Département Génie électrique
Filière : GTE
Semestre : S2
Année Universitaire 2019/2020

Cours Informatique 2:

Programmation en langage C

Pr. CHADLI

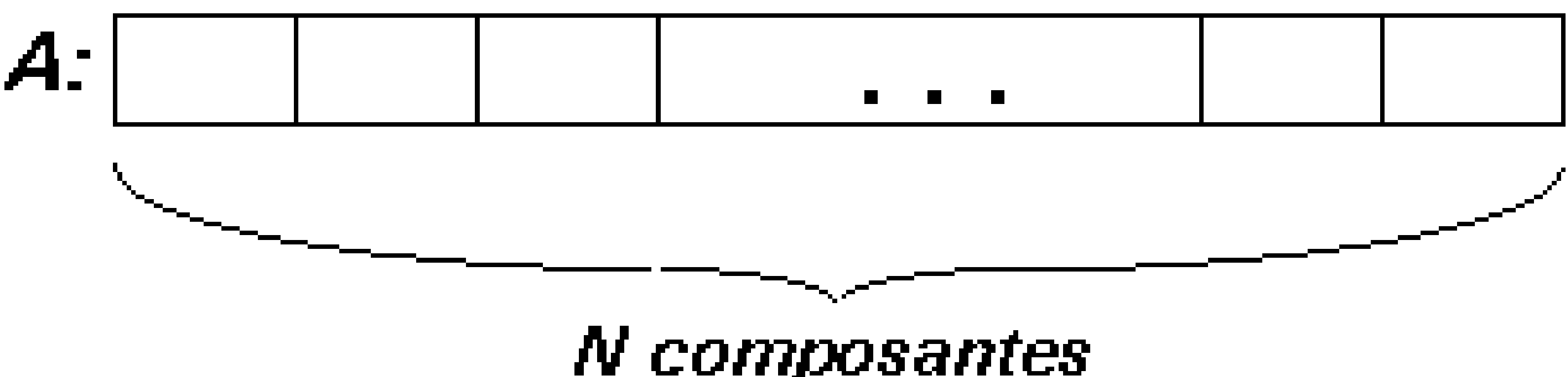
Sommaire

1. **Les Tableaux en Langage C**
2. **Les pointeurs en Langage C**
3. **Les fonctions en C**
4. **Les structures**

CHAPITRE 1: Les Tableaux en langage C

Un tableau est une variable structurée composée d'un nombre de variables simples de même appelés éléments du tableau

Ces éléments sont stockées en mémoires à des emplacements contigus (l'un après l'autre)



Tableaux à un seul indice

- **La syntaxe de la déclaration :**

Type nomtab[nbélément]

Déclaration directe :

```
int tab[4] ; /*tab est un tableau de 4 entiers:
tab[0], tab[1], ...,tab[3]
```

tab[0]	tab[1]	tab[2]	tab[3]

Déclaration structurée :

```
#define  SIZE     10
int         age[SIZE];                              /* age est un vecteur de 10 entiers */
```

Remarque: La déclaration d'un tableau permet de lui réserver un espace mémoire dont la taille (en octets) est égal à: dimension*taille du type.

Ainsi pour:

- **Short A[100] /* on réserve 100*2octets=200 octets**
- **Char mots[10] /* on réserve 10*1octets=10octets**

Tableaux à un seul indice

Exemple

Déclarer les tableaux age, taille, poids, sexe d'un groupe de 150 personnes ou moins.

```
#define  MAX_PERS  150
int    age[MAX_PERS];
float  taille[MAX_PERS], poids[MAX_PERS];
char   sexe[MAX_PERS];
int    nbPers ; /* le nombre effectif de personnes   traitées */
```

Accès aux composantes d'un tableau

L'accès à un éléments du tableau se fait au moyen de l'indice. Par exemple, T[i] donne la valeur de l'élément i du tableau T.

En langage C l'indice du premier élément est 0. L'indice du dernier élément est égal à la dimension-1.
Ex: int T[5]={10,8,7,6,5}

Remarque:
On ne peut pas saisir, afficher ou traiter un tableau en entier, ainsi on ne peut pas écrire printf(« %d,T) ou scanf(« %d »,&T)
On traite les tableaux éléments par élément de facon répétitive en utilisant des boucles

Accès aux composantes d'un tableau

Exemple

```
#include<stdio.h>
#include<stdlib.h>
main (){
int Note[5];
Note[0]=10;
Note[1]=15;
Note[2]=05;
Note[3]=18;
Note[4]=20;
printf ("le premier element=%d\n",Note[0]);
printf ("le second element=%d\n",Note[1]);
printf ("le troisieme element=%d\n",Note[2]);
printf ("le quatrieme element =%d\n",Note[3]);
printf ("le cinquieme element =%d\n",Note[4]);
system ("pause");
}
```

!!! Les tableaux consomment beaucoup de place mémoire. On a donc intérêt à les dimensionner au plus juste.

Représentation d'un tableau à une dimension en mémoire

Exemple:

```
main()
{
    const int N = 10;
    double x[N],y[N];
    x[0] = 10;
    x[1] = 20;
    y[0] = 33;
    y[1] = 43;
    ….
```

x	y
10	33
20	43

Tableaux: saisie et affichage

Remplissage: Saisie des éléments d’un tableau T d’entier de taille N:

```
for(i=0,i<N;i++)
{
        printf(" entrer l’élément %d: ",i);
        scanf(" %d ",&T[i]);
}
```

Remarque: En C, on peut déclarer et initialiser un tableau :

```
#define NB_NOTES 7
float bareme[NB_NOTES]={100.0, 100.0, 25.0, 35.0, 40.0, 100.0, 100.0 } ;
```

Affichage des éléments d’un tableau T d’entier de taille N:

```
for(i=0,i<N;i++)
{
        printf(" T[%d]=%d \n ",i,T[i]);
}
```

