

# INF3105 – Structures de données et algorithmes

## Examen final – Automne 2013

Éric Beaudry  
Département d'informatique  
Université du Québec à Montréal

Jeudi 12 décembre 2013 – 13h30 à 16h30 (3 heures) – Locaux SH-3620 + SH-3320

### Instructions

- Aucune documentation n'est permise.
- 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 caractéristiques 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 ;
  - le respect exact de la syntaxe de C++ n'est pas sujet à la correction.
- 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 4 questions.
- À l'exception de l'annexe à la fin du questionnaire, ne détachez aucune feuille.
- Le côté verso peut être utilisé comme brouillon.

### Identification

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

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

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

### Résultat

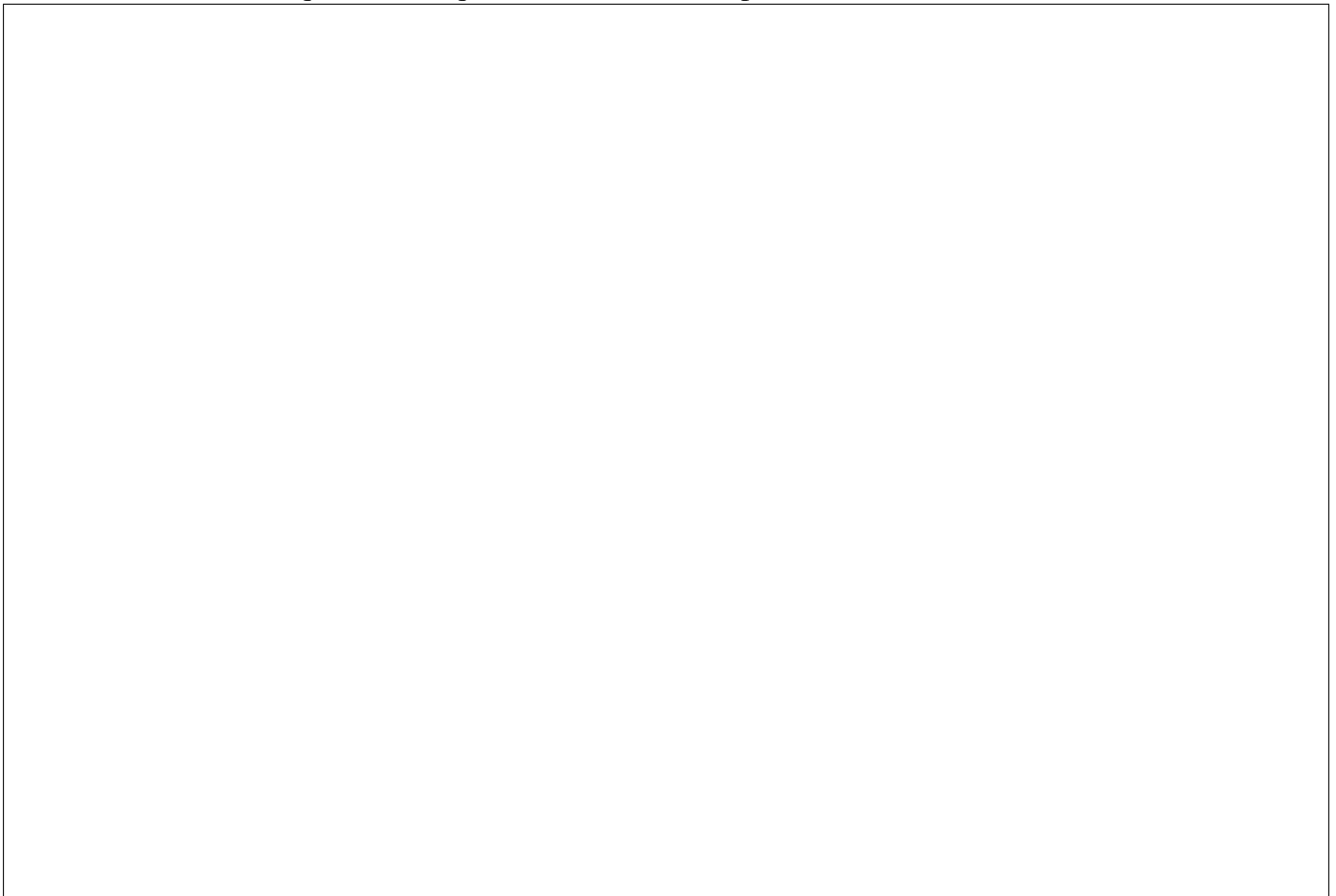
Q1		/ 5
Q2		/ 6
Q3		/ 9
Q4		/ 5
Total		/ 25

