Université Pierre et Marie Curie Paris VI
Licence EEA

Corrigé des Travaux Dirigés de Langage C

B. GAS
Laboratoire LIS
Perception et Connexionnisme

T.D. N°1 : Blocs d'instructions, Structure alternative, Opérateurs et Expressions

Pour tous les TDs, les algorithmes doivent être expliqués à l'aide d'un pseudo langage simple suivi d'un corrigé écrit en langage C.

EXERCICE N°1:

Ecrire un programme qui déclare la variable constante π et la variable R contenant la valeur 20. Déclarer trois variables D, P et S et affecter respectivement à ces variables les valeurs du diamètre, du périmètre et de la surface d'un cercle dont le rayon est R. On affichera à l'écran le contenu de ces différentes variables selon le format suivant :

```
Un cercle de rayon WW a pour diamètre XX, pour circonférence YY
et pour surface ZZ.
```

- *les notions de variables et de constantes auront été abordées en cours.*
- *La syntaxe exacte de la chaine de format d'un* `printf()` *n'est évidemment pas connue des étudiants. Contentons nous de parler de* `%d` *et* `%f`*.*
- *Travaillez éventuellement sur des variantes de cet exercice afin de vous assurer de la compréhension par tout le monde de la déclaration, initialisation et utilisation d'une variable.*
- *insister sur l'aspect séquentiel de la programmation.*

```
#include <stdio.h>

main ()
{
    const float pi = 3.1415;
    const float R = 20;
    float d, p, s;

     d = 2*pi;
     p = d*R;
     s = pi*R*R;

     printf("Un cercle de rayon %f a pour diamètre %f, \
pour circonférence %f et pour surface %f", R, d, p, s);
}
```

EXERCICE N°2:

Ecrire les programmes permettant de trouver le minimum de 2 entiers, le maximum de 3 entiers, la valeur médiane de trois réels. On utilisera d'abord les instructions de test, puis les opérateurs.

- *Des rudiments sur les opérateurs arithmétiques, de comparaison et conditionnel seront donnés en cours. J'aurai surtout alors tenté de faire passer la notion d'opérateur, d'expression et de valeur d'une expression, etc.*
- *Insister sur l'importance des commentaires.*
- *expliquer que dans une expression* `(TEST1 op TEST2 op TEST3 ...)` *où* `op` *désigne un opérateur relationnel, les tests sont réalisés de la gauche vers la droite et que de plus, ils ne sont pas forcément tous réalisés.*
- *Le point précédent peut être vu en TP.*

– minimum de trois entiers :

```
#include <stdio.h>

main ()
{
    int x, y, z, resultat;

/* saisie des valeurs de x, y et z */

    printf("donnez les valeurs de x, y et z : ");
    scanf("%d %d %d",&x, &y, &z);

/* comparaisons */

    if (x<y)
        resultat = x;
    else
        resultat = y;

    if (resultat > z)
        resultat = z;

    printf("resultat = %d", resultat);
}
```

– minimum de trois entiers avec des opérateurs :

```
#include <stdio.h>

main ()
{
    int x, y, z, resultat;

    printf("donnez les valeurs de x, y et z : ");
    scanf("%d %d %d",&x, &y, &z);

    resultat = x<y? x : y;
    resultat = resultat<z? resultat : z;

    printf("resultat = %d", resultat);
}
```

– maximum de trois entiers :

```
#include <stdio.h>

main ()
{
    int x, y, z, resultat;

    printf("donnez les valeurs de x, y et z : ");
    scanf("%d %d %d",&x, &y, &z);

    if (x>y)
        resultat = x;
    else
        resultat = y;
```

```
    if (resultat < z)
        resultat = z;

    printf("resultat = %d", resultat);
}
```

– maximum de trois entiers avec des opérateurs :

```
#include <stdio.h>

main ()
{
    int x, y, z, resultat;

    printf("donnez les valeurs de x, y et z : ");
    scanf("%d %d %d",&x, &y, &z);

    resultat = x>y? x : y;
    resultat = resultat>z? resultat : z;

    printf("resultat = %d", resultat);
}
```

– la valeur médiane (version *bourrin* destinée a travailler sur les structures **if** emboités, parler de la possible importance des accolades) :

```
#include <stdio.h>

main ()
{
    int x, y, z, resultat;

    printf("donnez les valeurs de x, y et z : ");
    scanf("%d %d %d",&x, &y, &z);

    if (x<y)
        if (y>z)
            if (x<z)
                resultat = z;
            else
                resultat = x;
        else
             resultat = y;
    else if (x>z)
        if (y<z)
            resultat = z;
        else
            resultat = y

    printf("resultat = %d", resultat);
}
```

– la valeur médiane (version utilisant les opérateurs de comparaison):

```
#include <stdio.h>

main ()
{
    int x, y, z, resultat;
```

```
    printf("donnez les valeurs de x, y et z : ");
    scanf("%d %d %d",&x, &y, &z);

    if (x<y && x>z)
        resultat = x;
    else if (y<x && y>z)
        resultat = y;
    else
        resultat = z;

    printf("resultat = %d", resultat);
}
```

– la valeur médiane (version utilisant l'opérateur conditionnel):

```
#include <stdio.h>

main ()
{
    int x, y, z, resultat;

    printf("donnez les valeurs de x, y et z : ");
    scanf("%d %d %d",&x, &y, &z);

    resultat = x<y? (y>z? (x<z? z:x) :y) : (x>z? (y<z? z:y) :x);

    printf("resultat = %d", resultat);
}
```

– la valeur médiane (version utilisant les opérateurs conditionnel et de comparaison):

```
#include <stdio.h>

main ()
{
    int x, y, z, resultat;

    printf("donnez les valeurs de x, y et z : ");
    scanf("%d %d %d",&x, &y, &z);

    resultat = x<y && x>z? x : y<x && y>z? y : z;

    printf("resultat = %d", resultat);
}
```

EXERCICE N°3:

Ecrire le programme de résolution d'une équation du second degré $ax^2 + bx + c = 0$ dont les coefficients entiers a, b et c sont entrés au clavier. Des affichages permettront de faciliter l'utilisation du programme par un utilisateur.

