STRUCTURES

Structures

- Une structure est un type qui regroupe, sous un seul nom, plusieurs variables appelées **champs** et pouvant être de types différents.

- Syntaxe :

```
struct Nom_Structure
{
type1 champ1;
type2 champ2;
..... ......
typeN champN;
};
```

Exemples

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

Définition

```
struct Individu
{
char nom[15];
char prenom[15];
int num_secu;
struct Date date_naissance;
};
```

Définition

- Les instructions ci-dessus déclarent deux modèles de structure appelées **Date** et **Individu**.

Déclaration de variables de type structure

- Attention : les déclarations précédentes ne sont pas des déclarations de variables; ce sont des déclarations de **modèles** (de types) à partir desquels on peut déclarer des variables.

- Syntaxe :

```
struct Nom_structure ident1, ident2;
```

- Exemple :

```
struct Date anniversaire, date_mariage;
struct Individu pere, mere, fille;
```

Initialiser une structure

Syntaxe : struct **Nom_structure** Nom_variable ={champ1,champ2 ...}

- Exemples :

```
struct point{
 int x ;
 int y ;
};

int main(){
 struct point p1 = {42,3} ;
 struct point p2 = {17,6} ;
 retrun 0;
}
```

Accès à un champ d'une structure

- Syntaxe :

  `Nom_variable.champ_de_structure`

- Exemples :

```
pere.num_secu = 12678;
pere.date_naissance.annee = 1964;
scanf("%s", fille.nom);
printf("nom = %s", fille.nom);
p1.x = 2*p2.y ;
```

Opérations globales sur les structures

- L'affectation globale entre deux variables d'un même type structuré est la seule opération autorisée.

- Exemples :

```
struct point p1, p2;
p1 = p2;
```

```
if (p1 == p2)
   printf("même date\n");
```

INTERDIT

Définition d'un type structure avec **typedef**

- Les définitions précédentes peuvent s'écrire en utilisant le mot clé **typedef** :

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

```
typedef struct {
int x;
int y;
} point;
```

```
typedef struct
{
char nom[15];
char prenom[15];
int num_secu;
Date date_naissance;
} Individu;
```

Définition d'un type structure avec **typedef**

- Avec **typedef, Date, Individu et point** deviennent des types.

- Simplification dans les déclarations des variables :

```
Date anniversaire, date_mariage;
Individu pere, mere, fille;
point p1, p2 ;
```

Structures et pointeurs

- Comme pour les autres types, il est possible de définir des pointeurs sur des structures :

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

Date date_naissance;
Date *p;
p = &date_naissance;
```

Accès aux champs d'une structure

- Avec un pointeur, l'accès aux champs se fait en utilisant les opérateurs :

  "->" (flèche), ou "(*p)."

- Exemples :

```
Date date_naissance;

Date *p;

p = &date_naissance;

p->jour = 1 ;        ⇔   (*p).jour = 1
p->annee = 1988 ;    ⇔   (*p).annee = 1988
```

Résumé

- Si on utilise le nom de la variable :
  - l'accès se fait par l’opérateur ' . ' (point)

- Si on utilise un **pointeur** sur la variable :
  - l'accès se fait via ' `->` ' (flèche)
  - ou par ' `(*ptr).` '

Remarque

- Dans une même expression, on peut être amené à combiner les deux opérateurs '.' (point) et '->' (flèche)

- **Exemple** :

```
individu *personne;
    personne = &pere;
    personne ->date_naiss.a = 1984 ;
    personne ->date_naiss.m = 10 ;
    personne ->date_naiss.j = 25 ;
    printf("nom:%s,naiss: %d\n",pere.nom,personne ->date_naiss.a);
    return 0;
    }
```

Structures et fonctions

Comme avec les autres types, une structure peut être passée par valeur ou par adresse.

Exemple :

```
void afficher_date(Date date_naissance)
/* passage par valeur */
{
   printf("%d/", date_naissance.jour);
   Printf("%d/", date_naissance.mois);
   printf("%d\n", date_naissance.annee);
}
```

Passage par valeur

```
void saisir_date(Date date) /* passage par valeur */
{
    date.jour = 1;
    date.mois = 1;
    date.annee = 1988;
}

void main(void){
Date nouvelle_date = {0,0,0};
saisir_date(nouvelle_date);
afficher_date(nouvelle_date);
}
```

Produit l'affichage 0/0/0

Pour modifier la valeur, utiliser le passage par adresse

Passage par adresse

```
void saisir_date(Date *date) /* passage par adresse */
{
    date->jour = 1;
    date->mois = 1;
    date->annee = 1988;
}

void main(void){
Date nouvelle_date;
saisir_date(&nouvelle_date);
afficher_date(nouvelle_date);
}
```

Produit l'affichage 1/1/1988

Passage par adresse

- Saisie par la fonction scanf :

```
void saisir_date(Date *date)
{
scanf("%d",&(date->jour));
scanf("%d",&(date->mois));
scanf("%d",&(date->annee));
}
```

Passages par valeur ou par adresse ?

- Le passage d'un paramètre par valeur entraîne **la recopie** de la variable passée dans le paramètre de la fonction.

- Si la structure est grande, passer les structures par adresse même si la fonction ne modifie pas la variable structure.
  - Réduire le temps d’exécution (temps de la recopie)

Retourner une structure

- Une fonction peut retourner une structure :

```
Date saisir_date(void)
{
Date d;
scanf("%d",&(d.jour));
scanf("%d",&(d.mois));
scanf("%d",&(d.annee));
return d;
}
```

```
int main(void)
{
Date date_saisie;
date_saisie = saisir_date();
afficher(date_saisie);
....
return 0;
}
```

Tableaux de structures

- Comme avec les autres types, on peut définir des tableaux de structures :

```
Date liste_date[10];
```

- Chaque élément du tableau est une structure :

```
liste_date[0].jour = 1;
liste_date[0].mois = 1;
liste_date[0].annee = 1988;

liste_date[1].jour = 17;
liste_date[1].mois = 12;
liste_date[1].annee = 2011;
```