CIT 5950 Section 0 Solutions - C, Pointers, and Docker

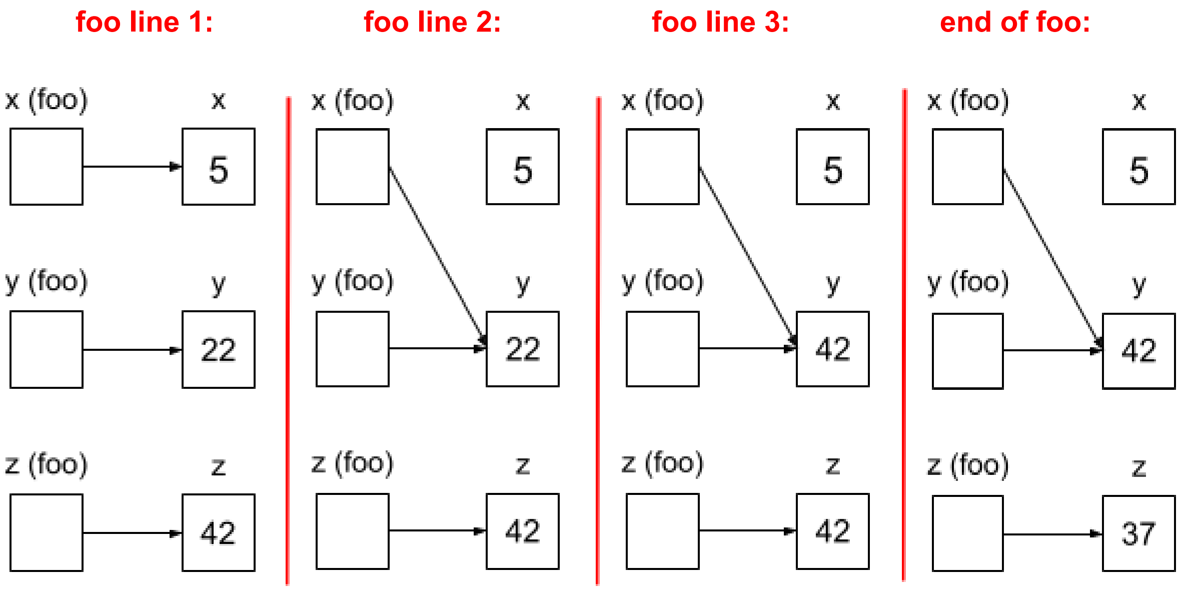
Pointers

Exercise 1:

Draw a memory diagram like the one above for the following code and determine what the output will be.

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

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



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

The following code has a bug. What's the problem, and how would you fix it?

```
void bar(char *str) {
  str = "ok bye!";
}

int main(int argc, char *argv[]) {
  char *str = "hello world!";
  bar(str);
  printf("%s\n", str);  // should print "ok bye!"
  return EXIT_SUCCESS;
}
```

The problem is that modifying the argument `str` in `bar` will not effect `str` in `main` because arguments in C are always passed by value. In order to modify `str` in `main`, we need to pass a pointer to a pointer (`char **`) into `bar` and then dereference it:

```
void bar_fixed(char **str_ptr) {
  *str_ptr = "ok bye!";
}

int main(int argc, char *argv[]) {
  char *str = "hello world!";
  bar(&str);
  printf("%s\n", str);  // should print "ok bye!"
  return EXIT_SUCCESS;
}
```

Output Parameters

Exercise 2:
strcpy is a function from the standard library that copies a string src into an output parameter called dest and returns a pointer to dest. 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.

<u>Exercise 3</u>:
More practice with output parameters and arrays.

Write a function to compute the sum of values and product of all values in an array. The function is given a pointer to the first element in an array, the length of the array, and two output parameters to return the product and sum.

```
void product_and_sum(int *input, int length, int *product,
                                                      int *sum) {
  int temp_sum = 0;
  int temp_product = 1;
  for (int i = 0; i < length; i++) {
    temp_sum += input[i];
     temp_product *= input[i];
  }
  *sum = temp_sum;
  *product = temp_product;

}
```

<u>Exercise 4</u>:

```
size_t filter(int *input, size_t length, int filter, int** out){
size_t new_len = 0;
  for (size_t i = 0; i < length; i++) {
    if (input[i] != filter) {
      new_len += 1;
    }
  }
  int* res = new int[new_len];
  size_t j = 0;
  for (size_t i = 0; i < length; i++) {
    if (input[i] != filter) {
      res[j] = input[i];
      j += 1;
    }
  }
  *out = res;
  return new_len;
}
```