CS1010 Tutorial 7

Agenda

- Assignment 4 - Comments
- Problem Set 17
- Problem Set 18
- Problem Set 19
- Assignment 5 - Social

Assignment 4 Comments

Selection Sort

- The `selection_sort` methods were usually too cluttered
- Split up the logic into different functions
  - Finding the max for a sub-array
  - Printing the array to the standard output
  - Swapping two values in the array

Selection Sort

```c
// Find max in the range list[start] to list[end] inclusive
long find_max(long list[], long start, long end);

// Swaps the element in list at indices x and y
void swap(long list[], long x, long y);

// Prints a whitespace-separated long array to the standard output
void print_array(long list[], long len);

void selection_sort(long list[], long len)
{
    print_array(list, len); // print input array
    for (long i = len - 1; i >= 0; i -= 1) {
        long max_index = find_max(list, 0, i);
        swap(list, max_index, i);
        print_array(list, len);
    }
}
```

Mastermind

- Failing solutions tend to
  - Forget to "mark" the elements that have already been considered
  - Did not increment variables correctly
- There are many ways to do this question
- My proposed solution requires two additional arrays besides G and A
- Denote two boolean arrays G' and A' with the same size as G and A respectively
  - Check for same color and same position
  - Check for same color, but do not double count elements considered in the previous step

Mastermind

- First, check for same color and same position
- Since we're checking this first, we don't need to check if any element is marked
  - The boolean arrays are reset at every round of playing the game
- Mark the corresponding positions in *both* G' and A'

```
#define N 4
long n_same_color_pos(char *code, char *guess,
                      bool is_marked_code[N], bool is_marked_guess[N])
    long count = 0;
    for (long i = 0; i < N; i += 1)
        if (code[i] == guess[i])
            is_marked_code[i] = true;
            is_marked_guess[i] = true;
            count += 1;
    return count;
```

Braces removed for brevity

Mastermind

- Then, check for same color but different positions
- If either position in G' or A' is marked, then any matching pair is discarded

```
long n_same_color(char *code, char *guess,
                  bool is_marked_code[N], bool is_marked_guess[N])
    long count = 0;
    for (long i = 0; i < N; i += 1)
        for (long j = 0; j < N; j += 1)
            if (code[i] == guess[j] && !is_marked_code[i] && !is_marked_guess[j]) {
                is_marked_code[i] = true;
                is_marked_guess[j] = true;
                count += 1;
                break;
    return count;
```

Braces removed for brevity

Problem Set 17

Call By Reference

Problem 17.1

Complete the function `find_min_max` that takes in a length and an array containing long values of size `length`, and update the parameter `min` and `max` with the minimum and the maximum value from this array, respectively.

Show how to call this function from main.

```
void find_min_max(long length, long array[length], long *min, long *max)
{
    :
}

int main()
{
    long list[10] = {1, 2, 3, 4, -4, 5, 6, -8, 3, 1};
    :
}
```

Problem 17.1

- This is a simple pointers exercise
- Practice how to pass a variable "by reference" to another function
- Usually we do this if
    - We need to return more than 1 value at a time from a function
    - We want the function to pass back the result of some logic, but the return of a function is its *error code*

Problem 17.1

```c
void find_min_max(long length, long array[length], long *min, long *max)
{
    *min = array[0];
    *max = array[0];
    for (long i = 1; i < length; i += 1) {
        if (array[i] < *min) {
            *min = array[i];
        }
        if (array[i] > *max) {
            *max = array[i];
        }
    }
}
// in main()
long list[10] = {1, 2, 3, 4, -4, 5, 6, -8, 3, 1};
long min, max;
find_min_max(10, list[10], &min, &max);
```

If you get confused about passing pointers into methods, refer back to this question

Problem 17.2

What would be printed in the program below?

```c
void foo(double *ptr, double trouble) {
    ptr = &trouble;
    *ptr = 10.0;
}

int main() {
    double *ptr;
    double x = -3.0;
    double y = 7.0;
    ptr = &y;

    foo(ptr, x);

    cs1010_println_double(x);
    cs1010_println_double(y);
}
```

