

# INF3105 – Structures de données et algorithmes

## Automne 2019 – Examen de mi-session / Partie B

Éric Beaudry et Carl Simard  
Département d'informatique  
Université du Québec à Montréal

Dimanche 27 octobre 2019 – 9h30 à 12h30 (3 heures) – Locaux PK-{1[36]20,R650}

### Instructions

1. Aucune documentation n'est permise, excepté l'aide-mémoire C++ (feuille recto verso).
2. Les appareils électroniques, incluant les téléphones et les calculatrices, sont strictement interdits.
3. Répondez directement sur le questionnaire à l'intérieur des endroits appropriés.
4. Pour les questions demandant l'écriture de code :
  - le fonctionnement correct, l'efficacité (temps et mémoire), la clarté, la simplicité du code et la robustesse 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.
5. **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.
6. Ne détachez pas les feuilles du questionnaire, à moins de les brocher à nouveau avant la remise.

### Identification

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

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

Groupe : [ ] 20 (mardi 13h30) [ ] 40 (jeudi 18h)

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

### Résultat

Q1		/ 20
Q2		/ 20
Q3		/ 20
<b>Total Partie B</b>		<b>/ 60</b>

# 1 Représentation en mémoire de diverses structures [20 points]

Soit `progD.cpp` et `text3.txt` à l'Annexe D. On exécute ce programme avec la commande `./progD < test3.txt`. Dessinez la représentation en mémoire lorsque l'exécution de ce programme est rendue la ligne 14. Montrez clairement les objets alloués sur la pile d'exécution (*call stack*) et ceux sur le tas (*heap*).

## 2 Arbres binaires de recherche [20 points]

Un extrait de la classe `ArbreAVL<T>` est fourni à l'Annexe C.

(a) Codez la fonction `precedent` qui retourne un pointeur sur l'élément dans l'arbre qui précède la valeur `e`. Si cet élément n'existe pas, la fonction retourne `nullptr` (un pointeur nul). (10 points)

```
template <class T> const T* ArbreAVL<T>::precedent(const T& e) const {
```

(b) Codez une fonction retournant le nombre d'éléments communs dans deux (2) arbres AVL. (10 points)

```
template<class T> int nbcommun(const ArbreAVL<T>& a1, const ArbreAVL<T>& a2){
```

### 3 Résolution d'un problème – Analyse d'inscriptions (20 points)

On vous fournit un fichier contenant une liste d'étudiants. Chaque ligne est composée d'un code permanent suivi des groupes-cours auxquels l'étudiant est inscrit, le tout séparé par des espaces et terminé par un point-virgule. Des exemples de fichier et de programme lisant ce fichier sont fournis à l'Annexe E.

**Codez un programme qui détermine les deux (2) étudiants qui ont le plus de groupes-cours communs.** En cas d'égalité, il suffit de retourner n'importe quelle paire d'étudiants parmi les solutions possibles. Contrainte / Indice : vous devez utiliser (appeler) la fonction `nbcommun` codée à la question 2b.

```
1 // Supposez toutes les directives #include voulues et using namespace std.  
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 }
```

## Annexe A – Programme progA.cpp

```
1 #include <iostream>
2 class Point{
3     int x, y, z;
4     public:
5     Point(int x_=0, int y_=0);
6     ~Point();
7 };
8 class Rectangle{
9     Point hg, bd; // haut-gauche, bas-droit
10    public:
11    Rectangle();
12    Rectangle(const Point& hg_, const Point& bd_);
13    ~Rectangle();
14 };
15 Point::Point(int x_, int y_)
16 : x(x_), y(y_) {
17     std::cout << "P" << x << y; }
18 Point::~~Point() {
19     std::cout << "p" << x << y; }
20 Rectangle::Rectangle() {
21     std::cout << "R_"; }
22 Rectangle::Rectangle(const Point& hg_, const Point& bd_)
23 : hg(hg_), bd(bd_) {
24     std::cout << "G_"; }
25 Rectangle::~~Rectangle() {
26     std::cout << "r_"; }
27 void f1(){
28     Point p1(1,2), p2(3,4);
29     Rectangle r1(p1, p2);
30     Rectangle r2(r1);
31 }
32 void f2(){
33     Rectangle* tr = new Rectangle[2];
34     tr[0] = Rectangle(Point(1,2),Point(3,4));
35 }
36 void f3(){
37     Point* p = new Point[3];
38     Rectangle* r = new Rectangle[2];
39     r = new Rectangle[4];
40     delete[] r;
41     delete[] p;
42 }
43 int main(){
44     f1(); std::cout << std::endl;
45     f2(); std::cout << std::endl;
46     f3(); std::cout << std::endl;
47     return 0; }
```

