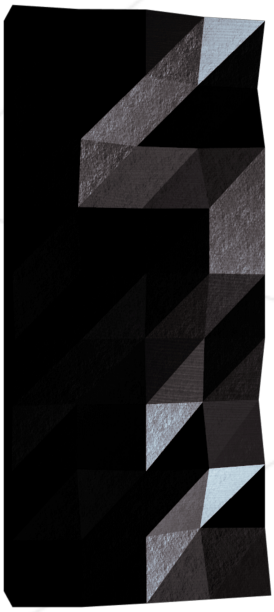

Examen

Bocal bocal@42.fr

Résumé: Ce document est votre sujet d'examen

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, d'écouter de la musique, de faire du bruit, ou de façon plus générale de produire toute 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 répertoire home contient deux dossiers : "rendu" et "sujet".
- Le répertoire "sujet" contient le sujet de l'examen. Vous avez dû le trouver, puisque vous êtes en train de lire 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. Le dépôt cessera d'accepter les pushs à l'heure précise de fin de l'examen, n'attendez donc pas le dernier moment pour pusher.
- Vous ne pouvez exécuter les programmes que vous avez compilés vous-même que dans votre dossier "rendu" et ses sous-dossiers. Cela est interdit (et d'ailleurs impossible) ailleurs.
- Chaque exercice doit être réalisé dans le répertoire 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. Le fichier auteur n'est PAS rétrovalidable.

  Par exemple :

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

- Certaines notions nécessaires à la réalisation de certains exercices sont à découvrir dans les mans.
- C'est un programme qui s'occupe du ramassage, et de la correction. Vous devez donc respecter les noms, les chemins, les fichiers et les répertoires...
- Les exercices stipuleront toujours les fichiers ramassés :
  - Lorsqu'un exercice demande des fichiers particuliers, ils seront nommés explicitement. Par exemple `"fichier1.c fichier1.h"`.
  - Sinon, quand les noms / le nombre de fichiers sont laissés à votre discrétion, l'exercice stipulera quelque chose de la forme `"*.c *.h"`.
  - Lorsqu'un Makefile est requis, cela sera toujours explicitement précisé.
- En cas de problème technique, de question sur le sujet, ou tout autre souci, vous devez vous lever en silence et attendre qu'un surveillant vienne à vous. Interdiction absolue de parler à vos voisins ou d'appeler oralement le surveillant.
- Tout matériel non explicitement autorisé est implicitement interdit.
- La correction ne s'arrête pas forcément au premier exercice faux. Voir la section `Types d'exercices`.
- Toute sortie de la salle est définitive.
- Un surveillant peut vous expulser de la salle sans préavis s'il le juge nécéssaire.
- 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).

I.2 Le Code

- Des fonctions utiles ou des fichiers supplémentaires sont parfois donnés dans un sous-répertoire 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 cela 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. L'utilisation d'une fonction qui n'est pas autorisée est assimilée à de la triche, et sera sanctionnée par un `-42`, sans appel.

- Toute fonction non autorisée explicitement est implicitement interdite.

I.3 Types d'exercices

Il y a plusieurs types d'exercices possibles, et ils ne sont pas tous corrigés de la même façon. Voici des explications :

- `Exercice obligatoire` - Un exercice de ce type arrête immédiatement la correction s'il n'est pas réussi. Comprendre par là que vous devez absolument le réaliser si vous voulez des points pour les exercices d'après.

- `Exercice rétrovalidable` - Si vous ne rendez rien pour cet exercice, la correction ne s'arrête `PAS`, et vous pourrez obtenir les points de cet exercice quand même si vous réussissez un exercice non-bonus plus loin dans l'examen. Cependant, si vous rendez `quoi que ce soit`, et que vous échouez à l'exercice, la correction s'arrête immédiatement. Vous devez donc décider entre tenter l'exercice et risquer de perdre les points de ceux d'après, ou ne pas le tenter, et faire directement un exercice plus difficile.

- `Exercice bonus` - Un exercice de ce type n'arrête jamais la correction s'il est raté. Il ne permet pas, par contre, d'obtenir les points pour les exercices d'avant.

