Lecture 1: Introduction to Basic C Programs

Review and Preview

- Course outline
- Programming Environment
- Labs

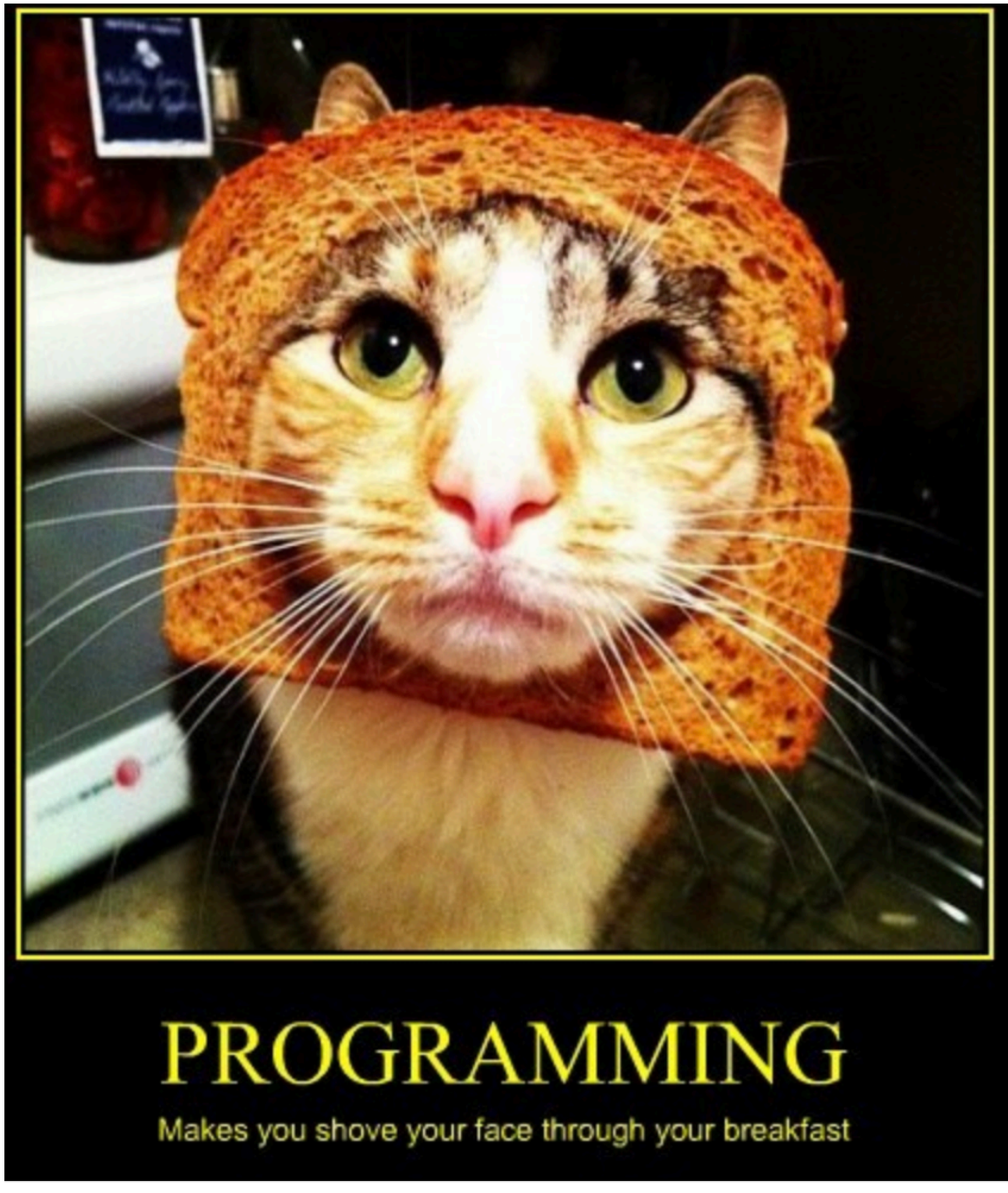
- Basic C program structure
- First C programs
- Shell, directories, pipes

K&R: Chapter 1
Sections 1.1 to 1.5

First C Programs

C Program #1: **Hello World**
(basic program structure, simple compilation and execution)

C Program #2: **Word Count**
*(if…else, while, **STDIN**, Unix pipes)*



Program #1: Hello World

Function: To print out a famous greeting and to demonstrate the way to structure a C program.

```
#include <stdio.h>

/*
 *  Name: mainline.c
 *  Author: Deb Stacey
 *  Last Update: May 29, 2019
 *  Function: This is a basic template for a C mainline.
 *  Compilation: gcc -ansi -o mainline mainline.c
 *  Execution: ./mainline
 */

int main ( int argc, char *argv[] ) {

   printf ( "Hello, world!\n" );

   return(0);

}
```

Preprocessor directive

Program comments

main()

statement / function

return code

Basic C Program Structure

- The mainline of a C program has 4 main components:
  - Preprocessor directives
  - `main()` function
  - Variables and statements
  - `return()` function

Preprocessor Directive: `#include`

- `#include <filename>` or `#include "filename"` tell the preprocessor to get a particular **header** file and add it to your code before compilation.

  - `<filename>` file is in a systems directory (on most Linux systems: `/usr/include`)

  - `"filename"` is a file in your directory

Preprocessor Directive: `#include`

- Typically they are used to add information about how standard **library** routines are described and called.

- The most commonly used one is **stdio.h** which describes the I/O (input/output) routines since I/O is **not** part of the C language.

Questions

- **Q1**: What is part of the C language and what is not?
- **Q2**: What are libraries?
- **Q3**: How do I find out which libraries to use?

Clue: https://fresh2refresh.com/c-programming/c-function/c-library-functions/

Why do we need header files?

- The standard library for the C programming language (**libc**), provides
    - macros
    - type definitions
    - functions

for tasks such as string handling, mathematical computations, input/output processing, memory management, and other operating system services [1].

Library: C Standard Library

- **Header**: `#include <stdio.h>`

- The C Standard Library provides many standard functions for **file** input and output.

- C abstracts all I/O operations into operations on streams of bytes: "input streams" and "output streams".

- There are 3 “pre-opened” file streams.

File Streams

- **STDIN**: input stream (typically the keyboard)
- **STDOUT**: output stream (typically the screen)
  - `printf()` writes to the **STDOUT** stream.
- **STDERR**: output stream for error messages

