



Examen

Bocal bocal@42.fr

Résumé: Ce document est votre sujet d'examen

Table des matières

I	Détails administratifs	2
I.1	Consignes générales	2
I.2	Le Code	3
I.3	Types d'exercices	4
II	Exercices	5
II.1	Exercice 00 - first_word	5
II.2	Exercice 01 - epur_str	6
II.3	Exercice 02 - ft_range	7
II.4	Exercice 03 - sort_int_tab	8
II.5	Exercice 04 - sort_list	9
II.6	Exercice 05 - count_island	10
II.7	Exercice 06 - infin_mult	12
II.8	Exercice 07 - half_life_3	13

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 dépôt de rendu dédié à cet examen. Vous y ferez vos commits et vos pushes.
- Seul le contenu que vous avez pushé sur votre dépôt de rendu sera corrigé. Le dépôt cessera d'accepter les pushes 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 nécessaire.
- 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 fichier1.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, lorsqu'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'exécutable.
- 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 optionnel** - 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 - first_word

	Exercice : 00
	first_word
	Dossier de rendu : <i>ex00/</i>
	Fichiers à rendre : first_word.c
	Fonctions Autorisées : write
	Remarques : Exercice rétrovalidable

Écrire un programme qui prend une chaîne de caractères en paramètre, et qui affiche le premier 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 :

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

II.2 Exercice 01 - epur_str

	Exercice : 01
epur_str	
Dossier de rendu : <i>ex01/</i>	
Fichiers à rendre : epur_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 un espace 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 :

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

II.3 Exercice 02 - ft_range

	Exercice : 02
ft_range	
Dossier de rendu : <i>ex02/</i>	
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 - sort_int_tab

	Exercice : 03
	sort_int_tab
	Dossier de rendu : <i>ex03/</i>
	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 : <i>ex04/</i>	
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 - count_island

	Exercice : 05
	count_island
	Dossier de rendu : <i>ex05/</i>
	Fichiers à rendre : *.c, *.h
	Fonctions Autorisées : open, close, read, write, malloc, free
	Remarques : Exercice rétrovalidable

Écrire un programme qui prend en paramètre un fichier contenant une série de lignes de longueurs égales contenant soit le caractère '.' soit le caractère 'X'. Ces lignes forment un rectangle de '.' comportant des îlots de 'X'.

Une ligne est une suite de caractères '.' et de caractères 'X' suivie d'un retour à la ligne. Les lignes font toutes la même taille. La taille maximum d'une ligne est 1024 caractères.

Une colonne de caractères est formée par l'ensemble des caractères dans un fichier qui sont séparés par le même nombre de caractères du début de leur ligne respective.

On dit que deux caractères se touchent s'ils sont soit sur la même ligne et contigus, soit sur la même colonne et sur des lignes contigues.

Un îlot de 'X' est formé par l'ensemble des caractères 'X' qui se touchent.

Le programme doit parcourir le fichier ligne par ligne et l'afficher à l'écran en remplaçant tous les 'X' des îlots par leur numéro d'apparition dans le fichier. Le programme devra effectuer ce traitement en commençant par le début du fichier.

Il ne peut y avoir qu'un seul résultat pour un fichier donné. Aucune variation n'est admise.

Si le fichier est vide, qu'une erreur s'est produite (lignes de taille différentes par exemple) ou qu'aucun fichier n'est passé en paramètre, le programme doit seulement afficher '\n'.

Le fichier comporte au maximum 10 îlots.

Vous trouverez des exemples de fichiers d'entrée dans le répertoire `misc/`.


Exemple :

```
$>cat toto
.....XXXXXXXX.....
.....XXXXXXXX.....XXXXXXXX.....
.....XXXXXXXX.....XXX.....XXXX.....
.....XXXXX.....X.....XXXXXXXXXX.....
.....X.....
.....XXXXXXXXXXXXX.....X.....
.....X.....XXXXXXX.....
.....X.....XXXXXXXXXXXX.....
.....X.....
XX.....XXXXX
XX.....XXXXXXXXXXXX.....X
.....X
.....XXXX.....XX
$>
$>./count_island toto
.....00000000.....
.....00000000.....11111111.....
.....00000000.....111.....1111.....
.....000000.....2.....1111111111.....
.....2.....
.....333333333333.....2.....
.....3.....22222222.....
.....3.....2222222222.....
.....3.....
44.....5555
44.....666666666666.....5
.....7
.....8888.....77
$>
```

```
$>cat qui_est_la
.....
..X.....X.....XXXXX.....XXXXXX.....XXXXXXXXXX.....XXXXXXX.....
..XX.....XX.....XX.....XX.....XX.....XX.....XXXX.....XXXXXXXXXX.....
..XXXX.....XXXX.....XX.....XX.....XX.....XX.....XX.....XX.....
..XX.....XX.....XX.....XX.....XX.....XX.....XX.....XX.....
..XX.....X.....XX.....XX.....XX.....XXXXXXXXX.....XX.....XXXXX.....
..XX.....XX.....XXXXXXXXX.....XXXX.....XX.....XXXXX.....
..XX.....XX.....XX.....XX.....XX.....XX.....XX.....XX.....
..XX.....XX.....XX.....XX.....X.....XX.....XX.....
..XX.....XX.....XX.....XX.....X.....XXXXXX.....XXXXXXXXXXXX.....
..XX.....XX.....XX.....XX.....XX.....XXXXXX.....XXXXXXXXXXXX.....X.....
.....
$>
$>./count_island qui_est_la
.....
..0.....0.....1111.....2222222.....3333333333.....4444444444.....
..00.....00.....11.....11.....22.....22.....3333.....4444444444.....
..0000.....0000.....11.....11.....22.....22.....33.....44.....
..00.....0000.....00.....11.....11.....22.....22.....33.....44.....
..00.....0.....00.....11.....11.....22222222.....33.....44444.....
..00.....00.....11111111.....2222.....33.....44444.....
..00.....00.....11.....11.....22.....22.....33.....44.....
..00.....00.....11.....11.....22.....5.....33.....44.....
..00.....00.....11.....11.....22.....6.....333333.....4444444444.....
..00.....00.....11.....11.....22.....77.....3333333333.....4444444444.....8...
.....
$>
```

```
$>cat -e rien
$>./count_island rien | cat -e
$
$>
```

II.7 Exercice 06 - infin_mult

	Exercice : 06
infin_mult	
Dossier de rendu : <i>ex06/</i>	
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 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, et il y a toujours exactement deux paramètres, pas de pièges.

Exemple :

```
$> ./infin_mult "879879087" "67548976597" | cat -e
59434931855952726939$
$> ./infin_mult "-876435" "987143265" | cat -e
-865166907460275$
$> ./infin_mult "-807965" "-34532" | cat -e
27900647380$
$> ./infin_mult "-807965" "0"
0
$>
```

II.8 Exercice 07 - half_life_3

	Exercice : 07
	half_life_3
	Dossier de rendu : <i>ex07/</i>
	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.