C

Ch 4 –Structures de données composées : tableaux et structures

I. Tableaux

A. Tableau en C

Un **TABLEAU** (ou **VECTEUR)** est une juxtaposition, sous un **NOM UNIQUE**, d'un certain nombre de **VARIABLES DE MEME TYPE**, auxquelles on accède individuellement grâce à un numéro d'ordre, ou rang, qu'on appelle **INDICE.**

En **C**[1], les **INDICES** sont numérotés **A PARTIR DE 0**.

ATTENTION : LE CONTROLE DES INDICES EST A LA CHARGE DU DEVELOPPEUR (LE COMPILATEUR NE VERIFIE PAS LE DEPASSEMENT DES LIMITES)

[1] C'est le cas dans la majorité des langages de programmation

B. Déclaration d'un tableau non initialisé en C (1)

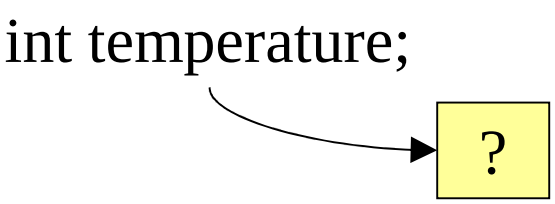
Syntaxe 1 : tableau à 1 dimension, non initialisé

```
type_de_donnee nom_tableau[nombre_d_elements];
```

- **type_de_donnee**
  - le type de donnée des éléments du tableau
- **nom_tableau**
  - identificateur du tableau
- **nombre_d_elements**:
  - nombre d'éléments du tableau
  - Attention : l'indice d'accès aux éléments sera compris entre **0** et (**nbre_elements – 1)**

Exemple d'une variable entière :

```
int temperature ; // permet la gestion d'une temperature
```



Exemple : déclaration d'un tableau de 12 entiers pour gérer les températures d'une année

```
int temperature[12];
```

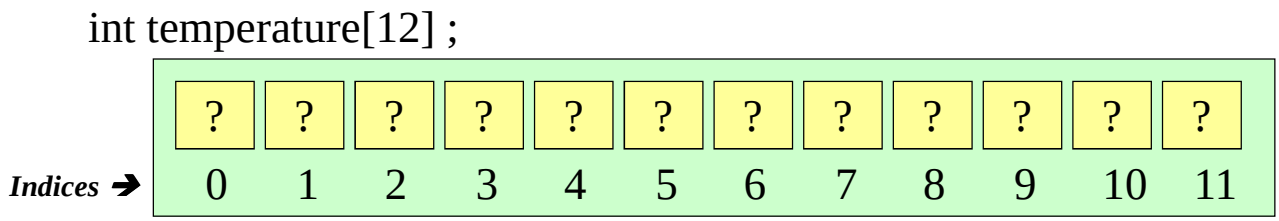


Figure 1 : un tableau de 12 entiers numérotés de 0 à 11

Exemple : initialisation du tableau à 0

```
int i ;
for (i=0 ;i <= 11 ;i++)
{
    temperature[i]=0 ;
}
// ou bien :
for (i = 0 ;i < 12 ;i++)
{
    temperature[i]=0 ;
}
```

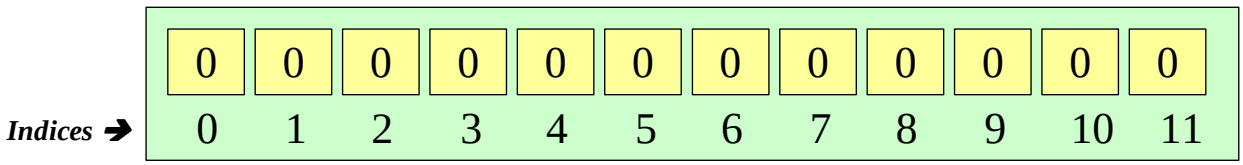


Figure 2 : contenu après initialisation

Exemple : initialisation à partir d'une saisie

```
int i ;
for (i=0 ;i<=11 ;i++)
{
    printf( "temperature du mois %d : " , i+1);
    scanf("%d", & temperature[i] );
}
```

