Examen de Programmation C/Java

– Examen (**1**) –

17 décembre 2018

Résumé

Ce sujet comprend deux parties, l'une à traiter en C et l'autre en Java. À l'issue de l'examen vous devrez envoyer votre travail par e-mail dans une archive contenant le code source des programmes à l'adresse `<emmanuel.fleury@u-bordeaux.fr>`.
L'archive devra avoir la forme suivante (remplacez les mots entre `<...>` par votre nom/prénom) :

```
<NOM>_<Prenom>-examen/
    +-- xmas-trees/
    +-- java/
```


1 Programmation C : Christmas Trees (12 points)

1.1 Mise en place du projet

Sachant qu'il n'y a aucune dépendance à une bibliothèque particulière, le but de cette section est de réaliser un build-system avec l'outil `make` qui permette de compiler l'exécutable demandé.

Questions

1. Mettre en place la structure complète des sources du programme comme suit :

```
NOM_Prenom-examen/
    +-- xmas-trees/
          +-- xmas-trees.c
          `-- Makefile
```

2. Créez le fichier `Makefile` et écrivez les cibles suivantes :
   - `all` : Lance la compilation de l'exécutable `xmas-trees` ;
   - `xmas-trees` : Compile le fichier source en un exécutable ;
   - `clean` : Nettoie le répertoire de tous les fichiers superflus et des fichiers crées par la compilation.
3. Au sein de `Makefile` utilisez (et positionnez) correctement les variables classiques, c'est à dire : `CFLAGS`, `CPPFLAGS` et `LDFLAGS`. Ainsi que la cible spéciale `.PHONY`.

1.2 Problème principal

Le père Noël est catastrophé car sa production de sapins de cette année a été sabotée par des hordes de lutins en gilets jaunes. Les malotrus ont profité de la nuit pour vandaliser les pépinières en détruisant les clôtures et en replantant les piquets des clôtures n'importe comment et en dépit du bon sens.

Les piquets sont magiques, ils offrent aux sapins des conditions idéales pour pousser et être prêt à décorer l'intérieur cossu des adorateurs de Noël. Le fait d'avoir modifié le placement des piquets dans la pépinière change totalement la plantation en cours et une partie de la récolte sera perdue.

Il s'agit donc d'estimer en urgence combien il aura de sapins prêts à être livrés à temps. Le service logistique commandera des sapins en Asie pour combler la différence et satisfaire les ~~clients~~ enfants.

Les piquets magiques marchent sur l'ensemble de la surface qu'ils entourent. Et, il faut compter environs un sapin par m^2. Le père Noël vous demande donc de calculer la surface de l'enveloppe convexe

des différents champs en fonction des positions des piquets qui sont plantés dedans. Attention, la surface finale doit rester convexe et il est tout à fait possible que certains piquets déplacés ne servent à rien car ils n'appartiennent pas à l'enveloppe convexe (ils sont trop à l'intérieur de la pépinière).

— **Entrée** : La première ligne de l'entrée donne le nombre de cas de tests, T. Puis, T cas de test suivent. Pour chacun des cas de test, la première ligne contient le nombre de piquets. Ceux-ci sont numérotés de 0 à $N-1$. Chacune des N lignes suivantes contient deux entiers X_i et Y_i séparés par un espace, ce sont les coordonnées du i-ème piquet.

— **Sortie** : Pour chaque cas de test, la sortie sera une ligne contenant "`Case #x: `" suivi par la surface convexe maximale que vous pouvez obtenir avec cet ensemble de piquets.

— **Petits ensembles de données** : $1 \leq T \leq 100$, $3 \leq N \leq 10$, $-100 \leq X_i, Y_i \leq 100$.

— **Gros ensembles de données** : $1 \leq T \leq 30$, $3 \leq N \leq 1000$, $-50000 \leq X_i, Y_i \leq 50000$.

Fichier d'entrée

```
3
4
1 2
2 0
0 0
1 1
5
0 0
1 1
2 2
0 2
2 0
3
0 0
1 0
0 1
```

Fichier de sortie

```
Case #1: 2.000000
Case #2: 4.000000
Case #3: 0.500000
```

Figure 1 – Exemple de fichier d'entrées/sorties.

La figure 1 présente un exemple de fichier d'entrée et une sortie possible. La sortie représente la surface totale que vous pouvez récupérer en utilisant une partie des piquets. Enfin, vous trouverez des exemples plus aboutis à l'URL suivante : `http://www.labri.fr/~fleury/courses/programming/exam/`

Questions

1. Tout d'abord, on utilisera la structure de donnée suivante pour représenter la position des piquets :

```c
/* 2D point data-structure */
typedef struct
{
  long long x;
  long long y;
} point_t;
```

   Comme on connait le nombre de points dès le début, écrivez une fonction qui alloue un tableau de points contenant assez de place pour `nb_points` : `point_t *vector_alloc (unsigned nb_points)`.