Chapitre II

Exercices

II.1 Exercice 00 - rotone

	Exercice : 00
rotone	
Dossier de rendu : *ex*00/	
Fichiers à rendre : `rotone.c`	
Fonctions Autorisées : `write`	
Remarques : `Exercice rétrovalidable`	

Écrire un programme nommé `rotone`, qui prend en paramètre une chaîne de caractères, et qui affiche cette chaîne en remplaçant chaque caractère alphabétique par le caractère suivant dans l'ordre alphabétique.

`'z'` devient `'a'` et `'Z'` devient `'A'`. Les majuscules restent des majuscules, les minuscules restent des minuscules.

L'affichage se termine toujours par un retour à la ligne.

Si le nombre de paramètres transmis est différent de 1, le programme affiche `'\n'`.

Exemple :

```
$>./rotone "abc"
bcd
$>./rotone "Les stagiaires du staff ne sentent pas toujours tres bon." | cat -e
Mft tubhjbjsft ev tubgg of tfoufou qbt upvkpvst usft cpo.$
$>./rotone "AkjhZ zLKIJz , 23y " | cat -e
BlkiA aMLJKa , 23z $
$>./rotone | cat -e
$
$>
$>./rotone "" | cat -e
$
$>
```

II.2 Exercice 01 - rstr_capitalizer

	Exercice : 01
rstr_capitalizer	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `rstr_capitalizer.c`	
Fonctions Autorisées : `write`	
Remarques : `Exercice rétrovalidable`	

Écrire un programme qui prend en paramètre une ou plusieurs chaînes de caractères, et qui, pour chaque argument, met le dernier caractère de chaque mot (s'il s'agit d'une lettre, évidemment) 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'`.

Exemple :

```
$> ./rstr_capitalizer | cat -e
$
$> ./rstr_capitalizer "Premier PETIT TesT" | cat -e
premieR petiT tesT$
$> ./rstr_capitalizer "DeuxiEmE tEST uN PEU moinS  facile" "   attention C'EST pas dur QUAND mEmE" "ALLer
     UN DeRNier 0123456789pour LA rouTE    E " | cat -e
deuxiemE tesT uN peU moinS  facilE$
   attentioN c'esT paS duR quanD memE$
alleR uN dernieR 0123456789pouR lA routE    E $
$>
```

II.3 Exercice 02 - ft_range

	Exercice : 02
ft_range	
Dossier de rendu : *ex*02/	
Fichiers à rendre : `ft_range.c`	
Fonctions Autorisées : `malloc`	
Remarques : `Exercice rétrovalidable`	

Écrire la fonction suivante :

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

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

Exemple :

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

II.4 Exercice 03 - inter

	Exercice : 03
inter	
Dossier de rendu : $ex03/$	
Fichiers à rendre : `inter.c`	
Fonctions Autorisées : `write`	
Remarques : `Exercice rétrovalidable`	

Écrire un programme qui prend en paramètres deux chaînes de caractères et qui affiche sans doublon les caractères communs aux deux chaînes.

L'affichage se fera dans l'ordre d'apparition dans la premiere chaîne. L'affichage doit être suivi d'un `'\n'`.

Si le nombre de paramètres transmis est différent de 2, le programme affiche `'\n'`.

Exemples :

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

II.5 Exercice 04 - fprime

Exercice : 04
fprime
Dossier de rendu : *ex*04/
Fichiers à rendre : `fprime.c`
Fonctions Autorisées : `printf, atoi`
Remarques : `Exercice rétrovalidable`

Écrire un programme qui prend en paramètre un entier strictement positif, et qui affiche sa décomposition en facteurs premiers sur la sortie standard, suivie d'un `'\n'`.

Les facteurs doivent être affichés dans l'ordre croissant et séparés par des `'*'`, de telle sorte que l'expression affichée donne le bon résultat.

Si le nombre de paramètres est différent de 1, le programme doit afficher `'\n'`.

L'entrée, quand elle est passée, sera toujours un nombre valide sans caractères parasites.

