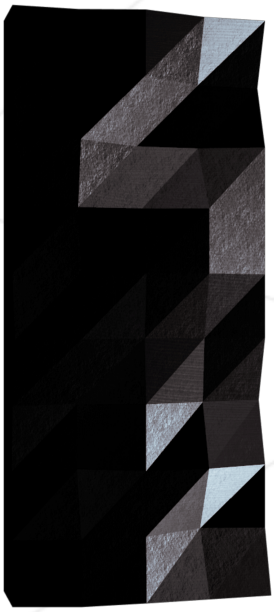

Examen

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 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 - rotone

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

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 - hidenp

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

Écrire un programme nommé `hidenp` qui prend en paramètres deux chaînes de caractères et qui affiche 1 suivi de `'\n'` si la première chaîne est cachée dans la deuxième chaîne. Sinon il affiche 0 suivi de `'\n'`.

Soit s1 et s2 des chaînes de caractères. On dit que la chaîne s1 est cachée dans la chaîne s2 si on peut trouver chaque caractère de s1 dans s2 et ce dans le même ordre que dans s1. En outre, la chaîne vide est cachée dans n'importe quelle chaîne.

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

Exemples :

```
$>./hidenp "fgex.;" "tyf34gdgf;'ektufjhgdgex.;.;rtjynur6" | cat -e
1$
$>./hidenp "abc" "2altrb53c.sse" | cat -e
1$
$>./hidenp "abc" "btarc" | cat -e
0$
$>./hidenp | cat -e
$
$>
```

II.3 Exercice 02 - union

	Exercice : 02
union	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `union.c`	
Fonctions Autorisées : `write`	
Remarques : `Exercice obligatoire`	

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.4 Exercice 03 - ord_alphlong

	Exercice : 03
	ord_alphlong
Dossier de rendu : *ex*03/	
Fichiers à rendre : `*.c, *.h`	
Fonctions Autorisées : `write, malloc, free`	
Remarques : `Exercice obligatoire`	

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 alphabétique, 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.5 Exercice 04 - g_diam

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

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.6 Exercice 05 - count_island

	Exercice : 05
count_island	
Dossier de rendu : *ex*05/	
Fichiers à rendre : `*.c, *.h`	
Fonctions Autorisées : `open, close, read, write, malloc, free`	
Remarques : `Exercice obligatoire`	

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 colone 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.....................................
...................XXXXXXXXX.......XXXXXXXX.....................
.................XXXXXXXX.............XXX...XXXXX.................
.....................XXXXXX.....X...XXXXXXXXXX...................
................................X.....................................
......XXXXXXXXXXXX.............X.....................................
...................X.............XXXXXXXXX.........................
...................X.........XXXXXXXXXXX.........................
...................X.....................................................
XX.....................................................XXXX
XX...................XXXXXXXXXXXX.............................X
.....................................................................
.....................................................................X.
.....................XXXXX.....................................XX
$>
$>./count_island toto
.................00000000.....................................
...................000000000.......11111111.....................
.................00000000.............111...11111.................
.....................000000.....2...11111111111...................
................................2.....................................
......3333333333333.............2.....................................
...................3.............222222222.........................
...................3.........222222222222.........................
...................3.....................................................
44.....................................................5555
44...................6666666666666.....................................5
.....................................................................
.....................................................................7.
.....................88888.....................................77
$>
```

```
$>cat qui_est_la
.....................................................................
...X........X.....XXXXX......XXXXXXX...XXXXXXXXXX..XXXXXXXXXX......
...XX......XX....XX...XX....XX.....XX.....XXXX.....XXXXXXXXXX......
...XXXX..XXXX...XX.....XX...XX.....XX......XX......XX..............
...XX.XXXX.XX...XX.....XX...XX.....XX......XX......XX..............
...XX...X..XX...XX.....XX...XXXXXXXX.......XX......XXXXX...........
...XX......XX...XXXXXXXXX...XXXX...........XX......XXXXX...........
...XX......XX..XX.......XX..XX.XX..........XX......XX..............
...XX......XX..XX.......XX..XX...X.........XX......XX..............
...XX......XX..XX.......XX..XX....X......XXXXXX....XXXXXXXXXX......
...XX......XX.XX.........XX.XX.....XX..XXXXXXXXXX..XXXXXXXXXX..X...
.....................................................................
$>
$>./count_island qui_est_la
.....................................................................
...0........0.....11111......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...111111111...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 - time_lord

	Exercice : 06
time_lord	
Dossier de rendu : *ex*06/	
Fichiers à rendre : `secret`	
Fonctions Autorisées : `Tout ce que vous voulez`	
Remarques : `Exercice obligatoire`	

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.