Université de Nice-Sophia Antipolis — POLYTECH
ELSE4-FISE — 2025–2026

Examen de C++ – COO

Durée : 1h00
Aucun document autorisé
Mobiles interdits

On veut représenter des *quadrilatères convexes* : *rectangles*, *carrés* et *trapèzes*. Rappel : un quadrilatère est dit *convexe* si tout segment joignant deux quelconques de ses points est entièrement contenu dans son intérieur. Voir annexe.

▶ 1. Dans le fichier `QuadrilatèreConvexe.hpp`, écrivez la classe *abstraite* `QuadrilatèreConvexe` qui possède :

- une variable membre protégée, `lesCôtés` de type tableau de réels (**double**) qui contient les longueurs (> 0) des 4 côtés du quadrilatère convexe courant ;
- un constructeur pour initialiser la variable membre à partir de 4 réels (**double**) ;
- la méthode virtuelle *pure* `surface` qui renvoie la surface du quadrilatère convexe courant ;
- la méthode `périmètre` qui renvoie le périmètre du quadrilatère convexe courant ;

Vous n'écrirez que les en-têtes du constructeur et des méthodes.

Question sur 2 pts

```
class QuadrilatèreConvexe {
protected:
  double lesCôtés[4];
public:
  // le constructeur
  QuadrilatèreConvexe(const double c1, const double c2,
                      const double c3, const double c4);
  // les méthodes
  double périmètre() const;
  virtual double surface() const =0;
};
```

.....

▶ 2. Dans le fichier `QuadrilatèreConvexe.cpp`, écrivez le contructeur `QuadrilatèreConvexe`.

Question sur 2 pts

```
QuadrilatèreConvexe::QuadrilatèreConvexe(const double c1, const double c2,
                                         const double c3, const double c4)
 : lesCôtés{c1, c2, c3, c4}
{   assert(c1>0 && c2>0 && c3>0 && c4>0); }
```


.....

▶ 3. Est-il possible d'écrire la méthode `périmètre` de la classe abstraite `QuadrilatèreConvexe` dans le fichier `QuadrilatèreConvexe.cpp` ? Expliquez. Si oui, écrivez cette méthode.

Question sur 2 pts

Oui bien sûr, puisque c'est la somme des 4 cotés contenus dans le tableau **this**`->lesCôtés`.

```
double QuadrilatèreConvexe::périmètre() const {
  double p=0.0;
  for (double x : this->lesCôtés)
    p+=x;
  //
  return p;
}
```

.....

▶ 4. Écrivez les classes `Rectangle`, `Carré` et `Trapèze`. Vous écrirez le code des constructeurs et des méthodes directement dans les classes. *Réfléchissez-bien aux relations entre les différentes classes, incluant `QuadrilatèreConvexe`.*

Question sur 4 pts

```
typedef QuadrilatèreConvexe super;

/*
 *   Classe T R A P È Z E
 */
class Trapèze : public QuadrilatèreConvexe {
protected:
  double hauteur;
public:
  Trapèze(const double c1, const double c2,
          const double c3, const double c4, const double h) :
    QuadrilatèreConvexe(c1, c2, c3, c4), hauteur(h) { assert(h>0); };

  virtual double surface() const override {
    return (super::lesCôtés[0]+super::lesCôtés[1])*this->hauteur/2;
  }
};
```

```
/*
 *   Classe R E C T A N G L E
 */
class Rectangle : public QuadrilatèreConvexe {
public:
  Rectangle(const double largeur, const double longueur) :
    QuadrilatèreConvexe(largeur, longueur, largeur, longueur) {};

  virtual double surface() const override {
```

```
    return super::lesCôtés[0]*super::lesCôtés[1];
  }
};
```

```
/*
 *   Classe C A R R É
 */
class Carré : public Rectangle {
public:
  Carré(const double c) : Rectangle(c,c) {};
};
```

.....

► 5. Écrivez le fragment de code qui :
   — déclare la variable `vq`, un vecteur de pointeurs sur `QuadrilatèreConvexe` initialisée avec un rectangle 12×3.5, un carré de coté 12.5, et un trapèze $4 \times 6 \times 5 \times 13$ et de hauteur 4 ;
   — écrit sur la sortie standard les surfaces de chacun des quadrilatères contenus dans le vecteur `vq`.

