Bases de la programmation impérative
Deuxième partie

Y. Guesnet

Département d'informatique
Université de Rouen

2 novembre 2025

Plan

Plan

Plan

Les tableaux

Introduction

- Les tableaux vont nous permettre d'agréger des valeurs de même type (les tableaux sont des types agrégés).
- Un type tableau dont les éléments seront de type T sera appelé tableau de T et par extension nous dirons d'une variable de type tableau de T qu'il s'agit d'un tableau de T (les tableaux sont des types dérivés).
- Le nombre d'éléments d'un tableau est appelé longueur du tableau.
- En C, les tableaux ont une longueur fixée lors de leur définition.

Les tableaux

Déclaration/définition

Pour déclarer et définir une variable *array* de type tableau de *T* de longueur *l*, nous utiliserons la syntaxe :

```
T array[l];
```

Exemple

Pour déclarer et définir un tableau de 10 entiers, nous pourrons écrire :

```
int array[10];
```

Les tableaux

Remarque

Il ne faut pas confondre la taille d'un tableau avec sa longueur. Si *array* est une variable de type tableau, on pourra passer de l'une à l'autre avec :

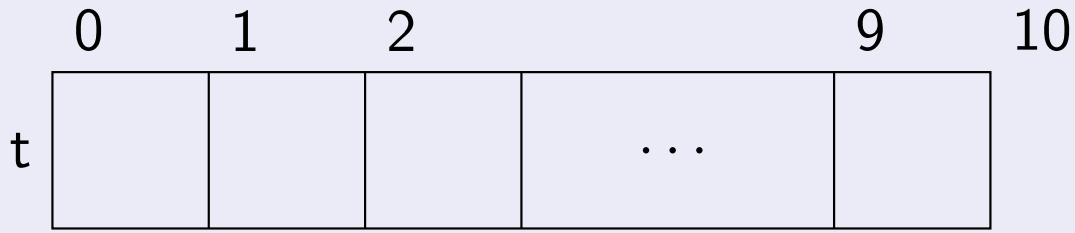
```
size_t array_length = sizeof array / sizeof array[0];
```

Plan

Les tableaux

Accès aux éléments

- L'accés aux éléments d'un tableau se fait grâce à leur indice, le premier élément a pour indice 0, le second a pour indice 1 et ainsi de suite.



- Pour accéder aux éléments d'un tableau nous utiliserons l'opérateur « `[ ]` », par exemple :

```
int array[10];
t[0] = 1;
t[1] = 2 * t[0];
t[2] = t[0] + t[1];
```

Les tableaux

Indices de tableaux

Les indices des tableaux sont de type `size_t`. Nous pourrons donc écrire :

```
int t[10];
...
for (size_t k = 0; k < 10; ++k) {
  printf("t[%zu] = %d\n", k, t[k]);
}
```

Les tableaux

Remarque

Par convention, nous nous interdisons les nombres « magiques », nous devrons donc plutôt écrire :

```
#define TAB_LENGTH 10
...
int t[TAB_LENGTH];
...
for (size_t k = 0; k < TAB_LENGTH; ++k) {
  printf("t[%zu] = %d\n", k, t[k]);
}
```

Les tableaux

Rappel : les macros constantes

Nous pouvons définir une nouvelle macro-constante à l'aide de la directive de pré-compilation `#define` :

```
#define MACRO_ID <macro replacement>
```

Notez que la ligne ne se termine pas par un point.

Les tableaux

Initialisation

- Nous pourrons initialiser un tableau lors de sa définition :

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

  Ici `t[0]` vaut 1, `t[1]` vaut 2, `t[2]` vaut 3 et les autres éléments valent 0.

- On peut aussi préciser les indices des éléments qu'on souhaite initialiser :

```
int array[TAB_LENGTH] = {[0] = 1, [9] = 2};
```

- Lorsqu'il est initialisé, on peut omettre la longueur du tableau :

```
int t1[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int t2[] = {[0] = 1, [9] = 2};
```

Plan

Les tableaux

kékifé ?

