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 👇

Answer 1: 13

Answer 2: 0010 0101
Can think of it as “filling the number” so the largest power of 2 that fits in 37 is 32, then we just have to figure out the 5. This works well for many numbers especially in the ASCII range.

Answer 3: printf(“%c %d”, 70, 70)

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**

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!

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

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

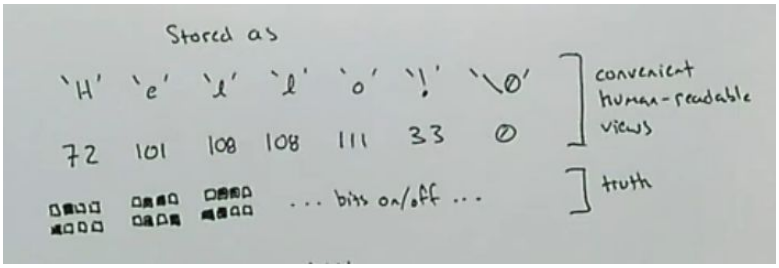
- C **does not** have an explicit string data type

```
'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.

if you wanted to have an array of integers, couldn't you just say the last one is 0? The problem with that is that in most cases, all the values of a data type are meaningful. And so it's like on a case-by-case basis, if you can pick one, that signals the end. like it's not reasonable to say that every int array is terminated by a zero or a negative one or whatever because there's a lot of int arrays that have that as a meaningful value so yeah it's there are cases where people do exactly what you're saying but it's sort of like on a per program basis that they give it meaning c strings are the main thing that are like this across all c programs with that definition of a null terminator good question yeah that's a great Great question. Yeah.

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
```

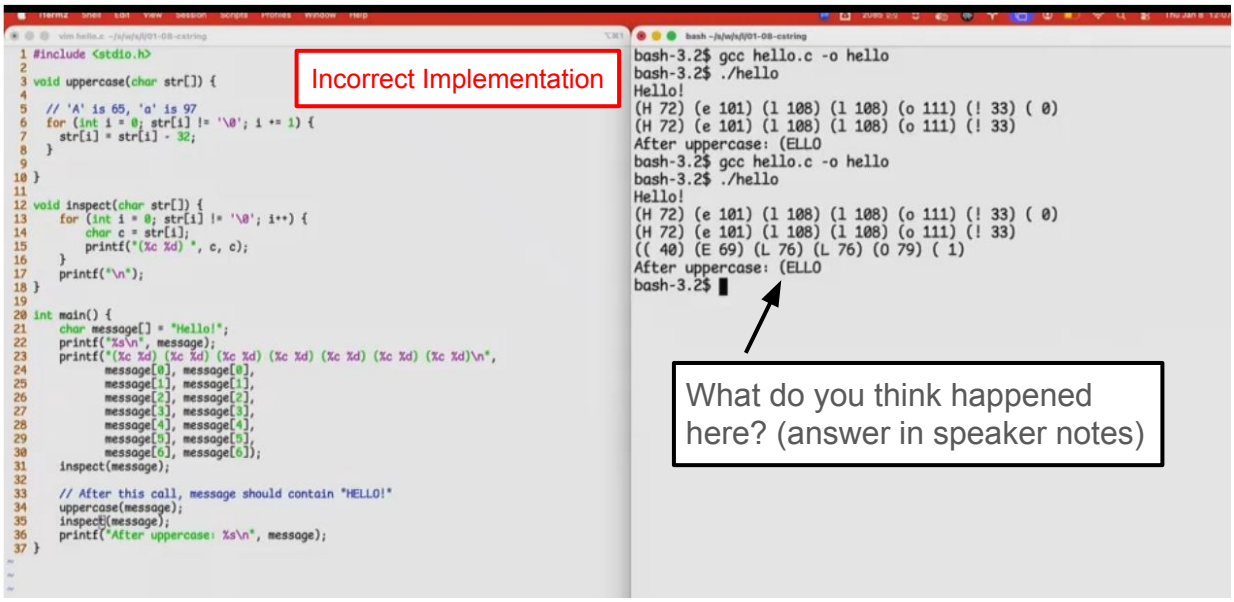
```
25 int main() {
26     char message[] = "Hello!";
27     printf("%s\n", message);
28     printf("(%c %d) (%c %d) (%c %d) (%c %d) (%c %d) (%c %d) (%c %d)\n",
29            message[0], message[0],
30            message[1], message[1],
31            message[2], message[2],
32            message[3], 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

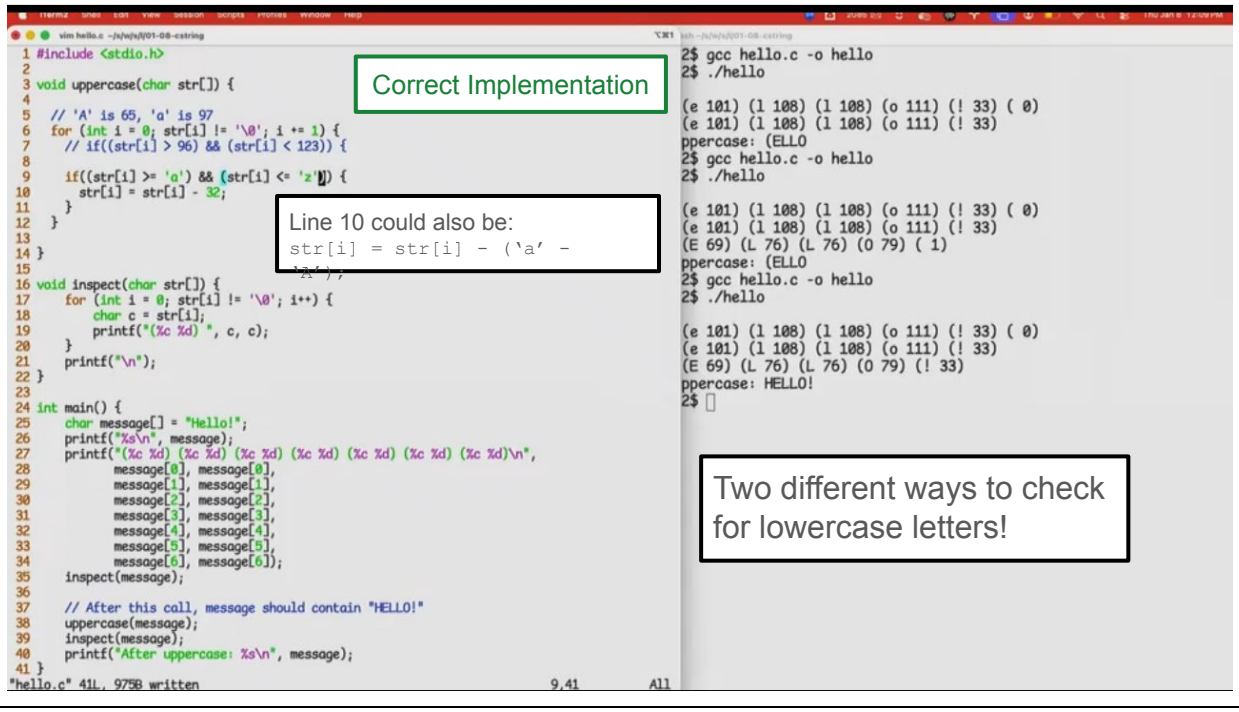
Function description: The function takes in a char array and makes all lowercase letters into uppercase letters. No other characters (uppercase characters, punctuation, etc.) should be changed.

