INF3105 – Structures de données et algorithmes
Examen de mi-session – Hiver 2014

Éric Beaudry
Département d'informatique
Université du Québec à Montréal

Lundi 3 mars 2014 – 18h00 à 21h00 (3 heures) – Locaux PK-R650 et PK-1780

Instructions

- Aucune documentation n'est permise, excepté l'aide-mémoire C++ (une feuille recto-verso).
- Les appareils électroniques, incluant les téléphones et les calculatrices, sont strictement interdits.
- Répondez directement sur le questionnaire à l'intérieur des endroits appropriés.
- Pour les questions demandant l'écriture de code :
  - le fonctionnement correct, la robustesse, la clarté, l'efficacité (temps et mémoire) et la simplicité du code sont des critères de correction à considérer ;
  - vous pouvez scinder votre solution en plusieurs fonctions à condition de donner le code pour chacune d'elles ;
  - vous pouvez supposer l'existence de fonctions et de structures de données raisonnables ;
- Aucune question ne sera répondue durant l'examen. Si vous croyez qu'une erreur ou qu'une ambiguïté s'est glissée dans le questionnaire, indiquez clairement la supposition que vous avez retenue pour répondre à la question.
- L'examen dure 3 heures et contient 5 questions.
- Ne détachez pas les feuilles du questionnaire, à moins de les brocher à nouveau avant la remise.
- Le côté verso peut être utilisé comme brouillon. Des feuilles additionnelles peuvent être demandées au surveillant.
- À l'exception de la question 4a, vous devez répondre à l'aide d'un crayon d'une autre couleur que rouge.

Identification

Nom : Solutionnaire

Résultat

Q1		/ 5
Q2		/ 4
Q3		/ 9
Q4		/ 3
Q5		/ 4
Total		/ 25

1 Connaissances techniques et C++ (5 points)

Pour répondre à cette question, référez-vous au code fourni à l'Annexe A (page 8).

(a) Expliquez la différence entre «`numerateur(n)`» et «`dedominateur=d`» aux lignes 6 et 8 de `fraction.cpp`. (1 point)

Ligne 6 : il s'agit d'une **initialisation** de l'objet `numerateur` de type `int`. Cela équivaut à **construire** `numerateur` en lui copiant la valeur `n`;.
Ligne 8 : il s'agit d'une **affectation**. L'objet n'a pas été préalablement initialisé auparavant, car il n'y a pas d'initialisation pour les types de base.

(b) La ligne 17 de `prog1.cpp` est-elle un suspecte ? Si oui, expliquez pourquoi ? Sinon, écrivez simplement «Non». (1 point)

Oui. La ligne 17 accède à la position [5] de tab alors que ce tableau n'a qu'une taille de 5, c'est-à-dire des positions [0] à [4]. Le contenu de `tab[5]` étant indéterminé, on ne sait pas ce qui aura dans `f3` après la ligne 17.

(c) Ce programme se compile-t-il (sans erreur avec `g++ prog1.cpp fraction.cpp`) et s'exécute-t-il sans «plantage» ? Si oui, écrivez simplement «Oui». Sinon, (1) indiquez la nature du problème, (2) expliquez brièvement et (3) indiquez la(les) correction(s) minimale(s) et nécessaire(s). (1 point)

Oui.

(d) Qu'affiche ce programme ? Considérez les sauts de ligne et les modifications en (c) s'il y a lieu. (1 point)

```
F1/2 F2/3 F5/4 F0/1 F0/1 F0/1 F0/1 F0/1
K0/1 K0/1
K?/? K2/3 K1/2
```

où `?` est un nombre indéterminé (mémoire précédente).
En pratique, on pourrait probablement observer `K0/0` au lieu de `K?/?`.

(e) Ce programme a au moins une fuite de mémoire. Évaluez la quantité de mémoire (en octets) qui n'a pas été libérée correctement à la fin du programme. (1 point)

1. Le tableau pointé par `tab` n'est jamais libéré, donc `5 * sizeof(Fraction)`.
2. À chaque appel de la fonction `test`, `*c` n'est pas libéré, donc `5 * sizeof(Fraction)`.