Exercice 1

1. Ecrire un programme c qui déclare et remplisse un tableau de 7 valeurs numériques en les mettant toutes à zéro.

```
#include<stdio.h>
main ( )
{
int Tab[7];
int i;
for (i=0; i <7; i++) Tab[i]=0;
}
```

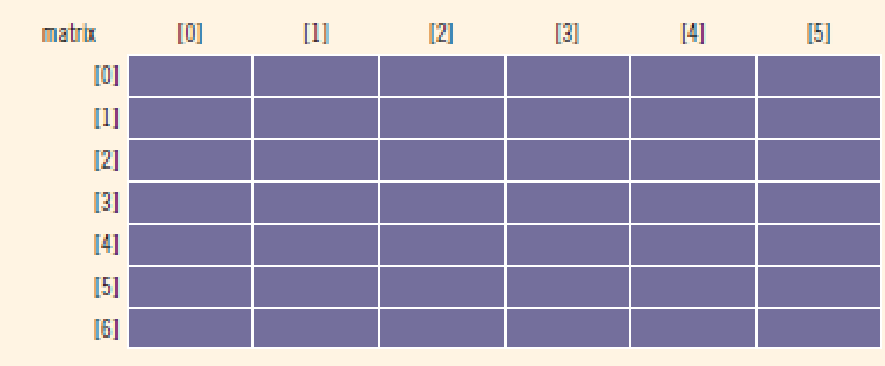
2. Ecrire un programme c qui déclare et remplisse un tableau contenant les six voyelles de l'alphabet latin.

```
#include<stdio.h>
main ( )
{
char Voy[6];
Voy[0]= 'a';
Voy[1]= 'e';
Voy[2]= 'i';
Voy[3]= 'o';
Voy[4]= 'u';
Voy[5]= 'y';
}
```

Tableaux à 2 dimensions: Matrices

Déclaration :

< Type > < NomTableau>[taille1][taille2] ;



- Exemple:

```
#define N 3
#define M 4
float Mat[N][M];
```

  - Si on considère un tableau à 2 dimensions comme une matrice, le premier indice représente les lignes et le second, le nombre de colonne.

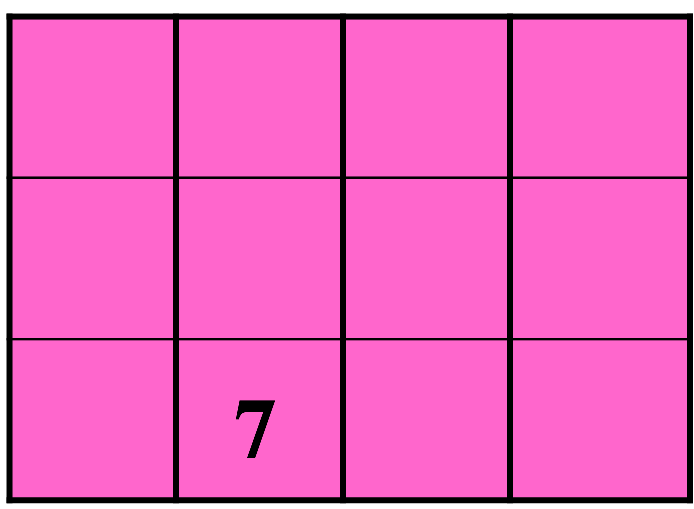
- Manipulation
  - Double boucles for
  - On accède à un élément du tableau en utilisant la syntaxe suivante :

Nomtableau[indice1][indice2]

Tableaux à 2 dimensions: Matrices

Exemple:

int Mat[3][4] ;



- L'instruction:
  - Mat[3][2]=7 ;Met la valeur 7 dans la case de la 3eme linge et de la 2eme colonne du Mat

Tableaux à 2 dimensions: saisie et affichage

Remplissage: Saisie des éléments d'un tableau T d'entier de taille NxM:

```
for(i=0,i<N;i++)
        {
for(j=0,j<N;j++)
            {
                printf(" entrer l'élément T[%d][%d]: ",i,j);
                scanf("%d",&T[i][j]);
            }
        }
```

Affichage des éléments d'un tableau T d'entier de taille NxM:

```
for(i=0,i<N;i++)
for(j=0,j<N;j++)
        printf(" T[%d][%d]=%d \n ",i,j,T[i][j]);
```

Exercice:

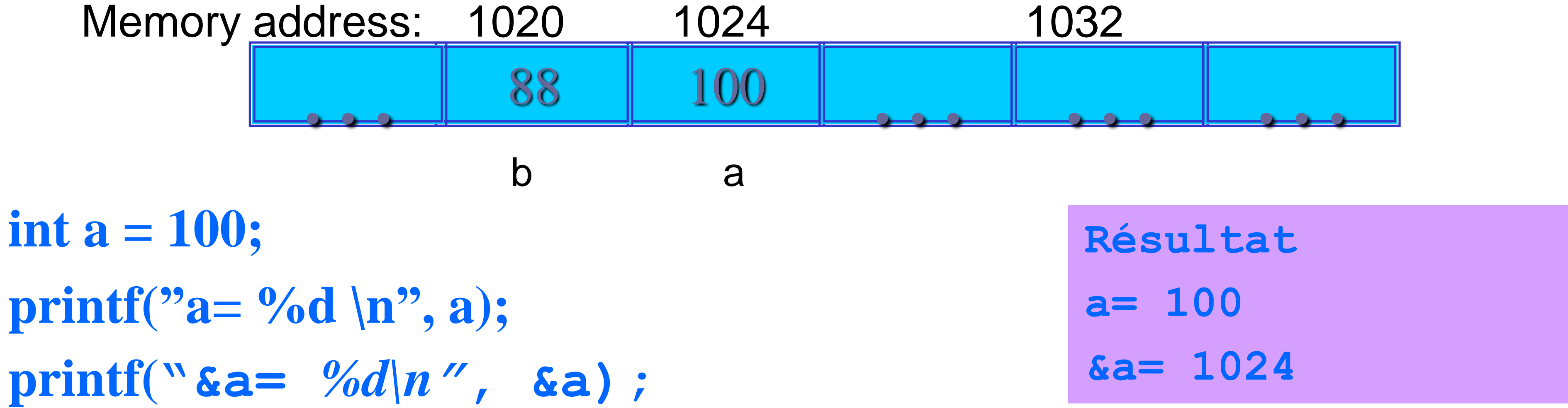
- Ecrire un programme qui lit la dimension N d'un tableau T du type **int** (dimension maximale: 50 composantes), **remplit** le tableau par des valeurs entrées au clavier et **affiche** le tableau.
- **Calculer** et afficher ensuite la somme des éléments du tableau.
- **Effacer** ensuite toutes les occurrences de la valeur 0 dans le tableau T et tasser les éléments restants. Afficher le tableau résultant.
- **Copiez** ensuite toutes les composantes strictement positives dans un deuxième tableau **TPOS** et toutes les valeurs strictement négatives dans un troisième tableau **TNEG**. Afficher les tableaux **TPOS** et **TNEG**.

