PROGRAMMATION C - TP N. 1

1. Input/Output standard

Exercice 1 : Pour se familiariser avec les instructions d'input et output standards

Écrire un programme qui utilise la librairie `stdio.h` (voir le polycopié de A. Canteaut, section 1.11 pour les détails) et :

- Demande le nom de l'utilisateur et lui souhaite une bienvenue personnalisée ("`Bonjour [nom_utilisateur] !!!`" par exemple);
- Demande l'age de l'utilisateur;
- Demande le numéro de téléphone de l'utilisateur;
- Imprime à l'écran toutes les données collectées.

Conseil : Déclarer les chaînes de caractères comme des tableaux de caractères (suffisamment longs pour contenir ce qu'on souhaite) et éventuellement ajouter une option à `scanf` pour éviter que l'utilisateur rentre une chaîne trop longue. (*exemple :* `char chaine[50];`)

Exercice 2 : petit convertiteur décimale/octale/hexadécimale

Écrire un programme qui demande à l'utilisateur le format d'entrée et sortie (décimale/octale/hexadecimale) (par exemple en associant un numéro a chacune des options) et un numéro à convertir, et imprime à l'écran la conversion démandée.

Attention à gérer aussi le cas ou l'utilisateur répond avec une option inexistante.

On pourra demander enfin à l'utilisateur si il a un autre nombre à convertir ou s'il veut arrêter le programme.

2. Tableaux unidimensionnels : vecteurs

Exercice 3 : simple manipulation de vecteurs

Écrire un programme qui, étant fixé un entier `N` :

(1) Initialise deux vecteurs d'entiers `v1` et `v2` (de taille `N`) et une constante `a` (le mieux serait de faire rentrer les valeurs par l'utilisateur);
(2) déclare un troisième vecteur `v0` (de taille `N`) sans l'initialiser, et copie le vecteur `v2` dans `v0`, puis imprime `v0` à l'écran;
(3) calcule le vecteur somme de `v1` et `v2` et l'imprime à l'écran;
(4) calcule le produit scalaire entre `v1` et `v2` et l'affiche à l'écran;
(5) calcule le vecteur obtenu en multipliant `v1` par la constante `a` et l'imprime à l'écran;
(6) calcule les normes de `v1` et `v2` et les imprime à l'écran.

Remarque : La taille des vecteurs sera fixée au debut du programme grâce à une commande au pré-processeur (`#define`).

(TOURNEZ SVP)

Date: 29 septembre 2010.

Exercice 4 : le crible d'Ératosthène
Le crible d'Ératosthène est la méthode la plus ancienne est simple pour créer une liste de nombres premiers.
L'idée du crible d'Ératosthène est de prendre une liste de tous les entiers $i \leq N$ et de la cribler : on commence par $n = 2$, et on enlève de la liste tous les éléments suivants qui sont divisibles par 2, puis on passe au premier nombre non enlevé ($n = 3$) et on fait la même chose, etc. On ne risque pas de tomber sur des nombres composés car ils auront déjà été enlevès par les cribles précedents. À la fin il ne nous reste donc que les nombres premiers.
Écrire un programme qui :

- Fixe la taille N du vecteur à cribler avec une commande au pré-processeur (`#define`)
- Construit un vecteur de N éléments, initialisé avec les valeurs de 0 à $N - 1$.
- Applique le crible d'Erathosthene au vecteur (en particulier ici "enlever" un élément correspondra par exemple à mettre un 0 à son emplacement).
- Imprime à l'écran les nombres premiers du vecteur.

En particulier, on pourra se demander quelle est la valeur de n maximale pour la quelle on a besoin de cribler (en fonction de N taille du vecteur) et on pourra utiliser des boucles `for` intelligentes pour parcourir les valeurs à cribler.
Remarque : On pourra utiliser la fonction `sqrt` de la librairie `math.h` mais pas d'autres fonction de cette librairie-ci...

3. Tableaux 2-dimensionnels : Matrices

Exercice 5 : simple manipulation de matrices
Écrire un programme pour manipuler des matrices carrées $N \times N$:

(1) Initialise deux matrices d'entiers `M1` et `M2` (de taille `N`×`N`) et une constante `a` (le mieux serait de faire rentrer les valeurs par l'utilisateur);
(2) déclare une troisième matrice `M0` (de taille `N`×`N`) sans l'initialiser, et copie la matrice `M1` dans `M0`, puis imprime `M0` à l'écran;
(3) calcule la matrice somme de `M1` et `M2` et l'imprime à l'écran;
(4) calcule le produit entre `M1` et `M2` et l'affiche à l'écran;
(5) calcule la matrice obtenue en multipliant `M2` par la constante `a` et l'imprime à l'écran.

Exercice 6 : Pivot de Gauss
La méthode du pivot de Gauss permet de calculer le rang d'une matrice ou encore de résoudre plus facilement des systèmes d'équations.
On considère une matrice A à M lignes et N colonnes.
À chaque étape du programme, on aura déjà traité les lignes $0, \ldots, i-1$ de la matrice, on considère donc que les lignes $i, \ldots, M-1$:

- On change l'ordre des lignes de manière que la ligne L_i soit celle avec l'élément non nul le plus à gauche (supposons en position j).
- On remplace les lignes de $i+1$ à $M-1$ par $L_k = L_k - \lambda L_i$ de manière à annuler l'élément de position (k, j) de chacune de ces lignes.

On continue tant qu'on n'a pas traité toutes les lignes.
Essayer le programme sur des simples exemples ({{1,2,3},{1,1,0}} et {{1,8,2,4},{1,1,0,2},{2,2,1,2},{0,7,5,2}} pour commencer...)
Remarque : On supposera initialement, par simplicité, que tous les coefficients et les matrices obtenus en cours d'éxecution sont de type entier (mais bien sur cela ne nous permet pas d'appliquer le pivot de Gauss sur toute matrice...)
Remarque 2 : Pour généraliser le programme de manière qu'il marche aussi quand les coefficients sont des fractions, on pourra soit considerer des matrices de nombres flottants (mais alors il y aura des erreurs possibles) soit construire un type `struct fraction` et des fonctions adaptées (pour les plus courageux...).