Donc,`7 * sizeof(Fraction) = 7 * sizeof(<int, int>) = 7 * 2 * 4 = 7*8 = 56` octets.

2 Analyse algorithmique et optimisation (4 points)

Pour répondre à cette question, référez-vous au code fourni à l'Annexe B (page 9).

(a) Que retourne la fonction `question2` si on lui envoit la liste `l1`= $\langle 0,1,2,3,4,5,6,7,8,9\rangle$? (1 point)

Retourne une liste contenant : $\langle 0,2,4,6,8,1,3,5,7,9\rangle$.

(b) Que fait la fonction `question2` ? Votre réponse ne doit pas se résumer à donner un exemple. (1 point)

La fonction retourne une liste dans laquelle les nombres paires sont placés en premiers et les nombres impaires en derniers. L'ordre relative entre les nombres pairs est préservé. Idem pour les nombres impaires.

(c) Quelle est la complexité temporelle de la fonction `question2` ? Exprimez votre réponse en notation grand O. Considérez le pire cas et n est le nombre d'éléments dans la liste `l1`. Justifiez brièvement. (1 point)

$O(n^2)$.
Cas moyen (mixte pairs et impaires) et pire cas (uniquement pairs) : La boucle `for` de la ligne 2 itère n fois. Quand `e` est impair, l'insertion se fait à la fin en temps constant. Quand `e` est pair, on itère sur la liste `l2` tant qu'on rencontre des nombres pairs. Le nombre de nombres pairs est en moyenne $n/2$ et au pire de n.
Donc, en moyenne, $n \times \left(\frac{1}{2} \times 1 + \frac{1}{2} \times n/2\right) = n \times (\frac{1}{2} + \frac{n}{4}) = \frac{n}{2} + \frac{n^2}{4}$.
Cas moyen : $\frac{n}{2} + \frac{n^2}{4} \in O(n^2)$. Pire cas : $O(n^2)$.

