Paramètres d'une fonction

- A l'appel d'une fonction avec paramètres, la valeur ou l'adresse du paramètre effectif est transmise au paramètre formel correspondant.
- **Passage par valeur:** Si le nom d'une variable (sauf le nom d'un tableau) apparaît dans l'appel d'une fonction, comme paramètre effectif, alors la fonction appelée reçoit la valeur de cette variable. Cette valeur sera recopiée dans le paramètre formel correspondant. Après l'appel de cette fonction, la valeur du paramètre effectif n'est pas modifiée.

Paramètres d'une fonction

- **Passage par adresse:** Lorsqu'on veut qu'une fonction puisse modifier la valeur d'une variable passée comme paramètre effectif, il faut transmettre l'adresse de cette variable. La fonction appelée range l'adresse transmise dans une variable pointeur et la fonction travaille directement sur l'objet transmis. Un tableau est toujours passé par adresse puisque le nom d'un tableau est un pointeur constant (c'est-à-dire une adresse).

Transmission des paramètres en C

- La transmission des paramètres en C se fait toujours par valeur
- Pour effectuer une transmission par adresse en C, on déclare le paramètre formel de type pointeur et lors d'un appel de la fonction, on envoie l'adresse et non la valeur du paramètre effectif
- Exemple :

```
void Increment (int x, int *y)
{ x=x+1;
 *y =*y+1; }
main( )
{ int n = 3, m=3;
 Increment (n, &m);
 printf("n = %d et m=%d \n", n,m);      }
```

Résultat :

n=3 et m= 4

Exercices

```
#include<stdio.h>
void fct_val(int)     //passage d'argument par valeur
void fct_adr(int*) //passage d'argument par adresse
void main( ) {
   int i = 4 ;
   fct_val( i );
   printf("Après passage d'argument par valeur   i :  %d\n", i );
fct_adr( &i );
  printf("Après passage d'argument par adresse  i :  %d\n", i );
}

 void fct_val( int a ) { a = a + 3; }
void fct_adr( int *a ) {      *a = *a + 3;       }
```

Exercices

Écrire un programme C qui saisie et affiche un tableau d'entiers en utilisant des fonctions

1. Permet de saisir la dimension du tableau: saisie_N

2. Permet de saisir les éléments du tableau : saisie_T

3. Permet d'afficher le tableau: affichage_T

Solution

```
#include<stdio.h>
void saisie_N( int *n ) {
 printf("n (<=20) : ? " ) ;
  scanf("%d", n ); }

Void saisie_T( intT[ ] , int n) {
 int i ;
      for( i=0 ; i<n ; i++ ) {
            printf("%d\t",i);
            scanf("%d",&T[i]);}
}
Void affichage_T( intT[ ] , int n) {
int i ;
       for( i=0 ; i<n ; i++ )
       printf("%d\t",T[i]);}
```

```
int main( ) {
 int T[20] , N;
saisie_N( &N );
saisie_T( T , N );
affichage_T( T , N );
return 0;
 }
```

Exercices

Écrire un programme C qui saisie et affiche une matrice d'entiers courts de M ligne et N colonne en utilisant des fonctions

1. Permet de saisir la dimension du tableau: saisie_M_N

2. Permet de saisir les éléments du tableau : saisie_A

3. Permet d'afficher le tableau: affichage_A

Solution

```
#include<stdio.h>
void saisie_M_N( short * m , short  *n ) {
printf("Entrer   m (<=20) et n (<=30) : ? " ) ;
scanf("%d", m , n ); }

Void saisie_A( short  B[ ][30] , int m , int n) {
  int i , j ;
       for( i = 0 ; i < m ; i++ )
          for ( j = 0 ; j < n ; j++ )
               scanf("%d", &B[ i ][ j ] ); }
```

Solution

```
voidaffichage_A( short  C[ ][30] , short m, short n) {
 int i , j  ;
for ( i = 0 ; i < m ;  i++  ) {
     for ( j = 0 ; j < n ; j++ )
                printf( "%d\t", C[ i ][ j ] );
                 printf("\n"); } }
int main( ) {
       short  A[20][30] , M , N;
       saisie_M_N( &M , &N );
       saisie_A( A , M , N );
       affichage_A( A , M,  N );
       return 0;
}
```

Récursivité

- Une fonction est récursive si elle contient dans sa définition un appel à elle même
- L'ordre de calcul est l'ordre inverse de l'appel de la fonction.
- Procédé pratique :Pour trouver une solution récursive d'un problème, on cherche à le décomposer en plusieurs sous-problèmes de même type, mais de taille inférieure. On procède de la manière suivante :

- Rechercher un cas trivial et sa solution (évaluation sans récursivité)
- Décomposer le cas général en cas plus simples eux aussi décomposables pour aboutir au cas trivial.