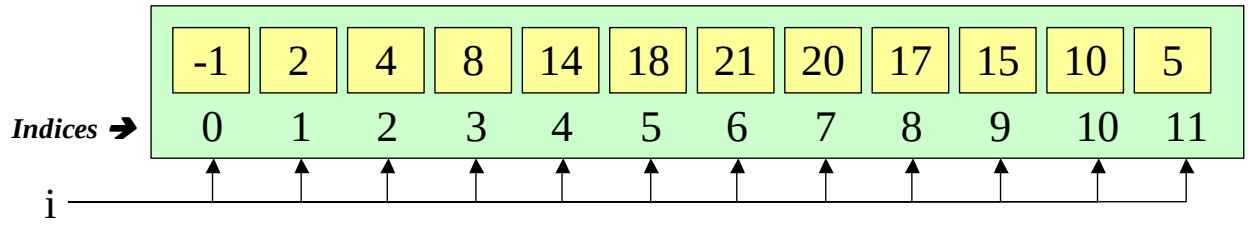


Figure 3 : 'i' va prendre successivement les valeurs 0, 1, 2, ...,10, 11

Exemple : parcourir le tableau et afficher toutes les valeurs

```
// parcours de 0 à 11
for (i=0 ;i<=11 ;i++)
{
    printf("%d", temperature[i]);
}
```

```
// ou bien en ordre inverse :
for (i=11 ;i>=0 ;i--)
{
    printf("%d", temperature[i]);
}
```

C. Déclaration d'un tableau initialise (2)

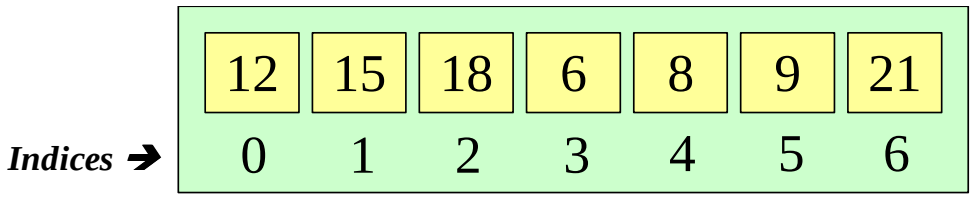
Syntaxe 2 : tableau à 1 dimension, initialisé

```
type_donnees nom_tableau [] = {val1, val2, . . . valN} ;
```

- **type_donnees**
  - Correspond au type de donnée des éléments du tableau
- **val1, val2, valN**:
  - valeurs d'initialisation du tableau
  - l'**indice** est alors compris entre **0** et le nombre de valeurs entrées **– 1**
- **si un nombre d'éléments est fourni, le nombre de valeurs en doit pas être supérieur au nombre d'éléments**

Exemple : déclaration d'un tableau d'entiers (7 éléments)

```
int temperature[] = {12,15,18,6,8,9,21};
```



D. Taille d'un tableau et fonction 'sizeof'

La fonction C **sizeof** attend le nom d'un type de donnée et renvoie le nombre d'octets occupés par une variable de ce type.

Exemple :

```
printf("Tester la fonction 'sizeof' :");
printf("\n\tType 'char' : %d octet(s)",sizeof(char));
printf("\n\tType 'short' : %d octet(s)",sizeof(short));
printf("\n\tType 'int' : %d octet(s)",sizeof(int));
printf("\n\tType 'long' : %d octet(s)",sizeof(long));
printf("\n\tType 'float' : %d octet(s)",sizeof(float));
printf("\n\tType 'double' : %d octet(s)",sizeof(double));
```

Le résultat obtenu est le suivant :

```
Tester la fonction 'sizeof' :
        Type 'char' : 1 octet(s)
        Type 'short' : 2 octet(s)
        Type 'int' : 4 octet(s)
        Type 'long' : 4 octet(s)
        Type 'float' : 4 octet(s)
        Type 'double' : 8 octet(s)
Process returned 0 (0x0)   execution time : 2.652 s
Press any key to continue.
```

L'application de la fonction 'sizeof' à un tableau (ou à tout autre variable) renvoie la longueur du tableau :

Exemple :

```
char tabChar[10];
short tabShort[10];
int tabInt[10];
long tabLong[10];
float tabFloat[10];
double tabDouble[10];
printf("Tester la fonction 'sizeof' sur un tableau ...");
printf("\n\t... de 10 'char' : %d octets",sizeof(tabChar));
printf("\n\t... de 10 'short' : %d octets",sizeof(tabShort));
printf("\n\t... de 10 'int' : %d octets",sizeof(tabInt));
printf("\n\t... de 10 'long' : %d octets",sizeof(tabLong));
printf("\n\t... de 10 'float' : %d octets",sizeof(tabFloat));
printf("\n\t... de 10 'double' : %d octets",sizeof(tabDouble));
```

Le résultat obtenu est le suivant :

```
Tester la fonction 'sizeof' sur un tableau ...
        ... de 10 'char' : 10 octets
        ... de 10 'short' : 20 octets
        ... de 10 'int' : 40 octets
        ... de 10 'long' : 40 octets
        ... de 10 'float' : 40 octets
        ... de 10 'double' : 80 octets
Process returned 0 (0x0)   execution time : 1.997 s
Press any key to continue.
```

Pour connaître le nombre d'éléments du tableau, il suffit donc de diviser la taille du tableau par la taille d'un élément (fonction 'sizeof' appliquée au type de l'élément).