CHAPITRE 2: Les pointeurs en Langage C

Operateur Adresse &

En C, on ajoute une caractéristique de plus à une variable :

- C'est son adresse (son emplacement en mémoire) déterminée par **l'opérateur &** (adresse de)
  - ex : `scanf("%d", &var);`
    - **scanf** a besoin de **l'adresse en mémoire** de la variable **var** pour y placer la valeur lue

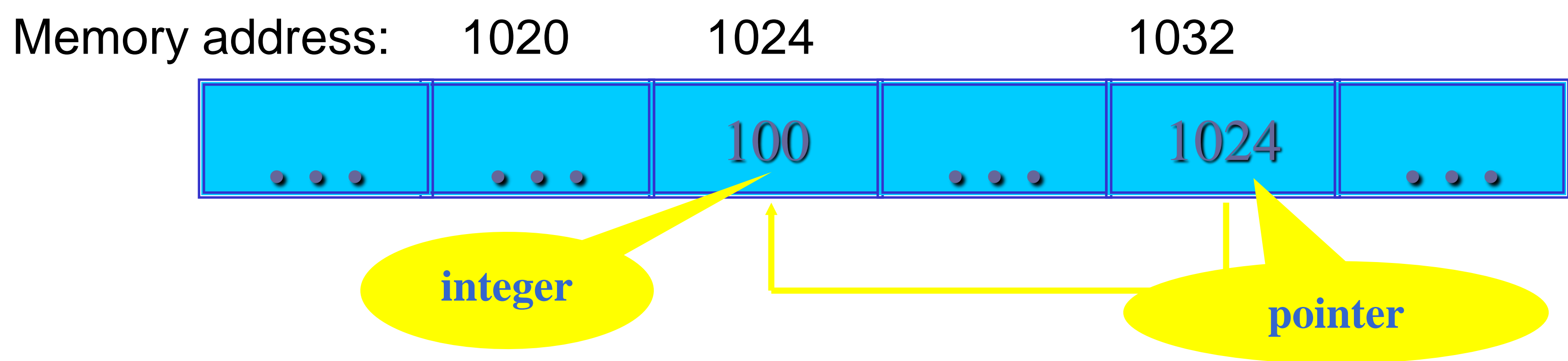


Pointeurs

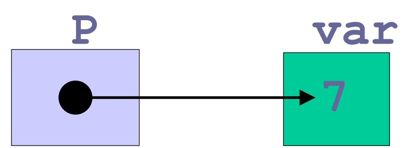
Un pointeur est une variable dont la valeur est une adresse

<u>**Déclaration**</u> : ***type * Nom_pointeur ;***

- * est l'opérateur qui indiquera au compilateur que c'est un pointeur
- Plusieurs pointeurs nécessitent l'utilisation d'un * avant chaque déclaration de variable: `int *P1, *P2;`
- Initialiser les pointeurs sur 0, NULL ou une adresse
- 0 ou NULL - ne pointe à rien (NULL préféré)

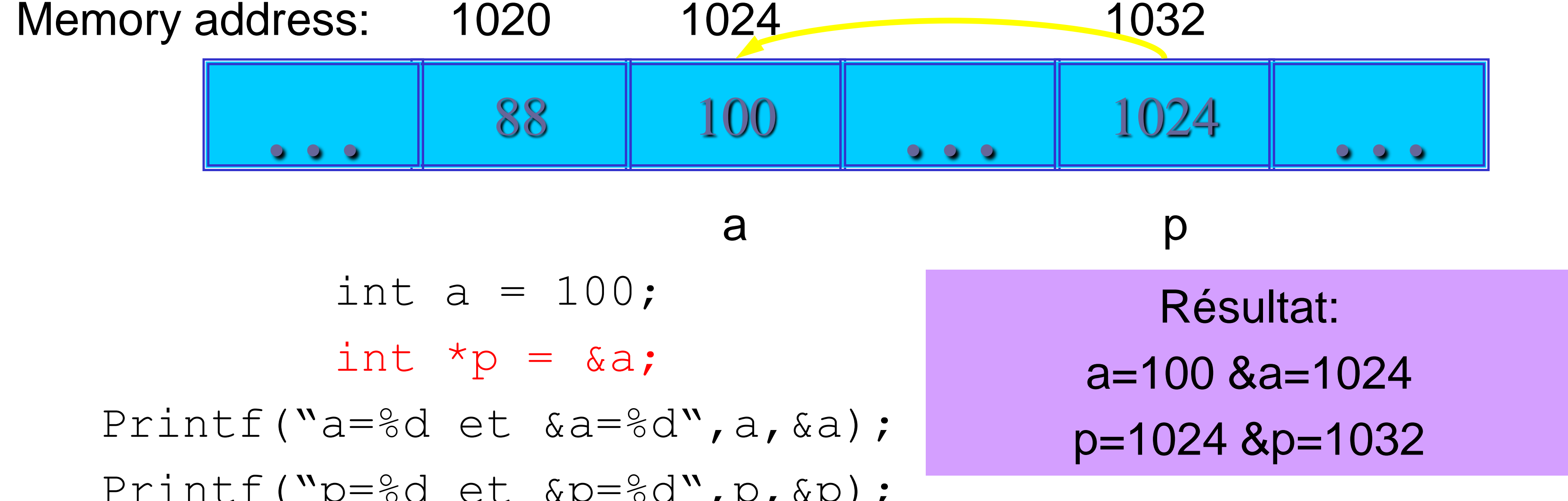



```
int *P;//P est un pointeur vers une variable int
int a=100;
P=&a;
```



