ECE PARIS • LYON
ÉCOLE D'INGÉNIEURS

Algorithmique et programmation structurée en C

Cours n°7

Langage C
Chaînes de caractères et Aléatoire

Antoine Hintzy

 Plan du cours

- Chaînes de caractères
- Génération de nombres aléatoire

Chaîne de caractères

0	1	2	3	4	5	6	7	8	9
‘E’	‘C’	‘E’	‘ ’	‘L’	‘y’	‘o’	‘n’	?	?

Une **chaîne de caractères** (🇬🇧 *String*) n'est ni plus ni moins qu'un **tableau de caractères** (`char`) :

```
char phrase[10] = "ECE Lyon";
```

Chaîne de caractères

0	1	2	3	4	5	6	7	8	9
‘E’	‘C’	‘E’	‘ ’	‘L’	‘y’	‘o’	‘n’	?	?

Dans cet exemple, nous stockons "**ECE Lyon**" (**8** caractères) dans un tableau pouvant contenir jusqu'à **10** caractères.

```
char phrase[10] = "ECE Lyon";
```

Chaîne de caractères - Déclaration

```
char nomDeLaChaine[TAILLE_PHYSIQUE];
```

avec **TAILLE_PHYSIQUE** un entier constant définissant le nombre maximum de caractères pouvant être stockés.

Chaîne de caractères - Affectation directe

Il est possible d'initialiser une chaîne de caractères lors de sa déclaration :

```
char phrase[100] = "ECE Lyon";
```

🚨 Mais il est impossible de faire une nouvelle affectation par la suite :

❌
```
phrase = "ECE Paris•Lyon";
```

Utiliser la fonction **`strcpy`** *de* **`string.h`** *pour réaffecter une nouvelle valeur (voir la fin du cours)*

Chaîne de caractères - Parcours et affichage

```c
#include <stdio.h>
#define MAX 10

int main(void) {
    int i = 0;
    // Tableau de 10 caractères initialisé à "Toto" (4 caractères)
    char firstname[MAX] = "Toto";
    int tailleLogique = 4;

    for(i = 0; i < tailleLogique; i++) {
        // On affiche un à un tous les caractères :
        printf("%c", firstname[i]);
    }
    return 0;
}
```

Chaîne de caractères - `\0`

En réalité, il n'est pas utile de stocker la taille logique d'une chaîne de caractères.

La partie logique dans une chaîne de caractères est toujours suivie du **caractère `\0`** (entier **`0`** dans la table ASCII) permettant d'**indiquer la fin**.

0	1	2	3	4	5	6	7	8	9
‘E’	‘C’	‘E’	‘ ’	‘L’	‘y’	‘o’	‘n’	‘\0’	?

Chaîne de caractères - \0

En initialisant une chaîne de caractères, un **\0** est automatiquement ajouté à la fin de sa partie logique, marquant la fin de sa partie logique.

```
char phrase[10] = "ECE Lyon";
```

0	1	2	3	4	5	6	7	8	9
‘E’	‘C’	‘E’	‘ ’	‘L’	‘y’	‘o’	‘n’	‘\0’	?

Chaîne de caractères - `\0`

Remarque

Si un tableau de caractères a pour taille physique **`MAX`**, alors il peut contenir une chaîne de caractères d'au plus **`MAX-1`** caractères.

Il faut garder une case pour le caractère **`\0`**.

Chaîne de caractères - Parcours et affichage avec \0

```c
#include <stdio.h>
#define MAX 10

int main(void) {
    int i = 0;
    // Tableau de 10 caractères initialisé à "Toto" (4 caractères)
    char firstname[MAX] = "Toto";

    for(i = 0; firstname[i] != '\0'; i++) {
        // On affiche un à un tous les caractères :
        printf("%c", firstname[i]);
    }
    return 0;
}
```

Chaîne de caractères - Affichage avec `printf` / `scanf`

Heureusement, nous ne sommes pas obligés de parcourir systématiquement les chaînes de caractères.