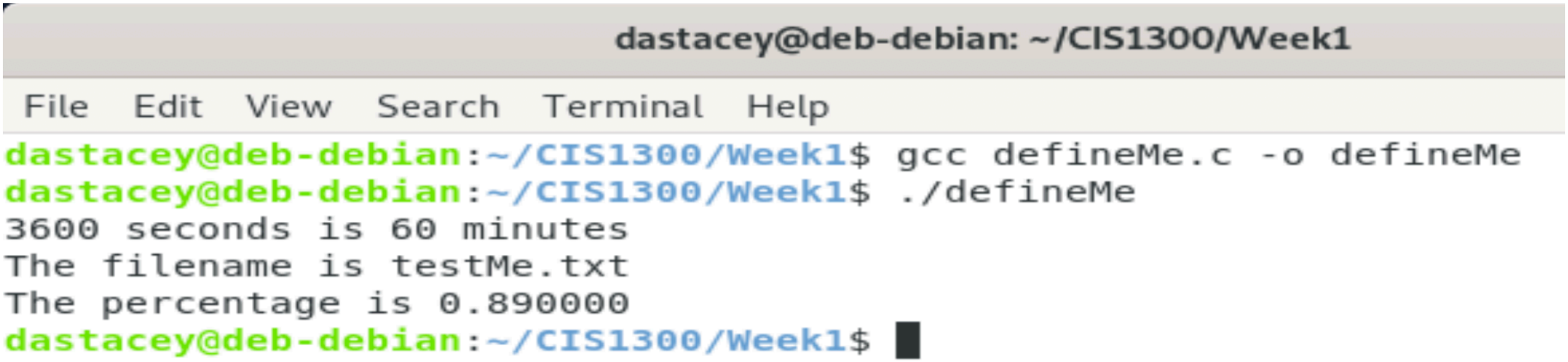
Preprocessor Directive: `#define`

- `#define` directive allows the definition of macros that are constants that are used throughout the code.
- These macros are constant - they cannot be changed.
- By convention the name of the constant is in uppercase but this is not required.
- Common error: there is no semi-colon at the end of the define statement [2].

Preprocessor Directive: *#define*

- `#define SECONDS 60` Number
- `#define FILENAME “testMe.txt”` String
- `#define PERCENTAGE ( a / 100.0 )` Expression
  - An expression must be enclosed in parentheses if it contains operators.

```c
#include <stdio.h>

#define SECONDS 60
#define FILENAME "testMe.txt"
#define PERCENTAGE ( a / 100.0 )

int main ( int argc, char *argv[] ) {

int minutes = 0;
int secondCount = 3600;
int a = 89;
float b = 0.0;

printf ( "%d seconds is %d minutes\n", secondCount, secondCount/SECONDS );
printf ( "The filename is %s\n", FILENAME );

b = PERCENTAGE;

printf ( "The percentage is %f\n", b);

return ( 0 );

}
```




```
$ gcc defineMe.c -o defineMe
```

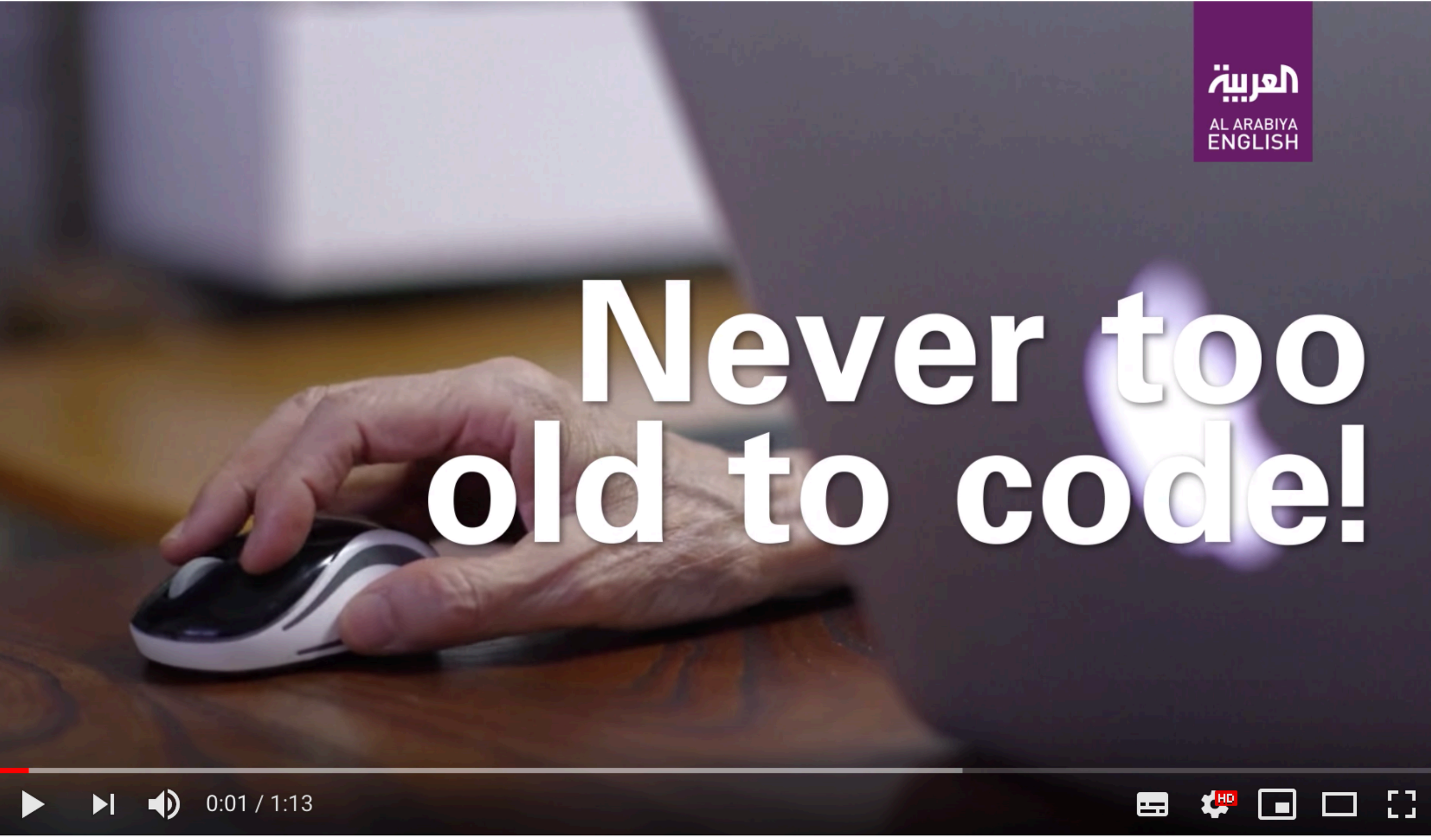
compiler

C source file

compiler argument -o: executable is put in file named defineMe

```
$ ./defineMe
```