Opérateurs unaires pour manipuler les pointeurs, & (adresse de) et * (contenu)

Exemple



La valeur d'un pointeur étant une adresse, l'opérateur * permet d'accéder à la valeur qui est à cette adresse

```
int a = 100;
int *p = &a;
Printf("a=%d et &a=%d",a,&a);
Printf("p=%d et &p=%d",p,&p);
Printf("*p=%d",*p);
```

Résultat:

a=100 &a=1024

p=1024 &p=1032

*p=100

Pointeurs

A ne pas confondre!!

- Déclarer un pointeur signifie seulement qu'il s'agit d'un pointeur**: int * p;**
- Ne pas confondre avec l'opérateur de déréférencement, qui est également écrit avec un astérisque ***.**
- Ce sont simplement deux tâches différentes représentées avec le même signe

```
int a = 100, b = 88, c = 8;
int *p1 = &a, *p2, *p3 = &c;
      p2 = &b;
      p2 = p1;
      b = *p3;
      *p2 = *p3;
      printf("a= %d  b=%d  c=%d",a,b,c);
```

Result is:

a=8 b= 8 c= 8

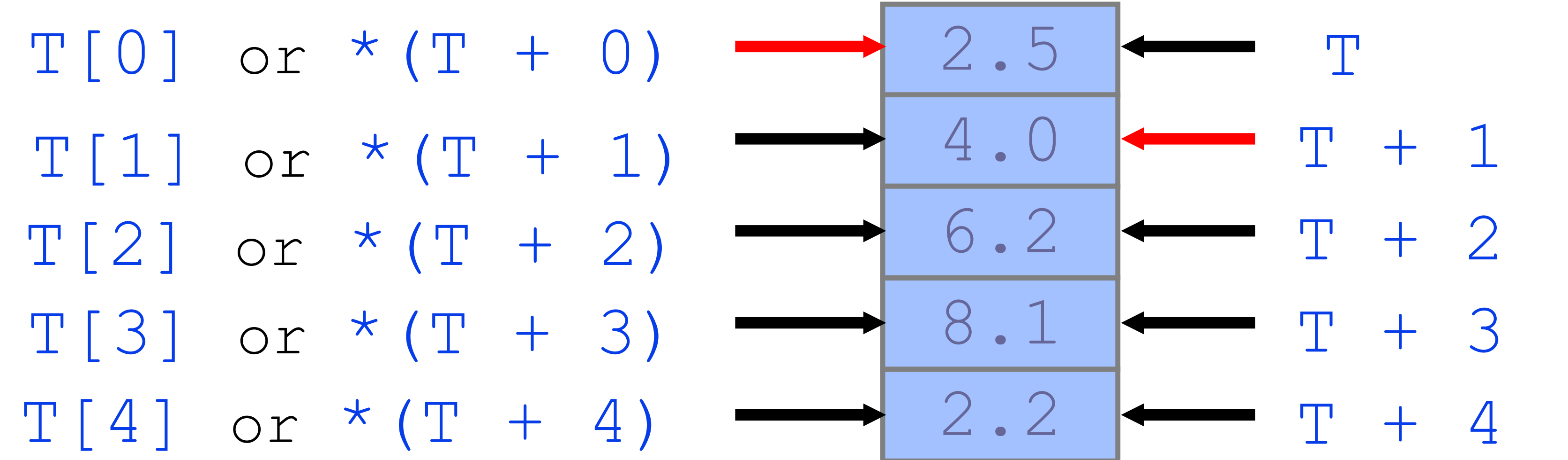
Tableaux et Pointeurs

Le nom d'un tableau n'est pas du tout une variable. C'est une constante de type pointeur : c'est l'adresse de son premier élément indice 0.

float T[10] ;

T est équivalent à **&T[0]**

Ainsi **T=** ..... ; est toujours invalide (on affecte à une variable, pas à une constante).



Ainsi, si T est le nom d'un tableau, on a :

T + i <====> &T[i]

*** (T + i) <====> T[i]**

Tableaux et Pointeurs

T[0] or *(T + 0)	2.5	T
T[1] or *(T + 1)	4.0	T + 1
T[2] or *(T + 2)	6.2	T + 2
T[3] or *(T + 3)	8.1	T + 3
T[4] or *(T + 4)	2.2	T + 4

Exemples : Écrire un programme qui affiche le contenu du tableau à l'écran:

- **<u>avec les indices</u> (plus simples à comprendre)**

```
for ( i = 0 ; i < 4 ; i++ )
    printf("T[%d]=  %f \n", i, T[i]) ;
```

- **<u>avec un pointeur</u> (un peu plus compliqué)**

```
for ( i = 0 ; i < 4 ; i++ )
    printf(" T[%d]= %f\n", i, *(T+i) )
```

Tableaux et Pointeurs

Soit P un pointeur qui "pointe" sur un tableau A:

```
int A[9] = {12, 23, 34, 45, 56, 67, 78, 89, 90};
int *P;
P = A;
```

Quelles valeurs ou adresses fournissent ces expressions:

a. ***P+2 -------------------- la valeur 14**
b. ***(P+2) ----------------- la valeur 34**
c. **&A[4]-3 --------------- l'adresse de la composante A[1]**
d. **A+3 ------------------ l'adresse de la composante A[3]**
e. **&A[7]-P ------------- la valeur (indice) 7**
f. **P+(*P-10) ----------l'adresse de la composante A[2]**

Chapitre 3: Les fonctions en C

- Un problème complexe est souvent plus facile à résoudre en le divisant en plusieurs parties plus petites, dont chacune peut être résolu par lui-même.