```
1  #include <stdio.h>
2
3  void uppercase(char str[]) {
4
5
6
7
8
9
10
11
12
13
14
15 }
```

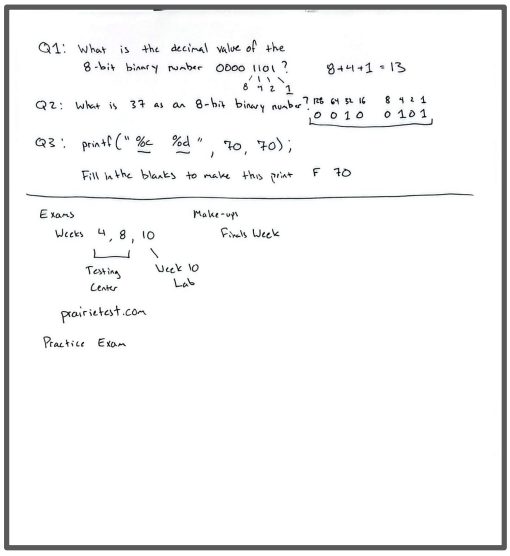


We did not check whether or not str[i] is a lowercase letter. When we subtract 32 from ‘H’ we get ‘(‘ and ‘!’ is 33 so subtracting 32 results in 1 which prints as nothing in Joe’s terminal.
Next slide has how to check if str[i] is a lowercase letter.

Also noteworthy that uppercase returns void, but message is augmented.


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

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

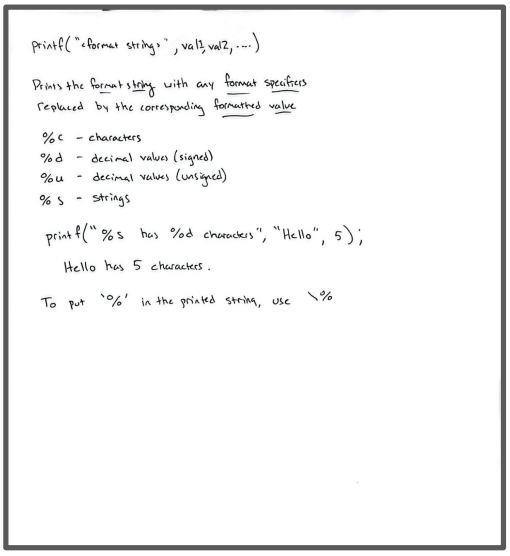
Week 10 Lab

Make-up

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 and bytes ...							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)
✗ an error!

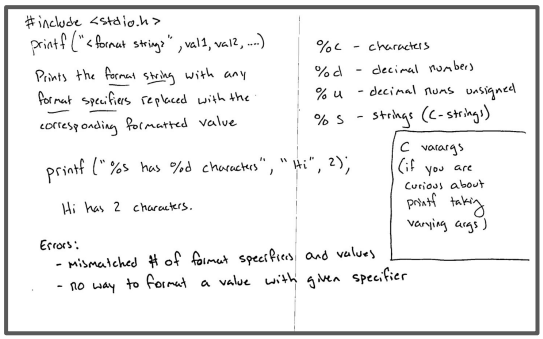
```
1  #include <stdio.h>
2
3  void uppercase(char str[]) {
4
5
6
7
8
9
10
11
12
13
14
15 }
16
17 void inspect(char str[]) {
18     for (int i = 0; str[i] != '\0'; i++) {
19         char c = str[i];
20         printf("(%c %d) ", c, c);
21     }
22     printf("\n");
23 }
24
25 int main() {
26     char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

