C 12 : tableaux à sentinelle et chaînes de caractères

Juliusz Chroboczek

13 décembre 2023

Dans le TP, nous avons représenté les tableaux en utilisation la *représentation Pascal* : un tableau était représenté comme une paire d'un pointeur sur les données et d'un entier indiquant sa taille :

```
struct array {
    int *a;
    int len;
}
```

Dans cette structure, la taille est codée de manière explicite, ce qui simplifie le code et évite les erreurs.

Pour des raisons historiques, le C utilise une autre représentation pour les chaînes de caractères — la représentation *à sentinelle*. La représentation Pascal n'a que des avantages [1], mais comme plusieurs des fonctionnalités du C l'utilisent, il est parfois nécessaire d'utiliser des tableaux à sentinelle.

1 Tableaux à sentinelle

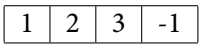

Figure 1 — Un tableau à sentinelle

Un tableau à sentinelle (figure 1) est un tableau où, au lieu de stocker explicitement la longueur du tableau, on marque la fin du tableau par la présence d'un élément distingué. Par exemple, si on travaille avec des tableaux d'entiers naturels, on peut marquer la fin du tableau avec l'entier -1 :

```
int *a;
a = malloc(4 * sizeof(int));
if(a == NULL)
```

1. Des gens vous diront le contraire. Ils ont tort.

```
        return NULL;
    a[0] = 1;
    a[1] = 2;
    a[2] = 3;
    a[3] = -1;
```

Pour calculer la longueur d'un tel tableau, il faut faire une boucle :

```
int array_length(int *a) {
    int i = 0;
    while(a[i] >= 0)
        i++;
    return i;
}
```

La valeur de la sentinelle dépend du type de base du tableau. Dans des tableaux de pointeur, on utilise souvent `NULL` comme valeur sentinelle. Dans les tableaux de `char`, c'est généralement l'octet nul `'\0'` qu'on utilise.

Comme la taille du tableau est implicite, il est facile de dépasser la sentinelle par erreur, ce qui cause souvent des failles de sécurité. Je vous déconseille d'utiliser des tableaux à sentinelle, sauf quand vous n'avez vraiment pas le choix.

2 Chaînes de caractères

Une *chaîne de caractères* est une suite finie de *caractères* et sert à représenter un fragment de texte en langage humain.

2.1 Caractères

Un *caractère* est une unité atomique d'un système d'écriture humain. Intuitivement, un caractère, c'est « une lettre, un chiffre, ou un truc comme-ça ». Cependant, comme les systèmes d'écriture humains ont évolué pendant des millénaires sans suivre les conseils des informaticiens, la notion de caractère n'est pas très claire dans beaucoup de systèmes d'écriture :

- les système d'écriture européens utilisent des caractères avec symboles diacritiques tels que « *é* » — s'agit t'il d'un seul caractère (e avec accent aigü), ou de deux (*e* et accent aigü) ?
- les systèmes d'écriture du proche orient utilisent plusieurs formes pour une seule lettre ; par exemple, la lettre arabe *'ain* a au moins quatre formes distinctes ; s'agit-il de quatre caractères distincts, ou d'un caractère qui peut être affiché de quatre manières différentes ?
- plusieurs systèmes d'écriture utilisent des ligatures : par exemple les lettres arabes *lam* et *alif* forment une ligature lorsqu'elles sont adjacentes ; s'agit-il encore de deux caractères, ou d'un caractère représentant deux lettres ?
- les système d'écrture dérivés du chinois ont des dizaines de milliers d'*idéogrammes*, dont beaucoup sont des variantes mineurs d'autres idéogrammes ; s'agit-il de caractères distincts ?

Fort heureusement, il n'est plus nécessaire de faire ces choix : la norme *Unicode* définit des dizaines de miliers de caractères, les numérote, et donne leur sémantique (par exemple, Unicode indique la majuscule correspondant à une lettre minuscule). Tous les systèmes d'exploitation modernes utilisent la notion de caractère définie par Unicode.

2.2 Codage des caractères

La norme Unicode définit un *j*eu de caractères codé, un ensemble de caractères indexés par des entiers. Il existe plusieurs manières de représenter une suite de caractères Unicode par une suite d'octets.

