CS1010 Tutorial 6
Group BC1A

8 October 2020

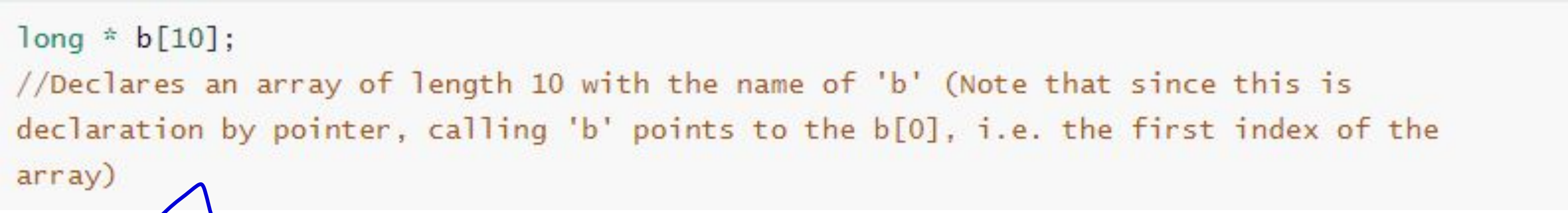
Topics for today

Objectives

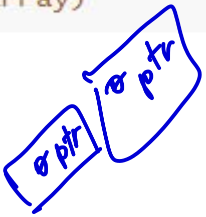
- Recap on Topics (String, Call by reference, Heap, Multi-Dimensional arrays)
- Going through problem set 16, 17, 18, 19
- Summary

Arrays (Correction)

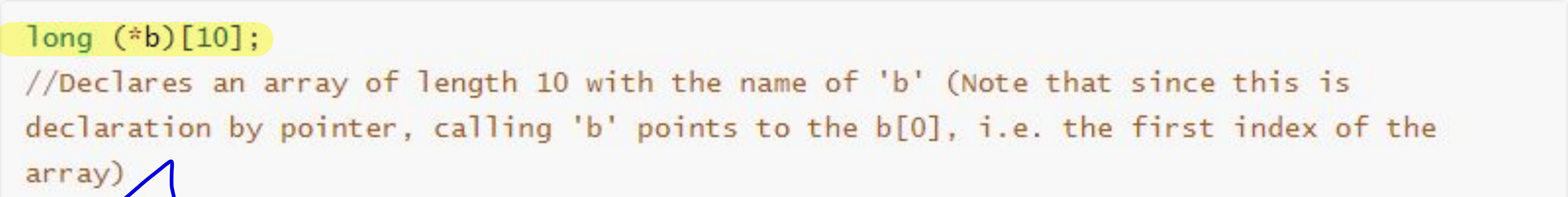
- Creating a new array with pointers



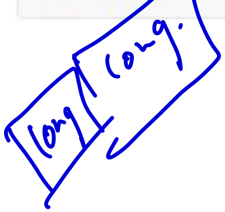
```
long * b[10];
//Declares an array of length 10 with the name of 'b' (Note that since this is
declaration by pointer, calling 'b' points to the b[0], i.e. the first index of the
array)
```



- Creating a new array with pointers

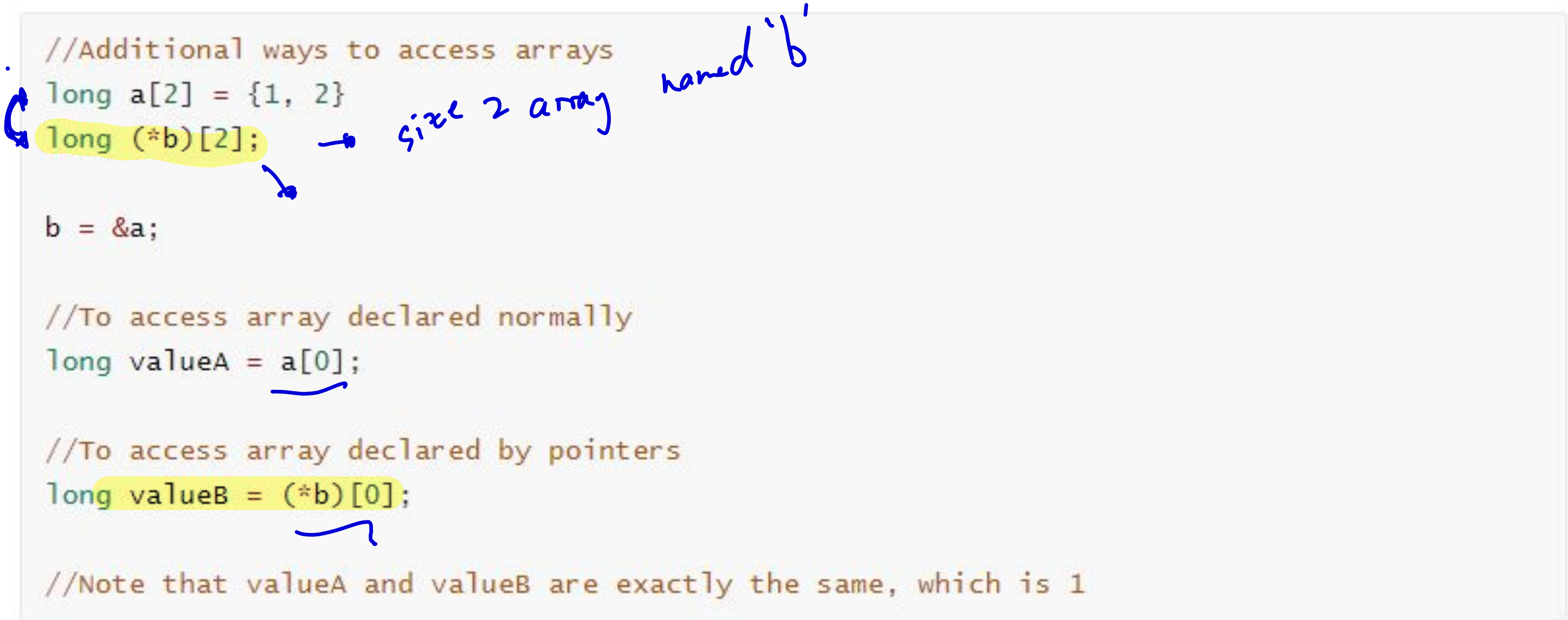


```
long (*b)[10];
//Declares an array of length 10 with the name of 'b' (Note that since this is
declaration by pointer, calling 'b' points to the b[0], i.e. the first index of the
array)
```



Arrays

- Ways to access the array

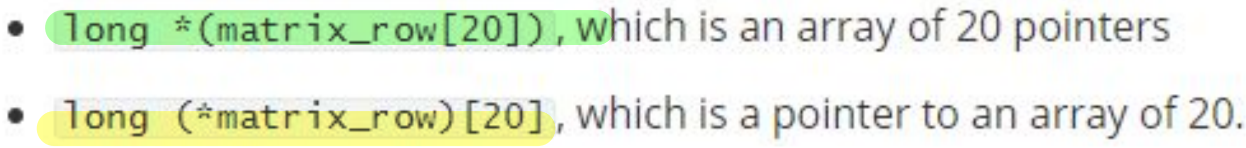


```
//Additional ways to access arrays
long a[2] = {1, 2}
long (*b)[2];

b = &a;

//To access array declared normally
long valueA = a[0];

//To access array declared by pointers
long valueB = (*b)[0];

//Note that valueA and valueB are exactly the same, which is 1
```

same size.

→ size 2 array named 'b'

- Should be able to identify the difference between



- `long *(matrix_row[20])`, which is an array of 20 pointers
- `long (*matrix_row)[20]`, which is a pointer to an array of 20.

Strings

- Note that strings are just an array of characters
- The last character of a string is always `'\0'` (Also known as a null character)
  - This means always +1 to the string (i.e. `"hello"` needs an array of size 6 to store the null character)
- CS1010 I/O Library
  - `cs1010_read_line()`
  - `cs1010_read_word()`
  - `cs1010_read_line_array()`
  - `cs1010_read_word_array()`

hello $\boxed{\backslash 0}$

hello _ ____

hello \0

Problem 16.1 Question

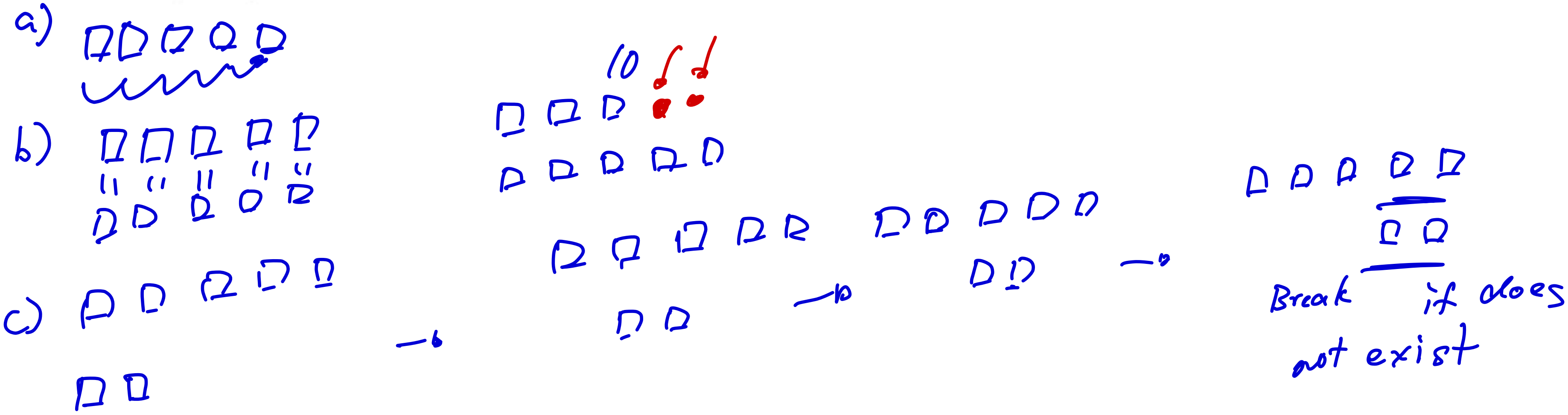
Write the following functions (without calling the standard C functions declared in <string.h> such as strlen, strcmp, strstr):

len()

