

# INF3105 – Structures de données et algorithmes

## Automne 2014 – Examen de mi-session

Éric Beaudry  
Département d'informatique  
Université du Québec à Montréal

Mardi 21 octobre 2014 – 13h30 à 16h30 (3 heures) – Locaux SB-R440 et A-2770

### Instructions

- Aucune documentation n'est permise, excepté l'aide-mémoire C++ (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 ;
  - 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, contient 5 questions et vaut 20 % de la session.
- Ne détachez pas les feuilles du questionnaire, à moins de les brocher à nouveau avant la remise.
- Dans l'entête de l'actuelle page, si vous encerclez le numéro de local où vous vous trouvez présentement, vous aurez un point boni pour avoir lu les instructions.
- Le côté verso peut être utilisé comme brouillon. Des feuilles additionnelles peuvent être demandées au surveillant.

### Identification

Nom : \_\_\_\_\_

Code permanent : \_\_\_\_\_

Signature : \_\_\_\_\_

### Résultat

Q1		/ 20
Q2		/ 20
Q3		/ 28
Q4		/ 12
Q5		/ 20
Total		/ 100

# 1 Connaissances techniques et C++ [20 points]

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

(a) Que fait la ligne 10 du programme `question1a.cpp` (`int* t = new int[8];`) ? [3 points]

(b) Dessinez la représentation en mémoire de ce programme après avoir exécuté la ligne 3 de la fonction `f1`. Montrez les objets des fonctions `main` et `f1` sur la pile (*stack*) et ceux sur le tas (*heap*). [5 points]

(c) Qu'affiche le programme `question1a.cpp` ? [3 points]

(d) Ce programme libère-t-il la mémoire correctement ? Si oui, dites simplement oui. Si non, écrivez la modification minimale pour corriger ce problème. [3 points]

(e) Alice et Bob ont respectivement écrit deux programmes `q1e-alice.cpp` et `q1e-bob.cpp`. Ces 2 programmes **affichent**-ils le même résultat ? Expliquez brièvement deux différences entre ces deux programmes. [3 points]

(f) Alice et Bob sont convaincus d'avoir une meilleure solution que l'autre. Selon vous, qui a raison ? Justifiez. [3 points]

## 2 Complexité algorithmique [20 points]

(a) Simplifiez les ordres de grandeur suivants (en notation grand O). [8 points]

$O(2)$		$O(4n \log n + 8m \log n)$	
$O(2n \times n/8)$		$O((4n + 3) \times (12n - 7))$	
$O(2^n + n^3 + 9n^2)$		$O(4n + 8n \log(7n) + 75)$	
$O((42n^2 + n + 8)(n + 8))$		$O(n! + 8n^3 + n^2)$	

Pour les sous-questions suivantes, référez-vous au code fourni à l'Annexe B (page 9).

(b) Que fait le programme à l'Annexe B (q2.cpp) ? [4 points]

(c) Quelle est la complexité temporelle de la fonction q2 notation grand O ? Justifiez brièvement. [4 points]

(d) Il est possible d'améliorer la fonction q2 pour réduire sa complexité temporelle. Expliquez (en français) comment améliorer la fonction q2 **OU** codez (C++ ou pseudo-code) une version améliorée. Écrivez aussi la complexité temporelle de cette nouvelle version. [4 points]

### 3 Arbres AVL [28 points]

(a) Insérez les nombres 3, 4, 2, 6, 5 et 7 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. [8 points]

(b) [6 points] L'insertion dans un arbre AVL provoque au maximum une rotation simple ou double. Un enlèvement peut provoquer une cascade de rotations simples ou doubles jusqu'à la racine. La présente sous-question vise à montrer un tel exemple. Étapes :

1. Dans le rectangle à gauche, dessinez un arbre AVL équilibré qui nécessite au moins deux rotations (simples et/ou doubles) suite à l'enlèvement d'un élément précis. Mettez des nombres dans les nœuds.
2. Indiquez clairement quel élément provoque une cascade de rotations lors de son enlèvement.
3. Montrez le résultat final de l'enlèvement dans le rectangle à droite. Indiquez clairement les rotations.

