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 ambigüité 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.

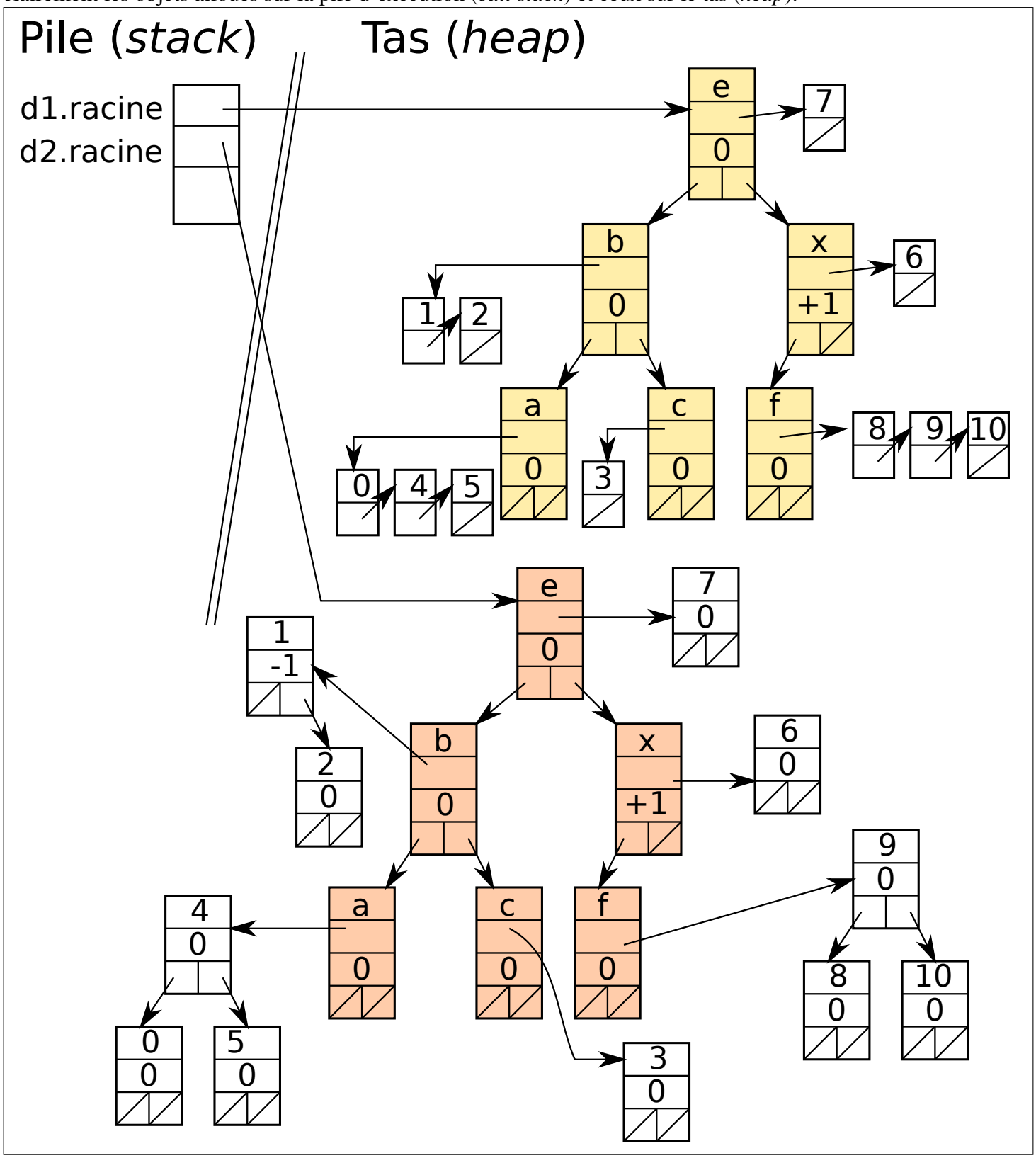
Résultat

Q1		/ 20
Q2		/ 20
Q3		/ 20
Total Partie B		/ 60

Solutionnaire

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 founi 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)
Solution en $O(\log n)$.

```
1  template <class T> const T* ArbreAVL<T>::precedent(const T& e) const {
2      T* precedent=nullptr;
3      const Noeud* courant = racine;
4      while(courant!=nullptr){
5          if(e <= courant->contenu) // on pourrait enlever <= plus tard...
6              courant = coutant->gauche;
7          else{
8              precedent = &(courant->contenu); // candidat à précéder e
9              courant = courant->droite;
10         }
11     }
12     return precedent;
13 }
```

(b) Codez une fonction retournant le nombre d'éléments communs dans deux (2) arbres AVL. (10 points)
Solution idéale en $O(n)$. **Idée :** comme dans la fusion dans le tri de fusion, on parcours les 2 arbres en même temps en avancant dans celui ayant l'élément courant le plus petit.

```
1  template<class T> int nbcommun(const ArbreAVL<T>& a1, const ArbreAVL<T>& a2){
2      int nb = 0;
3      ArbreAVL<T>::iterateur i1 = a1.debut();
4      ArbreAVL<T>::iterateur i2 = a2.debut();
5      while(i1 && i2){
6          if(*i1 == *i2){
7              ++i1;
8              ++i2;
9              nb++;
10         }else if(*i1 < *i2) ++i1;
11         else ++i2; // *i2 < i1
12     }
13     return nb;
14 }
```

