XI. Les variables structurées

1. Rappel sur les tableaux
2. Les chaînes de caractères
3. Manipulation des chaînes
4. Les structures
5. Les unions

1. Rappel sur les tableaux

- suite d'éléments de même type
- taille prédéfinie
- éléments consécutifs dans la mémoire
- t $\simeq$ &(t[0])
- affectation : t1 = t2 interdit , => boucle de recopie
- en paramètre d'une fct : on communique l'adresse d'un tableau

```
f(int t[], int nbElements)
{ corps de la fct }

int tab[100];
f(tab, 100);
```

2. Les chaînes de caractères

Une chaîne de caractère :

- suite de caractères
- longueur variable mais bornée
- stockée dans un tableau (1 caractère/case)

Les constantes : "coucou"

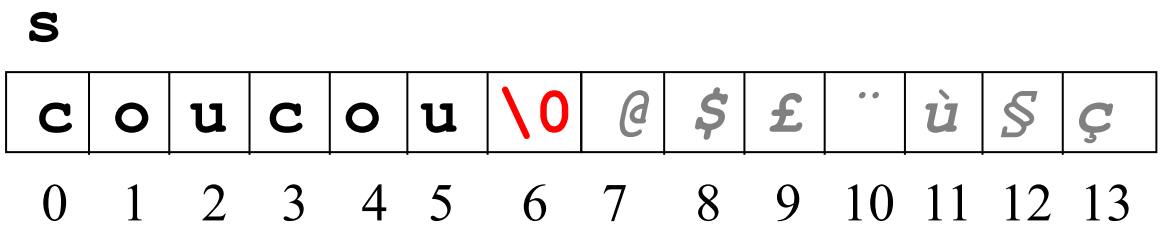
"Bonjour à tous !"

sont stockées à la compilation dans une suite d'octets consécutifs de la mémoire centrale (tableau de char sans identificateur)

variable *chaîne de caractères*

```
char s[14] ; /*pour 1 chaîne d'au plus 13 caractères*/
```

- chaîne stockée au début du tableau (de 0 à …)
- terminée par un caractère spécial `'\0'`
- accès au caractère de rang i : `s[i]`



Propriétés d'une chaîne : les mêmes que pour un tableau

```
char s[14] ;
```

- son identificateur représente son adresse,
- l'affectation globale n'est pas permise
- passée par adresse en argument d'une fonction,
- l'accès à un caractère se fait par son indice : `s[i]`
- **ATTENTION** : il faut s'assurer que ce caractère n'est pas au delà de `'\0'`

➔ fonctions pour manipuler les chaînes

3. Manipulation des chaînes

```
char chaine[80] ;
```

Entrées/sorties :

```
printf("%s", chaine) ;
```

- écriture de la suite de caractères non compris '\0'
- le curseur reste sur la ligne

```
puts(chaine) ;
```

- le curseur passe à la ligne suivante

`int scanf("%s", chaine)`

- saute les premiers séparateurs
- lit un « mot » (suite de caractères non-séparateurs)
- s'arrête au 1er séparateur sans le lire
- ajoute '**\0**' en fin de chaîne
- pas de contrôle sur la longueur de la chaîne

`char *fgets(chaine, Max, stdin)`

- lecture d'une ligne (jusqu'à '**\n**')
- stoppe la lecture si plus de *Max* caractères
- ajoute '**\0**' en fin de chaîne après '**\n**'
- retourne l'adresse de la chaîne ou **NULL**

La bibliothèque **`string.h`** :

```
#include <string.h>
```

`char *strcpy(char *a, const char *b)`

copie la chaîne b dans la chaîne a, rend a

```
strcpy(S, "Bonjour ")
```

`char *strcat(char *a, const char *b)`

copie la chaîne b à la suite de la chaîne a, rend a

`int strcmp(const char *a, const char *b)`

compare a et b et retourne une valeur négative, nulle ou positive selon que a < = > b