Problem 17.2

What would be printed in the program below?

```c
void foo(double *ptr, double trouble) {
    ptr = &trouble; // ptr pointing to trouble (local variable)
    *ptr = 10.0; // change trouble to 10.0
}

int main() {
    double *ptr;
    double x = -3.0;
    double y = 7.0;
    ptr = &y; // ptr pointing to y

    foo(ptr, x);

    cs1010_println_double(x); // -3.0
    cs1010_println_double(y); // 7.0
}
```

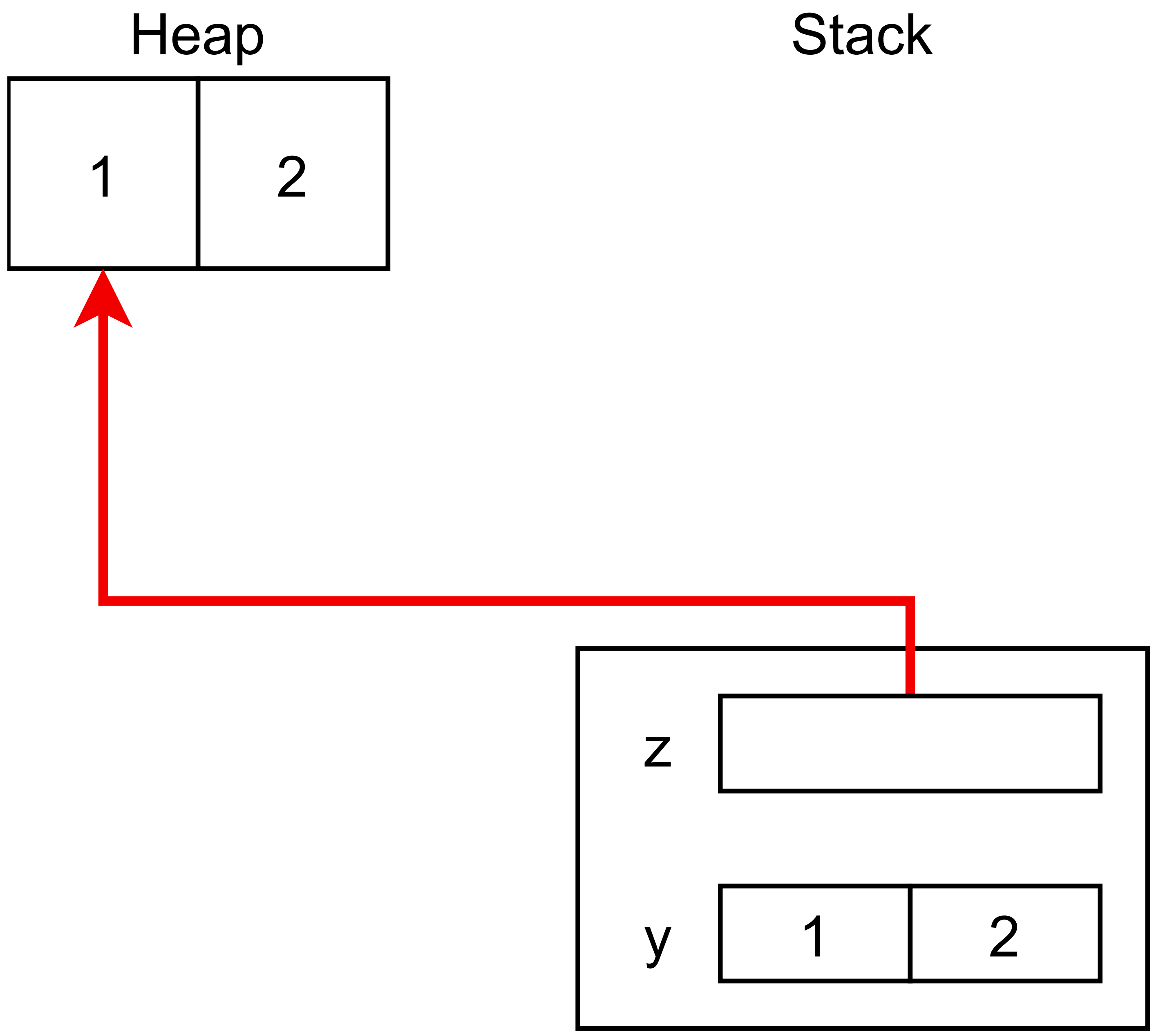
Problem Set 18

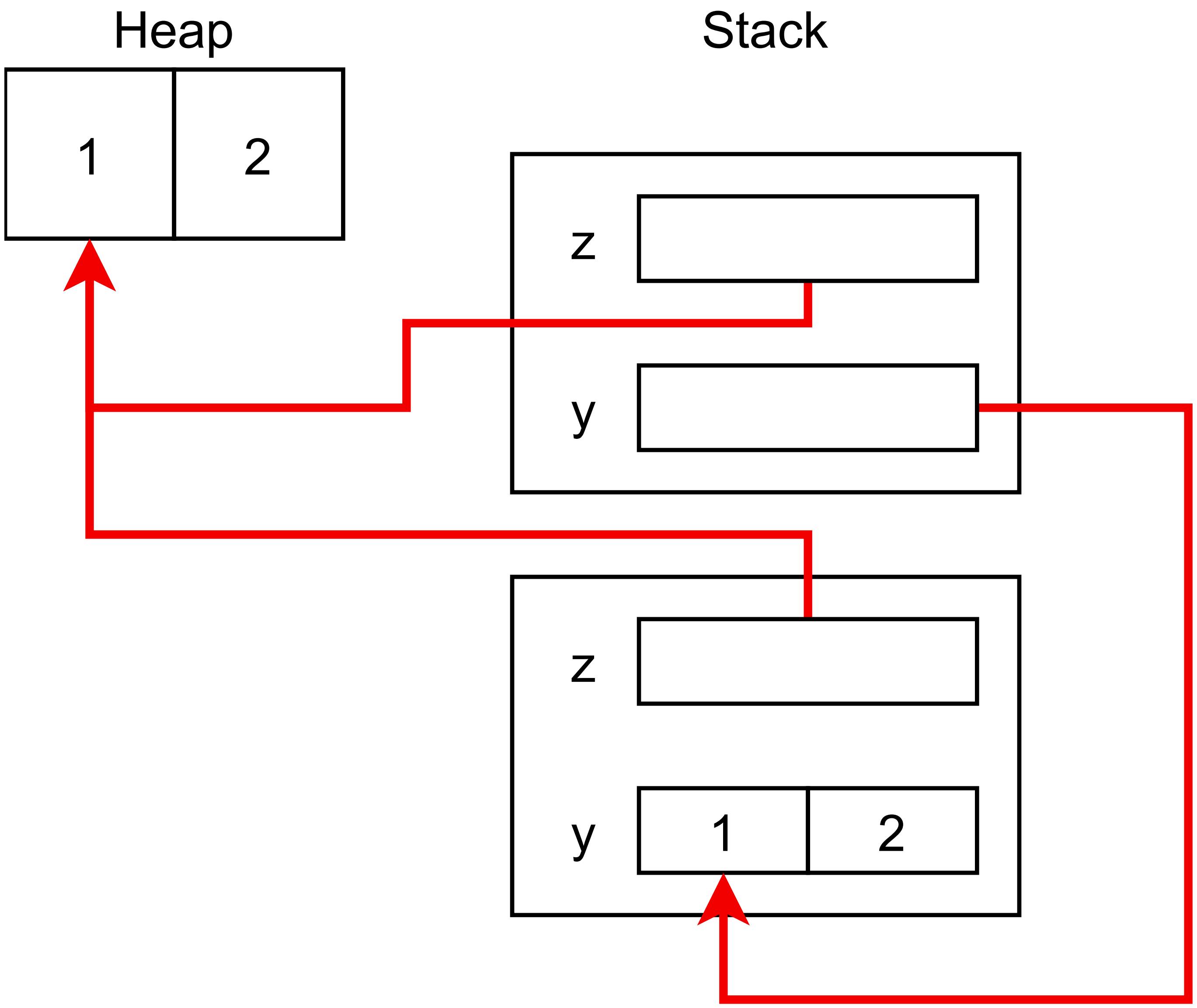
Heap

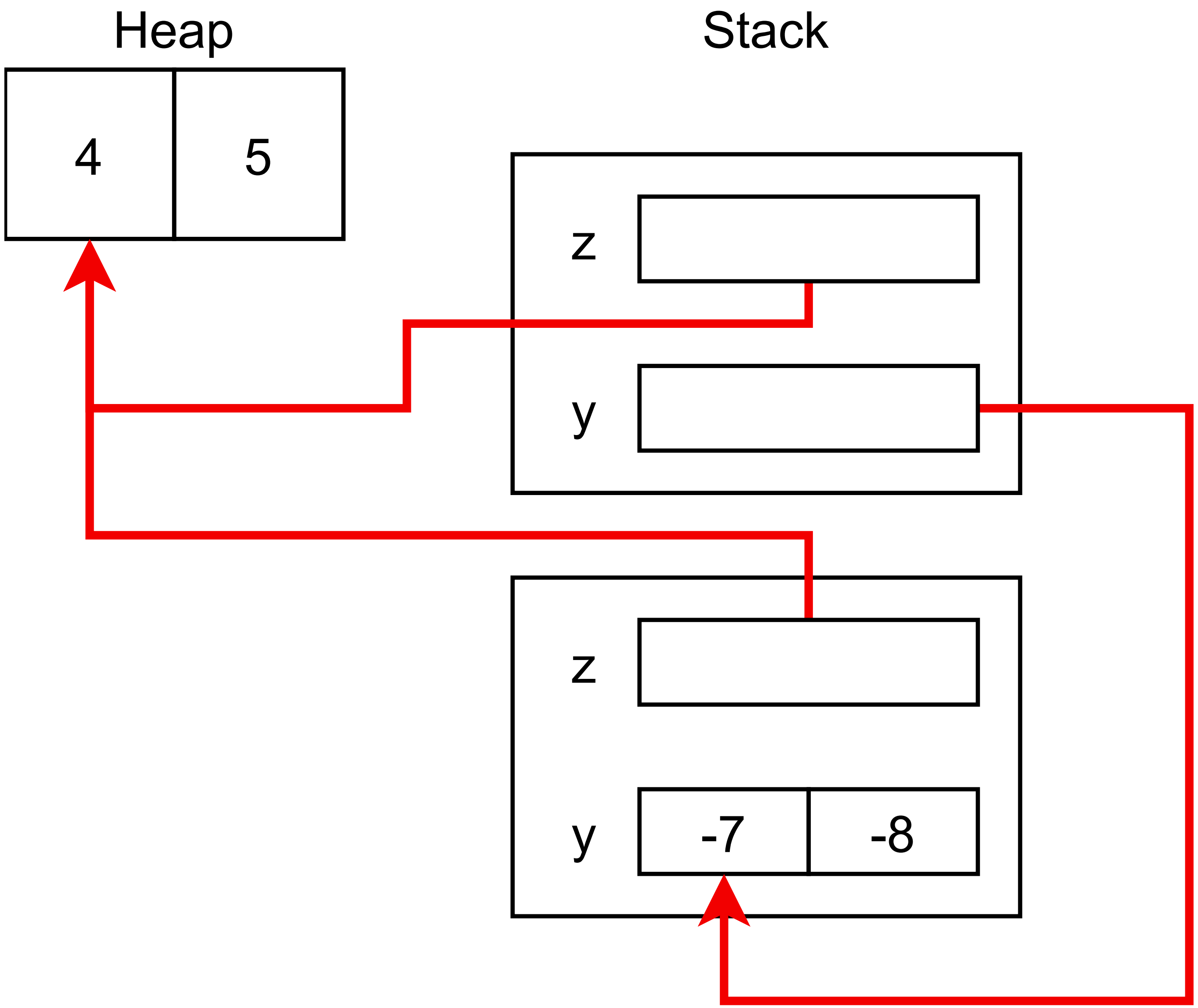
Problem 18.1

