INF3105 – Structures de données et algorithmes
Pratique avant l'examen de mi-session Hiver 2012

Éric Beaudry
Département d'informatique
Université du Québec à Montréal

Lundi 27 août 2012, 18h30 – 21h30, local SH-2140

1 Connaissances techniques (6 points)

1.1 C++ (2 points)

Qu'affiche le programme suivant ?

```
#include <iostream>
void f1(int* tab, int n){
 int* t = tab, *f=tab+n;
 while(t<f) *(t++) *= 2;
}
void f2(int& a, int& b){
 a += b++;
}
int main(){
 int* t = new int[8];
 for(int i=0;i<5;i++) t[i]=i;
 f1(t, 4);
 int x = 0;
 for(int i=0;i<5;i++)  f2(x, t[i]);
 for(int i=0;i<8;i++)  std::cout << " " << t[i];
 std::cout << " : " << x << std::endl;
}
```

1.2 Listes doublement chaînées (2 points)

```
template <class T>  class Liste{
   class Cellule{
     public:
       Cellule(const T& e, Cellule* s, Cellule* p) : precedente(p), suivante(s) {element=e;}
       Cellule* precedente;
       Cellule* suivante;
       T element;
   };
   Cellule* premiere;
 public:
   Liste() { premiere=NULL;}
   ~Liste();
   Liste inverse() const;
};
```

Donnez le code de la fonction `Liste::inverse()` qui retourne une copie de la liste avec les éléments dans l'ordre inverse.

1.3 Files (2 points)

Deux implémentations de files ont été présentées dans le cours. Pour chaque implémentation, nommez son principal avantage et un exemple d'application où son usage s'avère approprié.
(a) File circulaire implémentée à l'aide d'un tableau de dimension fixe.
(b) File implémentée à l'aide de pointeurs (chaîne de cellules).

2 Arbres binaires de recherche (8 points)

2.1 ArbreAVL : :== (2 points)

Complétez le code C++ de l'opérateur == de la classe générique ArbreAVL ci-dessous. Vous devez considérer tous les cas limite.

```
template <class T> class ArbreAVL{
 public:
  ArbreAVL();
  ~ArbreAVL();
  bool operator == (const ArbreAVL&);
 private:
  class Noeud{
   public:
    Noeud(const T&);
    Noeud *gauche;
    Noeud *droite;
    int equilibre;
    T element;
  };
  Noeud* racine;
};
```

2.2 Arbre AVL minimal

Dessinez un arbre AVL ayant une hauteur de 5 avec un nombre minimal de noeuds.

2.3 Insertion (2 points)

Dessinez un arbre AVL après l'insertion de chacun des nombres : 4, 5, 1, 2, 3, 7 et 1. L'arbre est initialement vide.

2.4 Suppression (2 points)

Supprimez l'élément 7 dans l'arbre précédent.

2.5 Trouver les erreurs (4 points)

Le programme suivant contient des erreurs. Identifiez les erreurs et proposez des corrections.

```
template <class T>
class ArbreAVL{
 public:
  //...
  T& minimum() {return minimum(racine); }
 private:
  T& minimum(Noeud* n){
   if(n->gauche==NULL) return n->element;
   return minimum(n->gauche);
  //...
};
int main()
{
 ArbreAVL<int> a;
 for(int i=0;i<10;i++) a.inserer(i);
 int& m = a.minimum();
 m += 10;
 std::cout << "m=" << m << std::endl;
}
```

2.6 Minimum et maximum dans un Arbre-B (1 point)

Quelle est la complexité algorithmique des fonctions qui retournent le minimum et le maximum d'un arbre B ?

3 Résolution de problème (5 points)

Vous devez écrire un programme qui reçoit en entrée des lectures de température. Chaque ligne comporte une date exprimée en heure et une température en Celsius séparées d'un espace blanc. Tous les nombres sont des nombres réels (`double`). Les données sont dans l'ordre chronologique. Voici un exemple d'entrée.

```
0.00 9.7
1.00 10.1
2.00 10.2
3.00 10.5
4.00 10.3
5.00 10.7
6.01 11.7
7.00 12.4
8.01 14.7
```

3.1 Partie A

Écrivez un programme qui permet (1) d'estimer la température à une date donnée et (2) de calculer la température moyenne entre deux dates. On suppose que la température varie de façon linéaire entres deux dates pour lesquelles la température est connue. Par exemple, la température au temps 0.5 est de 9.9. Votre solution doit être construite autour d'une classe `Historique` qui permet de garder l'historique des températures en mémoire. Vous devez choisir une représentation de la classe `Historique` et coder les fonctions `estimeTemperature` et `calculeMoyenne`. Si ces fonctions appellent d'autres fonctions, vous devez les donner. Vous n'avez pas à fournir le code pour lire les données. Vous devez vous soucier des qualités usuelles d'un logiciel, soit l'efficacité (mémoire et temps), la lisibilité du code, etc.

```
class Historique{
 private:
```

```
  // Representation
 public:
  Historique();
  ~Historique();
  void ajouter(double date, double temperature);
  double estimeTemperature(double date) const;
  double calculeMoyenne(double datedebut, double datefin) const;
};
```

3.2 Partie B

Donnez les modifications requises (si nécessaire) pour traiter des cas où les données n'arriveraient pas de façon chronologique. Si vous jugez d'aucune modification n'est requise, expliquez pourquoi la solution en A est «parfaite». Si vous jugez que des modification sont requises, expliquez pourquoi la solution en A demeure pertinente dans le cas où les données sont dans l'ordre chronologique.

4 Connaissances générales (5 points, -1 par mauvaise réponse)

(a) Vrai ou faux : dans un tableau linéaire et dynamique, un ajout peut dégénérer en temps O(n).
(b) Vrai ou faux : dans un tableau linéaire et dynamique, le temps amorti de l'ajout est toujours de O(1). Si vrai, expliquez pourquoi. Si faux, indiquez une condition requise.
(c) Vrai ou faux : dans un arbre rouge-noir, on peut avoir plusieurs nœuds rouges de suite sur un chemin vers une feuille.
(d) Quel est le temps d'exécution d'une fonction faisant le parcours en largeur d'un arbre général (pas nécessairement binaire) ? Utilisez la notation grand O et donnez une courte justification.
(d) Vrai ou faux : un parcours en largeur s'implémente naturellement à l'aide d'une fonction récursive.
(e) Dans vos propres mots, définissez ce qu'est un **type abstrait de données**.
(f) En une ou deux phrases, décrivez le rôle d'un **destructeur** de classe en C++ ?