CSE 29
Lecture 2 Summary

January 8, 2026

📣 Logistical Announcements

- Exams
  - We will have exams on Week 4, Week 8, and Week 10
  - Please schedule your **Week 4** and **Week 8** on us.prairietest.com
  - We will be doing the **Week 10** exam during Week 10 lab time
  - On finals week, we will have makeups for all 3 exams

💡 **Tip**
In some of the slides, there will be additional information in the **speaker notes**. We will try our best to indicate that there is more in the speaker notes, but just in case we forget, don’t forget to check!

🧠 Review Questions

1. What is the value written in decimal of the 8-bit binary number 0000 1101?
2. What is 37 as an 8-bit binary number?
3. `printf("__ _", 70, 70);` Fill in the blanks to make this print `F 70`

Try these questions out yourself before looking at the answers!
Answers in speaker notes of this slide 👇

printf

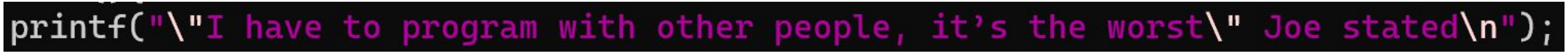
Brackets denote a placeholder

- `printf("<format string>", val1, val2, …)`
  - Prints the **format string** with any **format specifiers** replaced with the corresponding formatted value
  - %c - characters
  - %d - decimal numbers
  - %u - decimal numbers unsigned
- `printf("%s has %d characters", "Hi", 2);` -> Hi has 2 characters
- Errors:
  - Mismatched # of format specifiers and values
  - No way to format a value with given specifier (ex. Trying to print a string as a number)
- Look up C varargs if you are curious about printf taking varying arguments
- `#include <stdio.h>` to use `printf`

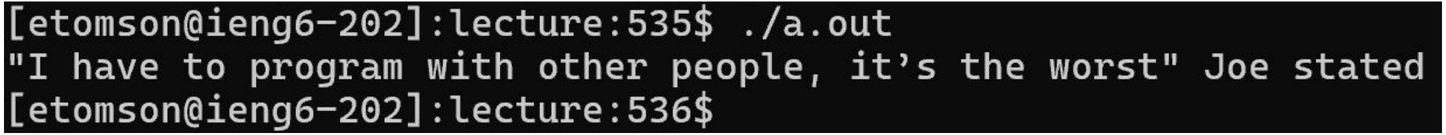
Printing questions asked and answers

If `printf` uses the ‘%’ character to indicate a format specifier, how do we print ‘%’?

To print most special characters like quotation marks, we can use ‘\’ which is called an “escape character” for example the following one line program prints as follows.

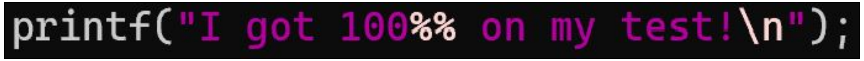


```
printf("\"I have to program with other people, it’s the worst\" Joe stated\n");
```

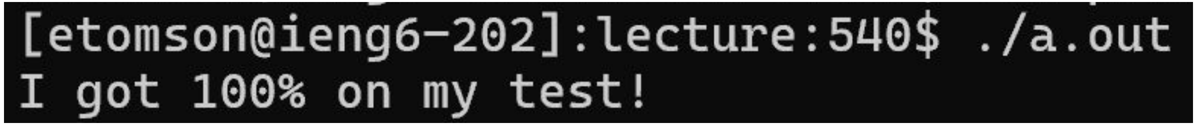


```
[etomson@ieng6-202]:lecture:535$ ./a.out
"I have to program with other people, it’s the worst" Joe stated
[etomson@ieng6-202]:lecture:536$
```

To print a ‘%’ however we use a second ‘%’ to escape it.



```
printf("I got 100%% on my test!\n");
```



```
[etomson@ieng6-202]:lecture:540$ ./a.out
I got 100% on my test!
```

C Strings

‘\0’ is the null character, which is 0 in decimal

- C strings are sequences of char (1 byte each) followed by a 0 byte, called a **null terminator**
- C strings are often created as **arrays**

```
char msg[] = “Hello!”;

char msg2[] = {‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘!’, 0};
```

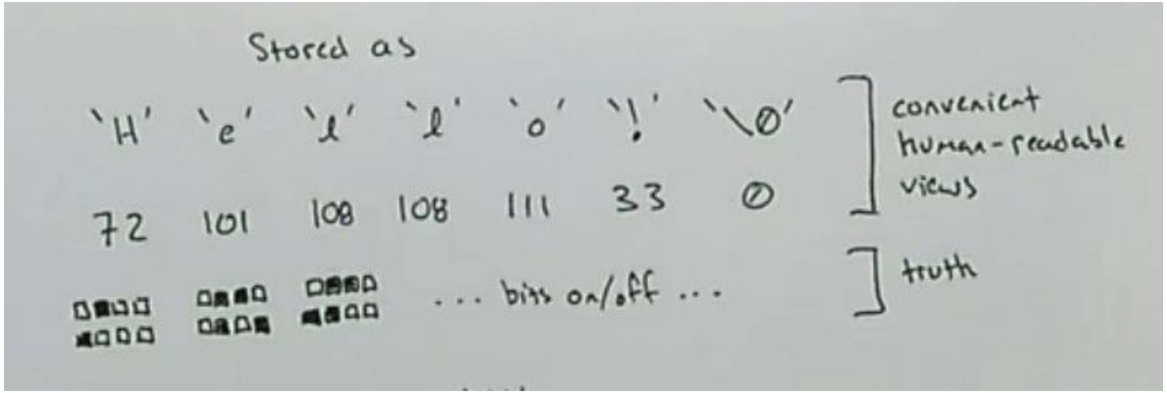
null terminator must be included. It is implicitly included in msg, making msg and msg2 exactly the same: stored consecutively as characters in memory Curious why? Come to lecture next week!

