Programmation impérative, TP1

Démarche

Les exercices doivent être compilés sans warning avec la ligne suivante :

- `gcc -Wall` *[source].c* `-o` *[exec]*

où *[source].c* est le nom du fichier contenant le code source et *[exec]* est le nom de l'exécutable produit.

Dans le cas où la bibliothèque `math.h` est requise, il faut ajouter `-lm` à la commande de compilation (cf. `man`).

On remarque que l'exécution d'un objet natif n'affiche rien, il faut donc introduire quelques primitives d'affichage et pour cela ajouter en tête du fichier source la ligne `#include<stdio.h>` qui donne l'accès aux fonctions d'entrée/sortie.

- Afficher un int *n*, un double *e* en notation scientifique : `printf("%i",` *n*`)`, `printf("%e",` *e*`)`
- Caractère de saut de ligne : `\n`
- Chaînes de caratères `"Life, cruel joke"`

Essayez quelques exemples... Jouez avec les formats et les conversions `i,e,f,u,x,X`... Avez-vous lu le (bon) `man` ?

Exercices

1. On impose la déclaration et la définition de deux variables `a` et `b`. Proposez une suite d'instructions qui permute les valeurs que vous aurez affectées à `a` et `b`. Vérifiez la correction de votre solution avec `printf`.

Les bibliothèques de fonctions dont l'installation est standard sont déclarées en ajoutant la ligne `#include<`*[bibli].h*`>` au début du fichier source. Regardons par exemple la bibliothèque `math.h`.

La bibliothèque `math.h` contient entre autres l'exponentielle `exp`, les logarithmes népérien `log` et décimal `log10`, l'élévation de puissance `pow`, la racine carrée `sqrt`, la plupart des fonctions trigonométriques : `cos, sin, tan, acos, asin, atan, cosh`... Ces fonctions prennent en général des paramètres de type `double` avec une exception dans certains cas pour `pow`.

2. Que se passe-t-il lorsqu'on essaye d'élever 0 à une puissance négative ?
3. Que se passe-t-il lorsqu'on essaye d'élever un nombre négatif à une puissance non entière ?
4. Calculer (et afficher) une valeur approchée de π à l'aide de la fonction arctangente.
5. Calculer le volume d'une sphère en fonction d'un rayon `r` défini. Comme toujours, penser à tester votre fonction, par exemple avec un rayon `r` = 1.
6. Reste de la division euclidienne : `%` sur les types entiers.
    1. Proposer une séquence d'instructions qui affecte à une variable `pair` la valeur `1` si la valeur d'une variable `param` est paire et la valeur `0` sinon.

    2. Proposer une séquence d'instructions qui affiche `Pair` si la valeur d'une variable `param` est paire et qui affiche `Impair` sinon.
    3. Proposer une séquence d'instructions qui affiche `L'entier` *n* `est pair.` si la valeur *n* affectée à `param` est paire, etc.
7. Proposer une séquence d'instructions qui, sur la donnée de trois entier c_1, c_2 et h, teste s'ils peuvent représenter les longueurs des côtés d'un triangle rectangle d'hypoténuse de longueur h, d'abord en affectant une variable `test` à 1 ou 0 suivant le cas (0 si ça ne marche pas), puis en affichant un message explicite.
8. Proposer une séquence d'instructions qui, sur la donnée de trois entier c_1, c_2 et c_3, teste s'ils peuvent représenter les longueurs des côtés d'un triangle rectangle.
9. Proposer une séquence d'instructions qui, sur la donnée d'un pas *dx* et d'un point *x*, calcule une approximation de la dérivée d'une fonction trigonométrique (ou d'une composition de fonctions) de la bibliothèque `math.h` en ce point.
10. Proposer une séquence d'instruction qui sur la donnée d'un entier `n`, affiche
    - "Il n'y a pas de carotte." si `n` vaut 0.
    - "Il y a une carotte." si `n` vaut 1.
    - "Il y a *k* carottes." sinon, avec *k* la valeur de `n`.