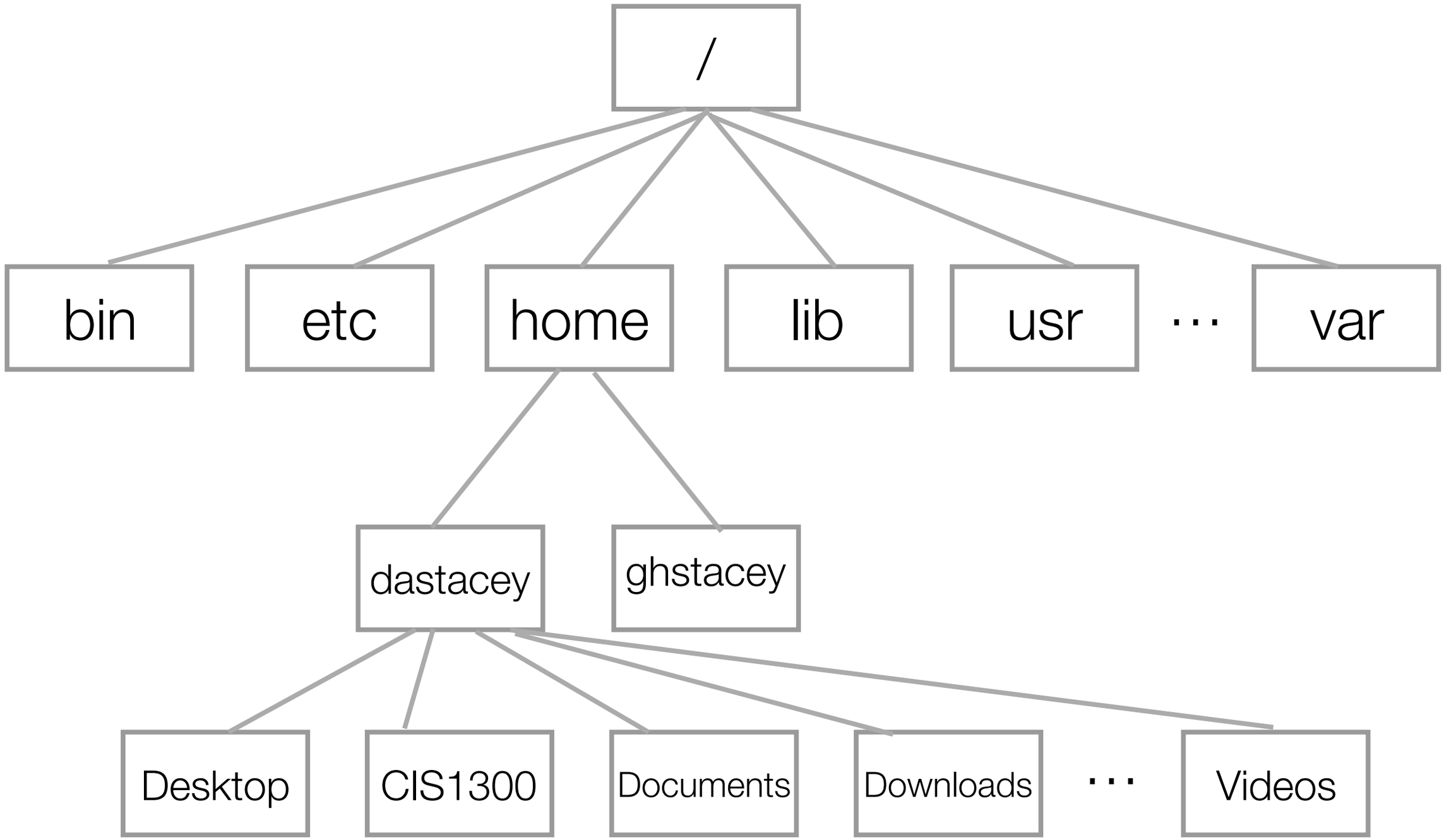
. (dot) represents the current directory and this tells the shell to run the executable defineMe



Break #1

https://www.youtube.com/watch?v=hr1xp3tVgUQ

UNIX/Linux Commands | Engaging the Shell



Unix/Linux Directory System

dastacey@deb-debian: ~/CIS1300/Assignment1

File Edit View Search Terminal Help

```
dastacey@deb-debian:~/CIS1300/Assignment1$ ls -al
total 88
drwxr-xr-x 2 dastacey dastacey  4096 Aug 15 21:30 .
drwxr-xr-x 6 dastacey dastacey  4096 Aug 15 21:57 ..
-rw-r--r-- 1 dastacey dastacey  1011 Aug 15 17:41 leapYear.c
-rw-r--r-- 1 dastacey dastacey  1336 Aug 15 17:43 leapYear.o
-rwxr-xr-x 1 dastacey dastacey 16608 Jul 18 17:29 mainline
-rw-r--r-- 1 dastacey dastacey   326 Jul 18 17:29 mainline.c
-rwxr-xr-x 1 dastacey dastacey 16608 Jul 18 16:27 test
-rw-r--r-- 1 dastacey dastacey    93 Jul 18 16:27 test.c
-rwxr-xr-x 1 dastacey dastacey 16728 Aug 15 17:44 testleap
-rw-r--r-- 1 dastacey dastacey   689 Aug 15 17:41 testleap.c
dastacey@deb-debian:~/CIS1300/Assignment1$
```

Current directory

Parent directory

~/CIS1300/Assignment1

/home/dastacey/CIS1300/Assignment1

Directory and File Listings and Commands

- pwd - the present working directory
- cd - change to a specific directory
- mkdir - making a new directory/sub-directory
- ls - listing files in the current working directory
  - ls -l *long listing format*
  - ls -tl *long listing format ordered by date/time*
  - ls -al *long listing format showing files starting with ‘.’*

dastacey@deb-debian: ~/CIS1300/Assignment2

File Edit View Search Terminal Help

```
dastacey@deb-debian:~$ pwd
/home/dastacey
dastacey@deb-debian:~$ ls
CIS1300  Documents  Mac    Pictures  Templates
Desktop  Downloads  Music  Public    Videos
dastacey@deb-debian:~$ cd CIS1300
dastacey@deb-debian:~/CIS1300$ ls -l
total 12
drwxr-xr-x 2 dastacey dastacey 4096 Aug 15 21:30 Assignment1
drwxr-xr-x 2 dastacey dastacey 4096 Aug 15 21:50 Assignment2
drwxr-xr-x 2 dastacey dastacey 4096 Aug  5 01:23 Week1
dastacey@deb-debian:~/CIS1300$ mkdir Assignment3
dastacey@deb-debian:~/CIS1300$ cd Assignment3
dastacey@deb-debian:~/CIS1300/Assignment3$ ls
dastacey@deb-debian:~/CIS1300/Assignment3$ cd ..
dastacey@deb-debian:~/CIS1300$ pwd
/home/dastacey/CIS1300
dastacey@deb-debian:~/CIS1300$ ls
Assignment1  Assignment2  Assignment3  Week1
dastacey@deb-debian:~/CIS1300$ cd Assignment2
dastacey@deb-debian:~/CIS1300/Assignment2$ ls -l
total 32
-rw-r--r-- 1 dastacey dastacey 25530 Aug 15 21:45 chapter1.txt
-rw-r--r-- 1 dastacey dastacey   759 Aug 15 21:50 toc.txt
dastacey@deb-debian:~/CIS1300/Assignment2$
```

More UNIX/Linux Commands

grep - print lines that match patterns

https://www.wikihow.com/Teach-Your-Cat-to-Sit

grep

- grep searches input files and looks for lines that match one or more patterns.

- The patterns can be simple or basic regular expressions.

- If grep finds a match then it writes it to STDOUT.