- C **does not** have an explicit string data type

```
 ‘H’  ‘e’  ‘l’   ‘l’  ‘o’  ‘!’  ‘\0’

 72   101   108 108 111   33   0
```

‘H’ cannot “turn into” 72, they are the same bit pattern in memory, displayed as an ‘H’ or 72 for our convenience

C Strings (cont.)



```
char msg[] = "Hello!";
```

msg[1] evaluates to ‘e’ 101

msg[4] evaluates to ‘o’ 111

msg[0] = ‘h’ changes msg to have ‘h’ at index 0

```
#include <string.h>
```

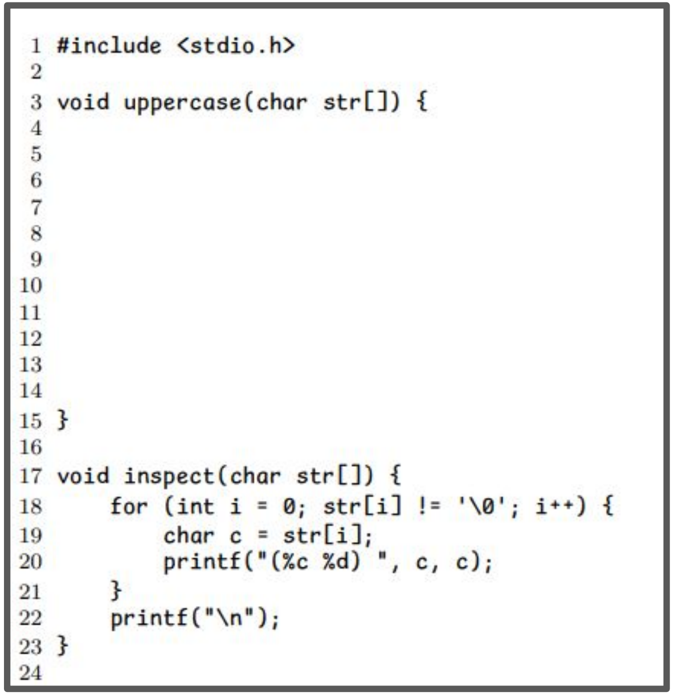
strlen(s) - returns # of characters up to (not including) null terminator

strlen is for C strings (char arrays)! Not all arrays are C strings. There’s no length function for arrays.

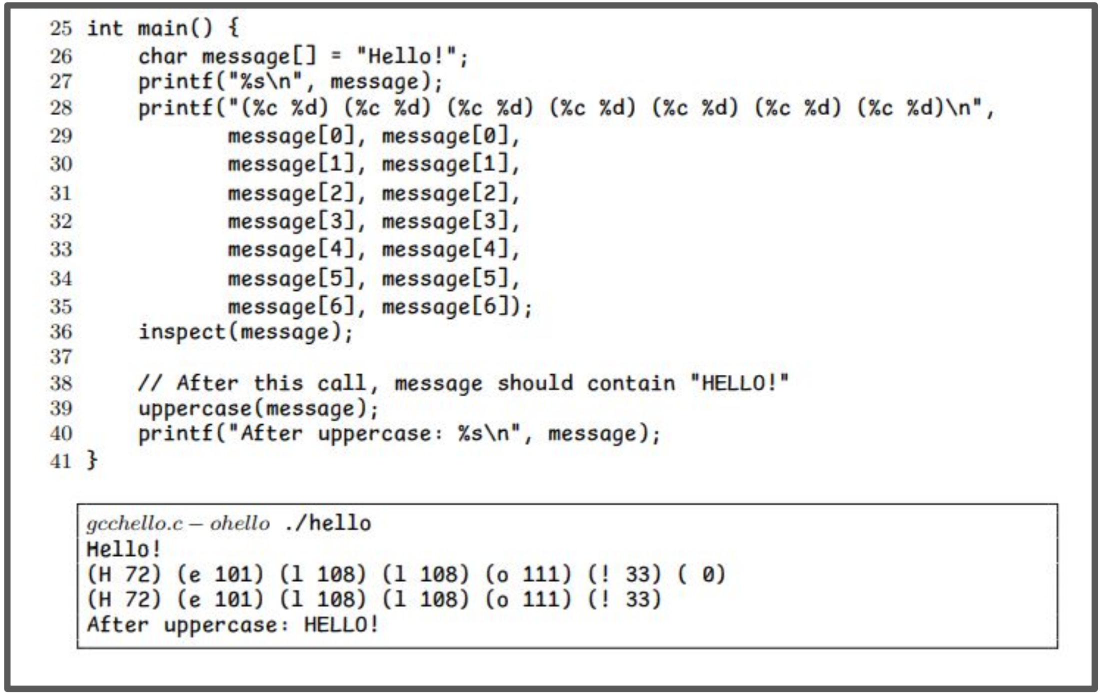
In lecture question asked: how do we terminate other kinds of arrays like int arrays?

- There’s no general solution as all values including 0 are often meaningful. Case-by-case solutions, longer answer in speaker notes from Joe.

What questions/observations do you have looking at this?



```c
#include <stdio.h>

void uppercase(char str[]) {











}

void inspect(char str[]) {
    for (int i = 0; str[i] != '\0'; i++) {
        char c = str[i];
        printf("(%c %d) ", c, c);
    }
    printf("\n");
}

```

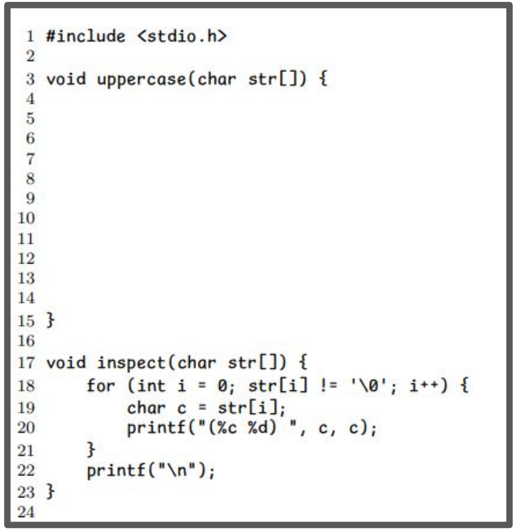


```c
int main() {
    char message[] = "Hello!";
    printf("%s\n", message);
    printf("(%c %d) (%c %d) (%c %d) (%c %d) (%c %d) (%c %d) (%c %d)\n",
            message[0], message[0],
            message[1], message[1],
            message[2], message[2],
            message[3], message[3],
            message[4], message[4],
            message[5], message[5],
            message[6], message[6]);
    inspect(message);

    // After this call, message should contain "HELLO!"
    uppercase(message);
    printf("After uppercase: %s\n", message);
}
```