– *Cet exercice utilise la fontion* `double sqrt(double)` *de la librairie mathématique. Il permet donc de donner un exemple supplémentaire d'inclusion d'un fichier ".h" pour l'utilisation de fonctions non écrites par l'utilisateur. A ce niveau du cours, les étudiants n'en savent pas plus.*

- *Les fonctions n'étant pas encore traitées, je propose que l'on parle de* `sqrt()` *comme on parle des fonctions* `printf()` *et* `scanf()`*, c'est à dire sans s'étendre dessus.*

- *Comme tout se fait en* `double` *dans cet exercice, il faut parler du* `%ld` *correspondant du* `printf()`*.*

```
#include <stdio.h>
#include <math.h>

main()
{
    int a, b, c;
    double delta, sol1, sol2;      /*double est un "grand" float ! */

    printf ("Forme de l'équation : aX2 + bX + C = 0 \n\n");
    printf ("Entrez les coefficients a, b, et c : ");
    scanf ("%d %d %d", &a, &b, &c);
                                   /* calcul du discriminant */
    delta = b*b - 4*a*c;
                                   /* calcul des solutions */
    if (delta >= 0)
    {
        sol1 = (-b - sqrt (delta))/(2*a);
        sol2 = (-b + sqrt (delta))/(2*a);
        if (delta==0)
            printf ("La seule solution est : %lf \n", sol1);
        else
            printf ("Les deux solutions sont : %lf et %lf \n", sol1, sol2);
    }
    else
        printf ("il n'y a pas de solutions réelles.\n");
}
```

T.D. N°2 : Itérations, la boucle `while()`

- *Seule la boucle* `while()` *doit être utilisée dans ce TD*
- *Pour tous les exercices, les algorithmes doivent être d'abord présentés.*

EXERCICE N°1:

Ecrire le programme permettant de calculer $N!$.

- *Exemple d'algorithme :*

  var n, i, fact;
  initialiser fact à 1;
  initialiser i à 1;
  tant que i ≤ n faire
  debut
  *fact = fact * i;*
  incrémenter i;
  fin

Programme en C :

```
#include <stdio.h>

main ()
{
    int n = ;           /* ordre */
    int i, fact;        /* indice de boucle et resultat */

   fact = 1;
    i = 1;              /* init. var. de boucle */
    while(i<=n)
    {
        fact *= i;
        i++;
    }

    printf("factoriel(%d) = %d \n",n,fact);
 }
```

EXERCICE N°2:

Ecrire le programme permettant de calculer x^n.

- *Exemple d'algorithme :*

  var x, n, i, puiss;
  initialiser puiss à 1;
  initialiser i à 1;
  tant que i ≤ n faire
  debut
  *puiss = puiss * x;*
  incrémenter i;
  fin

Programme en C :

```
#include <stdio.h>

main ()
{
    int n = ;           /* ordre */
    int x = ;           /* valeur de x */
    int i, puiss;       /* indice de boucle et resultat */


    puiss = 1;
    i = 1;              /* init. var. de boucle */
    while(i<=n)
    {
        fact *= x;
        i++;
    }

    printf("%d élevé à la puissance %d = %d \n",x,n,puiss);
 }
```

EXERCICE N°3:

Ecrire un programme qui affiche sous forme binaire le résultat d'un **OU**, d'un **ET**, ou d'un **OU EXCLUSIF** entre deux entiers entrés au clavier, selon le choix de l'utilisateur.

- *Cet exercice utilise la fontion* `putch(char)` *d'affichage d'un caractère de la librairie standard.*
- *Il donne aussi un exemple d'utilisation de l'opérateur* `sizeof()`
- *Il convient d'afficher les bits dans le bon ordre !*
- *exemple d'algorithme :*

  entrer le choix de l'utilisateur (1, 2 ou 3)
  selon le choix, calculer l'opération logique AND, OR, ou XOR
  afficher le resultat en binaire

- *algorithme d'affichage en binaire :*

  int i, mem;
  nombre = ...
  initialiser i à 0;
  tant que i < nombre de bits d'un entier faire
  debut
  mem = nombre décalé à gauche une fois;
  mem = mem décalé à droite une fois;
  si nombre=mem faire
  afficher le caractère 1;
  sinon faire
  afficher le caractère 0;
  nombre = nombre décalé à gauche une fois;
  incrémenter i;
  fin

```
#include <stdio.h>

main()
{
```

```
int nb1, nb2;       /* opérandes */
int choix;          /* choix de l'utilisateur */
int nombre;         /* résultat de l'opération logique */
int mem, i;         /* variables intermédiaire et de boucle */
nb1 = ;
nb2 = ;
                    /* choix de l'utilisateur */
printf ("votre choix (AND=1, OR=2, XOR=3) :");
scanf ("%d",&choix);
                    /* opération logique */
if (choix==1)
    nombre = nb1 & nb2;
else if (choix==2)
    nombre = nb1 | nb2;
else
    nombre = nb1^ nb2;
                    /* affichage binaire */
i = 0;
while (i<8*sizeof(int))
{
    mem = nombre << 1;
    mem >>= 1;
    putch (nombre==mem? '0' : '1');
    nombre <<= 1;
    i++;
}
```

EXERCICE N°4:

Ecrire le programme permettant de calculer le développement limité de $sin(x)$ avec une précision ϵ :

$$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - ... + \epsilon(x)$$

- *algorithme :*

  float x = ;
  float epsilon, resul;
  float precision = ;

  resul = x;
  *fact = 3*2*1;*
  n = 1;
  *epsilon = resul *x*x/fact;*
  tant que epsilon ¿precision faire
  debut
  resul = resul + epsilon;
  n += 2;
  *fact = fact*n*(n-1);*
  *epsilon = epsilon*x*x/fact;*
  fin

Programme en C :

```
#include <stdio.h>

main ()
{
```

```
    double x, epsilon, resul, precision;
    int n, fact;

    x = ;
    precision = ;
    resul = x;
    fact = 3*2*1;
    n = 1;
    epsilon = resul*x*x/fact;
    while (epsilon > precision)
    {
        resul = resul + epsilon;
        n += 2;
        fact = fact*n*(n-1);
        epsilon = epsilon *x*x/fact;
    }
}
```

EXERCICE N°5:

Ecrire et programmer en C l'algorithme de la multiplication par additions successives.