(c) **Sans utiliser d'itérateurs** d'arbre AVL, codez une fonction qui retourne une liste chaînée en **désordre** (du plus grand au plus petit) les éléments contenus dans un arbre. Il existe une solution en 10 à 11 lignes de code. [4 points]

```
1 template <class T> Liste<T> ArbreAVL<T>::convertirEnListeInv() const{
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

//

(d) On insère les nombres 1 à 100 dans un Arbre AVL dans un ordre aléatoire. Le nombre 25 peut-il être à la racine ? Répondez simplement par oui ou non. [2 points]

(e) La présente question est liée à la sous-question précédente (d). Quels nombres peuvent se retrouver à la racine d'un arbre AVL contenant les nombres 1 à 100 ? Justifiez brièvement en utilisant l'espace alloué. [4 points]

(f) Combien d'octets sont nécessaires pour stocker en mémoire un arbre AVL dans lequel on a inséré les nombres 1 à 100 ? Supposez `sizeof(int)=4` et `sizeof(X*)=4`. Supposez que l'arbre AVL est implémenté tel que présenté en classe et dans les notes de cours. Justifiez votre réponse. [4 points]

## 4 Arbres rouge-noir [12 points]

(a) Proposez une **représentation** pour implémenter un arbre rouge-noir. Ajoutez le code directement dans le squelette ci-dessous. Votre représentation doit être suffisante pour implémenter toutes les opérations usuelles d'un arbre binaire de recherche, soit la recherche, l'insertion et l'enlèvement. Un rappel des principales caractéristiques d'un arbre rouge-noir est disponible à l'Annexe C (page 9). [4 points]

```

1  template <class T> class ArbreRN {
2
3      struct Noeud{
4
5
6
7
8
9      };
10
11
12      ...
13  public:
14      ArbreRN();
15      ...
16      int fonctionB() const;
17  };

```

(b) Complétez la solution de la fonction `fonctionB` qui retourne le nombre maximal de nœuds pouvant être insérés dans l'arbre sans provoquer une réorganisation ou un recoloriage de nœuds existants. [4 points]

```

1  template <class T> int ArbreRN<T>::fonctionB() const{
2      return fonctionB(racine); }
3  template <class T> int ArbreRN<T>::fonctionB(const Noeud* n) const{
4
5
6
7
8
9
10
11
12
13
14

```

(c) Quel est le nombre minimal de nœuds dans un arbre rouge-noir de hauteur 8 ? Justifiez votre réponse. Vous pouvez donner le nombre exact [4 points] ou une approximation [3 points].

Nombre exact : \_\_\_\_\_ ou Approximation : \_\_\_\_\_  
Justification :

## 5 Résolution d'un problème – Analyse des résultats (20 points)

On souhaite vérifier si les étudiants obtiennent des notes comparables dans les cours INF3105 et INF4230. Vous trouverez à l'Annexe D (page 10) deux fichiers de notes `INF3105.txt` et `INF4230.txt` en exemple. Chaque ligne contient le code permanent d'un étudiant et sa note obtenue. Écrivez un programme qui produit la matrice d'analyse décrite à l'Annexe D. Les étudiants n'ayant suivi qu'un seul des deux cours doivent être ignorés. Votre solution doit utiliser au moins un dictionnaire basé sur un arbre binaire de recherche (`ArbreMap` ou `map`).

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42

//

## Annexe A pour la Question 1

Cette page peut être détachée. À noter que le code a été allégé (ex. : `#include`) pour rentrer sur une page.

```

1  /* question1a.cpp */
2  void f1(int* tab, int n){
3      int* t = tab, *f=tab+n;
4      while(t<f) *(t++) *= 2;
5  }
6  void f2(int& a, int& b){
7      a += b++;
8  }
9  int main(){
10     int* t = new int[8];
11     for(int i=0;i<5;i++)
12         t[i]=i;
13     f1(t, 4);
14     int x = 0;
15     for(int i=0;i<5;i++)
16         f2(x, t[i]);
17     for(int i=0;i<8;i++)
18         cout << " " << t[i];
19     cout<<" : "<<x<<endl;
20     return 0;
21 }
```

```

1  /* pile.h */
2  template <class T>
3  class Pile //Tout est correctement implemente
4  {
5  public:
6      Pile();
7      Pile(const Pile&);
8      ~Pile();
9      void empiler(const T&);
10     T depiler();
11     bool vide() const;
12     const Pile<T>& operator=(const Pile<T>&);
13
14 private:
15     struct Cellule{
16         Cellule(const T& e, Cellule* c);
17         T contenu;
18         Cellule* suivante;
19     };
20     Cellule* sommet;
21 };
```