Récursivité

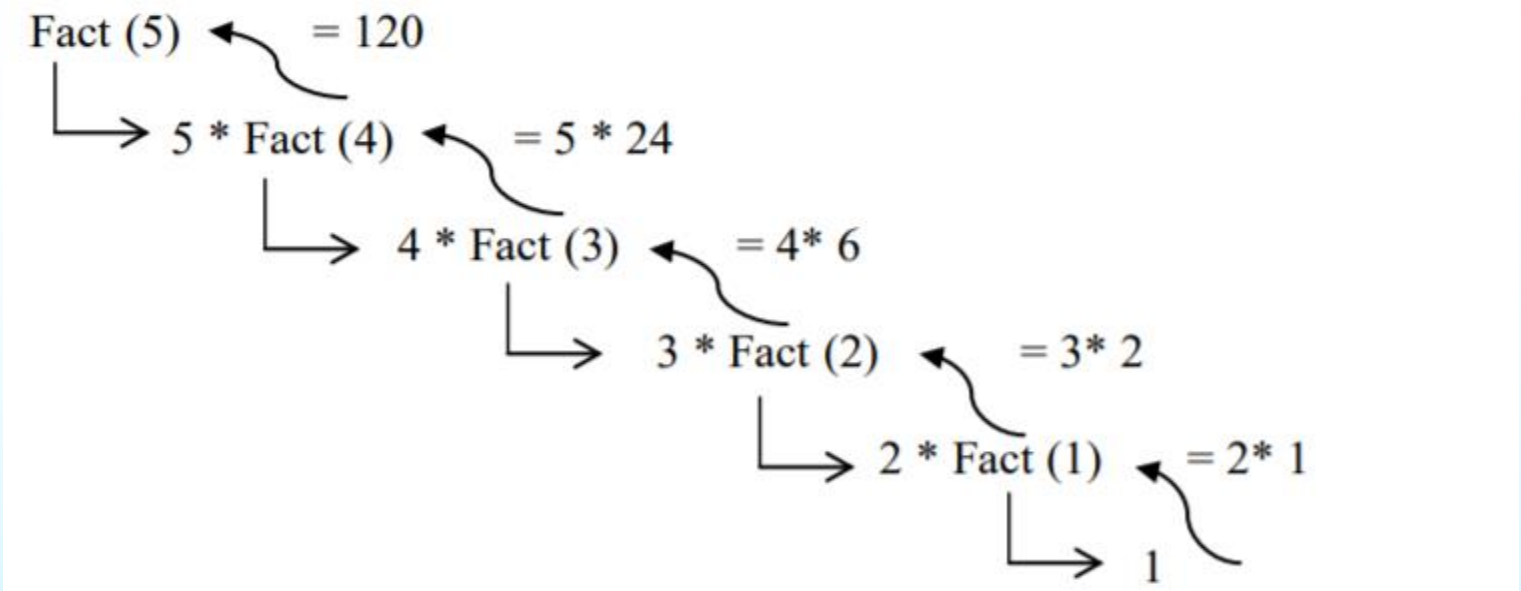
- Exemple : Calcul du factorielle

```
int fact (int n )
   {   if (n==0) /*cas trivial*/
                          return (1);
                  else
                  return (n* fact(n-1) );
   }
```

Remarque : l'ordre de calcul est l'ordre inverse de l'appel de la fonction

Récursivité

- Prenons la fonction récursive de la factorielle et déroulons la pour N = 5 : les flèches descendantes représentent les appels (de Fact (5) à Fact (1)) on arrive alors à l'exécution de fact (1) pour laquelle la condition d'arrêt est valide alors le retour se fait à rebous (dans le sens contraire) et à chaque appel correspond alors la valeur associée.



Fonctions récursives : exercice

1.Ecrire une fonction récursive qui calcule x^n, x un réel et n un entier positif.

