UNIVERSITÉ PARIS SUD

Examen final

Cours d'algorithmique

—Licence CFA / Troisième Année—

Les documents manuscrits, sujets de travaux pratiques et dirigés ainsi que les supports de cours sont autorisées. Tous les autres documents tels que livres, calculatrices, téléphones portables et ordinateurs sont interdits.

L'ensemble des questions demandant la rédaction d'un algorithme peuvent être rédigées en **pseudo-code, python, C, C++ ou Java**. La syntaxe du langage n'a pas d'importance tant que celle-ci reste **cohérente** et **compréhensible**. De plus, on pourra, si on le souhaite, **remplacer les fonctions** qui manipulent des listes, tableaux, *etc* **par des méthodes** des classes correspondantes. On précisera alors pour chaque code écrit, si c'est une méthode et à quelle classe elle appartient.

Les exercices sont indépendants. Si l'on ne sait pas justifier une question, on peut néanmoins utiliser la réponse dans la suite.

Durée : 2 heures

▸ **Exercice 1.** On travaille avec des listes simplement chaînées d'entiers avec une fausse tête suivant la structure de donnée vue en cours.

1. Écrire une fonction `list partition(list lst);` qui prend en paramètre une liste `lst` et qui supprime de cette liste les nombres négatifs en les conservant dans une deuxième liste `res`. Cette deuxième liste sera retournée par la fonction. Par exemple si `lst` $= \to 2 \to -1 \to -3 \to 6 \to 5 \to -2$, la fonction modifie la liste en `lst` $= \to 2 \to 6 \to 5$ et retourne une nouvelle liste contenant `res` $= \to -1 \to -3 \to -2$. On s'efforcera de faire le moins possible d'allocations.
2. Quelle est la complexité de cette fonction ?

▸ **Exercice 2.** On travaille avec des listes simplement chaînées d'entiers avec une fausse tête suivant la structure de donnée vue en cours.

1. Écrire une fonction `concat` qui prend en paramètre deux listes `l1` et `l2` et qui les modifie pour déplacer les éléments de `l2` à la fin de la liste `l1`, sans changer leur ordre. Au retour de la fonction, la liste `l2` devra être vide. Par exemple, si en entrée on a

   $$\texttt{l1} = \to 2 \to 7 \to 3 \qquad \text{et} \qquad \texttt{l2} = \to 5 \to 4$$

   au retour on doit avoir

   $$\texttt{l1} = \to 2 \to 7 \to 3 \to 5 \to 4 \qquad \text{et} \qquad \texttt{l2} = \texttt{vide}$$

2. Quel est la complexité de cette fonction ?

On souhaite changer la structure de données des listes (têtes, chaînage) pour avoir une meilleure complexité.

3. Proposer une nouvelle structure de donnée. On écrira les constructeurs nécessaires.
4. Écrire la nouvelle fonction `concat` adaptée à cette nouvelle structure.
5. Quelle est la complexité de cette nouvelle fonction ?

▸ **Exercice 3. (Éditeur de texte en utilisant un zipper)**

Dans cet exercice, on suppose défini un type `Pile` pour stocker des piles de caractères. On utilisera les fonctions suivantes **que l'on suppose déjà écrites** :

- `isEmpty(l : Pile) : Booléen` qui retourne `vrai` si la pile est vide et `faux` sinon ;
- `push(l : Pile, c : Caractère)` qui modifie la pile `l` en insérant `c` en tête ;
- `pop(l : Pile) : Caractère` qui modifie la pile `l` en supprimant le premier caractère et qui retourne ce caractère. Une `Erreur` est retournée si la pile est vide.

Note : on ne demande pas de réécrire ces fonctions.

Pour manipuler une ligne de texte dans un éditeur, on utilise la structure de donnée suivante :

```
Ligne = structure:
    left : Pile;
    right : Pile;
```

où la pile `right` contient les caractères qui sont *à droite du curseur* dans l'ordre, et la pile `left` contient les caractères qui sont *à gauche du curseur* dans **l'ordre inverse**. Par exemple, le texte «`Bonjou`|`r␣Jean␣!`», où «|» symbolise le curseur et «␣» un espace, est codé par

`Ligne:`
$$\texttt{left=} \to u \to o \to j \to n \to o \to B$$
$$\texttt{right=} \to r \to \text{␣} \to J \to e \to a \to n \to \text{␣} \to !$$

En utilisant `isEmpty`, `push` et `pop`, on demande d'écrire les fonctions suivantes et de préciser leurs complexités :

1. `moveRight(ln : Ligne)` qui déplace le curseur d'un caractère à droite, si c'est possible ;
2. `insert(ln : Ligne, c : Caractère)` qui insère le caractère `c` sous le curseur dans la ligne ; après l'insertion, le curseur est placé *à droite* du caractère inséré ;
3. `backSpace(ln : Ligne)` qui supprime le caractère *à gauche* du curseur dans la ligne, si c'est possible ;
4. `home(ln : Ligne)` qui déplace le curseur en début de ligne.

▸ **Exercice 4.** On considère le programme suivant :

```
1  int f(int n) {
2    if (n == 0) return 0
3    else return n - f(n-1)
4  }
```

1. Pour quelles valeurs de `n` la fonction `f` termine-t-elle ? Comment le modifier pour qu'elle termine quel que soit `n` ?
2. Que calcule la fonction `f` (indication : on pourra calculer les premières valeurs, trouver une règle générale et la justifier) ?