Week 3 Thursday
Strings II

String Theory Summarized:

I just had an awesome idea. Suppose all matter and energy is made of tiny, vibrating "strings."

Okay. What would that imply?

I dunno.

https://xkcd.com/171/
Announcements

- Friday 5pm: Add / Drop deadline
  - Please reach out if you have any questions or if you’d like to meet
- 1:1s (Saturday/Sunday, reach out if the times fill up or you can’t make these times): [https://calendly.com/adbenson/cs107a-1-1](https://calendly.com/adbenson/cs107a-1-1)
- Late Days
- assign2 walkthrough out
Unix Tip Spotlight

● Tab completion
  ○ Use it for programs you’re running!
  ○ Use it for filenames!
  ○ Use it in gdb!
  ○ Just mash <TAB>, what’s the worst that could happen
Agenda

- Creating Strings
- String Practice
- Memory Diagramming
- whatdoitdo.c
- peel_string
Creating Strings
Strings are Arrays

- This array can be in the stack, heap (Monday), or in the data segment
- All valid C strings have NUL terminators, whether you see them explicitly or not
- `strlen(my_string)` returns the length NOT including the NUL terminator

```
"Hello"

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>'H'</td>
<td>'e'</td>
<td>'l'</td>
<td>'l'</td>
<td>'o'</td>
<td>' \0'</td>
</tr>
</tbody>
</table>
```
Strings in the data segment

- Read-only
- A string created with a string literal (double-quoted string)
  - Exception: if a string literal is assigned to a `char []`, that string is on the stack

```c
char *str = "hello";
char *str2 = str;
char c = str[1];
char c2 = 'e';
```

- Draw out a memory diagram for this code
Strings on the stack

- Modifiable, like any stack variable
- Involves a local character array
- Don’t worry about memorizing all these methods, just be able to recognize these

```c
char s[] = "ab";
char s2[3] = "ab";
char s3[3409] = "ab";
char s4[] = {'a', 'b', '\0'};
char s5[3] = {'a', 'b', '\0'};
char s6[3409] = {'a', 'b', '\0'};
char s7[3]; strcpy(s7, "ab");
char s8[3409]; strcpy(s8, "ab");
char s9[3]; strncpy(s9, "ab", 3);
char s10[3409]; strncpy(s10, "ab", 3);
```
String Practice
What's wrong with these code snippets?

char *s = "stocks";
s[3] = 'n';
printf(s);

char s[] = {'a', 'b'};
s[0] = 'A';
printf("%s\n", s);
What’s wrong with these code snippets?

```c
char *s = "stocks";
s[3] = 'n';
printf(s);
```

This modifies a read-only string! Undefined behavior.

```c
char s[] = {'a', 'b'};
s[0] = 'A';
printf("%s\n", s);
```

Based on practice problems I originally wrote for CMU 15-122 Fall 2015 Recitation 12
What’s wrong with these code snippets?

```c
char *s = "stocks";
s[3] = ‘n’;
printf(s);
```

This modifies a read-only string! Undefined behavior.

```c
char s[] = {'a', 'b'};
s[0] = 'A';
printf("%s\n", s);
```

s is not a valid string – there’s no NUL terminator! Undefined behavior.

Based on practice problems I originally wrote for CMU 15-122 Fall 2015 Recitation 12
What’s wrong with these code snippets?

```c
char s[4];
strcpy(s, "stanford cs107a");
printf("%s\n", s);
```

```c
char s[] = 'work';
s[1] = "e";
printf("%s\n", s);
```
What’s wrong with these code snippets?

```c
char s[4];
strcpy(s, "stanford cs107a");
printf("%s\n", s);
```

`s` is not big enough for the string copied in! Overwrites unallocated memory - undefined behavior.

```c
char s[] = 'work';
s[1] = "e";
printf("%s\n", s);
```
What’s wrong with these code snippets?

```c
char s[4];
strcpy(s, "stanford cs107a");
printf("%s\n", s);
```

`s` is not big enough for the string copied in! Overwrites unallocated memory - undefined behavior.

```c
char s[] = 'work';
s[1] = "e";
printf("%s\n", s);
```

Doesn’t compile! Strings use double quotes, chars use single quotes.
Starter Code

git clone /afs/ir/class/archive/cs/cs107a/cs107a.1226/WWW/exercises/strings2
Memory Diagramming
String Pointers

```c
char *foo(char *s, char *t) {
    s[0] = t[2];
    t[2] = 'g';
    s = t;
    t = NULL;
    return s+1;
}

int main() {
    char s[] = "abc";
    char t[] = "def";
    printf("%s\n", foo(s, t));
    printf("%s\n", s);
    return 0;
}
```
peel_string