- Example: does a file contain the pattern ‘the’?

```
$ grep “the” example.txt
```

```
1  How long ago had it been that the staff members at the
2  AI Lab had welcomed the new printer with open arms?
3  Stallman wondered. The machine had been a donation from
4  the Xerox Corporation. A cutting edge prototype, it was
5  a modified version of the popular Xerox photocopier.
6  Only instead of making copies, it relied on software
7  data piped in over a computer network to turn that data
8  into professional looking documents. Created by
9  engineers at the world famous Xerox Palo Alto Research
10 Facility, it was, quite simply, an early taste of the
11 desktop printing revolution that would seize the rest
12 of the computing industry by the end of the decade.
```

```
$ grep “the” example.txt
```

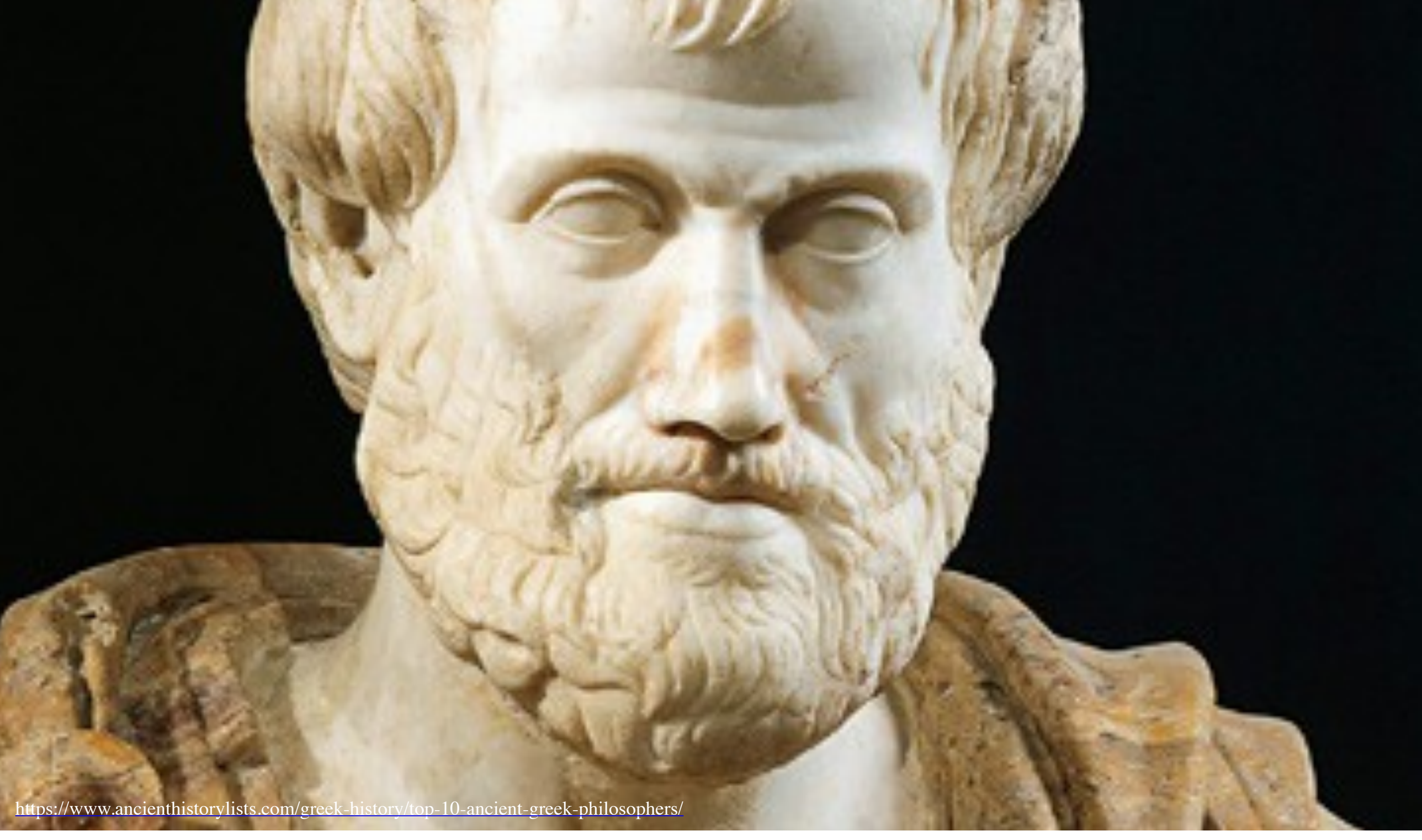
```
1  How long ago had it been that the staff members at the
2  AI Lab had welcomed the new printer with open arms?
3  Stallman wondered. The machine had been a donation from
4  the Xerox Corporation. A cutting edge prototype, it was
5  a modified version of the popular Xerox photocopier.
6  Only instead of making copies, it relied on software
7  data piped in over a computer network to turn that data
8  into professional looking documents. Created by
9  engineers at the world famous Xerox Palo Alto Research
10 Facility, it was, quite simply, an early taste of the
11 desktop printing revolution that would seize the rest
12 of the computing industry by the end of the decade.
```

```
$ grep “the” example.txt
```

```
1  How long ago had it been that the staff members at the
2  AI Lab had welcomed the new printer with open arms?
3  Stallman wondered. The machine had been a donation from
4  the Xerox Corporation. A cutting edge prototype, it was
5  a modified version of the popular Xerox photocopier.
6  Only instead of making copies, it relied on software
7  data piped in over a computer network to turn that data
8  into professional looking documents. Created by
9  engineers at the world famous Xerox Palo Alto Research
10 Facility, it was, quite simply, an early taste of the
11 desktop printing revolution that would seize the rest
12 of the computing industry by the end of the decade.
```

```
$ grep -i “the” example.txt
```

```
1  How long ago had it been that the staff members at the
2  AI Lab had welcomed the new printer with open arms?
3  Stallman wondered. The machine had been a donation from
4  the Xerox Corporation. A cutting edge prototype, it was
5  a modified version of the popular Xerox photocopier.
6  Only instead of making copies, it relied on software
7  data piped in over a computer network to turn that data
8  into professional looking documents. Created by
9  engineers at the world famous Xerox Palo Alto Research
10 Facility, it was, quite simply, an early taste of the
11 desktop printing revolution that would seize the rest
12 of the computing industry by the end of the decade.
```



https://www.ancienthistorylists.com/greek-history/top-10-ancient-greek-philosophers/

UNIX Philosophy

Why use UNIX to teach programming?

The Basics of the UNIX Philosophy

- Write programs that do **one** thing and do it **well**.
- Write programs to work **together**.
- Write programs to handle **text** streams, because that is a universal interface.

Doug McIlroy, the inventor of UNIX pipes and Ken Thompson, co-developer of UNIX

The Philosophy according to Brian Kernighan

1. Everything is a **file**.

2. Small, single-purpose programs (**modularity**).

3. Ability to chain programs together to perform complex tasks (**piping**).

4. Avoid captive user interfaces - most UNIX programs are non-interactive (requiring arguments instead of user input) which makes them useful within **scripts**.

https://www.brandt.us/commercial-plumbing-contractors/hygienic-pipes-fittings/

Unix Pipes

How to take advantage of STDIN and STDOUT

UNIX Pipes

- A pipe redirects output (**STDOUT**) from one program to the input (**STDIN**) of another program.

- This direct connection between commands/programs/processes allows for **simultaneously** operation.

- Data is transferred **continuously**. The other approach would be to transmit the data from one program to another through **temporary** text files.

Unix Pipes

- The direction is one way (from left to right) and the sequence of programs is referred to as a pipeline.

```
ls -l | grep " Jul "
```

- `ls -l` will list the files in the current directory using the long format

- `grep " Jul "` will find and print out all lines that contain the string " Jul " (*probably* all files created in Jul)

- Will this always find the files created/updated in July?