- *algorithme :*

  var result, a, b, b1;
  lire a et b;
  initialiser result à 0 et b1 à b;
  tant que b1 est positif, faire
  deb
  result := result + 1;
  b1 := b1 - 1;
  fin

Programme en C :

```
#include <stdio.h>

main ()
{
    int result, a, b, b1;

    a = ; b = ;
    result = 0; b1 = b;
    while (b1 > 0)
    {
        result += a;
        b1--;
    }
}
```

EXERCICE N°6:

Même question pour la multiplication par divisions par 2 successives.

- *algorithme :*

var result, a, b, a1, b1;
lire a et b;
initialiser result à 0, a1 à a et b1 à b;
tant que b1 est positif, faire
deb
    si b1 est impaire faire
        result := result + a1;
    b1 := b1 / 2;
    *a1 := a1 * 2;*
fin

- *Expliquer le test de parité et l'intérêt des opérations de décalage.*
- *Avantage de cet algorithme par rapport au précédent.*

Programme en C :

```
#include <stdio.h>

main ()
{
   int result, a, b, a1, b1;

   a = ; b = ;
   result = 0; a1 = a; b1 = b;
   while (b1 > 0)
   {
       if (b1&0x01)         /* test de parité */
           result += a1;
       b1 >>= 1;
       a1 <<= 1;
   }
}
```

T.D. N°3 : boucles `do while()` et `for ()`

En utilisant à nouveau l'exercice factoriel, *expliquer les boucles* **for** *et* **do while**. *Introduire dans ce TD la notion de* fonction. *Les exercices seront ensuite systématiquement écris en tant que fonctions.*
PS : en raison d'un nouveau décalage du cours a mercredi, les étudiants n'auront pas encore vu la notion de fonction. Il convient donc de faire une introduction sur les fonctions qui soit la plus simple possible.

EXERCICE N°1:

Réécrire le programme factoriel $N!$ à l'aide de la boucle `for ()`.

- *Exemple d'algorithme :*

  var n, i, fact;
  initialiser fact à 1;
  pour i allant de 1 à n faire
  *fact = fact * i;*

Programme en C :

```
#include <stdio.h>

main ()
{
    int n = ;           /* ordre */
    int i, fact;        /* indice de boucle et resultat */

    fact = 1;
    for (i=1; i<=n; i++)
        fact *= i;

    printf("factoriel(%d) = %d \n",n,fact);
 }
```

Fonction correspondante :

```
#include <stdio.h>
                        /* prototype */
void factoriel(int);

main ()
{
   int n = ;            /* ordre */
   int result;

   result = factoriel(n);
   printf("factoriel(%d) = %d \n",n,result);
}

factoriel(int n)
{
   int i, fact;         /* indice de boucle et resultat */

   fact = 1;
   for (i=1; i<=n; i++)
       fact *= i;

   return fact;
}
```

EXERCICE N°2:

Ecrire la fonction retournant le N^{eme} terme, N donné en argument, de la suite de fibonacci:

$$fibo(0) = 1;\ fibo(1) = 1;$$
$$fibo(n \geq 2) = fibo(n-2) + fibo(n-1)$$

```
#include <stdio.h>
                 /* prototype */
int fibonacci(int);

main ()
{
    int n = ;
    int result;

    result = fibonacci(n);
    printf("fibo(%d) = %d \n",n,result);
}

void fibonacci(int n)
{
    int f1, f2;       /* variables intermediaires */
    int fibo;         /* variables resultat */

    if (n==0 || n==1)
        fibo = 1;
    else
        for (i=2, fibo=2; i<n; i++)
        {
            f1 = result;
            fibo = f1 + f2;
            f2 = f1;
        }

    return fibo;
 }
```

EXERCICE N°3:

Ecrire une fonction qui calcule les n^{emes} (n passé en argument) termes des suites entières U_n et V_n définies ci-dessous et qui retourne en fonction d'un autre paramètre reçu en argument soit U_n, soit V_n :

$$\begin{cases} U_0 & = & 1 \\ U_n & = & V_{n-1} + 1 \end{cases} \qquad \begin{cases} V_0 & = & 0 \\ V_n & = & 2U_{n-1} \end{cases}$$

```
#include <stdio.h>
                 /* prototype */
 int suite(int, int);

main ()
{
    int choix = ;
    int n = ;
    int result;

    result = suite(n, choix);
    printf("suite(%d) = %d \n",n,result);
```

```
}

int suite(int n, int ch) /* n = indice, ch : 0 = U, 1 = V */
{
    int i,u,u_pr=1,v=0;      /* u_pr : sauvegarde val. pr. de u */

    for(i=1; i<=n; i++)      /* calcul des n premiers termes */
    {
        u = v+1;
        v = u_pr*2;
        u_pr = u;
    }
    return ch ? v : u;
}
```

EXERCICE N°4:

La formule de conversion des températures exprimées en degré Celsius en degré Fahrenheit est :

$$C = \frac{5}{9}F - 32$$

Ecrire une fonction permettant d'afficher une liste d'équivalence pour des températures comprises entre 0^oF et 300^oF. On choisit un incrément de 10^oF. On écrira cette fonction en utilisant successivement une boucle **for()**, **while()** et **do...while()**.
Boucle `for (;;)`

```
int temp(void)
{
    int i;

    for (i=0; i<=300; i+=10)
        printf("%d degré Celsius equivaut à %f degré Fahrenheit \n", i, 1.8*i+32);
}
```

Boucle `while ()`

```
int temp(void)
{
    int i=0;

    while (i<=300)
    {
        printf("%d degré Celsius equivaut à %f degré Fahrenheit \n", i, 1.8*i+32);
        i+=10;
    }
}
```

Boucle `do while ()`

```
int temp(void)
{
    int i=0;

    do
    {
        printf("%d degré Celsius equivaut à %f degré Fahrenheit \n", i, 1.8*i+32);
        i += 10;
```

```
        }
        while (i<=300);
    }
```

EXERCICE N°5:

Ecrire une fonction donnant en retour la somme des entiers pairs inférieurs à N, N passé en argument de la fonction.

T.D. N°4 : Tableaux et algorithmes de parcours

EXERCICE N°1:

Ecrire une fonction qui affiche le quantième d'une date. Il faudra mémoriser la longueur de chaque mois dans un tableau global.
Programme en C :

```
#include <stdio.h>

int nbjours[12] = {31,28,31,30,31,30,31,31,30,31,30,31};

main ()
{
    int jour, mois, annee;

    printf ("entrez la date (JJ MM AAAA) : ");
    scanf ("%d %d %d", &jour, &mois, &annee);
    printf("\nQuantieme de la date : %d", calcul(jour,mois));
}

int calcul (int jr, int ms)
{
    int i, quant=0;

    if (ms>1)
        for (i=0; i<ms-1; i++)
            quant += nbjours[i];

    return quant+jr;
}
```

EXERCICE N°2:

Ecrire une fonction calculant la moyenne des éléments d'un tableau de réels.

