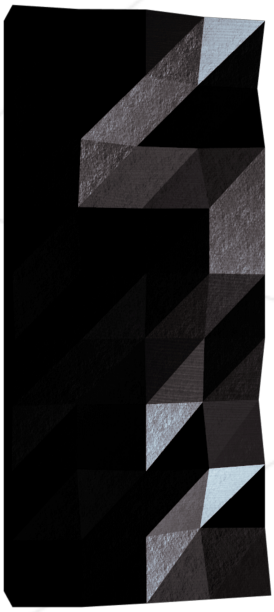

Exam

Bocal bocal@42.fr

Abstract: This document is the subject of your exam

Contents

Chapter I

Administrative details

I.1 General guidelines

- `No` communication `whatsoever` is allowed.
- This is an exam, you don't have a right to chat, listen to music, make noise, or generally do anything that may disturb the other students in any way.
- Your phones and other technological devices must be turned off `and` put away. If a phone rings, the whole row will be disqualified from the exam and kicked out immediately.
- Your home directory contains two folders: `"rendu"` and `"sujet"`.
- The `"sujet"` folder contains the subject of the exam. You have visibly found it.
- The `"rendu"` folder is a clone of your Git turn-in repository. You will work in it, and use it as any regular Git repository.
- We will only grade what you pushed on your Git repository. The system wil stop accepting pushes at the exact end time of the exam, so do NOT wait until the last minute to push your work.
- You can only run your programs in the `"rendu"` directory or one of its subdirectories.
- Each exercise must be carried out in the appropriate directory, specified in each exercise's header.
- You must turn in, at the root of your repository, a file named `"auteur"` that contains your login followed by a newline. If this file is absent or wrong in any way you will NOT be graded.

  Example:

```
$> cat -e ~/rendu/auteur
xlogin$
```

- You may need to read the man to carry out some exercises...
- You will be graded by a program. You must respect the specified file/path/function names to the letter.
- Exercises will always specify which files will be collected :
  - When an exercise asks for specific files, they will be explicitly named. For example `"file1.c file1.h"`.
  - Otherwise, when filenames and/or the number of files is up to you, the exercise will say something along the lines of `"*.c *.h"`.
  - When a Makefile is required, it will ALWAYS be explicitly stated.
- In case of technical problem, question about the subject, or any other problem, you must get up silently and wait for a member of the staff to come to you. It is forbidden to ask your neighbors, or to verbally call for a staff member.
- Any equipment not explicitly allowed is implicitly forbidden.
- The correction may not stop at the first wrong exercise. See the `Exercise types` section.
- Any exit is definitive, you can not come in again.
- Staff members may kick you out of the exam without warning if they deem it necessary.
- You are allowed blank pieces of paper, and a pen. No notebooks, notes, or any help of the sort. You are alone to face this exam.
- For any question after the exam, please open a ticket on dashboard.42.fr

I.2 Code

- Useful functions and files will sometimes be given to you in the `misc` subdirectory of the subject.
- The correction is fully automated, and performed by the program we know as the `Moulinette`.
- When an exercise asks you to write a program with one or more explicitly named files, it will be compiled with the following command: `gcc -Wall -Wextra -Werror file1.c file2.c file3.c -o program_name`.

- When the exercise leaves the filenames up to you, it will be compiled with: `gcc -Wall -Wextra -Werror *.c -o program_name`.

- Finally, when you must only turn in a function (so, one file), it will be compiled with `gcc -c -Wall -Wextra -Werror yourfile.c`, then we will compile our `main` function and link them together to create a test program.

- Allowed functions will be specified in the headers of the exercises. You may recode any other function you think is necessary. Using a function that's not explicitly allowed is considered cheating, and will result in a `-42` grade, with no possible discussion or appeal whatsoever. You've been warned.

- Any function that isn't explicitly allowed is implicitly forbidden.

I.3 Exercise types

There are three possible exercise types, and they are not graded the same way. Here they are:

- `Mandatory exercise` - An exercise that stops the correction if it's not done correctly. You MUST do it and do it right if you want to be graded on the exercises that come after it.

- `Retro-validated exercise` - If you do not turn in this exercise, the correction does NOT stop, and you can still gain points from it if you correctly do a non-optional exercise that comes AFTER. However, if you turn in ANY file, and you do not validate the exercise, the correction stops immediately. So, you can completely avoid this exercise and try to gain points with one that comes after it, or you may attempt it and risk losing the points that come after.

- `Optional exercise` - And exercise that never stops the correction, but can not retro-validate anything.

Chapter II

Exercises

II.1 Exercise 00 - ulstr

	Exercice : 00
ulstr	
Turn-in directory : *ex*00/	
Files to turn in : `ulstr.c`	
Allowed functions : `write`	
Remarks : `Mandatory exercise`	

Write a program that takes a string and reverses the case of all its letters. Other characters remain unchanged.