UNIX Pipes

```
dastacey@deb-debian: ~/CIS1300
File  Edit  View  Search  Terminal  Help
dastacey@deb-debian:~/CIS1300$ ls -l
total 52
-rwxr-xr-x 1 dastacey dastacey 16608 Jul 18 17:29 mainline
-rw-r--r-- 1 dastacey dastacey   326 Jul 18 17:29 mainline.c
-rwxr-xr-x 1 dastacey dastacey 16608 Jul 18 16:27 test
-rw-r--r-- 1 dastacey dastacey    93 Jul 18 16:27 test.c
drwxr-xr-x 2 dastacey dastacey  4096 Aug  4 23:11 Week1
dastacey@deb-debian:~/CIS1300$
dastacey@deb-debian:~/CIS1300$ ls -l | grep " Jul "
-rwxr-xr-x 1 dastacey dastacey 16608 Jul 18 17:29 mainline
-rw-r--r-- 1 dastacey dastacey   326 Jul 18 17:29 mainline.c
-rwxr-xr-x 1 dastacey dastacey 16608 Jul 18 16:27 test
-rw-r--r-- 1 dastacey dastacey    93 Jul 18 16:27 test.c
dastacey@deb-debian:~/CIS1300$
```

Remember using grep to find all the occurrences of ‘the’ in a file?

How could you find the number of matches in a file?

How could you find the number of matches?

- Using **`wc`** gives you the number of lines found, but what if you want to know how many matches were found?

```
dastacey@deb-debian:~/CIS1300$ grep 'the' example.txt | wc
      8      76     431
dastacey@deb-debian:~/CIS1300$ grep -i 'the' example.txt | wc
      9      85     487
```

- The command '**`man`**' is your friend!

```
-n, --line-number
       Prefix each line of output with the 1-based line number within  its
       input file.


-o, --only-matching
       Print  only  the matched (non-empty) parts of a matching line, with
       each such part on a separate output line.
```

How could you find the number of matches?

How could you find the number of matches?

Catastrophe In Space

In space, nobody can hear you … make assumptions…

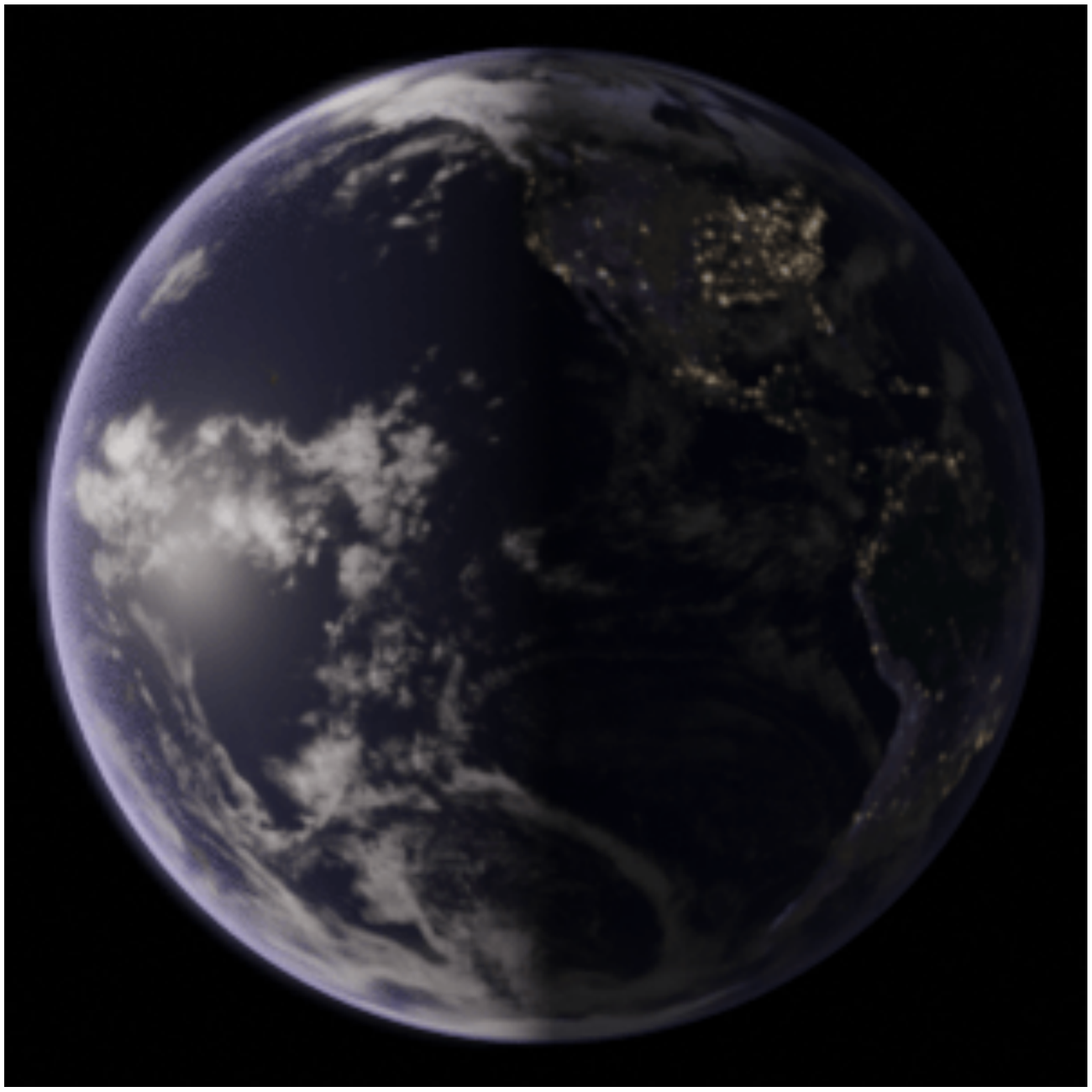
Assumptions and Details: Gemini V Story

- Gemini V was the first American space flight to top a duration record from the Soviet Union.
- Mission was incredibly boring, spacecraft just drifting to conserve fuel most of the time.
- Launched: 1965-08-21
- Returned: 1965-08-29
- Number crew: 2
- Duration: 7.96 days

The Gemini V Programming Error

- Gemini V landed 190 hours 55 minutes 14 seconds after launch.
- But it landed **130 kilometers** short of the planned landing site.
- The computer had worked perfectly but the programmers had not!
- One of the astronauts (Cooper) compensated for what he recognized as an erroneous reading so that they came down closer to the retrieval ship than they would have if they had of relied on the computer!
- http://www.astronautix.com/g/gemini5.html

Question



How many hours are there in a day?

How long does it take for the Earth to complete one rotation?

The Gemini V Programming Error

- Earth's rotation rate is **360.98** degrees per day.

- But, in the program guiding the flight computer, someone assumed that one day = 24 hours. (the two decimal-place numbers were left off).

- If our definition of a day was based on one complete rotation of the Earth on its axis — a **360** degree spin — then a day would be:

23 hours, 56 minutes, and 4 seconds

- Almost 4 minutes shorter than our 24-hour standard day!

Sometimes you think you know something…

- Solar day: the amount of time it takes for the Sun to move through the sky and return to roughly the same spot - 24 hours.

- Sidereal Day: the amount of time it takes for the Earth to turn once on its axis – the 23 hours, 56 minutes, 4.0916 seconds.

- But actually … the measurement of the Earth's rotation is affected by the Earth's precession, tidal locking and recent earthquakes.

Sometimes you think you know something…

Earth's Precession (wobbling of the Earth’s axis) makes the sidereal day 0.0084 seconds shorter.

Tidal Locking (tidal interactions with the Moon) has increased the length of a day on Earth by about 1.7 milliseconds over the last 100 years.