- *Exemple d'algorithme :*

```
fonction moyenne
arguments :
        var (tableau de floats) tab;
        var (int) dim;
debut
        var (int) i;
        var (float) moyenne;
        initialiser moyenne à 0;
        pour i allant de 0 à dim-1 faire
                moyenne := moyenne + tab[i];
        retourner moyenne/dim;
fin
```

```
float moyenne (float tab[], int dim)
{
    int i;
```

```
    float moy = 0;

    for (i=0; i<dim;i++)
        moy += tab[i];

    return moy/dim;
}
```

EXERCICE N°3:

Ecrire la fonction de recherche du maximum dans un tableau de réels,

- *Exemple d'algorithme :*

  fonction maximum
  arguments:
  var (tableau de floats) tab;
  var (int) dim;
  debut
  var (int) i;
  var (float) max;
  initialiser max à 0;
  pour i allant de 0 à dim-1 faire
  si max < tab[i] faire max := tab[i];
  retourner max
  fin

```
float maximum (float tab[], int dim)
{
    int i;
    float max = 0;

    for (i=0; i<dim;i++)
        max < tab[i]? max=tab[i] : 1;

    return max;
}
```

EXERCICE N°4:

Ecrire une fonction calculant le produit scalaire de deux vecteurs.

- *Exemple d'algorithme :*

  fontion produit
  arguments :
  var (int) dim;
  var (tableau de floats) vec1, vec2;
  debut :
  var (int) i;
  var (float) prod;
  initialiser prod à 0;
  pour i allant de 0 à dim-1 faire
  *prod := prod + vec1[i]*vec2[i];*
  retourner prod;
  fin

```
float produit (float vec1[], float vec2[], int dim)
{
    int i;
    float prod = 0;

    for (i=0; i<dim; i++)
            prod = prod + vec1[i]*vec2[i];

    return prod;
}
```

T.D. N°5 : Gestion des Piles et des Files, tableaux 2D

EXERCICE N°1:

Construire un programme de gestion d'une *pile* de données (nombres réels). On Ecrira les fonctions `void init()` d'initialisation de la pile, `void push(float)` et `float pop()` de gestion de la pile.

- *La résolution de cet exercice ne devrait pas poser de problème aux étudiants puisqu'ils ont déjà manipulé les tableaux, d'une part, et que d'autre part ils savent déjà comment fonctionne une pile de données (vu en assembleur).*
- *Je propose donc que l'on insiste sur l'aspect « structure des données » du problème. Un retour pourra d'ailleurs être fait plus tard lorsque l'on abordera les structures.*
- *On implémentera une pile de type « LIFO » en envisageant un stockage en tableau « static » et « global ». Les paramètres comme la taille maximum de la pile pourront être des constantes* `define`*. Enfin, le pointeur de pile sera également déclaré en variable globale. Il pointera sur la prochaine case à remplir du tableau.*
- *Mettre en évidence les défauts de cette approche, notamment les déclarations en « globales ». Un retour sur ce problème permettra plus tard de mettre en évidence l'intérêt des structures (regroupement des données et écriture de fonctions plus générales).*
- *Aborder la notion de « gestion des erreurs » en prévoyant par exemple les débordements de pile.*
- *pour la fonction* `void init(void)`*, on pourra discuter de l'intérêt d'initialiser le tableau à 0.*
- *Ajouter le cas échéant une fonction retournant la taille courante de la pile*

Exemple de fonctions en C :

```
#include <stdio.h>

#define PILE_TAILLEMAX     100     /* nbre max d'elements dans la pile */

float p_tab[PILE_TAILLEMAX];   /* tableau de stockage */
int    p_offset;                    /* pointeur de pile */

void init(void)         /*  fn. d'initialisation de la pile */
{
    int i;
    for(i=0; i<p_tab; i++)
        p_tab[i] = 0;
    p_offset = 0;
}

void push(float x)       /* ajout d'un nombre dans la pile */
{
    if (p_offset==PILE_TAILLEMAX)
        printf ("débordement de pile. \n");
    else
        p_tab[p_offset++] = x;
}

float pop(void)       /* retrait d'un nombre */
{
    if (p_offset==0)
    {
```

```
        printf ("pile vide. \n");
        return 0;
    }
    else
        return p_tab[--p_offset];
}

int getsize(void)     /* taille courante de la pile */
{
    return p_offset;
}
```

EXERCICE N°2:

Reprendre le programme précédent pour l'adapter à la gestion d'une *file* de données.

- *Seulement si vous avez le temps (à poser en exercice pour la prochaine séance ?)*
- *La gestion d'une file (FIFO) est plus complexe que la gestion d'une pile. La donner en exercice pour la prochaine séance est peut être un peu délicat.*
- *On peux détailler au moins trois méthodes de gestion. La première, la plus simple, consiste à opérer un décalage des éléments du tableau à chaque défilement. On considère deux indices de tableaux, l'un repérant le début de la file (prochain nombre à défiler) et l'autre indiquant la prochaine case à remplir (fin de file). L'indice de début de file reste constant et pointe sur le premier élément du tableau. L'indice de fin de file est incrémenté à chaque enfilement et décrémenté à chaque défilement. On remarquera que l'indice de début de file n'est pas indispensable ici.*
- *La deuxième méthode évite d'opérer un décalage des éléments du tableau à chaque défilement. Le pointeur de début de file doit donc être incrémenté à chaque opération de défilement. Le pointeur de fin de file n'est incrémenté qu'aux opérations d'enfilement. Se pose alors le problème du pointeur de fin de file en bout de tableau tandis que l'indice de début de file n'est pas en début de tableau. Il faut dans ce cas opérer un décalage de l'ensemble des données vers le début du tableau afin de pouvoir enfiler de nouveaux éléments.*
- *La troisième méthode est une gestion circulaire de la file : les indices sont incrémentés modulo la taille du tableau. Je propose de ne pas aborder cette méthode afin d'éviter un découragement des étudiants.*

