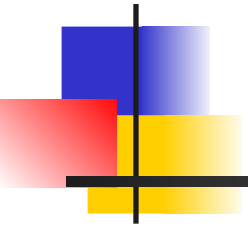


Les variables dimensionnées - Les tableaux

Pourquoi les tableaux ?

- Exemple
  - on souhaite calculer la moyenne de tous les étudiants d'une classe tout en sauvegardant les notes en mémoire.
- Comment faire ?
  - 1ère solution : Déclarer autant de variables qu'il y a d'étudiants
    - Exemple pour 120 étudiants !

```
int main(void)
{
    float note1,note2,…,note120 ;    /* 120 variables ! */

…
}
```

- Conclusion : La solution proposée n'est pas performante

Pourquoi les tableaux ?

- Idée
  - Il faudrait pouvoir stocker dans une seule variable plusieurs valeurs du même type !
  - C'est le rôle des tableaux
- Syntaxe de la déclaration
  - `type Nom_Tableau[nombre de valeurs];`
- Exemple

```
...
int    MonTab[10];
...
```

  - Tableau `MonTab` de type **`int`** permettant de stocker 10 valeurs de type **`int`**.

Structure d'un tableau (1dim)

- Exemple

```
int main(void)
{
    int tab[5];
}
```

- Le programme déclare un tableau de 5 éléments de type **int**
- Chaque élément du tableau est repéré par un indice
- Par définition, le premier élément est nommé `tab[0]`

tab [0]	tab [1]	tab [2]	tab [3]	tab [4]

Accès à un élément du tableau

- Chaque élément du tableau peut contenir une valeur .
- Pour accéder à un élément du tableau , il suffit de fournir le nom de cet élément
- Exemple

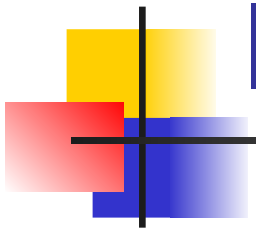
```
int main(void)
{
    short tab[5];
    tab[0]=3;
    tab[1]=14;
    …
}
```

Initialisation

- Lors de la déclaration du tableau
  - `int tab[10] = { 1, 2, 5, 6, 3, 9, 124, 3, 6, 9 };`
    - tous les éléments sont initialisés
  - `int tab[] = { 1, 2, 5, 6, 3, 9, 124, 3, 6, 9 };`
    - tous les éléments sont initialisés
    - le compilateur comprend que la dimension du tableau est 10
  - `int tableau[10] = {1,2};`
    - Seuls les 2 premiers éléments du tableau sont initialisés
  - `int tableau[10] = {0};`
    - Tous les éléments du tableau sont initialisés à 0
- Par affectation d'une valeur à chacun de ses éléments
  - `tab[0] = 1;`
  - `tab[1] = 2;`
  - `tab[2] = 5;`
  - etc

Exemple 1

```
int main(void)
{
    float TabNotes[4]={12,9,15,17};
    float moyenne;
    moyenne = (TabNotes[0] + TabNotes[1]
            +  TabNotes[2] + TabNotes[3])/4;
}
```



Initialisation

- Attention !

`float` `tab[];` est impossible

  - Le compilateur ne peut pas déterminer seul la dimension du tableau.

Les chaînes de caractères

- Une chaîne de caractères n'est rien d'autre qu'un tableau de caractère se terminant par le caractère de code ASCII 0 (NULL = '\0')
- On peut accéder à un caractère particulier de la chaîne de la même manière que pour les tableaux

- Initialisation d'une chaîne de caractères
  - `char message[6]= {'s','a','l','u','t','\0'};`
  - `char message[] = {'s','a','l','u','t','\0'};`

  - `char message[6] = "salut";`
  - `char message[] = "salut";`

Dans ces cas, le compilateur ajoute lui-même le caractère de fin de chaîne '\0'

