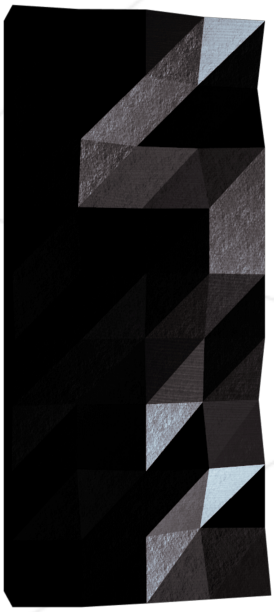

Sujet de l'examen 02

Version 1.0

Bocal bocal@42.fr

Résumé: Ce document est le sujet du troisième examen du module algo-1-001. Cet examen sera pour vous et pour nous l'occasion de faire un point sur votre avancement.

Table des matières

Chapitre I

Détails administratifs

I.1 Consignes générales

- Aucune forme de communication n'est permise.
- Ceci est un examen, il est interdit de discuter, écouter de la musique, faire du bruit ou produire toute autre nuisance pouvant déranger les autres étudiants ou perturber le bon déroulement de l'examen.
- Vos téléphones portables et autres appareils technologiques doivent être éteints et rangés hors d'atteinte. Si un téléphone sonne, toute la rangée concernée est éliminée et doit sortir immédiatement.
- Votre home contient deux dossiers : "rendu" et "sujet".
- Le répertoire "sujet" contient le sujet de l'examen. Vous avez du le trouver puisque vous lisez ce document.
- Le répertoire "rendu" est un clone de votre dépot de rendu dédié à cet examen. Vous y ferez vos commits et vos pushs.
- Seul le contenu que vous avez pushé sur votre dépot de rendu sera corrigé.
- Vous ne pouvez exécuter les programmes que vous avez compilés vous-même que dans votre dossier "rendu" et ses sous-dossiers. Cela est interdit ailleurs.
- Chaque exercice doit être réalisé dans le repertoire correspondant au nom indiqué dans l'en-tête de chaque exercice.
- Vous devez rendre, à la racine du repertoire "rendu", un fichier nommé "auteur" comprenant votre login suivi d'un retour à la ligne. Si ce fichier est absent ou mal formaté, vous ne serez pas corrigé. Par exemple :

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

- Certaines notions nécéssaires à la réalisation de certains exercices sont à découvrir dans les mans.
- C'est un programme qui s'occupe du ramassage, respectez les noms, les chemins,

les fichiers et les répertoires...

- Lorsqu'un exercice impose un nom de fichier, seul ce fichier sera ramassé. Sinon l'exercice stipule `*.c, *.h` et tous vos fichiers `.c` et `.h` seront ramassés, et ces fichiers là seulement.
- En cas de problème technique avec le sujet, on ne doit s'adresser qu'au surveillant uniquement. Interdiction de parler à ses voisins.
- En cas de question, on ne doit s'adresser qu'au surveillant uniquement. Interdiction de parler à ses voisins.
- Tout matériel non explicitement autorisé est implicitement interdit.
- La correction s'arrêtera au premier exercice faux.
- Toute sortie de la salle est définitive.
- Un surveillant peut vous expulser de la salle s'il le juge nécéssaire.