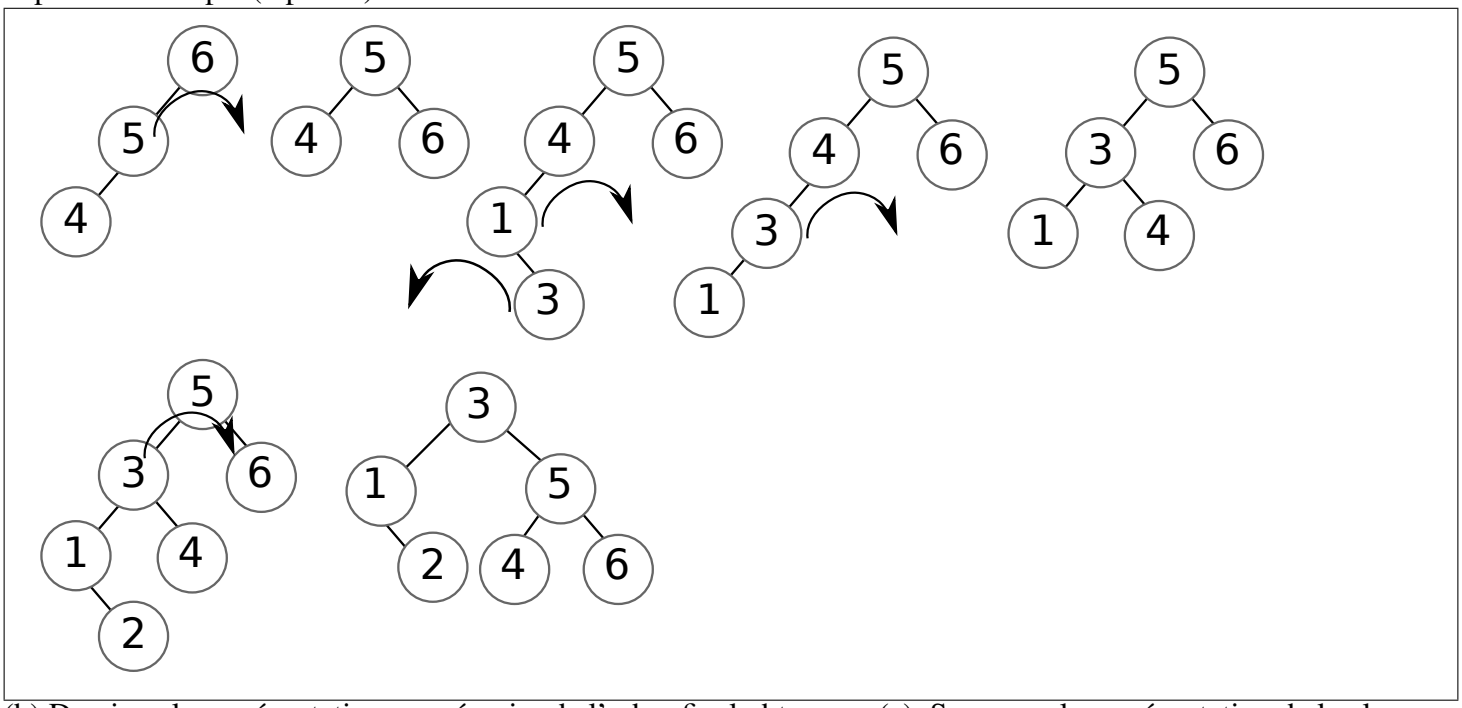
(d) Proposez une nouvelle fonction `question2` ayant une complexité temporelle moindre. Indiquez la complexité de cette nouvelle fonction. (1 point)

```
1  Liste<int> question2(const Liste<int>& l1){
2      Liste<int> l2;
3      Liste<int>::Iterateur i2 = l2.fin();
4      for(Liste<int>::Iterateur i1=l1.debut();i1;++i1){
5          int e = l1[i1];
6          if(e%2==0)
7              l2.inserer(i2, e);
8          else if(i2)
9                  l2.inserer(l2.fin(), e);
10             else{
11                 //assert(i2==l2.fin());
12                 i2 = l2.inserer(i2, e);
13             }
14     }
15     return l2;
16 }
```

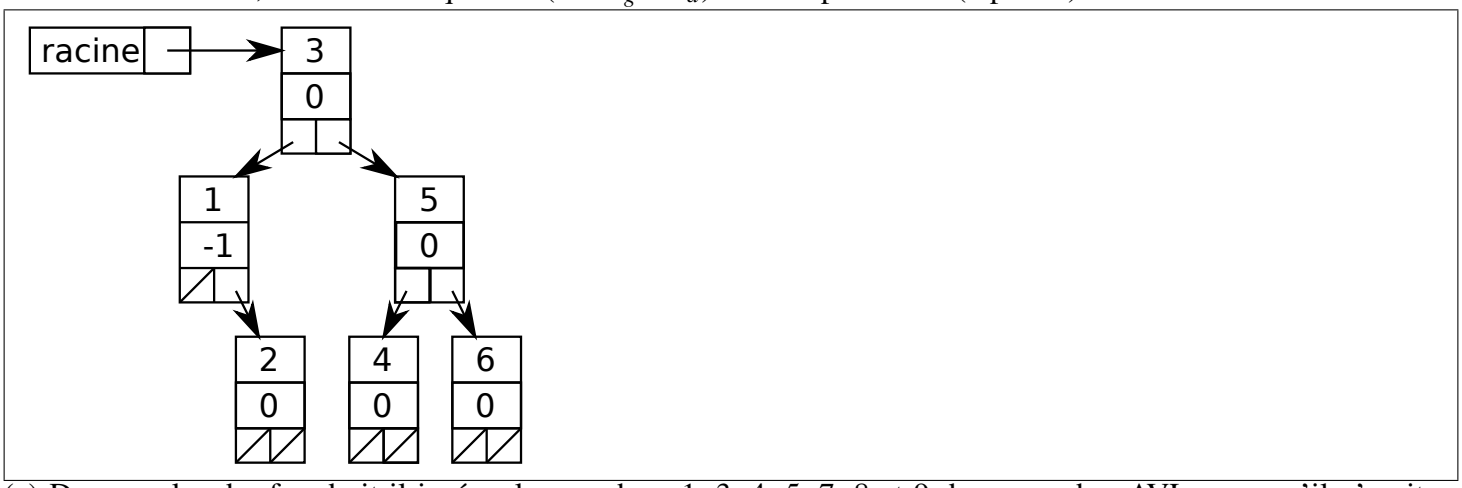
Complexité de $O(n)$.