2. Écrivez ensuite la partie du programme qui ouvre un fichier pour récupérer les données du problème. On suppose que l'interface utilisateur du programme se comportera comme suit :

```
$> ./xmas-trees
xmas-trees: error: No input file given !
$> ./xmas-trees xmas-trees-xsmall.in
```

```
Case #1: XXX
Case #2: XXX
...
```

En cas d'absence du fichier, le programme doit terminer en renvoyant une erreur sur `stderr` et avec un code de retour valant `EXIT_FAILURE`.

3. Enfin, pour le reste du parsing, on supposera que nous avons toujours affaire à des fichiers "parfaits", sans erreur de syntaxe, ni oubli de la part de l'utilisateur. Inutile, donc, de passer du temps à essayer d'être robuste lorsque votre programne lit le contenu du fichier. Par contre, on s'attend à ce que vous détectiez l'absence du fichier, ou un problème lors de l'ouverture (en lecture) de celui-ci.

   Pour récapituler les trois premières questions, il vous faut donc, dans l'ordre, vérifier la présence d'un argument sur la ligne de commande, ouvrir le fichier en lecture (et vérifier qu'il a bien été ouvert), récupérer le nombre de cas, puis faire en sorte de récupérer les éléments de chaque cas.
4. Faire une fonction '`bool angle_signedness (point_t p0, point_t p1, point_t p2)`' qui calcule si l'angle $(\texttt{p1}, \texttt{p0}, \texttt{p2})$ est positif ou non. Une astuce est de calculer le déterminant de la matrice donnée par :

$$\begin{vmatrix} \texttt{p1.x} - \texttt{p0.x} & \texttt{p2.x} - \texttt{p0.x} \\ \texttt{p1.y} - \texttt{p0.y} & \texttt{p2.y} - \texttt{p0.y} \end{vmatrix} = (\texttt{p1.x} - \texttt{p0.x}) * (\texttt{p2.y} - \texttt{p0.y}) - (\texttt{p2.x} - \texttt{p0.x}) * (\texttt{p1.y} - \texttt{p0.y}) \quad (1)$$

   Si le résultat est positif ou nul, alors la fonction retourne '`true`' et '`false`' sinon.
5. Pour trouver l'enveloppe convexe de la pépinière, on utilisera l'algorithme du "*Gift Wrapping*" [1] qui consiste à partir d'un point extrême (le plus à gauche par exemple), puis on énumère tous les points qui ne sont pas encore sur l'enveloppe convexe en prenant ceux qui minimisent les angles externes (comme montré sur la figure).

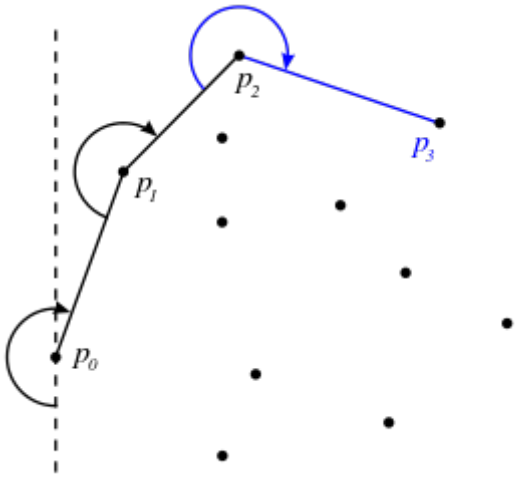

Figure 2 – Algorithm de "*Gift Wraping*" en action.

   On utilisera la fonction précédente avec `p0` le dernier point que l'on a détecté comme faisant partie de l'enveloppe convexe, `p1` le candidat que l'on examine actuellement, et `p2` le candidat retenu jusqu'à présent comme faisant partie de l'enveloppe convexe. Si jamais la fonction `angle_signedness()` retourne '`false`' alors on garde `p1`, sinon on laisse `p2` en tant que candidat.
6. Enfin, il faut calculer la surface de l'enveloppe convexe. Pour cela on utilise la formule du lacet (*Shoelace formula*) [2] :

$$A = \frac{1}{2} \left| \sum_{i=0}^{n-2} (x_i y_{i+1} - x_{i+1} y_i) + (x_{n-1} y_0 - x_0 y_{n-1}) \right| \quad (2)$$

7. Programmez la résolution du problème en utilisant l'algorithme suggéré (ou un autre). La clarté du code, son efficacité ainsi que les commentaires que vous y mettrez seront aussi évalués.