Dans **`printf`** et **`scanf`**, une chaîne de caractères est représentée par '**`%s`**' (*string*) :

```c
#include <stdio.h>
int main(void) {
  char firstname[101]; // cette chaine peut contenir un
                       // prénom de 100 caractères maximum (+ '\0').

  printf("Saisissez votre prénom (100 caractères max) :\n");
  scanf("%s", firstname); // firstname est déjà une adresse (tableau), donc pas d'&
  printf("%s\n", firstname); // %s utilise un tableau de caractères dans le printf

  return 0;
}
```

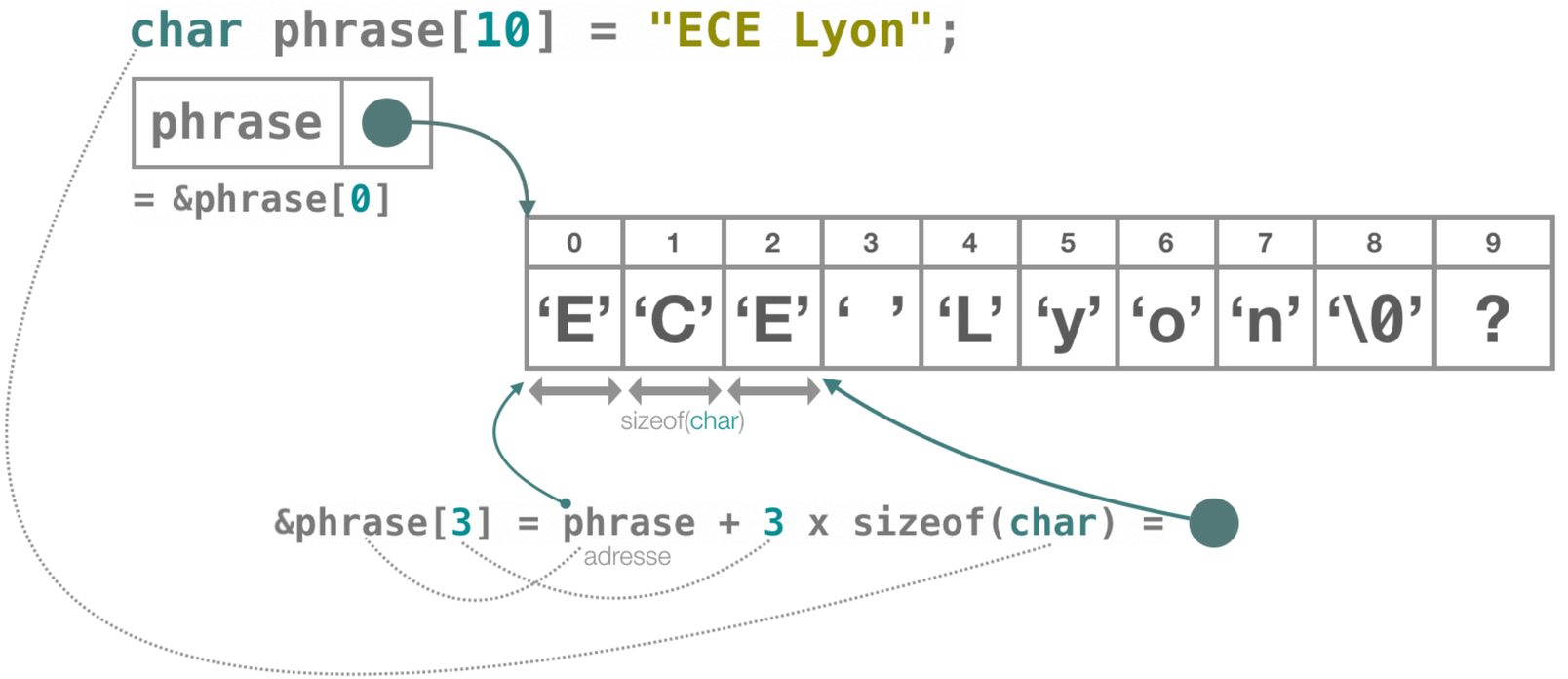
Chaîne de caractères - Lecture au clavier avec `fgets`

🚨 Dans un **`scanf`**, tout caractère blanc (espace, retour à la ligne…) est considéré comme un indicateur de fin de saisie. Pour saisir au clavier une phrase contenant des espaces, il faut alors utiliser la fonction **`fgets`** :

```c
#include <stdio.h>
#define TAILLE_PHYSIQUE 100
int main(void) {
  char chaine[TAILLE];
  printf("Entrez votre prénom (100 caractères max) :\n");
  fgets(chaine, TAILLE_PHYSIQUE, stdin); // stdin indique que les données proviennent du clavier
  printf("Vous avez saisi : %s", chaine);
  return 0;
}
```

⚠️ **`fgets`** ajoute un retour à la ligne à la fin de la chaîne lue (s'il reste de la place). C'est à nous de le supprimer si nous ne souhaitons pas le garder (voir fonction **`strlen`** à la fin du cours).

