Structures

Définition

La structure est un type complexe dont les parts ne sont pas du même type. Par exemple un étudiant a un nom qui est une chaîne de caractères, date de naissance qui est une date codée de plusieurs façons, sexe qui peut être codé soit par un caractère, soit par un nombre, et un ensemble de notes qui sont des nombres.

Etudiant	Nom	Texte
	Date_naiss	Date
	Sex	M ou F
	Notes	Nombres

Déclaration et présentation

Déclaration

La syntaxe de la déclaration est montrée au dessous. Chaque partie de la structure est appelée ***champ***. Chaque champ est déclaré par son type et son nom. La syntaxe est la pareille à cette de déclaration d'une variable.

```
struct nom_de_struct_opt{
    declare_champ_1; declare_champ_2; ... ;
    declare_champ_n; };
```

De cette syntaxe générale on peut dériver trois manières différentes. La première est de déclarer la structure avec un nom et plus tard de déclarer des variables de ce type. Dans l'exemple au dessous la structure *etudiant* est déclarée et puis sont déclarées deux variables de ce type *e1* et *e2*. La date de naissance est codée par un tableau de trois nombres entiers qui présentent le jour, le mois et l'année respectivement.

```
struct etudiant{
    int numero;
    char nom[30];
    char sex;
    int naiss[3];
    double notes[5];
};// déclaration de la structure étudiant
struct etudiant e1,e2;
```

On peut de déclarer les variables dans la déclaration de la structure, comme sont déclarés les points *p1* et *p2*.

```
struct point{
    int x;
    int y;
} p1,p2;// déclaration combinée
struct point p3,p4;
```

On peut omettre le nom de la structure mais l'inconvénient de cette méthode est qu'il sera par la suite impossible de déclarer une autre variable du même type.

Dans l'exemple au dessous sont deux structures identiques sont déclarées mais les variables *c1* et *c2* ne sont pas considérées de même type et on ne peut pas copier le contenue de *c1* dans *c2* par simple affectation. C'est à dire que l'instruction `c2=c1` ne sera pas valide.

```
struct {
    double r; double im;
} c1;

struct {
    double r; double im;
} c2; // c1 et c2 ne sont pas de même type
```

Pour abréger encore le texte du programme on peut déclarer le nom du type par ***typedef*** et puis utiliser le nom sans le mot ***struct*** devant lui, comme c'est fait pour le type *DATE* (fig.9.1).

Présentation

Dans la mémoire de l'ordinateur tous les champs sont empaquetés un après l'autre selon la déclaration Ils sont alignés à la limite d'octet ou de mot selon la configuration du compilateur (fig.9.1)

Représentation dans la mémoire		
etud	numero	
	nom	
	sex	
	naiss	jour
		mois
		anne
	notes	0
		1
		2
		3
		4

```
typedef struct {int jour;
       int mois;
       int annee;} DATE;
struct etudiant{
    int numero;
    char nom[30];
    char sex;
    DATE naiss;
    double notes[5];}
    etud ;
```

Fig. 9.1

Initialisation

L'initialisation est similaire à cette des tableaux, les **expressions constantes** sont mises en accolades et séparées par virgules. leur nombre et leur ordre doit correspondre à la déclaration.

```
typedef struct {double x, double y} Point;
struct fiche {
   int numero;
   char nom[32], prenom[32];
} u = { 1234, "DURAND", "Pierre" };
Point p1 ={3.45, 2.56}, p2=[0.0,0.0};
```

Opérateurs

Les opérateurs qu'on peut appliquer sur les objets du type structure sont l'affectation et l'accès aux champs de la structure.

Affectation

On peut affecter la valeur d'une structure à une variable du même type. C'est équivalent à l'affectation des champs correspondants.

```
typedef struct etudiant{
    int numero;
    char nom[30];
    char sex;
    DATE naiss;
    double notes[5];
    double moyenne ;} ETUDIANT ;
ETUDIANT e1 = {245, "Dubois", 'M',{12,2,1988},
              {12,10,8,14,6}}, e2;
...
e2 = e1;
```

Accès aux champs

L'opérateur d'accès et le point (.). La syntaxe est :

```
nom_de_varaible_structure.nom_du-champ
```

Exemples :

```
e1.numero = 1234;
e2.naiss.jour = 3;
e2.naiss.mois = 4;
e2.naiss.annee = 1983;
e2.notes[0] = 14;
```

On peut voir que si la structure comporte une autre structure l'accès vers les champs de la structure interne est faite par un deuxième point.

