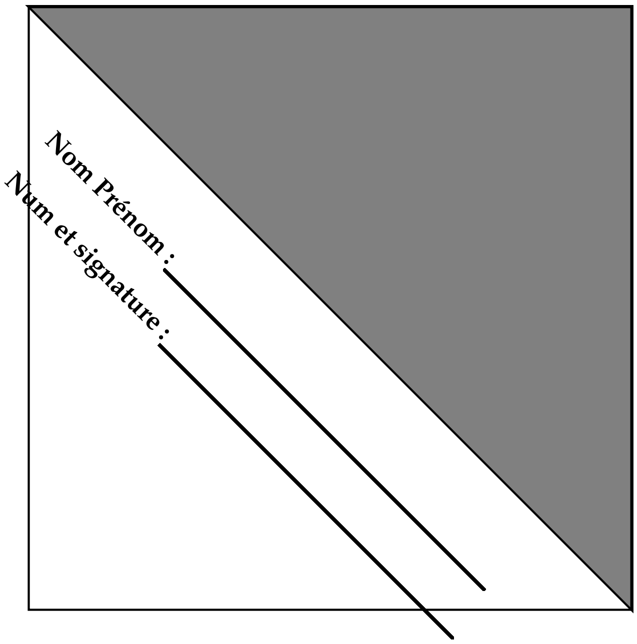
Université Claude Bernard, Lyon 1
Licence Sciences et Technologies - L3
Algorithmique, Programmation et Complexité - LIF9   2008–2009

Examen du 13 janvier 2009



Durée : 1h30 minutes
Documents, calculatrices et téléphones portables non autorisés.

Note :

Attention :

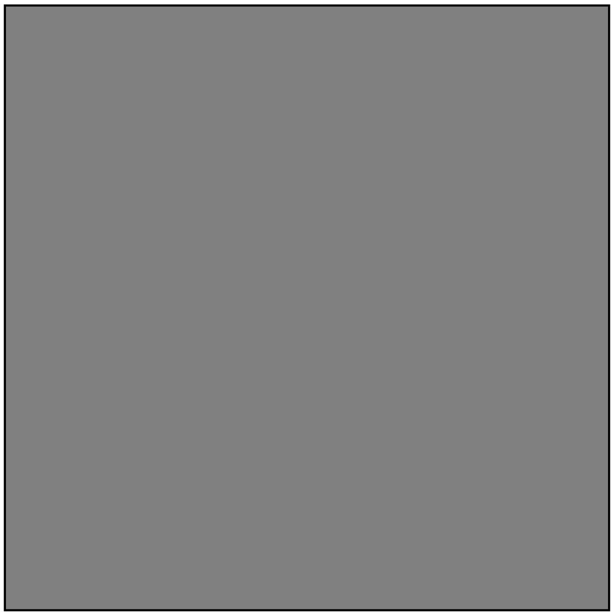
- La qualité de l'**écriture** et de la **présentation** sera prise en compte (utilisez votre brouillon pour composer vos réponses).
- Il sera attaché plus d'importance à la **lisibilité** et à la **justesse** des algorithmes qu'à leur rigueur syntaxique.
- Ne pas oublier de **commenter** les algorithmes, en signalant les résultats, préconditions et postconditions s'il y a lieu. Ne pas oublier non plus de **déclarer** les variables auxiliaires utilisées. Soigner les en-têtes de sous-programmes : noms, types et modes de transmission des paramètres, entre autres.
- Il est conseillé de lire **tout** le sujet avant de commencer à rédiger.

Généricité

On veut construire un type $Liste$ générique d'$Element$s dont l'utilisation correspond à une gestion saine de la mémoire : chaque liste est propriétaire des éléments qu'elle contient (c'est à dire qu'elle contient des clônes des éléments qu'on y insère). On utilise le langage C auquel on ajoute les références du C++.

1. Sachant que la généricité est ici gérée avec des $void*$, comment est défini le type $Element$?

2. Donnez les structure de données correspondant à la mise en œuvre du type $Liste$.



Voici à présent un exemple d'utilisation du type *Liste* générique.

```
Liste l1, l2;
initialise_liste(l1), initialise_liste(l2);
for (int i=0; i<5; i++)
  ajoute_en_tete_liste(l1, &i, initialise_elt_int);
for (double d=10.0; d>1.0; d=d/2)
  ajoute_en_tete_liste(l2, &d, initialise_elt_double);
testament(l1,testament_elt_int);
testament(l2,testament_elt_double);
```

3. A quoi correspondent les arguments *initialise_elt_int* et *initialise_elt_double* ? Donnez le prototype et le code correspondant à *initialise_elt_double*.

4. Donnez le prototype et le code de la procédure $ajoute\_en\_tete\_liste$ qui permet d'ajouter un élément en tête d'une $Liste$.

