Chapitre 6

Les types composés

Chapitre 6 : Les types composés

I Le type tableau

- La plupart des langages de programmation disposent du concept de tableau. Cette notion offre au programmeur deux possibilités :

  - la déclaration de tableaux

  - l'utilisation de l'opérateur d'indexation, noté () ou [], admettant en opérande un tableau **t** et un entier **i**, donnant la valeur de l'élément d'index **i** de **t**.

- En C, toute déclaration d'une variable de type tableau est une déclaration d'un ***pointeur*** à valeur **constante**, non modifiable, correspondant à l'**adresse du premier élément de la table**.

Chapitre 6 : Les types composés

Exemple **:**

```
int tab[5];
```

- On a bien déclaré une variable de type tableau mais cette variable n'a pas de nom.
- **tab** est le nom d'une **constante** de type ***pointeur*** vers un ***int*** et c'est **l'adresse** du premier élément.
- les bornes de la table vont de 0 à 4.
- **tab** est rigoureusement identique à **&**tab[0].
- De ce fait, on peut utiliser un **pointeur** pour parcourir les éléments de **tab**.

Chapitre 6 : Les types composés

Reprenons l'exemple :

```
int main()
{
 int tab[5] = {4, 6, 8, 1, 2}; /* déclaration + initialisation */
 int i;

  for (i = 0; i<5; i++)  printf (" %d  ", tab[i]);
}
```

Affiche le même résultat que :

```
int main()
{
 int tab[5]= {4, 6, 8, 1, 2};
 int i, *p;
 p = tab;     /* on affecte à p une adresse */
 for (i = 0; i<5; i++)
   {
     printf(" %d   ", *p);
     p++;
   }
}
```

Résultat affiché : 4 6 8 1 2

Chapitre 6 : Les types composés

Remarques :

✓ on peut utiliser la notion de tableau de manière classique telle que vue en algorithmique.

✓ à la déclaration, le tableau peut être initialisé.

✓ si **n** est le nombre d'éléments du tableau, ils sont indexés de **0** à **n-1**

✓ **int** t**[]** = **{**6, 3, 18**}**; ⇒ fixe la taille du tableau à 3 (*cases indexées 0, 1, 2*).

✓ pointeurs et tableaux se manipulent de la même manière ⇒

tab**[**i**]** est équivalent à ***(**tab+i**)** ou à ***(**i+tab**)** (ou à i[tab]! : à éviter).

Chapitre 6 : Les types composés

L'exemple précédent peut s'écrire :

```
int main()
{
 int tab[5]= {4, 6, 8, 1, 2};  /* déclaration + initialisation par un agrégat de valeurs */
                               /* la taille de l'agrégat peut être < 5 => le compilateur complète par des 0 */

 int i,  * p;
 p = tab;              /* idem : p = &tab[0]; */
 for (i = 0; i<5; i++)   printf ("%d  ", p[i]);
}
```

Chapitre 6 : Les types composés

❑ On peut créer un tableau dont la taille est une variable du programme => allocation dynamique de place mémoire

```
#include <stdlib.h>
int main()
{
  int * t;
  int n;   /* n : nombre d'éléments */
  ...
  n = ...    /* initialisation de n*/
  ...
 t = (int *) malloc (n * sizeof ( int ) );
/* ou :  t = (int *)calloc (n,sizeof(int)); pour initialiser à 0 */
  ...        /* traitement */
  free(t);    /* on libère la place mémoire réservée */
}
```

❑ On manipulera les éléments de **t** avec l'opérateur d'indexation **[]**

Chapitre 6 : Les types composés

Différences principales entre tableau et pointeur :

- un pointeur doit être obligatoirement ***initialisé***
  - soit par affectation d'une adresse
  - soit par allocation dynamique
- un tableau ne peut figurer en partie gauche d'une affectation :

  t++, t-- : ***opérations non autorisées***

  (contrairement au pointeur)
- Un nom de tableau n'est pas un équivalent à une variable pointeur:
  - une variable de type pointeur est une variable qui contient une adresse
  - un nom de tableau est une constante qui désigne une adresse

Chapitre 6 : Les types composés

Déclaration statiques de tableaux à plusieurs dimensions :

type nom_tableau [dimension1] [dimension2]... [dimensionN]

Exemple :

```
#include <stdio.h>
#define NBL 2
#define NBC 3
int main()
{
  int t[NBL] [NBC] = { { 1,2,3 }, { 5,6,7} };
  int l, c;                          /* indices de lignes et de colonnes */
  for (l = 0; l < NBL; l++)
    {
      for (c = 0; c < NBC; c++) printf("Elément de coordonnées ("  %d,%d ) : %d\n",l, c, t[l][c]);
      printf( "  \n" );
    }
}
```


Chapitre 6 : Les types composés

II Les chaînes de caractères

- Une chaîne de caractères est un tableau à une dimension dont les éléments sont de type *char :*

  char nom[taille];

- Elle peut être initialisée par une liste de caractères ou par une chaîne de caractères constante (littérale).

