TP 1 – semestre II

(Fonctions et procédures : passage d'arguments par valeur / adresse)

EXERCICE 35

Ecrire les fonctions et les procédures suivantes :

- Procédure **Lecture_tab** : elle permet la lecture des éléments d'un tableau d'entiers TAB de taille N. A chaque itération, et pour chaque lecture, un message sera affiché sur la console demandant la saisie du $i^{ème}$ élément du tableau. Les valeurs saisies doivent être comprises entre 5 et 50.
- Fonction **Max_tab** : elle permet de retourner la valeur maximale d'un tableau d'entiers TAB de taille N.
- Fonction **Pair_tab** : elle permet de retourner le nombre d'éléments pairs (divisibles par 2) d'un tableau d'entiers TAB de taille N.
- Fonction **Occurrence_tab** : elle permet de retourner le nombre d'occurrences d'un élément x dans un tableau d'entiers TAB de taille N.
- Procédure **EltPlusFrequent_tab** : elle permet d'afficher le nombre le plus fréquent (ayant le nombre d'occurrence le plus grand) dans un dans un tableau d'entiers TAB de taille N, ainsi que son nombre d'occurrence.

Utiliser ces procédures dans un programme en utilisant un tableau d'entiers de taille 10.

EXERCICE 36

Ecrire les fonctions et les procédures suivantes :

- Procédure **Lecture_tab** : elle permet la lecture des éléments d'un tableau d'entiers TAB de taille N. A chaque itération, et pour chaque lecture, un message sera affiché sur la console demandant la saisie du $i^{ème}$ élément du tableau.
- Procédure **Permuter** : elle permet de permuter deux entiers A et B.
- Procédure **Inverser_tab** : elle permet de transformer un tableau TAB de taille N d'une manière symétrique. Ex :

3	5	2	4

devient

4	2	5	3

Cette procédure peut faire appel à la procédure « Permuter ».
Utiliser ces procédures dans un programme en utilisant un tableau d'entiers de taille 10.

Attention au passage d'arguments par adresse ! Dans `&(TAB[i])`, les parenthèses sont-elles obligatoires ?

EXERCICE 37 (facultatif)

On appelle nombres d'Armstrong les nombres entiers tels que la somme des cubes de leurs chiffres (en base 10) est égale au nombre lui-même. Exemple :

$$1^3 + 5^3 + 3^3 = 153$$

Ecrire un programme qui affiche les nombres d'Armstrong inférieurs à 100000. Ce programme contient une fonction qui teste si un nombre donné est un nombre d'Armstrong.

TP 2 – semestre II
(Les pointeurs – passage par adresse)

EXERCICE 38

Les opérateurs & et * sont liés à l'utilisation des pointeurs.

1) Tester le programme suivant :

```
#include <stdio.h>
void main()
{
      int x,*p;     /* p est déclaré comme pointeur vers un entier. Il
                    contient initialement une valeur aléatoire. Voir adresse
                    de x aussi */
      printf("p=%d\n",p);
      printf("adresse de x = %d\n",&x);
}
```

2) Tester le programme en ajoutant (juste après la déclaration des variables) l'instruction :

```
p= &x ;
```

3) Garder la modification de la question 2), et remplacer provisoirement (juste pour la question 3)) dans les printf le `"%d"` par `"%p"` (affichage hexadécimal des adresses mémoires qui est souvent utilisé par les programmeurs)

4) Ajouter à la fin de ce programme le bloc suivant :

```
*p=5;
printf("x= %d\n",x);
printf("*p= %d\n",*p);
```

EXERCICE 39

```
#include <stdio.h>
void incrementation(int b)
{
     b++;
}
void main()
{
      int a=5; incrementation(a);
      printf("%d\n",a);
}
```

```
#include <stdio.h>
void incrementation(int *b)
{
     (*b)++;
}
void main()
{
      int a=5; incrementation(&a);
      printf("%d\n",a);
}
```

C'est quoi la valeur de a à la fin de chacun des deux programmes ? Justifier. Essayer d'enlever les parenthèses dans `(*b)++;` dans le second programme. Que remarquez-vous ?

EXERCICE 40 (facultatif)

Ecrire une procédure **transformer** qui transforme un entier A d'une manière symétrique. C'est-à-dire, si A=3524 alors A devient = 4253. Utiliser cette procédure dans un programme.