Solution naïve en $O(n \log n)$.

```
1  template<class T> int nbcommun(const ArbreAVL<T>& a1, const ArbreAVL<T>& a2){
2      int nb = 0;
3      for(ArbreAVL<T>::iterateur i = a1.debut();i;++i)
4          if(a2.contient(*i))
5              nb++;
6      return nb;
7  }
```

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  int main(){
3      ArbreMap<string, set<string> > etudiants; // codeperm --> ensemble cours-groupe
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             etudiants[etudiant].inserer(cg);
12         }
13     }
14     int nbmax=0;
15     string me1, me2;
16     for(ArbreMap<string, set<string> >::IterateurConst i1=etudiants.debut();i1;++i1){
17         for(auto i2=i1+1;i2;++i2){
18             int nb = nbcommun(i1.valeur(), i2.valeur());
19             if(nb>nbmax){
20                 nbmax = nb;
21                 me1 = i1.cle();
22                 me2 = i2.cle();
23             }
24             //cout << nb << '\t'; // pour afficher le tableau pour vérification
25         }
26         //cout << endl; // pour afficher le tableau pour vérification
27     }
28     cout << me1 << '\t' << me2 << endl;
29     return 0;
30 }
```

La solution ci-haut est la solution attendue, car elle respecte la contrainte imposée d'utiliser `nbcommun`. Coût : $O(n^2 \log m)$ où n est le nombre d'étudiants et m est le nombre de groupes-cours maximum (moyen) qu'un étudiant peut prendre (prend).
Sans cette contrainte imposée, il aurait été possible de faire mieux. En effet, la solution ci-haut nécessite de parcourir toutes les $n(n-1)/2$ paires d'étudiants. Dans une université raisonnable, on peut présumer qu'il y aura beaucoup de paires d'étudiants qui n'auront aucun cours en commun.
Au verso, une version alternative améliorée, mais non conforme à la contrainte imposée, est présentée.

Pour faire mieux, l'idée est d'itérer uniquement sur étudiants qui ont au moins un cours en commun.

```
1  // Supposez toutes les directives #include voulues et using namespace std.
2  int main(){
3      ArbreMap<string, set<string> > etudiants; // nom --> ensemble cours-groupe
4      ArbreMap<string, set<string> > groupescours; // groupes-cours --> étudiants
       inscrits
5      while(cin && !cin.eof()){
6          string etudiant;
7          cin >> etudiant >> ws; // ws: force la lecture d'un espace blanc
8          while(cin && !cin.eof()){
9              string cg; // cours-groupe
10             cin >> cg >> ws;
11             if(cg==";") break; // sort du 2e while
12             etudiants[etudiant].inserer(cg);
13             groupescours[cg].inserer(etudiant);
14         }
15     }
16     int nbmax=0;
17     string me1, me2;
18     for(ArbreMap<string, set<string> >::IterateurConst i1=etudiants.debut();i1;++i1){
19         ArbreAVL<string> autresetudiants;//tous les étu. ayant au moins 1 cours commun
20         for(auto i2=i1.valeur().debut();i2;++i2)
21             for(auto i3=groupescours[*i2];i3;++i3)
22                 autresetudiants.inserer(*i3);
23         for(auto i2=autresetudiants.debut();i2;++i2)
24             if(i1.cle() < *i2){ // éviter comparaison avec lui-même et doublons
25                 int nb = nbcommun(i1.valeur(), *i2);
26                 if(nb>nbmax){
27                     nbmax = nb;
28                     me1 = i1.cle();
29                     me2 = *i2;
30                 }
31             }
32     }
33     cout << me1 << '\t' << me2 << endl;
34     return 0;
35 }
```

Cette solution coûte : $O(nl\log n)$ pour les lignes 20-22, où l est le nombre maximum (moyen) d'étudiants inscrits dans un groupe-cours, et n est toujours le nombre d'étudiants.
Ensuite, les lignes 23-31 coûte $O(nlm)$ où m est toujours le nombre de groupes-cours maximum (moyen) qu'un étudiant peut prendre (prend).
Total : $O(nl(m+\log n))$. S'il est raisonnable de supposer que $m << \log n$, on peut simplifier à $O(nl\log n)$.
Généralement, dans une université raisonnable, $l\log n << n$. Par exemple, pour une session, dans le cas de l'UQAM : $n = 40000$, $m = 5$, $l = 42$. Donc, cette solution est meilleure que celle présentée à la page précédente.

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`

```cpp
#include <iostream>          // pour compiler: g++ -o progB progB.cpp
#include <string>
#include <map>
int main(){
    std::map<std::string, int> compte; // map : dictionnaire basé sur un arbre R-N
    int position=0, nbmax=0, k17=0;
    std::string mot, meilleur;
    while(std::cin && !std::cin.eof()){
        // hypothèse: en entrée, chaque mot est suivi d'au moins 1 espace blanc
        std::cin >> mot >> std::ws;
        position++;
        if(mot == ".")
            position=0;
        else if(position<=3 && ++compte[mot]>nbmax){
            k17++;
            nbmax = compte[mot];
            meilleur = mot;
        }
    }
    std::cout << meilleur << std::endl;
    std::cout << k17 << std::endl;
    return 0;
}
```

Fichier `test1.txt` :

```
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` :

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

Annexe C – Extrait de la classe ArbreAVL

```cpp
template <class T> class ArbreAVL{
    struct Noeud{
        T contenu;  //Rappel : gauche->contenu < contenu < droite->contenu
        int equilibre;
        Noeud *gauche, *droite;
    };
    Noeud* racine;
  public:
    // ...
    const T* precedent() const;
    Iterateur debut() const;
    // ...
};
```

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 chainée, 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 }
```