PROG2 : Programmation avancée et C++

Examen du 10 mai 2023 (2 h)

Tous documents interdits à l'exception d'un A4 recto verso manuscrit de votre main. Calculatrices, téléphones, ordinateurs et montres connectées interdites. La correction tiendra compte de la qualité de l'argumentaire et de la présentation. Un code illisible et/ou incompréhensible est un code faux.

★ **Exercice 1 :** Classe `Arbre` (4pts).

On souhaite écrire une classe `Arbre` permettant de représenter un noeud dans un arbre, doté d'une valeur entière ainsi que d'un éventuel fils gauche et un éventuel fils droit. Quand ils existent les fils sont eux-mêmes des `Arbre`s.

▷ **Question 1 :** Écrivez une première version de cette classe où les fils sont représentés par des pointeurs `Arbre*`. Vous doterez votre classe des constructeurs nécessaires et d'une fonction `max()` retournant le maximum de la valeur du noeud courant ainsi que des sous-arbres.

Testez votre code sur l'arbre (12, (23, (14), (34)), (54)).

▷ **Question 2 :** Écrivez une autre version de cet classe sans pointeurs en utilisant des mécanismes issus du C++ un peu plus moderne. Cette version sera également dotée de constructeurs et de la méthode `max()`.

★ **Exercice 2 :** Questions de cours (6pts).

▷ **Question 1** (4pts) **:** Définissez *surcharge de méthode*, *redéfinition de méthode* ainsi que *réécriture de méthode*, en mettant en évidence les similarités et les différences entre ces notions. Quel est l'intérêt du mot-clé `override` dans ce contexte ?

▷ **Question 2** (2pts) **:** Définissez objets automatiques, statiques, temporaires et dynamiques en C++. La réponse attendue est de l'ordre de la dizaine de lignes d'explications.

★ **Exercice 3 :** Design de code (10pts).

Vous trouverez sur l'autre feuille le code d'un calcul de l'enveloppe convexe d'un ensemble de points en utilisant l'algorithme de Graham. Quand on exécute ce code, puis qu'on utilise le code d'affichage en bas de la page (qui utilise SFML), on obtient la figure représentée à droite où les coordonnées de chaque point sont écrites à sa position, et où l'enveloppe convexe est représentée graphiquement.

Ce code utilise quelques calculs mathématiques. Aux lignes 22, 38 et 58, on calcule le module du vecteur étant le produit vectoriel de deux vecteurs $\|\overrightarrow{v_1} \wedge \overrightarrow{v_2}\| = v1.x \times v2.y - v1.y \times v2.x$. Si cette grandeur est positive, alors $\overrightarrow{v_1}$ et $\overrightarrow{v_2}$ sont dans le sens direct ; si elle est nulle, les deux vecteurs sont colinéaires ; dans le dernier cas, $\overrightarrow{v_1}$ et $\overrightarrow{v_2}$ sont dans le sens contraire ($\overrightarrow{v_2}$ et $\overrightarrow{v_1}$ sont donc dans le sens direct). Aux lignes 25, 26 et 27, on trie les points selon la distance euclidienne sans calculer de racine carrée. `distj0` est le carré de la distance entre le point 0 et le point j, tandis que `distj1` est celui de la distance entre le point 0 et le point $j+1$.

Ce code compile en C++, mais il caricature la philosophie du C. Il n'y a aucune classe, les conteneurs et algorithmes de la bibliothèque standard du C++ ne sont pas utilisés, et il n'y a même pas de fonctions déclarées. L'objectif de cet exercice est de refactorer le code du calcul dans la première boîte pour le rendre plus lisible. Le code d'affichage n'est donné que pour référence.

▷ **Question 1 :** Nommez l'algorithme de tri utilisé des lignes 18 à 33.

▷ **Question 2 :** Écrivez une classe `Point` permettant de représenter un point du plan, sous forme de deux doubles x et y. Vous doterez cette classe d'un constructeur prenant une `std::string` comme paramètre et desserialisera les coordonnées du point de la chaîne. Vous pouvez supposer que la chaîne à désérialiser est toujours valide, sans jamais comporter d'erreur de syntaxe.

▷ **Question 3 :** Écrivez le code nécessaire pour que les lignes numérotées 64-68 ci-dessous fassent la même chose que les lignes 12 à 16 du code fourni.

```
57 std::vector<Point> p;
58 std::ifstream input("2023-hull-points.txt");
59
60 std::string text_line;
61 while (std::getline(input, text_line))
62    p.push_back(Point(text_line));
63
64 // Trouve le pivot = point de coordonnées minimales
65 for (int i=1; i<p.size(); i++)
66   if (p[i] < p[0]) {
67     Point t = p[0];   p[0] = p[i];   p[i] = t;
68   }
```

On suppose donnée une classe `Vecteur` reliant deux points et dotée d'un opérateur `int Vecteur::operator*()` calculant le produit scalaire ainsi que d'un opérateur `bool Vecteur::operator<(Vecteur o)` retournant vrai si la norme de l'objet courant est inférieur à celle de l'objet `o`.