TP 3 – semestre II
(Allocation dynamique de la mémoire– tableaux)

EXERCICE 41 (facultatif)

Ecrire une procédure **modifier_ch** qui permet de modifier une chaîne de caractère ch en remplaçant les lettres minuscules par des espaces ' ' et les lettres restantes par des étoiles '*'. Utiliser cette procédure dans un programme qui commence par la lecture d'une chaîne de caractères.

EXERCICE 42

Ecrire un programme qui permet les opérations suivantes :

- saisir le nombre n de notes obtenues par un étudiant,
- en utilisant la fonction **malloc**, construire un tableau « dynamique », appelé NOTE, pour les enregistrer les notes et un autre, appelé COEF, pour enregistrer les coefficients correspondant à chaque matière,
- remplir les deux tableaux à partir du clavier en demandant pour chaque matière la note et le coefficient,
- calculer la moyenne de l'étudiant et l'afficher.
- en utilisant la fonction **free**, libérer en fin du programme la mémoire allouée pour les deux tableaux.

Attention les notes, les coefficients et la moyenne sont du type réel (float) !

EXERCICE 43

Ecrire un programme qui permet les opérations suivantes :

- saisir un nombre entier x,
- calculer et afficher à l'aide d'une fonction appelée « longueur » la longueur de l'entier x (c'est-à-dire le nombre de chiffres qu'il renferme : longueur(324) = 3 et longueur(150433)= 6).
- en utilisant la fonction **malloc**, construire un tableau « dynamique » d'entiers, appelé TAB, de taille exactement égale à la longueur de x,
- stocker les chiffres que contient x, dans l'ordre, dans le tableau TAB,
- afficher les éléments de ce tableau.
- libérer en fin de l'itération la mémoire allouée par le tableau.
- (facultatif) : ces dernières opérations peuvent se faire dans une boucle qui s'arrête une fois que l'utilisateur souhaite quitter : demander à l'utilisateur s'il souhaite refaire une nouvelle itération et lire un caractère à partir du clavier. Si ce dernier est différent de r ou R on quitte la boucle et on termine le programme.

EXERCICE 44

Refaire l'exercice 43 en utilisant la fonction **realloc** au lieu de **malloc**, et sans introduction de la fonction longueur (la taille du tableau évolue d'une manière dynamique).

TP 4 – semestre II
(Allocation dynamique de la mémoire– tableaux)

EXERCICE 45

Soit T un tableau d'entiers de taille n = 10.

- Remplir à partir du clavier le tableau T.
- Soit t un pointeur entier initialisé à la valeur NULL. A l'aide d'une procédure appelée « Elts_pairs », et en utilisant la fonction realloc, faire pointer t sur un nouveau tableau qui contient uniquement les éléments pairs du tableau T.
- Afficher le nouveau tableau t.

EXERCICE 46

Soit T un tableau d'entiers de taille n = 10.

- Remplir à partir du clavier le tableau T.
- Soit t un pointeur entier initialisé à la valeur NULL. En utilisant la fonction realloc, faire pointer t sur un nouveau tableau qui contient uniquement les éléments pairs du tableau T.
- Afficher le nouveau tableau t.

Remarque : dans cet exercice, toutes les instructions sont faites dans la fonction main()

EXERCICE 47

Soit T un tableau d'entiers de taille n = 10.

- A l'aide d'une procédure appelée « annuler_tab», affecter à chaque élément d'un tableau la valeur 0.
- Appliquer cette procédure au tableau T, et afficher le résultat.

EXERCICE 48 (facultatif)

Soit T un tableau d'entiers de taille n = 10.

- Remplir à partir du clavier le tableau T.
- A l'aide d'une procédure appelée « annuler_pairs_tab», modifier la valeur de chaque élément pair d'un tableau par la valeur 0.
- Appliquer cette procédure au tableau T, et afficher le résultat.

EXERCICE 49

Construire une matrice A d'ordre m x n d'éléments entiers: A est utilisée en tant que tableau à deux dimensions, les valeurs m et n seront lues à partir du clavier (utiliser les fonctions malloc et free).