E. Dimensions, tableaux multi dimensionnels

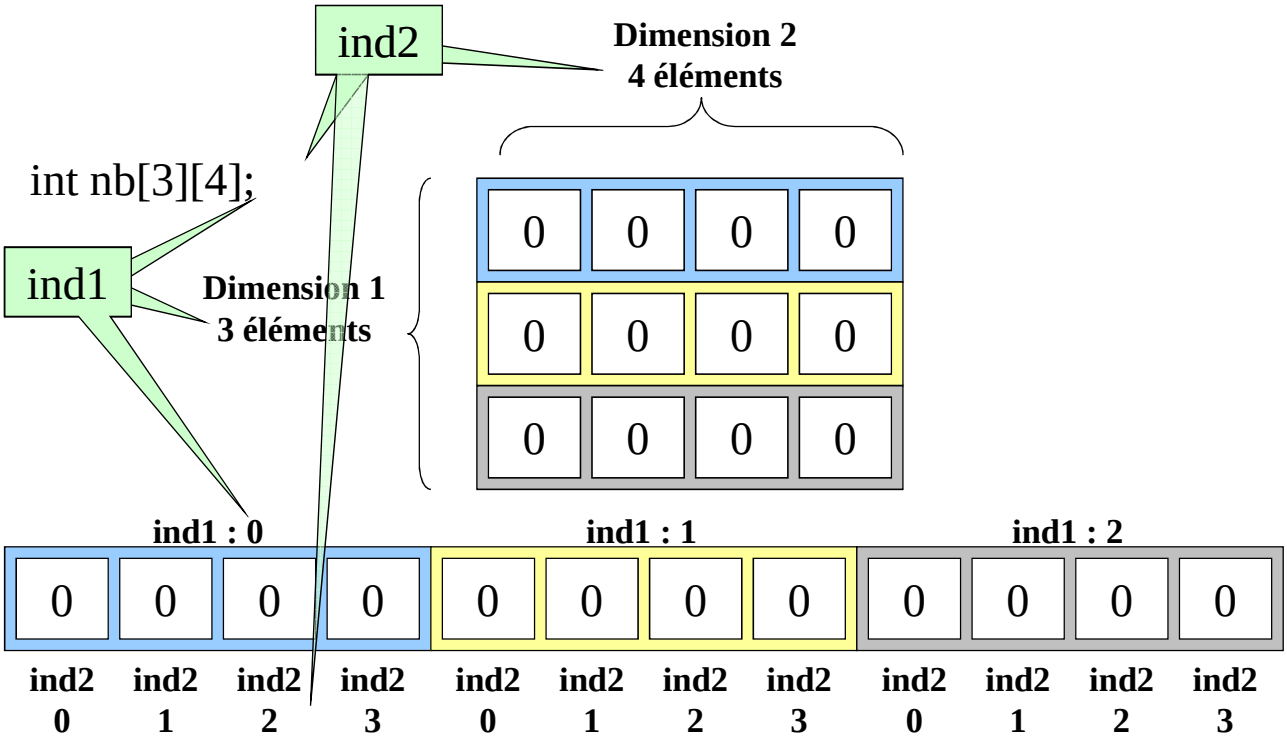
Syntaxe tableau à N dimensions :

```
type_de_donnees nom_tableau[elem1][elem2] . . .[elemN];
```

- **nom_tableau**
  - correspond au nom (identificateur) donné au tableau (=aux variables qui le composent)
- **elem1, elem2, elemN** : nombre d'éléments de chacune des dimensions
- **type_de_données**
  - le type de donnée des éléments du tableau