Rappel : Les tableaux, de simples pointeurs



Donc **`phrase[3]`** est équivalent à **`*(phrase + 3 * sizeof(char))`**.

Rappel : Les tableaux, de simples pointeurs

Lorsque nous prenons un tableau en paramètres d'un sous-programme, nous avons le choix entre 3 notations dans son prototype :

```
void maProcedure(char monTableau[MAX]);

// ou :
void maProcedure(char monTableau[]); // inutile de préciser la taille physique de la première dimension

// ou :
void maProcedure(char* monTableau); // Notation la plus utilisée pour les chaînes de caractères
```

Chaîne de caractères - Passage en paramètres

Avec taille logique (moins bien)

```c
#include <stdio.h>
#define MAX 10

void displayCaracteres(char* string, int tailleLogique) {
    int i = 0;
    for(i = 0; i < tailleLogique; i++) {
        // On affiche un à un tous les caractères :
        printf("%c", string[i]);
    }
}

int main(void) {
    // Tableau de 10 caractères contenant la chaîne "Toto" (4 caractères) :
    char firstname[MAX] = "Toto";
    displayCaracteres(firstname, 4);
    return 0;
}
```

Chaîne de caractères - Passage en paramètres

Sans taille logique (mieux)

```c
#include <stdio.h>
#define MAX 10

void displayCaracteres(char* str) { // plus besoin de la taille logique
    int i = 0;
    for(i = 0; str[i] != '\0'; i++) {
        // On affiche un à un tous les caractères :
        printf("%c", str[i]);
    }
}

int main(void) {
    // Tableau de 10 caractères contenant la chaîne "Toto" (4 caractères) :
    char firstname[MAX] = "Toto";
    displayCaracteres(firstname);
    return 0;
}
```

Empêcher la modification d'une chaîne reçue en paramètres

De manière générale, les constantes **`const`** sont très peu utilisées. Nous utilisons à la place les **macros** de préprocesseur.

En revanche, les constantes **`const`** sont très utilisées avec les paramètres de fonctions afin d'empêcher leur modification dans celles-ci (dans le cas de passage par adresse) :

```
void maProcedure(char* monTableau); // la chaîne nomTableau pourra être modifiée dans la procédure (l'originale
                                    // dans le programme appelant le sera aussi puisqu'il s'agit du même espace mémoire)

void maProcedure(const char* monTableau); // ici, impossible de modifier la chaîne dans la procédure
```

Voir : https://stackoverflow.com/questions/9834067/difference-between-char-and-const-char

Bibliothèque `string.h`

La bibliothèque standard **`string.h`** propose des fonctions permettant de simplifier le traitement des chaînes de caractères.

Nous allons en voir quelques-unes :

```
 char* strcpy(char* destination, const char* source);
 char* strcat(char* s1, const char* s2);
   int strcmp(const char* s1, const char* s2);
size_t strlen(const char* s);
...
```

Le type **`size_t`** est un alias (**`typedef`**) de **`unsigned int`**.

Bibliothèque string.h - `strcpy`

Copier une chaine de caractères dans une autre.

```c
#include <stdio.h>
#include <string.h>
#define MAX 50
void displayCaracteres(char* string, int size) {
    for(int i = 0; i < size; i++) {
        printf("string[%d]='%c'\n", i, string[i]);
    }
}
int main(void) {
    char chaine1[MAX],
         chaine2[MAX] = "ECE Lyon";
    strcpy(chaine1, "Hello !"); // équivalent à : chaine1 = "Hello !"; (affectation)
    displayCaracteres(chaine1, MAX);
    strcpy(chaine1, chaine2);
    displayCaracteres(chaine1, MAX);
    return 0;
}
```