Le codage dominant aujourd'hui s'appelle *UTF-8* (figure 2). En UTF-8, un caractère est codé par une suite de 1 à 6 octets. UTF-8 est auto-synchronisant : étant donné un flot UTF-8, il est toujours possible d'identifier les frontières de caractères. UTF-8 a aussi la propriété qu'un caractère ASCII est toujours codé par un seul octet : une suite de caractères ASCII est toujours codée de la même manière que la suite de caractères Unicode correspondante.

f	r	a	n	ç		a	i	s
66	72	61	6e	c3	a7	61	69	73

FIGURE 2 — Une chaîne de caractères codée en UTF-8

2.3 Caractères en C

Le C ne manipule pas les caractères. Le type `char` représente un entier codé sur un octet, signé ou non-signé (selon les systèmes). Seuls les caractères ASCII, dont le code Unicode est compris entre 0 et 127, sont codés par un `char`, les autres sont codés par une suite de plusieurs `char`.

Une constante de type `char` s'écrit soit comme un entier, soit comme un caractère ASCII entre apostrophes :

```
char c = 'a';
```

Il existe quelques caractères spéciaux, qu'on dénote avec des séquences magiques qui commencent par un *backslash* « \ » :

- le *backslash*, noté `'\\'` ;
- le *caractère nul*, qui sert de sentinelle, noté `'\0'` ;
- le *caractère de fin de ligne*, qui sert à indiquer à `printf` de passer à la ligne, noté `'\n'`.

2.4 Chaînes de caractères

Une chaîne de caractères est une suite finie de caractères, et sert à représenter un fragment de texte. En C, les chaînes de caractères sont représentés par des tableaux à sentinelle de `char` ; le caractère nul `'\0'` sert de sentinelle :

```
char a[4];
a[0] = 'H';
a[1] = 'i';
a[2] = '!';
a[3] = '\0';
```

En général, on manipulera des pointeurs sur de tels tableaux. Le C a une notation pratique pour de tels pointeurs : une suite de caractères entre doubles guillemets anglais « " » est évaluée en un pointeur sur le tableau à sentinelle correspondant :

```
char *p = "Hi!";
```

2.5 Fonctions utilitaires

Au chapitre sur les pointeurs, nous avons vu un certain nombre de fonctions dont le nom commence par `mem` et qui manipulent des tableaux Pascal de `char`. Il existe une collection analogue de fonction dont le nom commence par `str` et qui manipulent des tableaux à sentinelle de `char` (des chaînes de caractères) :

- La fonction `strlen` recherche la sentinelle et retourne la longueur de la chaîne en octets (pas en caractères). Par exemple, `strlen("été");` retourne 5.
- La fonction `strnlen` est semblable à `strlen`, mais prend en paramètre supplémentaire la taille maximale de la chaîne ; elle retourne même si la sentinelle n'a pas été trouvée.
- La fonction `strcpy` copie une chaîne de caractères. Il vaut mieux utiliser `strncpy`, qui prend en paramètre explicite la taille d'allocation de la destination, et évite donc de déborder d'un tableau.
- La fonction `strcmp` sert à comparer deux chaînes de caractères ; elle retourne 0 si ses paramètres sont égaux.

On peut afficher une chaîne à l'aide de `printf` ; le descripteur est « `%s` », et son paramètre est de type `char *`.

Dans le cas de `scanf`, le descripteur de format prend le nombre maximal de caractères à lire entre le « `%` » et le « `s` » :

```
char *a = malloc(101);
if(a == NULL) return NULL;
scanf("%100s", a);
```

3 Paramètres de ligne de commande

Lorsque vous exécutez votre programme (par exemple à l'aide de `./a.out`, le système invoque la fonction `main` en lui passant deux paramètres : un entier conventionnellement appelé `argc`, qui indique le nombre de *tokens* (mots) sur la ligne de commande, et un tableau de chaînes `argv` de taille `argc` qui contient les *tokens* de la ligne de commande.

Pour accéder à la ligne de commande, il faut déclarer la fonction `main` pour qu'elle accepte ces deux paramètres :

```
int main(int argc, char **argv)
```

Lors de l'exécution, `argv[0]` contient le nom du programme, et `argv[1]` à `argv[argc - 1]` contiennent les paramètres de la ligne de commande.

Il est conventionnel que les paramètres dont le nom commence par un signe moins « `-` » indiquent des options ; la fonction `getopt` de la bibliothèque standard permet de facilement interpréter les lignes de commande qui ont cette syntaxe.