Exemple :

```
int nb[3][4] ;
```

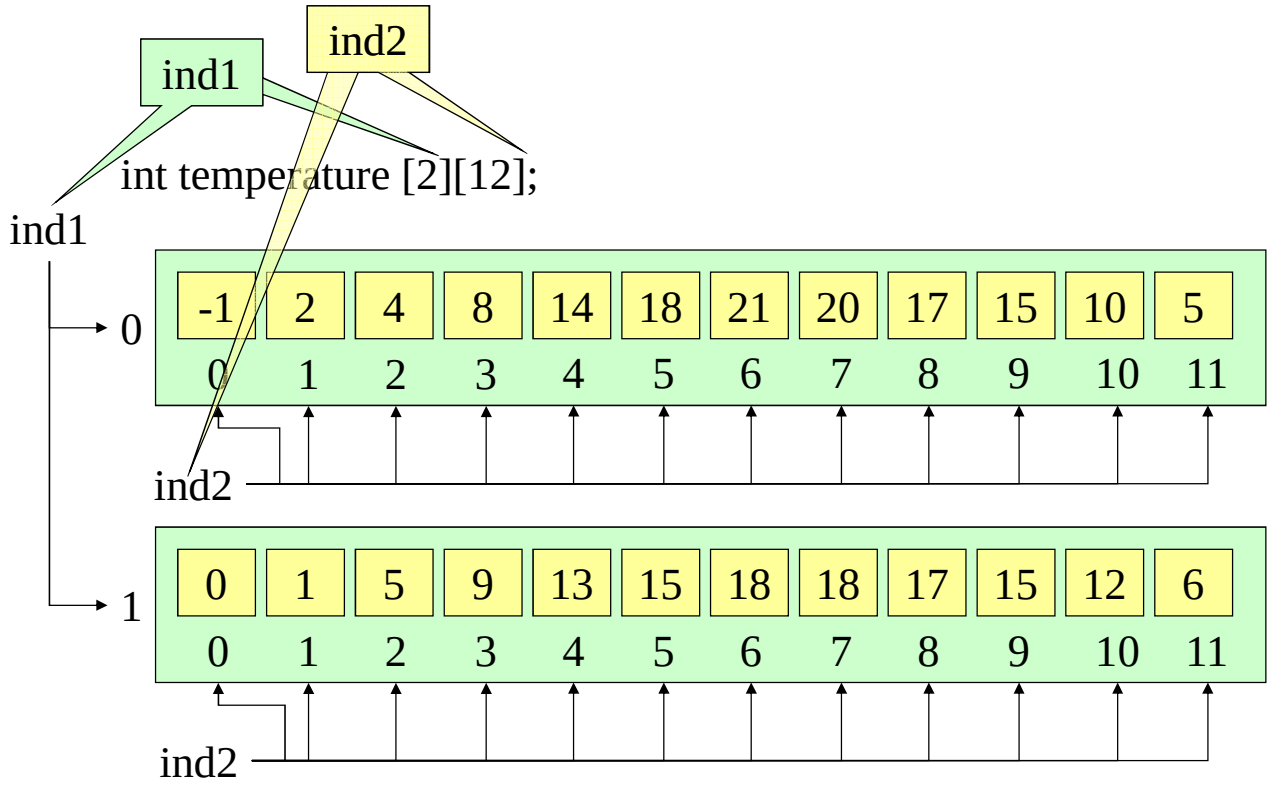


Exemple : tableau de températures de 2 ans, 12 mois par an

```
int temperature[2][12] ; // sur 2 ans, temp. des 12 mois

    // initialiser les 12 temperatures d'indice 0 %
    temperature[0][0] ← -1
    temperature[0][1] ← 2
    . . . etc. . . .
    temperature[0][11] ← 5

    // initialiser les 12 temperatures d'indice 1
    temperature[1][0] ← 0
    . . . etc. . . .
    temperature[1][10] ← 12
    temperature[1][11] ← 6
. . .
```



Exemple : initialiser les 2 dimensions du tableau

```
    // initialiser le tableau sur 2 ans %
    for (i=0 ;i<=1 ; i++) // boucle sur les années
       for (j=0;j<=11;j++) // boucle sur les temp. de chaque année
           temperature[i][j] = 0;
```

F. Initialisation des elements d'un tableau

1. Initialisation classique : initialiser chaque élément

Exemple :

```
    const int NB_ELEM = 1000; // ou #define NB_ELEM 1000
    const int VALEUR_INITIALE = 0; // ou #define VALEUR_INITIALE 0
    int tableau[NB_ELEM] ;
    int i ;
    for (i=0 ;i<NB_ELEM ; i++)
    {
       tableau[i] = VALEUR_INITIALE ;
    }
```

