
<td>RS (record separator)</td>
</tr>
<tr>
<td>31</td>
<td>1101</td>
<td>US (unit separator)</td>
</tr>
</tbody>
</table>

Source: [LookupTables.com](https://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
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
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
Addresses

• Every byte of memory has an address
  – Not Room 1, Room 2, Room 3
  – Instead in hexadecimal
    • 0x7fff5abb2b70
What is hexadecimal?

- 16 values -- 0-9 + A-F
- Denoted on computers with prefix 0x
- 0x100 == 256
- Get used to see memory addresses in hexadecimal
- Not going to lecture more on this ...
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 in C
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?
What is happening here?

```
#include <stdio.h>

int main()
{
    int A;
    A=4;
    printf("Value of A is \%d\n", A);
    return 0;
}
```

Answer: “automatic” allocation of an integer. This integer has an address in memory. The address is in hexadecimal.
New operator: &

- & == address of
- This will tell you the address of a variable.

```c
#include <stdio.h>

int main()
{
    int A;
    A=4;
    printf("Value of A is %d, address of A is %p\n", A, &A);
    return 0;
}
```

```
Hanks-iMac:Downloads hank$ cat t.c
```

```
Hanks-iMac:Downloads hank$ gcc t.c
Hanks-iMac:Downloads hank$ ./a.out
Value of A is 4, address of A is 0x7fff56486bc8
```
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 $2^{31}-1$
- 534563E-6

- Store 4 things:
  - Sign of number (1 bit)
  - “Mantissa” (534563)
  - Sign of exponent (1 bit)
  - Value of exponent (6)

- Inexact!!
Arrays

```
#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.
Assignment:

1) Write a C program that sorts 100 numbers in an array.
2) You can sort however you want.
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 should too.
4) You can confirm this:
   a. Download my “correct_output” file
   b. Run your program as “./a.out > my_output”
   c. Run the diff program to difference the two:
      i. diff /path/to/correct_output /path/to/my_output
      ii. And put the proper paths in the place of /path/to
   d. If diff gives no output, they are identical.

This project will be graded by:

1) Running the diff program against your output (as per above)
2) Inspection of your code

If the diff program shows any difference, you will get less than half credit.

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

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 19 | 40 | 45 | 66 | 70 | 77 | 95 | 133 | 134 | 145 | 158 | 159 | 166 | 166 | 181 | 190 | 202 | 214 | 216 | 230 |
| 446 | 466 | 474 | 477 | 508 | 517 | 552 | 577 | 577 | 599 | 600 | 603 | 625 | 634 | 635 | 637 | 647 | 649 | 657 | 673 |
| 673 | 679 | 682 | 682 | 687 | 697 | 705 | 712 | 717 | 729 | 737 | 752 | 755 | 760 | 765 | 765 | 781 | 794 | 801 | 804 |
| 805 | 821 | 855 | 858 | 876 | 899 | 902 | 904 | 916 | 920 | 927 | 941 | 949 | 950 | 953 | 954 | 963 | 964 | 968 | 987 |
```c
#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 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)
#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**
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; ← this example is pointing to an automatic variable, not a dynamic variable
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

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

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

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

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...

```c
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...

```c
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
```
And here it goes wrong...

```c
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]);
}
```

```bash
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 Hz, 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 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>RECOMMENDED CUSTOMER PRICING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CORES</td>
<td>GHZ</td>
</tr>
<tr>
<td><strong>7290¹</strong></td>
<td>72</td>
<td>1.5</td>
</tr>
<tr>
<td>Best Performance/Node</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>7250</strong></td>
<td>68</td>
<td>1.4</td>
</tr>
<tr>
<td>Best Performance/Watt</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>7230</strong></td>
<td>64</td>
<td>1.3</td>
</tr>
<tr>
<td>Best Memory Bandwidth/Core</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>7210</strong></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.

---

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 #3

- 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 late 90s onward
  – 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!

Emergent Tech

Bitcoin heist with a twist: This time it's servers that were stolen

Icelandic cops cuff 11 on suspicion of data centre robberies

By Simon Sharwood, APAC Editor 5 Mar 2018 at 04:57

Icelandic police have cuffed 11 people in connection with four raids on data centres that targeted cryptocurrency mining equipment.

The raids started in December 2017 when three data centres were cracked in December. Another raid took place in January. 600 servers went missing in the heists.

Icelandic police kept the raids secret while they pursued their investigations. Those efforts culminated in 11 arrests and an appearance before the Reykjanes District Court last Friday. Two of the 11 were detained and the matter held over for another day.

The 600 servers are all still missing, the Associated Press reports. Which is no surprise: x86 kit is pretty generic. The real prize inside a bitcoin-mining rig is either GPUs, RAM or nicely fast solid-state disks. Those components are all tiny compared to servers and could probably have been posted out of Iceland piecemeal without much hassle.

Iceland has become something of a hub for demanding workloads like cryptocurrency mining because cheap energy and low ambient temperatures make it a low-cost destination to run data centres and the kit inside them. The nation also has a low crime rate. 😊
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

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.