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

Éric Beaudry
Département d'informatique
Université du Québec à Montréal

Lundi 23 juin 2014 – 17h30 à 20h30 (3 heures) – Local PK-R610

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 4, vous devez répondre à l'aide d'un crayon d'une autre couleur que rouge.

Identification

Nom : Solutionnaire

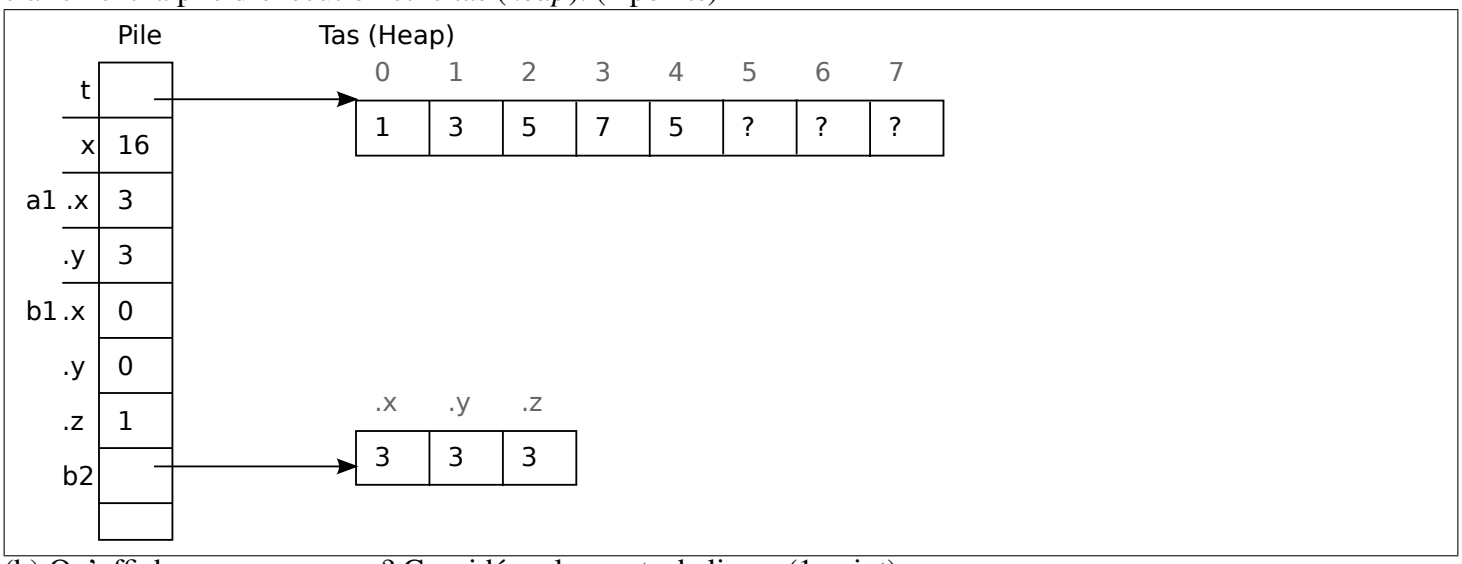
Résultat

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). À titre d'information, ce programme se compile sans erreur.

(a) Dessiner la représentation en mémoire de ce programme rendu à la ligne 22 de `question1.cpp`. Montrez clairement la pile d'exécution et le tas (*heap*). (2 points)



(b) Qu'affiche ce programme ? Considérez les sauts de ligne. (1 point)

```
 1 3 5 7 5 ? ? ? : 16
A(int) A() B() A(int) B(int)
~B() 1 ~A() 0,0 ~A() 3,3
```

où les

(c) Ce programme libère-t-il la mémoire correctement ? Si non : (1) évaluez la quantité de mémoire (en octets) qui n'a pas été libérée correctement à la fin du programme ; (2) quelle(s) ligne(s) devrait-on ajouter après la ligne 22 pour libérer la mémoire correctement ? (1 point)

Non. Deux blocs de mémoire sont alloués par `new` mais jamais libérés par `delete`. Le premier bloc alloué par `new int[8]` alloue `8*sizeof(int)` = 32 octets. Le deuxième bloc alloué par `new B(3)` alloue `sizeof(B)=3*sizeof(int)` = 12 octets. Donc, **44 octets** au total.
Pour libérer la mémoire correctement : `delete[] t; delete b2`.