Expression arithmétique

On considère la procédure $initialiseABExpression$ qui initialise un arbre binaire comme l'arbre d'une expression arithmétique lue dans un flot de données.

```
procédure initialiseABExpression(résultat a : AB, donnée-résultat f: flot)
début
 a.racine <- lit_expression(f)
 ...
fin
```

On rappelle les fonctions **internes** $lit\_expression$, lit_terme et $lit\_facteur$ utilisées dans cette procédure $initialiseABExpression$:

```
Noeud * lit_expression(donnée-résultat f : flot)
variables
 b1, b2 : Noeud *
 op : opérateur
début
 b1 <- lit_terme(f)
 tantque f contient ensuite + ou - faire
    op <- lit_opérateur(f)
    b2 <- lit_terme(f)
    b1 <- reserve noeud(op,b1,b2) // noeud ayant comme sous-arbres
                                  // gauche et droit b1 et b2
                                  // et comme info op
 fintantque
 retourne b1
fin

Noeud * lit_terme(donnée-résultat f : flot)
variables
  b1, b2 : Noeud *
  op : opérateur
début
  b1 <- lit_facteur(f)
  tantque f contient ensuite * ou / faire
    op <- lit_opérateur(f)
    b2 <- lit_facteur(f)
    b1 <- reserve noeud(op,b1,b2)
  fintantque
  retourne b1
fin

Noeud * lit_facteur(donnée-résultat f : flot)
variables
  Noeud * b, constante c
début
  si le prochain lexème dans le flot est ( alors
  lit_parenthèse_ouvrante(f)
  b <- lit_expression(f)
  lit_parenthèse_fermante(f)
  sinon
    c <- lit_constante(f)
    b1 <- reserve noeud(c,NULL,NULL)
  finsi
  retourne b1
fin
```

$lit\_constante$, $lit\_parenthese\_ouvrante$, $lit\_parenthese\_fermante$, $lit\_operateur$ permettent respectivement de récupérer la constante, la parenthèse ou l'opérateur censé être présent dans le flot (précondition).

5. Dessinez l'arbre binaire correspondant à la lecture, avec ce code, de l'expression arithmétique suivante 15-(25+3)/6-54/(3-15/(23-6+4)-2)+10.

6. Ecrivez la version récursive de $lit\_expression$. Quelle est la difficulté rencontrée dans cette version et comment peut-elle être facilement résolue par une lecture différente du flot ?

7. Ecrivez la fonction qui évalue la valeur d'une expression arithmétique stockée dans un arbre binaire.

D'autres arbres équilibrés : les arbres 2-3-4

Les arbres 2-3-4 sont des arbres de recherche, mais contrairement aux arbres binaires, leurs noeuds contiennent 1, 2 ou 3 valeurs triées et ils ont (sauf les feuilles) un nombre de fils égal à ce nombre de valeurs +1. Le premier fils est un sous-arbre dont les valeurs sont inférieures à la première valeur du noeud, le second contient des valeurs comprises entre la première et la deuxième, le troisième entre la deuxième et la troisième et le quatrième fils contient des valeurs plus grandes que toutes les valeurs du noeud. D'autre part, un arbre 2-3-4 est équilibré, c'est-à-dire que toutes ses feuilles sont à la même profondeur.

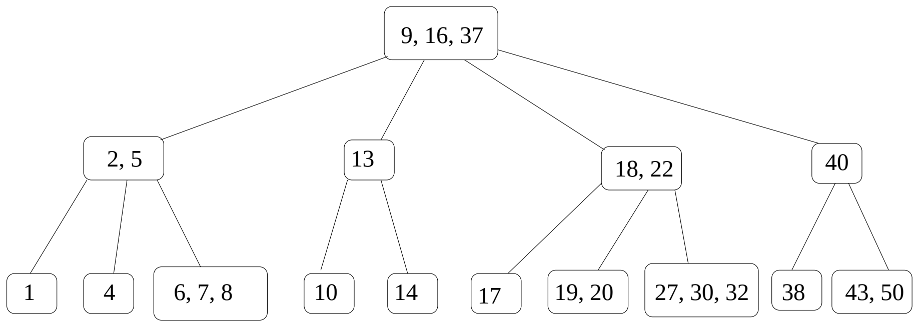


FIG. 1 – Exemple d'arbre 2-3-4

8. Proposez une structure de données permettant de construire un arbres 2-3-4.

Lorsqu'on veut ajouter un élément dans un arbre 2-3-4, on descend jusqu'à la feuille où l'élément doit aller. Si le nœud correspondant est plein, on l'éclate de la façon suivante, avant de réaliser l'insertion.

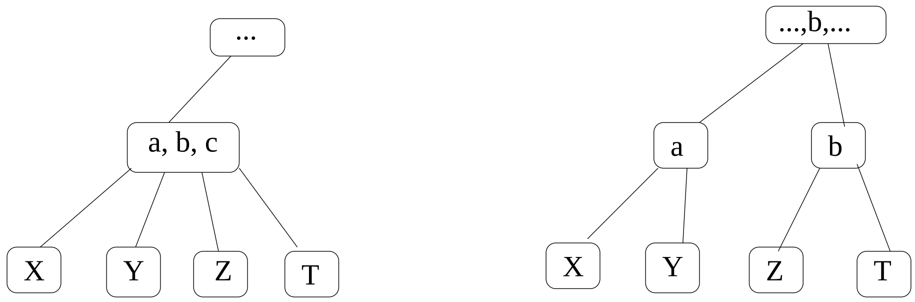


FIG. 2 – Eclatement d'un nœud contenant 3 informations : L'élément du milieu remonte dans le père. Si le père est plein, on l'éclate à son tour, etc...

9. Ajoutez les éléments 28 puis 29 aux feuilles de l'arbre dessiné plus haut, et dessinez l'état de l'arbre après chacune de ces insertions.

Une autre possibilité pour insérer un élément consiste, lorsqu'on descend dans l'arbre pour accéder à la feuille, à éclater préventivement tous les nœuds pleins (on dit un 4-nœud car il a 4 fils). C'est ce qu'on appelle l'éclatement à la descente.

10. Ajoutez les éléments 28 puis 29 à l'arbre dessiné plus haut en utilisant l'éclatement à la descente. Dessinez l'état de l'arbre après chacune de ces insertions.

11. Donnez le pseudocode de la procédure correspondant à l'insertion d'un élément dans un arbre 2-3-4.