Structures

1. Notion de structure

1.1 Introduction

La structure de tableau permet de traiter un nombre fini d'informations de même type auxquelles on peut accéder par un indice. Or souvent il est nécessaire d'organiser de façon hiérarchique un ensemble de données de types différents mais qualifiant un même objet comme par exemple :

- les dates chronologiques (année, mois, jour),
- les fiches bibliographiques (titre du livre, auteur, date de parution),
- les fiches personnelles (nom, prénom, âge, sexe, taille).

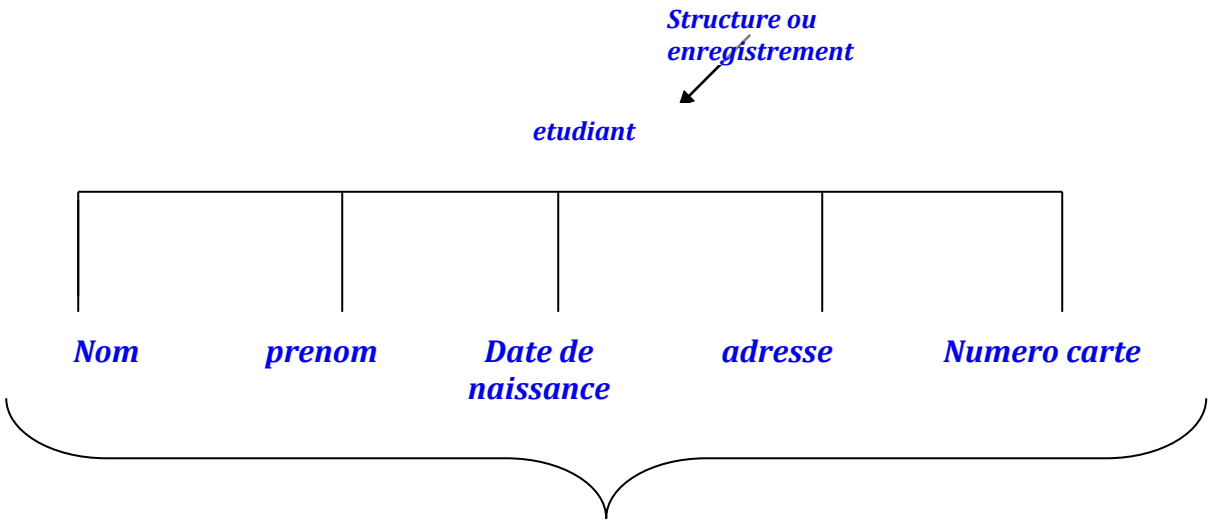
La nature différente de ces éléments (données) conduit le programmeur à utiliser une structure permettant la définition explicite de chacun de ces éléments : structure enregistrement.

1. Notion de structure

Définition:

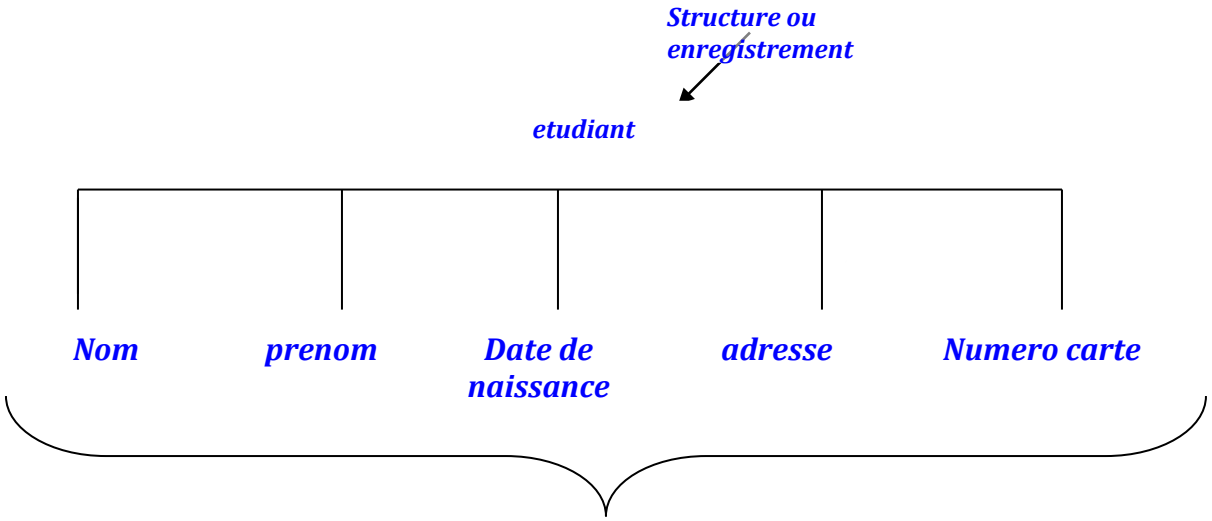
Un enregistrement est un type structuré formé d'un ou de plusieurs éléments pouvant être de types différents.

