ÉCOLE D'INGÉNIEURS EN INFORMATIQUE

Piscine : journée 1 — Sujet

version **nº 3**

Assistants C/Unix 2014 <acu@epita.fr>

Copyright

Ce document est destiné à une utilisation au sein d'EPITA (site) uniquement.

Table des matières

*. https://acu.epita.fr

Obligations

*Les obligations sont des règles **fondamentales** et partagées par tous les sujets.*
*Elles ne sont **pas négociables** et ne pas les appliquer vous expose à des sanctions. Aussi, n'hésitez pas à demander des éclaircissements si vous ne comprenez pas l'une de ces règles.*

Obligation 0 : Le fichier `AUTHORS` doit être présent à la racine de votre rendu. Son format est décrit dans les consignes (cf. *Données du rendu*). L'absence de ce fichier entraînera un zéro.

Obligation 1 : La **triche**, l'échange de code source, de tests, d'outils de test ou de correction de la norme sont pénalisés par la note -42.

Obligation 2 : Vous devez rendre votre travail **avant** l'heure de rendu (cf. *Données du rendu*) : **aucun retard**, même d'une seconde, n'est possible. Ne pas rendre entraîne la note -2.

Obligation 3 : Vous devez rendre un dépôt **propre**. Sauf contre-indication spécifique et *explicite* du sujet, un dépôt *non-propre* est un dépôt qui :

- contient des fichiers binaires [1] ;
- contient des fichiers ne possédant pas les bons droits ;
- contient des fichiers interdits : `*~`, `*.swp`, `*.o`, `*.a`, `*.so`, `*.class`, `*.log`, etc. ;
- ne respecte pas l'arborescence de fichiers demandée (cf. *Données du rendu*).

Rendre un dépôt non-propre entraîne automatiquement la perte de deux points sur votre note finale.

Obligation 4 : Toute fonction, commande ou bibliothèque qui n'est pas *explicitement* autorisée est **interdite**. Les abus peuvent mener jusqu'à l'obtention de la note -21.

Obligation 5 : Lorsque des formats de sortie vous sont donnés en exemple, vous devez les respecter scrupuleusement.

Obligation 6 : Le respect de la *coding style* est obligatoire si elle existe dans les langages concernés par ce sujet. Dans tous les cas, le code rendu doit être **propre**, **lisible** et ne pas dépasser **80 colonnes**.

Conseils

- ▷ Lisez le sujet en entier.
- ▷ Vous pouvez contacter les assistants au moindre problème relatif à ce projet.
  Utilisez le ***newsgroup*** `epita.cours.c-unix.piscine` (avec la balise `[J1]`) pour des questions à propos de ce document, ou envoyez un **ticket** à travers l'intranet.
- ▷ Dans les exemples, `42sh$` représente le prompt : utilisez-le comme point de repère.
- ▷ N'attendez pas la dernière minute pour commencer le projet.

1. Si un exécutable est demandé, vous devez uniquement fournir ses sources dans le répertoire `src/` (sauf mention contraire). Elles seront compilées par nos soins.

Données du rendu

Responsables du projet :

- Nicolas Geniteau <genite_n@acu.epita.fr>
- Xavier Grand <grand_x@acu.epita.fr>

Dates clés : **début** : mardi 17 septembre 2013 19 h 00
fin : mercredi 18 septembre 2013 5 h 42
durée : 10 heures, 42 minutes

Newsgroup dédié : `epita.cours.c-unix.piscine` *avec* `[J1]`

Membres par équipe : 1

Arborescence[1] **des fichiers du rendu** :

`./src/{…}`	Contient vos fichiers sources.
`./check/{…}`	Répertoire relatif à vos tests.
`./TODO *`	Décrit les tâches à accomplir. Il doit être régulièrement mis à jour.
`./README *`	Décrit le projet et les algorithmes utilisés dans un anglais correct. Explique aussi comment utiliser votre projet.
`./AUTHORS *`	Contient les noms des auteurs du projet, sous la forme suivante : une étoile `*`, une espace, votre *login* (ex. : `login_x`) et un retour à la ligne. Ainsi :

```
42sh$ cat -e AUTHORS
* login_x$
42sh$
```

1. Les fichiers marqués d'un astérisque sont obligatoires. `.` représente la racine de votre dépôt.

1 strlen

- Nom du fichier : `strlen.sh`
- Répertoire de rendu : `src/ex1`
- Commandes autorisées : *builtins*, `grep`, `sed`, `wc`, `tr`, `head`, `tail`, `cat`, `cut`, `expr`, `stat`, `find`