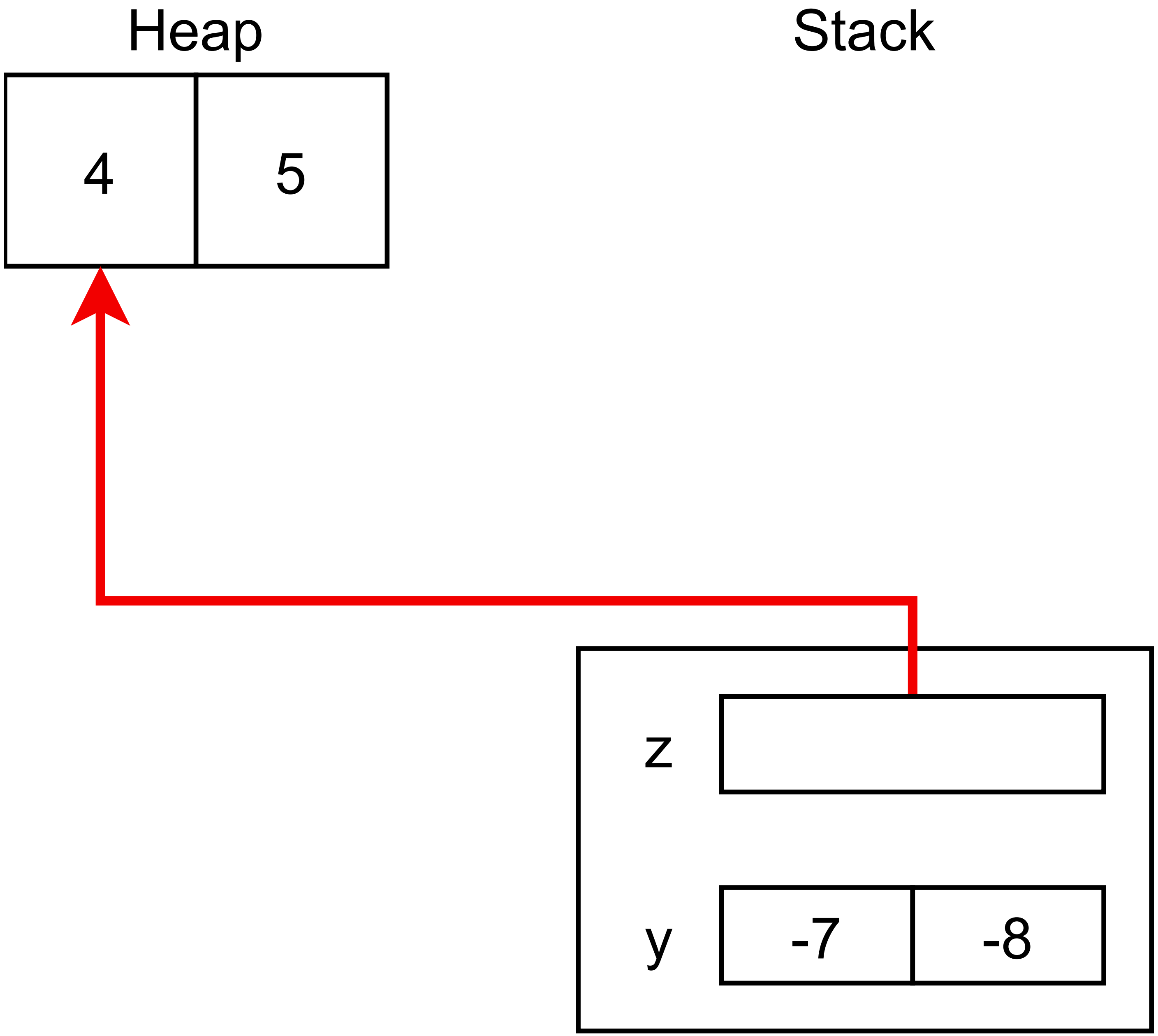
Draw the call stack and heap for the following code

```
void foo(long *y, long *z)
    y[0] = -7;
    y[1] = -8;
    z[0] = 4;
    z[1] = 5;

int main()
    long y[2] = {1, 2};
    long *z = calloc(2, sizeof(long));

    z[0] = y[0];
    z[1] = y[1];

    foo(y, z);
```


Heap
Stack
1
2
z
y
1
2


Heap
Stack
1
2
z
y
z
y
1
2


Heap
4
5
Stack
z
y
z
y
-7
-8


Heap
Stack
4
5
z
y
-7
-8

Problem Set 18.2

Read the man page for the function `realloc` and explain what does it do. Can you come up with a situation where it could be useful?

