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)).

Réponse

```cpp
#include <iostream>

class Arbre {
  int val_;
  Arbre* fg_ = nullptr;
  Arbre* fd_ = nullptr;

public:
  Arbre(int val, Arbre* fg, Arbre* fd) : val_(val), fg_(fg), fd_(fd) { }
  Arbre(int val) : val_(val) { }
  int max() {
    int max = val_;
    if (fg_ != nullptr)
      max = std::max(max, fg_->max());
    if (fd_ != nullptr)
      max = std::max(max, fd_->max());
    return max;
  }
};


int main() {
  auto* abr = new Arbre(12, new Arbre(23, new Arbre(14), new Arbre(34))
                        , new Arbre(54));
  std::cout << abr->max() << "\n";
  return 0;
}
```

Fin réponse

▷ **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()`.

Réponse

```
#include <iostream>
#include <memory>

class Arbre;
class Arbre {
  int val_;
  std::unique_ptr<Arbre> fg_;
  std::unique_ptr<Arbre> fd_;

public:
  Arbre(int val, Arbre fg, Arbre fd)
    : val_(val), fg_(std::make_unique<Arbre>(fg)), fd_(std::make_unique<Arbre>(fd))
    { }
  Arbre(int val) : val_(val) { }
  Arbre(Arbre& o):val_(o.val_) {
    fg_.swap(o.fg_);
    fd_.swap(o.fg_);
  }
  int max() {
    int max = val_;
    if (fg_)
      max = std::max(max, fg_->max());
    if (fd_)
      max = std::max(max, fd_->max());
    return max;
  }
};


int main() {
  auto abr = Arbre(12, Arbre(23, Arbre(14), Arbre(34))
                     , Arbre(54));
  std::cout << abr.max() << "\n";
  return 0;
}
```

Fin réponse

★ **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 ?

Réponse

Les trois mécanismes impliquent des méthodes de même nom, que ce soit dans une classe donnée (cas de la surcharge/overload) ou entre la classe mère et la classe fille (cas de la redéfinition/override ou de la réécriture/overwrite).

La surcharge consiste à définir une autre méthode de même nom, avec des paramètres différents. Un usage très courant consiste à surcharger les opérateurs pour redéfinir leur comportement pour de nouveaux types de paramètres.

Si une classe fille réutilise un nom de méthode déjà utilisé dans un ancêtre, il peut s'agir d'une redéfinition/override si le prototype est inchangé ET si la méthode de l'ancêtre est marquée `virtual`, ou d'une réécriture dans les autres cas.

	Signature	**Scope**	**Comportement**
`override`	Inchangée	Mère-fille	Remplacement dynamique
`overload`	Modifiée	Même classe	Co-existence
`overwrite`	Inchangée mais méthode pas `virtual` dans mère ou signature modifiée (même si marquée `virtual`)	Mère-fille	Remplacement statique

Le mot-clé `override` permet de détecter certaines erreurs classiques lorsque l'on souhaite mettre en place une redéfinition (ce qui arrive tout le temps dans un programme orienté objet puisque c'est ainsi qu'on fait de la liaison dynamique), mais qu'une typo ou autre erreur empêche la mise en place de cette redéfinition. Le compilateur va par exemple détecter si la méthode ancêtre n'est pas marquée `virtual` ou si le prototype est modifiée par rapport à l'ancêtre, au lieu de faire une réécriture dont le comportement est déroutant quand on attendait une redéfinition (choix de la fonction à la compilation, au lieu d'un choix dynamique à l'exécution).

Fin réponse

▷ **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.

Réponse

- Objet automatique : pour les variables locales. Créés à la demande quand on entre dans le bloc, et détruits en sortant
- Objets statiques : marqué avec le modifieur static, créés avant le main, et détruit automatiquement en fin de programme
- Objets temporaire : si on fait un appel explicite au constructeur dans une expression `a = Point(1,4)`, le compilateur crée un objet temporaire qui sera détruit dès qu'il ne sera plus utile. L'affectation d'objet temporaire n'utilise pas le même constructeur que l'initialisation d'un objet automatique.
- Objets dynamiques : avec `new` et `delete` à la place de malloc et free pour mettre sur le tas. Attention à tout bien libérer à la fin.

