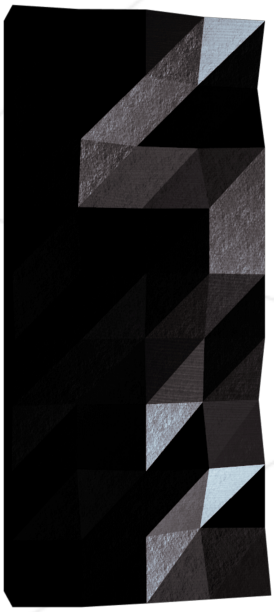

Examen

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 - max

	Exercice : 00
max	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `max.c`	
Fonctions Autorisées :	
Remarques : `Exercice rétrovalidable`	

Écrire la fonction suivante :

```
int                 max(int* tab, unsigned int len);
```

Le premier paramètre est un tableau d'int, le deuxième est le nombre d'éléments contenus dans ce tableau.

La fonction renvoie le plus grand nombre trouvé dans le tableau.

Si le tableau est vide, la fonction renvoie `0`.

II.2 Exercice 01 - str_capitalizer

Exercice : 01
str_capitalizer
Dossier de rendu : *ex*01/
Fichiers à rendre : `str_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 premier 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 :

```
$> ./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.3 Exercice 02 - ft_strdup

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

Reproduire à l'identique le fonctionnement de la fonction `strdup` (man strdup).
Votre fonction devra être prototypée de la façon suivante :

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

II.4 Exercice 03 - union

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

Èrcrire un programme nommé `union` qui prend en paramètre deux chaînes de caractères et qui affiche, sans doublon, les caractères qui apparaissent dans l'une ou dans l'autre.

L'affichage se fera dans l'ordre d'apparition dans la ligne de commande.

L'affichage doit etre suivi d'un retour à la ligne.

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

Exemple :

```
$>./union zpadinton "paqefwtdjetyiytjneytjoeyjnejeyj" | cat -e
zpadintoqefwjy$
$>./union ddf6vewg64f gtwthgdwthdwfteewhrtag6h4ffdhsd | cat -e
df6vewg4thras$
$>./union "rien" "cette phrase ne cache rien" | cat -e
rienct phas$
$>./union | cat -e
$
$>
$>./union "rien" | cat -e
$
$>
```

II.5 Exercice 04 - pgcd

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

Écrire un programme qui prend deux chaînes de caractères représentant des nombres entiers positifs non nuls en paramètre.

Les entiers représentés par les paramètres tiennent dans un int.

Afficher le plus grand diviseur commun à ces deux nombres suivi de `'\n'`. Le PGCD est toujours un entier positif non nul.

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

Exemple :

```
$> ./pgcd 42 10 | cat -e
2$
$> ./pgcd 42 12 | cat -e
6$
$> ./pgcd 14 77 | cat -e
7$
$> ./pgcd 17 3 | cat -e
1$
$> ./pgcd | cat -e
$
```

II.6 Exercice 05 - str_maxlenoc

	Exercice : 05
str_maxlenoc	
Dossier de rendu : $ex05/$	
Fichiers à rendre : `str_maxlenoc.c`	
Fonctions Autorisées : `write, malloc, free`	
Remarques : `Exercice rétrovalidable`	

Écrire un programme qui prend en paramètres n chaînes de caractères et qui affiche, suivie d'un retour à la ligne, la plus grande chaîne de caractères incluse dans toutes les chaînes passées en paramètres. Si plusieurs chaînes correspondent, on affichera celle qui apparaît en premier dans le premier paramètre. A noter que "" est forcément dans toutes les chaînes.

Si aucun paramètre n'est transmis, le programme doit afficher `'\n'`.

Soit `A` et `B` des chaînes de caracteres. On dit que `A` est inclus dans `B` si `A` est une sous-chaîne de `B` ou si `A` et `B` sont identiques.

Exemples :

```
$>./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.7 Exercice 06 - g_diam

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

Écrire un programme qui prend en paramètre une chaîne de caractères. Cette chaîne représente un graphe et est composée d'une suite d'arêtes entre les noeuds de ce graphe. Les arêtes sont séparées par un espace. Les noeuds sont représentées par des nombres et les arête par deux noeuds séparés par `'-'`. Par exemple, s'il existe une arête entre le noeud 2 et le noeud 3, les représentations possibles de cette arête sont `"2-3"` et `"3-2"`.

Le programme devra afficher le nombre de noeuds du plus long chemin, suivi d'un `'\n'`, en sachant qu'il est impossible de passer par un noeud plus d'une fois.

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

Exemple :

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

II.8 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.