Première méthode :

```
#define FILE_TAILLEMAX          100

float   f_tab[FILE_TAILLEMAX];
int    f_deb;                       /* premier élément de la file */
int    f_fin;                       /* dernier élément de la file */

void init(void)
{
    f_deb = f_fin = 0;              /* la file est vide */
}

void enfiler(float x)
{
    if (f_fin==FILE_TAILLMAX)       /* la file est elle pleine ? */
        printf("file pleine. \n");
    else
        f_tab[f_fin++] = x;
```

```
}

float defiler(void)
{
    float elem = 0;
    int i;

    if (f_fin==0)                   /* la file est elle vide ? */
        printf("file vide. \n");
    else
    {
        elem = f_tab[f_deb];        /* on retire l'élément */
        for (i=f_deb+1; i<f_fin; i++)    /* on remonte les autres éléments */
            f_tab[i-1] = f_tab[i];
        f_fin--;
    }

    return elem;
}
```

Deuxième méthode :

```
#define FILE_TAILLEMAX          100

float   f_tab[FILE_TAILLEMAX];
int    f_deb;                        /* premier élément de la file */
int    f_fin;                        /* dernier élément de la file */

void init(void)
{
    f_deb = f_fin = 0;               /* la file est vide */
}

void enfiler(float x)
{
    int i;

    if (f_fin==FILE_TAILLMAX)        /* fin du tableau atteint ? */
        if (f_deb==0)                /* la file est-elle pleine ? */
            printf("file pleine. \n");
        else
        {
            for (i=f_deb+1; i<f_fin; i++)     /* on remonte la file */
                f_tab[i-f_deb-1] = f_tab[i];
            f_fin -= f_deb+1;                 /* mise à jour des indices */
            f_deb = 0;
            f_tab[f_fin++] = x;               /* rangement */
        }
    else
        f_tab[f_fin++] = x;
}

float defiler(void)
{
    float elem = 0;
    int i;

    if (f_deb==f_fin)               /* la file est elle vide ? */
        printf("file vide. \n");
    else
```

```
        elem = f_tab[f_deb++];

    return elem;
}
```

EXERCICE N°3:

Ecrire la fonction de multiplication de deux matrices.
un exemple de fonction :

```
void mult(int t1[N][M], int t2[M][L], int t3[N][L], int N, int M, int L)
{
    int i, j, k;
    for (i=0; i<N; i++)
        for (j=0; j<L; j++)
            for (k=0; k<M; k++)
                t3[i][j] = t1[i][k] * t2[k][j];
}
```

EXERCICE N°4:

Ecrire le programme permettant de remplir les 10 premières lignes du triangle de Pascal.

T.D. N°6 : Pointeurs et allocation dynamique

EXERCICE N°1:

Expliquer les différences entre `(*p)++`, `*p++`, `*(p++)`

- `(*p)++` *: le contenu de l'adresse sur laquelle pointe* `p` *est incrémenté.*
- `*p++` *: équivalent au précédent car* `*` *est prioritaire par rapport à l'opérateur* `++`*.*
- `*(p++)` *: c'est la valeur de l'adresse qui est incrémentée (*`p + sizeof(type)`*).*

EXERCICE N°2:

Ecrire une fonction d'échange de valeurs de deux variables réelles données en argument.

- *Exemple de programme :*

```
void echange (double *var1, double *var2)
{
    double tmp;
    tmp = *var1;
    *var1 = *var2;
    *var2 = tmp;
}
```

EXERCICE N°3:

Inverser un tableau de réels en utilisant les pointeurs. On n'utilisera pas la notation pointeur pour passer l'adresse du tableau en argument de la fonction.

- *Exemple de programme :*

```
void inverse (double tab[], int n)
{
    double tmp;
    int i;
    for (i=0; i<n/2; i++)
    {
        tmp = *(tab+i);
        *(tab+i) = *(tab+n-i);
        *(tab+n-i) = tmp;
    }
}
```

EXERCICE N°4:

Reprendre la question précédente en utilisant cette fois la notation pointeur pour passer l'adresse du tableau en argument.

- *Exemple de programme :*

```
void inverse (double *tab, int n)
{
    double tmp;
    int i;
    for (i=0; i<n/2; i++)
    {
        tmp = *(tab+i);
        *(tab+i) = *(tab+n-i);
        *(tab+n-i) = tmp;
    }
}
```

EXERCICE N°5:

Ecrire une fonction de saisie des éléments d'un tableau de réels. Le nombre d'éléments du tableau sera donné en argument de la fonction qui retournera l'adresse du tableau. Ce tableau devra être alloué dynamiquement. Une gestion des erreurs permettra de renvoyer le pointeur `NULL` en cas d'erreur d'allocation mémoire.

- *Exemple de programme :*

```
double *tab_nouv (int n)
{
    double *t;

    t = (double *) malloc (n*sizeof(double));
    if (t!=NULL)
        for (i=0; i<n; i++)
        {
            printf("tapez la valeur N.%d du tableau : ",i);
            scanf("%lf",t+i);
            printf("\n");
        }

    return t;
}
```

EXERCICE N°6:

Ecrire la fonction qui affiche puis libère la mémoire d'un tableau alloué dynamiquement, comme dans l'exercice précédent.

- *Exemple de programme :*

```
double *tab_term (double *t, int n)
{
    for (i=0; i<n; i++)
        printf("tableau[%d] = %lf",i,*(t+i));

    free (t);
}
```

T.D. N°7 : Le traitement des chaînes de caractères

EXERCICE N°1:

Ecrire un programme qui déclare et initialise un tableau de cinq chaînes de caractères, et qui affiche la taille exacte de chacune des chaînes.

- *Dans cet exercice, comme dans les suivants, on considère que l'on ne connaît pas* `<string.h>` *...*
- *Les Etudiants pourraient découvrir la librairie de traitement des chaines de caractères en TP, par exemple.*
- *Tant qu'à faire, utilisons les pointeurs dans la résolution de ces exercices !*
- *Les corrections proposées peuvent être un peu difficiles pour les étudiants. Je pense qu'il ne faut pas hésiter, dans un premier temps, à proposer des versions volontairement pédagogiques.*
- *Exemple de programme :*

```
# include <stdio.h>

char tableau[][40] = {"Impossible n'est pas informatique",
              "Il n'y a pas de problème",
              "qui ne soit sans solution.",
              "Donc s'il n'y a pas de solution,",
              "c'est qu'il n'y a pas de problème !"
               };

main ()
{
    int i;
    for (i=0; i<5; i++)
        printf ("Taille de '%s' = %d \n", tableau[i],
            chnlgr(tableau[i]));
}

int chnlgr (char *ch)
{
    int lg = 0;
    while (*ch++ != '\0')  /* post-incr. et priorité de '++' sur '*' */
        lg++;

    return lg;
}
```

EXERCICE N°2:

Ecrire la fonction `char *chncpi(char *dest, char *src)` qui recopie les caractères de la chaîne `src` dans la chaîne `dest`. On supposera la chaîne destination déjà allouée.