I.2 Le Code

- Des fonctions utiles ou des fichiers supplémentaires sont parfois donnés dans un sous repertoires de ~/sujet/. Si ce dossier n'existe pas ou bien s'il est vide, c'est que nous ne vous fournissons rien. Ce dossier sera généralement nommé `misc`, mais celà peut varier d'un examen à l'autre.
- La correction du code est automatisée. Un programme testera le bon fonctionnement des exercices : la "Moulinette".
- Lorsqu'un exercice vous demande d'écrire un programme avec un ou plusieurs fichiers nommés, votre programme sera compilé avec la commande `gcc -Wall -Wextra -Werror ficher1.c fichier2.c fichiern.c -o nom_programme`.
- Lorsqu'un exercice vous demande d'écrire un programme et laisse les noms et le nombre de fichiers à votre discrétion, votre programme sera compilé avec la commande : `gcc -Wall -Wextra -Werror *.c -o nom_programme`.
- Enfin, lorqu'un exercice vous demande de rendre une fonction (et donc un seul fichier nommé), votre fichier sera compilé avec la commande `gcc -c -Wall -Wextra -Werror votrefichier.c`, puis nous compilerons notre `main` et linkerons l'éxécutable.
- Les fonctions autorisées sont indiquées dans l'en-tête de chaque exercice. Vous pouvez recoder toutes les fonctions qui vous semblent utiles à votre guise. Utiliser une fonction qui n'est pas autorisée est de la triche sanctionnée par un `-42`.
- Toute fonction non autorisée explicitement est implicitement interdite.
- Vous avez le droit à des feuilles blanches et un stylo. Pas de cahier de notes, de pense-bête ou autres cours. Vous êtes seuls face à votre examen.

- Pour toute question après l'examen, créez un ticket sur le dashboard (dashboard.42.fr).

Chapitre II

Exercices

II.1 Exercice 00 - str_capitalizer

	Exercice : 00
str_capitalizer	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `str_capitalizer.c`	
Fonctions Autorisées : `write`	
Remarques : `n/a`	

Ecrire un programme qui prend en paramètre une ou plusieurs chaînes de caractères, et qui, pour chaque argument, met la première lettre de chaque mot en majuscule et le reste en minuscule, et affiche le résultat sur la sortie standard suivi d'un '\n'.

On appelle "mot" une portion de chaîne de caractères délimitée soit par des espaces et/ou des tabulations, soit par le début / fin de la chaîne. Si un mot a une seule lettre, elle devra être mise en majuscule.

S'il n'y a aucun paramêtre, le programme devra afficher '\n'.

```
$> ./str_capitalizer | cat -e
$
$> ./str_capitalizer "Premier PETIT TesT" | cat -e
Premier Petit Test$
$> ./str_capitalizer "DeuxiEmE tEST uN PEU moinS  facile" "   attention C'EST pas dur QUAND mEmE" "ALLer
    UN DeRNier 0123456789pour LA rouTE    E " | cat -e
Deuxieme Test Un Peu Moins  Facile$
   Attention C'est Pas Dur Quand Meme$
Aller Un Dernier 0123456789pour La Route    E $
$>
```

II.2 Exercice 01 - ft_rrange

	Exercice : 01
ft_rrange	
Dossier de rendu : *ex*01/	
Fichiers à rendre : `ft_rrange.c`	
Fonctions Autorisées : `malloc`	
Remarques : `n/a`	

Écrire la fonction suivante :

```
int                 *ft_rrange(int start, int end);
```

Cette fonction doit allouer avec `malloc()` un tableau d'ints, le remplir avec les valeurs (consécutives) démarrant à `end` et finissant à `start` (`start` et `end` inclus !), et renvoyer un pointeur vers la première valeur du tableau.

Exemples :

- Avec (1, 3) vous devrez renvoyer un tableau contenant 3, 2 et 1.
- Avec (-1, 2) vous devrez renvoyer un tableau contenant 2, 1, 0 et -1.
- Avec (0, 0) vous devrez renvoyer un tableau contenant 0.
- Avec (0, -3) vous devrez renvoyer un tableau contenant -3, -2, -1 et 0.

II.3 Exercice 02 - rev_wstr

	Exercice : 02
rev_wstr	
Dossier de rendu : *ex*02/	
Fichiers à rendre : `rev_wstr.c`	
Fonctions Autorisées : `write, malloc, free`	
Remarques : `n/a`	

Écrire un programme qui prend en paramètre une chaîne de caractères, et qui affiche cette chaîne en inversant ses mots.

On appelle "mot" une portion de chaîne de caractères délimitée soit par des espaces et/ou des tabulations, soit par le début / fin de la chaîne.