3 Arbres AVL (9 points)

(a) Insérez les nombres 6, 5, 4, 1, 3 et 2 dans un arbre AVL initialement vide. À chaque insertion, ajoutez un nouveau nœud à l'arbre courant. Lorsqu'une rotation est requise : (1) dessinez une flèche pour montrer la rotation requise ; (2) dessinez un nouvel arbre pour montrer le résultat ; (3) le nouvel arbre devient l'arbre courant pour la prochaine étape. (2 points)

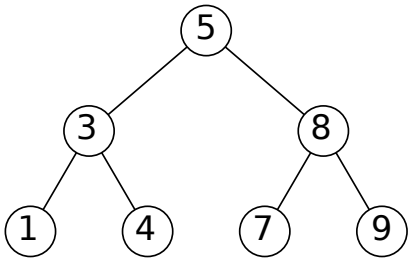


(b) Dessinez la représentation en mémoire de l'arbre final obtenu en (a). Supposez la représentation de la classe `ArbreAVL<T>` présentée en classe et dans les notes de cours. À titre de rappel, la structure `ArbreAVL<T>::Noeud` contient un élément, un indice d'équilibre ($e = h_g - h_d$) et deux pointeurs. (2 points)



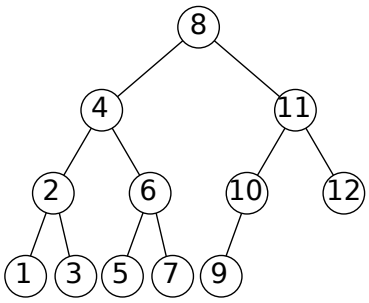
(c) Dans quel ordre fraudrait-il insérer les nombres 1, 3, 4, 5, 7, 8 et 9 dans un arbre AVL pour qu'il n'y ait aucune rotation nécessaire pour maintenir l'arbre équilibré ? Plusieurs réponses sont possibles. (1 point)

$\langle 5,3,8,1,4,7,9 \rangle$.

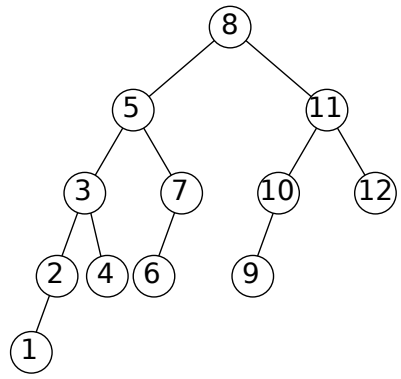


(d) Quelles sont la hauteur mimimale et la hauteur maximale d'un arbre AVL contenant 12 éléments. Illustrez chaque cas à l'aide d'un exemple. (1 point)