- Remplir A avec les éléments de la matrice "identité".
- Afficher A

TP 5 – semestre II
(Les structures)

EXERCICE 50

Définir une structure de données appelée **nb_complexes** qui représente l'ensemble des nombres complexes : un nombre complexe z contient une parie réelle pR et une parie imaginaire pI (z = pR + i pI, avec pR ∈IR et pI ∈IR). Votre programme permet les opérations suivantes :

- saisir la partie entière pR et la partie imaginaire pI d'un nombre complexe z
- calculer et afficher le module de z :

$$\text{module}(z) = \sqrt{pR^2 + pI^2}$$

NB : utiliser la bibliothèque math.h pour les fonctions mathématiques (la racine carrée est donnée par la fonction sqrt()).

EXERCICE 51

Définir une structure de données appelée **RECTANGLE** qui contient 4 composantes réelles: longueur (L), largeur (l), perimetre (P) et surface (S). Dans votre programme :

- Déclarer une variable de type RECTANGLE appelée rec1.
- Saisir la longueur et la largeur de rec1. Par la suite calculer et afficher sa surface et son périmètre.

EXERCICE 52 (suite exercice 51)

- Saisir un entier n et un tableau de taille n de type RECTANGLE appelé rec.
- A l'aide d'une procédure appelée «**modifier»,** transformer un rectangle en un rectangle plus petit en divisant sa longueur et sa largeur par 2.
- Afficher les composantes des rectangles après modification.

EXERCICE 53 (Devoir à la maison noté à rendre la prochaine séance)

Définir une structure de données appelée **DATE** qui contient 3 composantes entières: jour, mois et année. Votre programme permet les opérations suivantes :

1. Saisir deux dates différentes : D1 et D2,
2. A l'aide d'une fonction binaire appelée **testerDate**, vérifier la validité d'une date. Exemple la date {35, 22, 2009} n'est pas valide : aucun mois ne compte 35 jours, et le mois 22 n'existe pas. Ce test s'appuie sur un tableau appelé **nbJmois**, de taille 12, qu'il faut créer et remplir (déclaration en tant que variable globale, ou bien à l'intérieur de la fonction) :

Indice (mois)	1	2	3	4	5	6	7	8	9	10	11	12
nbJmois	31	28	31	30	31	30	31	31	30	31	30	31

Par ailleurs, une année bissextile est divisible par 4 (exemple année 2008 = 502*4). Dans une année bissextile, le mois de février compte 29 jours).

3. Créer une fonction **plusRecente** qui renvoie la date la plus récente de deux dates, Exemple {5, 3, 2009} est plus récente que {23, 1, 2009}.
4. Tester la validité des deux dates D1 et D2.
5. En cas de non validité de l'une des deux dates, renvoyer un message d'erreur. Dans le cas contraire, afficher la date la plus récente des deux.

TP 6 – semestre II
(Les algorithmes de tri)

EXERCICE 54 (tri d'un tableau d'entiers)

Soit T un tableau d'entiers de taille N = 10, que vous allez saisir à partir du clavier. Tester sur T l'un des algorithmes de tri suivant :

- Tri à bulles (ou par propagation)
- Tri par insertion
- Tri par sélection

NB : Chaque groupe d'étudiants choisira un algorithme à tester.

EXERCICE 55 (tri d'un tableau de structure)

Ecrire un code qui permet les opérations suivantes :

- Définir une **structure** appelée **Article** ayant deux attributs : référence notée **ref** de type entier, et **prix** de type réel (float).
- Ecrire une procédure lecture qui permet la lecture d'une variable x de type Article
- Dans le programme, définir un tableau A de taille n=10 de type Article. Avec boucle et utilisant la procédure lecture, lire tous les éléments du tableau A
- Trier après lecture avec un algorithme de tri par sélection (adapté) le tableau A selon le prix croissant des articles.
- Avec une procédure appelée afficher_tab qui affiche un tableau de type Article, afficher A.