## Annexe B – Programme progB.cpp

```

1 #include <iostream>           // pour compiler: g++ -o progB progB.cpp
2 #include <string>
3 #include <map>
4 int main(){
5     std::map<std::string, int> compte; // map : dictionnaire basé sur un arbre R-N
6     int position=0, nbmax=0, k17=0;
7     std::string mot, meilleur;
8     while(std::cin && !std::cin.eof()){
9         // hypothèse: en entrée, chaque mot est suivi d'au moins 1 espace blanc
10        std::cin >> mot >> std::ws;
11        position++;
12        if(mot == ".")
13            position=0;
14        else if(position<=3 && ++compte[mot]>nbmax){
15            k17++;
16            nbmax = compte[mot];
17            meilleur = mot;
18        }
19    }
20    std::cout << meilleur << std::endl;
21    std::cout << k17 << std::endl;
22    return 0;
23 }
```

Fichier test1.txt :

```
1 a b c d e f g . b c d e f g h j a . c e g z x y a . c b a k a l d b b b b . d c b a k .
```

Fichier test2.txt :

```
1 a b c d e . e d c b a . a b b c c .
```

## Annexe C – Extrait de la classe ArbreAVL

```

1 template <class T> class ArbreAVL{
2     struct Noeud{
3         T contenu; //Rappel : gauche->contenu < contenu < droite->contenu
4         int equilibre;
5         Noeud *gauche, *droite;
6     };
7     Noeud* racine;
8     public:
9         // ...
10        const T* precedent() const;
11        Iterateur debut() const;
12        // ...
13 };
```

## Annexe D – Programme progD.cpp

```

1 #include <string>
2 #include <arbreavl.h> // arbre AVL présenté dans le cours
3 #include <arbremap.h> // dictionnaire basé sur arbre AVL présenté dans le cours
4 #include <liste.h> // simplement chaînée, implément. naïve sans décalage de pointeurs
5 int main(){
6     ArbreMap<string, Liste<int> > d1;
7     ArbreMap<string, ArbreAVL<int> > d2;
8     for(int i=0; std::cin && !std::cin.eof(); i++){
9         string mot;
10        std::cin >> mot >> std::ws;
11        d1[mot].inserer(i);
12        d2[mot].inserer(i);
13    }
14    // Ligne 14 pour Question 1
15    return 0;
16 }
```

Fichier test3.txt :

```
1 a b b c a a x e f f f
```

## Annexe E – Partie B / Q3 : Exemple de fichier et de programme de lecture

```

1 AAAA01019600 INF1120-10 INF1070-10 INF1132-10 ;
2 BBBB02029600 INF1120-10 INF1070-20 INF1132-20 MET1110-05 ;
3 CCCC03039600 INF1120-10 INF1070-10 INF1132-20 ;
4 DDDD04049600 INF1120-20 INF1070-10 INF1132-20 MET1110-10 ;
5 EEEE05059600 INF1120-20 INF1070-10 INF1132-20 MET1110-10 ;
```

```

1 #include <iostream>
2 using namespace std;
3 int main(){
4     while(cin && !cin.eof()){
5         string etudiant;
6         cin >> etudiant >> ws; // ws: force la lecture d'un espace blanc
7         while(cin && !cin.eof()){
8             string cg; // cours-groupe
9             cin >> cg >> ws;
10            if(cg=="") break; // sort du 2e while
11        }
12    }
13 }
14 return 0;
15 }
```