Huitième partie

Les structures

Notion de structure

- *Il est habituel en programmation d'avoir besoin d'un mécanisme permettant de regrouper un certain nombre de variables.*
- *On a déjà vu que les tableaux permettaient de regrouper plusieurs éléments. Toutefois, ceux-ci devaient être tous du même type.*
- *Il est parfois nécessaire de regrouper des variables de types différents.*

Notion de structure

- ***Par exemple, pour créer un fichier de personnes on aurait besoin d'une structure de la forme :***
  - ***NOM*** *: chaîne de caractères*
  - ***AGE*** *: entier*
  - ***TAILLE*** *: réel*
- ***La solution est le concept de structure qui est un ensemble d'éléments de types différents repérés par un nom.***
- ***Les éléments d'une structure sont appelés membres ou champs de la structure.***

Déclaration des structures

- *Syntaxe :*

```
struct{
   liste des membres
}
```

- *Exemple :*

```
struct{
    char *Nom;
    int Age;
    float Taille;
} p1,p2;
```

Déclaration des structures

- ***La déclaration précédente déclare deux variables de noms p1 et p2 comme deux structures contenant trois membres.***
- ***L'espace mémoire nécessaire a été réservé pour que les deux variables contiennent chacune trois membres.***

p1	
Nom	char *
Age	int
Taille	float

p2	
Nom	char *
Age	int
Taille	float

Déclaration des structures

- *L'inconvénient de cette méthode est qu'il ne sera plus possible de déclarer d'autres variables du même type.*
- *Si nous déclarons ensuite :*

```
 struct{
    char *Nom;
    int Age;
    float Taille;
} p3;
```

- *p3 ne sera pas considérée du même type, il sera impossible en particulier d'écrire p1 = p3;.*

Déclaration des structures

- *Heureusement, il est possible de définir des modèle de structure.*
- *Syntaxe :*

```
struct identificateur{
  liste des membres
}
```

- *Exemple :*

```
struct personne{
   char *Nom;
   int Age;
   float Taille;
};
```

Déclaration des structures

- *Une telle déclaration n'engendre pas de réservation mémoire.*
- *Le modèle ainsi défini peut être utilisé dans une déclaration de variable.*
- *Exemple :*
  struct personne Michel, Anne;
  / Déclare deux variables de type "struct personne" */*

Déclaration des structures

- ***Dans une structure, tous les noms de champs doivent être distincts.***
- ***Par contre rien n'empêche d'avoir 2 structures avec des noms de champs en commun : l'ambiguïté sera levée par la présence du nom de la structure concernée.***

```
struct personne{
   char *Nom;
   int Age;
   float Taille;
}
```

```
struct pays{
   char *Nom;
   int nb_habitants;
   int superficie;
}
```

Initialisation de structures

- *Une déclaration de structure peut contenir une initialisation par une liste d'expressions constantes à la manière des initialisations de tableau.*
- *La valeur initiale est constituée d'une liste d'éléments initiaux (1 par champ) placée entre accolades.*
- *Exemple :*
  struct personne p = {"Dupond", 25, 1.80};

Accès à un champ

- *L'opérateur d'accès est le symbole "." (point) placé entre l'identificateur de la structure et l'identificateur du champ désigné.*
- *Syntaxe :*

  <ident_objet_struct>.<ident_champ>
- *Exemples :*

```
p1.Age = 25;
if(p2.Nom[0]=='D')
...
```

Affectation des structures

- *On peut affecter l'intégralité des valeurs d'une structure à une seconde structure ayant impérativement le même type.*
- *Exemple :*

```
struct personne p1,p2;
p1 = p2;
```

- *L'affectation d'une structure recopie tous les champs un à un.*
- *Attention la recopie est superficielle.*

Tableau de structures

- *Une déclaration de tableau de structures se fait selon le même modèle que la déclaration d'un tableau dont les éléments sont de type simple.*
- *Pour déclarer un tableau de 100 structures personne, on écrira :*
  ```
  struct personne t[100];
  ```
- *Pour référencer le nom de la personne qui a l'index i dans t, on écrira :*
  ```
  t[i].Nom = "Dupond";
  ```