Fin réponse

★ **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 $\|\overrightarrow{v1} \wedge \overrightarrow{v2}\| = v1.x \times v2.y - v1.y \times v2.x$. Si cette grandeur est positive, alors $\overrightarrow{v1}$ et $\overrightarrow{v2}$ sont dans le sens direct ; si elle est nulle, les deux vecteurs sont colinéaires ; dans le dernier cas, $\overrightarrow{v1}$ et $\overrightarrow{v2}$ sont dans le sens contraire ($\overrightarrow{v2}$ et $\overrightarrow{v1}$ 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.

Réponse

C'est un glorieux tri à bulle.

Fin réponse

▷ **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.

Réponse

```
12 struct Point { // struct = classe dont tous les champs sont publiques
13   int x, y;
14
15   Point(std::string line) {
16     std::istringstream in(line);
17     in >> x;
18     in >> y;
19   }
```

Fin réponse

▷ **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   }
```

Réponse

```
20   // On ajoute cet opérateur dans la classe Point
21   bool operator<(const Point other) {
22     return y < other.y || (y == other.y && x < other.x);
23   }
24 };
```

Fin réponse

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`.

Réponse

```
26 struct Vecteur {
27   Point a, b;
28   Vecteur(Point a_, Point b_) : a(a_), b(b_) {}
29   float operator*(const Vecteur other) {
30     // v1 x v2 = v1.x * v2.y - v1.y *v2.x
31     return (this->b.x-this->a.x) * (other.b.y-other.a.y)
32          - (this->b.y-this->a.y) * (other.b.x-other.a.x);
33   }
34   bool operator<(const Vecteur other) {
35     int dthis = (this->b.x-this->a.x)*(this->b.x-this->a.x)
36               + (this->b.y-this->a.y)*(this->b.y-this->a.y);
37     int dother= (other.b.x-other.a.x)*(other.b.x-other.a.x)
38               + (other.b.y-other.a.y)*(other.b.y-other.a.y);
39     return dthis < dother;
40   }
41 };
```

Fin réponse

▷ **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.

Réponse

```
70 // Tri les points par angle polaire (ou par distance si égalité)
71 Point pivot = p[0];
72 std::sort(p.begin()+1, p.end(), [pivot] (Point a, Point b) {// +1 pour laisser le pivot
73   Vecteur u (pivot, a);
74   Vecteur v (pivot, b);
75   int prod_vect = u * v;
76   if (prod_vect < 0 || (prod_vect == 0 && v < u))
77     return false; // La lambda retourne faux s'il faut inverser les éléments,
78   return true;    //   et vrai si c'est déjà dans l'ordre.
79 });
```

Fin réponse

▷ **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.

Réponse

```
87 // Supprime les points co-linéaires par rapport au pivot et leur voisin
88 auto filtered = p
89   | std::ranges::views::drop(3)
90   | std::ranges::views::filter([ pivot=p[0], prev=p[1] ](Point pi){
91     static Point last = prev; // Ruse: utiliser une variable statique
92     auto res = Vecteur(pivot, last) * Vecteur(pivot, pi) != 0;
93     last = pi; // Sauvegarde du point courant comme futur point précédent
94     return res;
95   });
```

Fin réponse

▷ **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).

Réponse

```
template <class T>
class Pile {
  std::vector<T> data;
public:
  Pile(const std::vector<T> input) : data(input) {}
  void push(T elem) {  data.push_back(elem);  }
  void pop() { data.pop_back(); }
  T top() { return data.back();  }
  T prev() {  return data.at(data.size()-2);  }
  int size() { return data.size(); }
  T operator[](int i) { return data[i]; }
};
```

Fin réponse

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

Réponse

Cette correction suppose que la question optionnelle est faite. Si non, il faut faire une boucle `for` classique avec un index, afin de sauter les 3 premiers éléments du tableau `p`.