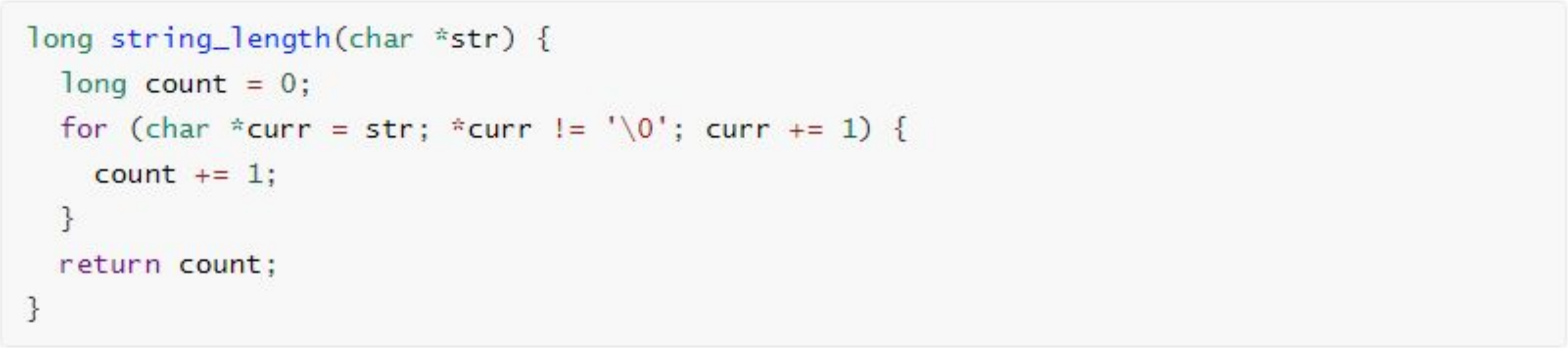
(a) long string_length(char *str) return the length (i.e., the number of characters) of the string str.

(b) bool string_equal(char *str1, char *str2) return true if the two strings str1 and str2 contains exactly the same content, false otherwise. (Note: str1 ~~==~~ str2 does not compare if two strings have the same content. (Why?))

(c) char *string_in_string(char *needle, char *haystack) returns a pointer to the first character of the first occurrence of needle in haystack if found. If needle does not occur anywhere in haystack, return NULL. If needle is an empty string, haystack is returned.

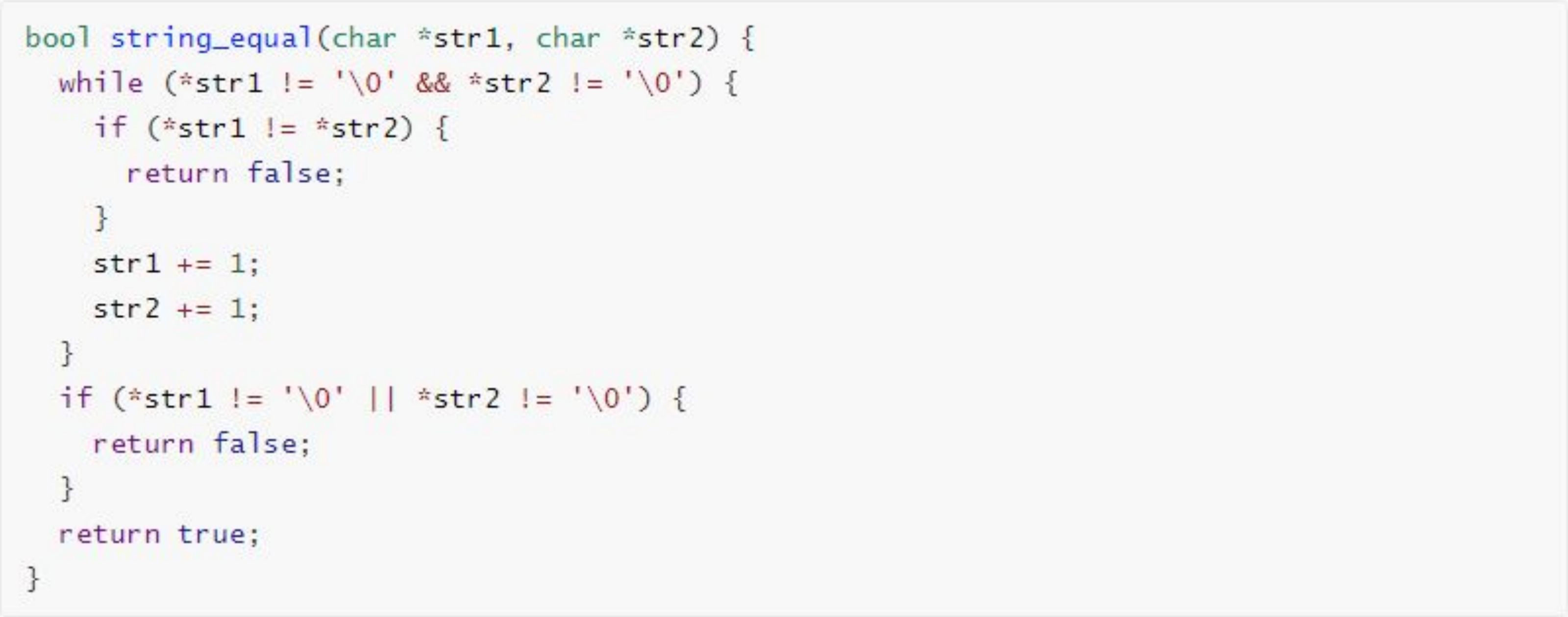


Problem 16.1(a) Answer



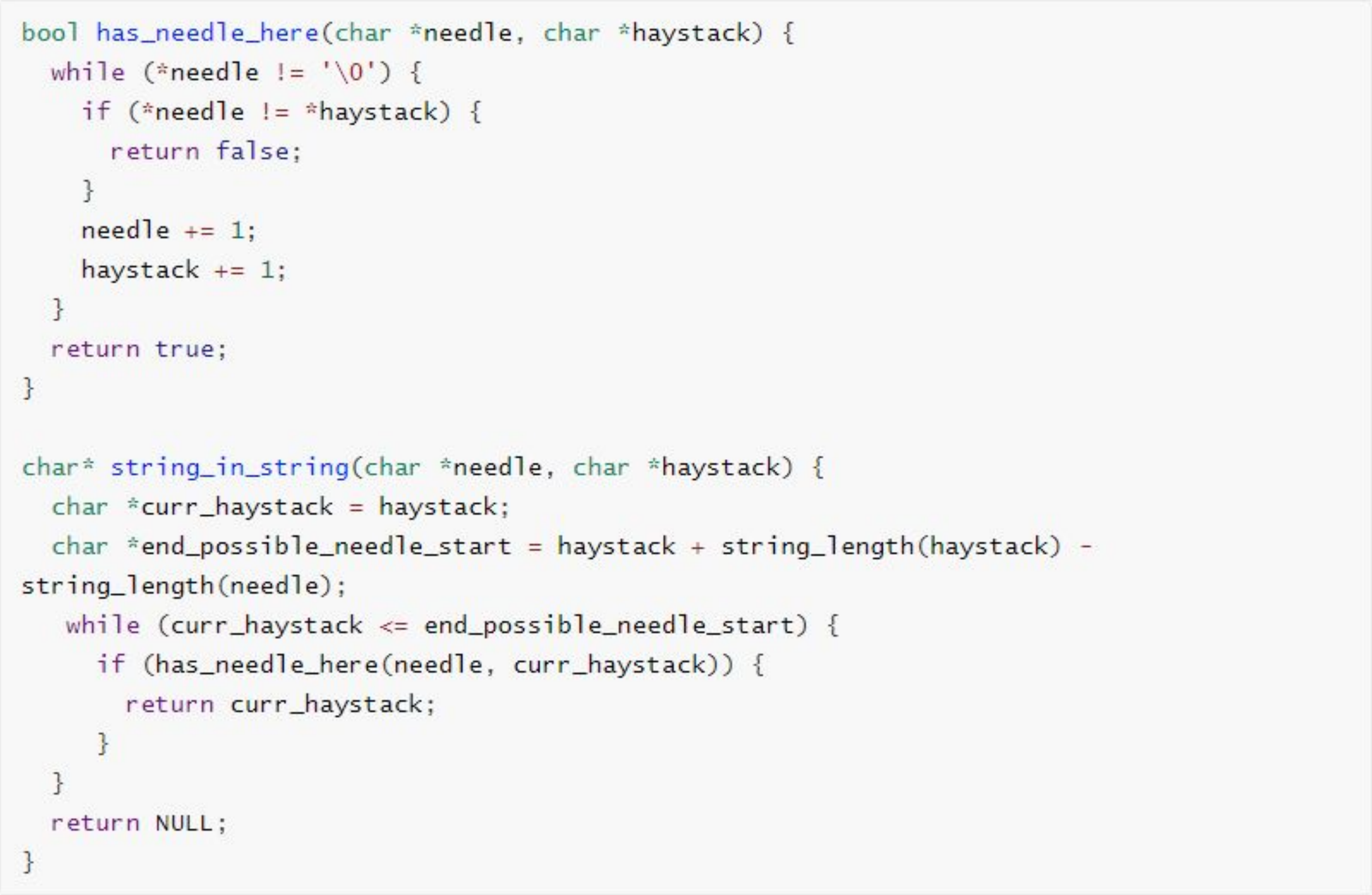
```
long string_length(char *str) {
  long count = 0;
  for (char *curr = str; *curr != '\0'; curr += 1) {
    count += 1;
  }
  return count;
}
```