Structures composées

- *Puisqu'une structure définit un type, ce type peut être utilisé dans une déclaration de variable mais aussi dans la déclaration d'une autre structure comme type de l'un de ses champs.*
- *Exemple :*

```
struct Date
{
   int Jour, Mois, Annee;
};
struct personne
{
   char *Nom;
   struct Date Naissance;
};
```

Structures composées

- *Exemple d'initialisation :*

```
struct personne p = {"Dupond",
{21,05,1980}};
```

- *Exemple d'accès aux membres :*

```
if (p.Naissance.Annee < 1985)
    if(p.Naissance.Mois == 5)
```

Pointeurs vers une structure

- *Supposons la structure personne déclarée, nous pouvons déclarer une variable de type pointeur vers cette structure de la façon suivante :*
  *struct personne *p;*
- *Nous pouvons alors affecter à p des adresses de struct personne.*

Pointeurs vers une structure

Exemple :

```
struct personne
{
    char *Nom;
    int Age;
};

int main()
{
  struct personne pers;
  struct personne p*;

  p = &pers;
}
```

Accès aux champs d'une structure pointée

- *Soit p un pointeur vers une structure personne, *p désigne la structure pointée par p.*
- *Mais *p.Nom ne référence pas le champ Nom de la structure car l'opérateur d'accès aux champs a une priorité supérieure à l'opérateur d'indirection.*
- **p.Nom est équivalent à *(p.Nom)*
- *Il faut donc écrire (*p).Nom pour forcer l'indirection à se faire avant l'accès au champ.*

Accès aux champs d'une structure pointée

- *Pour alléger la notation, le langage C a prévu un nouvel opérateur noté -> qui réalise à la fois l'indirection et la sélection.*
- *p->Nom est équivalent à (*p).Nom*
- *Exemple :*

```
struct personne pers;
struct personne *p;

p = &pers;
p->Nom="Dupont";
```

Champ pointant vers une structure

- *Exemple :*

```
struct Date
{
    int Jour, Mois, Annee;
};

struct personne
{
    char *Nom;
    struct Date *Naissance;
};
```

Champ pointant vers une structure

- *Accès aux champs :*

```
pers.Nom = "Dupont";
pers.Naissance->Jour = 21;
pers.Naissance->Mois = 05;
pers.Naissance->Annee = 1985;
```

- *Si p est un pointeur sur pers :*

```
p->Nom = "Dupont";
p->Naissance->Jour = 21;
p->Naissance->Mois = 05;
p->Naissance->Annee = 1985;
```

Structures récursives

- *Ce genre de structures est fondamental en programmation car il permet d'implémenter la plupart des structures de données employées en informatique (listes, files, arbres, etc...).*
- *Ces structures contiennent un ou plusieurs champs du type pointeur vers une structure du même type et non de simples champs du type structure.*

Structures récursives

- *Exemple :*

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

- *Déclaration incorrecte car la taille de Pere et Mere n'est pas connue :*

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

Passage comme argument

- *La norme ANSI a introduit le transfert direct d'une structure entière comme argument d'une fonction, ainsi que la remontée de cette structure depuis une fonction appelée via une instruction return.*
- *Le passage de paramètres de type structure se fait par valeur.*
- *Pour les compilateurs antérieur à la norme ANSI, comme le C K&R, il n'est pas possible de passer en paramètre une structure, il faut utiliser un pointeur sur cette structure.*

Passage comme argument

```
struct personne
{
  int jour, mois, annee;
}

int cmp_date(struct date d1, struct date d2)
{
  if (d1.annee > d2.annee)
            return (APRES);
  if(d1.annee < d2.annee)
            return (AVANT);
… /* comparaison portant sur mois et jour */

}
int main()
{
  struct date1, date2;

  if(cmp_date(date1,date2)==AVANT)
     …
}
```

typedef

- *Pour simplifier les écritures et éviter de répéter systématiquement struct personne, on utilise typedef pour définir les structure ou union comme des nouveaux types.*
- ***Exemple :***

```
typedef struct personne
{
    char * Nom;
    int Age;
} Personne;

int main()
{
  Personne p;
  …
}
```