```
// Ajoute les points de la collection un à un
for (auto pi : filtered) {
  // Tant que le point au sommet ne tourne pas à gauche, on dépile
  while (hull.size()>1 &&
         Vecteur(hull.prev(), hull.top()) * Vecteur(hull.prev(), pi) < 0) {

    hull.pop();
  }

  // On ajoute un nouveau point à considérer
  hull.push(pi);
}
```

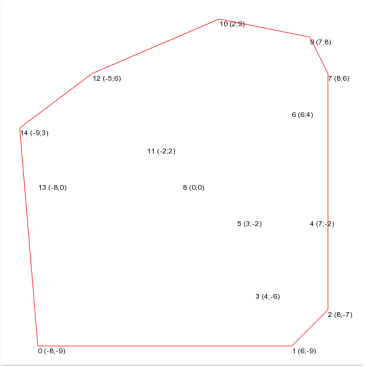
Fin réponse

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);
```



Réponse

```
// g++ 2023-hull-correction.cpp `pkg-config sfml-all -cflags --libs` --std=c++20 && ./a.out
#include <SFML/Graphics.hpp>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <vector>
#include <stack>
#include <iostream>
#include <sstream>
#include <fstream>
#include <ranges>
struct Point { // struct = classe dont tous les champs sont publiques
  int x, y;

  Point(std::string line) {
    std::istringstream in(line);
    in >> x;
    in >> y;
  }
  // On ajoute cet opérateur dans la classe Point
  bool operator<(const Point other) {
    return y < other.y || (y == other.y && x < other.x);
  }
};

struct Vecteur {
  Point a, b;
  Vecteur(Point a_, Point b_) : a(a_), b(b_) {}
  float operator*(const Vecteur other) {
    // v1 x v2 = v1.x * v2.y - v1.y *v2.x
    return (this->b.x-this->a.x) * (other.b.y-other.a.y)
         - (this->b.y-this->a.y) * (other.b.x-other.a.x);
  }
  bool operator<(const Vecteur other) {
    int dthis = (this->b.x-this->a.x)*(this->b.x-this->a.x)
              + (this->b.y-this->a.y)*(this->b.y-this->a.y);
    int dother= (other.b.x-other.a.x)*(other.b.x-other.a.x)
              + (other.b.y-other.a.y)*(other.b.y-other.a.y);
    return dthis < dother;
  }
};

template <class T>
class Pile {
  std::vector<T> data;
public:
  Pile(const std::vector<T> input) : data(input) {}
  void push(T elem) {  data.push_back(elem);  }
  void pop() { data.pop_back(); }
  T top() { return data.back();  }
  T prev() {  return data.at(data.size()-2);  }
  int size() { return data.size(); }
  T operator[](int i) { return data[i]; }
};

int main() {
   std::vector<Point> p;
   std::ifstream input("2023-hull-points.txt");

   std::string text_line;
   while (std::getline(input, text_line))
      p.push_back(Point(text_line));

   // Trouve le pivot = point de coordonnées minimales
   for (int i=1; i<p.size(); i++)
     if (p[i] < p[0]) {
       Point t = p[0];   p[0] = p[i];   p[i] = t;
     }

   // Tri les points par angle polaire (ou par distance si égalité)
```

```
71    Point pivot = p[0];
72    std::sort(p.begin()+1, p.end(), [pivot] (Point a, Point b) {// +1 pour laisser le pivot
73      Vecteur u (pivot, a);
74      Vecteur v (pivot, b);
75      int prod_vect = u * v;
76      if (prod_vect < 0 || (prod_vect == 0 && v < u))
77        return false; // La lambda retourne faux s'il faut inverser les éléments,
78      return true;    //   et vrai si c'est déjà dans l'ordre.
79    });
80
81
82    printf("Sorted : ");
83    for (int i=0;i<p.size(); i++)
84      printf("%d %d|", p[i].x,p[i].y);
85    printf("\n");
86
87    // Supprime les points co-linéaires par rapport au pivot et leur voisin
88    auto filtered = p
89      | std::ranges::views::drop(3)
90      | std::ranges::views::filter([ pivot=p[0], prev=p[1] ](Point pi){
91        static Point last = prev; // Ruse: utiliser une variable statique
92        auto res = Vecteur(pivot, last) * Vecteur(pivot, pi) != 0;
93        last = pi; // Sauvegarde du point courant comme futur point précédent
94        return res;
95      });
96
97    // Initialise l'enveloppe convexe avec les 3 premiers points
98    Pile<Point> hull ({p[0], p[1], p[2]});
99
100   // Ajoute les points de la collection un à un
101   for (auto pi : filtered) {
102     // Tant que le point au sommet ne tourne pas à gauche, on dépile
103     while (hull.size()>1 &&
104            Vecteur(hull.prev(), hull.top()) * Vecteur(hull.prev(), pi) < 0) {
105
106       hull.pop();
107     }
108
109     // On ajoute un nouveau point à considérer
110     hull.push(pi);
111   }
```

Fin réponse