Strings

CS 211

Road map

Key things to know about C strings

- C represents strings as 0-terminated arrays of chars
- Don’t confuse the pointer with its contents
- Be careful with string literals

Road map

- Key things to know about C strings
  - C represents strings as 0-terminated arrays of chars
  - Don’t confuse the pointer with its contents
  - Be careful with string literals
- Style for strings
  - Avoid out-of-date assumptions
  - Avoid extra work
  - A convenient thing about C string literals

Initial code setup

The code in this course is available in your Unix shell account. You can get your own copy like this:

```
% cd cs211
% tar -xvkf ~cs211/lec/05_strings.tgz
⋮
% cd 05_strings
```

Up next

- Key things to know about C strings
    - C represents strings as 0-terminated arrays of chars
    - Don’t confuse the pointer with its contents
    - Be careful with string literals
- Style for strings
    - Avoid out-of-date assumptions
    - Avoid extra work
    - A convenient thing about C string literals

Up next

- Key things to know about C strings
    - C represents strings as 0-terminated arrays of chars
    - Don’t confuse the pointer with its contents
    - Be careful with string literals
- Style for strings
    - Avoid out-of-date assumptions
    - Avoid extra work
    - A convenient thing about C string literals

Key points

- The storage size of a string is 1 more than its length
- To find the end of a string, watch for the terminating `0`

Key points

- The storage size of a string is 1 more than its length(-in-`char`s)
- To find the end of a string, watch for the terminating `0`

Up next

- Key things to know about C strings
  - C represents strings as 0-terminated arrays of chars
  - Don’t confuse the pointer with its contents
  - Be careful with string literals
- Style for strings
  - Avoid out-of-date assumptions
  - Avoid extra work
  - A convenient thing about C string literals

How so?

- Comparing the pointers to two strings does not compare their contents
- The size of the pointer is not the length of the string

Up next

- Key things to know about C strings
  - C represents strings as 0-terminated arrays of chars
  - Don’t confuse the pointer with its contents
  - Be careful with string literals
- Style for strings
  - Avoid out-of-date assumptions
  - Avoid extra work
  - A convenient thing about C string literals

How so?

How so?

- You aren’t allowed to modify them

How so?

- You aren’t allowed to modify them
- However, it’s easy to initialize an array from a string literal and modify that

How so?

- You aren’t allowed to modify them
- However, it’s easy to initialize an array from a string literal and modify that
- These aren’t the same thing

Escaping is only skin deep

The representation is not the information

Up next

- Key things to know about C strings
    - C represents strings as 0-terminated arrays of chars
    - Don’t confuse the pointer with its contents
    - Be careful with string literals
- Style for strings
    - Avoid out-of-date assumptions
    - Avoid extra work
    - A convenient thing about C string literals

Up next

- Key things to know about C strings
  - C represents strings as 0-terminated arrays of chars
  - Don’t confuse the pointer with its contents
  - Be careful with string literals
- Style for strings
  - Avoid out-of-date assumptions
  - Avoid extra work
  - A convenient thing about C string literals

Like what?

- Never assume that a particular numeric `char` value corresponds to a particular glyph

Like what?

- Never assume that a particular numeric `char` value corresponds to a particular glyph
- Don’t assume that 1 `char` equals 1 character-as-you’d-normally-count

What to Be Aware Of When It Comes to Character Representations

glyph	ASCII
a	`0x61`
b	`0x62`
z	`0x7A`
1	`0x31`

What to Be Aware Of When It Comes to Character Representations

glyph	ASCII
a	`0x61`
b	`0x62`
z	`0x7A`
1	`0x31`
ä	N/A
“	N/A

What to Be Aware Of When It Comes to Character Representations

glyph	ASCII	EBCDIC
a	`0x61`	`0x81`
b	`0x62`	`0x82`
z	`0x7A`	`0xA9`
1	`0x31`	`0xF1`
ä	N/A	`0x43`
“	N/A	N/A

What to Be Aware Of When It Comes to Character Representations

glyph	ASCII	EBCDIC	CP-1252
a	`0x61`	`0x81`	`0x61`
b	`0x62`	`0x82`	`0x62`
z	`0x7A`	`0xA9`	`0x7A`
1	`0x31`	`0xF1`	`0x31`
ä	N/A	`0x43`	`0xE4`
“	N/A	N/A	`0x93`

What to Be Aware Of When It Comes to Character Representations

glyph	ASCII[1,2,3]	EBCDIC[4]	CP-1252[1,3]
a	`0x61`	`0x81`	`0x61`
b	`0x62`	`0x82`	`0x62`
z	`0x7A`	`0xA9`	`0x7A`
1	`0x31`	`0xF1`	`0x31`
ä	N/A	`0x43`	`0xE4`
“	N/A	N/A	`0x93`

[1] common on Windows

[2] common on Unix (includes Mac and Linux)

[3] common on the web

[4] common on the old IBM mainframes

What to Be Aware Of When It Comes to Character Representations

glyph	ASCII[1,2,3]	CP-1252[1,3]
a	`0x61`	`0x61`
b	`0x62`	`0x62`
z	`0x7A`	`0x7A`
1	`0x31`	`0x31`
ä	N/A	`0xE4`
“	N/A	`0x93`

[1] common on Windows

[2] common on Unix (includes Mac and Linux)

[3] common on the web

What to Be Aware Of When It Comes to Character Representations

glyph	ASCII[1,2,3]	CP-1252[1,3]
a	0x61	0x61
b	0x62	0x62
z	0x7A	0x7A
1	0x31	0x31
ä	N/A	0xE4
“	N/A	0x93
字	N/A	N/A
발	N/A	N/A