Que produit le code suivant ?

```
int main(void) {
  int t[TAB_SIZE] = { };

  foo(TAB_SIZE, t);
  display_tab_int(TAB_SIZE, t); // Affiche le tableau

  return EXIT_SUCCESS;
}
```

avec la fonction *foo* définie comme suit :

```
void foo(size_t n, int t[n]) {
  for (size_t k = 0; k < n; ++k) {
    t[k] += (int) k;
  }
}
```

Les tableaux

Tableau et expressions

Lorsqu'une expression a pour type « tableau de T », elle est implicitement convertie en « pointeur sur T » et vaut l'adresse du premier élément du tableau.

Les tableaux

Corollaire 1

Les paramètres de fonction de type tableau peuvent s'écrire en utilisant des types pointeurs.
Ainsi, les signatures suivantes sont équivalentes :

```
void foo(int t[]);
void foo(int *t);
```

Corollaire 2

Les variables de type tableau ne sont pas des *l-values* :

```
int t1[3] = { 1, 2, 3 };
int t2[3];
t2 = t1; // Non valide
```

Plan

Les tableaux

Tableau à plusieurs dimensions

Il est possible de déclarer des tableaux à plusieurs dimensions :

```
int mat[][5] = {
  {1, 2, 3, 4, 5},
  {6, 7, 8, 9, 10},
  {11, 12, 13, 14, 15},
  {16, 17, 18, 19, 20},
};
for (size_t k = 0; k < sizeof mat / sizeof mat[0]; ++k) {
  for (size_t j = 0;
       j < sizeof mat[0] / sizeof mat[0][0]; ++j) {
    printf("%5d", mat[k][j]);
  }
  printf("\n");
}
```

Les tableaux

Remarque

Lors de la déclaration d'un paramètre de fonction de type tableau à plusieurs dimensions, dans le type, seule la première dimension peut être omise. Il en est de même lorsqu'on initialise un tableau lors de sa définition.

```
int mat[][5] = {
  ...
};

...

void print_matrix(const int mat[][5], size_t rows_num);
```

Plan

Les variables non modifiables

Remarque

L'attribut `const` permet d'indiquer qu'une variable (ou un paramètre) n'est pas modifiable :

```
// Une variable non modifiable
const int n = 10;

// Un tableau dont les valeurs des éléments ne sont
// pas modifiables
const int t[n] = {0};

// msg pointe sur une chaine de caractères non modifiable
const char *msg = "Bonjour";

// msg est non modifiable et elle pointe sur une chaine
// de caractères non modifiable
const char *const msg = "Bonjour";
```

Les alias de types

Alias de type

Nous pouvons déclarer des synonymes pour des types à l'aide du mot clef `typedef` :

```
typedef const char* string_type;
string_type msg = "Bonjour";
```

Plan

Plan

Les chaines de caractères

Introduction

- En C, les chaines de caractères sont des suites finies de caractères se terminant par le caractère nul (`'\0'`).
- Il n'existe pas de type dédié pour les chaines de caractères.
- Elles peuvent être vues comme une suite d'éléments de type `char` stockés en mémoire dans des espaces contigus.
- Une variable de type `char []` ou de type `char *` peut donc faire référence à une chaine de caractères.

