

# Week 04 1

## CSC209 Fall 2023

Dr. Demetres (dee) Kostas

October 3, 2023

### Announcements

- A1 is out!
  - due the 18th
- No room yet for midterm
  - but it won't be here
  - and it's on the **24th**
    - \* prepare that week
    - due on Thursday (26th)

### C Strings

- we've seen them a lot already
- usually pointers to an array of characters
  - but how does `printf` know
    - \* when to stop printing?

### These aren't objects

- they are a standardized way
  - of interacting with character arrays
- the standard is backed by a library

- `string.h`
- which defines functions
  - \* that are **a lot** like string methods
  - \* but are for arbitrary `char *`

### You often need to ask:

1. Where is this string?
  - in terms of read-only, stack or heap
    - memory
2. Is this buffer big enough?

### Is this buffer big enough?

- Though, only if you are creating
  - new or concatenation of strings
- Remember it needs to hold
  - one extra character `'\0'`
    - \* not counted by `strlen`

### `sizeof` vs `strlen`

- such a common mistake
- `sizeof` does not tell you the length
  - of some string
- it will tell you the number of bytes
  - needed for the variable you pass as argument
- Note: if this is a `char *`
  - it will only tell you the `sizeof(char *)` (8)

## Some more common “string functions”

- strcpy (potentially unsafe)
  - null-character aware copy-between-arrays
- strcat
  - concatenation (potentially unsafe)
- strlen (safe)
  - length of characters
  - up to first null-char exclusively
- strchr (safe – like ‘find’)
  - get a pointer to the first instance
    - \* of a particular character

## Initialization

*where do the actual chars end up?*

- `char *str = "Hello World";`
  - **read only** memory (after code, under heap)
  - but the pointer `str` is on the local stack
- `char str[] = "Hello World"`
  - on the stack! (enough space inferred)
- `char str[8] = "< 8";`
  - on the stack again!
    - \* but only fills in up to 8 chars
    - includes null-char **if it can**

## Heap string initialization (safe)

Completely thought out:

```
char *opt_start_string = "read only, or maybe from file";
char *str = (char *) malloc(SOME_BUFFER_SIZE);
size_t num_chars = strlen(opt_start_string);
if (num_chars > SOME_BUFFER_SIZE){
    num_chars = SOME_BUFFER_SIZE;
}
```

```
strncpy(str, opt_start_string, num_chars);
// note this index is after the last copied char
str[num_chars] = '\0';
// don't forget to free str
```

- notice consideration of number of characters
  - including thought about the null-char

## Heap string initialization (problematic)

```
char *opt_start_string = "read only or something else";
char *str = (char *) malloc(SOME_BUFFER_SIZE);
strncpy(str, opt_start_string, SOME_BUFFER_SIZE);
```

- will it work..?
  - yes
- is it safe?
  - potentially not...
- could you be copying a decrypted password
  - into the heap for later?
- all it takes is one non-null terminated string
  - it all comes crashing down...

## Heap string (unacceptable)

```
#define BUFFER_SIZE 256
char *str = malloc(sizeof(BUFFER_SIZE));
str = "Initialization string";
```

- **but this has no warning during compile!!**
- notice the memory leak...
  - many things wrong
- `sizeof` still works, because it uses 256
  - same as `sizeof(int)`
    - \* `malloc`'s 4 bytes

## WORKSHEET

strings.pdf