▷ **Question 4 :** Écrivez un équivalent des lignes 18 à 33 du code fourni permettant de trier la collection de points par angle polaire ou par distance en cas d'égalité. Vous vous attacherez à utiliser les mécanismes offerts par la STL.

▷ **Question 5** (bonus) **:** Écrivez un filtrage basé sur des ranges C++20 pour remplacer les lignes 35 à 47 du code fourni où l'on supprime les points successifs colinéaires de la collection.

▷ **Question 6 :** Écrivez une classe templatée `Pile` permettant de retrouver le sommet de la pile ainsi que l'élément sous le sommet sans modifier la pile. Votre implémentation peut utiliser la classe `std::vector` pour stocker les données. Il doit s'agir d'une template permettant de stocker n'importe quel type d'éléments.

Votre classe doit implémenter les opérations suivantes : `push` (empile), `pop` (dépile), `top` (renvoyant l'élément au sommet), `prev` (renvoyant l'élément juste avant top), `size` (renvoyant la taille) et l'opérateur d'indexation `operator[]` (renvoyant l'élément à l'index indiqué).

▷ **Question 7 :** Réécrivez l'algorithme de Graham avec les éléments ainsi constitués, en vous inspirant autant que nécessaire du code fourni.

Calcul de l'enveloppe convexe en pseudo-C (à refactorer)

```
int main() {
  int x[] = { 7,-5,2,6,8, 7, 4, 8,0, 3, 6, 0,-8,-8,-8,-9,-2 };
  int y[] = { 8, 6,9,4,6,-2,-6,-7,0,-2,-9,-9,-9,-2, 0, 3, 2 };
  int length = sizeof(x)/sizeof(x[0]);

  // Trouve le pivot = point de coordonnées minimales
  for (int i=1; i<length; i++)
    if (y[i] < y[0] || (y[i] == y[0] && x[i]<x[0])) {
      int tx = x[0];    x[0] = x[i];   x[i] = tx;
      int ty = y[0];    y[0] = y[i];   y[i] = ty;
    }

  // Tri les points par angle polaire (ou par distance si égalité)
  int sorted = 0;
  while (!sorted) {
    sorted = 1;
    for (int j=1; j<length-1;j++) {
      int prod_vect = (x[j]-x[0]) * (y[j+1]-y[0]) - (y[j]-y[0]) * (x[j+1]-x[0]);

      int distj0 = (x[j+0]-x[0])*(x[j+0]-x[0]) + (y[j+0]-y[0])*(y[j+0]-y[0]);
      int distj1 = (x[j+1]-x[0])*(x[j+1]-x[0]) + (y[j+1]-y[0])*(y[j+1]-y[0]);
      if (prod_vect < 0 || (prod_vect == 0 && distj1<distj0)) {
        sorted = 0;
        int tx = x[j];    x[j] = x[j+1];   x[j+1] = tx;
        int ty = y[j];    y[j] = y[j+1];   y[j+1] = ty;
      }
    }
  }

  // Supprime les points co-linéaires par rapport au pivot et leur voisin
  int modified_size=1;
  for (int i=1; i<length; i++) {
    while (i < length-1 && (x[i]-x[0]) * (y[i+1]-y[0]) - (y[i]-y[0]) * (x[i+1]-x[0]) == 0)
      i++;

    if (i!=modified_size) {
      x[modified_size] = x[i];
      y[modified_size] = y[i];
    }
    modified_size++;
  }
  length = modified_size;

  // Initialise l'enveloppe convexe avec les 3 premiers points
  int x2[length] = {x[0], x[1], x[2]};
  int y2[length] = {y[0], y[1], y[2]};
  int top = 2;

  // Ajoute les points de la collection un à un
  for (int i = 3; i < modified_size; i++) {
    // Tant que le point au sommet ne tourne pas à gauche, on dépile
    while (top>1 &&
           (x2[top]-x2[top-1])*(y[i]-y2[top-1]) - (y2[top]-y2[top-1])*(x[i]-x2[top-1]) < 0)
      top --;

    // On ajoute un nouveau point à considérer
    top++;
    x2[top] = x[i];
    y2[top] = y[i];
  }
```

Code d'affichage (pour info)

```
// Chaque point
for (int i=0; i<length; i++) {
  text.setString(std::to_string(i)+" ("
       +std::to_string(x[i])+";"+std::to_string(y[i])+")");
  text.setPosition(x[i]*30+300, y[i]*-30+300);
  window.draw(text);
}
// Enveloppe
for (int i=0; i<top; i++) {
  line[0] = sf::Vector2f(x2[i]*30+300,y2[i]*-30+300);
  line[1] = sf::Vector2f(x2[i+1]*30+300,y2[i+1]*-30+300);
  line[0].color = line[1].color = sf::Color::Red;
  window.draw(line, 2, sf::Lines);
}// Ferme l'enveloppe
line[0] = sf::Vector2f(x2[top]*30+300,y2[top]*-30+300);
line[1] = sf::Vector2f(x2[0]*30+300,y2[0]*-30+300);
line[0].color = line[1].color = sf::Color::Red;
window.draw(line,2, sf::Lines);
```