Question sur 2 pts

```
// déclaration du vecteur
std::vector<QuadrilatèreConvexe *> vq= {
  new Rectangle(12, 3.5),
  new Carré(12.5),
  new Trapèze(4, 6, 5, 13, 4)
};
// on affiche sur la S/S les surfaces des quadrilatères convexes
for (const auto &q : vq)
   std::cout << q->surface() << std::endl;
```

et pour ceux/celles qui ont voulu utiliser les *smart pointers* :

```
#include <memory>
// déclaration du vecteur
std::vector<std::shared_ptr<QuadrilatèreConvexe>> vq = {
   std::make_shared<Rectangle>(12, 3.5),
   std::make_shared<Carré>(12.5),
   std::make_shared<Trapèze>(4, 6, 5, 13, 4)
};
// on affiche sur la S/S les surfaces des quadrilatères convexes
for (const auto &q : vq)
  std::cout << q->surface() << std::endl;
```

.....

► 6. Dans le fragment de code précédent, *expliquez clairement* les mécanismes de programmation objet qui sont mis en jeu.

Question sur 3 pts

Tout d'abord, il y a le *polymorphisme*. En POO, le polymorphisme s'applique à travers le graphe d'héritage. Les éléments du vecteur `vq` sont des pointeurs sur `QuadrilatèreConvexe` ce qui permettra de mettre dans le vecteur des objets issus de classes dérivées de `QuadrilatèreConvexe`. Ensuite, il y a la *liaison dynamique*. La méthode `surface` appliquée est celle de l'objet désigné à l'exécution par la variable `q` et non pas celle du type de `q` à la compilation.

.....

► 7. D'expliquez de façon claire et synthétique, d'une part, la notion *générale* d'itérateur, et d'autre part, telle qu'elle existe en C++.

Question sur 2 pts

D'une façon générale, un itérateur permet un accès séquentiel *uniforme* à tous les éléments d'une structure de données (*i.e. conteneur*), de façon *indépendante* de sa représentation interne.

En C++, un itérateur offre la fonctionnalité générale précédente, mais en fait, c'est simplement un *pointeur* qui définit une *position* d'un élément dans un conteneur qu'on peut incrémenter (ou décrémenter).

.....

On veut écrire la classe `ItérateurÀPas` paramétrée par l'itérateur d'un conteneur quelconque. Le but de cette classe est de parcourir le conteneur selon un `pas` grâce une méthode `foreach` qui renvoie une `std::string` qui contient les valeurs parcourues du conteneur. Voici un exemple d'utilisation :

```
std::vector<char> vect = {'a','b','c','d','e','f','g','h','i','j','k','l','m','n'};
ItérateurÀPas<std::vector<char>::iterator> valeurs(vect.begin(), vect.end(), 3);
std:: cout << valeurs.foreach(); // => a d g j m
```

► 8. Écrivez la classe `ItérateurÀPas` qui possède :
   — 3 variables membres privées : l'itérateur de début, de fin et le pas ;
   — un constructeur pour initialiser les variables membres ;
   — la méthode `foreach` qui renvoie une chaîne de caractères qui contient les éléments du conteneur parcourus selon le pas.

```
template <typename Iterator>
class ItérateurÀPas {

private:
    ....
    ...
public:

  ItérateurÀPas(.....
  {}

  std::string foreach() const {
    ...
    ...
```

```
  }
};
```

Question sur 4 pts

```
template <typename Iterator>
class ItérateurÀPas {

private:
  Iterator begin;
  Iterator end;
  int pas;

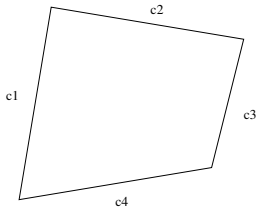
public:

  ItérateurÀPas(const Iterator &b, const Iterator &e, const int p=0)
    : begin(b), end(e), pas(p) { assert(p>=0); }

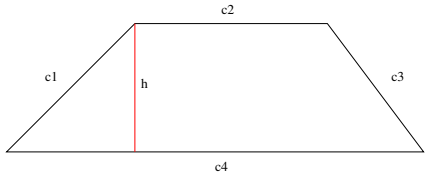
  /*
   * Rôle : renvoie une std::string qui contient les éléments
   *         du conteneur parcouru entre this->begin et this->end
   *         selon le pas this->pas
   */
  std::string foreach() const {
    std::ostringstream s;
    for (Iterator it=this->begin; it<this->end; it+=this->pas)
      s << *it << " ";
    //
    s << "\n";
    return s.str();
  }
};
```

.....

1 Annexe

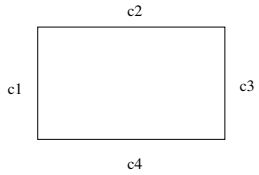


Quadrilatère convexe

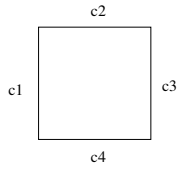


Trapèze

La surface d'un trapèze est égale à $\frac{(c2+c4)\times h}{2}$



Rectangle c1=c3 et c2=c4



Carré c1=c2=c3=c4