PROGRAMMATION C - TP N. 2

1. Pointeurs

Exercice 1 (Pour se familiariser avec les pointeurs)
Écrire un programme qui :

(1) Initialise des variables `a`,`b`,`c` de type entier, `d` de type caractère, `e` de type flottant (`double` ou `float`).
(2) Déclare des pointeurs (du bon type) `p` sur `b`, `q` sur `d` et `r` sur `e`.
(3) Imprime à l'écran, pour chaque pointeur, la valeur du pointeur (=adresse de l'objet pointé) et la valeur de l'objet pointé.
(4) Faire maintant pointer `p` à `c` et imprimer son addresse et la valeur de l'objet pointé.
(5) Modifier la valeur pointée par `p` pour qu'elle soit égale à la valeur de `a`. Imprimer à l'écran l'adresse et la valeur pointée par `p`, puis les valeurs de `a`, `b` et `c`.

Exercice 2 (allocation dynamique)
Écrire un programme qui utilise l'allocation dynamique pour créer un tableau d'entiers à deux dimensions, tel que la première ligne ait n éléments, la deuxième $n-1$, ... et la dernière 1.
Initialiser à votre choix ce tableau (soit tout à zéro, soit avec des entiers de votre choix, soit par entrée de l'utilisateur, ...) et imprimez-le à l'écran (une ligne pour chaque ligne du tableau).
Rappelez-vous de vider la mémoire avec la fonction `free` avant la sortie du programme.

Exercice 3 (listes) Utiliser la struct `cellule` vue en classe :

```
struct cellule
{
   int valeur;
   struct cellule *suivant;
};

typedef struct cellule *liste;
```

(TOURNEZ SVP)

Date: 13 octobre 2010.

(1) Construire une liste chaînée de 5 éléments et l'initialiser avec des valeurs de votre choix.
(2) Imprimer à l'écran la liste.
(3) Demander à l'utilisateur une valeur et la position de la liste où mettre cette valeur (en remplaant la prcedente), et éxecuter cette commande.
(4) Ajouter une cellule au debut de la liste.
(5) Ajouter une cellule à la fin de la liste.
(6) Ajouter une cellule entre la position n-ième et la position $(n+1)$-ième de la liste (où n est entré par l'utilisateur).

Ceux qui savent utiliser des fonctions secondaires, pourront implementer chaque point de l'exercice dans une fonction différente (sauf le (1) qui va etre dans le `main()`) et les faire appeler par la fonction `main()`.

Exercice 4 (arbres) En utilisant la structure `noeud` vue en cours :

```
struct noeud
{
   int valeur;
   struct noeud *fils_gauche;
   struct noeud *fils_droit;

};

typedef struct noeud *arbre;
```

(1) Construire un arbre binaire de hauteur 4 (avec toutes les feuilles possibles). Initialiser cet arbre à votre choix.
(2) Demander à l'utilisateur de rentrer une chaîne de caractères formée que de 0 et de 1, de longueur maximale 4. On parcourira donc l'arbre en choisissant à chaque étape gauche si le caractère corréspondant est 0, droite si le caractère corréspondant est 1. Imprimer à l'écran la valeur du noeud d'arrivée.

Le point (2) pourra être implementé dans une fonction secondaire.