2. Ecrire une fonction récursive qui réalise le produit (x*y) sachant que X et Y sont des entiers

3. Ecrire une fonction récursive qui calcule la valeur du nème terme de la suite de Fibonacci.

U(0)=U(1)=1

U(n)=U(n-1)+U(n-2)

Fonctions récursives : exercice

1.

```
int puiss( float x, int n )
    {   if (n==0) /*cas trivial*/
                        return (1);
                else
                return (x* puiss(x, n-1));
    }
```

2.

```
int produit (int x, int y )
    {   if (y==0) /*cas trivial*/
                        return (0);
                else
                return (x+ produit(x, y-1));
    }
```

Fonctions récursives : exercice

3.

```
int     Fib (int n)
  {     if (n==0 || n==1)
          return (1);
        else
        return ( Fib(n-1)+Fib(n-2));

  }
```

Fonctions récursives

- Le processus récursif remplace en quelque sorte la boucle, c'est-à-dire un processus itératif.
- Il est à noter que l'on traite le problème à l'envers : on part du nombre, et on remonte à rebours jusqu'à 1, pour pouvoir calculer la factorielle par exemple.
- Cet effet de rebours est caractéristique de la programmation récursive.

Types structures, unions et Types structures, unions et synonymes

Structures

- Une structure est un nouveau type de données constitué par un ensemble de variables (champs) qui peuvent être hétérogènes et de types différents

- La différence avec le type tableau est que les champs d'un tableau sont tous homogènes et du même type

- Les structures permettent de représenter des objets réels caractérisées par plusieurs informations, par exemple :
  - Une personne caractérisée par son nom (chaîne), son âge (entier), sa taille (réel), …
  - Une voiture caractérisée par sa marque (chaîne), sa couleur (chaîne), son année modèle(entier), …

Type structure : Exemples

- Une date est un objet réel défini par : jour (entier), mois (entier ou chaîne) et année(entier).

- Un nombre complexe est défini par sa parties réelle (réel) et sa partie imaginaire (réel).

- Un étudiant est défini par : nom (chaîne), prénom (chaîne), num_CIN (chaîne), code (entier), num_CNE (entier),….

- Un article est défini par : numéro (entier), libellé (chaine), quantité en stock (entier), prix (réel).

Déclaration d'une structure

- La déclaration d'une structure s'effectue en précisant le nom de la structure, ainsi que le nom et le type de ses champs :

- Syntaxe en C :

```
struct nom_structure {
        type 1 nom_champ1;
        type 2  nom_champ2;
        … …
        type N  nom_champN ;
        } ;
```

Déclaration d'une structure

- Par l'intermédiaire du nom de la structure, on peut déclarer plusieurs variables de ce type de structure chaque fois que c'est nécessaire.

- Exemple :

```
struct Personne {
        char Nom[20];
        int Age;
        float taille;
                };
```

Rq: Le nom d'une structure n'est pas un nom de variable, c'est le nom du type ou modèle de la structure

Déclaration d'une variable structure

- La déclaration d'une structure ne réserve pas d'espace mémoire
- La réservation se fait quand on définit des variables correspondant à ce modèle de structure. Ceci peut se faire soit :
  - après la déclaration de la structure, par exemple :

struct Personne p1, *p2, tab[10];

//p2 est une variable de type pointeur sur une structure Personne

// tab est une variable de type tableau de 10 éléments (de type Personne)

Déclaration d'une variable structure

- ou au moment de la déclaration de la structure

```
struct Personne {
                    char Nom[20];
                    int Age;
                    float taille;
                                    }; p1, *p2, tab[10];
```

Exercices

- Déclarer Les structure suivantes:

1. **Un article** est défini par : numéro (entier : short), libellé(chaîne de 29 caractères), quantité en stock (entier : short), prix (réel : float).

2. **Un étudiant** est défini par : code (entier :short) ; nom (chaîne : 29) ; prénom (chaîne : 19) ; adresse (une adresse)

Une adresse est défini par : numéro de domicile (entier : short) ; nom de la rue (chaîne : 29) code postale (entier : short) ; nom de la ville (chaîne : 19) ; nom du pays (chaîne : 19).