Les chaines de caractères

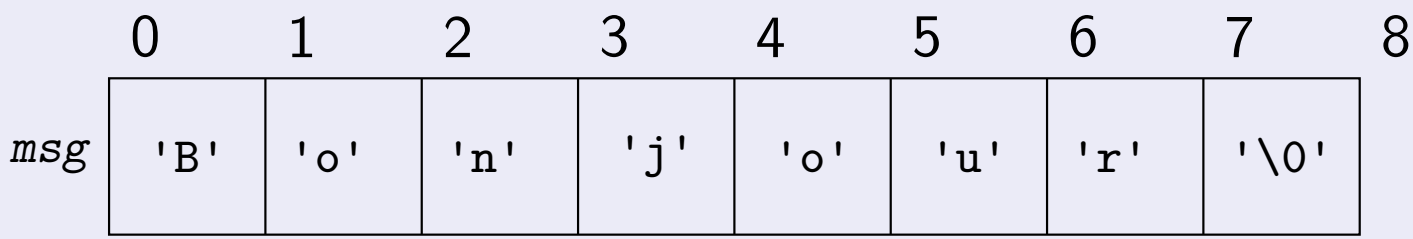
Constantes littérales

- Nous avons déjà utilisé des constantes littérales chaines de caractères comme avec `printf("Bonjour")`.
- Nous pourrons aussi utiliser des variables pour accéder aux chaines, par exemple :

```
char msg[] = "Bonjour";
```

qui est équivalent à

```
char msg[] = {'B', 'o', 'n', 'j', 'o', 'u', 'r', '\0'};
```

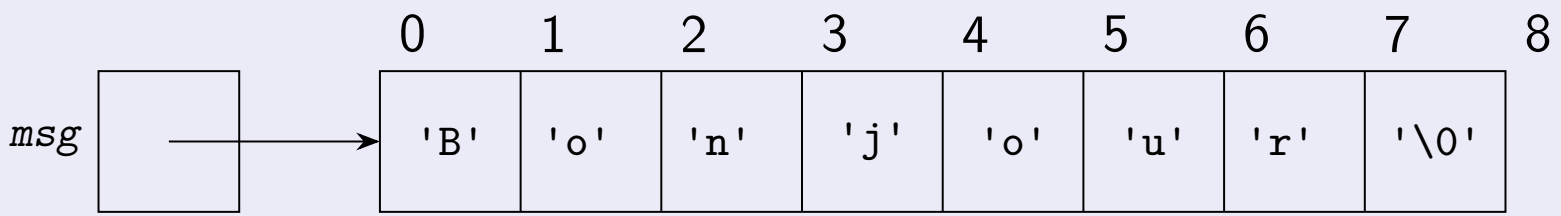


Les chaines de caractères

Constantes littérales

Les constantes littérales sont stockées dans un tableau de caractères. Les éléments de ce tableau ne doivent pas être modifiés. On peut accéder à l'adresse du premier élément de ce tableau à l'aide d'une variable de type pointeur :

```
const char *msg = "Bonjour";
```



Plan

Les chaines de caractères

Outils sur les fonctions

Le fichier d'en-tête *string.h* déclare de nombreuses fonctions aidant à la manipulation des chaines. On trouve, par exemple :

- `size_t strlen(const char *s);`
- `int strcmp(const char *s1, const char *s2);`
- `char *strncpy(char *dst, const char * src, size_t n);`
- `char *strncat(char *dst, const char * src, size_t n);`

Les chaines de caractères

strlen

La fonction *strlen* retourne la longueur de la chaine. Par exemple :

```
const char *msg = "Bonjour";
printf("Longueur du message : %zu\n", strlen(msg));
```

affichera

```
Longueur du message : 7
```

Remarque

Il ne faut pas confondre la longueur de la chaine avec le nombre d'octets la constituant : si la longueur de la chaine est n alors elle est constituée de $n + 1$ octets (il faut tenir compte du `'\0'` final).

Les chaines de caractères

strcmp

- Que produit le code suivant (et pourquoi) ?

```
const char *msg1 = "Bonjour";
const char msg2[] = "Bonjour";
printf("%s == %s : %d\n", msg1, msg2, msg1 == msg2);
```

- Pour comparer deux chaines de caractères on pourra utiliser la fonction *strcmp(s1, s2)*, elle retourne 0 si les deux chaines sont identiques, une valeur strictement négative si *s1* est lexicographiquement strictement plus petite que *s2* et une valeur strictement positive sinon.

```
const char *msg1 = "Bonjour";
const char msg2[] = "Bonjour";
printf("%s == %s : %d\n", msg1, msg2,
       strcmp(msg1, msg2));
```

