

# Programmation en langage C

# 1. Les tableaux

## Tableaux à une dimension

- Déclaration d'un tableau à une dimension:

```
type tab[dim];
```

- **type**: indiquer le type de données que va contenir le tableau: **int**, **float**, **char**, *etc.*
- **dim** nombre de cases que va contenir le tableau

```
int myArray[12];
```

- Dans la version C89, **dim** doit être une valeur constante **const**
- Dans la version C99, **dim** peut être une simple variable

# 1. Les tableaux

## Tableaux à une dimension

- Accéder à une case d'un tableau en utilisant le nom du tableau suivi de deux crochets "[ ... ]"
- L'indice de la première case d'un tableau est 0
  - `tab[0]` permet d'accéder à la première valeur du tableau `tab`
- L'indice de la dernière case est **dim-1**
  - `tab[dim-1]` permet d'accéder à la dernière case
- Chaque case d'un tableau peut être manipulée comme une simple variable.

```
int i = myArray[0];  
int j = myArray[n-1];  
myArray[3] = 8;
```

# 1. Les tableaux

## Tableaux à une dimension

```
#include <stdio.h>

int main()
{
    int tab[5];
    int i, min;
    for(i = 0; i < 5; i++) {
        printf("Value %d: ", i+1);
        scanf("%d", &tab[i]);
    }
    min = tab[0];
    for(i = 1; i < 5; i++) {
        if(tab[i] < min) min = tab[i];
    }
    printf("min = %d\n", min);
    return 0;
}
```

# 1. Les tableaux

## Tableaux à une dimension

- Un tableau peut être initialisé au moment de sa déclaration

```
int myArray[5] = {1, 3, 5, 7, 11};
```

- Il est possible de déclarer un tableau et l'initialiser sans préciser sa taille. La taille peut être déduite.

```
int myArray[] = {1, 3, 5, 7, 11};
```

# 1. Les tableaux

## Tableaux à une dimension

- Si à l'initialisation d'un tableau, uniquement quelques cases sont initialisées, le reste des cases sera par défaut initialisé à 0

```
int myArray[10] = {1, 3, 5, 7, 11};
```

1	3	5	7	11	0	0	0	0	0
---	---	---	---	----	---	---	---	---	---

# 1. Les tableaux

## Tableaux à une dimension

```
#include <stdio.h>

int main()
{
    int myArray[5] = {1, -3, 5};
    for(int i = 0; i < 7; i++) {
        printf("myArray[%d] = %d\n", i, myArray[i]);
    }
    return 0;
}
```

```
myArray[0] = 1
myArray[1] = -3
myArray[2] = 5
myArray[3] = 0
myArray[4] = 0
myArray[5] = 5
myArray[6] = 6422284
```

# 1. Les tableaux

## Tableaux à deux dimensions

- Déclaration:

```
type nom_tab[dim1][dim2];
```

- **type** : **int**, **float**, **char**, *etc.*
- **dim1** nombre de lignes
- **dim2** nombre de colonnes

```
int myArray[8][9];
```

- Accéder à la  $i^{\text{ième}}$  ligne et  $j^{\text{ième}}$  colonne

```
myArray[i-1][j-1]
```



# 1. Les tableaux

## Tableaux à deux dimensions

```
#include <stdio.h>

int main()
{
    int a, i, j;
    int mat[3][3];
    printf("a = ");
    scanf("%d", &a);
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            printf("mat[%d][%d] = ", i, j);
            scanf("%d", &mat[i][j]);
            mat[i][j] *= a;
        }
    }
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            printf("%d\t", mat[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

# 1. Les tableaux

## Tableaux à deux dimensions

- Initialisation d'un tableau à deux dimensions:

```
int mat[3][3] = {{1, 2, 3}, {1, 2, 3}, {1, 2, 3}};
```

- Si un tableau est initialisé au moment de sa déclaration, Il est possible de préciser la taille d'une seule dimension, la deuxième dimension peut être déduite.