(d) Dans la fonction `main` dans `question1.cpp`, la variable `i` est déclarée 3 fois. Pourtant, un compilateur C++ relativement récent ne signale aucune erreur à la compilation. Pourquoi ? (1 point)

Les 3 variables `i` sont dans des portées (*scope*) différentes. Ils ne sont donc pas en conflit l'un avec un autre.

2 Analyse et modification d'un algorithme (4 points)

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

(a) Que fait la fonction `q2` ? Expliquez dans vos propres mots. Soyez aussi précis que possible. (1 point)

La fonction `q2` retourne vrai si et seulement si il existe un sous-ensemble de nombres dans le tableau `t` dont la somme égale `x`.

(b) Quelle est la complexité temporelle de la fonction `q2` ? Exprimez votre réponse en notation grand O et utilisez n comme étant la taille du tableau `t`. (1 point)

Exprimons le nombre d'opérations à l'aide de la fonction $f(i)$.
Si $i = n$ alors $f(i) = 1$.
Si $i < n$ alors $f(i) = 2 \cdot (n-i) \cdot f(i+1) + 1$
$f(0) = 2 \cdot n \cdot f(1) + 1$
$f(0) = 2n(2(n-1)f(2)+1)+1$
$f(0) = 2n(2(n-1)(2(n-2)f(3)+1)+1)+1$
$f(0) = 2n(2(n-1)(2(n-2)(...(1))+1)+1)+1$
$f(0) \approx 2 \cdot 2 \cdot 2...n(n-1)(n-2)(n-3)...(1)$
$f(0) \approx 2^n \cdot n!$
$f(0) \in O(2^n \cdot n!)$

(c) Adaptez la fonction `q2` afin qu'elle reçoive une liste chaînée plutôt qu'un tableau. Votre solution devrait être en deux fonctions, car le paramètre `i` n'est plus un `int` et ne peut avoir une valeur par défaut. (1 point)