- *on évite de perdre l'adresse de la chaîne destination puisqu'on doit la retourner !*
- *on n'oublie pas le caractère de fin de chaîne !*
- *Exemple de programme :*

```
char *chncpi (char *dest, char *src)
{
    char *d;
    for (d=dest; *src != '\0'; *d++ = *src++);
    *d = '\0';

    return dest;
}
```

Utilisation avec le tableau de l'exercice précédent :

```
main()
{
    int i;
    char dest [1024];
    for (i=0; i<5; i++)
        printf ("copie de tableau[%d] =  %s \n",i,
            chncpi(dest, tableau[i]));
}
```

EXERCICE N°3:

Ecrire une fonction qui change toutes les lettres minuscules d'une chaîne dont l'adresse est passée en argument, en majuscules. Le prototype de la fonction prendra la forme `char *min2maj(char *dest, char *source);`. On supposera la chaîne destination déjà allouée. La fonction retournera l'adresse de la chaine destination.

- *Exemple de programme :*

```
char *min2maj (char *dest, char *source)
{
    char *ptr = dest;
    for(; *source != '\0'; source++, dest++)
        *dest=*source<='z' && *source>='a'?*source-'a'+'A':*source;
    *ptr = '\0';

    return dest;
}
```

Utilisation avec le tableau du 1er exercice :

```
main()
{
    int i;
    char dest [1024];
    for (i=0; i<5; i++)
        printf ("min2maj[%d] =  %s \n",i, min2maj(dest, tableau[i]));
}
```

EXERCICE N°4:

Ecrire une fonction d'inversion d'une chaine de caractères en utilisant le *mécanisme de pile* étudié au TD 5. On supposera que la pile est une pile de caractères.

- *Exemple de programme :*

```
#define PILE_TAILLEMAX     100  /* nbre max d'elements dans la pile */

char p_tab[PILE_TAILLEMAX];     /* tableau de stockage */
int    p_offset;                /* pointeur de pile */

char *chninv (char *dest, char *src)
{
    int lg  = chnlgr();         /* nombre de caractères */
    char *d = dest;

    init();                     /* initialisation de la pile */
    while (*src++ != '\0')      /* empiler les caractères */
        push(*src);
    while (lg--)                /* dépiler les caractères */
        *dest++ = pop();
    *dest = '\0';

    return d;
}
```

EXERCICE N°5:

Ecrire la fonction `int chnncmp(char *s1, char *s2, int n)` qui compare les `n` premiers caractères de `s1` et de `s2` et renvoie 0 si `s1 == s2`, une valeur positive si `s1 >s2` et une valeur négative si `s1 <s2`.

– *Exemple de programme :*

```
int chnncmp(char *s1, char *s2, int n)
{
    int resul;

    while(n!=0)
    {
        if (*s1 != *s2)    /* caractères différents */
        {
             resul = (int)*s1 - (int)*s2;
             break;
        }                  /* caractères identiques */
        else if (*s1 == '\0')
        {                  /* fin de chaine */
             resul = 0;
             break;
        }                  /* caractères suivants */
        s1++;
        s2++;
        n--;
    }                      /* n caractères analysés */

    return resul;
}
```

EXERCICE N°6:

Ecrire une fonction retournant 1 ou 0 selon qu'elle détecte ou non la présence d'un palindrome dans la chaine de caractères passée en argument. La fonction devra pouvoir traiter des mots autant que des

phrases ex : *Laval* ou *Esope reste ici et se repose.*

T.D. N°8 : Algorithmes de tri

EXERCICE N°1:

Ecrire une fonction de recherche séquentielle d'un élément dans un tableau croissant.
Exemple de fonction :

```
int rech_seq(float *t, int n, float e)
{
    int i;

    for (i=0; i<n && t[i]!=e; i++);

    return i==n? -1 : i;
}
```

EXERCICE N°2:

Ecrire une fonction de recherche dichotomique d'un élément dans un tableau. On supposera les éléments du tableau ordonnés par ordre croissant. Discuter de l'intérêt de la méthode en comparant avec la méthode de recherche exhaustive de la question précédente.
Exemple de fonction :

```
int rech_dich(float *t, int n, float e)
{
    int deb, fin, j;
    int trouve;

    if (e<t[0] || e>t[n-1])
        trouve = 0;
    else
    {
        while(e>=t[deb+1])
        {
            j = (deb+fin)/2;
            e >= t[j]? deb=j : fin=j;
        }
        trouve = (t[deb]==e);
    }

    return trouve? deb : -1;
}
```

EXERCICE N°3:

Ecrire le programme de *tri par sélection et permutation*. Application par exemple au tri par ordre alphabétique d'une liste de noms rangés dans un tableau de chaines (utiliser la fonction `int strcmp(char *, char *)`).
Exemple de fonction :

```
void tri_selec(float *t, int n)
{
    int i, min;
```

```
    float q;

    for (i=0; i<n; i++)
    {
        min = i;
        for (j=i+1; j<n; j++)       /* sélection du min du tableau restant */
            t[j]<t[min]? min=j : 1;
        q = t[min];                 /* permutation */
        t[min] = t[i];
        t[i] = q;
    }
}
```

EXERCICE N°4:

Ecrire le programme du *tri par bulle*
Exemple de fonction :

```
void tri_bulle(float *t; int n)
{
    int i, k;
    float q;

    for (i=n-1; i>0; i--)
    {
        for (k=0; k<i-1; k++)    /* remonter le maximum du sous tableau */
            if (t[k+1]<t[k])     /* inférieur à la position courante */
            {
                q = t[k+1];
                t[k+1] = t[k];
                t[k] = q;
            }
    }
}
```

T.D. N°9 : Structure des données

EXERCICE N°1:

Définir la structure permettant de stocker un point. Ecrire les fonctions permettant de saisir un point et d'afficher un point. Définir un tableau permettant de stocker un ensemble de points (version statique et version dynamique).
Exemple de structure et de fonctions :

```
typedef struct t_point
{
    float x;
    float y;
}POINT;

void affiche_point(POINT p)
{
    printf("[%f %f]\n",p.x,p.y);
}

POINT saisir_point()
{
    POINT p;
    printf(" x="); scanf("%f",&p.x);
    printf("\n y="); scanf("%f",&p.y);

    return p;
}

#define TAILLE_MAX  100
```

Déclaration d'un tableau en static :

```
POINT tab[TAILLE_MAX];
```

Fonction d'allocation dynamique d'un tableau :

```
POINT *new_tab_point(int N)
{
    POINT *tab = (POINT *)malloc(sizeof(POINT)*N);
    if (!tab)
        printf("Allocation mémoire impossible");
    return tab;
}
```

EXERCICE N°2:

Définir la structure permettant de représenter un nombre complexe. définir les fonctions complexes suivantes (passages par valeur):

- `double imag(Complex z)`,
- `double real(Complex z)`,
- `Complex mul(Complex z1, Complex z2)`,

– `double abs(Complex z)`,

– Réécrire ces fonctions en utilisant le passage par adresses.

```
#include <math.h>

typedef struct t_complex
{
    double re;
    double im;
}COMPLEX;

double imag(COMPLEX z)
{
    return z.im;
}

double real(COMPLEX z)
{
    return z.re;
}

COMPLEX mul(COMPLEX z1, COMPLEX z2)
{
    COMPLEX z;
    z.re = z1.re*z2.re - z1.im*z2.im;
    z.im = z1.re*z2.im + Z1.im*z2.re;
    return z;
}

double abs(COMPLEX z)
{
    return sqrt(z.re*z.re + z.im*z.im);
}
```

En utilisant le passage d'arguments par adresses :

```
double imag(COMPLEX *z)
{
    return z->im;
}

double real(COMPLEX *z)
{
    return z->re;
}

COMPLEX* mul(COMPLEX *z1, COMPLEX *z2)
{
    COMPLEX *z = (COMPLEX*)malloc(sizeof(COMPLEX));
    z->re = z1->re*z2->re - z1->im*z2->im;
    z->im = z1->re*z2->im + Z1->im*z2->re;
    return z;
}

double abs(COMPLEX *z)
{
    return sqrt(z->re*z->re + z->im*z->im);
}
```

EXERCICE N°3:

Définir les structures nécessaires à la gestion des piles et files du TD 5.

```
#define PILE_TAILLEMAX   100
#define FILE_TAILLEMAX   100

typedef struct t_pile
{
    float  p_tab[PILE_TAILLEMAX];    /* tableau de stockage */
    int    p_offset;                 /* pointeur de pile */
}PILE;


typedef struct t_file
{
    float   f_tab[FILE_TAILLEMAX];
    int     f_deb;                   /* premier élément de la file */
    int     f_fin;                   /* dernier élément de la file */
}FILE;
```

EXERCICE N°4:

Ecrire un programme de saisie de données pour un répertoire (nom, prénom, téléphone). Ces données doivent être placées dans un tableau de structures, chacune d'elles contenant un enregistrement. Le programme devra contenir une fonction d'affichage de toutes les données.

```
#define MAX_FICHES  100

typedef struct t_fiche
{
    char   nom[64];
    char   prenom[64];
    char   tel[11];
}FICHE;

FICHE tab[MAX_FICHES];

void saisir()
{
    int termine = 0;
    int num;
    char resp;

    while (!termine)
    {
        printf ("\n num. de la fiche à saisir = "); scanf("%d",&num);
        if (num<0 || num>=MAX_FICHES)
            printf ("erreur, recommencez. \n");
        else
        {
            printf ("nom="); scanf ("%s",tab[num].nom);
            printf ("prénom="); scanf ("%s",tab[num].prenom);
            printf ("tel="); scanf ("%s",tab[num].tel);

            printf ("fiche suivante (o,n) ?"); scanf("%c",&resp);
            termine = resp=='o'? 1 : 0;
```

```
        }
    }
}

void afficher()
{
    etc...
}
```

EXERCICE N°5:

On souhaite gérer un stock d'articles. Il est donc nécessaire de déterminer la structure de données permettant de tous les représenter. pour cela on dispose des renseignements suivants :

- Tous les articles ont une référence et une désignation
- Chaque article fait partie d'une catégorie qui peut être *électroménager*, *produit comestible* où *vêtement*.
- Les vêtements ont tous une taille, un coloris et une liste de pays où ils peuvent être vendus (maximum 5).
- Les produits comestibles ont une provenance et une date limite de consommation seulement s'il s'agit de viandes ou de poissons. Dans les autres cas, ils n'ont pas de caractéristiques particulières.
- Les appareils électroménagers sont caractérisés par une alimentation électrique (c'est à dire par un voltage, une impédance et une puissance).

Donner les structures de données permettant de gérer 500 articles en mémoire. Ecrire les instructions permettant d'affecter à un des articles les informations indiquant qu'il s'agit d'oeufs en provenance du Poitou. Ecrire les fonctions permettant d'affecter à un des articles les informations indiquant dans quel pays une chemise rouge de taille 46 est vendable.
Exemple de structures (les unions n'ont pas été vues en cours...) :

```
typedef struct t_vetement
{
    int taille;
    char couleur[16];
    char pays[5][32];
}VETEMENT;

typedef struct t_viande_ou_poisson
{
    char provenance[32];
    char date_limite[8];
}VIANDE_OU_POISSON;

typedef struct t_autre
{
    char provenance[32];
}AUTRE_COMESTIBLE;

typedef union t_comestible
{
    VIANDE_OU_POISSON proteines;
    AUTRE_COMESTIBLE  autre;
}COMESTIBLE;
```

```
typedef struct t_electromenager
{
    int volt;
    int imp;
    int puiss;
}ELECTROMENAGER;

typedef union t_categorie
{
    ELECTROMENAGER  menager;
    COMESTIBLE      produit;
    VETEMENT        habit;
}CATEGORIE;

typedef struct t_article
{
    int       ref;
    char      design[10];
    CATEGORIE cat;
}ARTICLE;
```