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

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

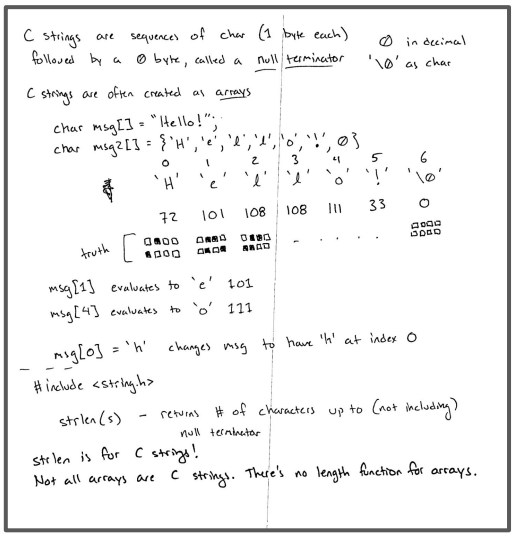
printf("%s has %d characters", "Hi", 2);

Hi has 2 characters.

%c - characters
%d - decimal numbers
%u - decimal nums unsigned
%s - strings (C-strings)

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

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



C strings are sequences of char (1 byte each) followed by a 0 byte, called a null terminator

0 in decimal
'\0' as char

C strings are often created as arrays

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

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

msg[1] evaluates to 'e' 101
msg[4] evaluates to 'o' 111

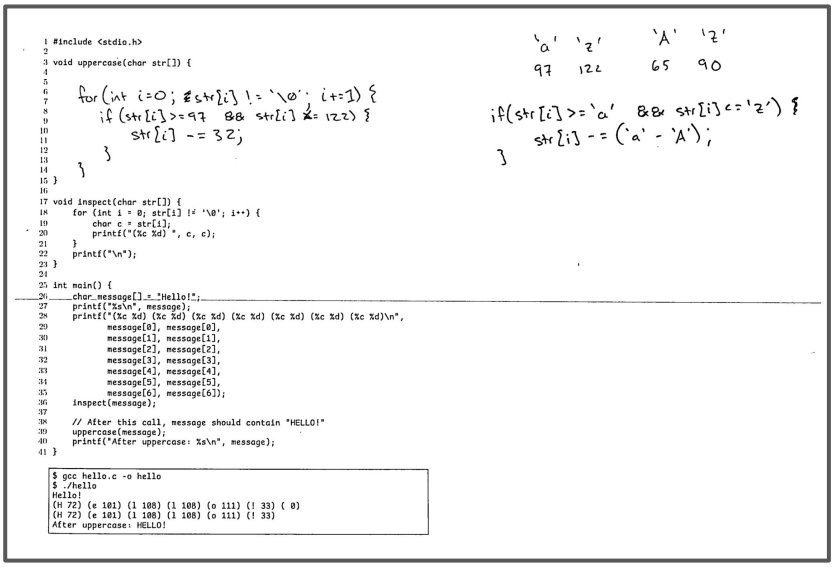
msg[0] = 'h' changes msg to have 'h' at index 0

#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');
}
```