**Question 4**

**My Range**

* Submit directory: ex00
* Submit file: ["my\_range.c"]

Create a function my\_range which returns a malloc'd array of integers. This integer array should contain all values between min and max.

Min included - max excluded.

If min value is greater or equal to max's value, a null pointer should be returned.

Don't worry about "free", it will be done in our main().

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- my\_range

\*\*

\*\* @param {int} param\_1

\*\* @param {int} param\_2

\*\*

\*\* @return {int\*}

\*\*

\*/

int\* my\_range(int param\_1, int param\_2)

{

}

**Example 00**

Input: 1 && 4

Output:

Return Value: [1, 2, 3]

**Example 01**

Input: 7 && 10

Output:

Return Value: [7, 8, 9]

**Example 02**

Input: 10 && 11

Output:

Return Value: [10]

**My Strdup**

* Submit directory: ex01
* Submit file: ["my\_strdup.c"]

Let's allocate a string (or array of characters).

We have this: "abc" and we want a copy in a new part of memory that you will have to malloc.

(Reproduce the behavior of strdup from man strdup)

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- my\_strdup

\*\*

\*\* @param {char\*} param\_1

\*\*

\*\* @return {char\*}

\*\*

\*/

char\* my\_strdup(char\* param\_1)

{

}

**Example 00**

Input: "abc"

Output:

Return Value: "abc"

**Example 01**

Input: "RaInB0"

Output:

Return Value: "RaInB0"

**Example 02**

Input: ""

Output:

Return Value: ""

*Tip* (In C) Don't worry about free, it will be done in our main().

**My Print Words Array**

* Submit directory: ex02
* Submit file: ["my\_print\_words\_array.c"]

["Hello", "World", "."]

Create a function that displays the content of an array of strings. One word per line.

Each word will be followed by a newline, including the last one.

(You can't use printf, time to reuse your my\_putstr function ;-))

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- my\_print\_words\_array

\*\*

\*\* @param {string\_array\*} param\_1

\*\*

\*\* @return {void}

\*\*

\*/

#ifndef STRUCT\_STRING\_ARRAY

#define STRUCT\_STRING\_ARRAY

typedef struct s\_string\_array

{

int size;

char\*\* array;

} string\_array;

#endif

void my\_print\_words\_array(string\_array\* param\_1)

{

}

**Example 00**

Input: ["abc", "def", "gh"]

Output: abc

def

gh

Return Value: nil

**Example 01**

Input: ["123"]

Output: 123

Return Value: nil

**Example 02**

Input: ["", "", "hello"]

Output:

hello

Return Value: nil

**My Count On It**

* Submit directory: ex03
* Submit file: ["my\_count\_on\_it.c"]

Count the size of each elements in an array.

Create a function my\_count\_on\_it, which receives a string array as parameter and returns an array with the length of each strings.

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- my\_count\_on\_it

\*\*

\*\* @param {string\_array\*} param\_1

\*\*

\*\* @return {integer\_array\*}

\*\*

\*/

#ifndef STRUCT\_STRING\_ARRAY

#define STRUCT\_STRING\_ARRAY

typedef struct s\_string\_array

{

int size;

char\*\* array;

} string\_array;

#endif

#ifndef STRUCT\_INTEGER\_ARRAY

#define STRUCT\_INTEGER\_ARRAY

typedef struct s\_integer\_array

{

int size;

int\* array;

} integer\_array;

#endif

integer\_array\* my\_count\_on\_it(string\_array\* param\_1)

{

}

**Example 00**

Input: ["This", "is", "the", "way"]

Output:

Return Value: [4, 2, 3, 3]

**Example 01**

Input: ["aBc", "AbcE Fgef1"]

Output:

Return Value: [3, 10]

**Example 02**

Input: ["aBc"]

Output:

Return Value: [3]

*Tips* Google while YOURCODINGLANGUAGE Google for YOURCODINGLANGUAGE Google array.length YOURCODINGLANGUAGE (In C) Yes, you have to allocate for the struct AND then to allocate for the array inside it. :-)

(In C) Don't forget to set the size. ;-)

**My Join**

* Submit directory: ex04
* Submit file: ["my\_join.c"]

Create a function that join an array of strings on a separator characters.

["Hello", "World", "!"] && ' '

=> "Hello World !"

You will receive two parameters, an array with all the string and a separator.

The function returns a string where all the string from array are joined with the separator.

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- my\_join

\*\*

\*\* @param {string\_array\*} param\_1

\*\* @param {char\*} param\_2

\*\*

\*\* @return {char\*}

\*\*

\*/

#ifndef STRUCT\_STRING\_ARRAY

#define STRUCT\_STRING\_ARRAY

typedef struct s\_string\_array

{

int size;

char\*\* array;

} string\_array;

#endif

char\* my\_join(string\_array\* param\_1, char\* param\_2)

{

}

**Example 00**

Input: ["abc", "def", "gh", "!"] && "-"

Output:

Return Value: "abc-def-gh-!"

**Example 01**

Input: ["abc", "def", "gh", "!"] && "blah"

Output:

Return Value: "abcblahdefblahghblah!"

**Example 02**

Input: ["abc", "def", "gh", "!"] && ""

Output:

Return Value: "abcdefgh!"

**Example 03**

Input: [] && " "

Output:

Return Value: nil

**My Robot Simulator**

* Submit directory: ex05
* Submit file: ["my\_robot\_simulator.c"]

Write a robot simulator.

++ Instructions:

The robot factory manufactures robots that have three possible movements:

* turn right
* turn left
* advance

Robots are placed on a hypothetical infinite grid, facing a particular direction (north, east, south, or west) at a set of {x,y} coordinates, e.g., {3,8}.

Robots can pivot left and right.

The factory's test facility needs a program to verify robot movements.

The floor of the room is a grid, each square of which measures 1 square RU (Robot Unit).

The rooms are always oriented so that each wall faces east, south, west, and north.

The test algorithm is to place a robot at a coordinate in the room, facing in a particular direction.

The robot then receives a number of instructions, at which point the testing facility verifies the robot's new position, and in which direction it is pointing.

The robot factory's test facility has a simulator which can take a string of letters and feed this into a robot as instructions.

* The letter-string "RAALAL" means:
  + Turn right
  + Advance twice
  + Turn left
  + Advance once
  + Turn left yet again
* Say a robot starts at {7, 3} facing north. Then running this stream of instructions should leave it at {9, 2} facing west.

++ Your job

Write a function that takes a string of instructions as parameters and return string with the following format:

"{x: X, y: Y, bearing: DIRECTION}"

X and Y with last coordinates of the robot. DIRECTION will be the orientation (north, west, south, east)

Robot always starts in 0, 0 facing north.

Going south will increase Y. Going west will decrease X.

The grid is unlimited, positions can be negative.

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- my\_robot\_simulator

\*\*

\*\* @param {char\*} param\_1

\*\*

\*\* @return {char\*}

\*\*

\*/

char\* my\_robot\_simulator(char\* param\_1)

{

}

**Example 00**

Input: "RAALALL"

Output:

Return Value: "{x: 2, y: -1, bearing: 'south'}"

**Example 01**

Input: "AAAA"

Output:

Return Value: "{x: 0, y: -4, bearing: 'north'}"

**Example 02**

Input: "RAARA"

Output:

Return Value: "{x: 2, y: 1, bearing: 'south'}"

**Example 03**

Input: ""

Output:

Return Value: "{x: 0, y: 0, bearing: 'north'}"