- Chaque enregistrement est désigné par un identificateur (son nom).
- Chacun des éléments de l'enregistrement occupe un emplacement appelé champ désigné à son tour par un identificateur et caractérisé par un type. Une information représente la valeur du champ.



1. Notion de structure

Définition:



Exemple :

Champs de l'enregistrement

Etudiant est la variable enregistrement,

Nom, prenom, date de naissance, adresse, numero carte sont les champs de l'enregistrement.

2:Définition d'une structure en C

- Déclaration d'une structure : syntaxe

```
struct nomdelastructure {
        typeelement_1 identificateur_1 ;
        typeelement_2 identificateur_2 ;
        …
        typeelement_n identificateur_n ;
        };
```

- Exemple : compte bancaire

```
struct compte {
        int num_compte ;
        char nature ;
        char nom[50];
        float solde;
        };

struct compte a,b,c; /*déclaration de 3 variables de ce type*/
```

2.1:Déclarations de variables

- Autres façons de déclarer des variables structure

```
struct compte {
       int no_compte ;
       char etat ;
       char nom[50];
       float solde;
       } a, b;   /*déclaration de 2 variables de ce type*/
struct compte c; /*déclaration de 1 variable de ce type*/
```

```
struct          { /* le nom de la structure est facultatif */
       int no_compte ;
       char etat ;
       char nom[50];
       float solde;
       } a,b,c;   /*déclaration de variables de ce type ici */
/* mais plus de possibilité de déclarer d'autres variables de
ce type*/
```

Déconseillé

2.1Déclarations de variables

- Autres façons de déclarer des variables structure

```
typedef struct compte {
        int no_compte ;
        char nature ;
        char nom[50];
        float solde;
        } cpt ;
/* cpt est alors un type équivalent à struct  compte*/

cpt a,b,c; /*déclaration de variables de ce type*/
```

- Dans ce cas puisque on ne se sert plus de "struct compte" par la suite

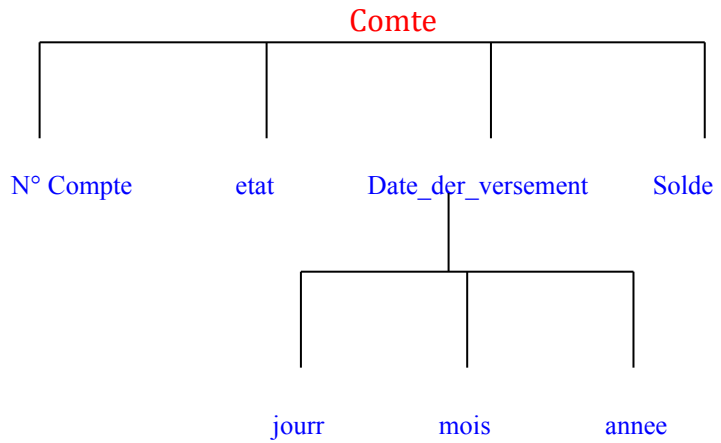
```
typedef struct {
        int no_compte ;
        char nature ;
        char nom[50];
        float solde;
        } cpt ;
```

2.2 Structures imbriquées

- Une structure peut être membre d'une autre structure

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

struct compte {
        int no_compte ;
        char nature ;
        char nom[50];
        float solde;
        struct date dernier_versement;
        };
```



- Remarque : ordre de déclaration des structures

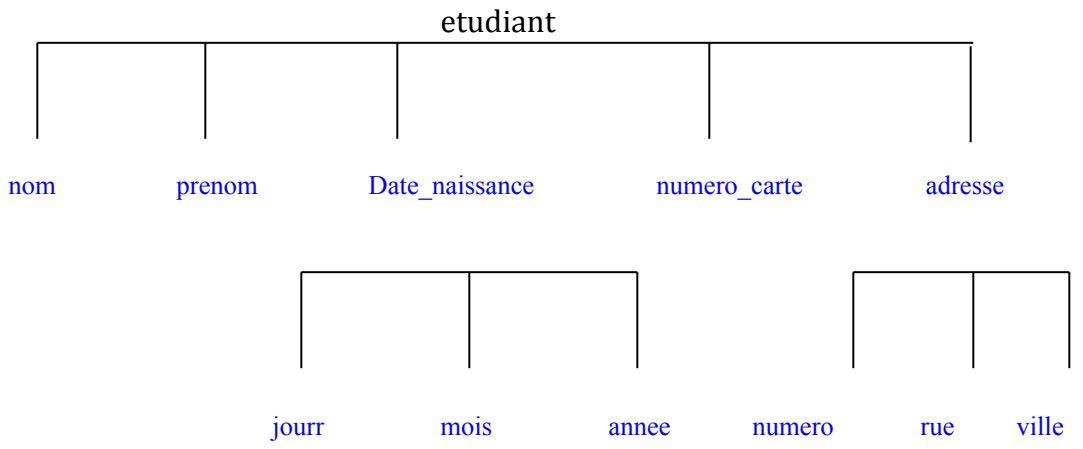
Structures Imbriquées

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

Struct  adresse {
      int  numero;
      char rue;
      char ville;
      };


Struct Etdiant {
     char nom ;
     char prenom[50];
     struct date Date_naissance;
     int num_carte ;
     struct adresse Adresse_etudiant;

      };
```