Problem 16.1(b) Answer



```
bool string_equal(char *str1, char *str2) {
  while (*str1 != '\0' && *str2 != '\0') {
    if (*str1 != *str2) {
      return false;
    }
    str1 += 1;
    str2 += 1;
  }
  if (*str1 != '\0' || *str2 != '\0') {
    return false;
  }
  return true;
}
```

Problem 16.1(c) Answer



```
bool has_needle_here(char *needle, char *haystack) {
  while (*needle != '\0') {
    if (*needle != *haystack) {
      return false;
    }
    needle += 1;
    haystack += 1;
  }
  return true;
}

char* string_in_string(char *needle, char *haystack) {
  char *curr_haystack = haystack;
  char *end_possible_needle_start = haystack + string_length(haystack) -
string_length(needle);
   while (curr_haystack <= end_possible_needle_start) {
     if (has_needle_here(needle, curr_haystack)) {
       return curr_haystack;
     }
  }
  return NULL;
}
```

Pass by reference

~~t i ++;~~
~~t ++j;~~
~~t j += j~~
~~what is t;~~

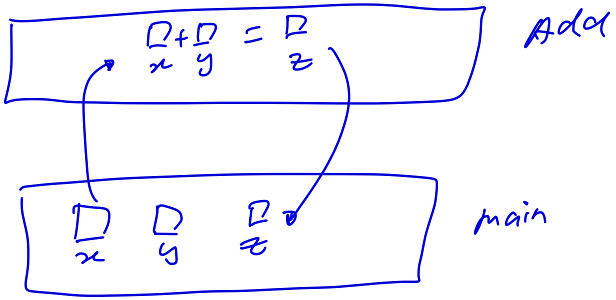
- Pure function are functions that does not affect variables in the other scopes when executing
- Side effects are the effects and changes to variables in other scopes when functions are executing
- The keyword `const` ensures that the variable will not be modified
- Document using `Doxygen` format
  - `@param[in]` for read only
  - `@param[out]` for write only
  - `@param[in, out]` for both read and write

~~i++~~
i += 1
~~++i~~

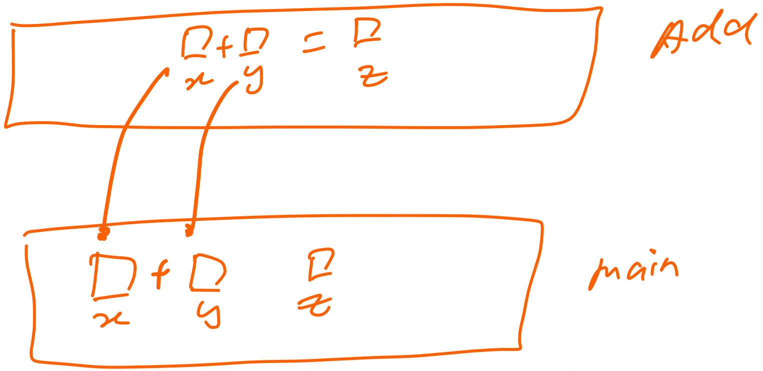
x += y
z = x
Return &z

without Ref good!

Bad!

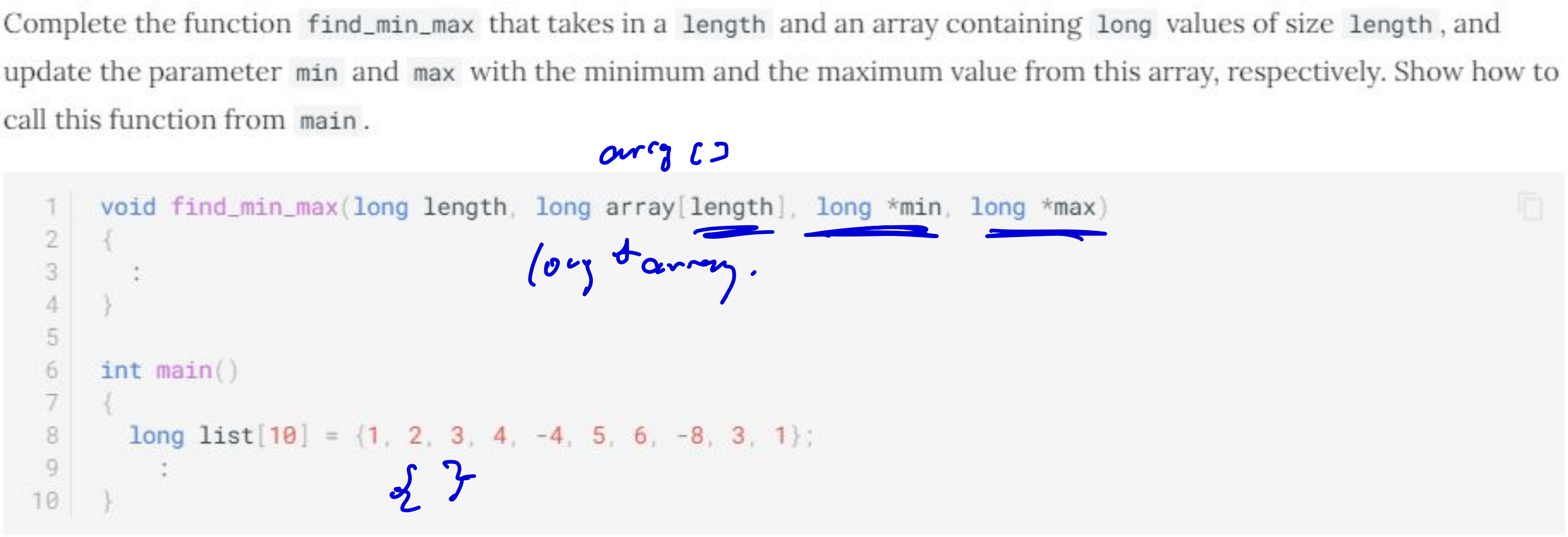


new x = x   z = x+y.
new y = y



New x = x+y.   z = x.
new y = y

Problem 17.1 Question



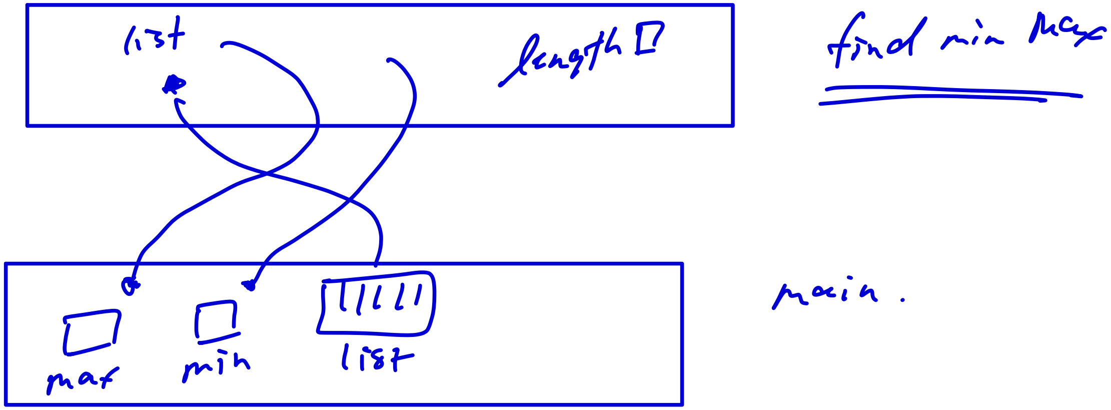
Complete the function `find_min_max` that takes in a `length` and an array containing `long` values of size `length`, and update the parameter `min` and `max` with the minimum and the maximum value from this array, respectively. Show how to call this function from `main`.

```
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};
  :
}
```

array[]

long *array.