Les chaines de caractères

strncpy

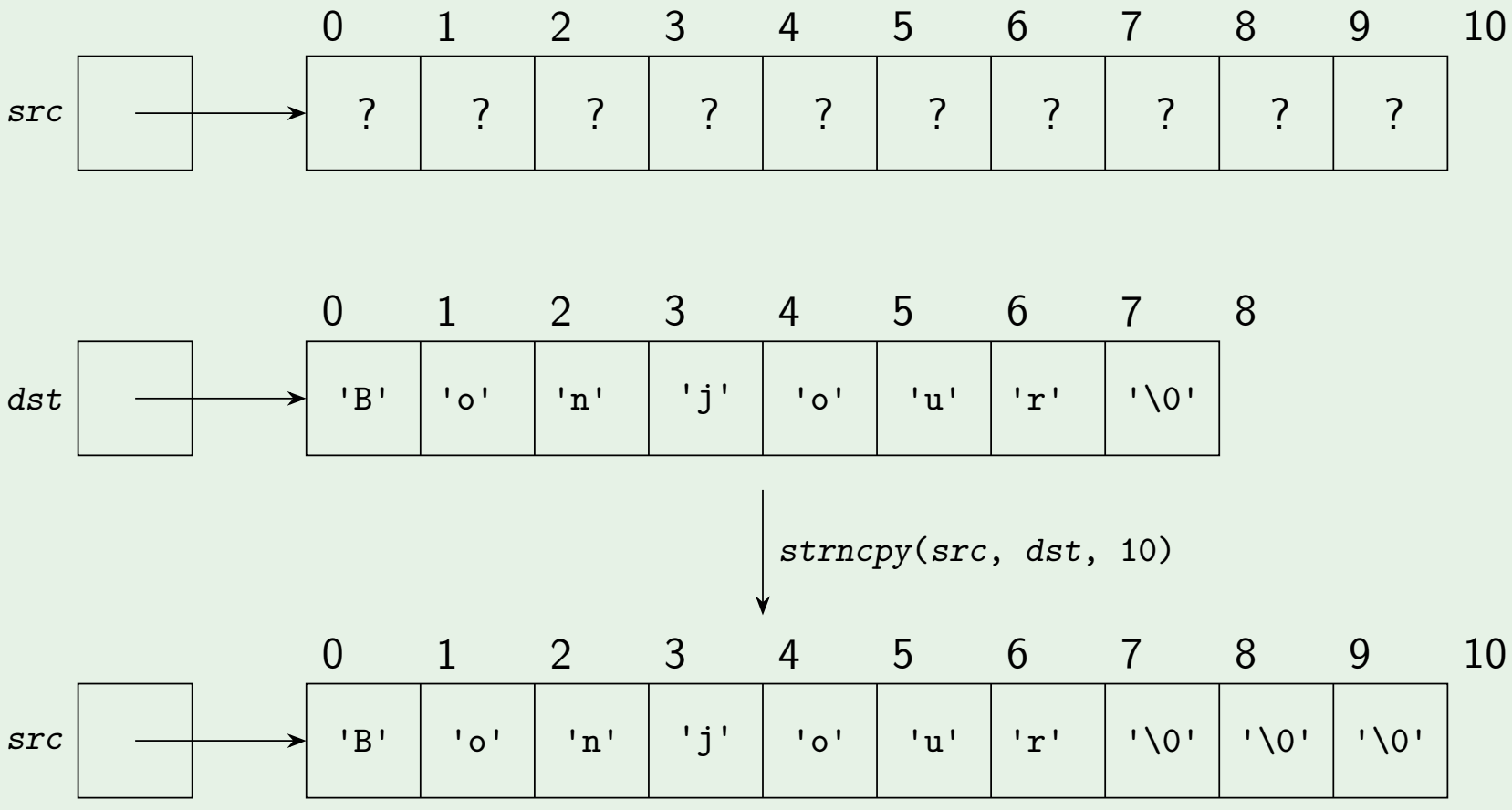
- La fonction *strncpy*(*dst*, *src*, *n*) copie les caractères de *src* dans *dst*. La fonction ne copie pas plus de *n* caractères.
- Si moins de *n* caractères sont copiés alors des `'\0'` sont ajoutés à la fin de *src* de façon à atteindre *n* écritures.
- La valeur de retour de *strncpy* est *dst*.