[1] common on Windows

[2] common on Unix (includes Mac and Linux)

[3] common on the web

What to Be Aware Of When It Comes to Character Representations

glyph	ASCII[1,2,3]	CP-1252[1,3]	UTF-16[1]	UTF-8[2,3]
a	`0x61`	`0x61`	`0x0061`	`0x61`
b	`0x62`	`0x62`	`0x0062`	`0x62`
z	`0x7A`	`0x7A`	`0x007A`	`0x7A`
1	`0x31`	`0x31`	`0x0031`	`0x31`
ä	N/A	`0xE4`	`0x00E4`	`0xC3A4`
“	N/A	`0x93`	`0x201C`	`0xE2809C`
字	N/A	N/A	`0x5B57`	`0xE5AD97`
발	N/A	N/A	`0xBC1C`	`0xEBB09C`

[1] common on Windows

[2] common on Unix (includes Mac and Linux)

[3] common on the web

What to Be Aware Of When It Comes to Character Representations

glyph	ASCII[1,2,3]	UTF-16[1]	UTF-8[2,3]
a	`0x61`	`0x0061`	`0x61`
b	`0x62`	`0x0062`	`0x62`
z	`0x7A`	`0x007A`	`0x7A`
1	`0x31`	`0x0031`	`0x31`
ä	N/A	`0x00E4`	`0xC3A4`
“	N/A	`0x201C`	`0xE2809C`
字	N/A	`0x5B57`	`0xE5AD97`
발	N/A	`0xBC1C`	`0xEBB09C`

[1] common on Windows

[2] common on Unix (includes Mac and Linux)

[3] common on the web

What to Be Aware Of When It Comes to Character Representations

glyph	ASCII	UTF-16	UTF-8
a	`0x61`	`0x0061`	`0x61`
b	`0x62`	`0x0062`	`0x62`
z	`0x7A`	`0x007A`	`0x7A`
1	`0x31`	`0x0031`	`0x31`
ä	N/A	`0x00E4` *or* `0x00610308`	`0xC3A4` *or* `0x61CC88`
“	N/A	`0x201C`	`0xE2809C`
字	N/A	`0x5B57`	`0xE5AD97`
발	N/A	`0xBC1C` *or* `0x11071161·11AF`	`0xEBB09C` *or* `0xE18487E1·85A1E186AF`

What to Be Aware Of When It Comes to Character Representations

glyph	ASCII	UTF-16	UTF-8
a	0x61	0x0061	0x61
b	0x62	0x0062	0x62
z	0x7A	0x007A	0x7A
1	0x31	0x0031	0x31
ä	N/A	0x00E4	0xC3A4
“	N/A	0x201C	0xE2809C
字	N/A	0x5B57	0xE5AD97
발	N/A	0xBC1C	0xEBB09C

What to Be Aware Of When It Comes to Character Representations

glyph	ASCII	UTF-16	UTF-8
a	0x61	0x0061	0x61
b	0x62	0x0062	0x62
z	0x7A	0x007A	0x7A
1	0x31	0x0031	0x31
ä	N/A	0x00E4	0xC3A4
“	N/A	0x201C	0xE2809C
字	N/A	0x5B57	0xE5AD97
발	N/A	0xBC1C	0xEBB09C
🤦🏻‍♂️	N/A	0xD83EDD26·D83CDFFB·200D2642·FE0F	

What to Be Aware Of When It Comes to Character Representations

glyph	ASCII	UTF-16	UTF-8
a	0x61	0x0061	0x61
b	0x62	0x0062	0x62
z	0x7A	0x007A	0x7A
1	0x31	0x0031	0x31
ä	N/A	0x00E4	0xC3A4
“	N/A	0x201C	0xE2809C
字	N/A	0x5B57	0xE5AD97
발	N/A	0xBC1C	0xEBB09C
🤦🏻‍♂️	N/A	0xD83EDD26·D83CDFFB·200D2642·FE0F	0xF09FA4A6·F09F8FBB·E2808DE2·9982EFB8·8F

Up next

Key things to know about C strings

- C represents strings as 0-terminated arrays of chars
- Don’t confuse the pointer with its contents
- Be careful with string literals

Style for strings

- Avoid out-of-date assumptions
- Avoid extra work
- A convenient thing about C string literals

How?

When iterating over a string, look for the `0` terminator as you go—don't call `strlen(3)` just to find a loop limit.

Up next

- Key things to know about C strings
    - C represents strings as 0-terminated arrays of chars
    - Don’t confuse the pointer with its contents
    - Be careful with string literals
- Style for strings
    - Avoid out-of-date assumptions
    - Avoid extra work
    - A convenient thing about C string literals

They concatenate

```
const char* DO_NOT = "Don't write a really really really long lon
```

They concatenate

```
const char* DO_NOT = "Don't write a really really really long long

const char* DO = "Instead, break your really really really lo"
                 "ng long long line into sub-line-length chun"
                 "ks (though maybe do it between words to mak"
                 "e it easier to read).";
```

They concatenate

```
const char* DO_NOT = "Don't write a really really really long lon

const char* DO = "Instead, break your really really really lo"
                 "ng long long line into sub-line-length chun"
                 "ks (though maybe do it between words to mak"
                 "e it easier to read).";

const char* WHY_DOES_IT_WORK_ = "C " "concatenates " "adjacent " "
                                "literals " "at " "compile " "tim
```

– Next time: Dynamic Memory –