Algorithmique
Contrôle n° 2

Info-Sup – Epita

D.S. 312225.79 BW (24 mar 2011 - 10 :00)

Remarques (à lire !) :

- □ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - Aucune réponse au crayon de papier ne sera corrigée.
- □ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- □ **Les algorithmes :**
  - Tout algorithme doit être écrit dans le langage Algo vu en td (pas de C(#), Caml ou autre).
  - Tout code Algo non indenté ne sera pas corrigé.
  - Tout ce dont vous avez besoin (types, routines) est indiqué en annexe.
  - Aucune fonction ou procédure non écrite ne peut être utilisée.
- □ Respectez les impositions du sujet !
- □ Durée : 2h00.

Exercice 1 (ABR : chemin de recherche – *2 points*)

Soit un arbre binaire de recherche contenant des valeurs entières. On désire chercher la valeur 42. Quelle(s) séquence(s) parmi les suivantes, **ne pourrai(en)t pas** être une suite de noeuds parcourus ? Indiquer sur les feuilles de réponses le(s) numéro(s) de séquence impossible.

① 18 - 27 - 60 - 47 - 29 - 42
② 57 - 18 - 53 - 55 - 48 - 42
③ 30 - 38 - 40 - 48 - 50 - 42
④ 36 - 46 - 38 - 44 - 40 - 42

Exercice 2 (ABR : insertions – *3 points*)

On veut créer un arbre binaire de recherche par insertions successives, à partir d'un arbre vide, des valeurs `U, N, M, O, C, H, E, A, B, R`.

Donner le résultat (dessiner l'arbre final) lorsque ces valeurs sont ajoutées :

1. en feuille ;
2. en racine.

Exercice 3 (ABR : recherche du minimum – *4 points*)

1. Où se trouve la valeur minimale d'un arbre binaire de recherche ?
   En déduire le principe d'un algorithme qui recherche la valeur minimale dans un arbre binaire de recherche non vide.
2. Écrire la fonction qui, à partir d'un arbre binaire de recherche non vide, retourne la valeur de son plus petit élément. Utiliser les opérations du type abstrait *ArbreBinaire* données en annexe.

Exercice 4 (Listes chaînées - Croissance – *5 points*)

Après avoir donné son principe, écrire la fonction qui détermine si une liste chaînée (du type `t_pListe` donné en annexe) est strictement croissante.

Exercice 5 (Listes chaînées : suppression – *6 points*)

Écrire un algorithme **itératif** qui supprime la première occurrence d'un élément dans une liste chaînée (de type `t_pListe`) triée en ordre croissant. L'algorithme devra indiquer si la suppression a eu lieu.

Annexe

Implémentation dynamique des listes

Les listes chaînées sont les mêmes que celles utilisées en TD, et sont représentées par le type suivant :

```
types
    /* déclaration du type t_element*/
   t_pListe = ↑t_noeud

   t_noeud = enregistrement
       t_element   valeur
       t_pListe    suivant
    fin enregistrement t_noeud
```

Type algébrique abstrait d'un arbre binaire :

TYPE

ArbreBinaire

UTILISE

Nœud, Élément

OPÉRATIONS

arbre-vide	:	→ ArbreBinaire
<_, _, _>	:	Nœud × ArbreBinaire × ArbreBinaire → ArbreBinaire
racine	:	ArbreBinaire → Nœud
contenu	:	Nœud → Élément
g	:	ArbreBinaire → ArbreBinaire
d	:	ArbreBinaire → ArbreBinaire

PRÉCONDITIONS

$racine(B_1)$ **est-défini-ssi** $B_1 \neq$ *arbre-vide*
$g(B_1)$ **est-défini-ssi** $B_1 \neq$ *arbre-vide*
$d(B_1)$ **est-défini-ssi** $B_1 \neq$ *arbre-vide*

AXIOMES

$racine(<o, B_1, B_2>) = o$
$g(<o, B_1, B_2>) = B_1$
$d(<o, B_1, B_2>) = B_2$

AVEC

B_1, B_2 : ArbreBinaire
o : Nœud