Représentation des chaînes de caractères en langage C

Représentation marqueur de fin

strlen strcmp strcat strcpy

fonctions exploitant des chaînes

Enseignement de programmation en langages C et C++.

Olivier Baiffeux

Maître de conférences HDR

à l'université de Bourgogne.

Version 2021.

C et C++ Partie D

Olivier Bailleux, version 2021

Prérequis : Être capable de mettre en application les notions des parties A, B, C et de prévoir le comportement de programmes utilisant ces notions.

Notions abordées : Représentation en langage C des chaînes de caractères, fonctions standards **strlen**, **strcat**, **strcpy**, **strcmp**, réalisation de fonctions exploitant des chaînes.

Étape D1 : Représentation C des chaînes

Nous allons aborder la représentation des **chaînes de caractères** en langage C.

Une **chaîne de caractère** est représentée par une **suite de codes ascii** suivie de la valeur 0 qui marque la fin de la chaîne, stockée dans un tableau de **char**. Le code ascii d'un caractère est un entier. Pour représenter le code ascii du caractère A, la langage C utilise la notation 'A', et il en va de même pour tous les caractères imprimables.

```
void presentation()
{
    char s[10] = "abcdef";

    char c = s[3];

    printf("%c %d %x\n",c, c, c);

    printf("%s\n", s);

    for(int i=0; i<10; i++)
    {
        printf("%x ", s[i]);
    }
    printf("\n");

    s[3] = 0;

    printf("%s\n", s);
}
```

Ici on affiche le contenu de la variable **c** en trois format différents : caractère imprimable, entier en base 10 et hexadécimal.

d 100 64

Le format %s permet l'affichage d'une chaîne en format C.

abcdef

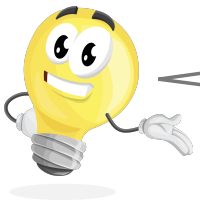
Ici on affiche une par une les valeurs du tableau. On reconnait les codes ascii des cararactères de la chaîne. Au delà du premier 0, les valeurs ne font plus partie de la chaîne.

61 62 63 64 65 66 0 0 0 0

En plaçant un marqueur de fin en position 3, on tronque la chaîne puisque par convention, ce qui suit n'en fait plus partie.

abc

Étape D1 : Représentation C des chaînes



Il est important que vous tentiez de répondre à ses questions en revenant au besoin à la diapo précédente.
Aidez vous d'un dessin ! Vérifiez sur machine après avoir bien réfléchi.

XD01

Vous devez tenter de déterminer sans utiliser d'ordinateur les affichages qui seront réalisés par l'exécution de la fonction suivante.
Rappelez-vous qu'un nom de tableau est un pointeur, qu'un pointeur peut être utilisé comme un nom de tableau, que la somme d'un pointeur et d'un entier est un pointeur qui « pointe un peu plus loin ». (Voir détails dans la partie C).
Rappelez-vous qu'une donnée de type **char** est un entier qui peut être le code ascii d'un caractère imprimable. Vous n'avez pas besoin de connaître la table des codes ascii, il vous suffit de savoir que dans cette table, les codes des lettres minuscules se suivent, tout comme ceux des lettres majuscules.

```
void exemple()
{
    char s[10] = "abc";

    s[5] = 'x';
    s[6] = 'y';
    s[7] = 0;

    printf("%s %s\n",s, s+2, s+4);

    s[3] = 't';

    printf("%s\n", s);

    printf("%d %c\n", 'f'-'b', 'A'+3);
}
```

Étape 12 : Longueur, concaténation, copie de chaînes en C

Revenons à la représentation C et voyons comment manipuler des chaînes.