`int strlen(const char *a)`

rend le nombre de caractères de la chaîne a

Pourquoi **`fgets, strcpy`** *et* **`strcat`** *sont en* `char*` *?*

Ces fonctions retournent l'adresse de la chaîne résultat (1er paramètre)

➔ autorise une notation fonctionnelle

Exemple :

```
char S[60],N[50];
strcat(strcpy(S,"Bonjour "),fgets(N,50,stdin));
```

Complément syntaxe :

```
char *strcpy(char *a, const char *b)
```

- le mot **`const`** garantit que la variable pointée ne sera pas modifiée
- permet de donner une constante (ex : *"Bonjour"*) en paramètre

Les erreurs à éviter :

Soient deux chaînes `char s1[20], s2[20] ;`

affectation :

~~`s1 = s2`~~ `strcpy(s1, s2)`

comparaison :

~~`s1 < s2`~~ `strcmp(s1, s2) < 0`

~~`s1 == s2`~~ `strcmp(s1, s2) == 0`

4. Les structures

Rappel sur les tableaux :

- Suite d'éléments de **même type**
- Élément accessible via un indice

Les structures :

- Suite d'éléments de **types variés**
- de faible cardinal
- éléments accessibles via un ~~indice~~ identificateur : un ***champ***

4.1 Définition d'une structure :

struct *identificateur*$_{opt}$
{*type champ* **[;** *type champ***]}**
[*variable* **[,** *variable***]];**

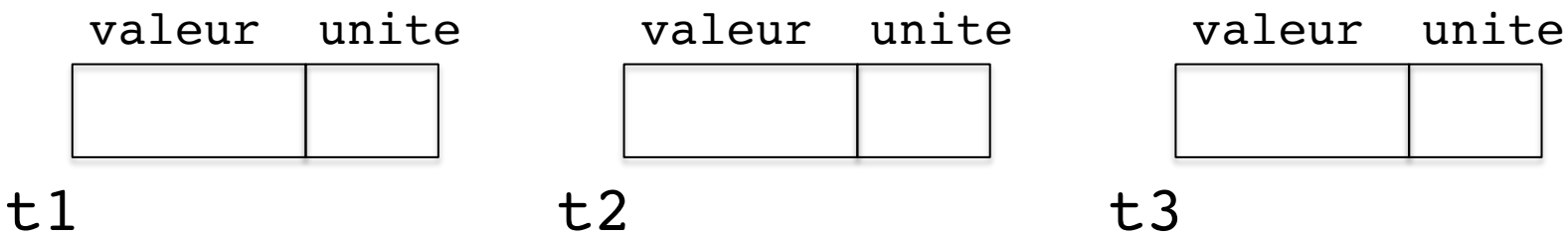
- Suite de champs de type quelconque
- **struct** *identificateur* se comporte ensuite comme un type

```
struct temperature
{ float valeur ;
  char unite ;  /* 'c' ou 'f' */
} t1, t2 ;

struct temperature t3 ;
```

- Utilisation de **typedef** fortement conseillée

- Utilisation de **`typedef`** fortement conseillée

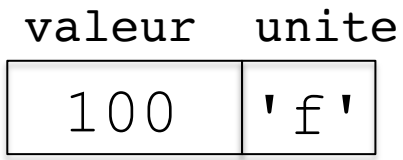
```
typedef struct temp     <- identificateur facultatif
{ float valeur ;
  char unite ;  /* 'c' ou 'f' */
} Temperature ;

Temperature t1, t2, t3 ;
```



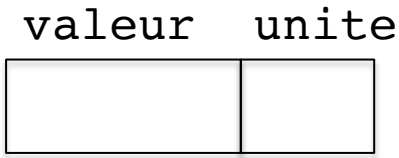
4.2 Accès à un champ d'une variable :

- se fait avec l'opérateur **.**
- syntaxe : *variable.champ*
- Possède toutes les propriétés liées au type du champ

```
Temperature t1, t2 ;
t1.valeur = 100 ;
t1.unite = 'f' ;
printf("température et unité c/f ?\n");
scanf("%f %c", &(t2.valeur), &(t2.unite)) ;
if (t2.valeur >= 38 && t2.unite == 'c')
   printf("Vous êtes malade ! ");
```



t1



t2

Exemple pour représenter :