Exemple 2 version 1

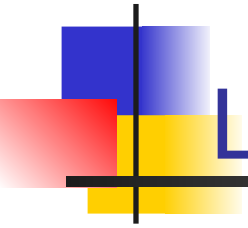
```
int main(void)
{
   char message[6] = {'s','a','l','u','t','\0'};

   message[0] = message[0] - 0x20;
   message[1] = message[1] - 0x20;
   message[2] = message[2] - 0x20;
   message[3] = message[3] - 0x20;
   message[4] = message[4] - 0x20;
}
```

Exemple 2 version 2

```
int main(void)
{
   char message[]=   "salut";
   message[0] = message[0] - 0x20;
   message[1] = message[1] - 0x20;
   message[2] = message[2] - 0x20;
   message[3] = message[3] - 0x20;
   message[4] = message[4] - 0x20;
}
```

Exemple 2 version 3

```
int main(void)
{
   int i;

   char message[ ]="salut";
   i = 0;
   message[i] = message[i] - 0x20;
   i++;
   message[i] = message[i] - 0x20;
   i++;
   message[i] = message[i] - 0x20;
   i++;
   message[i] = message[i] - 0x20;
   i++;
   message[i] = message[i] - 0x20;
}
```

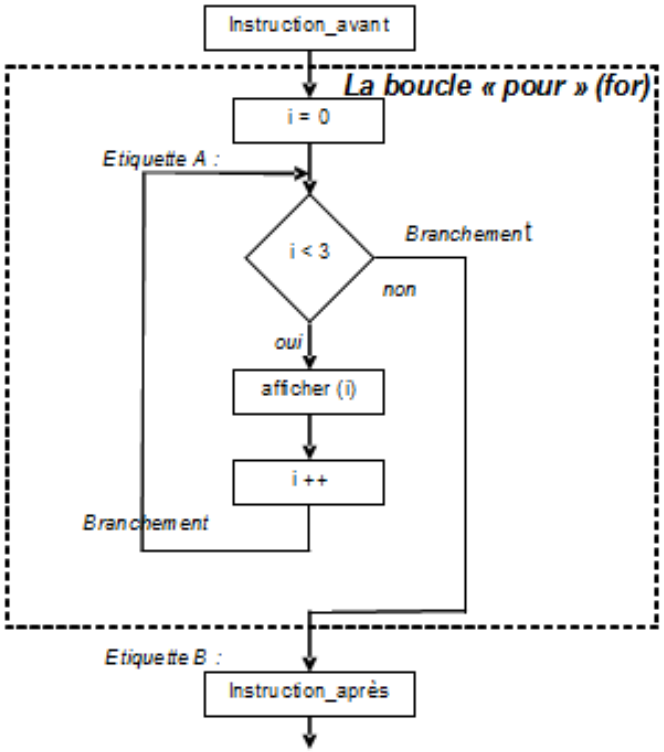


Les instructions de contrôle - boucles déterministes

L'instruction for

Syntaxe:

```
for(expr1; expr2; expr3)
 {
   bloc d'instructions;
 }
```

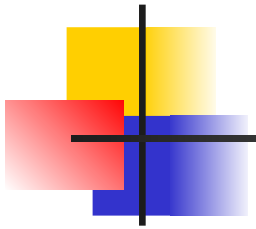


- Principe
  - Exécuter le bloc d'instructions en fonction d'un nombre d'itérations
- Fonctionnement
  - Evaluation de l'expression1
  - Evaluation de l'expression2
    - Si expression2 est VRAI alors
      - Exécution des instructions du bloc
      - Evaluation de l'expression 3
    - Si expression2 est FAUX alors
      - sortir de la boucle

Exemple

```
int main(void)
{
   char message[6] = {'s','a','l','u','t','\0'};
   int i;
   for(i=0 ;  i<5   ; i++)
   {
      message[i]=message[i]-0x20;
   }
}
```

Attention:
Pas de point virgule



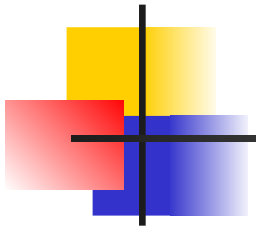
- Fonctionnement pas à pas
  - Déclaration du tableau message

```
int main(void)
{
   char message[6]={'s','a','l','u','t','\0'};
   int i;
   for(i=0 ; i<5 ; i++) {
       message[i]=message[i]-0x20;
   }
}
```

Avant l'exécution de l'instruction

Après l'exécution de l'instruction

i=?



