Week 3 Lecture 1: Data in C

Review and Preview

- if else branching, looping (while, for)
- wcount.c program
- Problem solving - Assign 1

- Types in C
- Storing data in C

K&R: Chapter 1
Sections 1.1 to 1.6
Sections 2.1 to 2.6

UNIX Command of the Day: `ssh`

- `ssh` or Secure Shell allows you to login to a remote computer or server using secure encrypted communications.
- Once you have logged in to the server it is like you are in a `terminal` on that remote machine.
- This command allows me to login as dastacey on the remote computer named linux.socs.uoguelph.ca

```
$ ssh dastacey@linux.socs.uoguelph.ca
```

Types in C

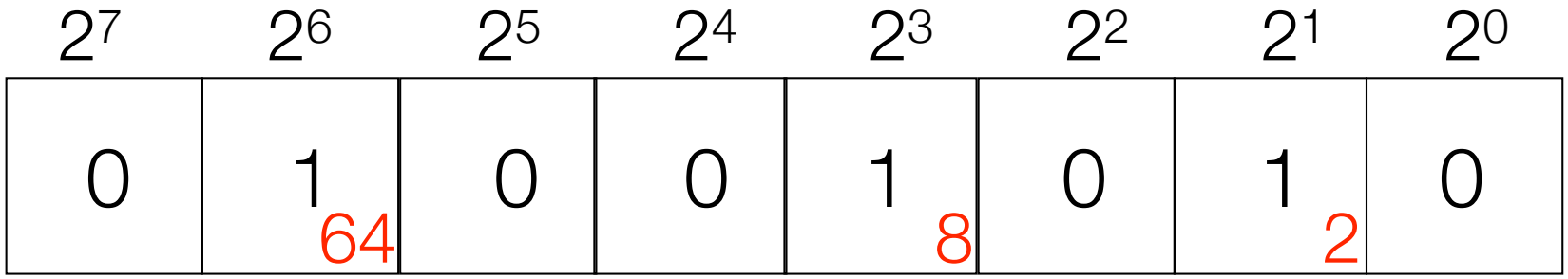
Storing Data in Variables

Types in C

- **Variables** are the basic data objects used in a program.
- **Declarations** list the variables to be used, give them a name and state their **type**.
- Often declarations also include the initial values of the variables. This is called **initialization**.
- There are very few basic types in C: `int, char, float, double`.

Basic Types

- `char` is a single **byte**, capable of holding one **character**, or letter, from a **character set**.
- What is a **byte**?
  - Everything in memory is represented by **bits** or binary digits - either a **1** or a **0**.
  - Bits are grouped into **bytes** - a byte is **8** bits long.



Basic Types: Numbers

- `int` is an integer or whole number.
- `float` is a single-precision floating point number.
- `double` is a double-precision floating point number.
- One of the most important things about number variables is their "**size**" or the number of bytes used to represent them.
- Size will determine the **range** of numbers that can be represented.

Basic Types: `int`

- The other important characteristic of integers is whether they are **`signed`** or **`unsigned`**.
- If an integer variable is signed then it can contain negative as well as positive numbers.
  - The range of a signed integer (4 bytes or 32 bits) is -2,147,483,648 to 2,147,483,647.
  - The range of an unsigned integer is 0 to 4,294,967,295.

Basic Types: `int`

- **`unsigned int`** - represented as an ordinary binary number, e.g. 0000 0101 = 5

- **`signed int`** - positive numbers represented as an ordinary binary number

- **`signed int`** - negative numbers represented in 2's complement format

  - 0000 0101 - change 0's to 1's and 1's to 0's and add 1

sign bit

  - 1111 1011 = -5

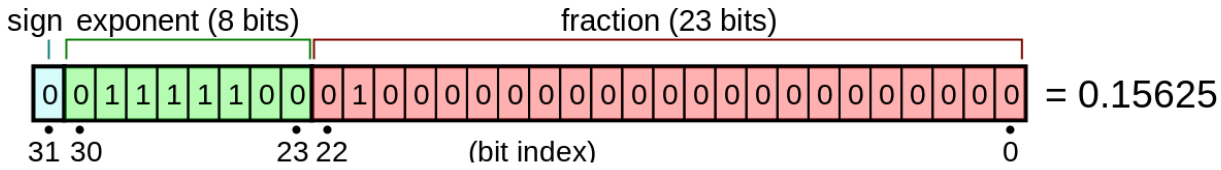
https://en.wikipedia.org/wiki/Two%27s_complement

```
#include <stdio.h>
int main (int argc, char *argv[]) {
   int i,cnt;

   i = 2147483645;  /* max positive value of int minus 2 */
   for ( cnt=1; cnt<4; cnt++ ) {
      i++;
      printf ( "2147483645 + %d = %d\n", cnt, i );
   }
   printf ( "\n" );

   i = -2147483646;  /* max negative value of int plus 2 */
   for ( cnt=1; cnt<4; cnt++ ) {
      i--;
      printf ( "-2147483646 - %d = %d\n", cnt, i );
   }

   return ( 0 );
}
```

```
$ ./outofrange
2147483645 + 1 = 2147483646
2147483645 + 2 = 2147483647
2147483645 + 3 = -2147483648

-2147483646 - 1 = -2147483647
-2147483646 - 2 = -2147483648
-2147483646 - 3 = 2147483647
$
```