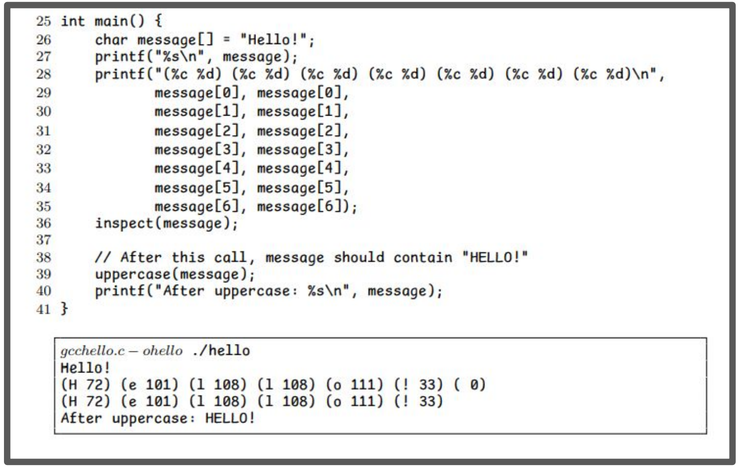
```
gcchello.c − ohello ./hello
Hello!
(H 72) (e 101) (l 108) (l 108) (o 111) (! 33) ( 0)
(H 72) (e 101) (l 108) (l 108) (o 111) (! 33)
After uppercase: HELLO!
```

Questions/observations from students

- `inspect` looks for null terminator as the for loop condition on line 18
- What does `stdio.h` do?
  - defines `printf` and other input/output operations
- What does `.h` mean?
  - This is the file extension for a header file in C. We will learn more about this in future labs!
- What does the `inspect` function do?
  - Loops through each character of a string and prints out each character and the corresponding ASCII value
- No character is printed in (0)
  - The null character would not be printed



```
1  #include <stdio.h>
2
3  void uppercase(char str[]) {
4
5
6
7
8
9
10
11
12
13
14
15 }
16
17 void inspect(char str[]) {
18     for (int i = 0; str[i] != '\0'; i++) {
19         char c = str[i];
20         printf("(%c %d) ", c, c);
21     }
22     printf("\n");
23 }
24
```

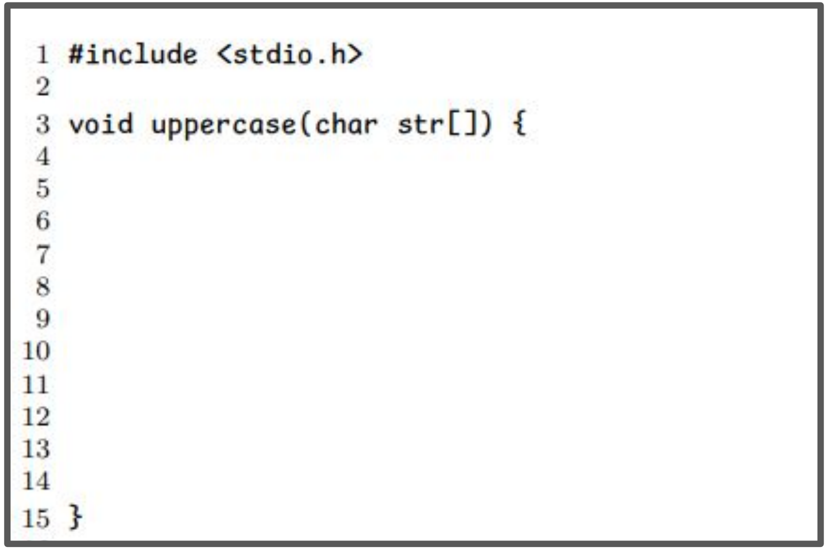


```
25 int main() {
26     char message[] = "Hello!";
27     printf("%s\n", message);
28     printf("(%c %d) (%c %d) (%c %d) (%c %d) (%c %d) (%c %d) (%c %d)\n",
29            message[0], message[0],
30            message[1], message[1],
31            message[2], message[2],
32            message[3], message[3],
33            message[4], message[4],
34            message[5], message[5],
35            message[6], message[6]);
36     inspect(message);
37
38     // After this call, message should contain "HELLO!"
39     uppercase(message);
40     printf("After uppercase: %s\n", message);
41 }
```