You must display the result following by a `\n`.

If the number of arguments is not 1, the program displays `\n`.

Examples :

```
$>./ulstr "L'eSPrit nE peUt plUs pRogResSer s'Il staGne et sI peRsIsTent VAnIte et auto-justification." |
    cat -e
l'EspRIT Ne PEuT PLuS PrOGrESsER S'iL STAgNE ET Si PErSiStENT vaNiTE ET AUTO-JUSTIFICATION.$
$>./ulstr "S'enTOuRer dE sECreT eSt uN sIGnE De mAnQuE De coNNaiSSanCe.  " | cat -e
s'ENtoUrER De SecREt EsT Un SigNe dE MaNqUe dE COnnAIssANcE.  $
$>./ulstr "3:21 Ba  tOut  moUn ki Ka di KE m'en Ka fe fot" | cat -e
3:21 bA  ToUT  MOuN KI kA DI ke M'EN kA FE FOT$
$>./ulstr | cat -e
$
```

II.2 Exercise 01 - wdmatch

	Exercice : 01
wdmatch	
Turn-in directory : $ex01/$	
Files to turn in : `wdmatch.c`	
Allowed functions : `write`	
Remarks : `Mandatory exercise`	

Write a program that takes two strings and checks whether it's possible to write the first string with characters from the second string, while respecting the order in which these characters appear in the second string.

If it's possible, the program displays the string, followed by a `\n`, otherwise it simply displays a `\n`.

If the number of arguments is not 2, the program displays a `\n`.

Examples:

```
$>./wdmatch "faya" "fgvvfdxcacpolhyghbreda" | cat -e
faya$
$>./wdmatch "faya" "fgvvfdxcacpolhyghbred" | cat -e
$
$>./wdmatch "quarante deux" "qfqfsudf arzgsayns tsregfdgs sjytdekuoixq " | cat -e
quarante deux$
$>./wdmatch "error" rrerrrfiiljdfxjyuifrrvcoojh | cat -e
$
$>./wdmatch | cat -e
$
```

II.3 Exercise 02 - ft_strdup

	Exercice : 02
ft_strdup	
Turn-in directory : *ex*02/	
Files to turn in : `ft_strdup.c`	
Allowed functions : `malloc`	
Remarks : `Mandatory exercise`	

Reproduce the behavior of the function `strdup` (man strdup).
Your function must be declared as follows:

```
char    *ft_strdup(char *src);
```

II.4 Exercise 03 - inter

	Exercice : 03
inter	
Turn-in directory : $ex03/$	
Files to turn in : `inter.c`	
Allowed functions : `write`	
Remarks : `Mandatory exercise`	

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

The display will be followed by a `\n`.

If the number of arguments is not 2, the program displays `\n`.

Examples:

```
$>./inter "padinton" "paqefwtdjetyiytjneytjoeyjnejeyj" | cat -e
padinto$
$>./inter ddf6vewg64f gtwthgdwthdwfteewhrtag6h4ffdhsd | cat -e
df6ewg4$
$>./inter "rien" "cette phrase ne cache rien" | cat -e
rien$
$>./inter | cat -e
$
```

II.5 Exercise 04 - str_maxlenoc

	Exercice : 04
	str_maxlenoc
Turn-in directory : $ex04/$	
Files to turn in : `str_maxlenoc.c`	
Allowed functions : `write, malloc, free`	
Remarks : `Mandatory exercise`	

Write a program that takes one or more strings and displays, followed by a newline, the longest string that appears in every parameter. If more that one string qualifies, it will display the one that appears first in the first parameter. Note that the empty string technically appears in any string.

If there are no parameters, the program displays `\n`.

Examples:

```
$>./str_maxlenoc ab bac abacabccabcb
a
$>./str_maxlenoc bonjour salut bonjour bonjour
u
$>./str_maxlenoc xoxAoxo xoxAox  oxAox oxo  A ooxAoxx oxooxo Axo | cat -e
$
$>./str_maxlenoc bosdsdfnjodur atehhellosd afkuonjosurafg headfgllosf fghellosag afdfbosnjourafg
os
$>./str_maxlenoc | cat -e
$
```

II.6 Exercise 05 - count_island

	Exercice : 05
count_island	
Turn-in directory : $ex05/$	
Files to turn in : `*.c, *.h`	
Allowed functions : `open, close, read, write, malloc, free`	
Remarks : `Mandatory exercise`	

Write a program that takes a file that contains lines of equal length. Those lines contain characters that are either `'.'` or `'X'`. All those lines put together form rectangles of `'.'` containing "islands" of `'X'`.

The maximum size of a line is 1024 characters, including the terminating newline.

A column if formed of the set of characters in the file that are separated from the start of their respective lines by the same number of characters.

Two characters are said to be touching if they are contiguous and on the same line, or on contiguous lines and on the same column.

An "island" of `'X'` means a set of `'X'` touching each other.