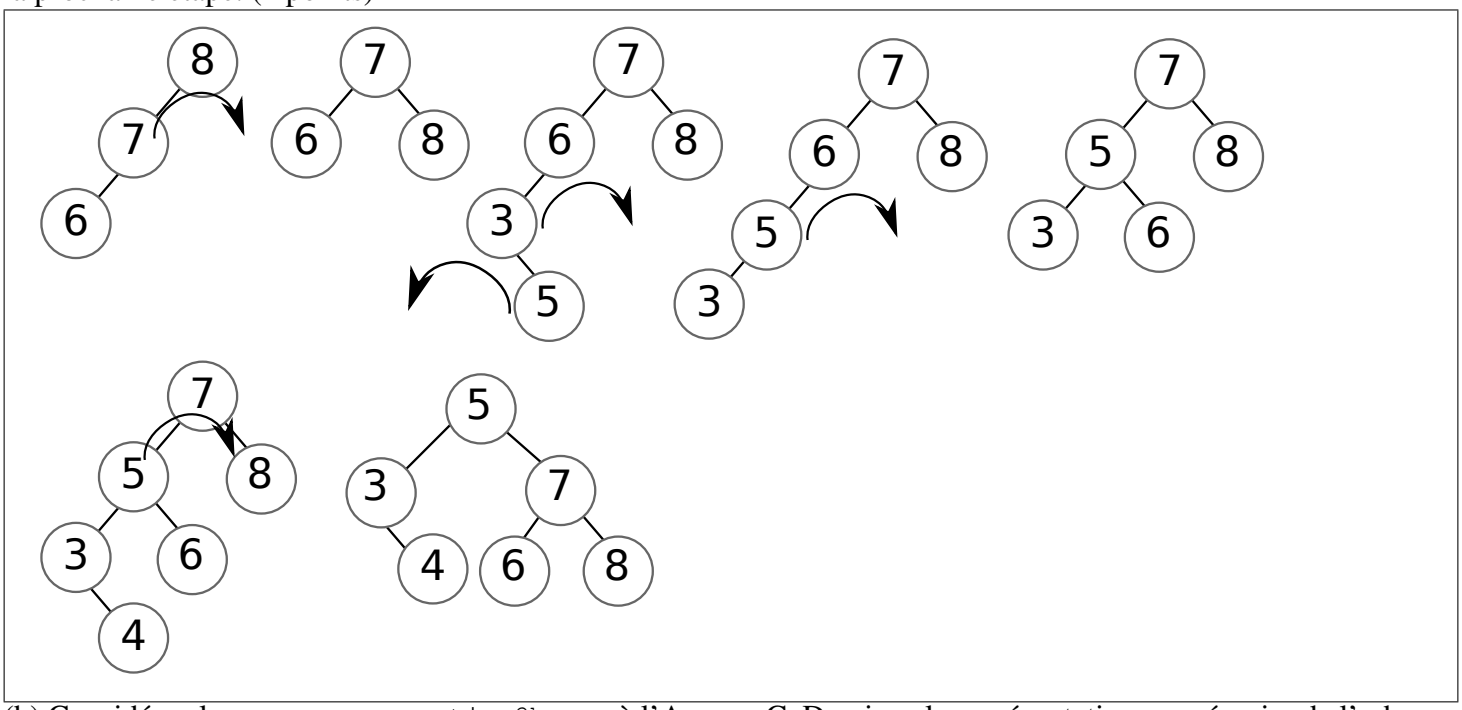
```
1  bool q2(int x, const Liste<int>& l){
2      return q2(x, l, l.debut());
3  }
4  bool q2(int x, const Liste<int>& t, Liste<int>::Iterateur i, int s=0){
5      if(x==s) return true;
6      for(;i;++i){
7          if(q2(x, l, i+1, s)) return true;
8          if(q2(x, l, i+1, s+(*i))) {
9              cout << *i << ' ';
10             return true;
11         }
12     }
13     return false;
14 }
```

(d) Selon vous, y aurait-il un avantage (efficacité, simplicité, etc.) à utiliser une autre structure de données que le `Tableau` ? Si oui, laquelle ? Justifiez votre réponse. (1 point)