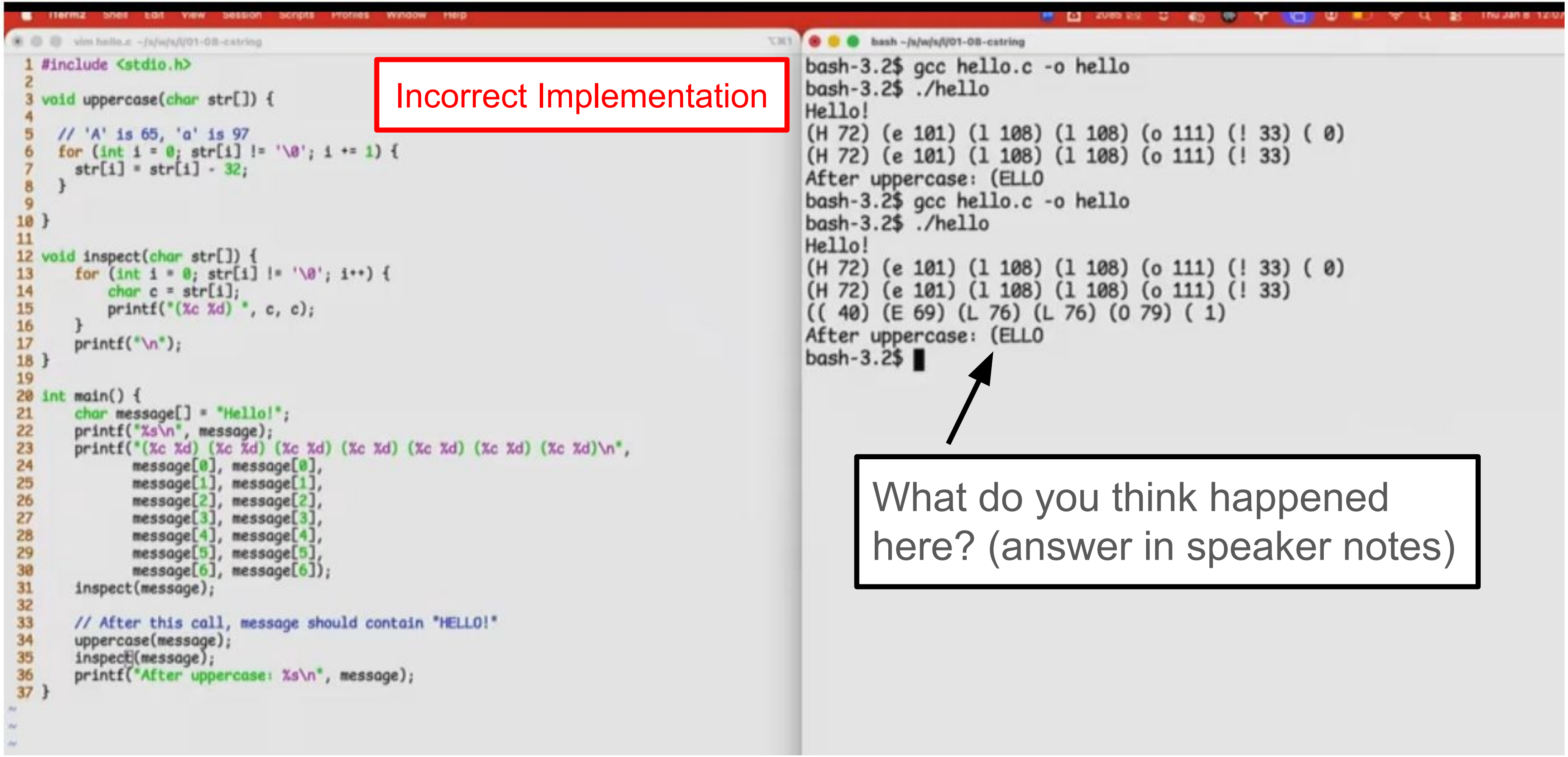
```
gcchello.c – ohello ./hello
Hello!
(H 72) (e 101) (l 108) (l 108) (o 111) (! 33) ( 0)
(H 72) (e 101) (l 108) (l 108) (o 111) (! 33)
After uppercase: HELLO!
```

Try implementing the `uppercase` function

Function description: The function takes in a char array and makes all lowercase letters into uppercase letters. No other characters (uppercase characters, punctuation, etc.) should be changed.



```c
#include <stdio.h>

void uppercase(char str[]) {











}
```

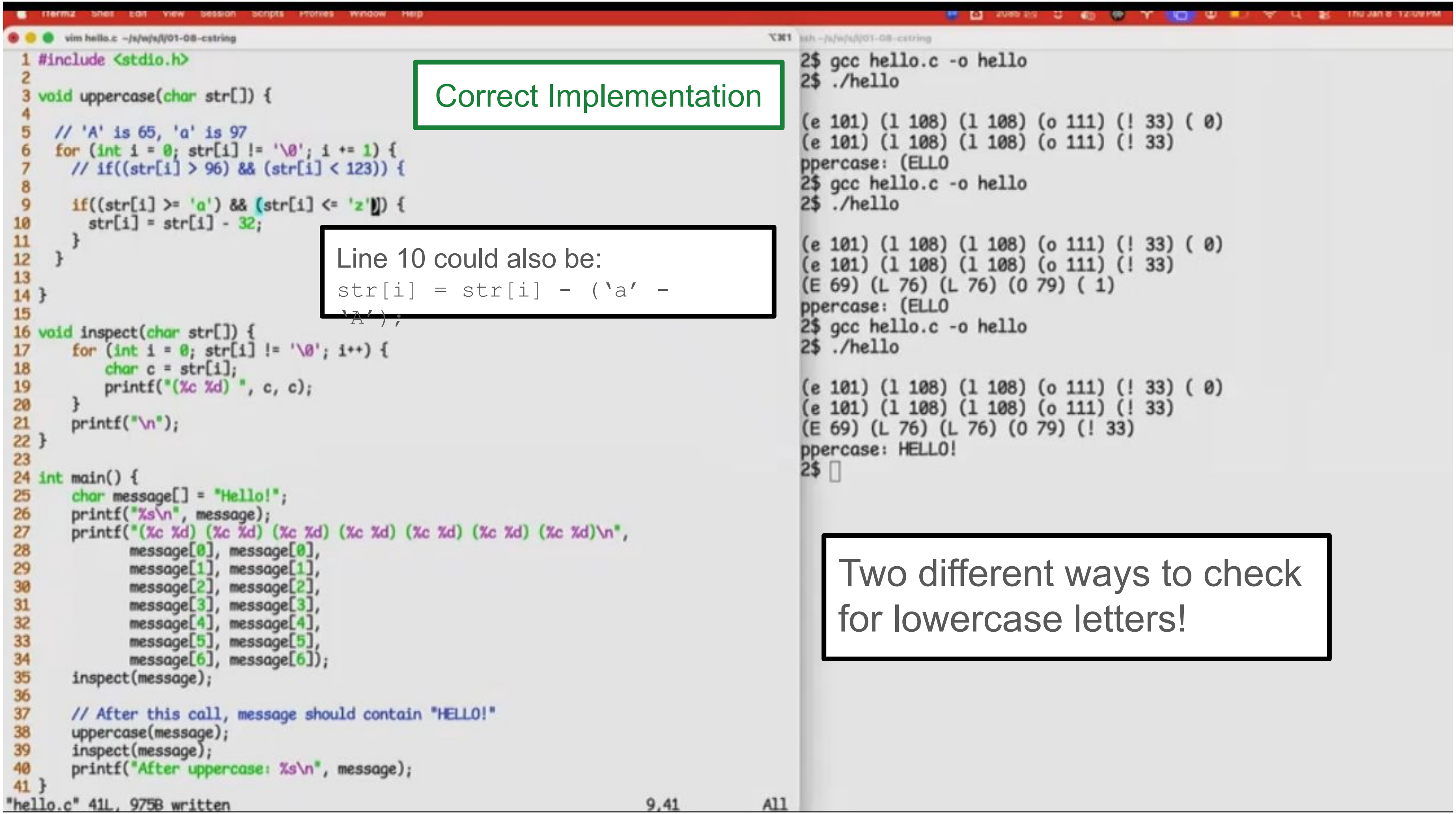

iTerm2 Shell Edit View Session Scripts Profiles Window Help
vim hello.c ~/s/w/s/l/01-08-cstring
bash ~/s/w/s/l/01-08-cstring

