TP : Algorithmes et langage C

Exercice 1 :

Créer un répertoire tp_algo
```
mkdir tp_algo
```
Aller dans ce répertoire et regarder ce qu'il y a dedans
```
cd tp_algo
ls -la
```
Créer le fichier `coucou.c` (écrire coucou ! à l'écran)
Editer son contenu avec xemacs.
```
xemacs coucou.c &
```
Compiler dans la fenêtre d'exécution avec la commande
```
gcc -o coucou coucou.c
```
Exécuter le programme dans la fenêtre de commande
```
coucou
```
Regarder quels fichiers ont été créés et les détruire avec la commande
```
rm nom_de_fichier
```

Exercice 2 :

Lire un entier N strictement positif et imprimer tous les nombres premiers compris entre 1 et N.

Exemple : si on indique l'entier 10, le résultat doit être : 1 2 3 5 7.

A. Utiliser seulement une fonction `main`
B. Utiliser une fonction `est_premier`

Exercice 3 : impression d'une matrice de Hilbert

Demander à l'utilisateur un entier compris entre 1 et 6 puis imprimer de façon claire la matrice de Hilbert d'ordre n.

La matrice de Hilbert (a_{ij}) est telle que $a_{ij}=1 / (i + j + 1)$.
Le spécificateur de format qui permet d'écrire un réel en précisant le nombre total p de caractères et le nombre d de chiffres après la virgule est le suivant: %p.df ; par exemple %8.3f si on veut 8 caractères dont 3 décimales (en comptant le point).

Exercice 4 : recherche dichotomique

L'utilisateur indique la valeur d'un "réel" `a` et deux "réels" nommés `min` et `max`. Ces trois réels seront codés avec des variables de type `double`. Le programme contient la définition de la fonction `f(x) = x³ - a`. On pourra avoir une variable globale `a` ; la fonction `f` aura pour prototype : `double f(double x);` le paramètre `x` correspond à la valeur de la variable `x` et la valeur de retour à la valeur de `f(x)`.
Il s'agit de chercher une éventuelle solution comprise entre `min` et `max` de l'équation `f(x) = 0` et cela par dichotomie.
Si `f(min)` et `f(max)` sont de même signe, le programme indique qu'il ne peut pas répondre à la question et s'arrête. Dans le cas contraire, la solution est calculée à 10^{-6} près.

On prévoira le programme pour qu'il puisse fonctionner si on change de fonction `f`.

Un point technique : pour saisir un `double` avec `scanf`, l'indicateur de format est `%lf`.

Exercice 5 : algorithmes de Horner

Réaliser l'implémentation du calcul de la valeur d'un polynôme P en un point en utilisant l'algorithme de Horner :

$P(x) = ((...((a_n x + a_{n-1})x + a_{n-2})... a_1) + a_0)$.

Le polynôme sera saisi en indiquant son degré puis tous ses coefficients qui seront mis dans un tableau.
Après la saisie du polynôme, le programme demandera à l'utilisateur s'il veut l'évaluer; si oui, celui-ci répondra par le caractère 'o' ; il devra alors donner la valeur de x pour lequel il demande le calcul de P(x). le programme redemandera alors si l'utilisateur veut à nouveau évaluer le polynôme P, et cela jusqu'à ce que l'utilisateur réponde par le caractère 'n'.

Un point technique : L'instruction :

```
fflush(stdin);
```

devra probablement être utilisée avent la saisie d'un caractère pour vider le buffer de lecture (qui pourrait contenir un caractère de retour à la ligne).

Exercice 6 : Algorithme de Tri Tas

Trier un certain nombre d'entier à l'aide de l'algorithme de tri tas.

Ces entiers figureront *dans un fichier* dont on demandera le nom à l'utilisateur *ou dans un tableau*. On écrira le résultat (entier triés) *dans un fichier* dont on demandera le nom à l'utilisateur *ou dans un tableau*.

Explications :

Le tri tas est un tri "en place", c'est-à-dire que les éléments sont insérés et triés dans un unique tableau. Après leur insertion dans le tableau, les éléments sont tout d'abord rangés "en tas", le tableau simulant ce tas.

Un tas est un arbre binaire "parfait" : toutes les rangées de l'arbre sont pleines sauf éventuellement la dernière, la dernière rangée est remplie de gauche à droite. Si on numérote les sommets de cet arbre de gauche à droite, dans chaque rangée, et de haut en bas, la racine ayant le numéro 1, on voit que les fils du sommet numéroté i sont numérotés 2i et 2i+1. Il y a donc une bijection naturelle entre les nœuds d'un tel arbre et les positions d'indice i, pour i variant entre 1 et n dans le tableau. A chaque sommet de l'arbre on associe un des éléments à trier.

Pour que l'arbre binaire parfait soit un tas, il faut qu'en chaque sommet l'élément qui s'y trouve soit plus grand que ceux situés en ses deux fils.

Un algorithme possible de construction d'un tas est le suivant :

- On considère que l'on introduit les uns après les autres les éléments dans le tas, dont la structure est reconstituée après chaque insertion.
- Pour mettre à sa place l'élément en cours d'insertion, on l'introduit à la première place disponible, puis on le compare à son père : s'il est plus grand que son père, on l'échange avec son celui-ci, on recommence ces échanges jusqu'à ce que l'élément à insérer soit plus petit que son père ou qu'il ait atteint la racine.
- Une fois le tas construit avec tous les éléments à trier, le maximum de ceux-ci se trouve à la racine. On peut donc utiliser le tas pour trier les éléments : on retire la racine, on restaure la structure de tas sur les éléments restants et on recommence.
- Pour restaurer la structure de tas, on place le dernier élément du tas à la racine (à la place de l'élément qu'on vient de retirer pour construire la liste triée , en fait, pour ne pas consommer de place supplémentaire, on échange la racine du tas courant et le dernier élément du tas courant, et on considère que le tas courant est maintenant l'ancien tas courant privé de son dernier élément), puis on le fait "descendre" dans le tas en l'échangeant avec le plus grand de ses fils, si le plus grand de ses fils est plus grand que lui.
- Quand on a fini, le tableau contient les éléments dans l'ordre croissant.