- Fonctionnement pas à pas
  - Déclaration du tableau message
  - Déclaration de la variable i

```
int main(void)
{
   char message[6]={'s','a','l','u','t','\0'};
   int i;
   for(i=0 ; i<5 ; i++) {
       message[i]=message[i]-0x20;
   }
}
```

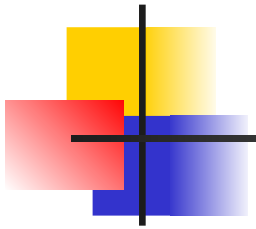
Avant l'exécution de l'instruction

Après l'exécution de l'instruction

i=?

En mémoire

's'	'a'	'l'	'u'	't'	'\0'				



- Fonctionnement pas à pas
  - Exécution de la boucle
    - Evaluation de l'expression1

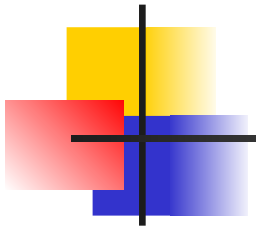
```
int main(void)
{
   char message[6]={'s','a','l','u','t','\0'};
   int i;
   for(i=0 ; i<5 ; i++) {
       message[i]=message[i]-0x20;
   }
}
```

Avant l'exécution de l'instruction : i=?

Après l'exécution de l'instruction : i=0

En mémoire

's'	'a'	'l'	'u'	't'	'\0'				



- Fonctionnement pas à pas
  - Exécution de la boucle
    - Evaluation de l'expression2

      (i<5) ? VRAI
    - Exécution des instructions du bloc

```
int main(void)
{
   char message[6]={'s','a','l','u','t','\0'};
   int i;
   for(i=0 ; i<5 ; i++) {
       message[i]=message[i]-0x20;
   }
}
```

Avant l'exécution de l'instruction

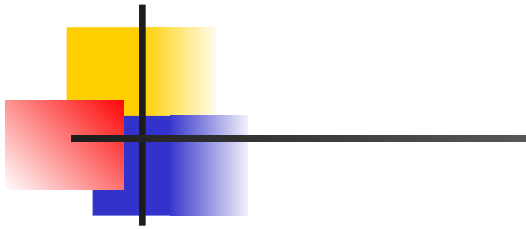
i=0

Après l'exécution de l'instruction

i=0

En mémoire

's'	'a'	'l'	'u'	't'	'\0'				



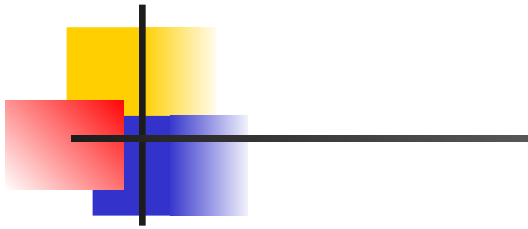
- Fonctionnement pas à pas
  - Conversion 1er élément de la chaîne : minuscule ⇒ Majuscule

```
int main(void)
{
   char message[6]={'s','a','l','u','t','\0'};
   int i;
   for(i=0 ; i<5 ; i++) {
       message[i]=message[i]-0x20;
   }
}
```

Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
i=0	i=0

En mémoire

's'	'a'	'l'	'u'	't'	'\0'				



- Fonctionnement pas à pas
  - Evaluation de l'expression3
  - Incrémentation de i

```
int main(void)
{
   char message[6]={'s','a','l','u','t','\0'};
   int i;
   for(i=0 ; i<5 ; i++) {
       message[i]=message[i]-0x20;
   }
}
```

Avant l'exécution de l'instruction

i=0

Après l'exécution de l'instruction

i=1

En mémoire

'S'	'a'	'l'	'u'	't'	'\0'				

