Sup Galilée
Mathématiques Appliquées et Calcul Scientifique
Deuxième année
Année universitaire 2018–2019

Langage C++(G4SCCP)
Cours : X. Juvigny
TPs : S. Landier
samlandier@gmail.com

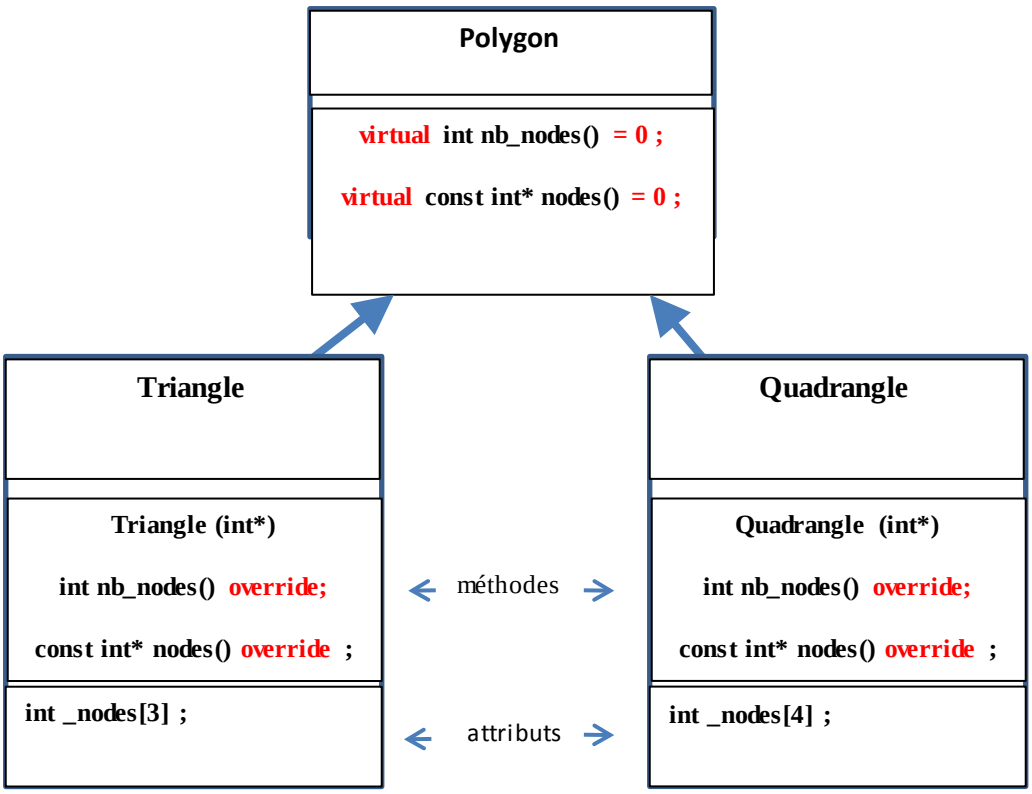
Travaux pratiques N°4

Les bases du langage C++

https://github.com/Macs1718/Promotion_2020
https://en.cppreference.com/w/
https://stackoverflow.com/

Exercice 14 (classes Triangle, Quadrangle et interface Polygon)

L'objectif de cet exercice est de mettre en place la hiérarchie de classe pour les éléments surfaciques de maillage permettant de spécialiser les calculs et d'abstraire les algorithmes.



1) Création de la hiérarchie de classe

Il s'agit ici de créer 3 classes :

- Classe Polygon : classe *abstraite* (non instantiable) qui définit uniquement une interface pour les éléments surfaciques. Créer uniquement le fichier Polygon.h
  Elle ne possède pas d'attributs, pas de constructeurs, et possède au moins une méthode virtuelle pure (signature commençant par le mot clé « virtual » et se terminant par « =0 » pour dire que la classe n'implémente pas cette fonction mais oblige les classe héritantes à l'implémenter).

- Classe Triangle : classe *concrète* (i.e. possède au moins un constructeur et définit le destructeur) qui hérite de la classe Polygon. Créer les fichiers Triangle.h et Triangle.cpp.
  - Elle définit un constructeur qui prend un pointeur d'entier en paramètre. L'implémentation consiste à renseigner l'attribut « _nodes » à partir de ce pointeur.
  - Définir la méthode virtuelle nb_nodes qui retourne le nombre de sommets.
  - Définir la méthode nodes() qui retourne un pointeur vers l'attribut _nodes.
- Classe Quadrangle : même chose que la classe Triangle, sauf qu'on a 4 sommets.

2) Mise en action du polymorphisme
Créer un fichier exo14.cpp, y définir une fonction main dans laquelle :

a) Un objet *t* de type Triangle est créé (choisir n'importe quoi pour les données du constructeur)
b) Idem avec un objet de type Quadrangle : *q*
c) Définir 2 pointeurs *p1* et *p2* de type Polygon, affecter les adresses respectives des objets t et q.
d) Appeler la fonction nb_nodes() via les pointeurs p1 et p2 et additionner les 2 valeurs. Vérifier que l'on obtient bien 7.

