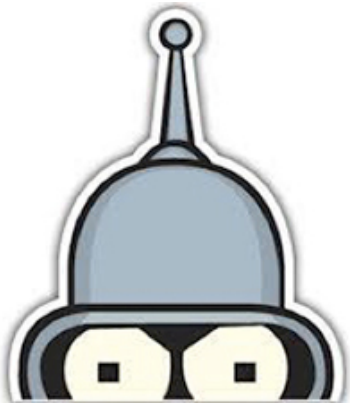

B-1- Elementary Programming in C

B-CPE-330

Exam 2

Solid Lion

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.

Task 1

intersect

Delivery: ~/rendu/task01/*
Language: C
Compilation: gcc intersect.c ~/looneytunes/main.c
Allowed functions: malloc

Write a function that takes two strings and returns the characters in common to both strings, without duplicates.
The output will be in the same order of apparition as in the first parameter.
It must be prototyped the following way:

```
char *intersect(char *s1, char *s2);
```

In case of error, the function must return *NULL*.

Task 2

ulstr

Delivery: ~/rendu/task02/*
Language: C
Compilation: gcc -o ulstr ulstr.c ~/looneytunes/my_putchar.c ~/looneytunes/main.c
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:

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

Here are some examples:

```
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

msquare

Delivery: ~/rendu/task03/*
Language: C
Compilation: make
Binary name: msquare
Allowed functions: write, malloc, free

You have to write a program that computes and display the 3x3 magic square corresponding to the 9 digits passed.
A magic square is a square in which the sum of the numbers in each line is equal to the sum of the numbers in each column, and each diagonal.

Example:
8 1 6
3 5 7
4 9 2
is a magic square where the sums are always equal to 15

You must handle digits from 1 to 9. If a number is equal to '?' you must complete this cell.

If there is a solution, the program displays the 9 numbers separated by a space followed by "OK\n".
Otherwise it displays "KO\n". (See examples)

> Your main function must return 0.
> You must provide a Makefile to build your program.

Examples

Terminal

```
$> ./msquare | cat -e
KO$
$> ./msquare "8 1 6 ?  5 ?  4 9 ?" | cat -e
8 1 6 3 5 7 4 9 2 OK$
$> ./msquare "8 1 6 3 5 7 4 9 2" | cat -e
8 1 6 3 5 7 4 9 2 OK$
$> ./msquare "6 1 8 ?  5 ?  4 9 ?" | cat -e
KO$
$> ./msquare "6 1 8 3 5 7 4 9 2" | cat -e
KO$
$> ./msquare "3 4 5 2 8 7" | cat -e
KO$
$> ./msquare "3 1 5 7 9 8 2 0 4 5 6" | cat -e
KO$
```

Task 4

g-diam

Delivery: ~/rendu/task04/*
Language: C
Compilation: make
Binary name: g-diam
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 the length of its longest path.
It is not possible to go back, that is to say no cell can be used more than once. (See examples).

Your main function must return 0.
You must provide a Makefile to build your program.

Examples

Terminal

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

Task 5

g-nbr-islands

Delivery: ~/rendu/task05/*
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).

Your main function must return 0.
You must provide a Makefile to build your program.

Examples

Terminal

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