Exercices: solution 1

```
/* Déclaration du type structure article*/
struct article {
    short numero;  // un numéro qui identifie l'article
     char libelle[30];  // le nom de l'article
     short qte_stock;  //la quantité disponible en stock de l'article
  float prix;   //le prix avec lequel est commercialisé l'article } ;

/*Déclaration des variables du type structure article */
struct article art1 ; // art1 est une variable  structure article
struct article art1 , art2; // art1 et art2 deux variables structure article
struct article *Pt_art; // Pt_art est une variable pointeur susceptible de pointer une variable structure article
 struct article tab_art[30] ; //  tab_art est une variable tableau de 30 éléments de type structure article : tableau des articles
```

Exercices: solution 2

```
                    // type structure adresse
struct adresse  {
      short n_domicile;  /* numéro de la maison */
      char rue[30] ;     /* nom de la rue */
      short code_postale; // le code postale
      char ville[20] ;  /* nom de la ville */
      char pays[20] ;  /* nom du pays */ } ;
```

Exercices: solution 2

```
                    // type structure etudiant
struct etudiant{
        short code ;  /* code de l'étudiant */
        char nom[30] ;     /* nom de l'étudiant */
         char prenom[20] ;  // prénom de l'étudiant
          struct adresse adr;  // l'adresse de l'étudiant } ;
```

Exercices: solution 2

/*Déclaration des variables du type structure étudiant*/

```
struct etudiant etd1 , edt2,  // etd1 et edt2 deux variables
                *Pt_etd,  // Pt_etd est une variable
pointeur susceptible de pointer une structure étudiant
                tab_etd[30] ; //  tab_etd est tableau de 30
étudiants
```

Initialisation d'une structure

- Lors de la déclaration d'une variable structure, on peut initialiser ses champs avec une notation semblable à celle utilisée pour les tableaux en indiquant la liste des valeurs respectives entre accolades.

- Exemple :

```
struct date
        { unsigned short jour;
          char mois[10];
          unsigned short annee;
        } ;
```

struct date d1= {15,"Novembre", 2013};

Accès aux champs d'une structure

- L'accès à un champ d'une variable structure se fait par le nom de la variable suivi d'un point et du nom du champ

nom_variable.nom_champ

```
struct article
{
    short numero ;
   char libelle[30];
   short qte_stock;
   float prix;
} ;
```

Exemple :

struct article art;

/* Initialisation, depuis le clavier, des champs de la structure art */

scanf("%d %d %f",& art.numero, & art.qte_stock, &art.prix);

gets(art.libelle);

Accès aux champs d'une structure

```
/* Affichage du contenu de la structure art */

printf("Cette article a pour :  \n");
printf("\t numéro: %d \n",art.numero);
printf("\t libellé: %s \n", art.libelle) ;
printf("\t quantité en stck: %d \n", art.qte_stock) ; printf("\t
prix: %f \n", art.prix) ;
```

Accès à un champ via un pointeur de structure:

- Dans le cas d'une variable structure de type pointeur (ex : **struct Personne *p2**), on utilise en général l'opérateur **->** pour accéder aux champs nom_variable ->nom_champ (ex: p2->age)

```
struct article
{
    short numero ;
   char libelle[30];
   short qte_stock;
   float prix;
} ;
```

Exemple :

```
struct article *pt_art, art ;
 pt_art= &art ;
/* Initialisation, depuis le clavier, des champs de la structure
art via la variable pointeur pt_art*/
scanf("%d %d %f",&(pt_art->numero) , &(pt_art->qte_stock),
&(pt_art->prix) ) ;
 gets(pt_art->libelle);
```

Accès à un champ via un pointeur de structure:

```
/* Affichage du contenu de la structure art via la variable pointeur pt_art*/

 printf("Cette article a pour :  \n");
 printf("\t numéro: %d \n", pt_art->numero);
 printf("\t libellé: %s \n", pt_art->libelle) ;
 printf("\t quantité en stck: %d \n", pt_art->qte_stock) ;
printf("\t prix: %f \n", pt_art->prix) ;
```

Composition des structures

- Les structures peuvent être composées de champs de n'importe quel type connu: types de base, pointeur, tableau ou structure.