{ }

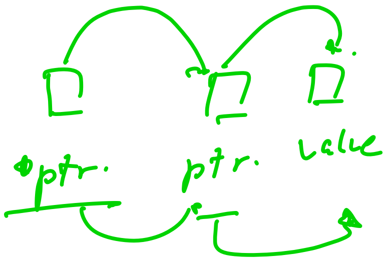


Problem 17.1 Answer

```
#include <limits.h>

...

void find_min_max(long length, long array[length], long *min, long *max)
{
  *max = -LONG_MAX;
  *min = LONG_MAX;

  for (long i = 0; i < length; i++) {
    if (array[i] > *max) {
      *max = array[i];
    }
    if (array[i] < *min) {
      *min = array[i];
    }
  }
}
```

** array.

Problem 17.2 Question



Consider the program below:

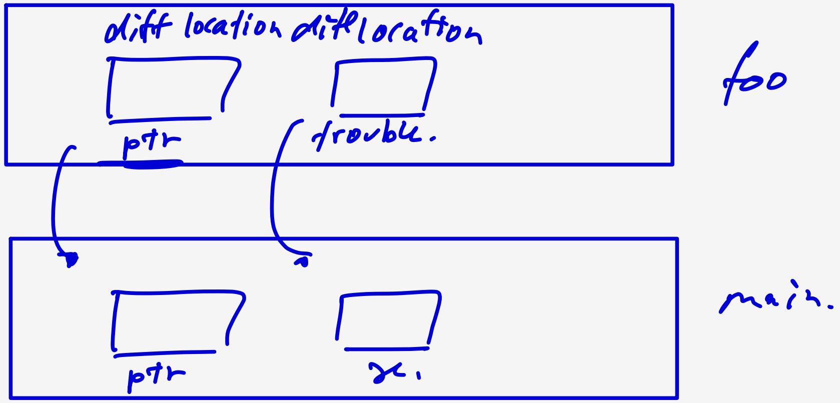
copy of ptr.

copy of x.

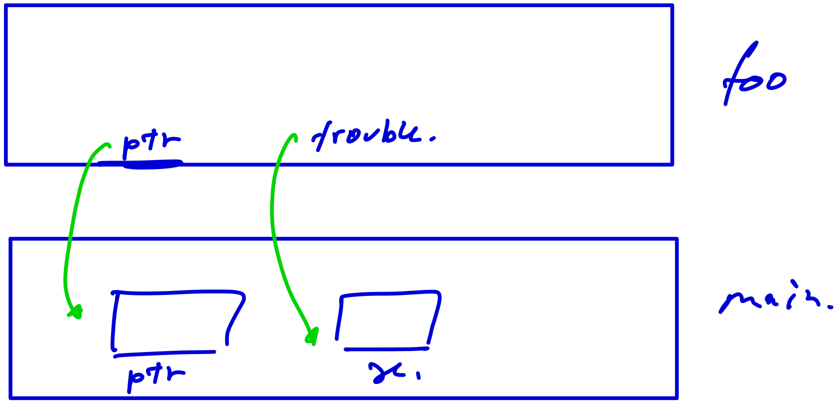
```
void foo(double *ptr, double trouble) {
  ptr = &trouble;
  *ptr = 10.0;
}

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);
}
```

double ** ptr, double * trouble.

foo(&ptr, &x)

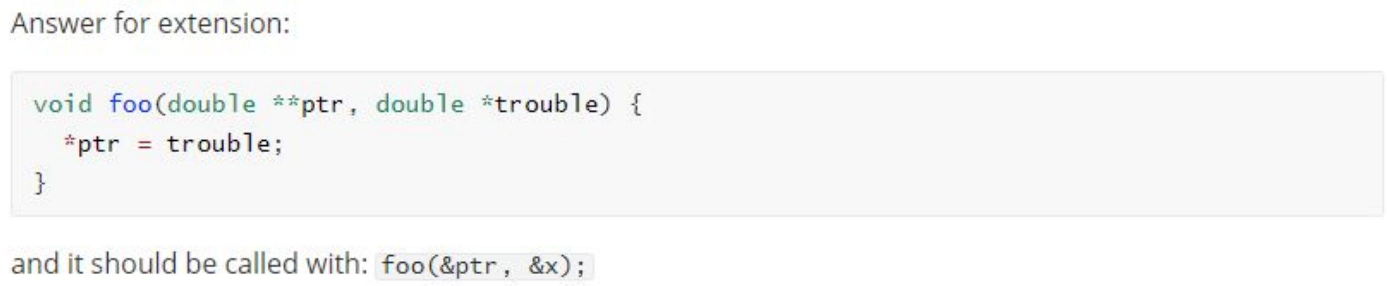


What would be printed?



Problem 17.2 Answer

- Both ptr and x will not change as foo is updated its own copy of ptr and x, instead of the copy from main
- (Extension) How do we make it such that foo updates ptr and x from main?

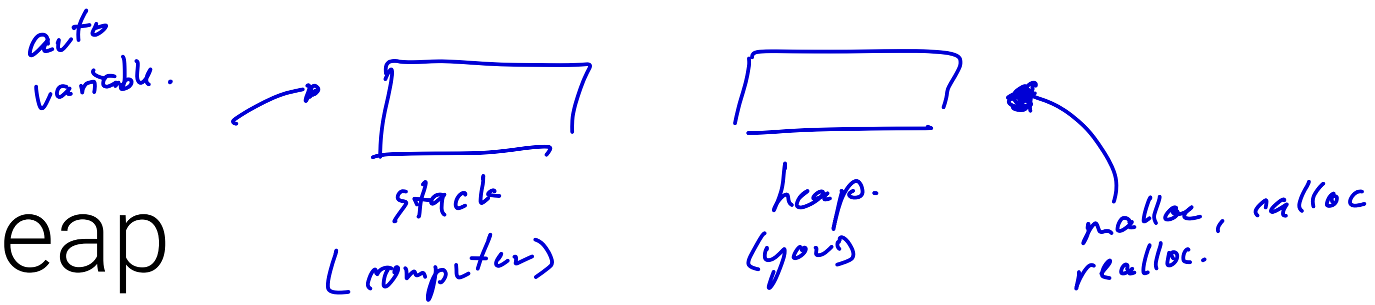


Answer for extension:

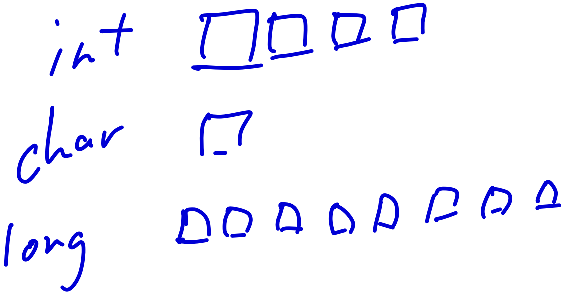
```
void foo(double **ptr, double *trouble) {
  *ptr = trouble;
}
```