- C'est ce qu'on appelle **la programmation structurée**.

- Ces parties sont parfois transformées en **fonctions en C.**

- **main()** utilise ensuite ces fonctions pour résoudre le problème initial.

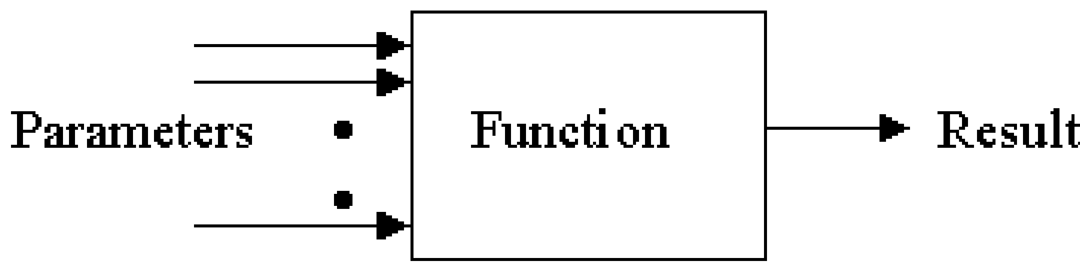
Fonctions C standard

- Langage C est livré avec un grand nombre de fonctions qui sont connus comme des fonctions standard
- Ces fonctions standard sont groupées dans différentes bibliothèques qui peuvent être inclus dans le programme C, par

- ***exemple:***
  Les fonctions mathématiques sont déclarés dans la bibliothèque <math.h>

Les fonctions en C

Fonction "naturelle" qui retourne un seul résultat avec return :



Syntaxe

TypeRésultatRetour nomFonction (paramètres)

{

- **déclarations locales si nécessaire**
- **calcule et retourne (avec instruction return) le résultat calculé**

}

Les fonctions en C: Arguments/Paramètres

correspondance un à un entre les arguments d'un appel de fonction et les paramètres de la définition de la fonction.

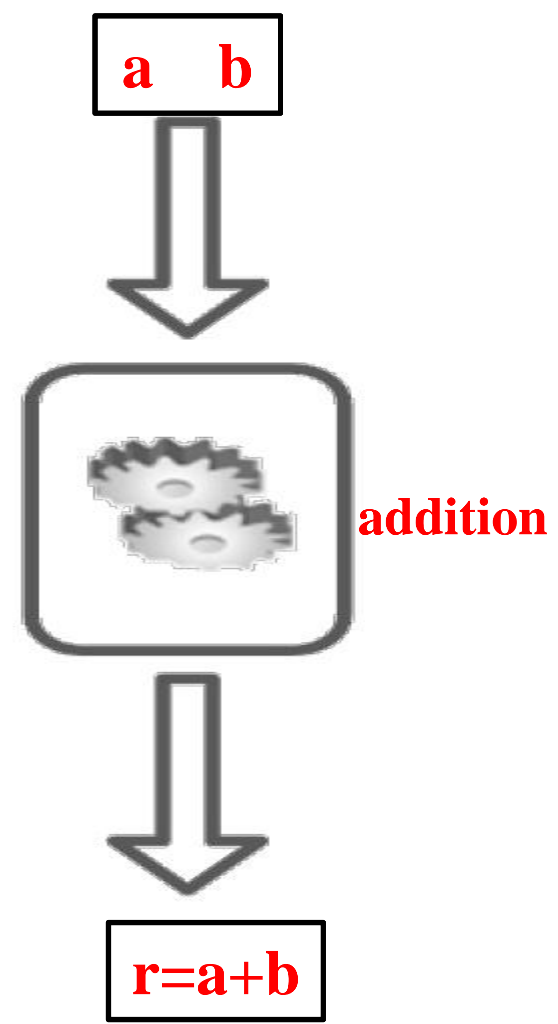
Parameters • • Function Result

```
int argument1;
double argument2;
// Applel de la fonction
result = NomFonction(argument1, argument2);
.
.
// definition de la function
int NomFonction(intparameter1,double parameter2){
// La fonction utilise les deux paramètres
//   parameter1   =   argument   1,   parameter2   =
  argument2
```

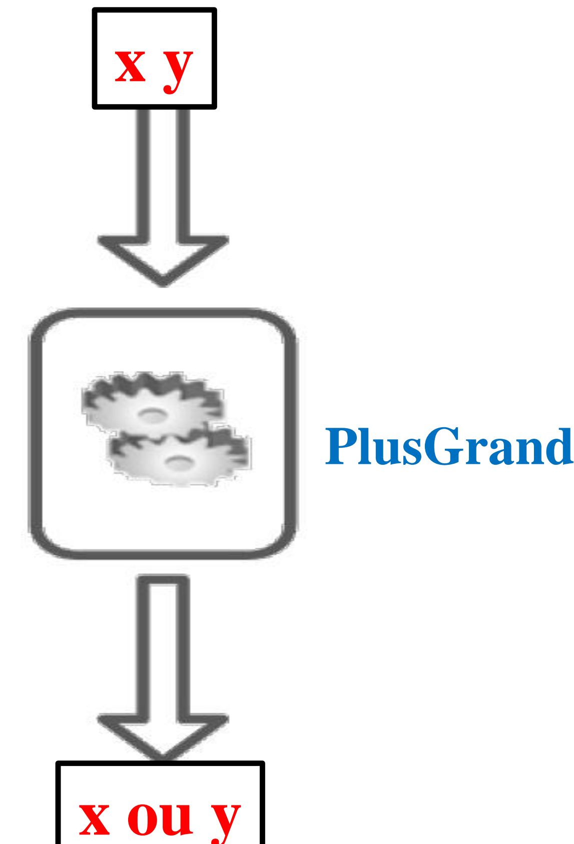
Les fonctions en C

Exemple 1

```
#include<stdio.h>
int addition (int a, int b)
                    {  int r;
                     r=a+b;
                    return (r);
                    }
main()
{  int z;
z =addition (5,3);
 printf("5+3=%d",z);
}
```