- Exemple de structure comportant un tableau et une structure :

```
struct Etudiant
        { int code;
          char Nom[20];
          struct date date_naissance;
          float notes[8]; // notes de l'étudiant dans 8 modules
        } E1,E2;
```

- on peut écrire E1.date naissance.annee pour accéder au champ annee de E1.date naissance qui est de type date

- E2.notes[3] représente la note du module 4 de l'étudiant E2

Opérations sur les variables structures

- Les structures sont manipulées en général champ par champ. Toutefois, la norme ANSI permet d'affecter une structure à une autre structure de même type, par exemple : struct Etudiant E1,E2;

  E2=E1; est une instruction valide qui recopie tous les champs de E1 dans les champs de E2

- Il n'est pas possible de comparer deux structures. Les instructions if(E1==E2) ou if(E1!=E2) ne sont pas permises
- L'opérateur & permet de récupérer l'adresse d'une variable structure (ex : struct Personne p1, *p2; **p2=&p1**)

Exemple:

```
struct article art= {1, "Ecran TFT 19" ,12 , 2500.0} , *pt_art ; pt_art = &art ;
pt_art->qte_stck = 10; // On modifie la quantité en stock
pt_art->prix = 2000.0; //On modifie le prix : l'article est en promotion
```

Opérations sur les variables structures

- L'opérateur sizeof permet de récupérer la taille d'un type structure ou d'une variable structure (ex : sizeof (struct Personne) ou sizeof (p1))

Exemple:

```
 struct article art;
 printf("taille en octets de la structure article : %d\n", sizeof(art)) ;
/* ou */
printf("taille en octets de la structure article : %d\n", sizeof(struct
article)) ;
```

Structureset fonctions

Une structure peut être utilisée comme argument d'une fonction et transmise par valeur (ou par adresse via un pointeur)

Exemple de transmission par valeur

```
struct couple { int a;
                float   b;};
void zero (struct couple s)
   { s.a=0; s.b=0;
     printf(" %d %f \n ", s.a, s.b);
   }
main()
  { struct couple x;
    x.a=1;x.b=2.3;
    printf("avant: %d %f \n ", x.a, x.b);
    zero(x);
    printf("après: %d %f \n ", x.a, x.b);
  }
```

Exemple de transmission par adresse

```
struct couple { int a;
                float   b;};
void zero (struct couple *s)
   { s->a=0; s->b=0;
     printf(" %d %f \n ", s->a, s->b);
   }
main()
  { struct couple x;
    x.a=1;x.b=2.3;
    printf("avant: %d %f \n ", x.a, x.b);
    zero(&x);
    printf("après: %d %f \n ", x.a, x.b);
  }
```

Définition de types synonymes: typedef

- En C, on peut définir des types nouveaux synonymes de types existants (simples, pointeur, tableau, structure,…) en utilisant le mot clé **typedef**. Ces nouveaux types peuvent être utilisées ensuite comme les types prédéfinis

- **Exemple d'un type synonyme d'un type simple :**

- **typedef int entier;** définit un nouveau type appelé **entier** synonyme du type prédéfini **int**

- **entier i=4,T[10] ;** le type entier est utilisé ici pour déclarer des variables
- Remarque : l'intérêt de typedef pour les types de base est limité puisqu'il remplace simplement un nom par un autre

Définition de types synonymes: typedef

Exemple d'un type synonyme d'un type pointeur :

```
typedef int *ptr_ entier;
ptr_entier p1,p2;
```

p1 et p2 sont des pointeurs sur des int

```
typedef char *chaine;
chaine ch;
```

- **Type synonyme d'un type tableau :**

```
typedef float  tableau[10];
tableau T;
```

T est un tableau de 10 réels

```
typedef    int matrice[4][5];
matrice A;
```

A est une matrice d'entiers de 4 lignes et 5 colonnes

Exemples de typedef

- **Type synonyme d'un type structure :**

```
typedef struct
  { int jour;
    int mois;
    int annee;
  } date;
```

- date est le nom du nouveau type et non d'une variable
- On peut utiliser directement ce type, par exemple: **date d, *p;**
- Ceci permet simplement d'éviter l'emploi du mot clé struct dans les déclarations de variables

Structures récursives

- Une structure est dite récursive si elle contient des pointeurs vers elle même, par exemple :

```
struct Individu
{ char*Nom;
  int Age;
 struct Individu *Pere, *Mere;
};
```

```
struct noeud
{ int val;
 struct noeud *suivant;
};
```

- Ce genre de structures est fondamental en programmation car il permet implémenter la plupart des structures de données employées en informatique