Correction du Partiel de Compilation

Promo 2003 - Janvier 2001

Dans cette épreuve, vous avez le droit d'être intelligent(e)s et de prendre des initiatives. Beaucoup de choses ne sont pas écrites, d'autres semblent douteuses, certaines paraissent triviales, mais c'est parce que c'est à vous à vous débrouiller, ce ne sont pas des erreurs. Mais si ! Puisque je vous le dis !

1 Backus Naur

Donnez une grammaire EBNF pour chacun des langages suivants.

1. Les nombres hexadécimaux à virgule fixe (sans exposant). Par exemple `CafeDeca.42`.
2. Nombres hexadécimaux à virgule flottante (avec exposant). Donner des exemples.

On se donnera le droit d'utiliser - comme dans `[ 'a' - 'z' ]`. On sera rigoureux dans la différenciation entre terminaux et non terminaux.

2 Egrep

`egrep` fonctionne comme `grep` avec quelques différences de syntaxe. On utilise les parenthèses pour grouper, le point d'interrogation pour signifier "0 ou 1 fois", et le plus pour "1 fois ou plus".

`\1`, `\2`... font référence à ce qui a été apparié avec le premier, second etc. groupe formé par des parenthèses (i.e., `echo "aba" | egrep '(.).\1'` réussit, mais `echo "abc" | egrep '(.).\1'` échoue)

Remarquez que `echo a | egrep '(.)?.\1'` ne réussit pas, le premier groupe n'est *pas* défini, alors que `echo a | egrep '(.?).\1'` réussit : le premier groupe attrape le mot vide, donc `\1` également.

L'option `-v` demande l'impression des ligne *non* concordantes.

L'option `-c` demande à `egrep` de retourner le nombre de lignes qui auraient été affichées.

1. Que fait `egrep '[^a-c]*' file` ? (`file` étant un fichier, au cas où ça n'aurait pas été clair...)
2. Écrire une ligne de commande qui attrape les lignes contenant un palindrome de 3, 5, 7 ou 9 caractères (quelconques).
3. Écrire une ligne de commande qui attrape toutes lignes composées d'un palindrome de 3, 5, 7 ou 9 caractères (quelconques).
4. Écrire une ligne `egrep` qui simule `wc -l` (compte du nombre de lignes).
5. Le fichier `dots` uniquement des lignes de points. Écrire une ligne de commande qui n'imprime que les lignes dont la longueur est un nombre composé (i.e., un nombre pas premier ni 1 : 4, 6, 8, 9...).

3 Analyse LR(1)

```
1. G ::= P          3. P ::= 'id' ':' R       4. R ::=
2.     | P G                                  5.     | 'id' R
```

1. Comment classeriez-vous cette grammaire dans la hiérarchie de Chomsky ?
2. Comment classeriez-vous le langage qu'elle engendre ?
3. Construire l'automate LR(1) pour cette grammaire. On donnera le diagramme de l'automate, et le tableau.
4. Est-elle LR(0), SLR, LR(1), LR(k), LL(0), LL(k) ? Justifier pour chacune de ces catégories.

4 Tiger

Écrire un programme Tiger qui introduise :

type list le type enregistrement composé de `head`, un entier, et de `tail`, une `list`.

fonction list qui prend deux paramètres, un entier et une liste, et retourne la liste commençant par cet entier, suivie par la liste.

fonction tsil qui affiche une liste dans l'ordre inverse.

variable list qui contient la liste 1, 2, 3.

Le programme devra afficher la liste `list` à l'envers.

5 Egrep le retour

Ceux qui ont fait le reste peuvent, plutôt que de sortir se les peler dans le froid, rester au chaud et écrire un petit shell script qui écrive les nombres premiers plus petits que 100 sans utiliser aucun des utilitaires de calcul (`bc`, `expr` etc.) : uniquement des outils très classiques.

À tout hasard, je vous rappelle que `wc` permet de compter le nombre de lignes (option `-l`) ou le nombre de caractères (`-c`) *y compris* le retour à la ligne : `echo toto | wc -c` donne 5. Quant à `tr`, il permet de supprimer certains caractères, par exemple `echo toto | tr -d o` donne `tt`.

Correction:

```
dots='.'
while test `echo $dots | wc -c` -lt 100
do
  dots=.$dots
  echo $dots | egrep -v '^(..+)\1+$' | tr -d '\n' | wc -c | grep -v 0
done
```