The program must walk though the file and display it after replacing all the `'X'` by a number corresponding to the position their island appears in the file, starting at the beginning of the file.

There can be only one result.

If the file is empty, or there is an error (Incoherent input, for example), or no parameters are passed, the program must display a newline.

The file contains at most 10 islands.

You will find examples in this subject's `misc` directory.

Example :

```
$>cat toto
.................XXXXXXXX.....................................
...................XXXXXXXXX.......XXXXXXXX.....................
.................XXXXXXXX.............XXX...XXXXX.................
...................XXXXXX.....X...XXXXXXXXXX...................
.............................X.................................
......XXXXXXXXXXXX.............X.................................
..................X.............XXXXXXXXX.........................
..................X.........XXXXXXXXXXX..........................
..................X.....................................................
XX.....................................................XXXX
XX..................XXXXXXXXXXXX.............................X
.....................................................................
.....................................................................X.
....................XXXXX.....................................XX
$>
$>./count_island toto
.................00000000.....................................
...................000000000.......11111111.....................
.................00000000.............111...11111.................
...................000000.....2...11111111111...................
.............................2.................................
......3333333333333.............2.................................
..................3.............222222222.........................
..................3.........222222222222.........................
..................3.....................................................
44.....................................................5555
44..................6666666666666.............................5
.....................................................................
.....................................................................7.
....................88888.....................................77
$>
```

```
$>cat qui_est_la
.....................................................................
...X........X.....XXXXX......XXXXXXX...XXXXXXXXXX..XXXXXXXXXX......
...XX......XX....XX...XX....XX.....XX.....XXXX.....XXXXXXXXXX......
...XXXX..XXXX...XX.....XX...XX.....XX......XX......XX..............
...XX.XXXX.XX...XX.....XX...XX.....XX......XX......XX..............
...XX...X..XX...XX.....XX...XXXXXXXX.......XX......XXXXX...........
...XX......XX...XXXXXXXXX...XXXX...........XX......XXXXX...........
...XX......XX..XX.......XX..XX.XX..........XX......XX..............
...XX......XX..XX.......XX..XX...X.........XX......XX..............
...XX......XX..XX.......XX..XX....X......XXXXXX....XXXXXXXXXX......
...XX......XX.XX.........XX.XX.....XX..XXXXXXXXXX..XXXXXXXXXX..X...
.....................................................................
$>
$>./count_island qui_est_la
.....................................................................
...0........0.....11111......2222222...3333333333..4444444444......
...00......00....11...11....22.....22.....3333.....4444444444......
...0000..0000...11.....11...22.....22......33......44..............
...00.0000.00...11.....11...22.....22......33......44..............
...00...0..00...11.....11...22222222.......33......44444...........
...00......00...111111111...2222...........33......44444...........
...00......00..11.......11..22.22..........33......44..............
...00......00..11.......11..22...5.........33......44..............
...00......00..11.......11..22....6......333333....4444444444......
...00......00.11.........11.22.....77..3333333333..4444444444..8...
.....................................................................
$>
```

```
$>cat -e rien
$>./count_island rien | cat -e
$
$>
```

II.7 Exercise 06 - g_diam

	Exercice : 06
g_diam	
Turn-in directory : $ex06/$	
Files to turn in : `*.c, *.h`	
Allowed functions : `write, malloc, free`	
Remarks : `Mandatory exercise`	

Write a programe that takes a string. This string represents a graph and is composed of series of links between this graph's nodes. Links are separated by a single space. Nodes are represented by numbers, and links by two nodes separated by a `'-'`. For exemple, if there is a link between nodes 2 and 3, it could be written as `"2-3"` or `"3-2"`.

The program will display the number of nodes comprised in the longest chain, followed by a `'\n'`, knowing it's impossible to traverse a node more than once.

If the number of parameters is different from 1, the program displays a `'\n'`.

Examples:

```
$>./g_diam "17-5 5-8 8-2 2-8 2-8 17-21 21-2 5-2 2-6 6-14 6-12 12-19 19-14 14-42" | cat -e
10$
$>./g_diam "1-2 2-3 4-5 5-6 6-7 7-8 9-13 13-10 10-2 10-11 11-12 12-8 16-4 16-11 21-8 21-12 18-10 18-13
    21-18" | cat -e
15$
```

II.8 Exercise 07 - time_lord

	Exercice : 07
time_lord	
Turn-in directory : $ex07/$	
Files to turn in : `secret`	
Allowed functions : `Tout ce que vous voulez`	
Remarks : `Mandatory exercise`	

You'll find an executable called `time_lord` in the `misc/` directory of this exam. When you run it, it displays the number of seconds before it will display the secret. When this number of seconds is elapsed, the program displays the secret (without ANY extraneous character). You must find the secret, and copy it (as-is, with NO extra characters whatsoever) in a file called `secret`. You must turn in this file to validate the exercise.