- Fonctionnement pas à pas
  - Exécution de la boucle
    - Evaluation de l'expression2
    - (i<5) ? VRAI
    - Exécution des instructions du bloc

```
int main(void)
{
   char message[6]={'s','a','l','u','t','\0'};
   int i;
   for(i=0 ; i<5 ; i++) {
       message[i]=message[i]-0x20;
   }
}
```

Avant l'exécution de l'instruction : i=1

Après l'exécution de l'instruction : i=1

En mémoire

'S'	'a'	'l'	'u'	't'	'\0'				

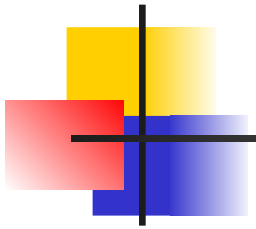
- Fonctionnement pas à pas
  - conversion 2eme élément de la chaîne : minuscule ⇒ Majuscule

```
int main(void)
{
   char message[6]={'s','a','l','u','t','\0'};
   int i;
   for(i=0 ; i<5 ; i++) {
       message[i]=message[i]-0x20;
   }
}
```

Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
i=1	i=1

En mémoire

'S'	'A'	'l'	'u'	't'	'\0'				



- Fonctionnement pas à pas
  - Evaluation de l'expression3
  - Incrémentation de i

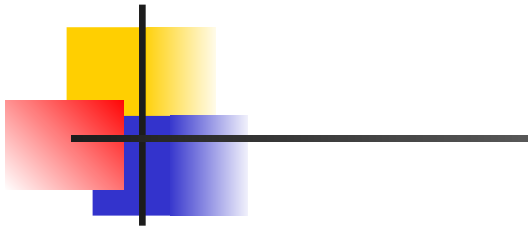
```
int main(void)
{
   char message[6]={'s','a','l','u','t','\0'};
   int i;
   for(i=0 ; i<5 ; i++) {
       message[i]=message[i]-0x20;
   }
}
```

Avant l'exécution de l'instruction : i=1

Après l'exécution de l'instruction : i=2

En mémoire

'S'	'A'	'l'	'u'	't'	'\0'				



- Fonctionnement pas à pas
  - Exécution de la boucle
    - Evaluation de l'expression2
    - (i<5) ? VRAI
    - Exécution des instructions du bloc

```
int main(void)
{
   char message[6]={'s','a','l','u','t','\0'};
   int i;
   for(i=0 ; i<5 ; i++) {
       message[i]=message[i]-0x20;
   }
}
```

Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
i=2	i=2

En mémoire

'S'	'A'	'l'	'u'	't'	'\0'				

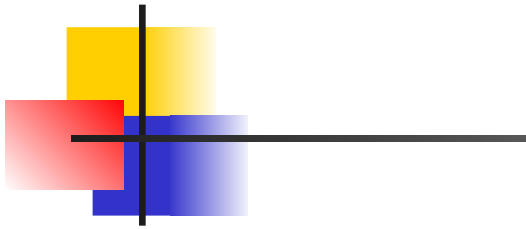
- Fonctionnement pas à pas
  - conversion 3eme élément de la chaîne : minuscule ⇒ Majuscule

```
int main(void)
{
   char message[6]={'s','a','l','u','t','\0'};
   int i;
   for(i=0 ; i<5 ; i++) {
       message[i]=message[i]-0x20;
   }
}
```

Avant l'exécution de l'instruction	*Après l'exécution de l'instruction*
i=2	i=2

En mémoire

'S'	'A'	'L'	'u'	't'	'\0'				



- Fonctionnement pas à pas
  - Evaluation de l'expression3
  - Incrémentation de i

```
int main(void)
{
   char message[6]={'s','a','l','u','t','\0'};
   int i;
   for(i=0 ; i<5 ; i++) {
       message[i]=message[i]-0x20;
   }
}
```

Avant l'exécution de l'instruction : i=2

Après l'exécution de l'instruction : i=3

En mémoire

'S'	'A'	'L'	'u'	't'	'\0'				

Au final...