Incorrect Implementation

```c
#include <stdio.h>

void uppercase(char str[]) {

  // 'A' is 65, 'a' is 97
  for (int i = 0; str[i] != '\0'; i += 1) {
    str[i] = str[i] - 32;
  }

}

void inspect(char str[]) {
    for (int i = 0; str[i] != '\0'; i++) {
        char c = str[i];
        printf("(%c %d) ", c, c);
    }
    printf("\n");
}

int main() {
    char message[] = "Hello!";
    printf("%s\n", message);
    printf("(%c %d) (%c %d) (%c %d) (%c %d) (%c %d) (%c %d) (%c %d)\n",
           message[0], message[0],
           message[1], message[1],
           message[2], message[2],
           message[3], message[3],
           message[4], message[4],
           message[5], message[5],
           message[6], message[6]);
    inspect(message);

    // After this call, message should contain "HELLO!"
    uppercase(message);
    inspect(message);
    printf("After uppercase: %s\n", message);
}
```

```
bash-3.2$ gcc hello.c -o hello
bash-3.2$ ./hello
Hello!
(H 72) (e 101) (l 108) (l 108) (o 111) (! 33) ( 0)
(H 72) (e 101) (l 108) (l 108) (o 111) (! 33)
After uppercase: (ELLO
bash-3.2$ gcc hello.c -o hello
bash-3.2$ ./hello
Hello!
(H 72) (e 101) (l 108) (l 108) (o 111) (! 33) ( 0)
(H 72) (e 101) (l 108) (l 108) (o 111) (! 33)
(( 40) (E 69) (L 76) (L 76) (O 79) ( 1)
After uppercase: (ELLO
bash-3.2$
```

What do you think happened here? (answer in speaker notes)


iTerm2 Shell Edit View Session Scripts Profiles Window Help
vim hello.c ~/s/w/s/l/01-08-cstring
⌥⌘1
zsh ~/s/w/s/l/01-08-cstring

```c
 1 #include <stdio.h>
 2
 3 void uppercase(char str[]) {
 4
 5   // 'A' is 65, 'a' is 97
 6   for (int i = 0; str[i] != '\0'; i += 1) {
 7     // if((str[i] > 96) && (str[i] < 123)) {
 8
 9     if((str[i] >= 'a') && (str[i] <= 'z')) {
10       str[i] = str[i] - 32;
11     }
12   }
13
14 }
15
16 void inspect(char str[]) {
17     for (int i = 0; str[i] != '\0'; i++) {
18         char c = str[i];
19         printf("(%c %d) ", c, c);
20     }
21     printf("\n");
22 }
23
24 int main() {
25     char message[] = "Hello!";
26     printf("%s\n", message);
27     printf("(%c %d) (%c %d) (%c %d) (%c %d) (%c %d) (%c %d) (%c %d)\n",
28            message[0], message[0],
29            message[1], message[1],
30            message[2], message[2],
31            message[3], message[3],
32            message[4], message[4],
33            message[5], message[5],
34            message[6], message[6]);
35     inspect(message);
36
37     // After this call, message should contain "HELLO!"
38     uppercase(message);
39     inspect(message);
40     printf("After uppercase: %s\n", message);
41 }
"hello.c" 41L, 975B written                                   9,41          All
```

```
2$ gcc hello.c -o hello
2$ ./hello

(e 101) (l 108) (l 108) (o 111) (! 33) ( 0)
(e 101) (l 108) (l 108) (o 111) (! 33)
ppercase: (ELLO
2$ gcc hello.c -o hello
2$ ./hello

(e 101) (l 108) (l 108) (o 111) (! 33) ( 0)
(e 101) (l 108) (l 108) (o 111) (! 33)
(E 69) (L 76) (L 76) (O 79) ( 1)
ppercase: (ELLO
2$ gcc hello.c -o hello
2$ ./hello

(e 101) (l 108) (l 108) (o 111) (! 33) ( 0)
(e 101) (l 108) (l 108) (o 111) (! 33)
(E 69) (L 76) (L 76) (O 79) (! 33)
ppercase: HELLO!
2$ 
```

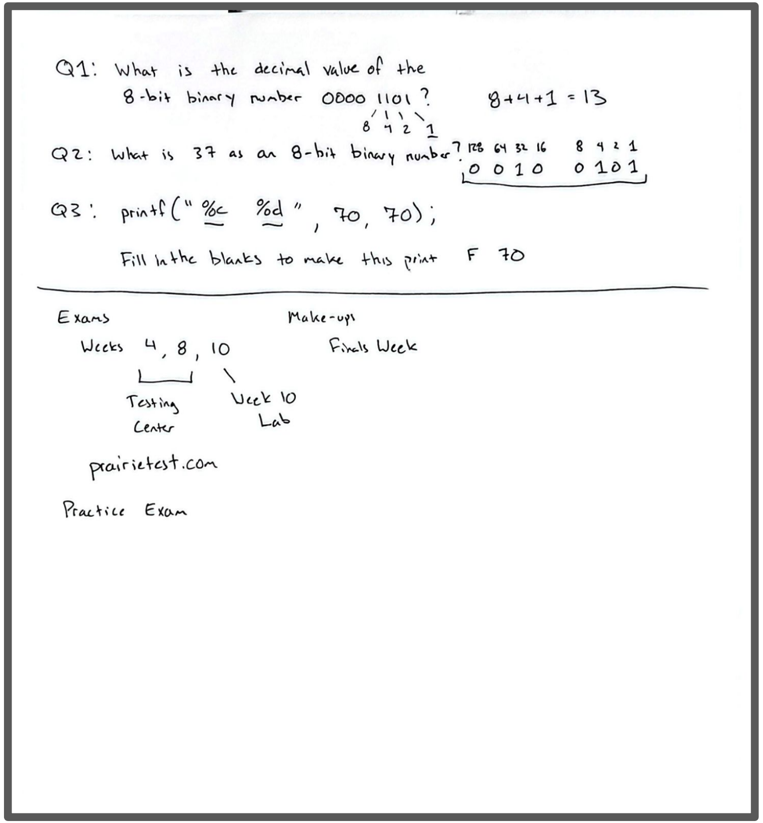
Correct Implementation

Line 10 could also be:

`str[i] = str[i] - ('a' - 'A');`

Two different ways to check for lowercase letters!