- A la saisie, toute chaîne est complétée par le caractère **nul** **'\0'**, qui est la ***marque de fin d'une chaîne*** ⇒ Il faut que la taille de la variable ait au moins un élément de plus que le nombre de caractères qui la composent.

Chapitre 6 : Les types composés

Exemple :

char ch [8] = "bonjour"; /* *7 caractères + '\0'* */

char ch [] = "bonjour";

/* *par défaut, la taille sera définie par le nombre de caractères de la valeur d'initialisation* */

❑Les opérations de manipulation de chaînes nécessitent l'utilisation de la librairie ***string***. Le fichier d'en-tête est inclus avec la directive :

#include <string.h>

Saisie : **scanf** ("**%s**",ch); /* *scanf rajoute automatiquement* **'\0'** */

Affichage : **printf**("**%s**",ch);

Remarque : Dans le scanf, ch est l'adresse du premier caractère.

Chapitre 6 : Les types composés

Entrés/sorties spécifiques de chaînes :

➢ La fonction ***scanf*** ne prend pas en charge les chaînes contenant les espaces. On peut y remédier en utilisant la fonction ***fgets*** :

fgets (*nom_chaîne*, *taille_chaîne*, *nom_entrée_standard*);

- **fgets** lit un nombre de caractères ***n*** = ***taille_chaîne -1*** et place le tout dans nom_chaîne avec le caractère de fin **'\0'**).
- *nom_entrée_standard* : ***stdin***
- *taille_chaîne* : taille physique
- la saisie est validée par **'\n'**
- **fonctionnement** : La fonction extrait le caractère **'\n'** (*touche entrée qui valide la saisie*) dans le flux saisi quand elle le peut (*selon l'espace libre dans la chaîne à ce moment*). Si ce caractère est présent à la fin de la chaîne, on sait que la saisie est réussie car tous les caractères ont été stockés dans la variable.

Chapitre 6 : Les types composés

Exemple : char ch[5];
fgets(ch, 5, stdin);
ou : **fgets**(ch, sizeof(ch), stdin);

Deux cas peuvent se présenter :

✓ On saisit 3 caractères au plus et on valide la saisie par **ENTREE**. Tous les caractères saisis, y compris le caractère de fin de ligne **'\n'**, sont copiés dans ch puis le caractère de fin de chaîne **'\0'** est ajouté.

✓ On saisit plus de 3 caractères (c'est-à-dire >= 4) et on valide la saisie par **ENTREE** ⇒ seuls les 4 premiers caractères sont copiés dans ch puis le caractère de fin de chaîne **'\0'** est ajouté.

Chapitre 6 : Les types composés

b) La fonction ***fputs*** permet d'afficher une chaîne dans la sortie standard :

fputs(*nom_chaîne*, *nom_sortie_standard*);

❑ *nom_sortie_standard* : ***stdout***

Exemple : **fputs**(ch, stdout);

Chapitre 6 : Les types composés

III Les structures

Dans le langage C on distingue la **création** d'un type **structure** de la **déclaration** d'une variable dans ce type.

Syntaxe :

- création type :

```
struct nom_type
{
 type_1  champ_1;
 type_2  champ_2;
 ...
 type_n  champ_n;
};
```

- déclaration de variable :

```
struct nom_type  nom_variable;
```

- accès à un champ :

nom_variable.nom_champ

(utilisation de la notation pointée)

Chapitre 6 : Les types composés

Exemple *:*

```
struct personne
{
  char nom[20];
  char prenom[25];
  int  age;
};
struct  personne  pers1, pers2 ;
```

On peut déclarer des variables de type **struct** sans donner de nom au type :

```
struct
{ char nom[20];
  char prenom[25];
  int  age;
} pers1, pers2;

/* pers1, pers2 : 2 variables de même type */
```

Chapitre 6 : Les types composés

Redéfinition du type struct :

On peut donner un nom à un type *structure* par la directive **typedef** pour rendre son utilisation plus aisée.

Exemple :

```
typedef  struct
{
  char nom[20];
  char prenom[25];
  int  age;
} Personne;

Personne p1, p2;
```

Autre écriture :

```
struct personne
{
  char nom[20];
  char prenom[25];
  int  age;
};
typedef struct personne Personne;
/* on renomme le type structure personne Personne */

Personne p1, p2;
```


Chapitre 6 : Les types composés

Pointeur sur une structure :

On peut manipuler des variables de type structure à travers des pointeurs.
L'adresse de la variable correspond à l'adresse du premier champ de la structure.