1.1 Objectif

Écrivez un script shell acceptant un nombre arbitraire d'arguments et affichant, pour chaque argument, le message suivant :

`"WORD" counts N character(s)`

avec `WORD` l'argument courant et `N` son nombre de caractères. Notez que *character* doit être accordé en nombre. Ainsi :

- pour `N` < 2, le message sera :

  `"WORD" counts N character`

- pour `N` $\geq$ 2 :

  `"WORD" counts N characters`

Si une erreur survient, votre script doit quitter avec le code d'erreur 1 ; sinon, il renvoie 0. Le support des guillemets et caractères spéciaux ne sera pas testé.

*N'oubliez pas de faire **des** commits !*

1.2 Exemple

```
42sh$ ./strlen.sh foo
"foo" counts 3 characters
42sh$ echo $?
0
42sh$ ./strlen.sh
42sh$ echo $?
1
42sh$ ./strlen.sh foo bar Solarium l33t a ""
"foo" counts 3 characters
"bar" counts 3 characters
"Solarium" counts 8 characters
"l33t" counts 4 characters
"a" counts 1 character
"" counts 0 character
```

2 rot13

- Nom du fichier : `rot13.sh`
- Répertoire de rendu : `src/ex2`
- Commandes autorisées : *builtins*, `grep`, `sed`, `wc`, `tr`, `head`, `tail`, `cat`, `cut`, `expr`, `stat`, `find`

2.1 Objectif

Votre script shell doit prendre en argument un (et un seul) fichier texte et afficher sur sa sortie standard le contenu de ce fichier après avoir un rot13.

En cas d'erreur, votre script ne doit rien afficher (ni sur la sortie standard ni sur la sortie d'erreur) et doit retourner 1. En cas de succès, il retourne évidemment 0. Les tests n'impliqueront que des fichiers en ASCII.

*N'oubliez pas de faire **des** commits !*

2.2 Exemple

```
42sh$ cat -te foo.txt
J'aime les ACU.$
La piscine, c'est cool !$
42sh$ ./rot13.sh foo.txt | cat -te
W'nvzr yrf NPH.$
Yn cvfpvar, p'rfg pbby !$
```

3 80 colonnes

- Nom des fichiers : `80cols.sh`, `80cols.grep`
- Répertoire de rendu : `src/ex3`
- Commandes autorisées : *builtins*, `grep`, `wc`, `tr`, `head`, `tail`, `cat`, `cut`, `expr`, `stat`, `find`

3.1 Objectif

Le but de cet exercice est d'afficher les lignes du fichier d'entrée qui ont strictement plus de 80 caractères (retour à la ligne inclus). Pour des raisons pratiques, considérez que la tabulation est un caractère comme un autre et compte donc comme *un seul* caractère.

Dans cet exercice, vous devez mettre en application ce que vous avez vu aujourd'hui. Il vous est donc demandé de réaliser **deux versions** de ce script : une version en **`grep`** et une version en **shell**. Il est évident que la version en shell ne doit en aucun cas faire appel à `grep`.

Considérez que les appels à vos scripts seront toujours semblables à l'exemple ci-dessous et toujours valides.

*N'oubliez pas de faire **des** commits !*

3.2 Exemple

```
42sh$ cat test.txt
Ceci est une ligne courte.
Ceci est une ligne très longue, vraiment très longue, et dépasse largement les 80 colonnes. C'est mal.
Ceci est une autre ligne courte.
42sh$ grep -f 80cols.grep test.txt
Ceci est une ligne très longue, vraiment très longue, et dépasse largement les 80 colonnes. C'est mal.
42sh$ ./80cols.sh test
Ceci est une ligne très longue, vraiment très longue, et dépasse largement les 80 colonnes. C'est mal.
```

4 seq

- Nom du fichier : `seq.sh`
- Répertoire de rendu : `src/ex4`
- Commandes autorisées : *builtins*, `grep`, `sed`, `wc`, `tr`, `head`, `tail`, `cat`, `cut`, `expr`, `stat`, `find`

4.0.1 Objectif

Vous devez écrire un script shell permettant de générer des intervalles de nombres. L'usage attendu en est le suivant :

```
./seq.sh FIRST INCREMENT LAST
```