Joe’s Annotated Handouts (11am)



Q1: What is the decimal value of the 8-bit binary number 0000 1101 ?   8+4+1 = 13

8 4 2 1

Q2: What is 37 as an 8-bit binary number?

128	64	32	16	8	4	2	1
0	0	1	0	0	1	0	1

Q3: printf(" %c %d ", 70, 70);

Fill in the blanks to make this print   F 70

Exams

Weeks 4, 8, 10

Testing Center

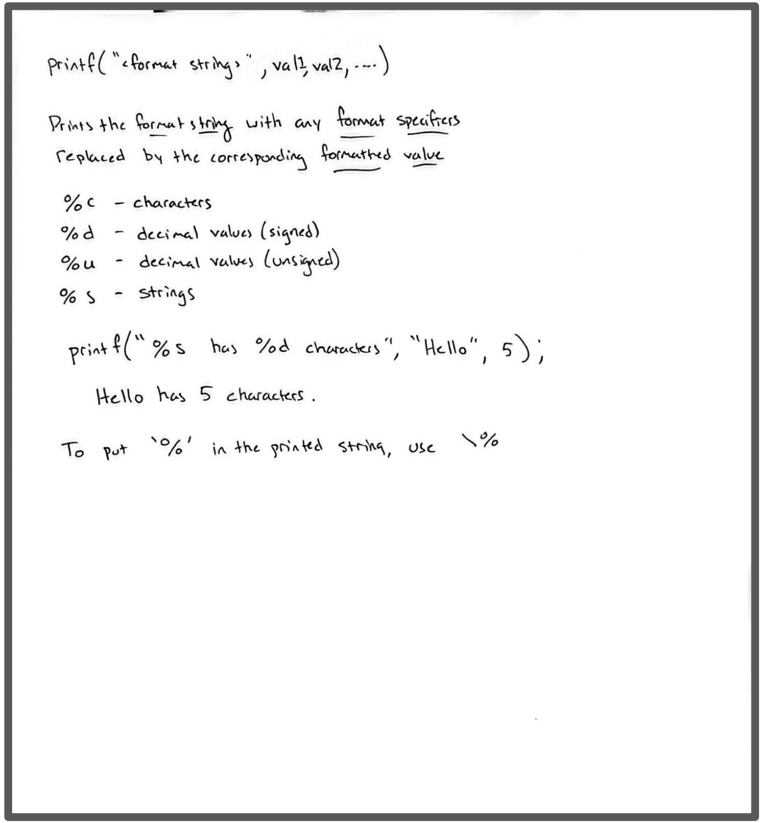
Week 10 Lab

Make-ups

Finals Week

prairietest.com

Practice Exam



printf("<format string>", val1, val2, ...)

Prints the format string with any format specifiers replaced by the corresponding formatted value

%c - characters

%d - decimal values (signed)

%u - decimal values (unsigned)

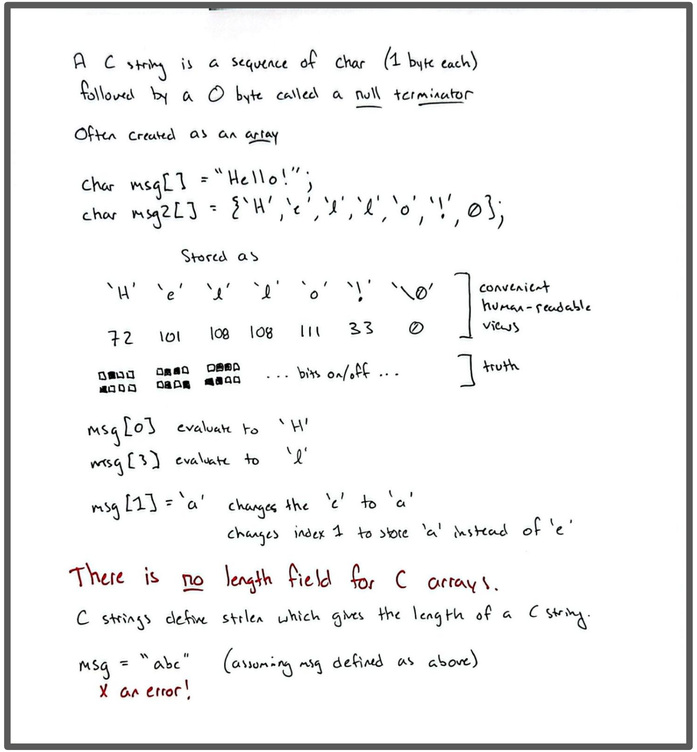
%s - strings

printf("%s has %d characters", "Hello", 5);

Hello has 5 characters.

To put '%' in the printed string, use \%

Joe’s Annotated Handouts (11am)



A C string is a sequence of char (1 byte each) followed by a 0 byte called a null terminator

Often created as an array

char msg[] = "Hello!";
char msg2[] = {'H','e','l','l','o','!',0};