```
int mat[][3] = {{1, 2, 3}, {1, 2, 3}, {1, 2, 3}};
```

```
int mat[3][3] = {{1, 2, 3}, {2, 3}};
```

# 1. Les tableaux

## Tableaux à deux dimensions:

```
#include <stdio.h>

int main()
{
    int mat[3][3] = {{1, 2, 3}, {1, 2, 3}, {1, 2, 3}};
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            printf("%d\t", mat[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

1	2	3
1	2	3
1	2	3

# 1. Les tableaux

## Les tableaux et à deux dimensions:

```
#include <stdio.h>

int main()
{
    int mat[3][3] = {{1, 2, 3}, {2, 3}};
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            printf("%d\t", mat[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

1	2	3
2	3	0
0	0	0

# 1. Tableaux

## Les tableaux à deux dimensions

```
#include <stdio.h>

int main()
{
    int mat[][3] = {{1, 2, 3}, {2, 3}};
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            printf("%d\t", mat[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

1	2	3
2	3	0
0	2	6422284

Les chaines de caractères

String

## 2. Strings

- En langage C, le type String n'existe pas.
- Les chaînes de caractères (String) sont représentées en C par un tableau à une dimension de caractères
- Un tableau de caractères se termine toujours par le symbole “\0” qui marque la fin de la chaîne de caractères.

## 2. Strings

Initialisation d'un tableau de caractères:

```
char myString[] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

```
char myString[] = "Hello";
```



## 2. Strings

- Un tableau de caractère est manipulé (ajout, modification, ...) de la même manière que les tableaux des autres types.

```
#include <stdio.h>

int main()
{
    char myString[] = "Hello";
    myString[1] = 'a';
    printf("%c", myString[1]);
    return 0;
}
```

a

## 2. Strings

- L'affectation d'une chaîne de caractères à un tableau de caractère en utilisant un seul "=" n'est correcte qu'au moment de la déclaration du tableau.

```
#include <stdio.h>

int main()
{
    char myString[] = "Hello";
    myString = "Bye";
    return 0;
}
```

File	Line	Message
		=== Build: Debug in Useless (compiler: GNU GCC Compiler) ===
C:\Users\blab...		In function 'main':
C:\Users\blab...	6	error: assignment to expression with array type
C:\Users\blab...	5	warning: variable 'myString' set but not used [-Wunused-but-set-variable]
		=== Build failed: 1 error(s), 1 warning(s) (0 minute(s), 0 second(s)) ===

## 2. Strings

### Afficher un tableau de caractères:

Pour afficher un tableau de caractères, utiliser la fonction **printf** avec le symbole **%s**

```
#include <stdio.h>

int main()
{
    char myString[10] = "Test";
    printf("%s", myString);
    return 0;
}
```

Test

## 2. Strings

### Afficher un tableau de caractères:

- Il est aussi possible d'utiliser la fonction **puts** de la librairie **stdio.h**

```
#include <stdio.h>

int main()
{
    char myString[10] = "Test";
    puts(myString);
    return 0;
}
```

Test

## 2. Strings

### Lire un tableau de caractères:

- Utiliser la fonction **scanf** avec le format **%s**
- L'inconvention de cette fonction est qu'elle ne considère pas les espaces. Le **scanf** considère uniquement la chaîne de caractères avant l'espace.

```
#include <stdio.h>

int main()
{
    char myString[10];
    printf("String: ");
    scanf("%s", myString);
    printf("Result: %s\n", myString);
    return 0;
}
```

```
String: Benjamin
Result: Benjamin
```

```
String: Benj amin
Result: Benj
```

## 2. Strings

### Lire un tableau de caractères:

- Il est possible de préciser le nombre de caractères à prendre en consideration de la chaine de caractères saisi comme suit:

```
#include <stdio.h>

int main()
{
    char myString[10];
    printf("String: ");
    scanf("%3s", myString);
    printf("Result: %s\n", myString);
    return 0;
}
```