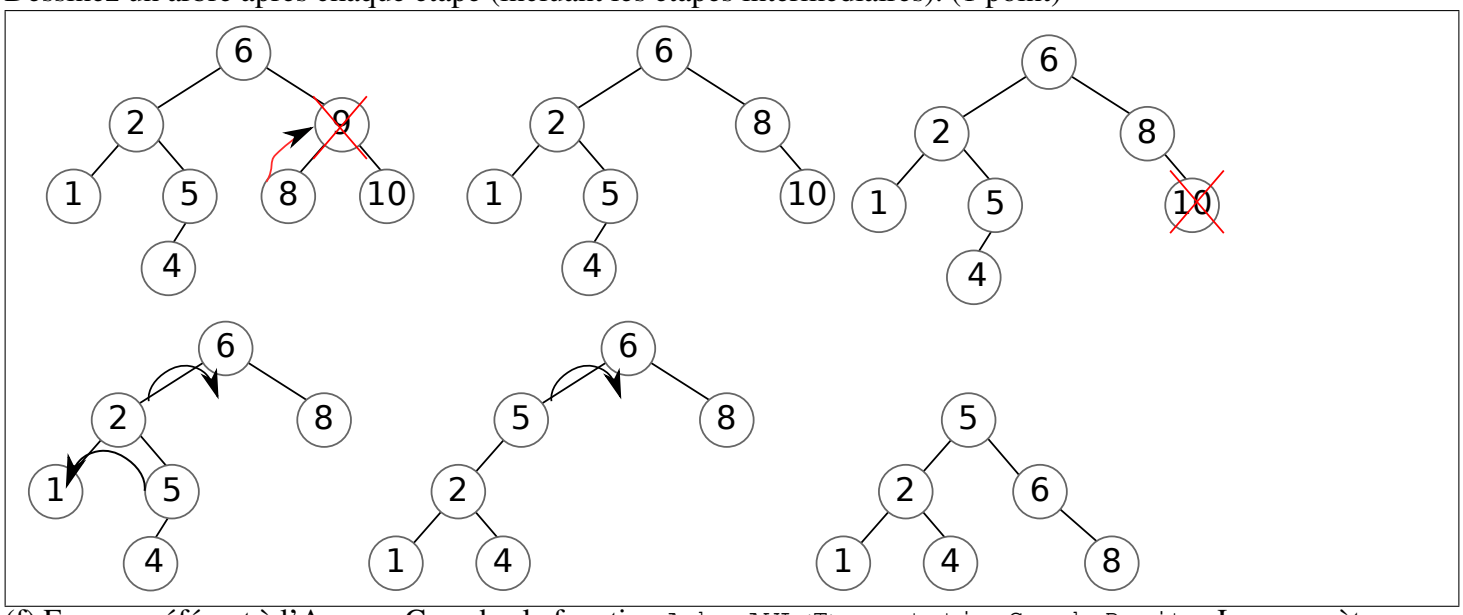
Hauteur minimale : 4



Hauteur maximale : 5



(e) Supprimez dans l'ordre les éléments 9 et 10 dans l'arbre AVL ci-dessous. Indiquez toutes les opérations. Dessinez un arbre après chaque étape (incluant les étapes intermédiaires). (1 point)



(f) En vous référant à l'Annexe C, codez la fonction `ArbreAVL<T>::rotationGaucheDroite`. Le paramètre `rs` est une référence sur la case mémoire contenant initialement un pointeur sur le noeud `b`. (2 points)

```
1  template <class T>
2  void ArbreAVL<T>::rotationGaucheDroite(Noeud*& rs)
3  {
4      Noeud *a = rs->gauche;
5      Noeud *b = rs;
6      int  ea = a->equilibre;
7      int  eb = b->equilibre;
8      int  ebp = - (ea > 0 ? ea : 0) - 1 + eb;
9      int  eap = ea + (ebp < 0 ? ebp : 0) - 1;
10     a->equilibre = eap;
11     b->equilibre = ebp;
12     b->gauche = a->droite;
13     a->droite = b;
14     rs = a;
15 }
```

4 Arbres rouge-noir (3 points)

(a) Expliquez en vos propres mots comment un arbre rouge-noir maintient un équilibre. Vous pouvez soutenir vos explications à l'aide d'un exemple. (1 point)

Dans un arbre rouge-noir, l'équilibre est maintenu à l'aide d'un concept de «profondeur noire», garantissant que toutes les sentinelles sont à égale «profondeur noire» de la racine. Comme il ne peut y avoir deux noeuds rouges de suite sur un chemin, cela procure un équilibre. En effet, la profondeur noire maximale est d'au plus du double de la profondeur noire minimale. Quand une insertion ou un enlèvement viole la contrainte de ne pas avoir deux rouges de suites, on réorganise les noeuds localement.