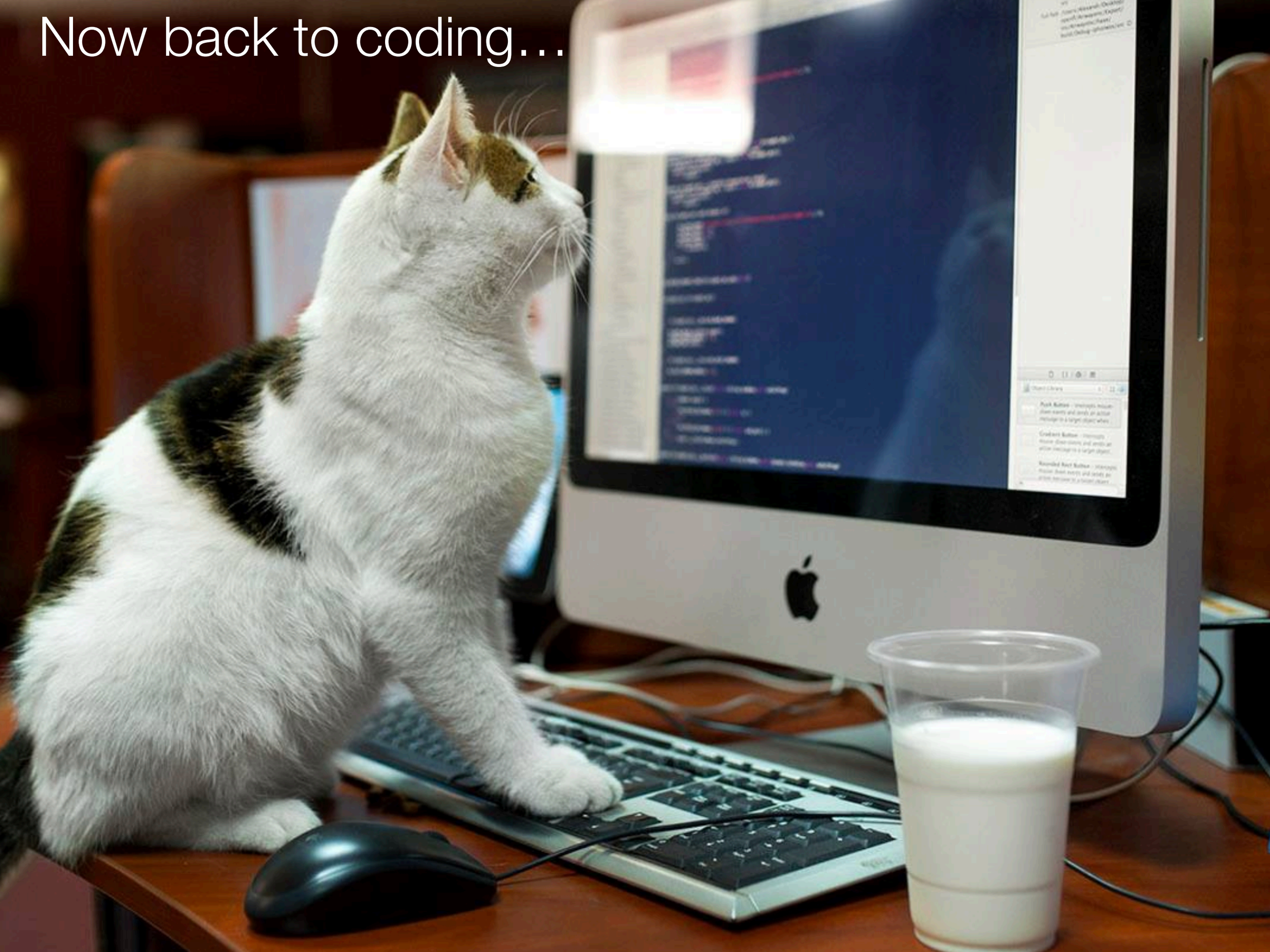
Powerful earthquakes can change the Earth’s rotation time by a few microseconds. Even melting glaciers can slow down the Earth’s rotation speed.

References

[1] https://www.universetoday.com/123218/how-long-is-a-day-on-earth-2/

[2] https://medium.com/the-philipendium/a-day-is-not-24-hours-c36ee96078c6

Now back to coding…



Program #2: Counting Characters

Function: To read in a text file from STDIN and count the number of characters, words, and lines and report on this.

K&R - Section 1.5.4 (page 20)

Program Design / Algorithm

- Read in the input, one character at a time until there is no more input and each time a character is read in, *increment* the **character counter**.

- If the character is a *Newline* character, *increment* the **line counter**.

- If the character is a Space, Newline, or Tab then you are now **outside** of a word,

- Else if the character is NOT a Space, Newline, or Tab and you were outside of a word then you are now inside of a word and you increment the **word counter**.

	T	h	i	s		i	s		o	n	e	.	\n
OUT	IN	IN	IN	IN	OUT	IN	IN	OUT	IN	IN	IN	IN	OUT
Char	1	2	3	4	5	6	7	8	9	10	11	12	13
Word	1	1	1	1	1	2	2	2	3	3	3	3	3
Line	0	0	0	0	0	0	0	0	0	0	0	0	1

	L	a	s	t		l	i	n	e	.	\n
	IN	IN	IN	IN	OUT	IN	IN	IN	IN	IN	OUT
Char	14	15	16	17	18	19	20	21	22	23	24
Word	4	4	4	4	4	5	5	5	5	5	5
Line	1	1	1	1	1	1	1	1	1	1	2

```
#include <stdio.h>

#define IN  1   /* inside a word */
#define OUT 0   /* outside a word */

/*
 * Program Name: wcount.c
 * Author(s): Deb Stacey
 * Date of Last Update: August 4, 2019
 * Purpose: reads input from stdin and counts the lines, words, and
 *          sentences and prints these counts to stdout.
 *          Modern version of code on page 20 of K&R (section 1.5.4).
 *          Similar to Linux command wc.
 * Testing: testfile(s): testWC1.txt
 *          protocol: compare to wc results
 * Change Log:
 */
```

a)

b)

c)

a) Preprocessor directive: standard I/O include
b) Preprocessor directive: define statements
c) Comments

```
int main ( int argc, char *argv[] ) {
    char c;
    int numLines = 0;   /* number of lines */
    int numWords = 0;   /* number of words */
    int numChars = 0;   /* number of characters */

    int state = OUT;

    while ( (c = getchar()) != EOF ) {
        numChars++;
        if ( c == '\n' ) {
            numLines++;
        }
        if ( c == ' ' || c == '\n' || c == '\t' ) {
            state = OUT;
        } else if (state == OUT) {
            state = IN;
            numWords++;
        }
    }
    printf ( "%d %d %d\n", numLines, numWords, numChars );

    return ( 0 );
}
```

Declare and initialize all variables

Read input from STDIN, one character at a time until end of file (EOF) is detected

Update char count

Update line count

Newline: \n

Tab: \t

Update word count

Control: `if…else`

- The most basic program control structure is `if…else`.
- Basic structure:

```
if ( boolean expression ) {
   Statements to execute if boolean is true
} else {
   Statements to execute if boolean is false
}
```

Control: `if…else` boolean expressions

- What is a boolean expression?
  - These expressions evaluate to either **true** or **false**.
    - Example: `a > 10`
- There are also boolean operators that can be used with multiple boolean expressions [3].