and it should be called with: `foo(&ptr, &x);`

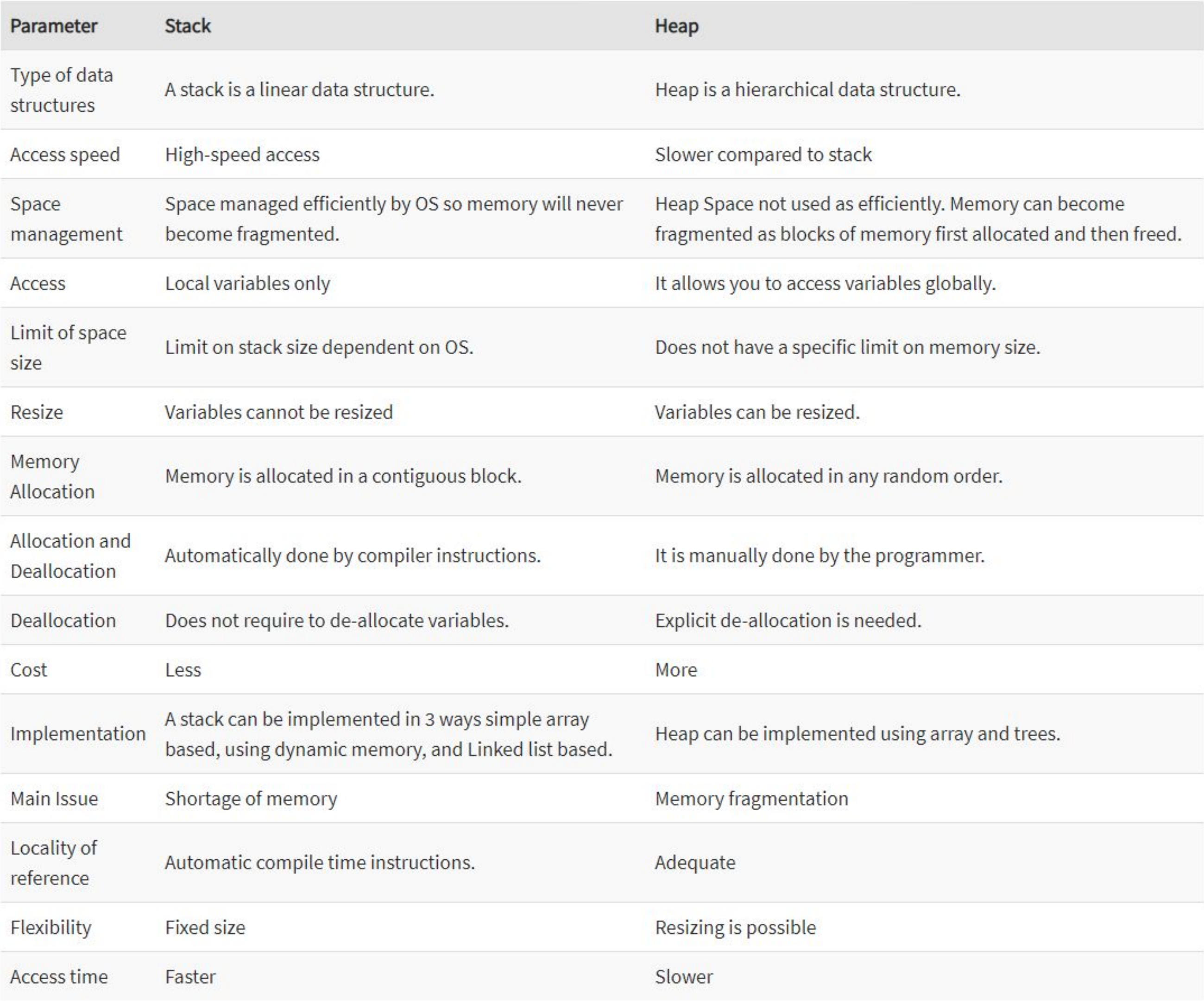
Heap



- Take note of the difference between heap and stack
  - A heap is memory that can be allocated by users
  - A stack is memory that can only be allocated by computer
  - Learn more here:
    - https://www.guru99.com/stack-vs-heap.html#:~:text=Stack%20is%20a%20linear%20data,you%20to%20access%20variables%20globally.
- Understand that memory space are allocated automatically for variables during declaration (Auto Variables)
- Use `malloc` and `calloc` to allocate memory to variables manually
- Note that `malloc` and `calloc` takes in byte size, not the actual size
  - Use `sizeof()` to help you get the byte size of any `type`



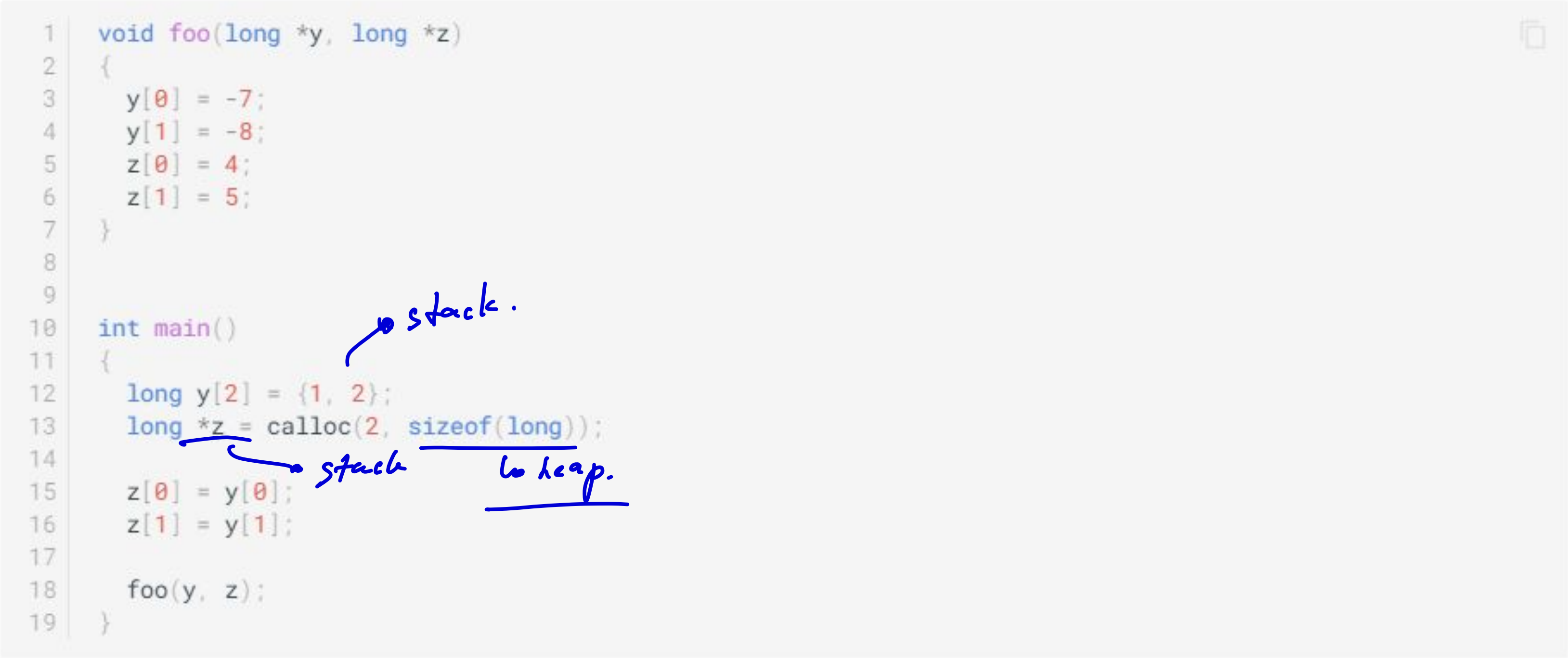
Heap



Parameter	Stack	Heap
Type of data structures	A stack is a linear data structure.	Heap is a hierarchical data structure.
Access speed	High-speed access	Slower compared to stack
Space management	Space managed efficiently by OS so memory will never become fragmented.	Heap Space not used as efficiently. Memory can become fragmented as blocks of memory first allocated and then freed.
Access	Local variables only	It allows you to access variables globally.
Limit of space size	Limit on stack size dependent on OS.	Does not have a specific limit on memory size.
Resize	Variables cannot be resized	Variables can be resized.
Memory Allocation	Memory is allocated in a contiguous block.	Memory is allocated in any random order.
Allocation and Deallocation	Automatically done by compiler instructions.	It is manually done by the programmer.
Deallocation	Does not require to de-allocate variables.	Explicit de-allocation is needed.
Cost	Less	More
Implementation	A stack can be implemented in 3 ways simple array based, using dynamic memory, and Linked list based.	Heap can be implemented using array and trees.
Main Issue	Shortage of memory	Memory fragmentation
Locality of reference	Automatic compile time instructions.	Adequate
Flexibility	Fixed size	Resizing is possible
Access time	Faster	Slower

Problem 18.1 Question

Draw the call stack and the heap, showing what happened when we run the following code:



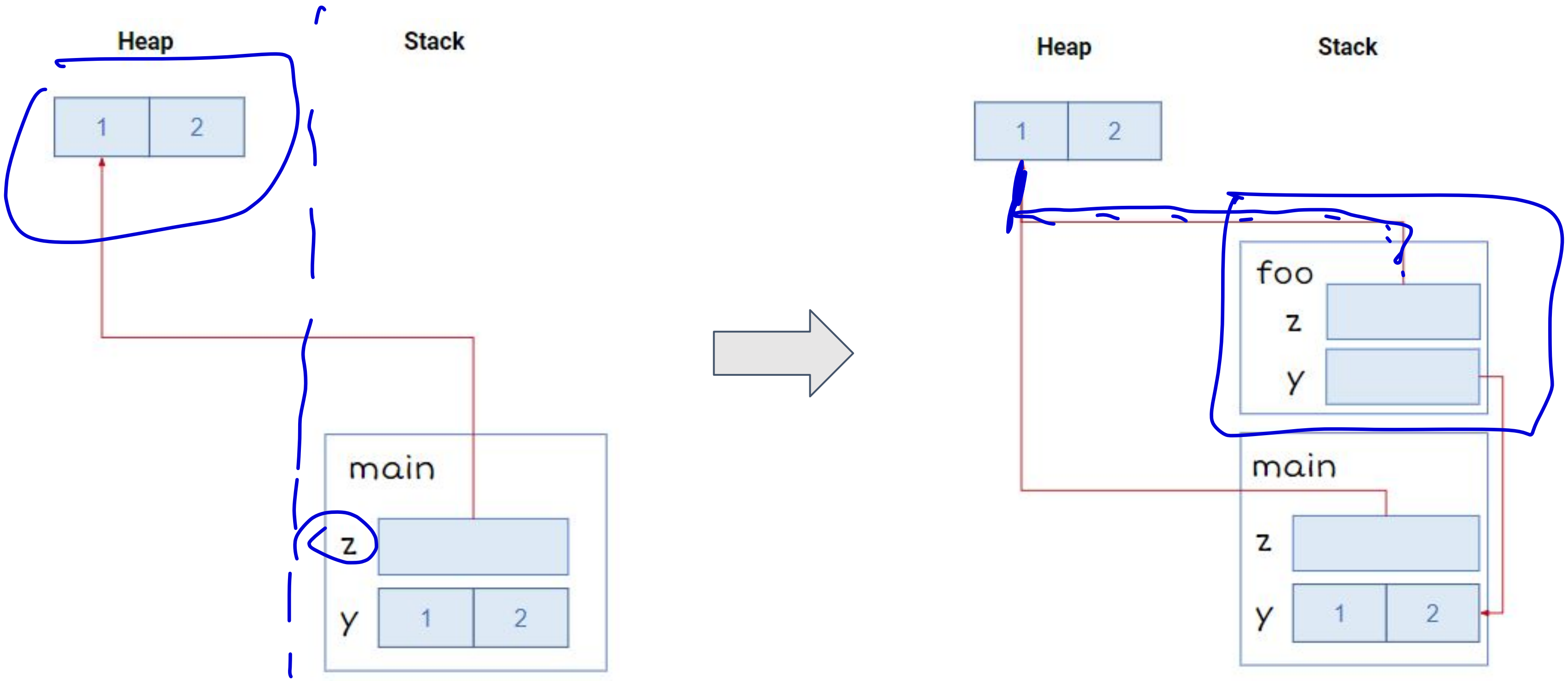
```
void foo(long *y, long *z)
{
  y[0] = -7;
  y[1] = -8;
  z[0] = 4;
  z[1] = 5;
}


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

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

  foo(y, z);
}
```