Les fonctions en C



Exemple 2

```
float  PlusGrand ( float x, float y)
{
if ( x > y )
 return x ;
else
 return y ;
}
```

Remarques:

1. L'instruction "return" provoque la fin de la fonction
2. Sur l'en-tête, on ne peut pas grouper les paramètres de même type Ne pas terminer l'en-tête par le point virgule

Les fonctions en C

Exercice: Ecrire une fonction appelée conversion qui convertit les euros en. dirhams sachant que : **1 euro= 10.85 DH**

```
#include<stdio.h>
double conversion(double euro) {
double dh;
dh=10.85*euro;
return dh;
}
```

```
//Programme test
main(){
printf("10  euros = %lf  DH\n", conversion(10));
printf("50  euros = %lf  DH\n", conversion(50));
printf("200 euros = %lf  DH\n" , conversion(200));
}
```

Les fonctions en C: Prototype de fonction

- Le prototype de la fonction déclare les paramètres d'entrée et de sortie de la fonction.
- **Syntaxe**

<type> <nom de la fonction> (<type list>);

Exemples:

- **Prototype de la fonction conversion**

  double conversion(double);

- **Prototype de la fonction PlusGrand**

  float PlusGrand (float, float)

- **Prototype de la fonction addition**

  ***int* addition (*int* , *int*);**

Les fonctions en C: Définition de fonction

- La définition de la fonction peut être placée n'importe où dans le programme **après** les prototypes de fonction.
- Si une définition de fonction est placée **avant** main (), il n'est pas nécessaire d'inclure son prototype de fonction.

```
#include<stdio.h>
double conversion(double);  // Prototype de la fonction conversion
main(){
printf("10  euros = %lf  DH\n", conversion(10));
printf("50  euros = %lf  DH\n", conversion(50));
printf("200 euros = %lf  DH\n" , conversion(200));
}
double conversion(double euro)  // définition de la fonct conversion
{
double dh; dh=10.85*euro;
return dh;}
```

Les fonctions en C

Fonction de type void (pas de return)

On utilise le type *void*, ce qui signifie « néant » en anglais. il n'y a vraiment rien qui soit renvoyé par la fonction.

La syntaxe : void nom_fonction (paramètres)

```
{
déclarations locales si nécessaire
réaliser l'action confiée
s'il y a des résultats de retour, ce sont des paramètres
transmis par pointeurs
}
```

Exemple

```
void direBonjour()
{
printf( "Bonjour !\n“);
//Comme rien ne ressort, il n'y a pas de return !
}
```

Fonction de type void (pas de return)

Arguments transmis par pointeur :

L'en-tête de la fonction:

void nomFunc (..... , type_résultat * P, ...)

Appel :

nomFunc (..... , &variable du type_résultat, ...) ;

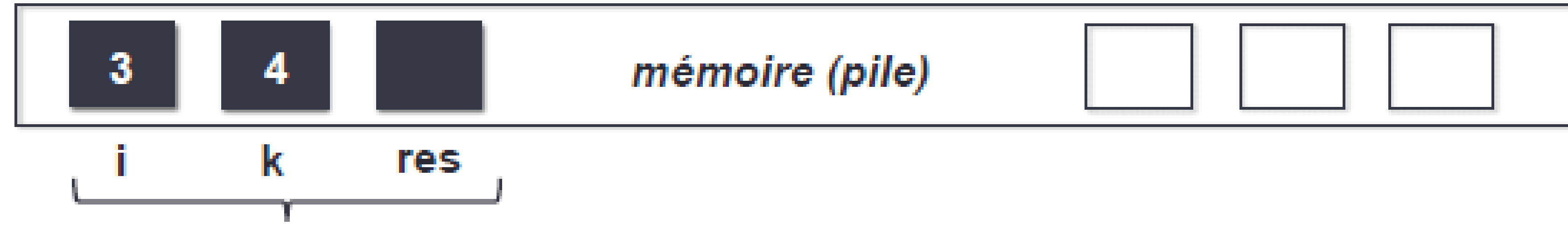
Sur l'en-tête c'est un pointeur qui pointe vers le type du résultat calculé. A l'appel c'est une adresse de la variable qui reçoit le résultat.

Passage de paramètres par valeur

- les arguments/paramètres formels d'une fonction sont initialisés avec une copie de la valeur des arguments effectifs.

```
main()
{  int i, k;
float res;
i=3; k=4;
res =norme(i,k);
 printf("%f",res);
}
```

```
float norme(int i, int j
{  int result;
i=i*i;
j=j*j;
result=sqrt(i+j);
return (result);
                    }
```

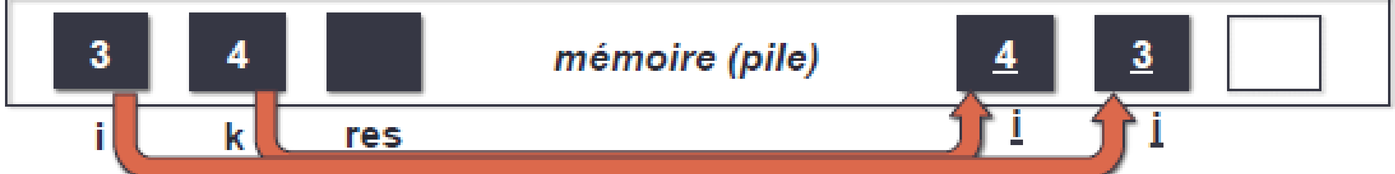


Passage de paramètres par valeur

- les arguments/paramètres formels d'une fonction sont initialisés avec une copie de la valeur des arguments effectifs.

```
main()
{  int i, k;
float res;
i=3; k=4;
res =norme(i,k);
 printf("%f",res);
}
```

```
float norme(int j, int i)
{  int result;
i=i*i;
j=j*j;
result=sqrt(i+j);
return (result);
                    }
```