Le tableau **s** comporte 10 cellules, dont 7 sont utilisées pour le stockage de la chaîne, en comptant le marqueur de fin. Les cellule restantes sont disponibles en cas d'utilisation ultérieure pour stocker une chaîne plus longue.

```
void présentation()
{
    char s[10] = "abcdef";

    char t[] = "xxx";

    strcpy(s,t);

    printf("%s\n", s);

    strcat(s,t);

    printf("%s -> %ld\n", s, strlen(s));
}
```

La taille du tableau **t** est automatiquement ajustée pour contenir la chaîne d'initialisation et son marqueur de fin. (Ici 4 cellules)

La fonction **strcpy** permet de recopier une chaîne **à la place** d'une autre. Le tableau de destination **doit** avoir une taille suffisante. Aucune vérification n'est faite. Le symbole = ne **peut pas** être utilisé dans ce cas car il ne permet pas l'assignation glogable d'un tableau avec le contenu d'un autre.

xxx

La fonction **strcat** permet de recopier une chaîne **à la fin** d'une autre. Le tableau de destination **doit** avoir une taille suffisante pour contenir la chaîne résultante et son marqueur de fin.

xxxxxx → 6

La fonction **strlen** retourne le nombre de caractères d'une chaîne sans compter le marqueur de fin. (Elle retourne un entier long de type **size_t** dont le format d'affichage est **%ld**).

Étape 2 : Longueur, concaténation, copie de chaînes en C

Quelques questions pour vérifier que vous avez compris les points importants

```c
void test1()
{
    char s[20] = "abcdef";

    printf("%s\n", s+2);

    strcat(s+4, "xyz");

    printf("%s\n", s);
}
```

Essayez d'expliquer les affichages constatés lors de l'exécution de la fonction **test1**.

XD02

cdef

abcdefxyz

XD03

Donnez les affichages réalisés par l'exécution de la fonction **test2**. Aidez-vous d'un dessin pour bien comprendre ce qui se passe en mémoire. Attention, pour les fonctions **strcat**, **strcpy** où même **printf**, une chaîne commence à l'adresse qu'on leur passe en paramètre et pas nécessairement au début du tableau dans lequel elle est stockée.

```c
void test2()
{
    char s[20] = "abcdef";

    printf("%s\n", s+3);

    strcat(s+2, "xyz");

    printf("%s\n", s);

    strcpy(s+strlen(s), "abc");

    printf("%s\n", s);
}
```

Étape 12 : Comparaison de chaînes en C

En C, les chaînes se présentent sous forme de tableaux de caractères désignés par leurs adresses mémoire. Si on utilise ==, <, > etc., ce sont les adresses et non les contenus qui sont comparés. Pour comparer les contenus il faut utiliser la fonction standard **strcmp**.

```
void testCompare()
{
    char s1[] = "Tim";

    char s2[] = "Tim";

    char s3[] = "Tom";

    printf("%d\n", s1==s2);

    printf("%d\n", strcmp(s1,s2));

    printf("%d\n", strcmp(s1,s3));
}
```


0

La comparaison des **adresses mémoire** des chaînes désignées par s1 et s2 apour résultat **false**, puisque bien qu'ayant le même contenu, ces chaînes sont implantées à des adresses différentes.

Dans le cas où les deux chaînes comparées ont le même contenu, par convention la fonction **strcmp** retourne 0, mais ce 0 n'est PAS à interpréter comme le Booléen **false**.


0

Lorsque les deux chaînes comparées sont différentes, **strcmp** retourne :
- un entier négatif si la première est située avant la deuxième en ordre lexicographique,
- un entier positif dans le cas contraire.

Ici, "Tim" est avant "Tom", donc la valeur de retour est négative.


-6

L'ordre lexicographique utilisé par **strcmp** place les chiffres avant les lettres et les lettres majuscules avant les minuscules. L'ordre des caractères est celui de leurs codes ASCII.

Étape 12 : Chaînes en C : attention danger

Les chaînes en format C et les fonctions standards qui les manipulent sont très rudimentaires et sources possible de bugs dans les programmes. Examinons leurs limitations.

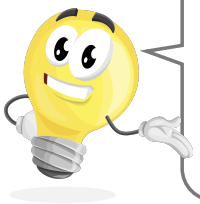
```
void danger()
{
    char s1[] = "Tim";

    char s2[] = "Tim";

    strcat(s1, s2);

    printf("%d\n", strcmp(s1,s3));
}
```

Cet appel de strcat écrit des caractères au delà des limites du tableau s1.