stack.

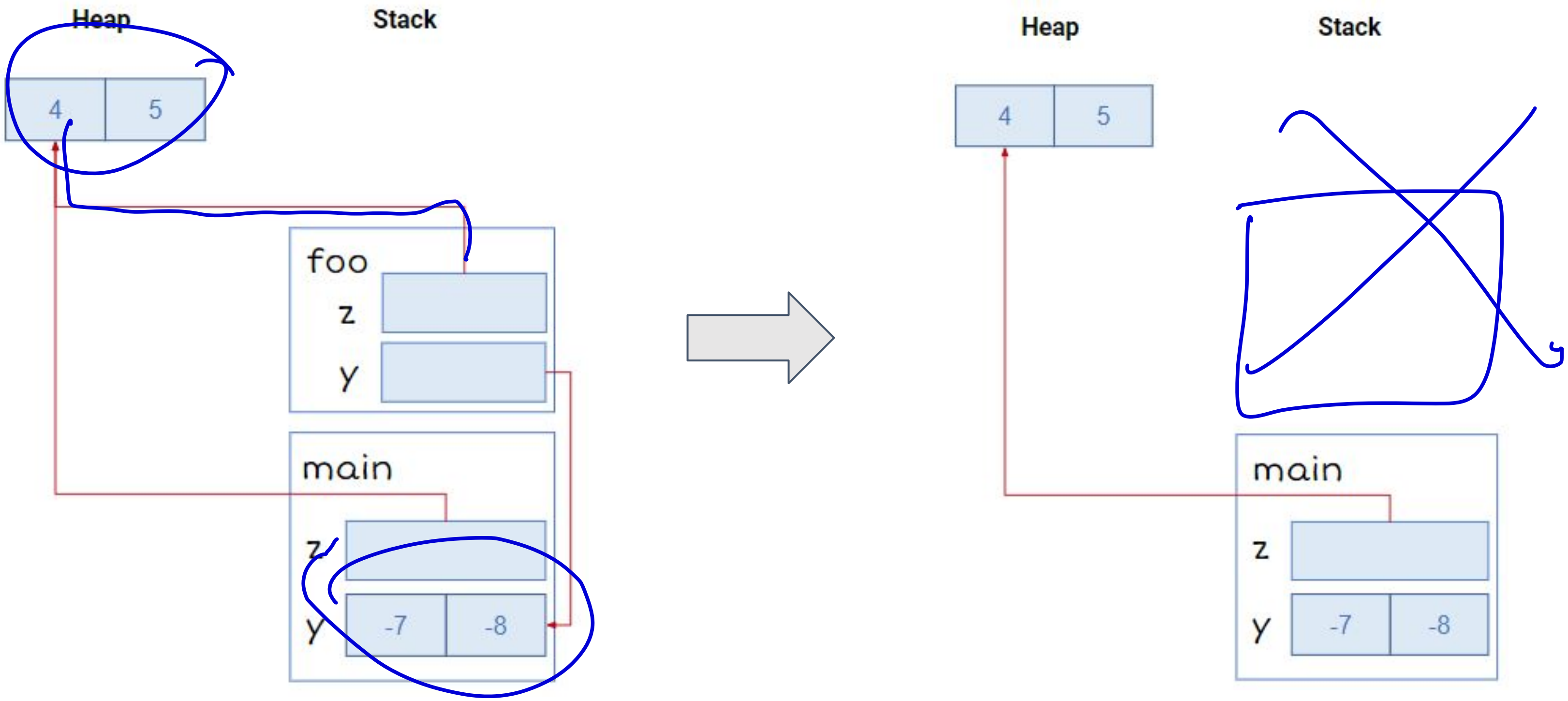
stack

to heap.

Problem 18.1 Answer (Part 1)



Problem 18.1 Answer (Part 2)

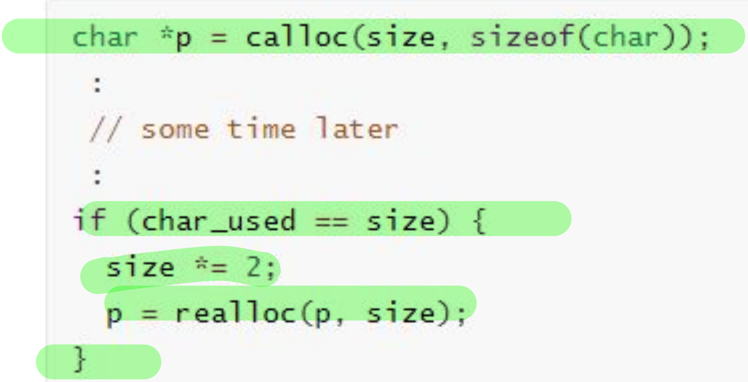


Problem 18.2 Question & Answer

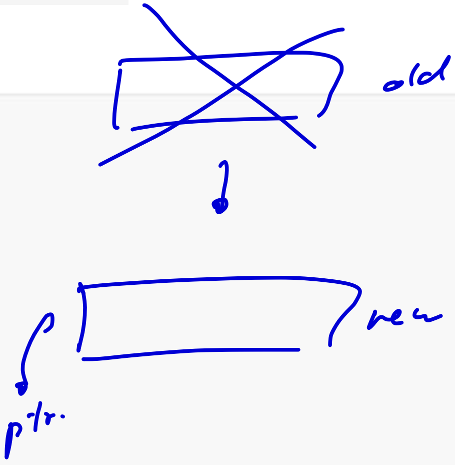
Read the ~~main~~ man page for the function `realloc` and explain what does it do. Can you come up with a situation where it could be useful?

Answer:

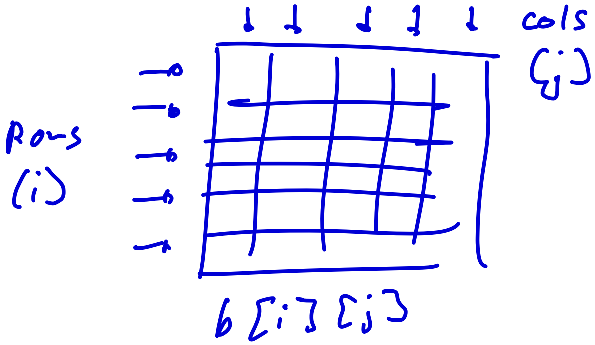
- `realloc` is used when we need to change the size of the memory previously allocated with `malloc` or `calloc`
- `realloc` takes the new size in byte instead of the number of elements
- `realloc` takes care of freeing the previous memory space before allocating a new one, with new size for you
- Below example shows how `realloc` is used to double the size of an array



```
char *p = calloc(size, sizeof(char));
 :
 // some time later
 :
if (char_used == size) {
  size *= 2;
  p = realloc(p, size);
}
```



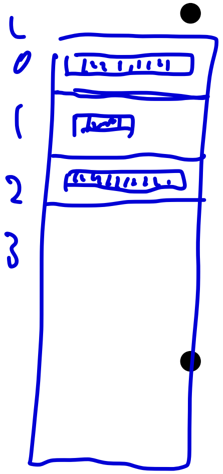
Multidimensional Array



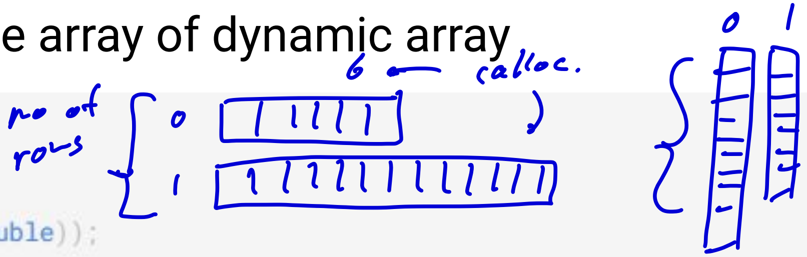
Bucket[0][2]

