CIS 4480/5480 Recitation 0 - C Refresher & Env. Setup SOLUTION

Welcome to recitation!!! 😃

Exercise 1: Hello World!

I wanted to print “Hello world!” in a fancier way using my new knowledge of pointers in C. For some reason, things aren’t working. Help!

```c
#include <stdio.h>

int main() {
  char** a;
  fill_a(a);
  printf("%s %s", a[0], a[1]);
}

void fill_a(char** a) {
  a[0] = "Hello";
  a[1] = "world!\n";
}
```

Questions to answer:

- Does this code compile? Why or why not?

No. There are is an issue: fill_a should either come before main or be forward declared (include only its declaration, not its definition, before main)

- What is this code trying to do?

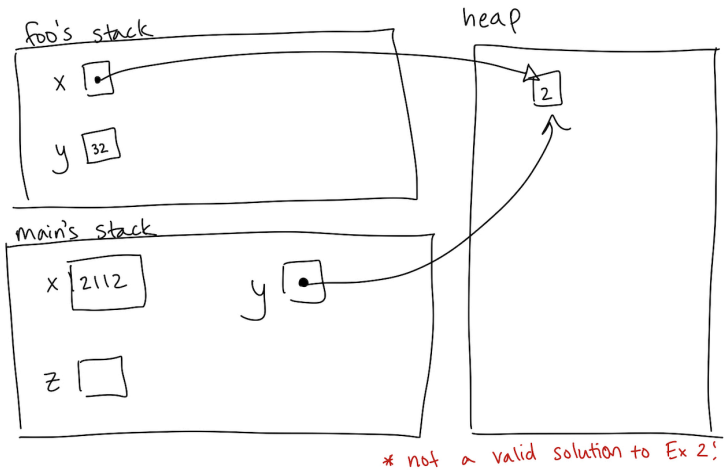
This code is trying to create an array of strings, and then set those strings, printing out the first string in the array.

- Why doesn’t this work? How would you fix it?

This code doesn’t work because declaring char** a doesn’t actually allocate memory for the array, so trying to access it using fill_a doesn’t work, as it’s out of bounds and invalid memory. This can be fixed by changing the line char** a to malloc space for the array, i.e.

char** a = malloc(sizeof(char*) * 2);

Also, using malloc requires including stdlib.h, by adding #include <stdlib.h> to the top of the code.



Exercise 2: Memory Diagram

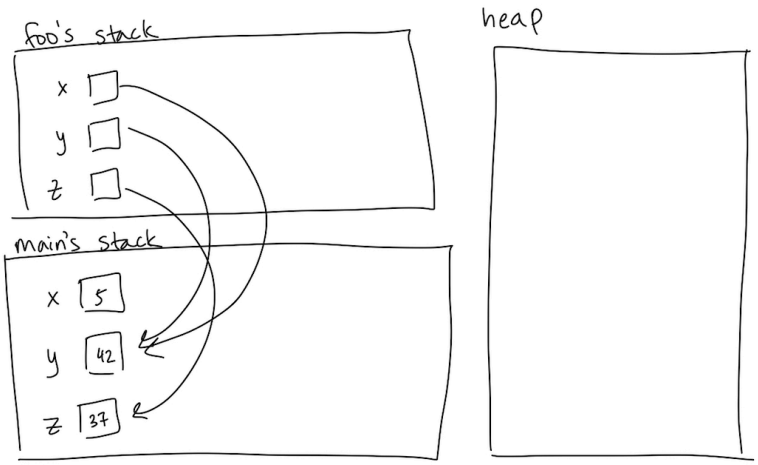
```
void foo(int32_t *x, int32_t *y, int32_t *z) {
  x = y;
  *x = *z;
  *z = 37;
}

int main() {
  int32_t x = 5, y = 22, z = 42;
  foo(&x, &y, &z);
  printf("%d, %d, %d\n", x, y, z);
  return EXIT_SUCCESS;
}
```