- Fonctionnement pas à pas
  - Exécution de la boucle
    - Evaluation de l'expression2
    - (i<5) ? FAUX
    - sortir de la boucle

```
int main(void)
{
   char message[6]={'s','a','l','u','t','\0'};
   int i;
   for(i=0 ; i<5 ; i++) {
       message[i]=message[i]-0x20;
   }
}
```

Avant l'exécution de l'instruction

i=5

Après l'exécution de l'instruction

i=5

En mémoire

'S'	'A'	'L'	'U'	'T'	'\0'				

Exemple 1

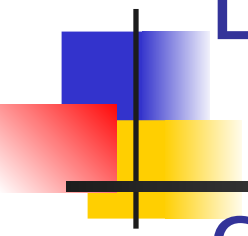
```
int main(void)
{
   float TabNotes[4]={12,9,15,17};
   float moyenne, somme =0;
   int i;
   for(i=0;i<4;i++)
   {
      somme = somme + TabNotes[i];
   }
   moyenne =    somme/4;
}
```

Que fait le programme suivant ?

```
int main(void)
{
     int i;
     int Tableau[10];

     for(i=0; i<10; i++)
     {
          Tableau[i]=i+2;
     }
}
```

Réponse

2	3	4	5	6	7	8	9	10	11



Les instructions de contrôle -les instructions conditionnelles

Les instructions conditionnelles

- Jusqu'à présent, on a uniquement considéré des instructions s'exécutant successivement.

```
int main(void)
{
    int a=2, b=5, somme, produit;
    somme=a+b;
    produit=a*b;
}
...
```

Exécution successive une instruction après l'autre

- Dans un programme, il se peut que l'on veuille conditionner l'exécution de certaines instructions
  - Si une condition est vraie alors exécuter des instructions précises
- Il existe plusieurs types d'instructions conditionnelles
  - `if`
  - `If … else`
  - ….

L’instruction if

Syntaxe en C:

```
if (expression)
{
   …
}
…
```

- Principe
  - Si l'expression présente entre les parenthèses est vraie alors exécuter les instructions du bloc
  - Si l'expression présente entre les parenthèses est fausse, sauter le bloc et continuer l'exécution du programme

Remarque:
Pas de ; dans le if

Exemple 1

```
#include<stdio.h>
int main(void)
{
     int a;
     ……
     if (a>8)
     {
          ……
     }
     ……
}
```

Exemple 2

```
int main(void)
{
    float TabNotes[4]={12,9,15,17};
    float Note_Min;
    int i;
    Note_Min = TabNotes[0];

    for(i=0;i<4;i++)
    {
        if(TabNotes[i]  <  Note_Min)
        {
            Note_Min=TabNotes[i];
        }
    }
}
```

Possibilité d'imbrication

- Il est possible qu'un bloc contienne également d'autres types d'instructions conditionnelles
- Exemple

```
int a;
…..
if(a>0)
{
   if(a>=10 && a<=20)
   {
        ……
   }
    ……
}
……
```

L'instruction if...else

Syntaxe en C:

```
if (expression)
{
    bloc1
}
else
{
    bloc2
}
```

- Principe
  - Si l'expression présente entre les parenthèses est vraie alors
    - exécuter les instructions du bloc1
  - Sinon
    - exécuter les instructions du bloc 2

Exemple 3

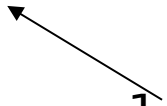
```
#include <stdio.h>

int main(void)
{
     int a;
     ……
     if (a<0)
     {
          ……
     }
     else
     {
          ……
     }
}
```

Remarque

- Lorsqu'un bloc contient uniquement 1 seule instruction, il est possible de supprimer les accolades
- Exemple

```
if (a<0)
{
   ……..
}
else
{
   …….
}
```

```
if (a<0)
   …….
else
   …….
```



1 seule instruction dans le bloc

Les tableaux de caractères

- Un caractère est de type **char**
- Exemple d'un tableau de caractères

```
int main(void)
{
   char tabCar[10];
   tabCar[0] = 's';
   tabCar[1] = 'a';
   tabcar[2] = 'l';
}
```