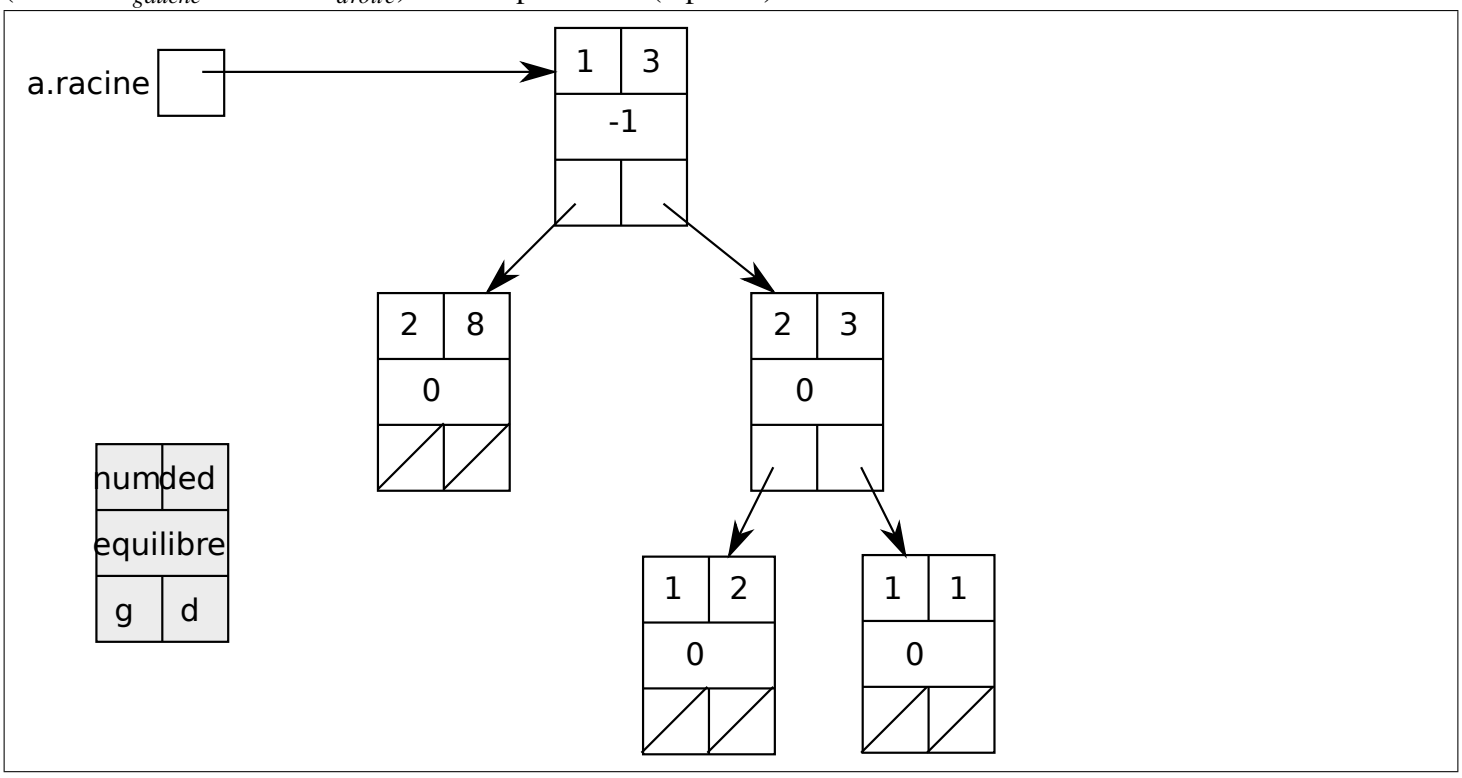
Aucun avantage. Ici, toutes les opérations d'accès au tableau (ou à la liste) se font en temps constant.

3 Arbres AVL (9 points)

(a) Insérez les nombres 8, 7, 6, 3, 5 et 4 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) Considérez le programme `question3b.cpp` à l'Annexe C. Dessinez la représentation en mémoire de l'arbre `a` obtenu à la ligne 24. 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 (=$hauteur_{gauche} - hauteur_{droite}$) et deux pointeurs. (2 points)



(c) Écrivez le code d'une fonction `ArbreAVL<T>::profmin` qui retourne la profondeur de la feuille la moins profonde d'un arbre AVL. (2 points)

```
1  template <class T> int ArbreAVL<T>::profmin() const {
2    if(racine==NULL) return -1; // ou assert(false) arbre vide=aucune feuille
3    return profmin(racine, 0);
4  }
5  template <class T> int ArbreAVL<T>::profmin(const Noeud* n, int prof) const {
6    if(n->gauche==NULL && n->droite==NULL) return prof; // si une feuille
7    if(n->gauche==NULL) profmin(n->droite, prof+1); // un seul enfant a droite
8    if(n->droite==NULL) profmin(n->gauche, prof+1); // un seul enfant a gauche
9    return min(profmin(n->gauche,prof+1), profmin(n->droite,prof+1));//min des 2 enfants
10 }
```

(d) Quelle est la complexité temporelle de votre fonction en (c) ? (1 point)

$O(n)$
L'algorithme visite tous les noeuds pour se rendre à toutes les feuilles. Comme il n'est pas possible de prédire de quel côté sera la feuille la moins profonde avec l'indique d'équilibre, il n'est pas possible de faire mieux que $O(n)$.