- Si le nombre d'arguments n'est pas le bon, le script doit renvoyer 1 et afficher l'usage sur la sortie d'erreur (voir exemples).
- On considère que les arguments `FIRST`, `INCREMENT`, `LAST` peuvent être négatifs mais qu'ils seront toujours entiers.
- `INCREMENT` doit être strictement positif si `LAST` > `FIRST`, strictement négatif sinon. Si ce n'est pas le cas, le script ne doit rien afficher et renvoyer 1.
- Si `LAST` > `FIRST`, on affiche les nombres n dans l'ordre croissant tel que :
  - $\texttt{FIRST} \leq n \leq \texttt{LAST}$
  - $n = \texttt{FIRST} + i \times \texttt{INCREMENT}$ ($\forall i \in \mathbb{N}, i \geq 0$)
- Si `FIRST` > `LAST`, on affiche les nombres n dans l'ordre décroissant tel que :
  - $\texttt{LAST} \leq n \leq \texttt{FIRST}$
  - $n = \texttt{FIRST} + i \times \texttt{INCREMENT}$ ($\forall i \in \mathbb{N}, i \geq 0$)
- Si `FIRST` = `LAST`, on affiche `FIRST` et le script renvoie 0.
- Le script renvoie 0 dans tous les autres cas.

Pour toutes les autres caractéristiques du script, suivre les exemples suivants.

*N'oubliez pas de faire **des** commits !*

4.0.2 Exemples

```
42sh$ ./seq.sh 1 -1 1
1
42sh$ ./seq.sh 42 -1 42
```

```
42
42sh$ ./seq.sh 42 -2 40
42
40
42sh$ ./seq.sh 42
Usage: ./seq.sh FIRST INCREMENT LAST
42sh$ echo $?
1
42sh$ ./seq.sh 3 2 1
42sh$ echo $?
1
```

Pour votre culture, notez que la commande `seq` existe réellement : si vous êtes curieux, allez voir le *man* !

5 Readable

- Nom du fichier : `readable.sh`
- Répertoire de rendu : `src/ex5`
- Commandes autorisées : *builtins*, `grep`, `sed`, `wc`, `tr`, `head`, `tail`, `cat`, `cut`, `expr`, `stat`, `find`

5.1 Objectif

Vous devez écrire un script shell qui prend en argument le nom d'un répertoire et affiche sur la sortie standard l'ensemble de ses fichiers et sous-fichiers -- mais pas les dossiers -- dont le propriétaire possède le privilège de lecture. Pour chaque dossier, les résultats devront être alphabétiquement triés.

Votre script renverra 0 si tout s'est bien déroulé, 1 si le nombre d'arguments est insuffisant et 2 pour toute autre erreur. Si une erreur se produit, le contenu de la sortie standard n'importe pas. La sortie d'erreur ne sera pas examinée.

Il existe différentes façons de procéder, à vous de trouver la plus facile !

*N'oubliez pas de faire **des** commits !*

5.2 Exemple

```
42sh$ ll -R
.:
total 8
drwxr-xr-x 4 grand_x epita_2014 4096 Sep  3 19:42 dossier1

./dossier1:
total 8
-rw-r--r-- 1 grand_x epita_2014   69 Sep  3 19:42 dossier1f1
--w------- 1 grand_x epita_2014   64 Sep  3 19:42 dossier1f2
drwxr-xr-x 2 grand_x epita_2014 4096 Sep  3 19:42 dossier2-1
drwxr-xr-x 2 grand_x epita_2014 4096 Sep  3 19:42 dossier2-2

./dossier1/dossier2-1:
total 0
-rw-r--r-- 1 grand_x epita_2014  512 Sep  3 19:42 dossier2-1-f1

./dossier1/dossier2-2:
total 0
-rw-r--r-- 1 grand_x epita_2014 1024 Sep  3 19:42 dossier2-2-f1
42sh$ ./readable.sh .
./dossier1/dossier1f1
./dossier1/dossier2-1/dossier2-1-f1
./dossier1/dossier2-2/dossier2-2-f1
./readable.sh
42sh$ echo $?
0
```

6 Dectobin

— Nom du fichier : `dectobin.sh`
— Répertoire de rendu : `src/ex6`
— Commandes autorisées : *builtins*, `grep`, `sed`, `wc`, `tr`, `head`, `tail`, `cat`, `cut`, `expr`, `stat`, `find`

6.1 Objectif

Le but de cet exercice est de convertir l'argument de la base 10 vers la base 2.
Votre script doit retourner 0 si tout s'est bien passé, 1 sinon.
*N'oubliez pas de faire **des** commits !*