1. **Appel à la fonction**:

   - création des variables **locales pour arguments**

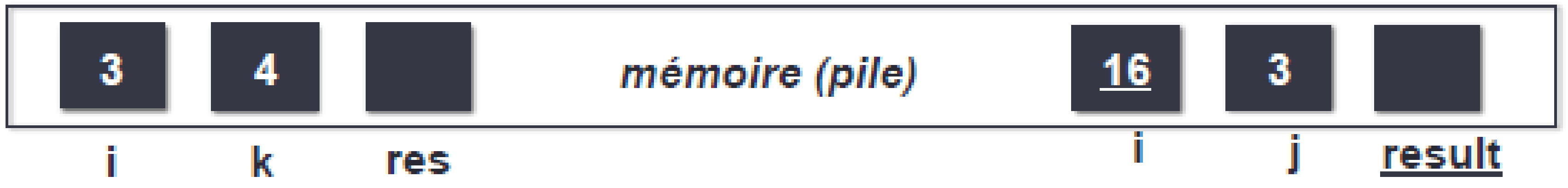
   - copie des valeurs

Passage de paramètres par valeur

```
main()
{  int i, k;
float res;
i=3; k=4;
res =norme(i,k);
 printf("%f",res);
                }
```

2 →

```
float norme(int i, int j
{  int result;
i=i*i;
j=j*j;
result=sqrt(i+j);
return (result);
                }
```



2. Exécution de la fonction:

- création des variables locales
- exécution des instructions ici(i = i*i;)

Passage de paramètres par valeur

```
main()
{  int i, k, res;
i=3; k=4;
res =addition (i,k);
 printf("%d",res);
}
```

3

```
int norme(int i, int j
{  int result;
i=i*i;
j=j*j;
result=sqrt(i+j);
return (result);
                 }
```

3	4	5.0	mémoire (pile)	16	9	5.0
i	k	res		i	j	result

2. Retour par valeur:

- copie de la valeur retournée vers la fonction appelante

Passage de paramètres par valeur

```
main()
{  int i, k, res;
i=3; k=4;
res =addition (i,k);
 printf("%d",res);
}
```

3

```
int norme(int i, int j
{  int result;
i=i*i;
j=j*j;
result=sqrt(i+j);
return (result);
                    }
```

Inconvénient du passage par valeur!!!!

une fonction ne peut modifier les paramètres effectifs par passage par valeur

Passage de paramètres par adresse

Solution pour modifier les arguments effectifs avec une fonction!!

- passer par **valeur** l'adresse de la variable
  - la fonction initialise des pointeurs avec la copie de l'adresse
  - et accède directement en mémoire aux paramètres effectifs par indirection sur les pointeurs

- **On parle de passage de paramètres par adresse**.

Passage de paramètres par adresse

Exemples d'illustration

```
void echanger( int* i, int* j){
    int tmp;
        tmp=*i;
        *i=*j;
        *j=tmp;  }
 main(){
 int x=1, y=2;
  printf("Avant echange, x = %d, y = %d\n", x, y);
 echanger (&x, &y ) ;
 printf("Apres echange, x = %d, y = %d\n", x, y);
}
```

```
Avant echange,x=1,y=2
Apres echange, x = 2, y = 1
```

Passage de paramètres par adresse

```
void echanger( int* i, int* j)
 {      int tmp;
        tmp=*i;
        *i=*j;
        *j=tmp;  }
 main()
{int x=1, y=2;
 printf("Avant, x = %d, y = %d\n", x, y);
 echanger (&x, &y ) ;
 printf("Après, x = %d, y = %d\n", x, y);
}
```

1 2 mémoire (pile) &x &y
x y i j

1 2 mémoire (pile) &x &y 1
x y *i i j tmp

2 1 mémoire (pile) &x &y 1
x y *i i *j j tmp

En Résumé: type function (type *i)

paramètre effectif **modifiable** par la fonction en utilisant indirection (*) sur les pointeurs

Passage de paramètres par adresse

Exemple 2:

```
Void pluspetit ( float a, float b, float * m )
 {
    if ( a < b ) *m = a ;
    else         *m = b ;
 }
```

Utilisation :

```
float x, y , min;
printf("Entrez 2 réels ");
scanf(« %f%f»,&x, &y);
pluspetit(x, y, &min);
printf("La plus petite valeur est %8.2f\n", min);
```

Exercice

Écrire une fonction qui reçoit deux réels a et b comme paramètres d'entrée. Elle calcule et retourne (par pointeurs) les résultats suivants :

- **la plus grande valeur parmi a et b**
- **la plus petite valeur parmi a et b**
- **la différence positive entre a et b (absolue de (a-b)).**

Choix de type de fonction(return vs void)

La fonction a t-elle une seule tâche de calculer (déterminer, compter, …) un seul résultat de type simple ?:

- **Si oui => choisir une fonction avec return**
- **Autres cas : choisir une fonction de type void**

Passage de tableaux à une fonction

- **Il va falloir envoyer 2 informations à la fonction : le tableau (enfin, l'adresse du tableau) et aussi et surtout sa taille !**
- **Un tableau peut être modifié dans une fonction. Il est passé par adresse et non par valeur.**
- **Ils ne peuvent pas être retournés comme résultat d'une fonction.**