Problem Set 18.2 - `realloc`

- `void *realloc(void *ptr, size_t size)`

What does it do?

- Modifies the size of the memory pointed to by `ptr` to `size` number of bytes
- `realloc` does not change the contents of memory from offset `0` until `min(old_size, size)`
- Does not re-initialize new memory if `realloc` increases the amount of memory
- If `ptr` is `NULL`, then the call is equivalent to `malloc(size)`
- If `size == 0`, and `ptr != NULL`, then the call is equivalent to `free(ptr)`
- Unless `ptr == NULL`, `ptr` must be a pointer returned by an earlier call to `malloc`, `calloc` or `realloc`
- If the area pointed to was moved, then the call is equivalent to `free(ptr)`

Problem Set 18.2 - `realloc`

- What is a probable use case?
- Example : **Reading from a very large file** (without prior knowledge of the file size)
- First allocate a buffer of n bytes using `malloc` or `calloc`
  - Read and store until the end of the buffer
  - If we need more space, call `realloc` and double the amount of memory allocated (e.g $2n$ number of bytes)
  - Once done, `realloc` down to the correct size
- This is how `cs1010_read_word()` works internally

Problem Set 19

Multi-dimensional Arrays

Problem 19.1

1. Write a function `add` that performs 3x3 matrix addition
2. Write a function `multiply` that performs 3x3 matrix multiplication

Problem 19.1 (a)

Write a function `add` that performs 3x3 matrix addition

Recall for matrix addition:

$$\begin{bmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ c_0 & c_1 & c_2 \end{bmatrix} + \begin{bmatrix} i_0 & i_1 & i_2 \\ j_0 & j_1 & j_2 \\ k_0 & k_1 & k_2 \end{bmatrix} = \begin{bmatrix} a_0 + i_0 & a_1 + i_1 & a_2 + i_2 \\ b_0 + j_0 & b_1 + j_1 & b_2 + j_2 \\ c_0 + k_0 & c_1 + k_1 & c_2 + k_2 \end{bmatrix}$$

i.e for two matrices A and B, and the result matrix being R

$$R_{ij} = A_{ij} + B_{ij}$$

Problem 19.1 (a)

```
void add(long **m1, long **m2, long **result)
{
    for (long i = 0; i < 2; i += 1) {
        for (long j = 0; j < 2; j += 1) {
            result[i][j] = m1[i][j] + m2[i][j];
        }
    }
}
```

Problem 19.1 (b)

Write a function `multiply` that performs 3x3 matrix multiplication

Recall for matrix multiplication:

- Let A and B be two 3x3 matrices, and the result matrix be R
- Then, for each $R[i][j]$,

$$R_{ij} = \sum_{k=1}^{3} A_{ik} \cdot B_{kj}$$

Problem 19.2(b)

```c
void calculate_ij(long **m1, long **m2, long **result, long i, long j)
{
    for (long k = 0; k < 3; k++) {
        result[i][j] += m1[i][k] * m2[k][j];
    }
}

void multiply(long **m1, long **m2, long **result)
{
    for (long i = 0; i < 3; i++) {
        for (long j = 0; j < 3; j++) {
            calculate_ij(m1, m2, result, i, j);
        }
    }
}
```

A Note on Efficiency

Given an $n \times n$ matrix,

- Matrix addition runs in $\mathcal{O}(n^2)$ time
- Naive matrix multiplication runs in $\mathcal{O}(n^3)$ time
  - The fastest known matrix multiplication algorithms runs in $\mathcal{O}(n^{2.3737})$
  - The more common one learnt in algorithms classes is Strassen's Algorithm that runs in $\mathcal{O}(n^{2.807})$ time

Source

Problem 19.2

1. We have a list of n cities
2. We have the distance between every pair of cities
   i. The distance between any city $i \to j$ is the same as from $j \to i$
   ii. The distance can be represented with a `long`

Explain how you would represent this information with a *jagged* 2-D array in C efficiently.