6.2 Exemple

```
42sh$ ./dectobin.sh 42
101010
42sh$ echo $?
0
42sh$ ./dectobin.sh to
42sh$ echo $?
1
```

7 Mypar

— Nom du fichier : `my_par.sh`
— Répertoire de rendu : `src/ex7`
— Commandes autorisées : *builtins*, `grep`, `sed`, `wc`, `tr`, `head`, `tail`, `cat`, `cut`, `expr`, `stat`, `find`

7.1 Objectif

Le but de cet exercice est de réécrire un utilitaire pour couper un fichier en plusieurs fichiers de plus petite taille.
Vous devez écrire un script shell qui prend deux arguments :

— le nom du fichier à découper ;
— la taille de chaque fichier résultant (en octets).

Celui-ci doit créer les fichiers `fichier.ext.1`, `fichier.ext.2`, `fichier.ext.3`, `fichier.ext.4`, etc. correspondant au fichier d'entrée découpé d'après la taille demandée, sauf pour le dernier fragment qui sera probablement plus petit.
Votre script doit retourner 0 si tout s'est bien passé, 1 sinon.
*N'oubliez pas de faire **des** commits !*

7.2 Exemple

```
42sh$ head -c 10000 /dev/urandom > bigfile.bin
42sh$ ./my_par.sh bigfile.bin 1024
42sh$ echo $?
0
42sh$ ls -l bigfile.bin.*
-rw-r--r-- 1 root root 1024 2011-08-16 03:12 bigfile.bin.1
-rw-r--r-- 1 root root  784 2011-08-16 03:12 bigfile.bin.10
-rw-r--r-- 1 root root 1024 2011-08-16 03:12 bigfile.bin.2
-rw-r--r-- 1 root root 1024 2011-08-16 03:12 bigfile.bin.3
-rw-r--r-- 1 root root 1024 2011-08-16 03:12 bigfile.bin.4
-rw-r--r-- 1 root root 1024 2011-08-16 03:12 bigfile.bin.5
-rw-r--r-- 1 root root 1024 2011-08-16 03:12 bigfile.bin.6
-rw-r--r-- 1 root root 1024 2011-08-16 03:12 bigfile.bin.7
-rw-r--r-- 1 root root 1024 2011-08-16 03:12 bigfile.bin.8
-rw-r--r-- 1 root root 1024 2011-08-16 03:12 bigfile.bin.9
42sh$ cat bigfile.bin.1 bigfile.bin.2 bigfile.bin.3\
    bigfile.bin.4 bigfile.bin.5 bigfile.bin.6 bigfile.bin.7\
    bigfile.bin.8 bigfile.bin.9 bigfile.bin.10 > out.bin
42sh$ cmp bigfile.bin out.bin
```

```
42sh$ echo $?
0
```

Pour votre culture, sachez qu'il existe une commande, `split`, capable de ce genre de découpage : si vous êtes curieux, allez voir le *man* !

8 my_zmv

- Nom du fichier : `my_zmv.sh`
- Répertoire de rendu : `src/ex8`
- Commandes autorisées : *builtins*, `grep`, `sed`, `wc`, `tr`, `head`, `tail`, `cat`, `cut`, `expr`, `stat`, `find`

8.1 Objectif

Bientôt, vous aurez l'occasion de découvrir *The Z Shell*, le shell des Gens Biens, qui offre une fonction nommée `zmv`. Le but de cet exercice est de fournir une fonctionnalité similaire à `zmv` (`man zshcontrib`), c'est-à-dire changer l'extension de plusieurs fichiers d'un coup.

Si le nombre d'arguments n'est pas valide, seul l'usage doit être affichée (sur la sortie d'erreur). Les messages à afficher sont donnés dans les exemples ci-dessous.

L'appel `./my_zmv.sh '*.a' '*.b'` doit ainsi changer l'extension de tous les fichiers dont le nom se termine en `.a` en `.b`. Avant d'effectuer le renommage, votre script doit afficher :

```
mv `source.a' `destination.b'
```

sur sa sortie standard (le nom de chaque fichier doit être précédé d'un *backquote* et suivi d'un *simple quote*).

Si tous les renommages sont effectués avec succès, votre script doit retourner 0. Dans le cas où au moins l'un d'entre eux n'aurait pu être renommé, votre script doit retourner 1.

