Université Bordeaux 1 — Interro du 15 octobre 2013
Département Licence, **J1IN3W01 Algorithmique S3** - groupe A3 — 14h-14h30

Nom : Prénom :

Aucun document n'est autorisé

Exercice 1 On considère des suites d'entiers dans $\{0, 1\}$ stockés dans une liste L, comme par exemple $L = (0, 0, 1, 0, 0, 1, 1, 1)$. On dira qu'une liste est *équilibrée* si elle contient autant de 0 que de 1.

1. Ecrire la fonction `estEquilibre` qui permet de savoir si une liste est équilibrée. La fonction doit retourner 0 lorsque la liste est équilibrée. Lorsque la liste n'est pas équilibrée, la fonction retourne la *différence* entre le nombre de 0 et de 1 (une valeur positive lorsque le nombre de 0 est supérieure au nombre de 1, négative sinon).

   On utilise ici le type liste chaînée d'entiers `listeSC_int` pour lequel on pourra utiliser les primitives du type abstrait `listeSC`.

   ```
   fonction estEquilibre(val L: listeSC_int): entier;
   ```

2. On souhaite maintenant écrire une fonction `equilibrer` qui complète la liste L pour la rendre équilibrée si elle ne l'est pas, en insérant en fin de liste un nombre minimum d'entiers 0 ou 1.

   ```
   fonction equilibrer(ref L: listeSC_int): vide;
   ```

Exercice 2 On considère maintenant des listes d'objets.

1. Ecrivez une fonction qui teste si une liste L contient un objet donné X :
   `fonction appartient(val L : listeSC, val X: objet): booleen;`
2. On souhaite écrire une fonction : `supprimer_occurences_multiples` qui permet de mettre la main sur les éléments de cette liste, sans qu'il n'y ait de doublons. Par exemple, si de la liste (de caractères) $L = (a, b, a, c, d, b, b, c, a)$ on supprime les doublons, on doit obtenir une liste contenant les éléments $\{a, b, c, d\}$, une fois chacun.
   – Quelle peut être la signature (définition) de cette fonction ?
   – Donnez une implémentation de cette fonction à l'aide des primitives de `listeSC`.
   (Les deux questions sont évidemment liées.)

Exercice 3 On considère des chaînes de caractères stockées comme liste chaînée de caractères. On souhaite écrire une fonction qui permet d'inverser la chaîne de caractères (par exemple, si on inverse `bonjour`, on obtient `ruojnob`).

1. Qu'est-il préférable d'utiliser pour implémenter cette opération ? Une liste simplement chaînée ou une liste doublement chaînée ? Justifiez votre réponse.

Exercice 4 Soit L une `listeSC_car` implémentée avec un tableau :

Taille stock	10
vListe	**T**
Premier	2
Premier libre	3
Clé	6

T	1	2	3	4	5	6	7	8	9	10
valeur	A	S	B	L	R	U	S	E	F	T
indexSuivant	4	1	5	6	8	10	0	9	0	7

1. Dessinez la liste L correspondante ; dessinez aussi la liste des cellules libres.
2. Indiquez comment la liste L est modifiée après l'action `supprimerApres(L)` : donnez cette fois les tableaux et structures correspondantes.

Annexe. Liste des primitives du type abstrait liste simplement chainée. **Rappel** : les primitives n'autorisent pas la manipulation directe de la clé.

```
listeSC = liste de type_predefini;
```

Accès

```
fonction valeur(ref L:listeSC d'objet) : objet;
/* si la cle == NIL alors le resultat est NULL */

fonction debutListe(ref L:listeSC d'objet) :vide;
/* positionne la cle sur le premier objet de la liste */

fonction suivant(ref L:listeSC d'objet) : vide;
/* avance la cle d'une position dans la liste */

fonction listeVide(ref L:listeSC d'objet) : booleen;

fonction estFinListe(ref L:listeSC d'objet) : booleen;
/* determine si la cle est en fin de liste (apres le dernier element) */
```

Modification

```
fonction creerListe(ref L:listeSC d'objet) : vide;

fonction insererApres(ref L:listeSC d'objet, val x:objet;) : vide;
/* insere un objet apres  la cle, la cle ne change pas */

fonction insererEnTete(ref L:listeSC d'objet, val x:objet) : vide;
/* insere un objet en debut de liste,
    la cle est positionnee sur la tete de liste */

fonction supprimerApres(ref L:listeSC d'objet) : vide;
/* supprime l'objet apres la cle, la cle ne change pas */

fonction supprimerEnTete(ref L:listeSC d'objet) : vide;
/* supprime un objet en debut de liste,
   la cle est positionnee sur la tete de liste */

fonction detruireListe(ref L:listeSC d'objet) : vide;
```

Primitives supplémentaires pour le type abstrait liste doublement chainée.

```
fonction finListe(ref L:listeDC_car): vide;
fonction precedent(ref L:listeDC_car) : vide;
```