2. Initialisation à la declaration (1)

Exemple :

```
    #define NB_ELEM 1000 // const int NB_ELEM = 10; non accepté
    const int VALEUR_INITIALE = 0; // ou #define VALEUR_INITIALE 0
    int tableau[NB_ELEM] = { VALEUR_INITIALE };
```

3. Initialisation à la declaration (2)

Exemple : (limité aux tableaux ayant un nombre faible d'élements…)

```
    #define NB_ELEM 5 // const int NB_ELEM = 5; non accepté
    int tableau[NB_ELEM] = { 1,3,5,7,9 };
```

4. Initialisation à la declaration (3)

Exemple : (limité aux tableaux ayant un nombre faible d'élements…)

```
    #define NB_ELEM 5 // const int NB_ELEM = 5; non accepté
    int tableau[NB_ELEM] = { 1,3 };
```

Les 2 premiers éléments sont initialisés à 1 et 3, les suivants à 0 (valeur par défaut pour le type de donnée).

Exemple : (non défini dans C99/C11, mais peut être pris en charge par certains compilateurs)

```
    #define NB_ELEM 5
    int tableau[NB_ELEM] = { };
```

Les éléments sont initialisés à 0 (valeur par défaut pour le type de donnée)

II. Structures ou enregistrements

A. Définition et syntaxe

Un type de donnée **STRUCTURE** ou ENREGISTREMENT est un **ENSEMBLE D'ELEMENTS** simples ou composés **REGROUPES DANS UNE MEME ENTITE** identifiée par un nom.
Ce NOUVEAU TYPE permet la gestion de données complexes.
Les données composant le type structure sont appelés '**MEMBRES'**.

Des variables de ce nouveau type pourront ensuite être déclarées et utilisées, tout comme les variables des types de base.

1. Définition d'un type structure (1)

Syntaxe de définition d'une structure :

```
struct nom_structure {
    . . . declaration_1 . . . ;
    . . . declaration_2 . . . ;
    . . .
    . . . declaration_N . . . ;
};
```

- **nom_structure**
  - l'identificateur du type de donnée structuré
- **declaration_1, declaration_1, declaration_N :**
  - déclarations des variables (type et nom) qui composent le type structuré (variables élémentaires, tableaux, autres variables structurées)

Exemple :

```
struct salarie {
    char nom[20] ;
    char prenom[20] ;
    int anneeNaissance ;
    float salaire[12] ;
};
```

2. Déclaration des variables de type structure (2)

Syntaxe de définition d'une variable de type enregistrement :

Pendant la définition de la structure :

```
struct nom_structure {
     . . . declaration_1 . . . ;
     . . . declaration_2 . . . ;
     . . .
     . . . declaration_N . . . ;
} var1 [,var2, . . ., varN ];
```

Ou bien après la définition de la structure :

```
struct nom_structure var1[,var2, . . ., varN ];
```

- **nom_structure** : identificateur du type structure
- **var1, var2, varN** : identificateur des variables déclarées

Remarque : le mot clef **struct** est optionnel

Exemple 1 : (définition d'un type et déclaration d'une variable de ce type)

```
struct salarie {
     char nom[20] ;
     char prenom[20] ;
     int anneeNaissance ;
     float salaire ;
} unSalarie;
```

Exemple 2 : (définition d'un type puis déclaration séparée d'une variable de ce type

```
struct salarie {
     char nom[20] ;
     char prenom[20] ;
     int anneeNaissance ;
     float salaire ;
};
 . . .
struct salarie unSalarie ;
```

3. Déclaration et initialisation d'une variable de type structure (3)

L'initialisation d'une variable de type structure peut intégrer l'initialisation des variables membres :

Exemple : déclaration et initialisation d'une variable de type structure

```
salarie unSalarie = {
     "dupont",
     "pierre",
     1980,
     1350.80} ;
```

B. Accès aux membres

L'accès aux membres utilise la notation pointée : nom de variable structure, un point, nom de la variable membre.