2.3 Tableaux de structures

- **Tableaux de structures**

```
struct compte client[200];
```

- La portée du nom d'un membre est limité à la structure dans laquelle il est défini. On peut avoir des membres homonymes dans des structures distinctes.

```
struct s1 {
        float x;
        int y ;
        };

struct s2{
        char x;
        float y;
        };
```

Pas de confusion

3. Manipulation des structures

- Initialisation à la compilation

```
struct compte {
       int num_compte ;
       char etat ;
       char nom[50];
       float solde;
       struct date dernier_versement;
       };

struct compte c1 = {12345,'C', »NDIAYE »,5000.45,02,04,2008};
```

- Accès aux membres : opérateur . Syntaxe : variable.membre

```
1/  c1.solde = 3834.56;

2/  struct compte c[200];
    y=c[33].solde;

3/ c1.dernier_versement.jour = 15;
   c[12].dernier_versement.mois = 11;
```

Structures et pointeurs

- L'adresse de début d'une structure s'obtient à l'aide de l'opérateur &

```
typedef struct {
    int num_compte ;
    char nature ;
    char nom[50];
    float solde;
    struct date dernier_versement;
    } cpt ;
cpt c1 , * pc;
```

- c1 est de type cpt, pc est un pointeur sur une variable de type cpt

```
pc = &c1;
```

- Accès au membres à partir du pointeur

```
*pc.num_compte = ...
```

Incorrect . est plus prioritaire que *

```
(*pc).num_compte = ...
```

- Opérateur ->

```
pc->no-compte = ...
```

Structures et fonctions

- Les membres d'une structure peuvent être passés comme paramètres à des fonctions avec ou sans modification
- Ex1 (sans modification)

```
float ajoute_compte(float solde1, float somme1)
{
    solde1 = solde1+somme1;
    return (solde1);
}

void main () {
......
cpt c1;
c1.solde = 0.;
Ajoute_compte(c1.solde,1500.0);
printf("%f\n",c1.solde);  -> 0.00
c1.solde=ajoute_compte(c1.solde,1500.0);
printf("%f\n",c1.solde);  -> 1500.00
```

Structures et fonctions

- Ex2 (avec modification)

```
void ajoute_au_compte(float * solde1, float somme1)
{
   *solde1 = *solde1+somme1;
}

void main () {
......
cpt c1;
c1.solde = 0.;
ajoute_au_compte(&(c1.solde),1000.0);  /* ou
&c1.solde */
printf("%f\n",c1.solde);  -> 1000.000000
```

Structures et fonctions

- Un argument de fonction peut-être de type structure

```
float ajoute_compte(cpt c, float somme1) {
    return(c.solde+somme1);
}

void main () {
cpt c1;
c1.solde = ajoute_compte(c1,1500.0);
printf("%f\n",c1.solde);  -> 1500.00
```

- Ou pointeur sur structure

```
void ajoute_compte (cpt * c, float somme1) {
    c->solde = c->solde + somme1;
}

void main () {
cpt c1;
Ajoute_compte(&c1 ,1500.0);
printf("%f\n",c1.solde);  -> 1500.00
```

Structures et fonctions

- La valeur de retour d'une fonction peut être une structure

```
cpt ajoute_compte(cpt c, float somme1) {
   cpt c2;
   c2=c;
   c2.solde=c.solde+somme1;
   return(c2);
}

void main () {
......
cpt c1;
c1.solde = 0.;
c1=ajoute_compte(c1,1500.0);  /* ou &c1.solde */
printf("%f\n",c1.solde);  -> 1500.0
```