(b) En considérant le fragment de la classe `ArbreRN` ci-dessous, écrivez la fonction `ArbreRN<T>::hauteur()` qui retourne la hauteur de l'arbre. (2 points)

```
1 template <class T> class ArbreRN{
2     struct Noeud{       T contenu;
3                         Noeud *gauche, *droite;
4                         enum Couleur{rouge, noir} couleur;          };
5     Noeud* racine;
6     ... };
```

```
1 template <class T> int ArbreRN<T>::hauteur() const{
2     return hauteur(racine);
3 }
4 template <class T> int ArbreRN<T>::hauteur(const Noeud* n) const{
5     if(n==NULL) return 0;
6     return max(hauteur(n->gauche), hauteur(n->droite))+1;
7 }
```

Complexité : $O(n)$. Il n'y a pas moyen de faire mieux.
Complexité : $O(\log n)$.

5 Résolution d'un problème – L'examen le moins téléchargé (4 points)

Un professeur manquant de temps et d'inspiration pour rédiger un nouvel examen se résigne à réutiliser intégralement un examen d'une session précédente. Puisqu'il a déjà publié ces anciens examens dans son site web, le professeur désire choisir l'examen qui a été téléchargé par le moins d'étudiants. Pour choisir cet examen, il dispose du fichier journal de son serveur web *Apache* (`/var/log/apache2/access.log`). Un petit exemple est montré ci-bas. À chaque fois qu'un examen est téléchargé, une ligne est inscrite dans ce fichier. Chaque ligne est formée de quatre champs ordonnés : adresse IP, date, examen et navigateur. On suppose que chaque étudiant dispose d'une seule adresse IP unique. Chaque examen est téléchargé au moins une fois.

```
67.230.141.117  [01/Mar/2014:10:03:12] INF3105-2013A-ex1.pdf FireFox-Linux
67.230.141.117  [01/Mar/2014:10:03:13] INF3105-2013E-ex1.pdf FireFox-Linux
70.27.221.220   [01/Mar/2014:10:03:14] INF3105-2013A-ex1.pdf FireFox-Windows
67.230.141.117  [01/Mar/2014:10:04:13] INF3105-2013E-ex1.pdf Safari-iPhone
192.222.133.226 [01/Mar/2014:10:24:32] INF3105-2013H-ex1.pdf Safari-MacOS
192.222.133.226 [01/Mar/2014:10:24:43] INF3105-2013H-ex1.pdf Chrome-Android
78.224.123.240  [01/Mar/2014:10:25:33] INF3105-2013E-ex1.pdf IE-Windows
```

Vous devez écrire un programme qui affiche l'examen que le professeur devrait choisir et le nombre d'étudiants qui l'a téléchargé. Dans l'exemple ci-haut, le résultat attendu est «`INF3105-2013H-ex1.pdf 1`». Vous pouvez écrire votre solution directement dans le squelette suivant ou au verso de cette feuille. Utilisez les structures vues dans le cours. Indiquez la complexité temporelle en notation grand O, où n est le nombre de lignes dans le fichier journal, m le nombre d'étudiants et k le nombre d'examens antérieurs.

```
// Supposez tous les #include voulus et using namespace std.
int main(){
    ArbreMap<string, ArbreAVL<string> > exams; //exam->arbres contenant IP

    while(cin && !cin.eof()){
        string ip, date, exam, nav;
        cin >> ip >> date >> exam >> nav >> std::ws;
        exams[exam].inserer(ip);
    }
    string meilleur;
    int nbmin = std::numeric_limits<int>::max(); // 0x7fffffff
    for(ArbreMap<string, ArbreAVL<string> >::Iterateur iter=exams.debut();
        iter;++iter)
            if(iter.valeur().taille() < nbmin){
                nbmin = iter.valeur().taille();
                meilleur = iter.cle();
            }
    cout << meilleur << '\t' << nbmin << endl;
    return 0;
}
```

Complexité : $O(n(\log m + \log k))$.