Write a function `long dist(long **d, long i, long j)` to retrieve the distance between any cities i and j.

Problem 19.2

This problem is extremely important for Assignment 5 Social

Problem 19.2

> Explain how you would represent this information with a *jagged* 2-D array in C efficiently.

For a jagged 2-D matrix M, each M_{ij} represents the distance between cities i and j. If $i = j$, then the distance is 0.

```
   0  1  2  3  4  5
0  0
1  3  0
2  9  5  0
3  7  1  5  0
4  8  2  6  4  0
5  8  2  4  1  8  0
```

Problem 19.2

> Write a function `long dist(long **d, long i, long j)` to retrieve the distance between any cities i and j.

```c
long dist(long **d, long i, long j)
{
    if (i < j) {
        return d[j][i];
    }
    return d[i][j];
}
```

Assignment 5 Tips

Question 2 - Social

One of the most difficult assignment question in all the assignments

Problem Statement

- We are given a friend network represented by a jagged 2-D array, call it D_1
- Each person i and another person j are friends with each other if the location $D_1[i][j]$ in the jagged 2-D array is `'1'`, they are not friends otherwise (`'0'`)
- We say that these two persons i and j are 1 "hop" away from each other
- What about 2 hops away?
  - There's exists a person k that both i and j are friends with. Then we can reach j from i using k as an intermediary i.e $i \to k \to j$.
- We wish to generalise this to k-hops

Input and Output

- We are given three parts of input
  - n - the number of people in the network
  - k - the degree of hops we wish to find
  - A 2-D jagged matrix that should be stored in a `char**` variable
    - Denote this matrix as D_1, as it is a network of degree 1
- We should output
  - The resulting friend network represented by k hops, i.e D_k

Separating Representation from Operation

- How do we check if two persons i and j are friends with each other?
- We check the jagged matrix at indices $D_1[i][j]$
- But since it's jagged, we will access index-out-of-bounds if $i < j$
- **We should create a function similar to `dist` from Problem 19.2**

```c
#define FRIEND '1'

bool is_friend(char **network, long i, long j)
{
    if (i < j) {
        return network[j][i] == FRIEND;
    }
    return network[i][j] == FRIEND;
}
```

- Now we don't have to worry about how we access the matrix when we loop

How do we find a network of degree 2?

- This is relevant for the first question (Contact)
- Given two persons i and j, does there exist a person m for which both i and j are contacts with?
- How do we find this out?
- Use the `is_friend` function to check if i and m are friends, and if m and j are friends
- For each possible person m
  - If m is a friend of i, and m is a friend of j, then i and j are contacts
  - If both are true, then i and j are 2 hops away from each other
- Call the above algorithm for every pair i and j and store the result in D_2

What about a network of degree 3?

- Given two persons i and j, does there exist a person m for which m is 1 hop away from i, and 2 hops away from j?
- We already have D_2
  - If there exists a person m that is a common friend of i and j in D_2, then i and j are 3 hops away from each other (Think why this is so)
- **Use the degree 2 network and the degree 1 network to compute a 3-hop network**
- Generalize to h hops (next slide)

h-hops

- Given i and j, does there exist a person m which is 1 hop away from i, but is $(h-1)$ hops away from j?
- If we need to find the network of h
  - We need the network of degree $h-1$ and a network of degree 1
  - You always have the latter (it is the input), but the former must be computed iteratively

Further Reading

- Social is actually a thinly-veiled graph theory question
- The best solution to Social is an algorithm known as a Breadth-First Traversal.
  - However, such a solution is not required nor expected of you to learn
  - Most implementations are done on a data structure known as an adjacency list
- The third question, Life, is significantly easier than Social