Control: `if…else` boolean operators

- **OR** - true when either boolean expression is **true**

```
if ( a > 10 || b > 10 ) …
```

- **AND** - true when all boolean expressions are **true**

```
if ( a > 10 && b > 10 ) …
```

- **NOT** - true when boolean expression is false

```
if ( !( a > 10 && b > 10 ) ) …
```

Control: `if…else` Chains

```
if ( boolean expression 1 ) {
   Statements to execute if expression 1 is true
} else if ( boolean expression 2 ) {
   Statements to execute if expression 2 is true
} else {
   Statements to execute if none are true
}
```

More UNIX/Linux Commands

---

cat - print out a file to STDOUT
more - a paginated version of cat
wc - char, word, line counts
od - dump files in octal and other forms

https://www.wikihow.com/Teach-Your-Cat-to-Sit

`cat` - printing a file to STDOUT

- `cat` - concatenate files and print on the standard output
- `more` - print file to standard output with pauses

dastacey@deb-debian: ~/CIS1300/Assignment2

File Edit View Search Terminal Help

```
dastacey@deb-debian:~/CIS1300/Assignment2$ cat toc.txt
Free As in Freedom: Richard Stallman's Crusade for Free Software.
By Sam Williams
Produced under the Free Documentation License

Table of Contents

Chapter 1 For Want of a Printer
Chapter 2 2001: A Hacker's Odyssey
Chapter 3 A Portrait of the Hacker as a Young Man
Chapter 4 Impeach God
Chapter 5 Small Puddle of Freedom
Chapter 6 The Emacs Commune
Chapter 7 A Stark Moral Choice
Chapter 8 St. Ignucius
Chapter 9 The GNU General Public License
Chapter 10 GNU/Linux
Chapter 11 Open Source
Chapter 12 A Brief Journey Through Hacker Hell
Chapter 13 Continuing the Fight
Chapter 14 Epilogue:
Chapter 15 Appendix A : Terminology
Chapter 16 Appendix B Hack, Hackers, and Hacking
Chapter 17 Appendix C GNU Free Documentation License (GFDL)
dastacey@deb-debian:~/CIS1300/Assignment2$
```

dastacey@deb-debian: ~/CIS1300/Assignment2

File Edit View Search Terminal Help

```
For Want of a Printer

I fear the Greeks. Even when they bring gifts.
---Virgil The Aeneid

The new printer was jammed, again.

Richard M. Stallman, a staff software programmer at the
Massachusetts Institute of Technology's Artificial
Intelligence Laboratory (AI Lab), discovered the
malfunction the hard way. An hour after sending off a
50-page file to the office laser printer, Stallman, 27,
broke off a productive work session to retrieve his
documents. Upon arrival, he found only four pages in
the printer's tray. To make matters even more
frustrating, the four pages belonged to another user,
meaning that Stallman's print job and the unfinished
portion of somebody else's print job were still trapped
somewhere within the electrical plumbing of the lab's
computer network.

Waiting for machines is an occupational hazard when
you're a software programmer, so Stallman took his
frustration with a grain of salt. Still, the difference
between waiting for a machine and waiting on a machine
--More--(4%)
```

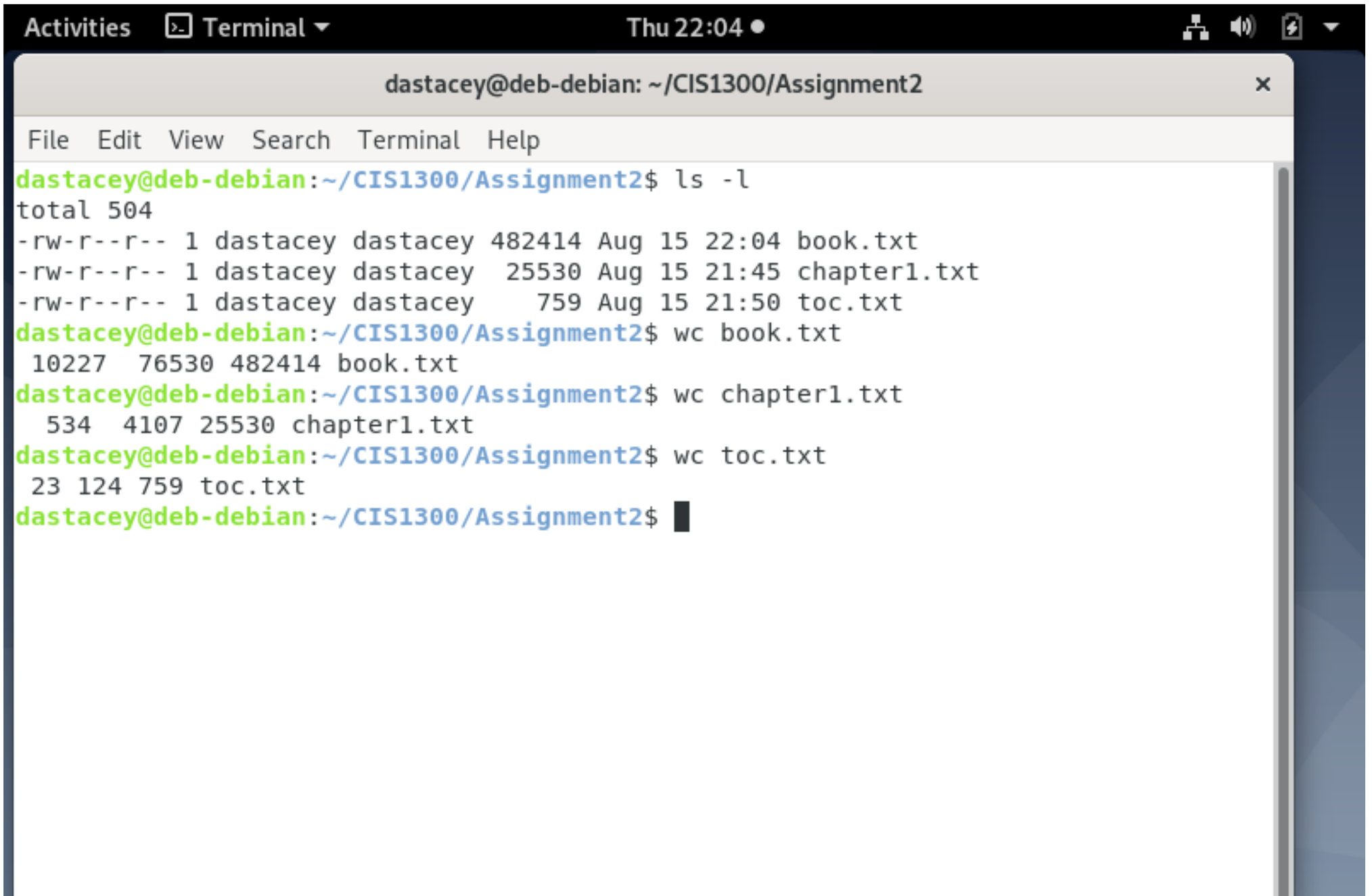
is a sizable one. It wasn't the first time he'd been
forced to stand over the printer, watching pages print
out one by one. As a person who spent the bulk of his
days and nights improving the efficiency of machines
and the software programs that controlled them,
Stallman felt a natural urge to open up the machine,
look at the guts, and seek out the root of the problem.

