Week 4 Tuesday
Memory I

Fill in the check-in form on cs107a.stanford.edu!

https://xkcd.com/138/
Starter Code

git clone /afs(ir/class/archive/cs/cs107a/cs107a.1226/WWW/exercises/memory1
Announcements

● If you missed your 1:1 or haven’t scheduled one yet, please do so, make my life easier for me 😊
  ○ Slack me to schedule a time, I can no longer add more general timeslots this week
● Please make sure you can receive Slack messages / notifications
● assign2 due Wednesday midnight
Unix Tip Spotlight

● History Search
  ○ Hit <CTRL+R>
  ○ Start typing, and it’ll search your history for matching commands
  ○ To go to the previous match, hit <CTRL+R> again
  ○ Press <ENTER> to run, or use arrow keys to edit the command before running
  ○ Takes some practice before you get used to it, but really really useful
Agenda

- Review of Heap Memory
- alloc_demo.c
- Buggy Code Exercises
- Memory Diagramming “Messy Pointer Arrays”
- Coding Exercise
Review of Heap Memory
Memory Layout
## Memory Segment Comparison

<table>
<thead>
<tr>
<th>What’s it for?</th>
<th>Stack</th>
<th>Heap</th>
<th>Data Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>For local variables</td>
<td>For values that need to last beyond this function</td>
<td>For literal strings hardcoded into the program</td>
<td></td>
</tr>
<tr>
<td>How long does it last?</td>
<td>Temporary until function returns</td>
<td>Persists until freed</td>
<td>Part of the program itself, can’t be freed</td>
</tr>
<tr>
<td>How fast is it to allocate?</td>
<td>Predictably fast</td>
<td>Unpredictably slow, relatively</td>
<td>It was never allocated, so N/A</td>
</tr>
<tr>
<td>Any gotchas?</td>
<td>Don’t use stack memory after the function returns</td>
<td>Lots - don’t double free, don’t go out of bounds, don’t forget to free</td>
<td>Read-only, don’t modify</td>
</tr>
</tbody>
</table>
malloc

- `void *malloc(size_t size);`
- Allocates you `size` bytes of memory on the heap
- Always calculate your size with `sizeof`
- In certain situations, `malloc` can return `NULL`
  - In CS 107, `assert(ret_value != NULL)`
  - In real life, either assert, don’t worry about it, or handle that case
free

- `void free(void *ptr);`
- Signals to the computer that the memory pointed to by `ptr` can be reclaimed
- During the lifetime of a program, every `malloc` should have a corresponding `free`
  - Not necessarily in the code, since loops can hide how many calls exist
- If you’re concerned, wait to add calls to `free` until you pass all test cases and your code is correct
calloc and realloc

- `void *calloc(size_t count, size_t size);`
  - You should almost never use this in CS 107.
  - 1) Use `malloc` to get `count * size` bytes.
  - 2) Set every byte's value to 0.
  - 3) Return that.

- `void *realloc(void *ptr, size_t size);`
  - 1) `malloc(size);`
  - 2) Copy over old memory to new memory
  - 3) `free(ptr);`
  - 2) `return` the malloced memory
Buggy Code
Exercises
malloc Gotchas (buggy1.c, buggy2.c)

```c
#include <stdio.h>
#include <string.h>

void print_char_by_char(char arr[]) {
    for (int i = 0; i < sizeof(arr); i++) {
        printf("The %dth char is %c\n", i, arr[i]);
    }
}

int main() {
    char stack_string[] = "stack";
    // strdup creates a heap-allocated copy of a string.
    char *heap_string = strdup("heap");
    print_char_by_char(stack_string);
    print_char_by_char(heap_string);
    return 0;
}
```

```c
#include <stdlib.h>

int main() {
    int *arr = malloc(5);
    for (int i = 0; i < 5; i++) {
        arr[i] = 5*i;
    }
    free(arr);
    return 0;
}
```
malloc Gotchas (buggy1.c, buggy2.c)

#include <stdio.h>
#include <string.h>

void print_char_by_char(char arr[]) {
    for (int i = 0; i < sizeof(arr); i++) {
        printf("The %dth char is %c\n", i, arr[i]);
    }
}

int main() {
    char stack_string[] = "stack";
    // strdup creates a heap-allocated copy of a string.
    char *heap_string = strdup("heap");
    print_char_by_char(stack_string);
    print_char_by_char(heap_string);
    return 0;
}

#include <stdlib.h>

int main() {
    int *arr = malloc(5);
    for (int i = 0; i < 5; i++) {
        arr[i] = 5*i;
    }
    free(arr);
    return 0;
}

malloc(5) only allocates 5 bytes! Should be
malloc(5*sizeof(int))
**malloc Gotchas (buggy1.c, buggy2.c)**

```c
#include <stdlib.h>

int main() {
    int *arr = malloc(5);
    for (int i = 0; i < 5; i++) {
        arr[i] = 5*i;
    }
    free(arr);
    return 0;
}
```

`malloc(5)` only allocates 5 bytes! Should be

`malloc(5*sizeof(int))`

```c
#include <stdio.h>
#include <string.h>

void print_char_by_char(char arr[]) {
    for (int i = 0; i < sizeof(arr); i++) {
        printf("The %dth char is %c\n", i, arr[i]);
    }
}

int main() {
    char stack_string[] = "stack";
    // strdup creates a heap-allocated copy of a string.
    char *heap_string = strdup("heap");
    print_char_by_char(stack_string);
    print_char_by_char(heap_string);
    return 0;
}
```

`sizeof` returns 8 here since `arr` is a pointer type! You must explicitly pass the size.
malloc Gotchas (buggy3.c, buggy4.c)

```c
#include <stdio.h>
#include <stdlib.h>

void print_int(int *ptr) {
    printf("%d\n", *ptr);
}

int main() {
    int *ptr = malloc(sizeof(int));
    *ptr = 42;
    print_int(ptr);
    free(ptr);
}
```

```c
#include <stdio.h>
#include <stdlib.h>

void print_int(int *ptr) {
    printf("%d\n", *ptr);
    free(ptr);
}

int main() {
    int *ptr = malloc(sizeof(int));
    *ptr = 42;
    print_int(ptr);
    free(ptr);
}
```
malloc Gotchas (buggy3.c, buggy4.c)

1) *ptr reads uninitialized memory
2) free(ptr) only frees the memory if ptr was returned
### malloc Gotchas (buggy3.c, buggy4.c)

```c
#include <stdio.h>
#include <stdlib.h>

void print_int(int *ptr) {
    printf("%d\n", *ptr);
    free(ptr);
}

int main() {
    int *ptr = malloc(sizeof(int));
    *ptr = 42;
    print_int(ptr);
    free(ptr);
}
```

1) `*ptr` reads uninitialized memory
2) `free(ptr)` only frees the memory if `ptr` was returned

```c
#include <stdlib.h>
#include <stdio.h>

int is_it_two(int *ptr) {
    if (*ptr == 2) {
        return ptr;
    }
    return NULL;
}

int main() {
    int *ptr = malloc(sizeof(int));
    ptr = is_it_two(ptr);
    free(ptr);
}
```

`ptr` is freed twice over the course of the program
Memory Diagramming
“Messy Pointer Arrays”
practice.c
What does `create_messy_ptr_array` do?

Original:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>2</td>
<td>352</td>
<td>90</td>
</tr>
</tbody>
</table>

Returned:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>2</td>
<td>352</td>
<td>90</td>
</tr>
</tbody>
</table>
Coding Exercise
What should `consolidate_messy_ptr_array` do?

Original:

```
  56  2  352  90
```

After:

```
  56  2  352  90
```