Exemple :

```
struct etudiant
{
  char nom[20];
  int age;
};

typedef struct  etudiant  * promo;  /* on renomme le type pour le réutiliser */

promo ensemble ;  /* ensemble est une variable de type pointeur sur un objet de type struct etudiant */
```

Chapitre 6 : Les types composés

Si on veut gèrer **n** étudiants, on réserve la place mémoire :

ensemble = (**promo**) malloc (**n** * sizeof (**struct etudiant**));
/ la variable ensemble peut être utilisée comme un tableau */*

Accès :

Si **p** est un *pointeur* sur une *structure*, l'accès à la **valeur** (*contenu*) d'un champ de la structure se fait de deux manières :
(* p) . nom_champ **ou** p -> nom_champ

Remarque : Dans la 1° forme les parenthèses sont importantes car le . est prioritaire sur *

Chapitre 6 : Les types composés

IV Les énumérations

On peut définir un type par la liste des valeurs qu'il peut prendre.

Syntaxe : **enum** nom_type **{** *constante_1*, *constante_2*,..., *constante_n* **};**

Les valeurs constantes sont codées par des entiers entre **0** et **n-1**.

Exemple :
enum couleur **{** bleu, rouge, blanc, jaune **};** */* création du type */*
enum couleur C; */* déclaration de la variable C dans le type énuméré couleur */*
C = blanc; */* initialisation */*
Autre écriture : **typedef enum** couleur **{** bleu, rouge, blanc, jaune**}** **Couleur**;
*/*création+renommage */*
Couleur C = blanc; */* déclaration de la variable C dans le type + initialisation */*

Chapitre 6 : Les types composés

V Les unions

- Il est parfois nécessaire de manipuler des variables auxquelles on désire affecter des valeurs de types différents.
- On veut par exemple manipuler des nombres qui seront implémentés par des entiers ***(****int****)*** tant que la précision de la machine le permet et qui passeront à une représentation sous forme de réels (*float*) dès que ce ne sera plus le cas.
- Il sera nécessaire et utile de disposer de variables pouvant prendre soit des valeurs *entières* soit des valeurs *réelles*.
- Dans le langage C, ceci peut se réaliser grâce au mécanisme des **unions.**
- La définition d'une **union** a la même syntaxe qu'une définition de structure : on remplace le mot clé **struct** par le mot clé **union**.

Chapitre 6 : Les types composés

Exemple **:**

```
union nombre
{
  int i;
  float f;
};
```

Déclaration **:** **union nombre** nb**;**

Différence entre les types **struct** et **union :**

- dans une variable de type **struct** tous les champs peuvent avoir en même temps une valeur
- une variable de type **union** ne peut avoir, à un instant donné, qu'**un seul champ ayant une valeur**.

Exemple **:** la variable nb pourra posséder soit une valeur entière soit une valeur réelle mais pas les deux à la fois.

Chapitre 6 : Les types composés

Accès aux champs de l'union :

Utilisation de la notation pointée : **.** (*comme pour le type structure*)

Exemple **:**

✓ pour affecter à **nb** une valeur *entière* : nb.i = 10;

✓ pour affecter à **nb** une valeur *réelle* : nb.f = 8.5;

Utilisation pratique des unions :

✓ Lorsqu'on manipule des variables de type **union**, il n'y a aucun moyen de savoir, à un instant donné, quel champ de l'union possède une valeur.

✓ Pour être utilisable, une **union** doit être ***associée*** à une variable dont le but est d'indiquer le champ de l'union qui est valide.

✓ En pratique, une **union** et son **champ indicateur** sont généralement englobés à l'intérieur d'une **structure**.

Chapitre 6 : Les types composés

Exemple **:**

```
#define ENTIER 0
#define REEL 1

struct arith
{
 int type_val;       /* champ à valeur ENTIER ou REEL */
 union
  {
   int i;
   float f;
  } u;
};
```

Déclarations de variables :

```
struct arith  n1, n2;
```

Utilisation :

```
n1.type_val = ENTIER;      n1.u.i = 10;
n2.type_val = REEL;        n2.u.f = 8.5;
```

✓ Pour savoir si **l'union** possède un type *entier* ou un type *réel* on teste la valeur du champ type_val si on passe la variable en paramètre à une fonction/procédure.

Une méthode pour alléger l'accès aux champs :

Pour accéder à un champ, on peut alléger l'écriture en utilisant les facilités du pré-processeur. On peut écrire par exemple :

```
#define I  u.i
#define F  u.f

n1.type_val = ENTIER;      n1.I = 10;
n2.type_val = REEL;        n2.F = 8.5;
```