🚨 Attention à avoir suffisamment de cases pour copier !
https://www.tutorialspoint.com/c_standard_library/c_function_strcpy.htm

Bibliothèque string.h - `strcat`

Concaténation de deux chaînes de caractères.

```c
#include <stdio.h>
#include <string.h>
#define MAX 50

int main(void) {
    char chaine1[MAX] = "Hello",
         chaine2[MAX] = "ECE Lyon";
    strcat(chaine1, " "); // on commence par rajouter un espace
    strcat(chaine1, chaine2); // puis la chaîne 2

    printf("%s", chaine1); // "Hello ECE Lyon"
    return 0;
}
```

https://www.tutorialspoint.com/c_standard_library/c_function_strcat.htm

Bibliothèque string.h - `strcmp`

Comparaison alphabétique de deux chaines.

```c
#include <stdio.h>
#include <string.h>
#define MAX 50

int main(void) {
    int res = 0;
    char chaine1[MAX] = "ECE Paris",
         chaine2[MAX] = "ECE Lyon";
    res = strcmp(chaine1, chaine2);
    printf("Chaînes triées alphabétiquement :\n");
    if (res < 0) { // res < 0 => chaine1 avant chaine2
        printf("%s\n%s", chaine1, chaine2);
    }
    else if (res > 0) { // res > 0 => chaine1 après chaine2
        printf("%s\n%s", chaine2, chaine1);
    }
    else { // res == 0 => chaine1 identique à chaine2
        printf("%s (les deux chaînes sont identiques)", chaine1);
    }
    return 0;
}
```

https://www.tutorialspoint.com/c_standard_library/c_function_strcmp.htm

Bibliothèque string.h - `strlen`

Récupération de la taille de la chaîne de caractères (jusqu'à '\0' exclu)

```c
#include <stdio.h>
#include <string.h>
#define MAX 501

void saisirLigne(char chaine[MAX]) {
  printf("Veuillez saisir une phrase (%d caractères maximum) :\n", MAX-1);
  fgets(chaine, MAX, stdin);
  // on remplace le dernier caractère écrit par fgets ('\n') par un '\0'
  chaine[strlen(chaine)-1] = '\0';
}

int main(void) {
    char chaine[MAX];
    saisirLigne(chaine);
    printf("%s", chaine);
    return 0;
}
```

https://www.tutorialspoint.com/c_standard_library/c_function_strlen.htm

Chaînes de caractères - Mise en garde

⚠️ Il faut toujours faire attention à ce que le contenu de la chaîne de caractères ne dépasse pas la taille physique du tableau - 1.

Elle doit toujours être suffisante pour stocker le contenu textuel et le caractère de fin de chaîne : '`\0`'.

Aléatoire - `rand()`

La bibliothèque **`stdlib.h`** propose une fonction intéressante : **`rand()`**.

Cette fonction retourne un **nombre aléatoire entre `0` et `RAND_MAX`** (**`2147483647`**).

`rand()` est implémentée via un **algorithme déterministe**, dont la particularité est de toujours produire la même sortie, pour une même entrée.
Cela pose problème, car à chaque exécution du programme, les mêmes valeurs sont systématiquement retournées.

Aléatoire - `rand()` - exemple

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

int main(void) {
    int i = 0;
    for(i = 0; i <= 5; i++) {
        printf("%d", rand());
        if (i != 5) printf(", ");
        else printf(".");
    }
    return 0;
}
```

Retourne systématiquement et pour chaque exécution la même chose, par exemple :

16807, 282475249, 1622650073, 984943658, 1144108930, 470211272.

Aléatoire - `rand()` + `time()`

Pour éviter d'optenir les mêmes nombres à chaque exécution, nous faisons intervenir le temps dans les valeurs d'entrée :

```c
#include <time.h>
...
int main(void) {
  srand(time(NULL)); // ⚠ à n'écrire qu'une seule fois, avant le premier appel de rand()
  ...
  return 0;
}
```

La fonction **`srand`** permet de modifier l'une des valeurs d'entrée utilisée par **`rand()`** pour retourner un nombre aléatoire. Nous ajoutons ici en entrée le temps au moment de l'exécution qui garantira d'avoir à chaque fois une entrée différente, et donc une sortie différente.