Stored as

'H' 'e' 'l' 'l' 'o' '!' '\0'] convenient human-readable views

72 101 108 108 111 33 0

. . . bits on/off . . .] truth

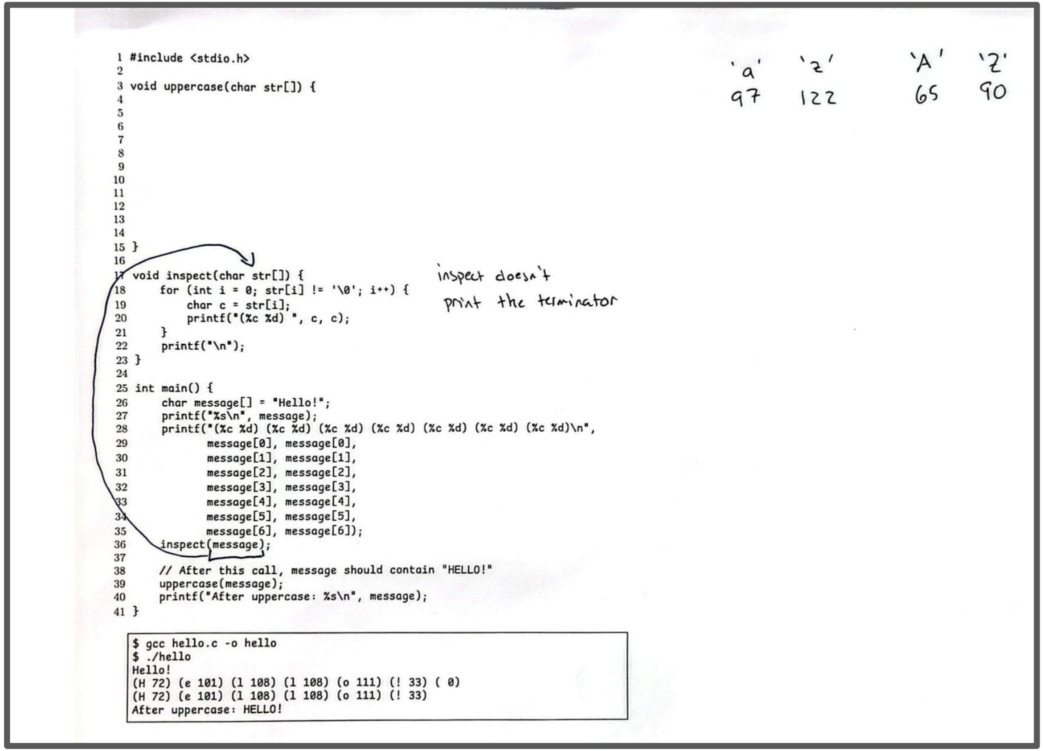
msg[0] evaluate to 'H'
msg[3] evaluate to 'l'

msg[1] = 'a' changes the 'e' to 'a'
changes index 1 to store 'a' instead of 'e'

There is no length field for C arrays.

C strings define strlen which gives the length of a C string.

msg = "abc" (assuming msg defined as above)
X an error!



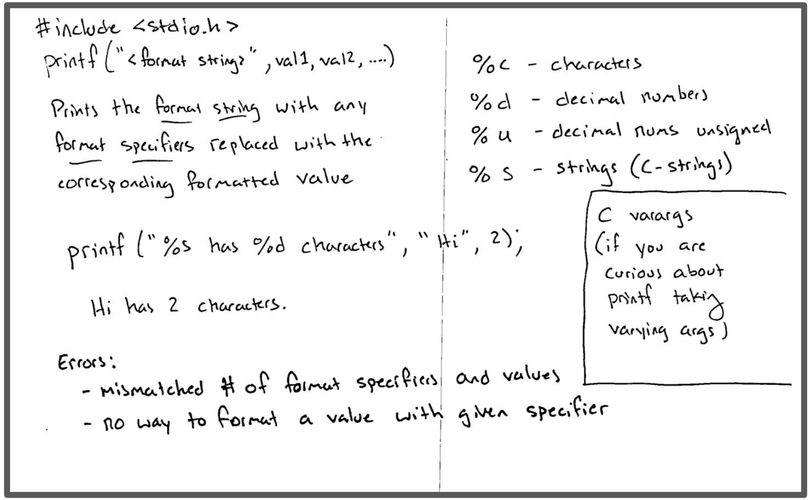
'a' 'z' 'A' 'Z'
97 122 65 90

```
#include <stdio.h>

void uppercase(char str[]) {











}

void inspect(char str[]) {
    for (int i = 0; str[i] != '\0'; i++) {
        char c = str[i];
        printf("(%c %d) ", c, c);
    }
    printf("\n");
}

int main() {
    char message[] = "Hello!";
    printf("%s\n", message);
    printf("(%c %d) (%c %d) (%c %d) (%c %d) (%c %d) (%c %d) (%c %d)\n",
            message[0], message[0],
            message[1], message[1],
            message[2], message[2],
            message[3], message[3],
            message[4], message[4],
            message[5], message[5],
            message[6], message[6]);
    inspect(message);

    // After this call, message should contain "HELLO!"
    uppercase(message);
    printf("After uppercase: %s\n", message);
}
```

inspect doesn't print the terminator

```
$ gcc hello.c -o hello
$ ./hello
Hello!
(H 72) (e 101) (l 108) (l 108) (o 111) (! 33) ( 0)
(H 72) (e 101) (l 108) (l 108) (o 111) (! 33)
After uppercase: HELLO!
```

Joe’s Annotated Handouts (12:30pm)



```
#include <stdio.h>
printf("<format string>", val1, val2, ....)
```

Prints the <u>format</u> <u>string</u> with any <u>format</u> <u>specifiers</u> replaced with the corresponding formatted value

```
printf("%s has %d characters", "Hi", 2);
```

Hi has 2 characters.

Errors:

- mismatched # of format specifiers and values
- no way to format a value with given specifier

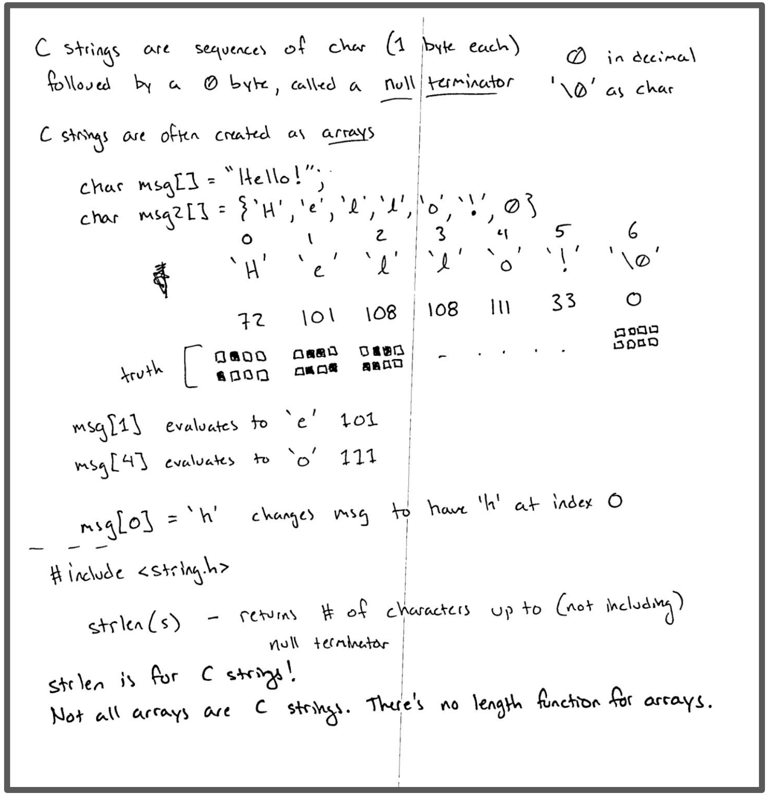
%c - characters

%d - decimal numbers

%u - decimal nums unsigned

%s - strings (C-strings)

