Corrigé du TP8

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.

Constructeur par copie

* Le constructeur par copie permet de créer un élément d'une classe comme une copie d'un élément.

  Ce constructeur est créé automatiquement avec la classe et s'utilise de la façon suivante :

  nomClasse nouvelleVariable(variableACopier) ;

* Il est également possible de définir le constructeur par copie de la même façon que n'importe quelle constructeur en utilisant un entête de la forme suivante :

  nomConstructeur (nomClasse const& nomVariable) ;

  Attention, on n'utilise comme nom du constructeur le nom de la classe.

* Lorsque la classe utilise des pointeurs, il est important de ne pas utiliser le constructeur de copie par défaut.

  En effet, si on utilise le constructeur de copie par défaut, il se contentera de copier la valeur du pointera qui pointera donc vers la même case mémoire que la variable copiée alors qu'on aimerait avec un pointeur qui pointe vers une autre case qui a la même valeur que celle de la variable copiée.

  Il faut alors définir à la main le constructeur par copie et spécifier dans les instructions que l'on veut que le pointeur de la nouvelle variable est une allocation dynamique par la commande :

  nomAttribut = **new** typeAtrribut ;

  Puis copier le contenu de la case mémoire dans la nouvelle case mémoire crée précédemment par la commande :

  ∗nomAttribut=∗nomVariable.nomAttribut ;

Destructeur

* Le destructeur est une méthode qui permet de supprimer une variable, ce qui libère la case mémoire qu'elle utilisait.
* Un destructeur n'est utile que dans le cas où la classe que l'on considère utilise des allocation de mémoire dynamique (par l'intermédiaire de la commande **new**).
* Un destructeur commence par un tilde , ne renvoie aucune valeur pas même void et il ne prend aucun paramètre.
* Son entête est nomClasse() ;
* On le définit dans le fichier nomClasse.cpp par :

  ```
  nomClasse : : nomClasse()
  {
  instructions ;
  }
  ```

  Dans les instructions, figureront des **delete** pour détruire les variables allouées dynamiquement.

Opérateur d'affectation

* Lorsque l'on crée une classe, un opérateur d'affectation par défaut est défini par c++. Cependant, cet opérateur d'affectation par défaut possède les mêmes limitations que le constructeur de copie lorsque l'on manipule des pointeurs dans une classe.
* L'entête de la surcharge de l'opérateur d'affectation est :

  nomClasse & operator=(const nomClasse & nomVariable) ;
* Il faut alors copier chaque composante qui n'est pas un pointeur et affecter une copie du contenu de la case mémoire de la variable à copier dans la variable copie dans le cas d'un attribut de type pointeur.

Quelques conseils pour retravailler le TP

* Ce TP est vraiment délicat, il est nécessaire pour le comprendre de bien maîtriser les notions de pointeurs et de classe. N'hésitez donc pas à reprendre les TP6 et 7, ainsi que les cours et TD sur ces notions si vous en avez besoin.
* L'intérêt de ces notions est d'utiliser le plus efficacement la mémoire. Il vous sera donc nécessaire de les maitriser si vous souhaitez utiliser C++ pour du traitement de données par exemple.