Les seuls caractères utilisables dans les chaînes au format C avec l'assurance d'une parfaite portabilité sur différents sytèmes sont ceux ayant un code ASCII compris entre 1 et 127, c'est à dire les lettres majuscules et minuscules non accentuées, les chiffres et quelques caractères de ponctuation (, ; . / etc.) et caractères spéciaux.
L'utilisation de caractères accentués peut se faire avec des codes ASCII étendus de valeurs supérieure à 127, mais il existe de nombreuses variantes de tels codes avec des risques en matière de portabilité. Il est donc très fortement déconseillé de les utiliser.
Il existe un codage beaucoup plus universel et étendu appelé UNICODE UTF8, mais son utilisation en C sort du cadre de cet enseignement.

Les fonctions **strcpy** et **strcat** ne peuvent pas vérifier qu'à l'adresse de destination il y a de la mémoire utilisable pour stocker la chaîne à recopier ou à prolonger. Il est donc possible de les conduire à écrire dans une zone mémoire interdite, ou par dessus des données stockées dans d'autres variables ou tableaux du programme, ce qui est évidemment catastrophique pour la sécurité et la stabilité des applications concernées. Il existe des versions **strncpy** et **strncat** qui limitent les tailles des chaînes de destination, mais c'est loin d'apporter toutes les garanties de sécurité. Toutes les vérifications utiles doivent être faites avant l'appel des fonctions standards de gestion des chaînes.

Étape D3 : Fonctions exploitant des chaînes au format C

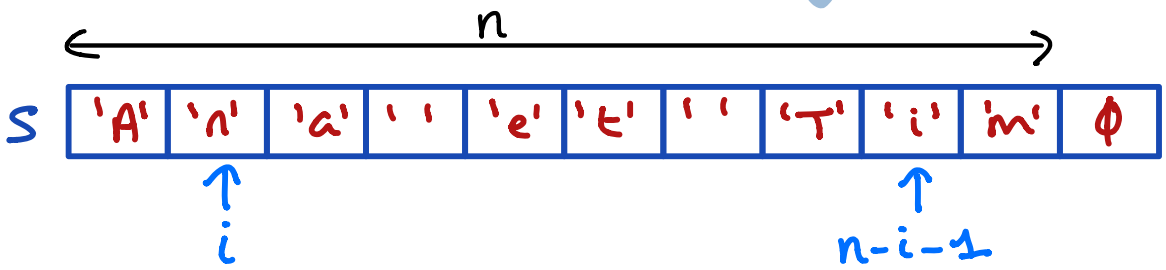


En C les chaînes se présentent sous la forme de tableaux de caractères. En l'état actuel de nos connaissances, une fonction ne peut pas créer et retourner une nouvelle chaîne mais peut modifier une chaine existante.

La fonction **miroir** remplace la chaîne passée en paramètre par sa chaîne miroir. Pour ce faire, elle modifie directement le tableau contenant cette chaîne.

```
void miroir(char* s)
{
    int n = (int)strlen(s);

    for(int i=0; i < n/2; i++)
    {
        char tmp = s[i];

        s[i] = s[n-1-i];

        s[n-1-i] = tmp;
    }
}
```

La chaîne miroir est obtenue par des échanges de caractères situés à des positions symétriques par rapport au centre.



Exemple d'utilisation de la fonction miroir et d'affichage du résultat.

```
void test()
{
    char s[] = "Ana et Tim";

    miroir(s);

    printf("%s\n", s);
}
```

miT te anA

Qu'avons nous appris ?

- Représentation C des chaînes : suite de codes ASCII dans un tableau, terminée par ϕ.
  - Ne pas confondre l'entier zéro, noté 0 ou ϕ ou '\ϕ', et le caractère 'ϕ'.
    - On barre parfois le 0 (ϕ) pour ne pas le confondre avec la lettre O.
- Représentation d'un caractère ascii : entre simples cotes exemple : 'A'.
- Affichage d'une chaîne au format C : format %s de printf
- strlen : longueur d'une chaine
- strcpy : recopie d'une chaine
- strcat : concaténation de 2 chaines
- strcmp : comparaison de 2 chaines
- Passage d'une chaîne à une fonction : par paramètre de type char*
- Fonction retrournant une chaîne : on ne sait pas faire pour l'instant mais une fonction peut modifier une chaine via un pointeur passé en paramètre

Exercices d'assimilation

Ces exercices vous permettront de monter en compétence et de vérifier vos acquis. Ne vous précipitez pas. Si possible, testez vos solutions sur machine. Essayez de comprendre vos erreurs. Au besoin demandez des indices ou explications complémentaires.

