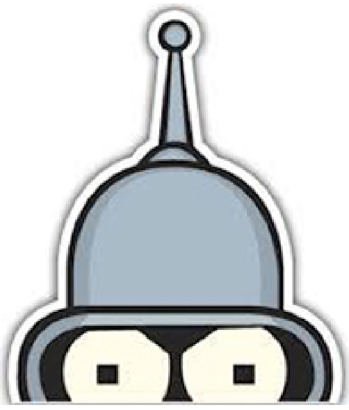

B4- Elementary Programming in C

B-CPE-330

Exam 3

Frozen Shower

v1.0

Instructions

Administrative details

- Any kind of communication is forbidden.
- You must put your student card, face up, on your desk.
- This is an exam, it is forbidden to talk, listen to music, make noise, or produce any kind of nuisance that may disturb other students or prevent this exam from running smoothly.
- Your cell phones must be OFF.
- In case of a technical problem with the subject, you have to ask a supervisor (and ONLY a supervisor).
- Any equipment that is not explicitly allowed is strictly forbidden.
- You have the right to use blank sheets of paper and a pen.
- Any exit is definitive.

Delivery

You will have to use the '∼**/rendu**' directory as the root directory for your delivery.
Each exercise must be carried out in the directory mentioned in the task's header.
The '∼**/rendu**' directory must contain a file called '**author**', containing your login followed by a '\n'.

```
Terminal
$> cat -e ~/rendu/author
firstname.lastname@epitech.eu$
```

EXERCISES MUST BE CARRIED OUT IN ORDER. Grading will stop as soon as an exercise is deemed false.

To make a delivery, at any time, use the **exam_rendu** command and follow the instructions.

The collection will be handled by a program, so you must respect the names, paths, of all the files and directories...

The code

- Useful fonctions are sometimes given in the '~**/sujet/misc**' directory.
- Collection of the code is automated, and a program will check that your exercices work correctly.
- Allowed functions are specified in each task's header. You can re-code any function you like.
- Any function not explicitly allowed is forbidden.

If a task mention files from the ~**/looneytunes** directory, it means that these files will be added by the autograder during the compilation.

If a task ask you to make a program, it's main function must always return 0 (even in case of failure).

Task 1

union

Delivery: ~/rendu/union/*
Language: C
Compilation: make
Binary name: union
Allowed functions: write

Write a program that takes two strings and displays, without doubles, the characters that appears in one or the other followed by a '\n'.

The output will be in order of apparition in the command line.
If there isn't two parameters, display a '\n'.

Terminal

```
$> ./union "zpadinton" "paqefwtdjetyiytjneytjoeyjnejeyj" | cat -e
zpadintoqefwjy$
$> ./union ddf6vewg64f gtwthgdwthdwfteewhrtag6h4ffdhsd | cat -e
df6vewg4thras$
$> ./union "nothing" "this sentence hides nothing" | cat -e
nothigs ecd$
$> ./union | cat -e
$
```

Task 2

ulstr

Delivery: ∼/rendu/ulstr/ulstr.c
Language: C
Compilation: gcc -o ulstr ulstr.c ∼/looneytunes/my_putchar.c ∼/looneytunes/main.c
Binary name: ulstr
Allowed functions: write

Write a function that takes a string, and transforms every lower case letter to upper case, and every upper case to lower case. Other characters remain unchanged.
The function writes the result to the standard output followed by a '\n'.