Une structure de données permettant de gérer 500 articles en mémoire :

```
ARTICLE magasin [500];
```

Affecter à l'article N. i les informations indiquant qu'il s'agit d'oeufs en provenance du Poitou :

```
strcpy(magasin[i].design,"oeufs");
strcpy(magasin[i].cat.produit.autre.provenance,"Poitou");
```

Affecter à l'article N. i les informations indiquant dans quel pays une chemise rouge de taille 46 est vendable.

```
strcpy(magasin[i].cat.design,"chemise");
magasin[i].cat.produit.habit.taille = 46;
strcpy(magasin[i].cat.produit.habit.couleur,"rouge");
strcpy(magasin[i].cat.produit.habit.pays[0],"???");
strcpy(magasin[i].cat.produit.habit.pays[1],"???");
....
strcpy(magasin[i].cat.produit.habit.pays[4],"???");
```

T.D. N°10 : Listes chaînées

EXERCICE N°1:

Rappeler l'intérêt du passage par adresse des structures.

- Le passage par valeur d'une structure en argument d'une fonction est possible mais a pour principal inconvénient la recopie des membres de la structure. Le passage par adresse n'entraîne que la recopie de l'adresse de la structure. Il est donc plus rapide.
- La modification par la fonction d'une structure passée par valeur en argument n'affecte pas la structure donnée en argument. La structure modifiée ne peut être récupérée que par l'instruction `return`. Le passage par adresse évite ce problème.
- ...

EXERCICE N°2:

Manipulation des listes chaînées : reprendre l'exercice 2 du TD 9 (gestion d'un répertoire).

- Redéfinir la structure représentant une adresse et permettant un *chaînage avant* des éléments de la liste ;
- Définir la fonction de création et de saisie d'un nouvel élément ;
- Ecrire la fonction d'ajout d'un élément en tête de liste ;
- Ecrire la fonction d'ajout d'un élément en fin de liste ;
- Ecrire la fonction permettant l'affichage de la liste ;
- Ecrire la fonction permettant de retirer un élément de la liste selon le critère suivant : *nom=...* ET *prenom=...*;
- Ecrire la fonction permettant de saisir une liste
- Ecrire la fonction permettant de libérer la liste.

Exemples de fonctions :

- *Ces fonctions sont données a titre indicatif ;*
- *Le style employé est déjà moins "pédagogique". N'hésitez pas à clarifier les expressions conditionnant les choix dans les if() ...*
- *Les boites et les flèches vers les boites dessinées sur le tableaux seront certainement indispensables !*

```
typedef struct t_fiche
{
    char   nom[64];
    char   prenom[64];
    char   tel[11];
    struct t_fiche *suivant;          /* !! FICHE pas encore défini ici !! */
}FICHE;

FICHE* repertoire = NULL;             /* pointeur de liste */

FICHE* creer_fiche()                  /* création d'un nouvel élément */
```

```
{
    FICHE *p_elm = (FICHE*)malloc(sizeof(FICHE));
    if (!p_elm)
        printf("creer_fiche: allocation mémoire impossible. \n");
    else
        p_elem->suivant = 0;          /* aucun elem. suivant */

    return elm;
}

void saisir_fiche(FICHE *f)
{
    printf ("nom="); scanf ("%s",f->nom);
    printf ("prénom="); scanf ("%s",f->prenom);
    printf ("tel="); scanf ("%s",f->tel);
}

FICHE *ajout_deb_liste(FICHE *liste, FICHE *f)
{                                     /* retourne le nouveau ptr. de liste */
    f->suivant = liste;               /* insertion entre pointeur de */
    return f;                         /* liste et 1er element        */
}

FICHE *ajout_fin_liste(FICHE *liste, FICHE *f)
{
    FICHE *ptr = liste;

    if (!liste)                       /* liste vide */
        liste = f;
    else
    {
        while (ptr->suivant)          /* aller en bout de liste */
            ptr = ptr->suivant;
        ptr->suivant = f;             /* insertion */
    }
    return liste;                     /* nv. ptr. de liste */
}

FICHE *supprime(FICHE *liste, char *nom, char *prenom)
{
    FICHE *ptr = liste;               /* parcours la liste */
    FICHE *pr = 0;                    /* pointe sur l'elm. precedent */

    if (!liste)                       /* liste vide */
        printf ("supprime: la liste est vide.\n");
    else
    {
        while (ptr && (strcmp(ptr->nom,nom) || strcmp(ptr->prenom,prenom)))
        {                             /* tant qu'on est pas au bout */
            pr = ptr;                 /* et tant que pas trouvé */
            ptr = ptr->suivant;
        }
        if (!ptr)                     /* pas dans la liste */
              printf("supprime: fiche non trouvée.\n");
        else if (!pr)                 /* 1er elm. ou unique elm. */
        {
            liste = ptr->suivant;
            free (ptr);
        }
```

```
        else                        /* au milieu ou a la fin
        {
            pr->suivant = ptr->suivant;
            free (ptr);
        }
    }

    return liste;                   /* retourne le ptr. de liste */
}

void affiche_liste(FICHE *liste)
{
    printf("Affichage du répertoire. \n");

    while (liste)
    {
        printf ("\t nom    : %s\n",liste->nom);
        printf ("\t prénom : %s\n",liste->prenom);
        printf ("\t tel    : %s\n\n",liste->tel);
        liste = liste->suivant;
    }
    printf ("Affichage terminé. \n\n");
}

FICHE *saisir_liste()               /* saisie d'un réperotoire */
{
    int termine;
    int num;
    char resp;
    FICHE *liste=0, *f;

    printf("Voulez vous saisir une liste ?"); scanf("%c",&resp); printf("\n");
    termine = (resp=='o'||resp=='O')? 0 : 1;

    while (!termine)
    {
        f = creer_fiche();
        if (!f)
            termine = 1;
        else
        {
            saisir_fiche(f);
            liste = ajout_deb_liste(liste,f);
        }

        printf("Fiche suivante ?"); scanf("%c",&resp); printf("\n");
        termine = (resp=='o'||resp=='O')? 0 : 1;
    }

    printf("Au revoir ...\n\n");
    return liste;
}


FICHE *detruit_liste(FICHE *liste)
{
    FICHE *ptr;

    while(liste)
```

```
    {
        ptr = liste->suivant;
        free (liste);
        liste = ptr;
    }
    return 0;
}
```