**Quest07**

Remember to git add && git commit && git push each exercise!

We will execute your function with our test(s), please DO NOT PROVIDE ANY TEST(S) in your file

For each exercise, you will have to create a folder and in this folder, you will have additional files that contain your work. Folder names are provided at the beginning of each exercise under submit directory and specific file names for each exercise are also provided at the beginning of each exercise under submit file(s).

**Introduction**

Macro! There are [4 steps of compilation](https://www.calleerlandsson.com/the-four-stages-of-compiling-a-c-program/#:~:text=Compiling%20a%20C%20program%20is,compilation%2C%20assembly%2C%20and%20linking) and macro are run at the preprocessing one.

Very useful to define values.

Objective: after experiencing with define, you will do something hard is to build algorithm with one loop and one nested loop. A great way to break down the complexity of them is to split it into smaller functions. (my\_union and inter can be done with my\_string\_index() for example :-)

Enjoy!

**My Define**

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

We lost a part of the following code, can you make it work! :-)

#include <unistd.h>

#XXXX EVEN(N) XXXXXX

#XXXX SUCCESS XXXXXX

#XXXX EVEN\_MSG "I have an even number of arguments."

#XXXX ODD\_MSG XXXXXXXXXXXXXX

typedef enum s\_bool

{

XXXX,

XXXX

} t\_bool;

void my\_putchar(char c)

{

write(1, &c, 1);

}

void my\_putstr(char \*str)

{

int index;

index = 0;

while (str[index] != '\0') {

my\_putchar(str[index]);

index++;

}

}

t\_bool my\_is\_even(int nbr)

{

return ((EVEN(nbr)) ? TRUE : FALSE);

}

void my\_define(int argc)

{

if (my\_is\_even(argc) == TRUE) {

my\_putstr(EVEN\_MSG);

my\_putchar('\n');

}

else {

my\_putstr(ODD\_MSG);

my\_putchar('\n');

}

}

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- my\_define

\*\*

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

\*\*

\*\* @return {void}

\*\*

\*/

void my\_define(int param\_1)

{

}

**Example 00**

Input: 1

Output: I have an odd number of arguments.

Return Value: nil

**Example 01**

Input: 2

Output: I have an even number of arguments.

Return Value: nil

**Example 02**

Input: 3

Output: I have an odd number of arguments.

Return Value: nil

*Tip* (In C) Google the following: define in C

**My Union**

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

Write a function my\_union that takes two strings and returns, without doubles, the characters that appear in either one of the strings.

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- my\_union

\*\*

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

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

\*\*

\*\* @return {char\*}

\*\*

\*/

char\* my\_union(char\* param\_1, char\* param\_2)

{

}

**Example 00**

Input: "zpadinton" && "paqefwtdjetyiytjneytjoeyjnejeyj"

Output:

Return Value: "zpadintoqefwjy"

**Example 01**

Input: "ddf6vewg64f" && "gtwthgdwthdwfteewhrtag6h4ffdhsd"

Output:

Return Value: "df6vewg4thras"

**Example 02**

Input: "rien" && "cette phrase ne cache rien"

Output:

Return Value: "rienct phas"

**Inter**

* Submit directory: ex02
* Submit file: ["inter.c"]

Write a function that takes two strings and return, without doubles, the characters that appear in both strings, in the order they appear in the first one.

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- inter

\*\*

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

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

\*\*

\*\* @return {char\*}

\*\*

\*/

char\* inter(char\* param\_1, char\* param\_2)

{

}

**Example 00**

Input: "padinton" && "paqefwtdjetyiytjneytjoeyjnejeyj"

Output:

Return Value: "padinto"

**Example 01**

Input: "ddf6vewg64f" && "gtwthgdwthdwfteewhrtag6h4ffdhsd"

Output:

Return Value: "df6ewg4"

**Example 02**

Input: "nothing" && "This sentence hides nothing"

Output:

Return Value: "nothig"

**Rcapitalize**

* Submit directory: ex03
* Submit file: ["rcapitalize.c"]

Write a function that takes one string and, capitalize the last character of each word in uppercase and the rest in lowercase.

A word is a section of string delimited by spaces/tabs or the start/end of the string. If a word has a single letter, it must be capitalized.

A letter is a character in the set [a-zA-Z]

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- rcapitalize

\*\*

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

\*\*

\*\* @return {char\*}

\*\*

\*/

char\* rcapitalize(char\* param\_1)

{

}

**Example 00**

Input: "a FiRSt LiTTlE TESt"

Output:

Return Value: "A firsT littlE tesT"

**Example 01**

Input: ""

Output:

Return Value: ""

**Example 02**

Input: "SecONd teST A LITtle BiT Moar comPLEX"

Output:

Return Value: "seconD tesT A littlE biT moaR compleX"

**Example 03**

Input: " But... This iS not THAT COMPLEX"

Output:

Return Value: " but... thiS iS noT thaT compleX"

**Is Anagram**

* Submit directory: ex04
* Submit file: ["is\_anagram.c"]

An anagram is a sequence of characters formed by rearranging the letters of another sequence, such as 'cinema', formed from 'iceman'.

Given two strings as parameters, create a function able to tell whether or not the first string is an anagram of the second.

**Considerations**:

* Be careful: the naive solution won't work on our big input, you have to find an optimized solution which will run in O(sa + sb) time (where sa is the length of a and sb length of b).
* Our tested string will always be a sequence of ascii characters between 32 and 126 inclusive.
* The bigger test we will do is on 2 sequences of 1.000.000 characteres each. It should run in less than 2 seconds.

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- is\_anagram

\*\*

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

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

\*\*

\*\* @return {int}

\*\*

\*/

int is\_anagram(char\* param\_1, char\* param\_2)

{

}

**Example 00**

Input: "abcdef" && "fabcde"

Output:

Return Value: 1

**Example 01**

Input: "ad" && "bc"

Output:

Return Value: 0

**Example 02**

Input: "" && ""

Output:

Return Value: 1