- Declaration of fixed length multidimensional array

```
long b[10][2];
//Declares an fixed length 2D array named b with 10 rows and 2 columns
```



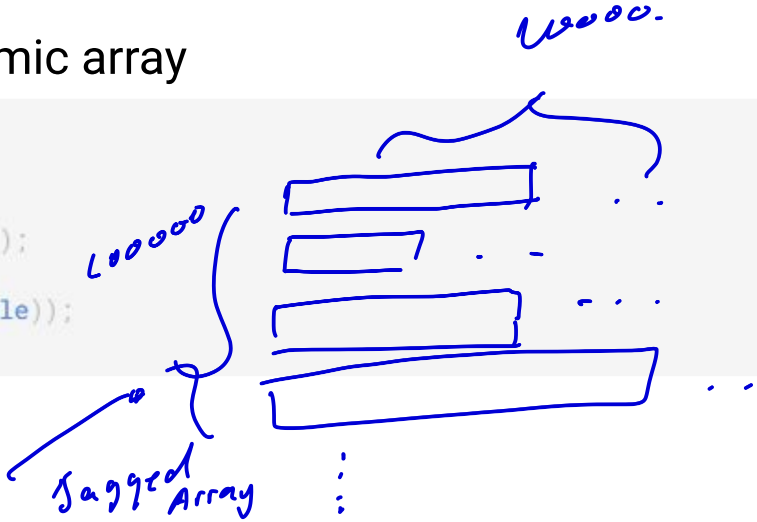
- Using calloc to create a fixed size array of dynamic array



```
double *buckets[10];
long num_of_cols = cs1010_read_long();
for (long i = 0; i < 10; i += 1) {
  buckets[i] = calloc(num_of_cols, sizeof(double));
}
```

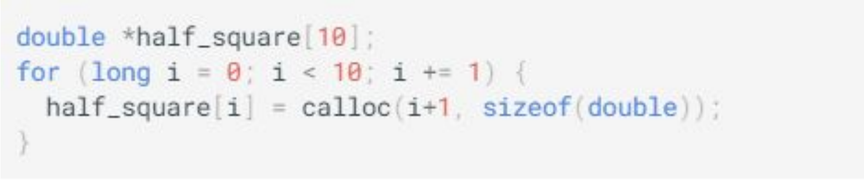
- Using calloc to create a 2D dynamic array

```
double **canvas;
long num_of_rows = cs1010_read_long();
long num_of_cols = cs1010_read_long();
canvas = calloc(num_of_rows, sizeof(double *));
for (long i = 0; i < num_of_rows; i += 1) {
  canvas[i] = calloc(num_of_cols, sizeof(double));
}
```

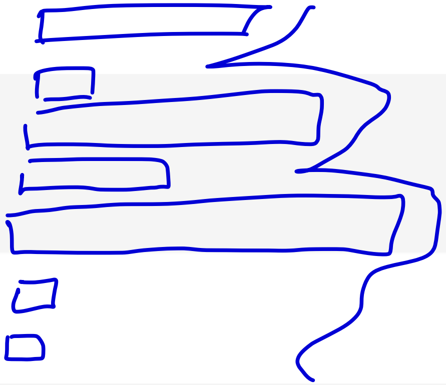


Multidimensional Array

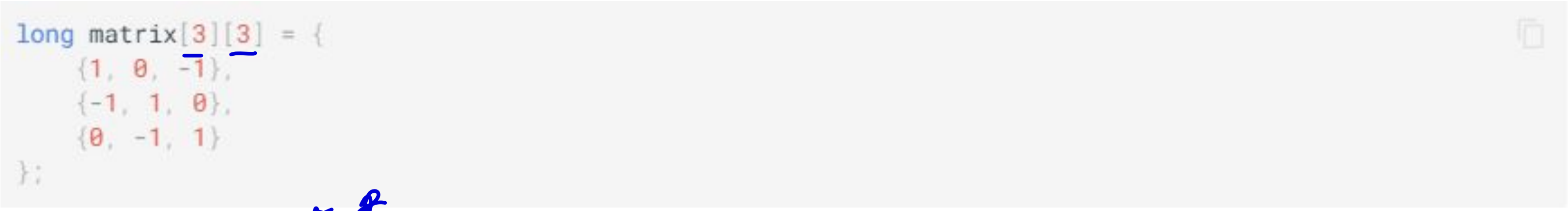
- Jagged Array (Special type of array)



```
double *half_square[10];
for (long i = 0; i < 10; i += 1) {
  half_square[i] = calloc(i+1, sizeof(double));
}
```

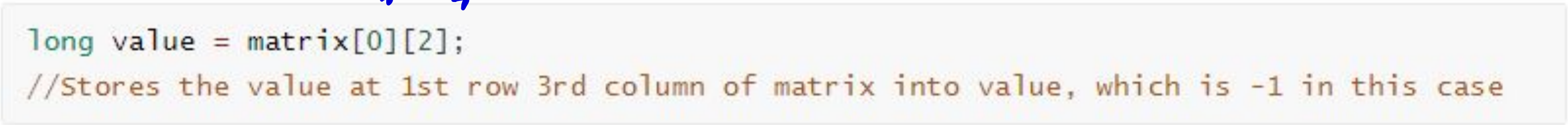


- Initialising 2D array



```
long matrix[3][3] = {
    {1, 0, -1},
    {-1, 1, 0},
    {0, -1, 1}
};
```

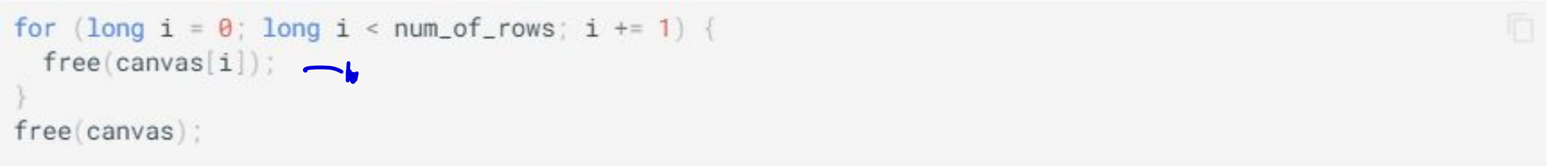
- Accessing 2D array i (row) j (col)



```
long value = matrix[0][2];
//Stores the value at 1st row 3rd column of matrix into value, which is -1 in this case
```

Multidimensional Array

- Freeing memory from both dimensions



```
for (long i = 0; long i < num_of_rows; i += 1) {
  free(canvas[i]);
}
free(canvas);
```

Problem 19.1 Question

Write two functions described below. Show how you would declare the parameters to each function and how you would call each function.

→ every index add to each other.

a) Write a function `add` that performs 3x3 matrix addition. The function should operate on 3x3 matrices of `long`, takes in three parameters, the first two are the operands for addition and the third is the result.

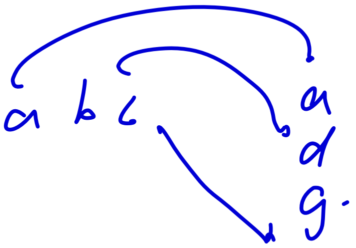
b) Write a function `multiply` that performs 3x3 matrix multiplication. The function should operate on 3x3 matrices of `long`, takes in three parameters, the first two are the operands for multiplication and the third is the result.