```
String: Benjamin
Result: Ben
```

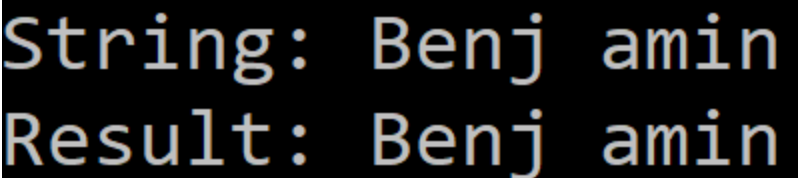
## 2. Strings

### Lire un tableau de caractères:

- Pour remédier à l'inconvénient de **scanf**, il est possible d'utiliser la fonction **gets** pour saisir une chaîne de caractères.
- L'opération de lecture s'arrête à l'insertion d'un retour à la ligne, les espaces sont donc considérés comme des caractères de la chaîne saisie.

```
#include <stdio.h>
```

```
int main()  
{  
    char myString[10];  
    printf("String: ");  
    gets(myString);  
    printf("Result: %s\n", myString);  
    return 0;  
}
```



String: Benj amin  
Result: Benj amin

## 2. Strings

### Opérations sur les chaines de caractères:

`#include <string.h>`

- **strlen**
  - Récupérer la longue de la chaine de caractères
  - Retourne un entier indiquant la taille de la chaine de caractères (le caractère est ignoré“\0”)

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

int main()
{
    char myString[10] = "Hello";
    printf("%d\n", strlen(myString));
    return 0;
}
```



## 2. Strings

### Opérations sur les chaines de caractères:

- **strcpy**
  - Permet de copier le contenu d'une chaine de caractère dans une autre ("**\0**" inclus)

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

int main()
{
    char myString1[10] = "Hello";
    char myString2[10];
    strcpy(myString2, myString1);
    printf("%s\n", myString2);
    return 0;
}
```

Hello

## 2. Strings

### Opérations sur les chaines de caractères:

- **strncpy**
  - Permet aussi de copier le contenu d'une chaine de caractères dans une autre, mais avec cette fonction, il est possible de préciser le nombre de caractères à

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

int main()
{
    char myString1[10] = "Hello";
    char myString2[10];
    strncpy(myString2, myString1, 4);
    myString2[4] = '\0';
    printf("%s\n", myString2);
    return 0;
}
```

Hell

## 2. Strings

### Opérations sur les chaines de caractères:

- **strcat**
  - Prend en parametres deux chaines de caractères et copie le contenu de la deuxième à la fin de la première chaine.

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

int main()
{
    char myString1[20] = "Hello ";
    char myString2[10] = "Benjamin";
    strcat(myString1, myString2);
    printf("%s\n", myString1);
    return 0;
}
```

Hello Benjamin

## 2. Strings

### Opérations sur les chaines de caractères:

- **strncat**
  - Meme fonctionnement que **strcat**, mais celle-ci prend un troisième parametre pour préciser le nombre de caractères à copier

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

int main()
{
    char myString1[20] = "Hello ";
    char myString2[10] = "Benjamin";
    strncat(myString1, myString2, 3);
    printf("%s\n", myString1);
    return 0;
}
```

Hello Ben

## 2. Strings

### Opérations sur les chaines de caractères:

- **strcmp**
  - Permet de comparer deux chaines de caractères, elle retourne une entiere positif si la 1ere chaine est plus logue, 0 si les deux chaines sont identiques et un entier negative si la 2eme est plus longue que la 1ere.

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

int main()
{
    char myString1[10] = "Hello";
    char myString2[10] = "Hell";
    printf("%d\n", strcmp(myString1, myString2));
    return 0;
}
```

## 2. Strings

### Opérations sur les chaines de caractères:

- **strncmp**
  - Elle a le meme principe que la fonction **strcmp**, mais celle-ci prend un 3eme parametre précisant le nombre de caractères à comparer.

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

int main()
{
    char myString1[10] = "Hello";
    char myString2[10] = "Hell";
    printf("%d\n", strncmp(myString1, myString2, 4));
    return 0;
}
```