Basic Types: `float`

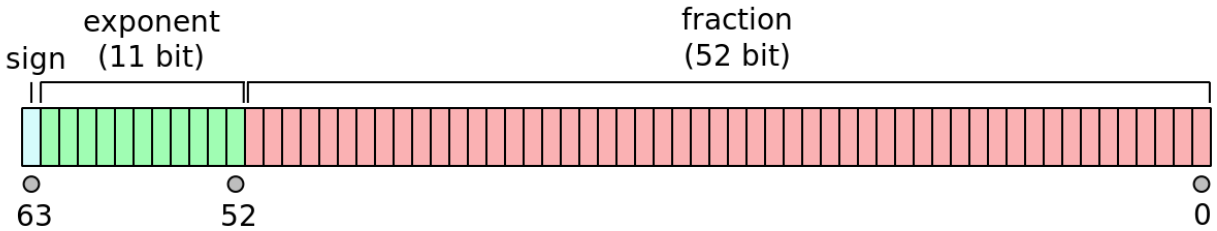
- Represents floating point numbers and has 6 decimal digits of precision
- Range of 1.2E-38 to 3.4E+38
- 32-bit IEEE 754 single precision floating point number
  - 1 bit for the sign
  - 8 bits for exponent
  - 23 bits for the value/precision



https://en.wikipedia.org/wiki/Single-precision_floating-point_format

Basic Types: `Double`

- Represents floating point numbers and has 15 decimal digits of precision.
- Range of 2.3E-308 to 1.7E+308
- 64-bit IEEE 754 double precision floating point
  - 1 bit for the sign
  - 11 bits for exponent
  - 52 bits for the value/precision



https://en.wikipedia.org/wiki/Double-precision_floating-point_format

```
#include <stdio.h>
int main( int argc, char *argv[] ) {
    printf("Size of a char is %lu byte\n\n", sizeof(char));

    printf("Size of an integer is %lu bytes\n", sizeof(int));
    printf("    short int = %lu bytes\n", sizeof(short int));
    printf("    unsigned short int = %lu bytes\n",
sizeof(unsigned short int));
    printf("    long int = %lu bytes\n", sizeof(long int));
    printf("    unsigned long int = %lu bytes\n\n",
sizeof(unsigned long int));

    printf("Size of a float is %lu bytes\n", sizeof(float));
    printf("Size of a double is %lu bytes\n", sizeof(double));

    return ( 0 );
}
```

sizeof() operator - when used with a data type, it returns the amount of memory allocated to that data type as an ***`unsigned long int`****.*

```
$ ./sizeof
```
Size of a char is 1 byte

Size of an integer is 4 bytes
    short int = 2 bytes
    unsigned short int = 2 bytes
    long int = 8 bytes
    unsigned long int = 8 bytes

Size of a float is 4 bytes
Size of a double is 8 bytes
```
$
```

Basic Types: `char`

- `char` stores a character/letter in 1 byte of storage.
- It really is a type of integer because C represents characters as integer numbers.
- These integer numbers are mapped to a character using a code.
  - ASCII (American Standard Code for Information Interchange) - limited to 255 mappings
  - Unicode - larger representation to allow for 1000's of mappings (many different alphabets can be represented)

What is ASCII?

Or how are characters represented?

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
- The character **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;` ← But you can also do this!
- `letter = 'a';`
- `letter = letter + 5;`

dastacey@deb-debian: ~/CIS1300/Week1

File Edit View Search Terminal Help

```
#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 );
}
```