*N'oubliez pas de faire **des** commits !*

8.1.1 Exemples

```
42sh$ ./my_zmv.sh
Usage: my_zmv.sh '*.ext1' '*.ext2'
42sh$ echo $?
1
42sh$ ./my_zmv.sh a b
Invalid argument `a'
Usage: my_zmv.sh '*.ext1' '*.ext2'
42sh$ echo $?
1
42sh$ ./my_zmv.sh \*.a b
Invalid argument `b'
Usage: my_zmv.sh '*.ext1' '*.ext2'
42sh$ echo $?
1
42sh$ touch foo.c
42sh$ ./my_zmv.sh '*.c' '*.cc'
mv `foo.c' `foo.cc'
42sh$ echo $?
0
42sh$ ls foo.*
foo.cc
42sh$ ./my_zmv.sh '*.wtf' '*.acu'
42sh$ echo $?
0
42sh$ touch bar.cc
42sh$ ./my_zmv.sh '*.cc' '*.c'
mv `bar.cc' `bar.c'
mv `foo.cc' `foo.c'
42sh$ echo $?
0
```

9 La tour, prends garde !

— Nom du fichiers : `tower.sh`
— Répertoire de rendu : `src/ex9`
— Commandes autorisées : *builtins*, `grep`, `sed`, `wc`, `tr`, `head`, `tail`, `cat`, `cut`, `expr`, `stat`, `find`

9.1 Objectif

Écrivez un script shell qui dessine une tour en ASCII-art. Ce script prend en argument le nombre d'étages de la tour et le type de ses fenêtres. Le reste du dessin doit être rigoureusement identique aux exemples ci-dessous. Les types de fenêtres sont `square` ou `triangle`, comme le montrent les exemples suivants.

Valeurs de retour :

— pas d'erreur : 0 ;
— sinon 1.

Remarques :

1. Si les arguments sont incorrects, le script ne doit rien afficher du tout et quitter avec 1 comme valeur de retour ;
2. Il faut bien faire attention à ne pas afficher d'espaces inutiles ;
3. Un nombre d'étages négatif est considéré comme une erreur.

*N'oubliez pas de faire **des** commits !*

9.2 Exemple

```
42sh$ ./tower.sh 2 square
 _________________
|  ___       ___  |
| |_|_|     |_|_| |
| |_|_|     |_|_| |
|                 |
|  ___       ___  |
| |_|_|     |_|_| |
| |_|_|     |_|_| |
|                 |
|        _        |
|       | |       |
|       | |       |
|_______| |_______|
42sh$ echo $?
0
42sh$ ./tower.sh 1 triangle
 _________________
|                 |
|   /\       /\   |
|  /__\     /__\  |
|                 |
|        _        |
|       | |       |
|       | |       |
|_______| |_______|
42sh$ ./tower.sh 0 triangle
 _________________
|        _        |
|       | |       |
|       | |       |
|_______| |_______|
42sh$ ./tower.sh 3 rectangle
42sh$ echo $?
1
42sh$ ./tower.sh -42 triangle
42sh$ echo $?
1
```

10 Mathématiques

— Nom du fichier : `add_matrix.sh`
— Répertoire de rendu : `src/ex10`
— Commandes autorisées : *builtins*, `grep`, `sed`, `wc`, `tr`, `head`, `tail`, `cat`, `cut`, `expr`, `stat`, `find`

10.1 Objectif

Pour cet exercice, vous devez écrire un script shell qui permet d'additionner deux matrices et qui affiche le résultat de cette addition sur la sortie standard. Les fichiers donnés en entrée (matrices) seront toujours valides.

Codes d'erreur : - 0 si tout s'est bien passé ; - 1 si l'un des arguments est manquant, si un des fichiers est manquant ou si les matrices ne sont pas compatibles ; - 2 pour tout autre type d'erreur.

*N'oubliez pas de faire **des** commits !*

10.2 Exemple

```
42sh$ cat -e mat_A.txt
1 6 9 0$
3 4 1 3$
8 1 7 2$
42sh$ cat -e mat_B.txt
8 2 1 6$
5 9 2 0$
3 6 6 2$
42sh$ ./add_matrix.sh mat_A.txt mat_B.txt | cat -e
9 8 10 6$
8 13 3 3$
11 7 13 4$
42sh$ echo $?
0
42sh$ ./add_matrix mat_A
42sh$ echo $?
1
```

11 Prototypes

— Nom du fichier : `prototypes.sh`
— Répertoire de rendu : `src/ex11`
— Commandes autorisées : `sed`

11.1 Objectif

À partir d'un fichier C donné en argument, écrivez un script qui affiche les prototypes des fonctions déclarées dans le fichier. On garantit que le fichier suit la *coding style* EPITA (cf. intranet).