Annexe A pour la Question 1

Cette page peut être détachée.

```
1  /* fraction.h */
2  class Fraction{
3  public:
4   Fraction(int n=0, int d=1);
5   ~Fraction();
6
7   Fraction& operator+=(const
      Fraction& f);
8   Fraction operator+(const
      Fraction& f)const;
9   bool operator<(const
      Fraction& f)const;
10
11 private:
12  int numerateur,
13      dedominateur;
14  void simplifier();
15 };
```

```
1  /* fraction.cpp */
2  #include "fraction.h"
3  #include <iostream>
4  using namespace std;
5  Fraction::Fraction(int n, int d)
6   : numerateur(n)
7  {
8    dedominateur = d;
9    cout << "F" << n << '/' << d << ' ';
10   simplifier();
11 }
12 Fraction::~Fraction(){
13   cout << "K"
14        << numerateur << '/'
15        << dedominateur << ' ';
16   numerateur = dedominateur = 0;
17 }
18 void Fraction::simplifier(){/*...*/}
```

```
1  /* prog1.cpp */
2  #include <iostream>
3  #include "fraction.h"
4  void test(Fraction a, Fraction& b){
5    Fraction* c = new Fraction(a);
6    a = b;
7    b = *c;
8    c = NULL;
9    return;
10 }
11 int main(){
12   Fraction f1(1,2), f2(2,3), f3(5,4);
13   Fraction* tab = new Fraction[5];
14   std::cout << std::endl;
15   test(f1, tab[2]);
16   test(tab[3], tab[4]);
17   f3 = tab[5];
18   std::cout << std::endl;
19   return 0;
20 }
```

Annexe B pour la Question 2

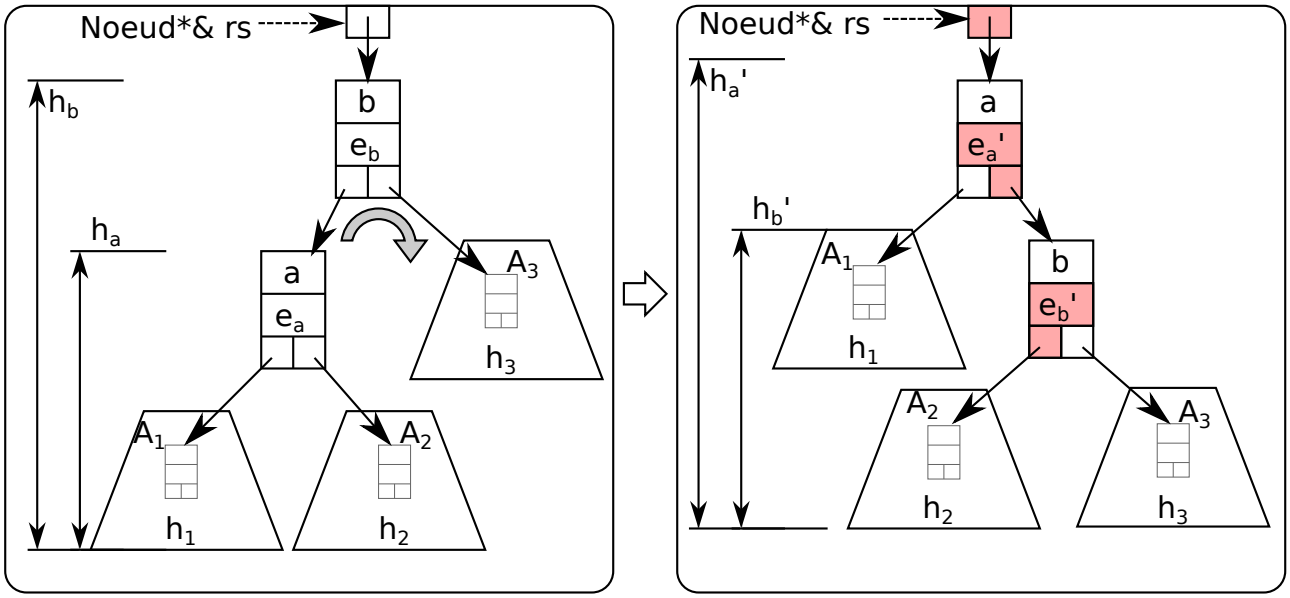
Cette page peut être détachée.

```
1  template <class T>
2  class Liste{
3    public:
4      Liste();
5      Liste(const Liste&);
6      ~Liste();
7      class Iterateur{
8        public:
9          // Pre-increment (passe au suivant). Complexite : O(1).
10         Iterateur& operator++();
11         // Retourne vrai ssi l'iterateur n'est pas rendu a la fin. Complexite : O(1).
12         operator bool() const;
13       private:
14         ...
15     };
16     // Insere e devant la position pos. Complexite : O(1).
17     // Retourne la position du nouvel element insere dans la liste.
18     Iterateur inserer(Iterateur& pos, const T& e);
19     // Vide la liste. Complexite O(n).
20     void vider();
21     // Retourne un iterateur sur le debut de la liste. Complexite : O(1).
22     Iterateur debut() const;
23     // Retourne un iterateur sur la fin de la liste. Complexite : O(1).
24     // La fin n'est pas le dernier element, mais la position qui suit la derniere
       position de la liste.
25     Iterateur fin() const;
26     // Retourne l'element a la position p. Complexite : O(1).
27     const T& operator[](const Iterateur& p) const;
28   private:
29     ...
30 };
```

```
1  Liste<int> question2(const Liste<int>& l1){
2      Liste<int> l2;
3      for(Liste<int>::Iterateur i1=l1.debut();i1;++i1){
4          int e = l1[i1];
5          Liste<int>::Iterateur i2 = l2.fin();
6          if(e%2==0){ // % est l'operateur modulo
7              i2 = l2.debut();
8              while(i2 && l2[i2]%2==0) ++i2;
9          }
10         l2.inserer(i2, e);
11     }
12     return l2;
13 }
```