# 1 Monceaux (*Heaps*) et connaissances techniques de C++ [5 points]

(a) Insérez les entiers 7, 8, 1, 5, 9, 10 et 5 dans un monceau initialement vide. Dessinez sous forme d'arbre l'état du monceau après chaque insertion. Les étapes intermédiaires ne sont pas demandées. [2 points]



(b) Référez-vous au code fourni à l'Annexe A (vous pouvez détacher la page 8). Dessinez la représentation en mémoire du programme `main.cpp` rendu à la ligne 32. Soyez aussi précis que possible. Montrez clairement ce qui est sur la pile d'exécution et ce qui est sur le tas (*heap*). Remarquez que les temps des objets `Alarme` sont les mêmes qu'à la sous-question (a) ci-haut ! [3 points]



## 2 Table de hachage (*Hashtable*) [6 points]

(a) Qu'est-ce qu'une collision dans une table de hachage ? Expliquez en vos propres mots. [1 point]

(b) Quelle est la complexité temporelle de l'insertion dans une table de hachage ? Supposez la gestion de collisions au moyen d'une liste chaînée externe. Supposez  $n$ =nombre d'éléments. [2 points]

Cas moyen (0.5 point) :	Pire cas (0.5 point) :	Décrivez le pire cas.

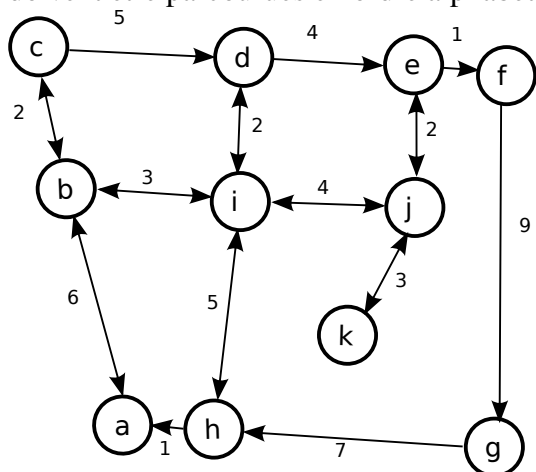
(c) Insérez dans l'ordre les clés 12, 72, 31, 170, 10, 25, 59 et 99 dans une table de hachage. Cette table contient 10 casiers (*buckets*) et n'est jamais redimensionnée. Chaque casier ne peut contenir qu'au plus une clé. Les collisions doivent être gérées à l'aide de la résolution linéaire. [1 point]

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

(d) Supposez qu'un de vos collègues souhaite améliorer les performances de son TP3 (Planification de déplacements sur l'île de Montréal) / partie A (génération du chemin le plus court). Ce dernier dit qu'il a utilisé les conteneurs `std::set` et `std::map` (arbres rouge-noir). Il songe maintenant à utiliser les conteneurs `std::unordered_set` et `std::unordered_map` (tables de hachage). Croyez-vous que cela peut améliorer «significativement» les performances de son programme `tp3a` ? Justifiez. [2 points]

### 3 Graphes [9 points]

Aux sous-questions (a) à (d), considérez le graphe ci-dessous. En (a) et (b), les arêtes sortantes d'un sommet doivent être parcourues en ordre alphabétique de leur sommet d'arrivée.



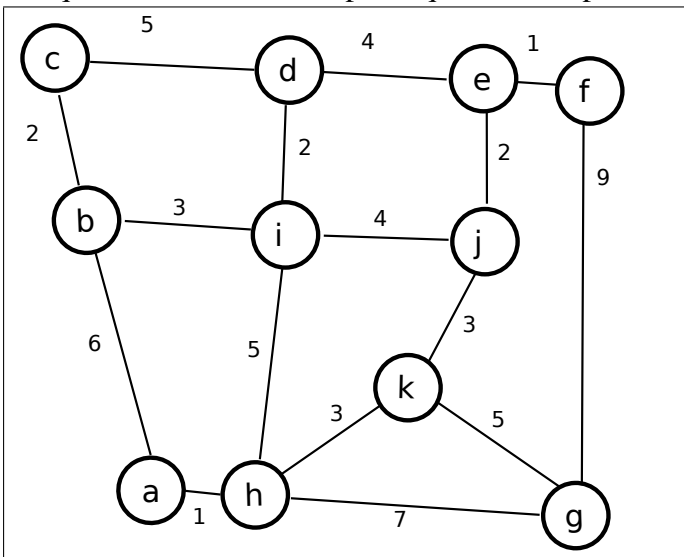
(a) Écrivez l'ordre de visite des sommets d'une recherche en **profondeur** à partir du sommet **f**. [1 point]

(b) Écrivez l'ordre de visite des sommets d'une recherche en **largeur** à partir du sommet **i**. [1 point]