- une adresse (n°, rue, codePostal, ville)
- un étudiant (nom, adresse, tableau de 10 notes)
- Une promo d'étudiants

Conseil :

- définir en global des types

```
typedef struct adresse { … } typeAdresse ;
typedef struct etudiant { … } typeEtudiant ;

typeAdresse a, b, c;
typeEtudiant e, promo[100];
```

Exemple pour représenter :

- une adresse (n°, rue, codePostal, ville)

```
typedef struct
{ short num ;
  char rue[50] ;
  long codePost ;
  char ville[50] ;
} Adresse ;
```

- un étudiant (nom, adresse, tableau de 10 notes)

```
typedef struct
{ char nom[40];
  Adresse adr ;
  float note[10] ;
} Etudiant ;
```

- Une promo d'étudiants

```
Etudiant promo2023[60];  // tableau de structures
```

Exemple d'utilisation d'un champ :

```
e.note[0] = 20;
scanf("%s", e.nom);
e.adr.codePostal = 13001 ;

promo[12].note[0] = 18;
strcpy(promo[12].adr.ville,"Marseille");
```

4.3 Pointeur sur une structure :

```
Etudiant e, *p=NULL;
p = &e;
```

Accéder à un champ de `e` via `p` : **`(*`***`p`**`).`***`champ`***

L'opérateur `->`

`(*`*`pointeur`**`).`***`champ`*** peut s'écrire aussi :

`pointeur` **`->`** ***`champ`***

Exemple :

```
p->note[0] = 20 ;
p->adr.codePostal = 13001 ;
p = &promo2023[3] ;
p->adr.codePostal = 13013 ;
```

4.5 Remarques :

```
Etudiant e1, e2;
```

Les structures supportent des manipulations "globales"

(contrairement aux tableaux !!!)

- affectation : `e1 = e2 ;` *(recopie de tous les champs)*
- passage par **valeur** en paramètre des fcts
  (recopie de toute la structure)
- peut-être le résultat d'une fct (renvoi de toute la structure)
  `return e1 ;`

Exemple avec la gestion d'une promotion d'étudiants :

```
typedef struct etudiant
{ int numero;
  char nom[50];
  float note[5];
} Etudiant;

Etudiant saisirEtudiant(void);
void afficherEtudiant(Etudiant e);
void saisirNotes(Etudiant *e) ;

 int main(void); /* menu de gestion */
```

```
void main(void);
{ Etudiant promo[200];
  int nbEtudiants, n, i ;
  nbEtudiants = 0;
  do
  { /* affichage du menu : */
    printf("0 : quitter l'application \n");
    printf("1 : créer une fiche étudiant  \n");
    printf("2 : affichage de la promo \n");
    printf("3 : saisie des notes d'un étudiant \n");
    /* sélection : */
    scanf("%d", &n);
    switch(n)
    { case 1 : promo[nbEtudiants++] = saisirEtudiant();
               break;
      case 2 : for (i=0 ; i<nbEtudiants ; i++)
                 afficherEtudiant(promo[i]);
               break;
      case 3 : printf("donnez le n° étudiant\n");
               scanf("%d", &i);
               saisirNotes(&promo[i]);
               break;
    }
  }
  while (n != 0); // while(n) ;
}
```

5. Les unions

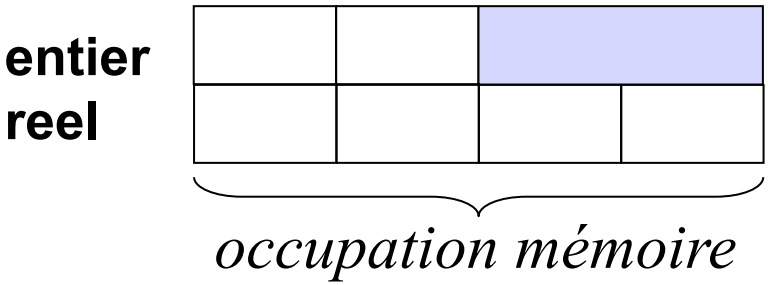
`struct` :

