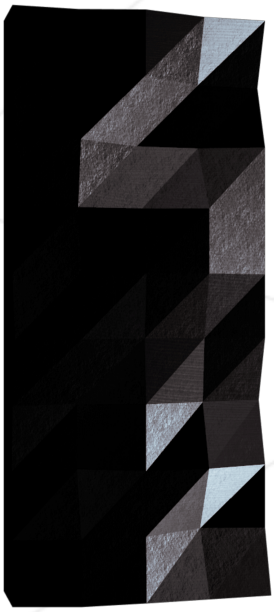

Examen

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.
  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 fi-

chier 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 - last_word

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

Écrire un programme qui prend une chaîne de caractères en paramètre, et qui affiche le dernier mot de cette chaîne, 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 le nombre de paramètres transmis est différent de 1, ou s'il n'y a aucun mot à afficher, le programme affiche `'\n'`.

Exemple :

```
$> ./last_word "FOR PONY" | cat -e
PONY$
$> ./last_word "this        ...       is sparta, then again, maybe    not" | cat -e
not$
$> ./last_word "   " | cat -e
$
$> ./last_word "a" "b" | cat -e
$
$> ./last_word "  lorem,ipsum  " | cat -e
lorem,ipsum$
$>
```

II.2 Exercice 01 - expand_str

Exercice : 01
expand_str
Dossier de rendu : *ex*01/
Fichiers à rendre : `expand_str.c`
Fonctions Autorisées : `write`
Remarques : `Exercice rétrovalidable`

Écrire un programme qui prend une chaîne de caractères en paramètre, et qui affiche cette chaîne avec exactement trois espaces entre chaque mot, sans espaces ou tabulations ni au début ni à la fin de la chaîne, suivie 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 le nombre de paramètres transmis est différent de 1, ou s'il n'y a aucun mot à afficher, le programme affiche `'\n'`.

Exemple :

```
$> ./expand_str "vous   voyez   c'est   facile   d'afficher   la   meme   chose" | cat -e
vous   voyez   c'est   facile   d'afficher   la   meme   chose$
$> ./expand_str " seulement          la c'est      plus dur " | cat -e
seulement   la   c'est   plus   dur$
$> ./expand_str "comme c'est cocasse" "vous avez entendu, Mathilde ?" | cat -e
$
$> ./expand_str "" | cat -e
$
$>
```

II.3 Exercice 02 - ft_rrange

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

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.

Exemple :

- 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.4 Exercice 03 - sort_int_tab

	Exercice : 03
	sort_int_tab
Dossier de rendu : *ex*03/	
Fichiers à rendre : `sort_int_tab.c`	
Fonctions Autorisées :	
Remarques : `Exercice rétrovalidable`	

Écrire la fonction suivante :

```
void sort_int_tab(int *tab, unsigned int size);
```

Cette fonction doit trier (en place !) le tableau d'ints `tab`, qui contient exactement `size` entrées, dans l'ordre croissant.

Les doublons doivent être préservés.

Les entrées seront toujours cohérentes.

II.5 Exercice 04 - sort_list

	Exercice : 04
sort_list	
Dossier de rendu : *ex*04/	
Fichiers à rendre : `sort_list.c`	
Fonctions Autorisées :	
Remarques : `Exercice rétrovalidable`	

Écrire la fonction suivante :

```
t_list        *sort_list(t_list* lst, int (*cmp)(int, int));
```

Cette fonction doit trier la liste passée en premier paramètre, en utilisant le pointeur sur fonction `cmp` pour déterminer l'ordre à appliquer, et renvoyer un pointeur vers le premier élément de la liste triée.
Les doublons doivent être préservés.

Les entrées seront toujours cohérentes.

Vous devez utiliser le type `t_list` décrit dans le fichier `list.h` qui vous est fourni. Vous devrez inclure (`#include "list.h"`) ce fichier, mais ne pas le rendre. Nous utiliserons le notre pour compiler votre exercice.

Les fonctions passées en tant que `cmp` renverront toujours une valeur différente de `0` si `a` et `b` sont dans le bon ordre, dans le cas contraire elles renverront `0`.

Par exemple, la fonction suivante utilisée en tant que `cmp` devra permettre de trier la liste par ordre croissant :

```
int croissant(int a, int b)
{
        return (a <= b);
}
```

II.6 Exercice 05 - md5

	Exercice : 05
md5	
Dossier de rendu : $ex05/$	
Fichiers à rendre : `md5.c`	
Fonctions Autorisées : `write, malloc, free`	
Remarques : `Exercice rétrovalidable`	

Écrire un programme qui prend une chaîne de caractères en paramètre, et qui affiche le digest `md5` de cette chaîne, suivi d'un `'\n'`.

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

Nous vous fournissons en annexe la RFC `md5` que vous devez utiliser.

Exemple :

```
$> ./md5 "" | cat -e
d41d8cd98f00b204e9800998ecf8427e$
$> ./md5 "ceci n'est pas une pipe" | cat -e
abec1071c807ebfec3045ce68ce0df47$
$> ./md5 "jamais de la vie je ne ferai ca dans mon sandwich ..." | cat -e
94f1547ed3f035b2baf450bc8e6516bb$
$> ./md5 "le bocal est beau" | cat -e
69b5af91d872e9b03cd76eb6d6a5bef6$
$> ./md5 "le bocal est" "fort" | 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 :

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