Corrigé du TP4

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.

Entête d'une fonction

* Pour définir une fonction la syntaxe est :

  ```
  type nomFonction (type nomVariable)
  {
      commandes;
  }
  ```

  Le type devant le nom de la fonction est le type de la variable que votre fonction va renvoyer, c'est le type de l'espace d'arrivée de votre fonction.

  Le type de la variable entre parenthèse après le nom de la fonction est le type de la variable utilisée par la fonction, c'est le type de l'espace de départ de votre fonction.

  Faites bien la différence entre les deux.

* Une fonction peut également prendre plusieurs variables, il faudra alors indiquer pour chaque variable son type.

  Une fonction peut également ne prendre aucun argument, dans ce cas, on ne met rien entre les parenthèses. C'est par exemple le cas de la fonction size().

  Une fonction peut aussi ne rien renvoyer. Dans ce cas, il faut indiquer **void** comme type en sortie et il n'est pas besoin d'écrire return.

* Vous pouvez définir des fonctions avant la fonction main, qui est elle même une fonction. Mais vous ne pouvez que définir des fonctions avant la fonction main.

  Il ne sert également à rien d'écrire quelque chose après le return 0; de la fonction main.

  Nous verrons dans des TPs précédents qu'il est également possible de définir les fonctions dans d'autres fichiers.

* Pour appeler une fonction dans la fonction main, il suffit d'utiliser le nom de la fonction comme vous avez pu le faire avec les fonctions exp, size, ...

Appel d'une fonction

* Si vous souhaitez utiliser des variables de la fonction main dans votre fonction **sans vouloir les modifier** et que ces variables n'utilisent pas trop de ressources mémoire, il vous suffit de les ajouter en argument en n'oubliant pas d'indiquer leur type.

  C'est ce que l'on appelle un passage par valeur.

  En pratique, lors d'un passage par valeur, C++ commence par copier la valeur des variables en entrée de la fonction et ne modifie par la suite que ces copies.

  Ainsi, si les variables utilisent beaucoup de ressources mémoires ou si vous souhaitez modifier les variables et non pas seulement leur copie, il va falloir procéder autrement.

* Si vous voulez modifier la valeur d'une variable de la fonction main par une fonction, vous pouvez faire

appel directement à l'emplacement mémoire qui lui est alloué (en passant par ce que l'on appelle un pointeur, mais nous reviendrons plus tard sur ces notions). Pour ce faire, il faut écrire un **&** entre le type et le nom de la variable.

```
type nomFonction (type& nomVariable)
{
    commandes;
}
```

C'est ce que l'on appelle un passage par référence.

Si vous ne le faites pas, la fonction n'utilisera que la copie des variables en entrée et ces copies seront effacées lorsque les commandes de la fonction seront effectuées.

* Vous pouvez également renvoyer les variables comme résultats de votre fonction et utiliser ses nouvelles valeurs en les affectant à vos variables dans votre fonction main.

  C'est cependant moins élégant et parfois moins pratique.

* Si vous ne voulez pas copier vos variables, mais que vous n'avez pas besoin de les modifier, il est plus prudent d'effectuer ce que l'on appelle un passage par référence constante. Pour ce faire, il faut ajouter un const avant l'esperluette.

  ```
  type nomFonction (type const& nomVariable)
  {
      commandes;
  }
  ```

  Dans ce cas C++ interdira toute commande qui modifie la variable et vous enverra un message d'erreur si vous tentez de la modifier.

  L'intérêt de cette méthode est, par exemple, de pouvoir utiliser un fichier de données expérimentales et de les analyser en étant sûr de ne pas les modifier.

Utilisation de nombres aléatoires

* Pour utiliser des nombres générés aléatoirement, vous aurez besoin d'inclure les bibliothèques **cstdlib** et **time.h**
* N'oubliez pas de commencer votre programme par **srand(time(NULL))**

Quelques conseils pour retravailler le TP

* La définition de fonction est beaucoup utilisé en C++, notamment lorsque l'on utilise des classes, ce qui est à la programmation orientée objet. Je vous conseille donc fortement de vous familiariser avec cette syntaxe.

  Faites bien attention à ne pas mélanger le type des arguments et le type du résultat. Ces notions sont très importantes en C++ et plus vous serez à l'aise à ce propos, mieux ce sera.

* L'utilisation du **&** et plus généralement des pointeurs est plus délicate, nous en rediscuterons plus dans le TP6. Si cette notion reste encore obscure pour vous, ne vous en inquiétez pas.