Unfortunately, Stallman's skills as a computer
programmer did not extend to the mechanical-engineering
realm. As freshly printed documents poured out of the
machine, Stallman had a chance to reflect on other ways
to circumvent the printing jam problem.

How long ago had it been that the staff members at the
AI Lab had welcomed the new printer with open arms?
Stallman wondered. The machine had been a donation from
the Xerox Corporation. A cutting edge prototype, it was
a modified version of the popular Xerox photocopier.
Only instead of making copies, it relied on software
data piped in over a computer network to turn that data
into professional-looking documents. Created by
engineers at the world-famous Xerox Palo Alto Research
Facility, it was, quite simply, an early taste of the
desktop-printing revolution that would seize the rest
--More--(8%)

`wc` - character, word and line counts

- print newline, word, and byte counts for each file



How are characters represented?

- Characters are stored in 8-bits.
- A bit is a binary digit (0 or 1).
- 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
- 128 64 32 16 8 4 2 1
- 0 1 0 1 1 0 0 0
- 64 + 16 + 8 = 88
- X

0		32		64	@	96	‘	128		160		192	À	224	à
1		33	!	65	A	97	a	129		161	¡	193	Á	225	á
2		34	"	66	B	98	b	130		162	¢	194	Â	226	â
3		35	#	67	C	99	c	131		163	£	195	Ã	227	ã
4		36	$	68	D	100	d	132		164	¤	196	Ä	228	ä
5		37	%	69	E	101	e	133		165	¥	197	Å	229	å
6		38	&	70	F	102	f	134		166	¦	198	Æ	230	æ
7		39	’	71	G	103	g	135		167	§	199	Ç	231	ç
8		40	(	72	H	104	h	136		168	¨	200	È	232	è
9		41	)	73	I	105	i	137		169	©	201	É	233	é
10		42	*	74	J	106	j	138		170	ª	202	Ê	234	ê
11		43	+	75	K	107	k	139		171	«	203	Ë	235	ë
12		44	,	76	L	108	l	140		172	¬	204	Ì	236	ì
13		45	-	77	M	109	m	141		173	-	205	Í	237	í
14		46	.	78	N	110	n	142		174	®	206	Î	238	î
15		47	/	79	O	111	o	143		175	¯	207	Ï	239	ï
16		48	0	80	P	112	p	144		176	°	208	Ð	240	ð
17		49	1	81	Q	113	q	145		177	±	209	Ñ	241	ñ
18		50	2	82	R	114	r	146		178	²	210	Ò	242	ò
19		51	3	83	S	115	s	147		179	³	211	Ó	243	ó
20		52	4	84	T	116	t	148		180	´	212	Ô	244	ô
21		53	5	85	U	117	u	149		181	µ	213	Õ	245	õ
22		54	6	86	V	118	v	150		182	¶	214	Ö	246	ö
23		55	7	87	W	119	w	151		183	·	215	×	247	÷
24		56	8	88	X	120	x	152		184	¸	216	Ø	248	ø
25		57	9	89	Y	121	y	153		185	¹	217	Ù	249	ù
26		58	:	90	Z	122	z	154		186	º	218	Ú	250	ú
27		59	;	91	[	123	{	155		187	»	219	Û	251	û
28		60	<	92	\	124	|	156		188	¼	220	Ü	252	ü
29		61	=	93	]	125	}	157		189	½	221	Ý	253	ý
30		62	>	94	^	126	~	158		190	¾	222	Þ	254	þ
31		63	?	95	_	127		159		191	¿	223	ß	255	ÿ

How to Store an ASCII Character in C

- `char letter;`
- `letter = ‘a’;`

- `int letter;`
- `letter = ‘a’;`
- `letter = letter + 5;`

dastacey@deb-debian: ~/CIS1300/Week1

File Edit View Search Terminal Help

```c
#include <stdio.h>

/*
 *  Name: characters.c
 *  Author: Deb Stacey
 *  Last Update: August 18, 2019
 *  Function: Illustrating character storage.
 *  Compilation: cc -ansi -o characters characters.c
 *  Execution: ./characters
 */

int main ( int argc, char *argv[] ) {
   char cletter;
   int iletter;

   cletter = 'a';
   printf ( "cletter = %c\n", cletter );

   iletter = 'a';
   iletter = iletter + 5;
   printf ( "iletter = %c\n", iletter );
   printf ( "iletter = %d\n", iletter );

   return ( 0 );
}
```

dastacey@deb-debian: ~/CIS1300/Week1

File Edit View Search Terminal Help

```
dastacey@deb-debian:~/CIS1300/Week1$ gcc -ansi -o characters characters.c
dastacey@deb-debian:~/CIS1300/Week1$ ./characters
cletter = a
iletter = f
iletter = 102
dastacey@deb-debian:~/CIS1300/Week1$
```

dastacey@deb-debian: ~/CIS1300/Week1

File Edit View Search Terminal Help

```
 *  Compilation: cc -ansi -o characters characters.c
 *  Execution: ./characters
 */

int main ( int argc, char *argv[] ) {
   char cletter, c2letter;
   int iletter;

   cletter = 'a';
   c2letter = cletter + 5;
   printf ( "cletter = %c\n", cletter );
   printf ( "c2letter = %c\n", c2letter );

   iletter = 'a';
   iletter = iletter + 5;
   printf ( "iletter = %c\n", iletter );
   printf ( "iletter = %d\n", iletter );

   return ( 0 );
}


dastacey@deb-debian:~/CIS1300/Week1$ ./characters
cletter = a
c2letter = f
iletter = f
iletter = 102
dastacey@deb-debian:~/CIS1300/Week1$
```

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
0	0	NUL	16	10	DLE	32	20	(space)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

A - Z: 65 - 90

a - z : 97 - 122

How could you change upper case characters to lower case characters?

https://ascii.cl/

Looping - repeat, repeat, ...

The `while` Loop

Loops: Repeating Actions

- Another basic tenet of programming is the repetition of actions - looping [4].

- A while loop executes while some condition is true.

```
while ( boolean expression ) {

   Statements to execute if the boolean
expression is true

 }
```

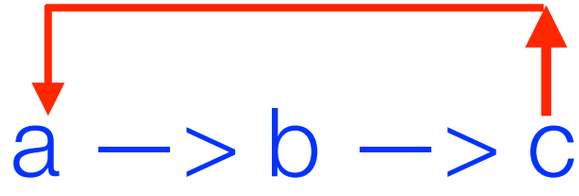
Loops: `while`

```
   a = 0;
a) while ( a < 10 ) {
b)     printf ( "%d \n", a );
b)     a = a + 2;
c) }
```

Test or Condition

Possible error

Output:
0
2
4
6
8



Loops: `while` Best Practices

- What if you wanted the previous example to print the numbers from 0 to 10?

```
while ( a <= 10 ) {
```

OR

```
while ( a < 11 ) {
```

- It is better to always use an **inequality** - consistency will help prevent errors in future code particularly when you make updates/changes.

References

[1] https://en.wikipedia.org/wiki/C_standard_library

[2] https://www.techonthenet.com/c_language/constants/create_define.php

[3] https://www.tutorialspoint.com/cprogramming/c_logical_operators.htm

[4] https://www.programiz.com/c-programming/c-do-while-loops