1. `https://en.wikipedia.org/wiki/Gift_wrapping_algorithm`
2. `https://en.wikipedia.org/wiki/Shoelace_formula`

2 Programmation Java (8 points)

Indications : *Pour les questions 1, 2, et 3, rendre un simple fichier ascii contenant les réponses. Pour la question 4, rendre dans une archive "*`tar`*" le code Java avec toutes les classes, et le diagramme de classes associé (en format "*`.dia`*").*

2.1 Questions de cours

Questions

1. Quel est l'inconvénient majeur de la "technique du type maximal" ("*maximal type technique*") ? Donner un exemple concret.
2. Pourquoi est-ce que l'héritage multiple d'interfaces (l'héritage multiple de spécification) n'est pas un problème, alors que l'héritage de classes concrètes (l'héritage multiple d'implémentation) peut l'être ?
3. Dans le chapitre introductif d'un livre important de la programmation objet "*Design Patterns, Elements of Reusable Object-Oriented Software*" par *Gamma et al. Addison-Wesley, 1995*, on trouve la phrase suivante : "*Don't declare variables to be instances of particular concrete classes. Instead, commit only to an interface defined by an abstract class*". Expliquez-la brièvement.

2.2 Algorithme de sélection des méthodes en Java

Questions

1. Considérer les définitions formelles :

```
class A1 {}
class A2 extends A1 {}
class A3 extends A2 {}

class B1 {
   void m(A1 x) {System.out.println("B1.m(A1)");}
   void m(A2 x) {System.out.println("B1.m(A2)");}
}

class B2 extends B1 {
   void m(A1 x) {System.out.println("B2.m(A1)");}
   void m(A2 x) {System.out.println("B2.m(A2)");}
}

class B3 extends B2 {
   void m(A3 x) {System.out.println("B3.m(A3)");}
}
```

   Et leur utilisation suivante :

```
B1 truc = new B3();
A2 arg2 = new A2();
A3 arg3 = new A3();
truc.m(arg2);
truc.m(arg3);
```

   Donner et expliquer son résultat en établissant les étapes successives selon l'algorithme présenté en p.26 de la partie 6 du cours (cf. site).
2. Quelle est l'intention générale de cet algorithme, et pourquoi contient-il deux phases, l'une statique et l'autre essentiellement dynamique ?

2.3 Encapsulation des objets

Questions

1. Considérer les deux classes suivantes qui, comme il est généralement requis, déclarent leurs attributs privés afin de leur assurer une bonne encapsulation :

```
class MutablePoint {
   private double x,y;
   MutablePoint (double x, double y) {
      this.x = x; this.y = y;
   }
   void set (double x, double y) {
      this.x = x; this.y = y;
   }
   double getx() { return x; }
   double gety() { return y; }
}

class Vector2D {
   private MutablePoint p;
   Vector2D(MutablePoint p) {
     this.p = p;
   }
   double norm() {
      return Math.sqrt(p.getx()*p.getx() + p.gety()*p.gety());
    }
}
```

   Soit alors leur utilisation suivante :

```
  MutablePoint p1 = new MutablePoint(3, 4);
  Vector2D v1 = new Vector2D(p1);
  System.out.println(v1.norm());

  p1.set(5, 12);
  System.out.println(v1.norm());
```

   Sa sortie est `5.0` et `13.0`. Autrement dit, on a réussit ici à modifier un vecteur 2D en modifiant un point ! Expliquer cet effet.

2. Proposer une modification du programme pour empêcher cet effet (*Indication : on se contentera de décrire avec précision cette modification ; il n'est pas nécessaire de rendre son implémentation*).

2.4 Simulation d'aquariums

Questions

1. Il s'agit ici d'améliorer quelque peu la simulation d'aquariums vue en cours (*le code est disponible sur le site*). L'idée est de lui ajouter l'approximation suivante de la réalité : les algues produisent de l'oxygène dans l'eau, et la vitesse de nage des poissons est proportionnelle au taux d'oxygène de l'eau dans laquelle ils se trouvent. Ainsi :

   (a). Ajouter une interface `Algae` qui se résumera à déclarer l'unique méthode `double getProducedOxygen()`.

   (b). Implémenter `Algae` avec une classe basique.

   (c). Ajouter une interface `Water` avec au moins une classe où l'on associera la notion d'eau à une liste d'algues, telle que `getTotalOxygenRate` soit issue de la somme de l'oxygène produit par ces algues.

   (d). Implémenter `Water` avec une classe basique où l'on associera la notion d'eau à une liste d'algues, telle que `getTotalOxygenRate` corresponde à la somme de l'oxygène produit par ces algues.

(e). Modifier alors le simulateur (poissons et styles de nage) de manière à ce que le taux d'oxygène issu de l'attribut `Water` intervienne dans la vitesse de nage des poissons.

2. Étendre et modifier le diagramme UML de classes de la simulation (*le fichier* `.dia` *est aussi disponible sur le site du cours*) afin d'y refléter toutes les modifications ci-dessus.

Indications : *Pour importer la solution vue en cours dans Eclipse :*

(a). Sauvegarder le fichier `basic_aquarium.tar` *disponible sur le site.*

(b). Sur Eclipse, créer un nouveau projet Java.

(c). Sur ce projet, avec un clic gauche sur son répertoire `src`*, choisir* `Import`*, puis dans* `General` *choisir* `Archive File`*.*

(d). Rentrer le nom de votre sauvegarde faite au point (1), et importer.