27,0-1 All

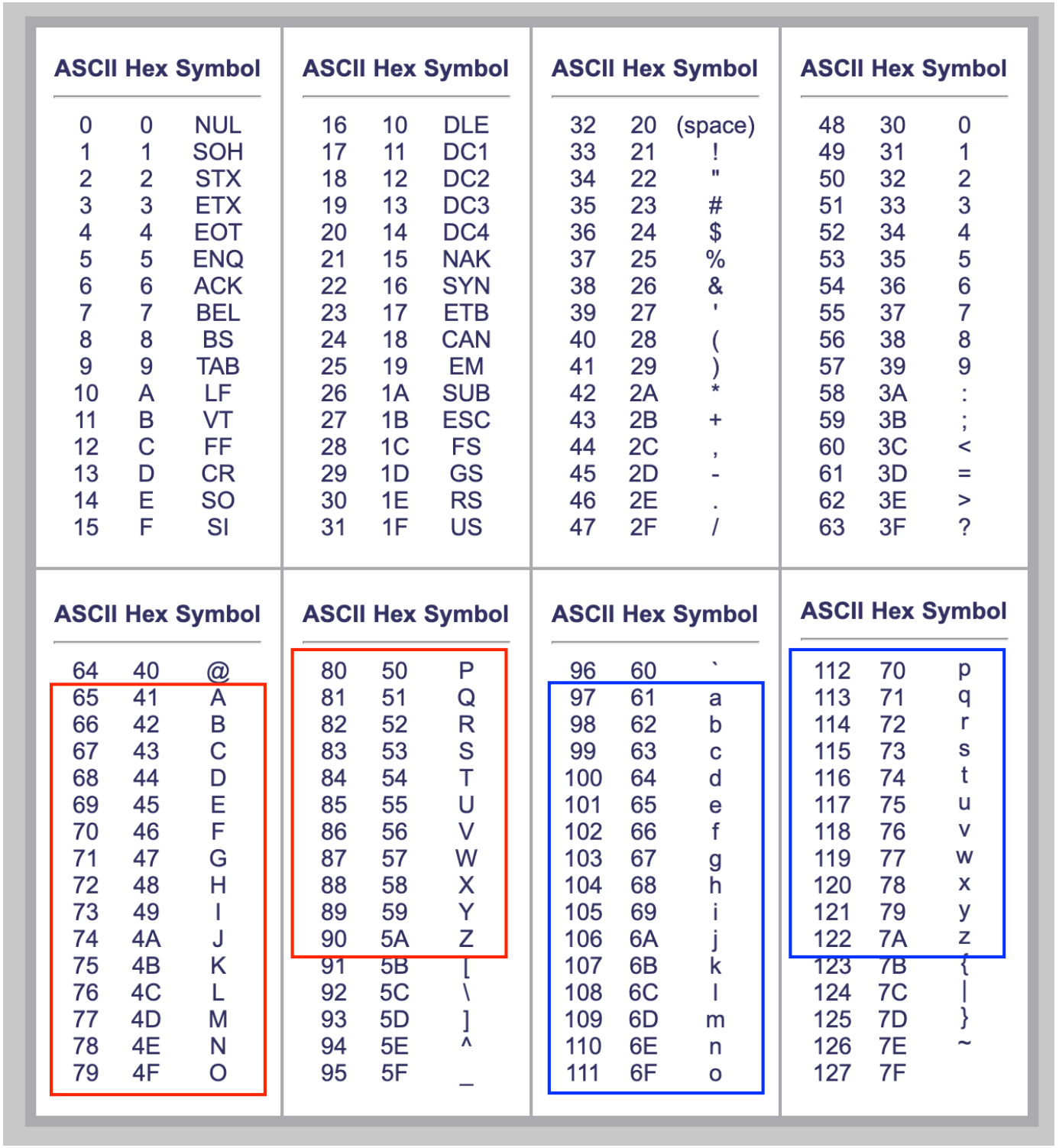
Activities Terminal Thu 22:39

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$
```

‘a’ + 5 = 97 + 5 = 102 = ‘f’



ASCII	Hex	Symbol
0	0	NUL
1	1	SOH
2	2	STX
3	3	ETX
4	4	EOT
5	5	ENQ
6	6	ACK
7	7	BEL
8	8	BS
9	9	TAB
10	A	LF
11	B	VT
12	C	FF
13	D	CR
14	E	SO
15	F	SI

ASCII	Hex	Symbol
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US

ASCII	Hex	Symbol
32	20	(space)
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/

ASCII	Hex	Symbol
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

ASCII	Hex	Symbol
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O

ASCII	Hex	Symbol
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

ASCII	Hex	Symbol
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o

ASCII	Hex	Symbol
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	

A - Z: 65 - 90

a - z : 97 - 122

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

https://ascii.cl/

```
debs@deb-socs:~/Programs$ more ascii.c
#include <stdio.h>

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

   char word[9] = { "aardvark" };

   int delta = 32;
   int i = 0;

   printf ( "%s\n", word ) ;

   for ( i=0; i<8; i++ ) {
      word[i] = word[i] - delta;
   }

   printf ( "%s\n", word ) ;

   return ( 0 );

}
debs@deb-socs:~/Programs$ ./ascii
aardvark
AARDVARK
debs@deb-socs:~/Programs$
```

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.

You know how to use **`grep`** to find all the lines where ‘the’ occurs 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 grep`**' 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?