- Champs consécutifs *(contient plusieurs valeurs)*

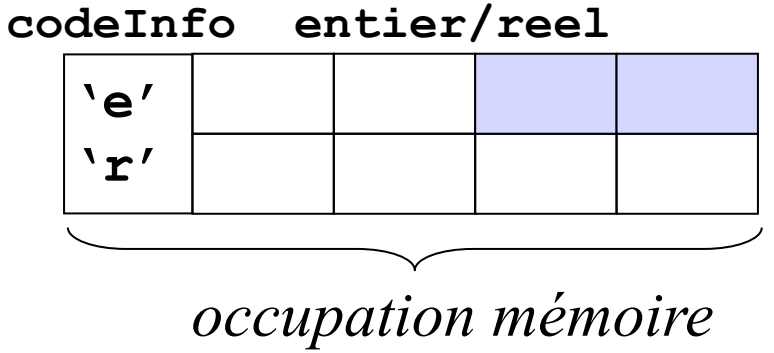
`union` :

- champs alternatifs *(contient une seule valeur à un instant t)*
- même zone mémoire pour coder différents types
- il faut savoir ce qu'on a mis !

```
typedef union boite
{  short entier;
   float reel;
} typeBoite;
typeBoite b;

b.entier = 1;
printf("%f\n", b.reel); /* n'écrit pas 1. !*/
```



```
typedef struct
{ char codeInfo; /* 'e' ou 'r' */
  union /*sans nom */
   { short entier;
     float reel;
   } val;
} boiteBis;

boiteBis b;
```

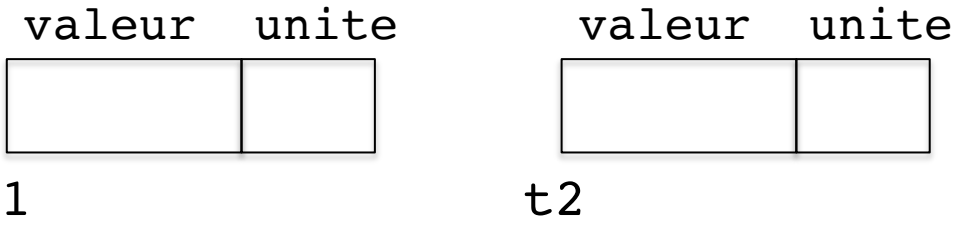


On utilise un champ pour identifier le type du champ **union**

```
b.val.entier = 1;
b.codeInfo = 'e';
...
if (b.codeInfo == 'e')
  printf("%d\n", b.val.entier);
else
  printf("%f\n", b.val.reel);
```

Résumé sur les structures

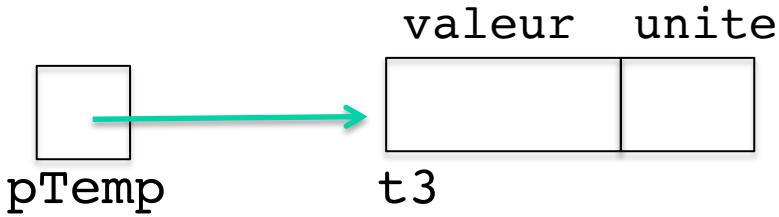
```
typedef struct temp    <- identificateur facultatif
{ float valeur ;
  char unite ;  /* 'c' ou 'f' */
} Temperature ;

Temperature t1, t2, t3, *pTemp ;
```



- Accès à un champ :

```
t1.valeur = 0.;  t1.unite = 'c';
```

- Affectation : `t2 = t1 ;`



- Pointeur sur une structure :

```
pTemp = &t3 ;
pTemp->valeur = 32.; pTemp->unite = 'f';
t2 = *pTemp ;
```