Exemple : Affectation d'une valeur à une des variables élémentaires du type structuré

```
unSalarie.anneeNaissance = 1980 ;
unSalarie.salaire[0] = 1350.80 ;
. . .
unSalarie.salaire = 1842.50 ;
```

```
strcpy(unSalarie.nom ,"dupont") ;
strcpy(unSalarie.prenom , "pierre") ;
```

C. Taille des enregistrements

La fonction «'sizeof » peut être utilisée pour déterminer la taille d'une variable structure :

Exemple :

```
struct salarie {
     char nom[20] ;
     char prenom[20] ;
     char titre; // 1 mr, 2 mme, 3 mmle
     int anneeNaissance ;
    float salaire ;
} unSalarie;
printf("Tester la fonction 'sizeof' sur un enregistrement ...");
printf("\n\t..struct salarie : %d octets",sizeof(struct salarie));
printf("\n\t..unSalarie : %d octets",sizeof(unSalarie));
```

Résultat obtenu :

```
Tester la fonction 'sizeof' sur un enregistrement ...
        ... struct salarie : 52 octets
        ... unSalarie : 52 octets
Process returned 0 (0x0)   execution time : 2.153 s
Press any key to continue.
```

Le détail du calcul manuel nous donne : 20 + 20 + 1 + 4 + 4 = 49 octets.
La compilation ajuste la taille à une frontière de mot mémoire (qui dépend de l'architecture de l'ordinateur) :

- Ainsi : 20 + 20 + 1 : occupe 41 octets
- **Mais** : 20 + 20 + 1 + 1 entier : occupera 48 octets (les 41 alignés sur une frontière de mot pour arriver à 44, plus les 4 octets de l'entier) ; ajouté aux 4 octets du float, cela donne 52 octets.

D. Imbrication de types structurés

Un type de donnée structuré peut être utilisé dans la définition d'un autre type de donnée structuré. L'accès aux membres sera réalisé en précisant la hiérarchie des noms de variables « traversées ».

Exemple :

```
// structure date
struct date {
     int jour ;
     int mois ;
     int annee ;
} ;

// la définition du type SALARIE utilise le type DATE
struct salarie {
     char nom[20] ;
     char prenom[20] ;
     struct date dateNaissance ;
     float salaire ;
```

```
} ;
struct salarie unSalarie ;
// accès aux membres
unSalarie.dateNaissance.jour = 1 ;
unSalarie.dateNaissance.mois = 12 ;
unSalarie.dateNaissance.annee = 1980 ;
```

E. Utilisation des types structurés dans les tableaux

La création de tableaux de variables de type structure permet le regroupement dans un seul tableau de toutes les variables nécessaires à la gestion d'un ensemble d'objets..

Exemple : un tableau au sein d'un type structure

```
. . .
struct salarie {
     char nom[20] ;
     char prenom[20] ;
     struct date dateNaissance ;
     float salaire;
     float salaireAnnee[12] ;
} ;
```

Exemple : un tableau de type structure

```
const int NB_FICHES = 100 ;
struct salarie ficheSalarie[NB_FICHES] ;
int i,j ;

for ( i=0 ;i<NB_FICHES; i++)
    {
          scanf("%s", ficheSalarie[i].nom);
          scanf("%d", & (ficheSalarie[i].dateNaissance.jour));
          . . .
          printf("saisie de 12 salaires :");
          for ( j=0 ; j<=11 ; j++)
              scanf("%f",& (ficheSalarie[i].salaireAnnee[j]));
    } ;
```

III.Définir des alias sur les types : typedef

L'ordre **typedef** permet la définition de synonymes sur des types de données existants (types élémentaires ou structurés).

A. Redéfinir le nom d'un type de base

On peut ainsi créer des noms personnalisés (intérêt pour redéfinir de manière plus claire des noms de types existants). Par convention, les noms des types de base redéfinis sont suffixés par '_t' (time_t)

Exemple : franciser le nom d'un type de base (pas d'intêret…)

```
. . .
typedef int ENTIER ; // création d'un synonyme

ENTIER i, j ; // déclaration de variables
```

Exemple : redéfinir un type pour lui donner un sens

```
. . .
typedef int distance ; // création d'un synonyme
typedef float vitesse;

distance d1, d2 ; // déclaration de variables
vitesse v1, v2 ; // déclaration de variables
```

B. Simplifier la déclaration des types structure

Exemple : utilisation pour les types structure

```
. . .
struct fiche_employe {
     char nom[20] ;
     char prenom[20] ;
     struct date dateNaissance ;
     float salaire;
     float salaireAnnee[12] ;
} ;

// déclaration de l'alias 'empl' pour 'fiche_employe'
typedef struct fiche_employe empl ;

empl e1, e2; // déclaration de variables
```

Exemple : utilisation pour les types structure

```
. . .
typedef struct fiche_employe {
     char nom[20] ;
     char prenom[20] ;
     struct date dateNaissance ;
     float salaire;
     float salaireAnnee[12] ;
} empl ; // alias


empl e1, e2; // déclaration de variables
```