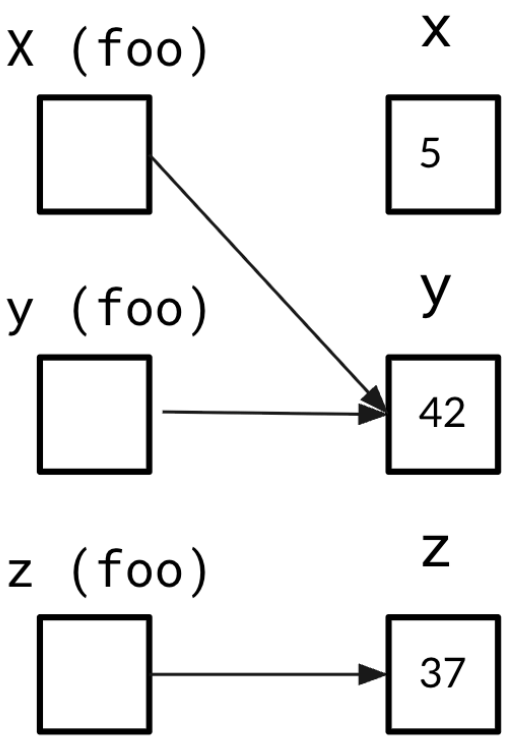
Draw a memory diagram like the one above for the following code and determine what the output will be. In the memory diagram, include what foo's stack looks like before it returns (that is, what foo looks like after running `*z = 37`).

(Solution on next page)

Many ways to draw it, here's two examples:



(don't need to include the heap to be correct)



So, the code will output 5, 42, 37.

Exercise 3: Struct Debugging

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

// define struct
struct Pokemon {
  char* name;
  int level;
};

void rename_pokemon(struct Pokemon x, char* new_name) {
  x.name = new_name;
}

int main() {
  // create struct variable + fill out fields
  struct Pokemon pikachu;
  pikachu.name = "Pikachu";
  pikachu.level = 100;

  // call our function
  rename_pokemon(pikachu, "Sparky");
  printf("%s\n", pikachu.name);
  return 0;
}
```

1. What is this code trying to do? What should the output or final state of pikachu be if everything worked as intended?

This code is trying to create a pokemon struct in main, with name = pikachu and level of 100, then change the created struct's name field to Sparky instead of Pikachu. Ideally, the code would output Sparky, instead of Pikachu.

2. Why doesn't Pikachu end up with the name "Sparky"? What happens to the value modified by rename_pokemon?

Pikachu doesn't end up with the name Sparky because rename_pokemon receives a copy of the struct by value, not by reference, so the modifications made don't persist after rename_pokemon returns.

3. What does C do when you pass a struct to a function like this? Is it passed by reference or by value?

Passing a struct as an argument to a function (or returning a struct from a function) makes a copy of that struct variable and passes that copy to the destination. Therefore, it is passed by value.

4. How could you fix this?

You could modify the code so that it passes a pointer to the struct to change rather than the struct directly (which causes it to be copied).

Exercise 4: Output Params

strcpy is a function from the standard library that copies a string src into an output parameter called dest and returns a pointer to the beginning of the destination string. Write the function below. You may assume that dest has sufficient space to store src.

```
char *strcpy(char *dest, char *src) {
  char *ret_value = dest;
  while (*src != '\0') {
    *dest = *src;
    src++;
    dest++;
  }
  *dest = '\0';  // don't forget the null terminator!
  return ret_value;
}
```

How is the caller able to see the changes in dest if C is pass-by-value?

The caller can see the copied over string in `dest` since we are dereferencing `dest`. Note that modifications to dest that do not dereference will not be seen by the caller(such as `dest++`). Also note that if you used array syntax, then `dest[i]` is equivalent to `*(dest+i)`.

Why do we need an output parameter? Why can't we just return an array we create in strcpy?

If we allocate an array inside strcpy, it will be allocated on the stack. Thus, we have no control over this memory after strcpy returns, which means we can't safely use the array whose address we've returned.

Exercise 5: GDB Debugging

In this exercise, you should use GDB to debug the program linked on the recitation files on the website.

You are given a file called pokemon_buggy.c. This program looked like it should print some info about a Pokémon… but it crashed or printed nonsense!

Questions to Answer:

1.

- What was the first bug you found and fixed?
  - There is a segfault caused by trying to dereference a NULL pointer.
- What line caused the crash?
  - Line 21 (struct Pokemon* p = malloc(sizeof(struct Pokemon));)
- Why did the program crash at that point?
  - The program tries to dereference a NULL pointer, which causes a segfault.
- What did you change to fix it?
  - Change NULL to properly malloc space for the struct.

2.

- What was the second bug you encountered after fixing the first?
  - Out of bounds memory access gives garbage value.
- Did the program crash again, or just behave strangely?
  - Behaves strangely, level is not 0, 5, 10, or 15, is a garbage value.
- How did you use gdb to identify the problem?
  - Used `print` command in GDB to print out value of index, used breakpoints to determine what line level was set to incorrect value
- Where was the invalid access? What caused it?
  - Invalid access outside of the array containing levels, caused when index was set to 3, which is equal to the length of the array.