Remarque

Cette fonction est dangereuse : si la longueur de *src* est plus grande ou égale à *n*, il n'y aura pas de `'\0'` ajouté à la fin des caractères copiés. Dans ce cas *dst* ne peut pas être considérée comme une chaine de caractères.

Les chaines de caractères

Exemple



Les chaines de caractères

strncat

- La fonction `strncat(dst, src, n)` copie les caractères de `src` à la suite des caractères de `dst`. L'ajout débute sur le caractère de fin de chaine de `dst`.
- La fonction ne copie pas plus de `n` caractères.
- Un caractère nul (`'\0'`) est ajouté à la suite des caractères copiés.

Remarque

Contrairement à la fonction `strncpy`, la fonction `strncat` ajoute toujours un caractère nul à la fin de la chaine de destination. Avant l'appel à la fonction, il faut donc que la zone mémoire pointée par `dst` puisse accueillir `strlen(dst) + n + 1` caractères.

La fonction *strncat*

Exemple

```c
#define STR_MAX_LEN 20

char dst[STR_MAX_LEN + 1] = "Bonjour";

const char *src = " tout le monde";
strncat(dst, src, sizeof dst - strlen(dst) - 1);
printf("%s\n", dst); // Bonjour tout le monde
```

Les chaines de caractères

Remarque 1

Il existe une fonction *strcat* :

```
char *strcat(char *dst, const char *src);
```

Cette fonction ajoute la chaine pointée par *src* à la suite de la chaine pointée par *dst*. Elle est plus simple à utilisée que *strncat* mais plus dangeureuse puisqu'on ne se pose pas la question de l'espace disponible restant dans *dst*.

Remarque 2

Il existe aussi une fonction *strcpy* :

```
char *strcpy(char *dst, const char *src);
```

Les chaines de caractères

À noter

Les zones mémoires *src* et *dst* ne doivent pas se recouvrir (que ce soit pour *strcpy*, *strncpy*, *strcat* ou *strncat*).

Plan

Les chaines de caractères

Lecture d'une chaine

Pour lire une chaine, nous avons déjà vu que nous pouvons utiliser *scanf*. Il faut cependant prendre des précautions afin d'éviter les débordements (les *buffer overflow*) :

```
#define STR_MAX_LEN 20
#define STR_MAX_LEN_S "20"

char s[STR_MAX_LEN + 1];
if (scanf("%" STR_MAX_LEN_S "s", s) != 1) {
  fprintf(stderr, "erreur de lecture\n");
  exit(EXIT_FAILURE);
}
```

Les chaines de caractères

Construction d'une chaine

Pour construire une chaine, nous pouvons utiliser la fonction *snprintf* :

```
int snprintf(char *s, size_t n, const char *format, ...);
```

- Cette fonction est similaire à *printf* mais elle écrit dans la chaine pointée par *s* au lieu d'écrire sur la sortie standard.
- De plus elle n'écrit pas plus de *n* caractères dans *s* (caractère de fin de chaine compris).
- La fonction retourne le nombre de caractères théoriquement écrits sans tenir compte du `'\0'` final ni de la limite des *n* - 1 caractères.
- Si *n* vaut 0, *s* peut être nul.

La fonction *snprintf*

Exemple

Pour construire une chaine dont on ne connait pas à l'avance la taille, on pourra s'inspirer du code suivant :

```
const char *msg_format = "%lg x² + %lg x + %lg = 0";

// En supposant a, b et c de type double déjà initialisés
int msg_len = snprintf(nullptr, 0, msg_format, a, b, c);
assert(msg_len >= 0);

// Enfreint la règle 85 de l'ANSSI (pas de VLA),
// mais on s'en contente pour le moment
char msg_eq[msg_len + 1];

snprintf(msg_eq, (size_t) msg_len + 1, msg_format,
         a, b, c);
```