Le but de l'exercice n'est pas de vous planter à cause d'expressions rationnelles trop compliquées. Considérez donc que chaque prototype tient sur une seule ligne.

Les tests que nous effectuerons ne seront pas plus compliqués que l'exemple qui suit.

*N'oubliez pas de faire **des** commits !*

11.2 Exemple

```
42sh$ cat facto.c
#include <stdio.h>

struct test
{
  int i;
};

int fact(int *val)
```

```
{
  if (val > 1)
    return (val * fact(val-1));
  else
    return 1;
}

int main(void)
{
  int val;
  scanf("%d", &val);
  printf("%d\n", fact(val));
  return 0;
}
42sh$ sed -f prototypes.sed facto.c
int fact(int *val);
int main(void);
```

12 Latin

- Nom du fichier : `latin.sh`
- Répertoire de rendu : `src/ex12`
- Commandes autorisées : *builtins*, `grep`, `sed`, `wc`, `tr`, `head`, `tail`, `cat`, `cut`, `expr`, `stat`, `find`

12.1 Objectif

Pour cet exercice, vous devez écrire un script shell qui traduira tout texte en pseudo langue latine. Si la traduction s'est correctement effectuée, le script affichera le texte traduit et retournera 0. Si un erreur est rencontrée, le script n'affichera rien du tout et retournera 1. Le script ne prendra qu'un seul paramètre. La sortie d'erreur ne sera pas vérifiée.

Principe : la moulinette qui vérifiera votre travail ne parle bien évidemment pas latin. Il vous est seulement demandé de placer un « um », « ae » ou « us » à la fin de chaque mot. Vous devrez aussi retirer toutes les espaces du texte original, ainsi que tous les signes de ponctuation. Tous les caractères alphabétiques devront finalement être capitalisés.

Note : L'ordre des suffixes est *UM* > *AE* > *US*.

Les chiffres doivent être traduits en chiffres romains :

- 42 devient XLII ;
- 0 reste 0.

Tous les caractères blancs (`\n`, `\t`, espace, etc.) devront être supprimés. Les *quotes* ne seront pas testés. Les mots composés de lettres et de chiffres doivent également être traduits par la méthode classique.

*N'oubliez pas de faire **des** commits !*

12.1.1 Exemple

```
42sh$ cat textefr.txt
Shell is good for you. You know it. 42. toto42
42sh$ ./latin.sh textefr.txt
SHELLUMISAEGOODUSFORUMYOUAEYOUUSKNOWUMITAEXLIITOTO42US
42sh$ echo $?
0
42sh$ ./latin.sh
42sh$ echo $?
1
```

13 Sed Warrior

- Nom du fichier : `warrior.sed`
- Répertoire de rendu : `src/ex13`
- Commandes autorisées : `sed`

13.1 Objectif

Le but de cet exercice est de lire un fichier source C, d'extraire le corps de chaque fonction et d'en afficher le nombre de lignes (et tout ceci avec `sed` uniquement !). Vous ne devez pas compter les lignes vides.

Le format de la sortie est :

- début du corps de la fonction : `[ Start ]` ;
- pour chaque ligne du corps : l'afficher normalement ;
- fin du corps de la fonction : `[ End ]` ;
- affichage du nom de la fonction suivi de `":"`, une espace et le nombre de lignes de la fonction si il y en a 25 ou moins, ou `"Too many lines (> 25)"`.

Votre script sera invoqué avec la commande

```
sed -n -f warrior.sed input.c
```

*N'oubliez pas de faire **des** commits !*

13.2 Exemples

```
42sh$ sed -n -f warrior.sed countme.c
[ Start ]
  int   i;

  i = 42;
  return (i);
[ End ]
main: 3
[ Start ]
  return (malloc(sizeof (int)));
[ End ]
alloc_int: 1
[ Start ]
[ End ]
empty_function: 0
42sh$ sed -n -f warrior.sed code_ing1.c
[ Start ]
  int i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
  ++i;
```

```
  ++i;
  return i;
[ End ]
fonction_de_goret: Too many lines (> 25)
42sh$
```

Why so sleepy?