Exemple :

```
$> ./fprime 225225 | cat -e
3*3*5*5*7*11*13$
$> ./fprime 8333325 | cat -e
3*3*5*5*7*11*13*37$
$> ./fprime 9539 | cat -e
9539$
$> ./fprime 804577 | cat -e
804577$
$> ./fprime 42 | cat -e
2*3*7$
$> ./fprime 1 | cat -e
1$
$> ./fprime 0 | cat -e
0$
$> ./fprime | cat -e
$
$> ./fprime 42 21 | cat -e
$
```

II.6 Exercice 05 - ord_alphlong

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

Écrire un programme qui prend en paramètre une chaîne de caractères et qui affiche les mots de cette chaîne par ordre de longueur puis dans l'ordre ASCII, avec une petite variante : En cas d'égalité alphabétique (par exemple `aA` et `Aa`) les mots doivent rester dans l'ordre où ils étaient dans la chaîne d'origine (Les majuscules et minuscules sont identiques dans l'ordre alphabétique). En cas de doublons, les doublons sont conservés.

Si le nombre de paramètres transmis est différent de 1, le programme affiche `\n`.

Dans les chaînes, il n'y aura que des espaces, des tabulations et des caractères alphanumériques.

Vous n'afficherez qu'un espace entre les mots. Aucun avant le premier ni après le dernier de chaque ligne.

Exemples :

```
$>./ord_alphlong
$
$>./ord_alphlong "De son baton il frappa la pierre et l eau jaillit" | cat -e
l$
De et il la$
eau son$
baton$
frappa pierre$
jaillit$
$>./ord_alphlong "A a b B cc ca cd" | cat -e
A a b B$
ca cc cd$
$>./ord_alphlong "Pour l Imperium de l humanite" | cat -e
l l$
de$
Pour$
humanite Imperium$
$>
```

II.7 Exercice 06 - infin_add

	Exercice : 06
infin_add	
Dossier de rendu : *ex*06/	
Fichiers à rendre : `*.c, *.h`	
Fonctions Autorisées : `write, malloc, free`	
Remarques : `Exercice rétrovalidable`	

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, et il y a toujours exactement deux paramètres, pas de pièges.

Exemple :

```
$> ./infin_add "879879087" "67548976597" | cat -e
68428855684$
$> ./infin_add "-876435" "987143265" | cat -e
986266830$
$> ./infin_add "-807965" "-34532"
-842497
$>
```

II.8 Exercice 07 - death_race

	Exercice : 07
death_race	
Dossier de rendu : *ex*07/	
Fichiers à rendre : `secret`	
Fonctions Autorisées : `Tout ce que vous voulez`	
Remarques : `Exercice bonus`	

Vous trouverez dans l'annexe du sujet un exécutable `death_race` ainsi que sa source (censurée), `race.c`.

Vous devez rendre un fichier `secret` contenant la phrase secrète qui vous est donnée par l'exécutable `death_race` fourni, sans aucun caractère ou saut de ligne supplémentaire.

II.9 Exercice 08 - time_lord

	Exercice : 08
time_lord	
Dossier de rendu : *ex*08/	
Fichiers à rendre : `secret`	
Fonctions Autorisées : `Tout ce que vous voulez`	
Remarques : `Exercice bonus`	

Vous trouverez un exécutable nommé `time_lord` dans le répertoire `misc/` de cet examen. Quand vous exécutez 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 (sans aucun caractère supplémentaire). 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.

II.10 Exercice 09 - half_life_3

	Exercice : 09
half_life_3	
Dossier de rendu : *ex*09/	
Fichiers à rendre : `secret`	
Fonctions Autorisées : `Tout ce que vous voulez, y compris les prieres vaudou`	
Remarques : `Exercice bonus`	

Vous trouverez en annexe de ce sujet un binaire `half_life_3`. Vous devez l'activer, et pour ça, il vous faut une clé.

Vous devez rendre un fichier `secret` contenant une clé acceptée par ce binaire, sans saut de ligne ni caractère supplémentaire.