

Strings and Evils

More than you think

C Strings

- Strings in C are sequences of characters contiguously stored
 - Not a native type like `int` or `float`
- A string terminates with the null character
 - `\0`
- That's *it!*

string functions

- Use the string library functions
 - strcmp
 - strlen
 - strcpy
 - strcat

Formatted printing

- Formatted means numbers correctly printed along side text
- Formatted printing is done with
 - `printf()`
 - Prints to standard out
 - `sprintf()`
 - Prints to a string (a char array)
 - `fprintf()`
 - Prints to a file

my string

```
char mystring[16];
```

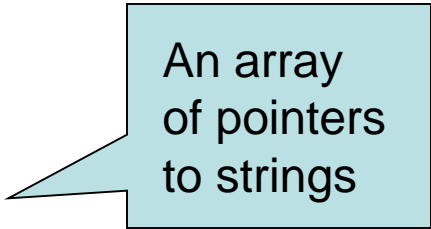
m	y		s	t	r	i	n	g	\0	b	o	g	u	s	\0
---	---	--	---	---	---	---	---	---	----	---	---	---	---	---	----

```
printf("%s", mystring);
```

my string

Declaring strings

- Two ways of declaring the same string
 - `char* mystring = "my string";`
 - `char mystring[10] = "my string";`
- Are they really the same?
- What does this mean:
`char* mystring[10];`



An array
of pointers
to strings

C String Issues

- What's wrong with this?

```
char* mystring = "my string";  
strcpy(mystring, "AA string");  
printf(mystring);
```

- Using a literal string means *constant*
- When is this error caught?
 - run-time

C String Issues

- What's wrong with this?

```
char fiveStr[5] = "five";  
strcpy(fiveStr, "five6");  
printf(fiveStr);
```

- “five6” is too long to store in fiveStr
- When is this error caught?
 - never!
 - Unless something you needed is overwritten...

strncpy

- strncpy
 - Copy only n characters
 - Won't let you copy more chars than the var can hold, but also won't null-terminate a full string.
- Does catch the run-time buffer overwrite
- Unfortunately, this still won't catch an overwrite in compile-time

Safety

- C string safety is a major problem
- New version of Microsoft's IDE Visual Studio uses `strcpy_s` and `strncpy_s`
 - You also specify the size of the buffer
- Both of these catch run-time buffer overwrites

Compile-time?

- `strncpy_s(destination, 10, source, 20);`
 - Should catch buffer overflow at compile
 - but doesn't!
- Unfortunately, neither `strcpy_s` nor `strncpy_s` will catch a buffer overwrite in compile-time!
 - Is this surprising? Is it the wrong question?
- Need an actual string type

What about C++?

- C++ has a string class
 - It is not part of the Standard Template Library (STL)
 - Is is part of the ANSI C++ standard library, to which the STL belongs
- C++ also can build strings using the operators << and >>

C++ String Class

- Powerful functions
 - at
 - compare
 - length
 - insert
 - append
 - substr
 - find (5!)
 - c_str

Meanwhile back on the ranch...

- C continues to provide dangerous string functions...
- Case: strtok – the C string tokenizer
 - Splits strings into chunks

strtok example

```
char input[18] = "This.is my/string";
```

```
char* token = strtok(input, " ./");
```



This

```
token = strtok(NULL, " ./");
```



is

```
token = strtok(NULL, " ");
```



my/string

`strtok` abandons all sense of propriety, chivalry, and decency

```
char* input = "This.is my/string";  
char* token = strtok(input, " ./");  
token = strtok(NULL, " ./");  
token = strtok(NULL, " ");
```

- Fails miserably... why?
 - Because input is a string literal, and `strtok` is about to mess with your strings

Expanded strtok example

```
char input[18] = "This.is my/string";
```

This.is my/string

```
char* token = strtok(input, " ./");
```

This

This

```
token = strtok(NULL, " ./");
```

is

This

```
token = strtok(NULL, " ");
```

my/string

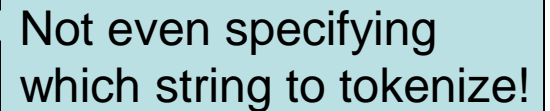
This

input is actually modified as
successive strtoks occur!

Further `strtok` horrors

- Not only does `strtok` modify the input...

```
char input[18] = "This.is my/string";  
char* token = strtok(input, " ./");  
token = strtok(NULL, " ./");  
token = strtok(NULL, " ");
```



Not even specifying
which string to tokenize!

- Mixing calls of `strtok` to different strings is not allowed - If you do it, the results of what is in each string could be anything

Horrors Explained

- This mixing of strtok calls is easy to do on accident in a large program
- strtok(1)
 - function
 - strtok(2)
- strtok(1)

Finality

- The Fatal flaw – strtok is keeping hidden, internal variables to track what it's doing
- Non-transparent
 - Very hard to debug
- Can't mix tokenizers
- Time for a higher-level (read: better) language