(e) Écrivez une fonction `ArbreAVL<T>::nbNoeuds(const T& min, const T& max)` qui retourne le nombre de noeuds dans l'arbres dont le contenu est entre `min` et `max` (inclusivement). (2 points)

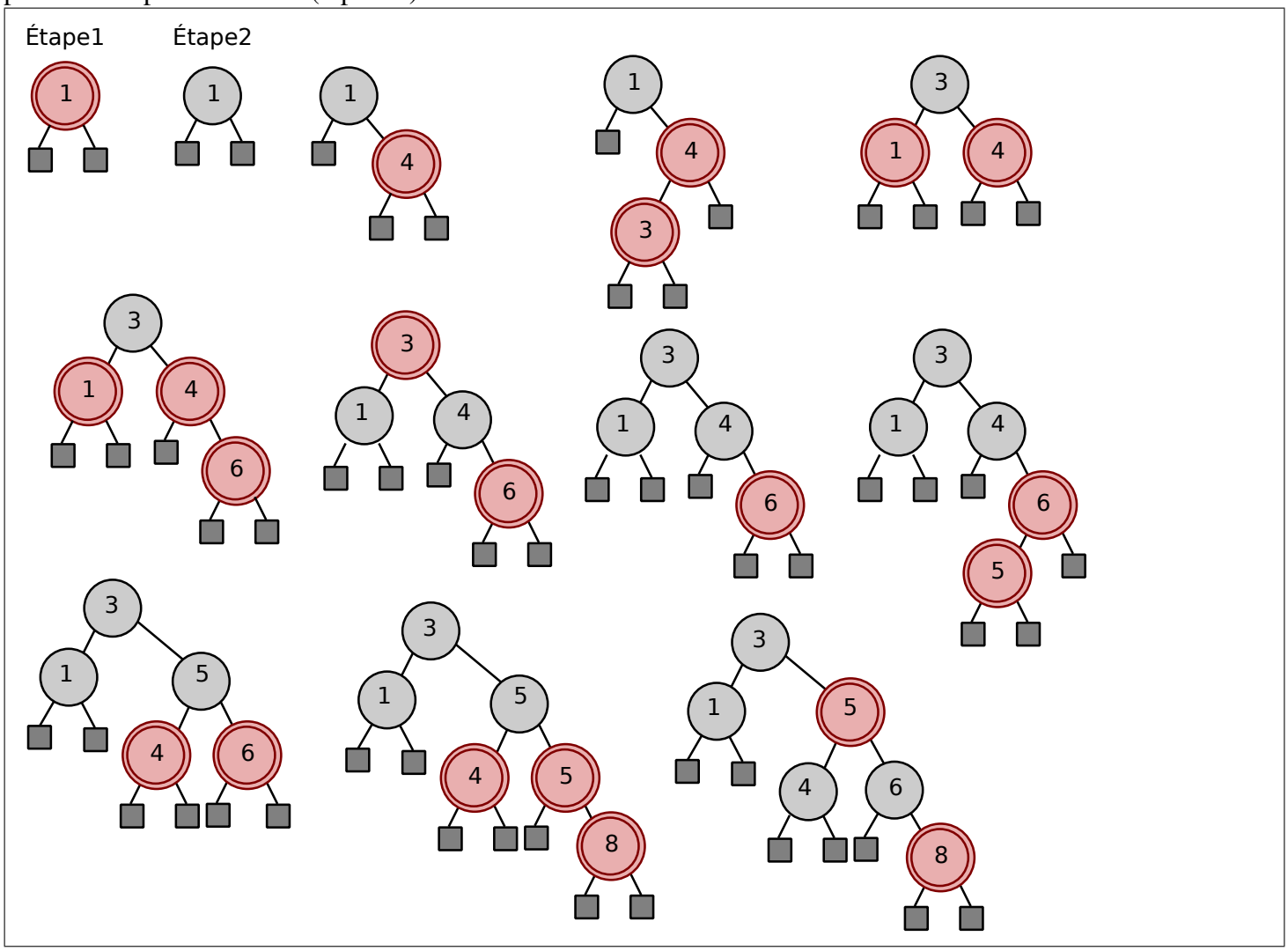
```
1  template <class T> int ArbreAVL<T>::nbNoeuds(const T& min, const T& max) const {
2    return nbNoeuds(racine, min, max);
3  }
4  template <class T> int ArbreAVL<T>::nbNoeuds(const Noeud* n, const T& min, const
       T& max) const {
5    if(n==NULL) return 0;
6    int nb = 0;
7    if(min <= n->contenu && n->contenu <= max) nb++;
8    if(min <= n->contenu)
9      nb += nbNoeuds(n->gauche, min, max);
10   if(max >= n->contenu) // n->contenu <= max
11     nb += nbNoeuds(n->droite, min, max);
12   return nb;
13 }
```

