Week 4 Thursday
Memory II

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

git clone /afs/ir/class/archive/cs/cs107a/cs107a.1226/WWW/exercises/memory2
Announcements

● assign3 walkthrough posted
● 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
Unix Tip Spotlight

- Moving your cursor and deleting around the command line
  - `<CTRL+A>`: moves your cursor to the beginning of the line
  - `<CTRL+E>`: moves your cursor to the end of the line
  - `<CTRL+W>`: deletes the last word behind the cursor
  - `<CTRL+U>`: deletes everything before the cursor
  - `<CTRL+K>`: deletes everything after the cursor
Agenda

- Pointer Arithmetic
- Structs and &
- Buggy Code Exercises
- Memory Diagramming a Struct Pointer Array
- Coding Exercise
Pointer Arithmetic
Pointer Arithmetic

- In general, the types of pointers don’t have any real effect on how your code works - with the exception of pointer arithmetic
- Pointer arithmetic adds bytes in multiples of sizeof(type pointed to)

```c
char arr[] = {0, 1, 2, 3, 4, 5, 6, 7};
char *cptr = arr;
int *iptr = (int*)arr;
```
pointer_arithmetic_demo.c
Structs and &
Structs

- Composite types that allow you to combine two other types into a single thing
  - structs can contain structs, if you hate yourself
- Declare the struct globally, declare and initialize instances in your code
- If you have a struct, access fields like `struct .field`; if a struct pointer, access fields like `struct ->field`

```c
struct int_array_with_length {
    int *array;
    size_t length;
};

struct int_array_with_length my_struct;
my_struct.length = 5;
my_struct.array = malloc(mystruct.length* sizeof(int));
```

**Stack**

```
my_struct
```

**Heap**

```
[array] [length] 5
```
Can only be applied to values with a true location in memory (e.g. cannot do &34)

- Gets the address of the piece of memory
- Useful for allowing functions to initialize your memory
- Almost never use it, DO NOT USE IT TO “FIX” YOUR COMPILATION ERRORS

```c
void make_it_2(int *p_int) {
    *p_int = 2;
}

int main() {
    int n = 0;
    printf("%d\n", n); // Prints 0
    make_it_2(&n);
    printf("%d\n", n); // Prints 2
    return 0;
}
```
Buggy Code Exercises
#include <stdio.h>

#include <stdlib.h>

int add_sorta_maybe(int a, int b) {
    int x = a + b;
    return &x;
}

int main() {
    int* sum = add_sorta_maybe(5, 7);
    printf("sum: %d\n", *sum);
    free(sum);
    return 0;
}
#include <stdio.h>

int main() {
    struct list_node {
        struct list_node *next;
    };
    struct list_node selfloop;
    selfloop.next = &selfloop;
    struct list_node *curr = &selfloop;
    while (curr != NULL) {
        curr = curr->next;
    }
    return 0;
}  

1)  Struct declarations shouldn’t be within functions - move globally
2) While loop is infinite

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

int* add_sorta_maybe(int a, int b) {
    int x = a + b;
    return &x;
}

int main() {
    int *sum = add_sorta_maybe(5, 7);
    printf("sum: %d\n", *sum);
    free(sum);
}
buggy1.c, buggy2.c (Two issues each)

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

int* add_sorta_maybe(int a, int b) {
    int x = a + b;
    return &x;
}

int main() {
    int* sum = add_sorta_maybe(5, 7);
    printf("sum: %d\n", *sum);
    free(sum);
    return 0;
}
```

1) Struct declarations shouldn’t be within functions - move globally
2) While loop is infinite

```c
#include <stddef.h>

int main() {
    struct list_node { 
        struct list_node *next;
    };
    struct list_node selfloop;
    selfloop.next = &selfloop;
    struct list_node *curr = &selfloop;
    while (curr != NULL) {
        curr = curr->next;
    }
    return 0;
}
```

1) Pointers to stack values are invalid once the function returns, so `sum` cannot be used
2) A stack pointer is freed - only pointers returned by `malloc` can be freed
Memory Diagramming a Struct Pointer Array

struct_pointer_array.c
If time: work on last Tuesday’s consolidate_messy_ptr_array