a	b	c

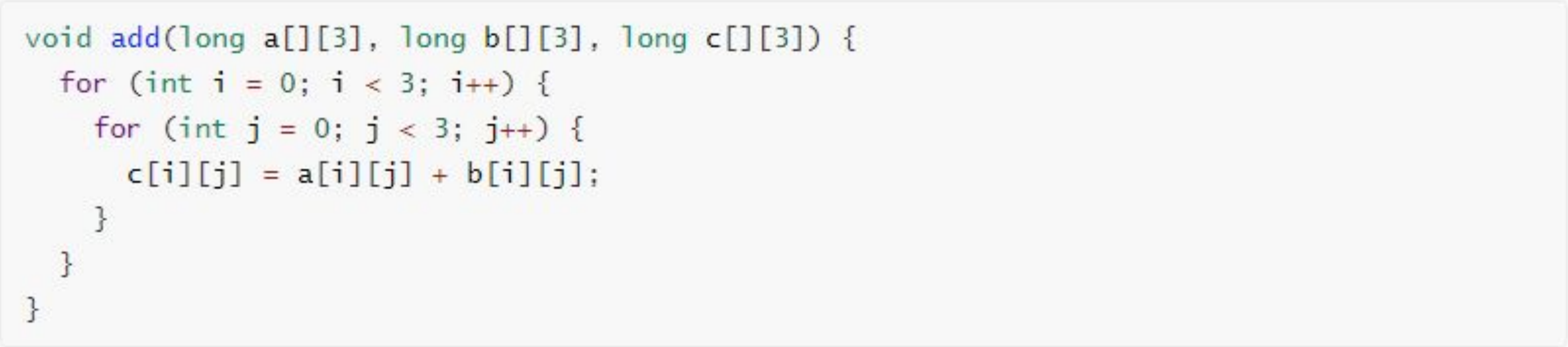
a	b	c
d	e	f
g	h	I

=

$\square = a \times a + b \times d + g \times c$

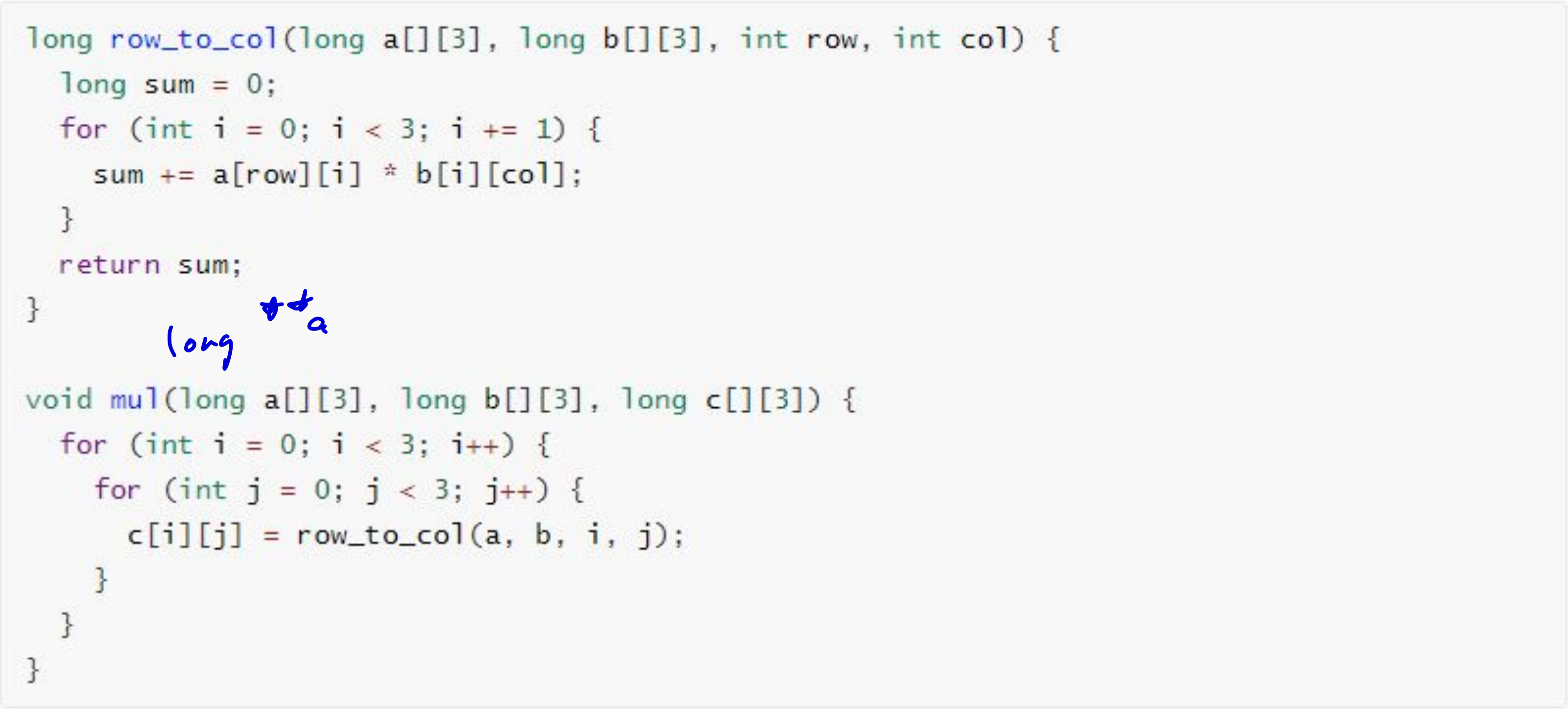


Problem 19.1(a) Answer



```
void add(long a[][3], long b[][3], long c[][3]) {
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      c[i][j] = a[i][j] + b[i][j];
    }
  }
}
```

Problem 19.1(b) Answer



```
long row_to_col(long a[][3], long b[][3], int row, int col) {
  long sum = 0;
  for (int i = 0; i < 3; i += 1) {
    sum += a[row][i] * b[i][col];
  }
  return sum;
}
```

long **a

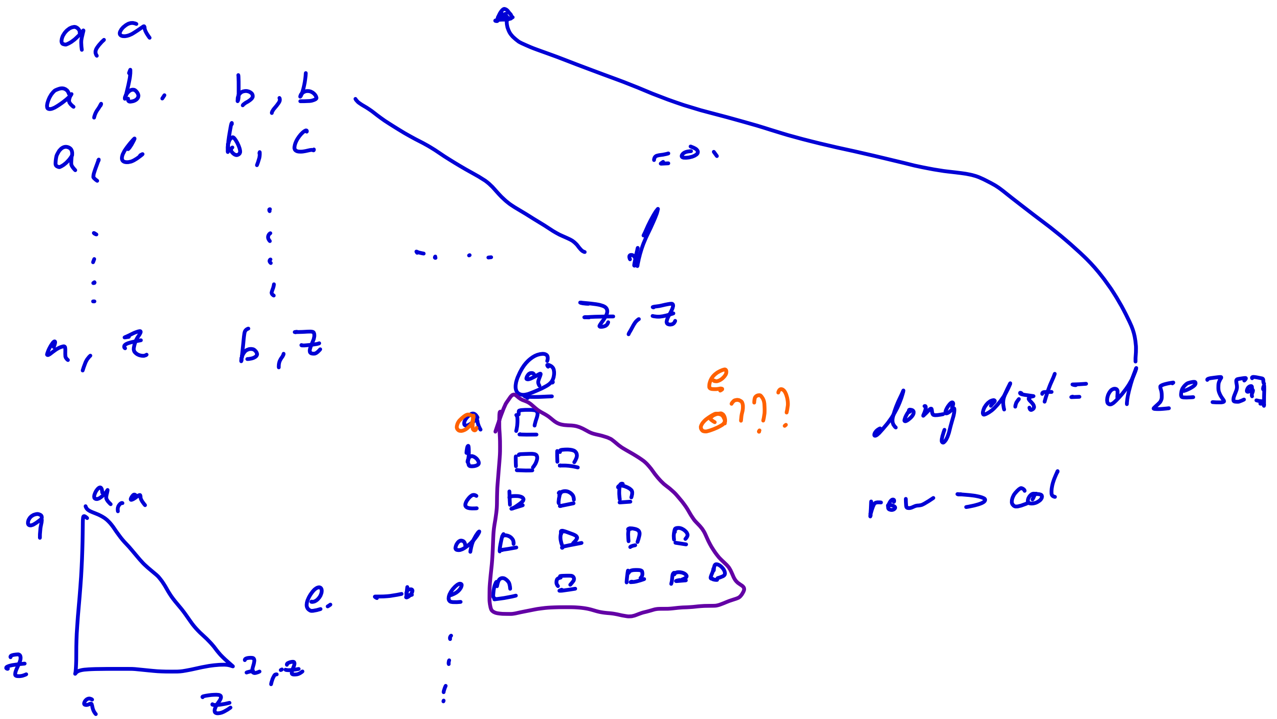
```
void mul(long a[][3], long b[][3], long c[][3]) {
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      c[i][j] = row_to_col(a, b, i, j);
    }
  }
}
```

Problem 19.2 Question

We need to represent the distance in km between every major city in the world. Let's label every city with a number, ranging from 0 .. $n-1$, where n is the number of cities. The distance between city i and j is the same as the distance between city j and i. The distance can be represented with `long`.

Explain how you would represent this information using a jagged two-dimensional array in C efficiently. We have information about a few thousand cities to store.

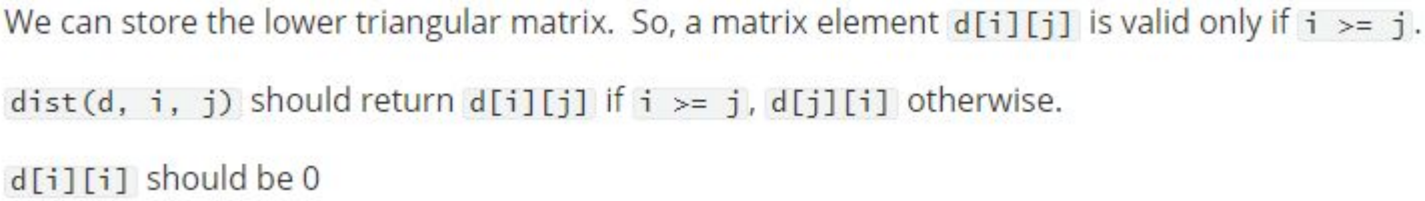
Explain how you would write a function `long dist(long **d, long i, long j)` to retrieve the distance between any two cities i and j.



long dist = d [e][a]

row > col

Problem 19.2 Answer



We can store the lower triangular matrix. So, a matrix element `d[i][j]` is valid only if `i >= j`.

`dist(d, i, j)` should return `d[i][j]` if `i >= j`, `d[j][i]` otherwise.

`d[i][i]` should be 0