```

1  /* q1e-alice.cpp */
2  int foo(Pile<int>* pile){
3      Pile<int>* c = new Pile<int>(*pile);
4      int s = 0;
5      while(!c->vide())
6          s += c->depiler();
7      delete c;
8      return s;
9  }
10 int main(){
11     Pile<int>* p = new Pile<int>();
12     while(cin){
13         int i=0;
14         cin >> i;
15         p->empiler(i);
16     }
17     cout << foo(p) << endl;
18     /* ... */
19     delete p;
20     return 0;
21 }
```

```

1  /* q1e-bob.cpp */
2  int foo(Pile<int> pile){
3
4      int s = 0;
5      while(!pile.vide())
6          s += pile.depiler();
7
8      return s;
9  }
10 int main(){
11     Pile<int> p;
12     while(cin){
13         int i=0;
14         cin >> i;
15         p.empiler(i);
16     }
17     cout << foo(p) << endl;
18     /* ... */
19
20     return 0;
21 }
```



## Annexe B pour la Question 2

Cette page peut être détachée.

```
1  /** q2.cpp pour la Question 2 */
2  #include <iostream>
3  #include <tableau.h>
4  #include <point.h>
5  using namespace std;
6
7  int q2(const Tableau<Point>& t){
8      int nb=0;
9      for(int i=0;i<t.taille();i++)
10         for(int j=i+1;j<t.taille();j++)
11             for(int k=j+1;k<t.taille();k++){
12                 double d1, d2, d3;
13                 d1 = t[i].distance(t[j]);
14                 d2 = t[i].distance(t[k]);
15                 d3 = t[j].distance(t[k]);
16                 if(d1==d2 && d2==d3)
17                     nb++;
18             }
19     return nb;
20 }
21 int main(){
22     Tableau<Point> nuage;
23     while(cin){
24         Point p;
25         cin >> p >> std::ws;
26         nuage.ajouter(p);
27     }
28     cout << q2(nuage) << endl;
29     return 0;
30 }
```

## Annexe C pour la Question 4

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).

## Annexe D pour la Question 5

Fichier INF3105.txt :

1	IIIII11050520	A
2	JJJJ11563388	A
3	MMMM26506769	A
4	CCCC04104447	B
5	GGGG10613710	B
6	AAAA02024750	B
7	KKKK15597411	B
8	LLLL14543156	B
9	VVVV22001013	B
10	BBBB18599738	C
11	DDDD10550203	C
12	EEEE10057851	C
13	FFFF04031148	C
14	HHHH01555089	C
15	NNNN13568130	C
16	WWW25596464	C

Fichier INF4230.txt :

1	IIIII11050520	A
2	MMMM26506769	A
3	GGGG10613710	A
4	KKKK15597411	A
5	LLLL14543156	A
6	NNNN13568130	A
7	AAAA02024750	B
8	JJJJ11563388	B
9	BBBB18599738	B
10	DDDD10550203	B
11	HHHH01555089	B
12	XXXX06016863	B
13	YYYY09539502	B
14	ZZZZ06022620	B
15	CCCC04104447	D
16	EEEE10057851	D
17	FFFF04031148	D

À noter que les données dans cette page sont purement aléatoires et ne reflètent pas la réalité !

Matrice à produire en sortie (en format texte simplifié) :

	A	B	C	D	E
A	2	1	0	0	0
B	3	1	0	1	0
C	1	3	0	2	0
D	0	0	0	0	0
E	0	0	0	0	0

Dans la matrice ci-dessus, les lignes et les colonnes correspondent respectivement aux notes obtenues en INF3105 et INF4230. Le nombre dans une case ligne  $l$  et colonne  $c$  indiquent le nombre d'étudiants qui a obtenu la note  $l$  en INF3105 et la note  $c$  en INF4230.

Exemple de code pour lire un fichier de notes et afficher une matrice.

```

1 void lireNotes() {
2     ifstream f1("INF3105.txt");
3     while(f1) {
4         string code;
5         char note='?';
6         f1 >> code >> note;
7     }
8 }
9 void afficherMatrice() {
10     for(char note3105='A'; note3105<='E'; note3105++) {
11         for(char note4230='A'; note4230<='E'; note4230++)
12             std::cout << '\t' << 1;
13         std::cout << std::endl;
14     }
15 }
```