Annexe TP 6

- Tri à bulles :

```
void tri_a_bulle(int t[], int n)
{
   int i,j, x, ordre = 0;
   for(i = 0 ; (i < n) && (ordre==0); i++)
   {
         ordre = 1;
         for(j = 1 ; j < n-i ; j++)
         {
               if(t[j] < t[j-1])
               {
                     x = t[j-1]; t[j-1] = t[j];
                     t[j] = x; ordre = 0;
               }
         }
   }
}
```

L'algorithme parcourt la liste d'éléments à trier et compare les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés. Après chaque parcours complet de la liste (chaque itération), l'algorithme recommence l'opération. Lorsqu' aucun échange n'a eu lieu pendant un parcours, cela signifie que la liste est triée : l'algorithme peut s'arrêter.

- Tri par insertion :

```
void insertion(int t[], int n)
{
    int i,j,p,x;
    for (i = 1; i < n; i++)
    {
        x = t[i];
        /* recherche du plus petit indice p
        inférieur à i tel que t[p] >= t[i] */
        p = 0; while(t[p] < x) p++;
        /* décalage avant des valeurs de t entre p
        et i */
        for (j = i-1; j >= p; j--) t[j+1] = t[j];
        t[p] = x; /* insertion de la valeur stockée
        à la place vacante */
    }
}
```

Pour procéder à un tri par insertion, il suffit de parcourir une liste : on prend les éléments dans l'ordre. Ensuite, on les compare avec les éléments précédents (étant par hypothèse déjà triés) jusqu'à trouver la place de l'élément qu'on considère. Il ne reste plus qu'à décaler une partie des éléments du tableau pour insérer l'élément considéré à sa place dans la partie déjà triée. Par conséquent, le principe du tri par insertion est d'insérer à la n-ième itération le n-ième élément à la bonne place.

- Tri par sélection :

```
void selection(int *t, int n)
{
     int i, min, j , x;
     for(i = 0 ; i < n-1 ; i++)
     {
         min = i;
         for(j = i+1 ; j < n ; j++)
              if(t[j] < t[min]) min = j;
         if(min != i)
         {
             x = t[i]; t[i] = t[min]; t[min] = x;
         }
     }
}
```

Le principe du tri par sélection est de chercher le plus petit élément pour le mettre en premier, puis de repartir du second élément et de chercher plus petit élément de l'ensemble restant pour le mettre en second, etc... Au ième passage, on sélectionne donc l'élément ayant le plus petit parmi {T[i]...T[n]} et on l'échange avec T[i].

TP 7 – semestre II
(Les fichiers) :1h30

EXERCICE 56

Créer un fichier appelé « perso.txt » dans lequel vous écrivez ce qui suit :
- votre nom à la première ligne. Exemple : (Nom : Ben Mohamed)
- votre prénom à la deuxième ligne. Exemple : (Prenom : Ali)
- votre age à la troisième ligne. Exemple (Age : 19)
- votre moyenne au bac à la quatrième ligne. Exemple (Moyenne : 10.85)

Après écriture fermer votre fichier.
Vérifier dans le même répertoire qui contient votre programme la création et le contenu du fichier.

EXERCICE 57

Ecrire un programme qui permet d'ouvrir le fichier « perso.txt » en mode lecture et de stocker les 10 premiers caractères dans le fichier dans un tableau de caractère T. Afficher T.
Remarque :
La fin d'un fichier n'est pas arrivée tant que **`feof(ptr)==0`**. (ptr est le pointeur du type FILE associé au fichier et feof est une fonction qui détecte la fin d'un fichier : end of file).

EXERCICE 58

Ecrire un programme qui permet de créer une copie du fichier « perso.txt » qui s'appelle « copie.txt ».

EXERCICE 59 (noté : à préparer à la maison pour la prochaine séance)

Créer une fonction **`int nb_caracteres(char *name)`** qui permet de :
- Ouvrir le fichier qui porte le nom name en mode lecture.
- Calculer et retourner le nombre de caractères dans ce fichier,

Dans votre programme principal, utiliser la fonction précédente pour calculer et afficher n1 : le nombre de caractères du fichier « perso.txt ».

EXERCICE 60 (noté : à préparer à la maison pour la prochaine séance)

Créer une fonction **`int nb_lignes(char *name)`** qui permet de :
- Ouvrir le fichier qui porte le nom name en mode lecture.
- Calculer et retourner le nombre de lignes dans ce fichier,

Dans votre programme principal, utiliser la fonction précédente pour le fichier « perso.txt ».
Indication :
Une ligne finit avec le caractère '\n' (retour à la ligne = la touche ENTREE).