Annexe C pour la Question 3f

Cette page peut être détachée.



Les équations (1) à (9) montrent comment obtenir le nouvel indice e'_b à partir des valeurs de e_a et e_b.

$$e'_b = h_2 - h_3 \quad \text{Par définition.} \quad (1)$$
$$h_3 = h_a - e_b \quad \text{Par définition.} \quad (2)$$
$$h_a = \max(h_1, h_2) + 1 \quad \text{Par définition.} \quad (3)$$
$$h_1 = h_2 + e_a \quad \text{Par définition.} \quad (4)$$
$$h_a = \max((h_2 + e_a), h_2) + 1 \quad (3),(4) \quad (5)$$
$$= h_2 + \max(e_a, 0) + 1 \quad (5) \quad (6)$$
$$e'_b = h_2 - (h_a - e_b) \quad (1)\ (2) \quad (7)$$
$$= h_2 - ((h_2 + \max(e_a, 0) + 1) - e_b) \quad (7) \quad (8)$$
$$= -max(e_a, 0) - 1 + e_b \quad (8) \quad (9)$$

Les équations (10) à (18) montrent comment obtenir le nouvel indice e'_a à partir des valeurs de e_a, e_b et e'_b.

$$e'_a = h_1 - h'_b \quad \text{Par définition.} \quad (10)$$
$$h'_b = \max(h_2, h_3) + 1 \quad \text{Par définition.} \quad (11)$$
$$h_3 = h_2 - e'_b \quad \text{Par définition.} \quad (12)$$
$$h'_b = \max(h_2, (h_2 - e'_b)) + 1 \quad (11), (12) \quad (13)$$
$$= h_2 + \max(0, -e'_b) + 1 \quad (13) \quad (14)$$
$$h_1 = h_2 + e_a \quad \text{Par définition.} \quad (15)$$
$$e'_a = (h_2 + e_a) - (h_2 + \max(0, -eb') - 1) \quad (10), (15), (14) \quad (16)$$
$$= e_a - \max(0, -eb') - 1 \quad (16) \quad (17)$$
$$= e_a + \min(0, eb') - 1 \quad (17) \quad (18)$$