Pointeurs vers structures

Déclaration

Les pointeurs vers les structures sont déclarés de la même façon que vers les types simples. La syntaxe est :

```
type_structure *nom
```

Exemples

```
ETUDIANT *p1;
struct point {int x, int y} pt1,pt2, *p4;
p1 = &e1;
p4 = &pt2;
```

Accès aux champs d'une structure désignée par un pointeur

L'opérateur est la flèche (->) :

```
nom_de_pointeur_vers_structure->nom_du-champ
```

Exemples

```
p1->numero = 1234;
p1->naiss.jour = 3;
p4->x = 3;
```

Si nous avons les déclarations et les affectation suivantes, les expression qui se trouvent dans chaque ligne du tableau sont équivalentes

```
ETUDIANT *p1,et;
struct point {
    int x, int y
   } pt1,pt2, *p4;
p1 = &e1;
p4 = &pt2;
```

et.numero	p1->numero	(&et)->numero	*numero* de *et*
et.naiss.annee	p1->naiss.annee	(&et)->naiss.annee	*annee* de *naiss* de *et*
pt2.x	p4->x	(&pt2)->x	*x* de *pt2*
&et.numero	&p1->numero	&(&et)->numero	L'adresse de *numero*
&pt2.x	&p4->x	&(&pt2)->x	L'adresse de *x*

Les structures et les fonctions

Paramètres de fonctions

Les structures peuvent être utilisées comme paramètres d'une façon identique que les types simples.

Exemples :

```
void affichEtud(ETUDIANT e) {
      printf ("%6d  %-15s  %2d/%2d/%4d
%5.2lf\n",
        e.numero,e.nom, e.naiss.jour,
       e.naiss.mois, e.naiss.annee, e.moyenne);
}
```

```
void affichEtud(ETUDIANT e) {
       DATE *d = &e.naiss;
      printf ("%6d  %-15s  %2d/%2d/%4d
%5.2lf\n",
        e.numero,e.nom, d->jour,
       d->mois, d->annee, e.moyenne);
}
```

Quand la structure est trop grande (plus de 10-15 octets) il est plus efficace de définir le paramètre comme un pointeur vers structure (pourquoi ?).

```
void affichEtud(ETUDIANT *e) {
       DATE *d = &e->naiss;
      printf ("%6d  %-15s  %2d/%2d/%4d  %5.2lf\n",
        e->numero,e->nom, d->jour,
       d->mois, d->annee, e->moyenne);
}
```

Résultat de fonction

Une valeur du type structure peut servir comme résultat d'une fonction L'exemple au dessous est simplifié.

```
ETUDIANT lireEtud(void){
    ETUDIANT e;
    int i;
    printf("Tapez le numero, le nom, et le gendre  d'etudiant…");
    scanf("%d , %s , %c \n",&e.numero,&e.nom,&e.sex);
    puts("Tapez la date de naissance…");
    scanf(", %2d / %2d / %d \n", &e.naiss.jour,
              &e.naiss.mois, &e.naiss.annee);
    printf("Tapez les %d notes d'etudiant \n", MAXNOTES);
    for (i=0; i <MAXNOTES; i++) scanf(", %lf",(e.notes+i));
    return e;
  }
```

Tableaux de structures

Le type de base d'un tableau peut être n'import quel type y compris une structure. Ils sont utilisée quand on a besoin de traiter plusieurs objets complexes du même type. Voici quelque exemples :

```
ETUDIANT groupe[40];
struct point polygone[20];
```

Chacun d'élément du tableau est une structure. Donc pour accéder les champs on doit utiliser les opérateurs d'accès point (.) ou flèche.

```
groupe[0].numero = 12345;
(groupe+1)->numero = groupe->numero +1;
    /* groupe[1].numero = groupe[0].numero+1; */
groupe[3].notes[2] = 12;
polygone[10].x = 4;
(polygone+10)->y = 6;
```

Exemple final

Trouver les moyennes des notes d'un groupe d'étudiants. On a tous les fonctions sauf main et la fonction qui trouve les moyennes.

```
void main(void){
      ETUDIANT groupe[MAXETUD];
      int netud,i;
      netud = lireGroupe(groupe, MAXETUD);
      calcMoyennes(groupe, netud);
    printf("num       nom               date        moyenne\n");
      for (i=0; i<netud;i++) affichEtud(groupe[i]);
  }
 void calcMoyennes(ETUDIANT g[], int n){
     int i,j;
     for (i=0 ; i< n; i++) {
       g[i].moyenne = 0;
       for (j=0; j<MAXNOTES; j++)g[i].moyenne +=g[i].notes[j];
     g[i].moyenne /= MAXNOTES;
     }
 }
```