4 Arbres rouge-noir (3 points)

À titre de rappel, revoici les principales caractéristiques d'un arbre rouge-noir :

- la racine est noire ;
- le nœud parent d'un nœud rouge doit être noir (on ne peut pas avoir deux nœuds rouges de suite) ;
- les feuilles sont appelées « sentinelles », ne stockent aucun élément et sont des nœuds noirs ;
- toutes les sentinelles sont à une « profondeur noire » égale, la profondeur noire étant définie par le nombre de nœuds noirs sur le chemin (à ne pas confondre avec la hauteur de l'arbre qui elle ne tient pas compte des sentinelles).

Insérez les nombres 1, 4, 3, 6, 5 et 8 dans un arbre rouge-noir initialement vide. Lorsqu'une réorganisation/re-coloration de nœud(s) est requise, redessinez l'arbre afin de bien montrer les étapes intermédiaires. Considérez les directives suivantes : (1) si vous n'avez pas de crayon rouge, dessinez un cercle simple pour un nœud noir et un cercle double pour un nœud rouge ; (2) dessinez de petits carrés pleins pour représenter les sentinelles. La première étape est donnée. (3 points)



5 Résolution d'un problème – Analyse de statistiques sportives (4 points)

Pour aider les commentateurs sportifs, vous devez écrire un programme analysant les séquences de victoires et de défaites des équipes de la Ligne nationale de Hockey. Le programme reçoit en entrée des résultats de matchs en ordre chronologique. Il n'y a jamais de match nul. Chaque ligne contient un résultat et est formatée ainsi : date, noms et nombres de buts des équipes locale et visiteuse. Voici un exemple.

```
2013-10-01 Toronto 4 Montreal 3
2013-10-03 Toronto 3 Philadelphie 1
2013-10-04 Philadelphie 1 Montreal 4
2013-10-08 Philadelphie 1 Caroline 2
2013-10-09 Calgary 3 Montreal 2
2013-10-10 Toronto 4 Nashville 0
```

Votre programme doit afficher le nom des équipes dont sa plus longue séquence de victoires est plus longue ou égale à sa plus longue séquence de défaites. Un squelette et un tableau sont fournis à l'Annexe D (page 10). Dans l'exemple précédent, le résultat affiché devrait être : `Calgary Caroline Montreal Toronto`. Si vous manquez d'espace, utilisez le verso de cette feuille.

```
class Equipe{
  int seqvic, seqdef, maxseqvic, maxseqdef;
public:
  Equipe() : seqvic(0), seqdef(0), maxseqvic(0),maxseqdef(0){}
  void victoire(){
    seqdef = 0;
    if(++seqvic>maxseqvic) maxseqvic = seqvic;
  }
  void defaite(){
    seqvic = 0;
    if(++seqdef>maxseqdef)maxseqdef = seqdef;
  }
  bool seqvicpluslongue() const{ return maxseqvic>=maxseqdef; }
};
int main(){
  map<string, Equipe> equipes;
  while(cin && !cin.fail()){
    string date, l, v;
    int bl, bv;
    cin >> date >> l >> bl >> v >> bv >> std::ws;
    if(!cin) break;
    if(bl>bv){equipes[l].victoire(); equipes[v].defaite();}
    else if(bv>bl) {equipes[v].victoire(); equipes[l].defaite();}
  }
  for(map<string,Equipe>::iterator i=equipes.begin();i!=equipes.end();++i)
    if(i->second.seqvicpluslongue())
      cout << i->first << " ";
  cout << endl;
}
```