⇨ *C'est ce que l'on appelle le polymorphisme : un pointeur de type Polygon* peut pointer en réalité vers un objet d'une classe dérivée : depuis ce pointeur, on ne peut appeler que des méthodes déclarées dans Polygon, mais si l'une d'elle est surdéfinie dans la classe dérivée, c'est cette dernière qui est appelée.*

3) Chainage des destructeurs

a) Ajouter un affichage de sortie dans chacun des destructeurs (de Polygon, Triangle et Quadrangle). Par exemple pour Triangle :

```
std ::cout << «Triangle destructor call » << std ::endl ;
```

b) Compilez et exécutez le programme.
- Quel est l'ordre d'appel des destructeurs entre classe de base et classe dérivée ?
- L'objet Triangle est instancié avant l'objet Quadrangle. Qu'en est-il de l'ordre de leur destruction ? Quelle règle générale pouvez-vous en tirer quant à l'ordre de création par rapport à l'ordre de destruction des objets ?

c) Instancier maintenant dynamiquement (avec new) un autre objet t2 de type triangle et affectez son adresse à un pointeur *pPoly* de Polygon. Appelez tout de suite après l'opérateur delete sur pPoly. Est-ce que la destruction de t2 vous semble correcte ? Pourquoi ? Comment y remédier ?

Exercice 15 (« elements factory » : création des objets Triangles et Quadrangle à partir des tableaux Array<int>)

Cet exercice nécessite d'avoir défini correctement la classe template Array (exo 13).

Télécharger le fichier catT3Q4.mesh depuis le github. Ce fichier contient une version du modèle de chat avec des éléments mixtes, triangles et quadrangles.

Lorsqu'on effectue l'instruction suivante :

```
medith::read("catT3Q4.mesh", crd, cntE2, cntT3, cntQ4, cntTH4, cntHX8);
```

- cntE2 contient en sortie tous les segments contenus dans le fichier
- cntT3 contient en sortie tous les triangles contenus dans le fichier
- cntQ4 contient en sortie tous les quadrangles contenus dans le fichier
- cntTH4 contient en sortie tous les tétraèdres contenus dans le fichier
- cntHX8 contient en sortie tous les hexaèdres contenus dans le fichier

Avec le fichier « catT3Q4.mesh », seuls les containers cntT3 et cntQ4 contiendront des données.

L'objectif de cet exercice est de convertir ces données dans un formalisme objet et de gérer le cycle de vie de ces objets, c'est-à-dire de fournir les fonctions de création et de destructions des objets triangles et quadrangles correspondants (cf. 1) et 2)). Afin de vérifier l'exactitude de votre implémentation, on fera comme à l'exercice précédent un calcul simple sur les sommets grâce au polymorphisme (cf. 3)).

Définir le main suivant et créer les 3 fonctions (en gras) utilisées et remplacer le commentaire en vert par une procédure de vérification :

```
int main()
{
 Array<int> cntE2(2), cntT3(3), cntQ4(4), cntTH4(4), cntHX8(8);
 Array<double> crd(3);

 medith::read("catT3Q4.mesh", crd, cntE2, cntT3, cntQ4, cntTH4, cntHX8);

 std::vector<Polygon*> pgs;
 create_polygons(cntT3, pgs);
 create_polygons(cntQ4, pgs);

 int total_nodes = count_nodes(pgs);
```

```
 // VERIFIER  ICI LA VALEUR DE total_nodes

 destroy_polygons(pgs);

 return 0;
}
```

Notes :

1) create_polygons
   Une fonction qui parcourt un container *cnt* de type Array<int> est qui créé un objet (opérateur new) de type Triangle ou de type Quadrangle selon le stride de cnt et qui stocke en sortie un pointeur de type Polygon sur l'objet créé, qu'il soit un triangle ou un quadrangle.

Voici la signature de la fonction :

```
void create_polygons(const Array<int>& cnt, std::vector<Polygon*> & pgs) ;
```

2) destroy_polygons
   Une fonction de nettoyage qui parcourt la liste des polygones créés et qui les détruit (appel à l'opérateur delete pour chaque élément de la liste)

```
void destroy_polygons(const Array<int>& cnt, std::vector<Polygon*> & pgs) ;
```

3) count_nodes
   Cette fonction parcourt la liste des polygones en entrée, appelle pour chaque polygone la fonction nb_nodes(), et retourne en sortie la valeur cumulée de ces appels.

4) Procédure de vérification :
   Il faudra s'assurer que la valeur retournée est bien la valeur attendue :
   Nombre de triangle x 3 + nombre de quadrangles x 4