S'il n'y a aucun paramêtre, le programme devra afficher '\n'.

```
$> ./rev_wstr "le temps du mepris precede celui de l'indifference" | cat -e
l'indifference de celui precede mepris du temps le$
$> ./rev_wstr "abcdefghijklm"
abcdefghijklm
$> ./rev_wstr "il contempla le mont" | cat -e
mont le contempla il$
$> ./rev_wstr | cat -e
$
$>
```

II.4 Exercice 03 - rsort_params

Exercice : 03
rsort_params
Dossier de rendu : *ex*03/
Fichiers à rendre : `rsort_params.c`
Fonctions Autorisées : `write, malloc, free`
Remarques : `n/a`

Écrire un programme qui prend en paramètres une ou plusieurs chaînes de caractères, et qui les affiche sur la sortie standard après les avoir triées dans l'ordre ASCII décroissant, chacune suivie d'un '\n'.

S'il n'y a aucun paramêtre, le programme devra afficher '\n'.

```
$> ./rsort_params | cat -e
$
$> ./rsort_params Rainbowdash " is" The best | cat -e
best$
The$
Rainbowdash$
 is$
$> ./rsort_params this "is SPARTA....." poil | cat -e
this$
poil$
is SPARTA.....$
$>
```

II.5 Exercice 04 - infin_add

	Exercice : 04
infin_add	
Dossier de rendu : *ex*04/	
Fichiers à rendre : `*.c, *.h`	
Fonctions Autorisées : `write, malloc, free`	
Remarques : `n/a`	

Écrire un programme qui prend en paramètres deux chaînes de caractères représentant des nombres de longueur potentiellement infinie, et affiche sur la sortie standard le résultat de l'addition de ces deux nombres, suivi d'un \n.

Un nombre négatif sera précédé d'un et un seul signe -. Les seuls caractères qui feront potentiellement partie de ces chaînes sont les chiffres et le signe -.

Tous les paramètres seront bien formatés, pas de piège.

```
$> ./infin_add "879879087" "67548976597" | cat -e
68428856684$
$> ./infin_add "-876435" "987143265" | cat -e
986266830$
$> ./infin_add "-807965" "-34532"
-842497
$>
```

II.6 Exercice 05 - infin_mult

	Exercice : 05
infin_mult	
Dossier de rendu : *ex*05/	
Fichiers à rendre : `*.c, *.h`	
Fonctions Autorisées : `write, malloc, free`	
Remarques : `n/a`	

Écrire un programme qui prend en paramètres deux chaînes de caractères représentant des nombres de longueur potentiellement infinie, et affiche sur la sortie standard le résultat de la multiplication de ces deux nombres, suivi d'un \n.

Un nombre négatif sera précédé d'un et un seul signe -. Les seuls caractères qui feront potentiellement partie de ces chaînes sont les chiffres et le signe -.

Tous les paramètres seront bien formatés, pas de piège.

```
$> ./infin_mult "879879087" "67548976597" | cat -e
59434931855952726939$
$> ./infin_mult "-876435" "987143265" | cat -e
-865166907460275$
$> ./infin_mult "-807965" "-34532"
27900647380
$>
```

II.7 Exercice 06 - Time Lord

	Exercice : 06
Time Lord	
Dossier de rendu : $ex06/$	
Fichiers à rendre : `secret`	
Fonctions Autorisées : `Tout, absolument tout ce que vous pouvez imaginer`	
Remarques : `n/a`	

Vous trouverez un executable nommé `time_lord` dans le répertoire `misc/` de cet examen. Quand vous éxécutez ce binaire, il affiche le nombre de secondes restantes avant d'afficher la phrase secrète. Quand le nombre de secondes est dépassé, le binaire affiche la phrase secrète. Votre travail consiste à trouver cette phrase secrète par n'importe quel moyen. Vous devez copier la phrase secrète telle quelle sans AUCUN caractère supplémentaire dans un fichier nommé `secret`. Vous devez rendre ce fichier avec la bonne phrase pour valider cet exercice.