(c) Simulez l'algorithme de Dijkstra pour calculer le plus court chemin de *e* à *a*. Il y a plusieurs façons de présenter votre réponse. L'important est de démontrer votre compréhension de l'algorithme. Les éléments clés à présenter sont l'ordre de visite des sommets et les valeurs *Dist* et *Parent*. [2 points]

(d) Dans un graphe, le plus court chemin reliant une paire de sommets n'est pas nécessairement unique. Par exemple, dans le graphe à la page précédente, il existe deux chemins optimaux de  $b$  à  $e$  :  $\langle (b,i), (i,j), (j,e) \rangle$  et  $\langle (b,i), (i,d), (d,e) \rangle$ . Expliquez comment adapter l'algorithme de Dijkstra pour vérifier l'unicité du chemin le plus court. En d'autres mots, l'algorithme doit calculer un booléen qui doit être mis à `true` si et seulement si la solution optimale est unique. [2 points]

(e) Calculez l'arbre de recouvrement minimal du graphe suivant en utilisant l'algorithme de Prim-Jarnik. Indiquez clairement les étapes requises. La réponse n'est pas forcément unique. [2 points]



(f) Vrai ou faux : l'algorithme de Dijkstra boucle à l'infini dans un graphe non connexe. Justifiez. [1 point]

## 4 Résolution d'un problème [5 points]

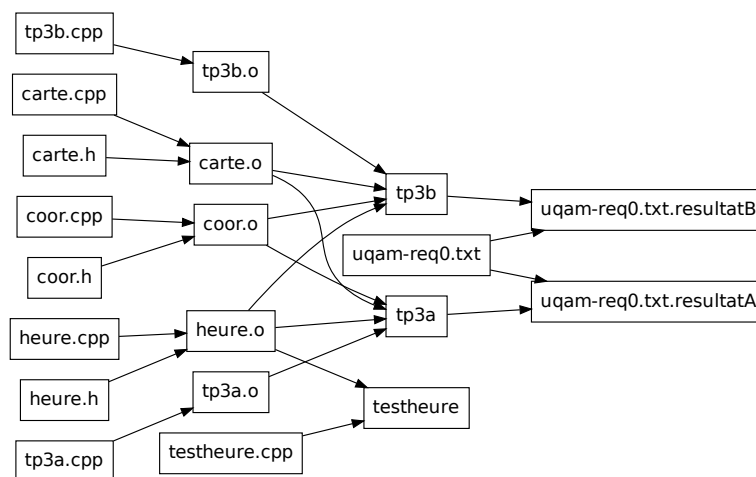
Un fichier `Makefile` décrit comment construire un projet à partir de ses fichiers sources. Pour simplifier le problème, considérons une syntaxe simplifiée de `Makefile`. Un fichier `Makefile` spécifie une liste de cibles (fichiers) à construire. Chaque cible est spécifiée sur deux lignes. Sur la première ligne, on retrouve le nom de fichier de la cible, un deux-points (`:`) et la liste des fichiers requis (dépendances). Sur la deuxième ligne, débutant par une tabulation, on retrouve la commande à exécuter pour construire la cible.

Voici ci-bas à gauche un exemple de fichier `Makefile` pour le TP3. Un `Makefile` peut être représenté à l'aide d'un graphe orienté, où les sommets sont des fichiers et les arêtes expriment les relations de dépendance et/ou d'ordonnancement. La figure ci-bas à droite montre le graphe pour le `Makefile` à sa gauche.

```

1 tp3a : carte.o coor.o heure.o tp3a.o
2   g++ -o tp3a carte.o coor.o heure.o tp3a.o # c1
3 tp3b : carte.o coor.o heure.o tp3b.o
4   g++ -o tp3b carte.o coor.o heure.o tp3b.o # c2
5 tp3a.o : tp3a.cpp
6   g++ -o tp3a.o tp3a.cpp # c3
7 tp3b.o : tp3b.cpp
8   g++ -o tp3b.o tp3b.cpp # c4
9 carte.o : carte.h carte.cpp
10  g++ -o carte.o carte.cpp # c5
11 coor.o : coor.h coor.cpp
12  g++ -o coor.o coor.cpp # c6
13 heure.o : heure.h heure.cpp
14  g++ -o heure.o heure.cpp # c7
15 uqam-req0.txt.resultatA : tp3a uqam-req0.txt
16   ./tp3a ... uqam-req0.txt>uqam-req0.txt.resultatA # c8
17 uqam-req0.txt.resultatB : tp3b uqam-req0.txt
18   ./tp3b ... uqam-req0.txt>uqam-req0.txt.resultatB # c9
19 testheure : testheure.cpp heure.o
20  g++ -o testheure testheure.cpp heure.o #c10

```



On vous demande d'écrire un programme `make`. Lisez la sous-question (b) avant de répondre à (a).