Annexe A pour la Question 1

Cette page peut être détachée.

```
1  /* a.h */
2  #include <iostream>
3  using namespace std;
4  class A{
5      int x, y;
6    public:
7      A();
8      A(int m);
9      ~A();
10 };
11 class B : public A{
12     int z;
13   public:
14     B();
15     B(int k);
16 };
```

```
1  /* a.cpp */
2  #include "a.h"
3  A::A() : x(0), y(0) {
4    cout << "A() ";
5  }
6  A::A(int m) : x(m),y(m) {
7    cout << "A(int) ";
8  }
9  A::~A(){
10   cout << "~A() " << x << "," << y << " ";
11 }
12 B::B() : z(1) {
13   cout << "B() ";
14 }
15 B::B(int m) : A(m), z(m) {
16   cout << "B(int) ";
17 }
18 B::~B(){
19   cout << "~B() " << z << " ";
20 }
```

```
1  /* question1.cpp */
2  #include <iostream>
3  #include "a.h"
4  void f1(int* tab, int n){
5    int* t = tab, *f=tab+n;
6    while(t<f) *(t++) *= 2;
7  }
8  void f2(int& a, int& b){
9    a += b++;
10 }
11 int main(){
12   int* t = new int[8];
13   for(int i=0;i<5;i++) t[i]=i;
14   f1(t, 4);
15   int x = 0;
16   for(int i=0;i<5;i++)  f2(x, t[i]);
17   for(int i=0;i<8;i++)  std::cout << " " << t[i];
18   std::cout << " : " << x << std::endl;
19   A a1(3);
20   B b1;
21   B* b2 = new B(3);
22   std::cout << std::endl;
23 }
```

Annexe B pour la Question 2

Cette page peut être détachée.

```
1  #include <iostream>
2  #include <tableau.h>
3  #include <cstdio>
4  using namespace std;
5
6  bool q2(int x, const Tableau<int>& t, int i=0, int s=0){
7      if(x==s) return true;
8      for(;i<t.taille();i++){
9          if(q2(x, t, i+1, s)) return true;
10         if(q2(x, t, i+1, s+t[i])){
11             cout << t[i] << ' ';
12             return true;
13         }
14     }
15     return false;
16 }
17 int main(int argc, const char** argv){
18     int x = 0;
19     cout << "x: ";
20     cin >> x;
21     cout << "Entrez des nombres (zero pour arreter):";
22     Tableau<int> tab;
23     while(true){
24         int nombre=0;
25         cin >> nombre;
26         if(nombre==0) break;
27         tab.ajouter(nombre);
28     }
29     q2(x, tab);
30     return 0;
31 }
```

Annexe C pour la Question 3

Cette page peut être détachée.

```
1  /* question3b.cpp */
2  class Fraction{
3    public:
4      Fraction(int n=0, int d=1);
5      bool operator<(const Fraction& f) const;
6    private:
7      int numerateur, dedominateur;
8  };
9  Fraction::Fraction(int n=0, int d=1)
10  : numerateur(n), dedominateur(d)
11 {}
12 bool Fraction::operator<(const Fraction& f)const{
13   return numerateur * f.dedominateur < f.numerateur * dedominateur;
14 }
15 #include <arbreavl.h>
16 int main(){
17   ArbreAVL<Fraction> a;
18   a.inserer(Fraction(1,4));
19   a.inserer(Fraction(1,3));
20   a.inserer(Fraction(1,2));
21   a.inserer(Fraction(1));
22   a.inserer(Fraction(2,3));
23   a.inserer(Fraction(2,8));
24   /* ICI */
25   return 0;
26 }
```

Annexe D pour la Question 5

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4    while(cin && !cin.eof()){
5      string date, l, v; // date, equipe local
6      int    bl, bv; // nb buts equipe local et equipe visiteuse
7      cin >> date >> l >> bl >> v >> bv >> std::ws;
8    }
9    return 0;
10 }
```

Équipe	Calgary	Caroline	Montreal	Nashville	Philadelphie	Toronto
Plus long. séq. victoires	1	1	1	0	0	3
Plus long. séq. défaites	0	0	1	1	3	0