It must be prototyped the following way:

```c
void ulstr(char *str);
```

Here are some examples:

```c
int     main(int ac, char **av)
{
  ulstr("LooneyTunes");
  ulstr("The GAME is afoot.");
  return (0);
}
```

```
Terminal
$> ./ulstr | cat -e
lOONEYtUNES$
tHE game IS AFOOT.$
```

Task 3

do-op

Delivery: ~/rendu/do-op/*
Language: C
Compilation: make
Binary name: do-op
Allowed functions: write, printf

Write a program called *do-op* that computes an operation. The program must take three arguments:

```
Terminal
$> ./do-op value1 operator value2
```

The character operator should correspond to the appropriate function within an array of function pointers (advised but not mandatory, using **if** can be enough).
If the expression is incorrect, the program must display **0**. If the number of arguments is incorrect, **do-op** must not display anything.
The output will always be followed by a '\n'. Here are some examples:

```
Terminal
$> make clean
$> make
$> ./do-op
$> ./do-op 1 + 1
2
$> ./do-op 42friends - ---20toto12
62
$> ./do-op 1 p 1
0
$> ./do-op 1 +toto 1
2
$> ./do-op 1 + toto3
1
$> ./do-op toto3 + 4
4
$> ./do-op foo plus bar
0
$> ./do-op 25 / 0
Stop : division by zero
$> ./do-op 25 % 0
Stop : modulo by zero
```

Task 4

g-nbr-islands

Delivery: ~/rendu/g-nbr-islands/*
Language: C
Compilation: make
Binary name: g-nbr-islands
Allowed functions: write, printf, malloc, free

If there is no parameter, the program displays '\n'.
The program takes a string that represent a series of links between cells.
Links are separated by spaces.
These cells are represented by numbers and the links by '-'.

Examples:
If the cell 2 is linked to the cell 3, representations will be either "2-3" or "3-2".

This string represents a graph.
The program must display number of islands in this graph followed by a '\n'. An island is a set of cells linked only between them. (See examples).

Terminal

```
$> ./g-nbr-islands | cat -e
$
$> ./g-nbr-islands "2-8 42-7 6-12 21-17 19-14 5-8 12-19 14-6" | cat -e
4$
$> ./g-nbr-islands "2-8 42-7 6-12 21-17 19-14 5-8 12-19 14-6 5-17 7-6" | cat -e
2$
```

Task 5

count_island

Delivery: ~/rendu/count_island/*
Language: C
Compilation: make
Binary name: count_island
Allowed functions: read, write, open, close, malloc, free

Write a program called '**count_island**'.

This program takes a file containing a series of lines of equal length, containing either the dot character (.) or the 'X' character.

These lines form a rectangle of '.' containing islands of 'X's.
A line is a series of '.' and 'X' characters terminated by a '\n'.
The lines are all of the same length. The maximum size of a line is 1024 characters.
A column of characters is formed by all the characters in the file that are separated from the start of their respective lines by the same amount of characters.

Two caracters are considered touching each other if they are :

- contiguous and on the same line
  OR
- on the same column and on contiguous lines

An island of 'X' is formed by the set of characters touching each other.

The program must look through the whole file line by line and display it on the screen with all the 'X's from the islands replaced by their number in order of apparition in the file.

The program must do this computation starting by the start of the file.
There can be no two different results from the same file.
If the file is empty, an error occured or no file is passed, the program simply writes the '\n' character to the standard output.
The file contains at most 10 islands.

You can find example files in the misc directory.

Examples

Content of the 'toto' file:

```
.................XXXXXXXX..........................................
....................XXXXXXXXX.......XXXXXXXX.......................
.................XXXXXXXX..............XXX...XXXXX.................
.....................XXXXXX.....X...XXXXXXXXXXX....................
................................X..................................
......XXXXXXXXXXXXX.............X..................................
..................X.............XXXXXXXXX..........................
..................X.........XXXXXXXXXXXX...........................
..................X................................................
XX.............................................................XXXX
XX..................XXXXXXXXXXXXX.................................X
...................................................................
.................................................................X.
.....................XXXXX.......................................XX
```

```
Terminal
$> ./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
$> ./count_island flop_likes
...................................................................
...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...
...................................................................
$> ./count_island titi | cat -e
..0.......$
...1.2....$
....22.33.$
.4..22....$
.4..2...5.$
$> ./count_island | cat -e
$
```