Corrigé du TP9

Disclaimer : Ce corrigé n'a pas pour but de vous fournir un détail des algorithmes répondant aux questions de l'énoncé, mais de vous résumer les notions clés dont vous aurez besoin pour résoudre ces exercices, les avertissements vis à vis de certaines erreurs classiques et les détails des passages les plus techniques, voire des précisions si vous souhaitez aller plus loin sur ces questions et plus généralement de synthétiser dans un même document la plupart des commentaires fait en TP afin qu'ils puissent profiter à tous. Pour obtenir un corrigé détaillé, il vous faudra donc vous référer à ce que vous avez pu faire en TP ou reprendre vous même les exercices à partir des notions présentées dans le présent corrigé pour écrire les algorithmes qui vous manquent.

En cas de difficultés, **n'hésitez pas à me contacter**, nous pourrons en discuter soit par mail soit en début de séance suivante. Ce corrigé n'a pour but que d'être une base de travail pour vous et de servir de complément par rapport à ce que vous avez fait en séance, il n'est en aucun cas fait pour se suffire à lui même.

Déclaration d'une classe dérivée

* Une classe dérivée est une sous-classe appelée aussi classe fille d'une autre classe appelée classe mère. Une classe fille héritera de tous les attributs et méthodes de sa classe mère, l'inverse étant faut.
* Pour déclarer une classe dérivée, on utilise la syntaxe :

  class classeFille : **public** classeMere

  {

  nouveaux attributs et méthodes ;

  } ;

  Les nouveaux attributs et méthodes sont définies comme dans le cas d'une classe normal, une classe dérivée étant une classe en tant que tel.
* Lorsque l'on utilise des classes dérivées, il est conseillé d'utiliser la portée **protected** au lieu de **private**.

  Protected permet en effet de bloquer l'utilisation des attributs hors des méthodes tout en autorisant à ce qu'ils soient utilisés dans les méthodes des classes filles ce qui est souvent très utile.

Objets de classe dérivée

* Les classes dérivées possèdent également des constructeurs. Il est conseillé et souvent bien plus pratique d'utiliser les constructeurs de la classe mère dans le constructeur des classes filles.
* En considérant deux attributs définis dans la classe mère et hérités à la classe fille et deux attributs définis dans la classe fille uniquement, on aurait un constructeur défini de la façon suivante :

  classeFille : :classeFille(type attributMere1, type attributMere2, type attributFille1, type attributFille2) : classeMere(attributMere1, attributMere2)

  {

  instructions pour la construction des attributs de la classe fille ;

  }

  Il est également possible de définir les attributs de la classe fille à la suite du constructeur de la classe mère comme vous en avez l'habitude et de ne rien mettre entre les accolades.
* Il est possible de redéfinir une méthode de la classe mère dans la classe fille. Dans ce cas, le programme appliquera la méthode en fonction de la classe à laquelle appartient l'argument.

  On dit alors que l'on a masqué la méthode de la classe mère.

Usage des pointeurs

* Il est possible d'utiliser directement les méthodes masquées dans la fonction main(). Cependant, lorsque l'on utilise ces méthodes masquées dans des fonctions que l'on utilise dans la fonction main(), le masquage n'est plus fonctionnel et on n'utilise plus que la version de la classe mère. Pour éviter ça, on peut passer

par les pointeurs et les méthodes virtuelles.

* Il est possible d'affecter une variable de type d'une classe fille dans une case mémoire pointée par un pointeur de la classe mère.

  Il est donc possible de créer des tableaux de type tableau de pointeurs de classe mère et de lui affecter dans certaines cases des pointeurs de classe fille.

* Pour que les méthodes s'appliquent à la classe appropriée sans surprise, il vous faudra les rendre virtuelle en déclarant leur entête de la façon suivante :

  virtual type nomFonction(type nomVariable) ;

  Il ne faut pas indiquer dans le fichier .cpp mais uniquement dans le fichier .h

* En appliquant ces méthodes et les fonctions utilisant ces méthodes à des pointeurs, elles utiliseront bien la bonne définition en fonction du type de l'argument.

  Il est également possible d'obtenir le même fonctionnement en utilisant, non pas des pointeurs, mais des passages par référence avec des méthodes virtuelles.

* Un constructeur ne peut pas être rendu virtuel. À l'inverse, un destructeur doit être rendu virtuel sinon il ne fonctionnera pas correctement.

* Lorsqu'une méthode virtuelle n'a pas de sens à être définie sur la classe mère, il est possible de la rendre virtuelle pure. Dans ce cas, elle ne s'appliquera pas aux éléments de la classe mère. Pour ce faire, il faut définir l'entête dans la fonction .h comme :

  virtual type nomFonction(type nomVariable)=0 ;

  Dans ce cas, vous n'avez pas besoin de définir la méthode dans le fichier .cpp

  ATTENTION. Une classe possédant une méthode virtuelle pure devient abstraite et on ne peut pas créer d'objet d'une classe abstraite.

  Cependant, il reste possible de créer un pointeur vers une case contenant un objet de type cette classe abstraite et de lui affecter ensuite un élément d'une classe fille.

  Cette classe est alors utilisable pour manipuler des vecteurs constitués d'éléments de différentes classes fille.

Quelques conseils pour retravailler le TP

* Ce TP est le dernier TP à porter sur de nouvelles notions. Il n'est vraiment pas facile, utilisant des notions déjà assez subtiles. S'il vous semble obscur dans un premier temps, c'est plutôt normal, il n'y a pas lieu de s'inquiéter.

* Il est important, avant de reprendre ce TP, de bien maîtriser les autres notions. Je vous conseille notamment d'être à l'aise sur la notion de pointeur et de classe.

* Le point le plus complexe porte sur le polymorphisme, globalement l'usage des méthodes virtuelles, concentrez vous dans un premier temps sur l'héritage et l'utilisation des classes filles avant de vous intéresser à cette notion qui est plus délicate.