Passage de tableaux à une fonction

```
// Prototype de la fonction d'affichage
void affiche(float *A, int n); // ou  void affiche(float*, int);
main()
{ flaot Tab[4] = {10.0, 15.0, 3.0}, i = 0;
// On affiche le contenu du tableau
affiche(Tab, 4);
}
void affiche(float *A, int n)
{ int i;
for (i = 0 ; i < n ; i++) printf("%f\n", A[i]);
}
```

Passage de tableaux à une fonction

Important : il existe une autre façon d'indiquer que la fonction reçoit un tableau. Plutôt que d'indiquer que la fonction attend un flaot* tab

```
// Prototype de la fonction d'affichage
void affiche(float A[], int n);
main()
{ float Tab[4] = {10.0, 15.0, 3.0}, i = 0;
// On affiche le contenu du tableau
affiche(Tab, 4);
}
void affiche(float A[], int n)
{ int i;
for (i = 0 ; i < n ; i++) printf("%f\n", A[i]);
}
```

Exercice

créer une fonction ***sommeTab qui renvoie la somme des valeurs contenues dans le tableau*** (***utilisez un return*** pour renvoyer la valeur).

Pour vous aider, voici le prototype de la fonction à créer :

Code : C

```
Int sommeTab(int T[], int taille);
```

Chapitre 4: Les structures

Concepts

- **Une structure est une collection de plusieurs variables (champs) groupées ensemble pour un traitement commode**
- **Les variables d'une structure sont appelées membres et peuvent être de n'importe quel type, par exemple des tableaux, des pointeurs ou d'autres structures**

```
struct complexe
{
    int img
    int      re;
    char var;
};
```

Chapitre 4: Les structures

Concepts

Les étapes sont:

- **déclarer le type de la structure**
- **utiliser ce type pour créer autant d’instances que désirées**
- **Accéder les membres des instances**

```
struct complexe
{
    int img
    int     re;
    char var;
};
```

Déclarer les structures

- Les structures sont définies en utilisant le mot-clé **struct**

```
struct Date
{
    int  jour;
    int  mois;
    int  an;
};
```

```
struct etudiant
{
    char              nom[50];
    int               N;
    struct   Date     emprunt;
    struct   Date     creation;
};
```

```
struct Livre
{
    char   titre[100];
    char   auteur[50];
    float  prix;
};
```

```
struct Pret
{
    struct   Livre     b;
    struct   Date      due;
    struct   personne*who
};
```

!

Déclarer les structures

- déclarer en utilisant "**typedef**" :

```
typdef struct
{
    char         nom[80];
    int          numero;
    struct Date emprunt;
    struct Date creation;
} personne; /* personne est le nom du type */
 // déclaration de variable type personne
personne pers1, pers2 ;
```

Déclarer des instances: *Définition d'une variable structurée*

- Une fois la structure définie, les instances peuvent être déclarées
- Par abus de langage, on appellera structure une instance de structure

```
struct Date
{
    int   jour;
    int   mois;
    int   an;
} hier, demain;

struct Date paques;
struct Date semaine[7];

struct Date nouvel_an = { 1, 1,
2001 };
```

- Déclaration
  - avant ‘;’ .
- Initialisation .

Accéder aux membres d'une structure

- Les membres sont accédés par le nom de l'instance, suivi de . , suivi du nom du membre

struct personne P;

```
printf("nom = %s\n", P.nom);
printf(”numéro de membre = %d\n", P.numero);

printf("\nDate d'emprunt %d/%d/%d\n", P.emprunt.jour,
                P.emprunt.mois, P.emprunt.an);
```

Remarques:

- **L'opération d'affectation = peut se faire avec des structures**
- **Tous les membres de la structure sont copiés (aussi les tableaux et les sous-structures)**

53

Quand la structure est un pointeur

Avec la déclaration : Personne * P ;

P est un pointeur vers le type Personne.

***P est une variable de type Personne.**

On peut accéder à n'importe quel champ de *P :

(*P).numero, etc ...

Le C permet de simplifier l'écriture en utilisant l'opérateur ->

(*P).champ <======> P->champ

54

Exemple :

Écrire une fonction permettant d'échanger les informations de deux personnes (de type Personne) :

Solution :

```
void echanger ( Personne * P1, Personne * P2)
{
    Personne Temporaire ;
    temporaire = *P1 ;
    *P1        = *P2 ;
    *P2        = temporaire ;
}
```

Passer des structures comme paramètres de fonction

- Une structure peut être passée, comme une autre variable, par valeur ou par adresse
- Passer par valeur n'est pas toujours efficace (recopiage à l'entrée)
- Passer par adresse ne nécessite pas de recopiage

Retour de structures dans une fonction

- **Par valeur (recopiage)**

```
struct Complex add(struct Complex a, struct Complex b)
{
    struct Complex  result = a;

    result.real_part    += b.real_part;
    result.imag_part  += b.imag_part;

    return result;
}
```

```
struct Complex   c1 = { 1.0, 1.1 };
struct Complex   c2 = { 2.0, 2.1 };
struct Complex   c3;

c3 = add(c1, c2);  /* c3 = c1 + c2 */
```

Exercice

Pour représenter un nombre complexe, créer une structure complexe qui contient les champs « partie réelle a » et la « partie imaginaire b ».

Ecrire un programme qui :

Saisit deux complexes c1 et c2

Affiche c1, c2

Solution

```
#include<stdio.h>
struct complexe{
        float a;
        float b;
};
main()
{ struct complexe c1,c2;
printf("entrer partie reelle de c1:");scanf("%f",&c1.a);
printf("entrer partie imaginaire de c1:");scanf("%f",&c1.b);
printf("entrer partie reelle de c2:");scanf("%f",&c2.a);
printf("entrer partie imaginaire de c2:");scanf("%f",&c2.b);

printf("c1=%f + %f i \n c2=%f +%f i  \n",c1.a,c1.b,c2.a,c2.b);
}
```