(a) Complétez la représentation de la classe `Makefile` pour stocker le contenu d'un `Makefile`. [2 points]

```

1 class Makefile{
2   public:
3     list<string> getSequenceCommandes() const; // option (b1)
4     list<list<string>> getSequenceGroupesCommandes(int n) const; // option (b2)
5   private: // Completez uniquement la representation
6     struct Cible{
7
8
9
10
11
12   };
13
14
15
16
17
18
19   friend istream& operator >> (istream&, Makefile&);
20 };

```

(b) Choisissez une seule fonction entre (b1) et (b2). Implémentez-la en C++ ou en pseudo-code. [3 points]

(b1) [maximum 1.5 point] La fonction `list<string> Makefile::getSequenceCommandes()` retourne une séquence correctement ordonnée de commandes à exécuter pour construire le projet. Par exemple,  $\langle c3, c4, c5, c6, c7, c1, c2, c8, c9, c10 \rangle$  est une liste correcte pour l'exemple, où `c3="g++ -o tp3a.o tp3a.cpp"`, `c4="g++ -o tp3b.o tp3b.cpp"`, etc. Indice : parcours en profondeur.

(b2) [maximum 3 points] Si on dispose de plusieurs unités de calcul (processeurs ou cœurs), certaines cibles peuvent être construites en parallèle. Pour simplifier le problème, on suppose que toutes les commandes ont la même durée. La fonction `list<list<string> > Makefile::getSequenceGroupesCommandes(int n)` retourne une séquence correctement ordonnée de groupes de commandes à exécuter pour construire le projet sur une machine disposant de  $n$  unités de calcul. Chaque groupe contient  $i$  commande(s) à exécuter en parallèle tel que  $1 \leq i \leq n$ . Dans l'exemple, si  $n = 2$ , alors la fonction pourrait retourner  $\langle \langle c3, c4 \rangle, \langle c5, c7 \rangle, \langle c6, c10 \rangle, \langle c1, c2 \rangle, \langle c8, c9 \rangle \rangle$ . Les commandes `c3` et `c4` s'exécutent en parallèle, ensuite `c5` et `c7`, etc.

## Annexe A pour la Question 1

Cette page peut être détachée.

```

1  /*** tableau.h ***/
2  template <class T> class Tableau{
3  public:
4      Tableau(int capacite_initiale=8);
5      Tableau(const Tableau&);
6      ~Tableau();
7
8      void ajouter(const T& element);
9      bool vide() const;
10     void vider();
11     int taille() const;
12     T& operator[] (int index);
13     const T& operator[] (int index)
14         const;
15     Tableau<T>& operator=(const
16         Tableau<T>&);
17     bool operator==(const
18         Tableau<T>&) const;
19
20 private:
21     T* elements;
22     int capacite;
23     int nbElements;
24 };
25 // ...

```

```

1  /*** monceau.h ***/
2  #include "tableau.h"
3  template <class T> class Monceau {
4  public:
5      void inserer(const T&);
6      void enleverMinimum();
7      void enleverMinimum(T& sortie);
8      const T& minimum() const;
9      bool estVide() const;
10
11 private:
12     Tableau<T> valeurs;
13     void remonter(int indice);
14     void descendre(int indice);
15 };

```

```

1  /*** main.cpp ***/
2  #include "monceau.h"
3  class Alarme{
4  public:
5      Alarme(int t=0, const char* c="XX");
6      Alarme(const Alarme&);
7      bool operator<(const Alarme&) const;
8  private:
9      int temps; // "date" de l'alarme
10     char[2] code;
11 };
12 Alarme::Alarme(int t, const char* c)
13 : temps(t) {
14     strncpy(code,c,2); //Voir [1] ci-bas
15 }
16 Alarme::Alarme(const Alarme& a)
17 : temps(a.temps) {
18     strncpy(code,a.code,2); }
19 bool Alarme::operator<(const Alarme&
20     a) const{
21     return temps < a.temps;
22 }
23
24 int main(){
25     Monceau<Alarme> alarmes;
26     Monceau<Alarme> alarmes2;
27     alarmes.inserer(Alarme(7, "SH"));
28     alarmes.inserer(Alarme(8, "PK"));
29     alarmes.inserer(Alarme(1, "CB"));
30     alarmes.inserer(Alarme(5, "SB"));
31     alarmes.inserer(Alarme(9, "KI"));
32     alarmes.inserer(Alarme(10, "SU"));
33     alarmes.inserer(Alarme(5, "CO"));
34     // Dessinez l'etat rendu ici
35     alarmes2 = alarmes;
36     while(!alarmes.estVide()){
37         // ...
38         alarmes.enleverMinimum();
39     }
40     return 0;
41 }

```

[1] La fonction **C** `strncpy(char* dest, const char* src, int n)` copie jusqu'à `n` caractères de la chaîne `src` vers la chaîne `dest`.