XD04

Réalisez une fonction nommée **altStrlen** qui fait exactement la même chose que **strlen** mais sans faire d'appel à d'autres fonctions.

XD05

Réalisez une fonction nommée **altStrcat** qui fait exactement la même chose que **strcat** mais sans faire d'appel à d'autres fonctions.

XD06

Réalisez une fonction **toUpper** qui accepte un paramètre désignant une chaine de caractères et qui remplace, dans cette chaîne, toutes les lettres minuscules par les majuscules correspondantes. Par exemple « Tim et Tom » devient « TIM ET TOM ».
Vous devez vous rappeler que la notation 'A' représente un entier de type **char** dont la valeur est le code ASCII de la lettre A. Et pareillement pour tous les caractères imprimables. Vous n'avez donc PAS besoin d'une table de codes ASCII pour traiter cet exercice. Si vous avez besoin d'utiliser le code ASCII de la lettre z, il suffit d'utiliser la notation 'z'.
Pour savoir si une variable **c** de type **char** représente une lettre majuscule, on peut faire le test :
`if((c >= 'A') && (c <= 'Z'))…`
D'autre part, dans la table des codes ASCII (dont vous navez PAS besoin pour traiter cet exercice), les codes des lettres majuscules se suivent, de même que ceux des lettres minuscules. Il en résulte que `('a'-'A') = ('b'-'B') = ('c'-'C') = …` et donc si une variable **c** de type **char** contient le code ASCII d'une lettre minuscule, le code ASCII de la lettre majuscule correspondante est **c**-`('a'-'A')`.

Exercice de consolidation

Cet exercice est plus compliqué que les précédents. Ne le faites que si vous avez un bon niveau en algorithmique et que vous maîtrisez la récursivité.

XD07

Vous devez compléter la définition de la fonction récursive **printAnagramme** de manière à ce que si **mot** est une chaine, l'appel **printAnagramme(mot, 0)** ait pour effet l'affichage de tous les anagrammes de **mot**. Par exemple si mot est la chaîne "abc" alors l'exécution de la fonction doit provoquer l'affichage de abc, acb, bac, bca, cba et cab.
Dans le cas où le deuxième paramètre à une valeur n supérieure à 0, la fonction doit afficher toutes les chaînes qui commencent par les n premières lettres de mot et se terminent par tous les anagrammes des lettres suivantes. Par exemple si mot est la chaîne "abcd" alors **printAnagramme(mot, 2)** affiche abcd et abdc.
Le contenu de **mot** à l'issue de l'exécution doit être strictement identique à son contenu initial avant l'appel de la fonction. Les seules opérations autorisées sur **mot** sont des échanges de caractères réalisés par appel de la fonction **exg**.

```
void printAnagramme(char* mot, int n)
{
    int fin = (int)strlen(mot);
    if(n == fin)
    {

    }
    else
    {




    }
}
```

Le principe peut se résumer ainsi :
On appelle préfixe du mot les premières lettres avant la position n et suffixe les suivantes. Si le suffixe est vide il suffit d'afficher le mot. Sinon on fait autant d'appels récursifs qu'il y a de lettres dans le suffixe, chacun de ces appels se faisant avec un préfixe augmenté d'une des lettres du suffixe.

Par exemple si le préfixe est "A" et le suffixe "BCD", on pourra faire un appel récursif avec préfixe "AB" et suffixe "CD", un avec préfixe "AC" et suffixe "BD" et un avec le préfixe "AD" et le suffixe "CB". Ces opérations se font par des échanges de lettres avant chaque appel récursif. Mais il est important qu'après tous ces appels, les lettres du suffixe soient remises dans leur ordre initial.

```
void exg(char* str, int i, int j)
{
    char tmp = str[i]; str[i] = str[j]; str[j] = tmp;
}
```