C varargs (if you are curious about printf taking varying args)



C strings are sequences of char (1 byte each) followed by a ∅ byte, called a <u>null</u> <u>terminator</u>

∅ in decimal
'\∅' as char

C strings are often created as <u>arrays</u>

```
char msg[] = "Hello!";
char msg2[] = {'H', 'e', 'l', 'l', 'o', '!', ∅}
```

	0	1	2	3	4	5	6
	'H'	'e'	'l'	'l'	'o'	'!'	'\∅'
	72	101	108	108	111	33	0
truth	□■□□ ■□□□	□■■□ □■□■	□■■□ ■■□□	-	. . .	.	□□□□ □□□□

msg[1] evaluates to 'e' 101

msg[4] evaluates to 'o' 111

msg[0] = 'h' changes msg to have 'h' at index 0

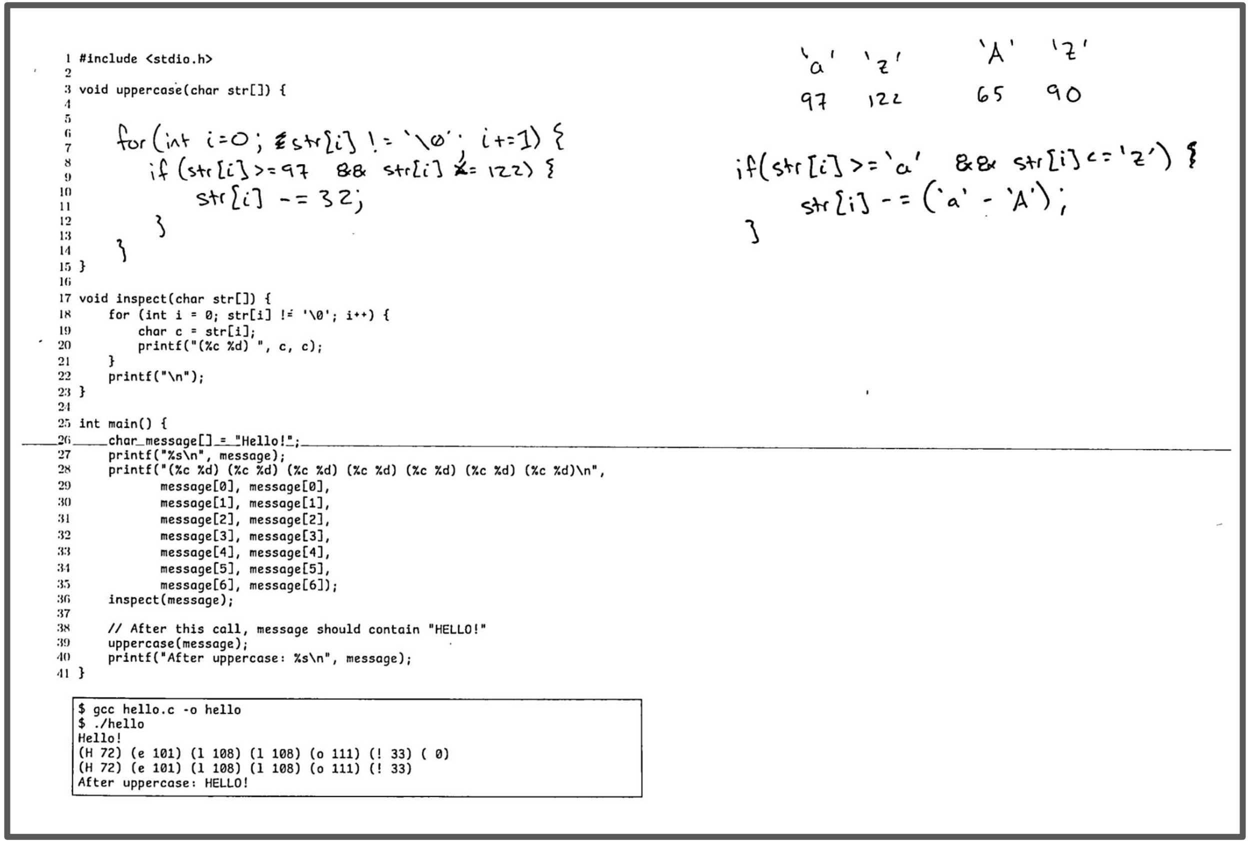
```
#include <string.h>
```

strlen(s) - returns # of characters up to (not including) null terminator

strlen is for C strings!

Not all arrays are C strings. There's no length function for arrays.

Joe’s Annotated Handouts (12:30pm)



```c
#include <stdio.h>

void uppercase(char str[]) {


    for (int i=0; str[i] != '\0'; i+=1) {
        if (str[i] >= 97 && str[i] <= 122) {
            str[i] -= 32;
        }
    }
}

void inspect(char str[]) {
    for (int i = 0; str[i] != '\0'; i++) {
        char c = str[i];
        printf("(%c %d) ", c, c);
    }
    printf("\n");
}

int main() {
    char message[] = "Hello!";
    printf("%s\n", message);
    printf("(%c %d) (%c %d) (%c %d) (%c %d) (%c %d) (%c %d) (%c %d)\n",
           message[0], message[0],
           message[1], message[1],
           message[2], message[2],
           message[3], message[3],
           message[4], message[4],
           message[5], message[5],
           message[6], message[6]);
    inspect(message);

    // After this call, message should contain "HELLO!"
    uppercase(message);
    printf("After uppercase: %s\n", message);
}
```

```
$ gcc hello.c -o hello
$ ./hello
Hello!
(H 72) (e 101) (l 108) (l 108) (o 111) (! 33) ( 0)
(H 72) (e 101) (l 108) (l 108) (o 111) (! 33)
After uppercase: HELLO!
```

'a'	'z'	'A'	'Z'
97	122	65	90

```c
if(str[i] >= 'a' && str[i] <= 'z') {
    str[i] -= ('a' - 'A');
}
```