chaînes de caractères

chaînes de caractères

Est-ce qu'il y a des chaînes de caractères en C ?

Il n'y a pas de type spécifique pour représenter les chaînes de caractères.

Une convention (définition):

une chaîne de caractères est une suite d'octets qui termine par le caractère `nul '\0'`

Les chaînes de caractères sont représentées par les pointeurs

char *

pointant vers le premier caractère de la chaîne.

```
Le caractère '\0' est un caractère dont tous les bits
sont 0, ce n'est pas le caractère '0'.

Le code décimal du caractère nul est 0.
```

différence entre une chaîne de caractères (string) et une suite de caractères

chaîne de caractères (string):

- représentée par un pointeur char * vers le premier caractère de la chaîne
- termine par le caractère nul '\0', qui est un marquer de la fin de la chaîne mais qui ne fait pas partie de la chaîne
- pas besoin de stocker la longueur de la chaîne : la longueur c'est le nombre de caractères avant le caractère nul '\0'
- une chaîne de caractères ne contient jamais de caractère nul '\0'

suite d'octets :

- représentée par un pointeur char * vers le premier caractère de la suite
- il faut connaître la longueur (pas de marqueur de la fin de la suite)
- peut contenir plusieurs caractères nuls '\0'

chaînes de caractères versus suites de caractères

Etant donné un pointeur

```
char *p
```

est-ce que c'est un pointeur vers une chaîne de caractères ? Ou vers une suite d'octets quelconque ? Ou juste un pointeur vers un caractère ?

C'est à vous et votre programme de maitriser vers quoi pointe p.

Les fonctions dont le paramètre est une suite quelconque d'octets :

`memmove(), memset()`

ont besoin de longueur comme paramètre.

chaîne de caractères

Si le prototype d'une fonction a une forme

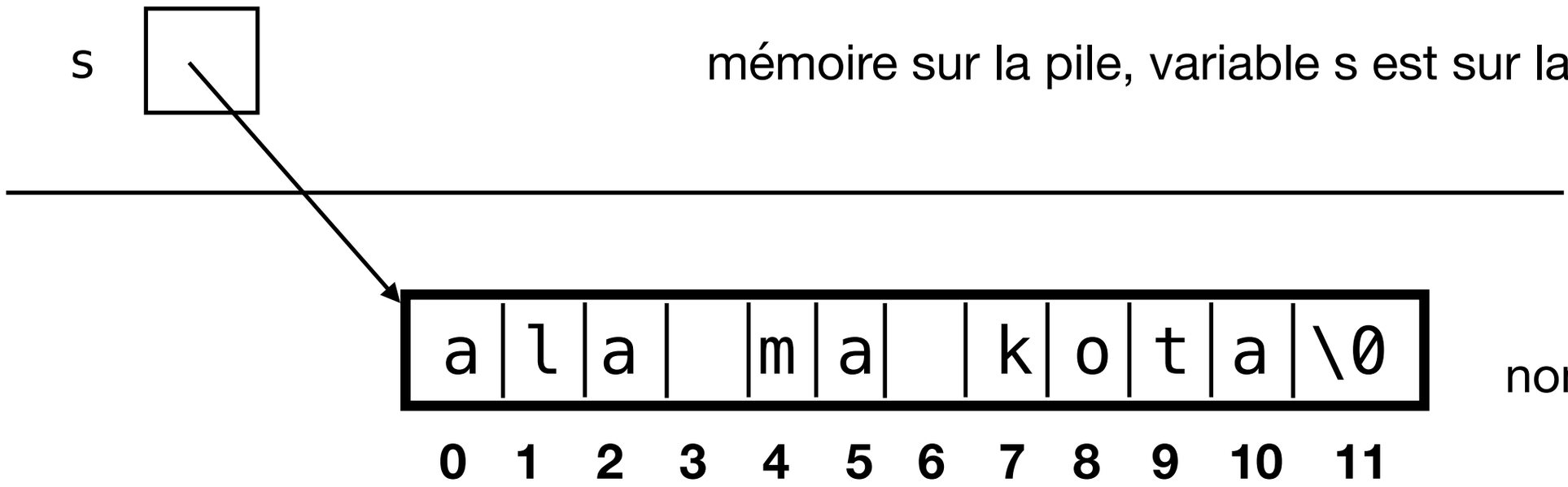
```
f(char *s, ...)  ou f(const char *s, ...)
```

et la description spécifie que s doit être une "chaîne de caractères" (string) alors la fonction f s'attend à ce que s pointe vers une chaîne de caractères qui termine avec le caractère '\0'.

C'est à vous de vous assurer que le paramètre qu'on passe dans f satisfait cette condition. Si vous ne respectez ce contrat le résultat de la fonction n'est pas défini (la fonction calcule n'importe quoi comme le résultat ou l'exécution s'arrête à cause d'erreur).

chaînes de caractères

```
char *s ;

s = "ala ma kota";
```



Compilateur place chaque chaîne de caractères qui apparaît en dur dans le code dans une mémoire qui n'est ni sur la pile ni dans le tas. Cette mémoire est non-modifiable.

chaine_caracterees.c

```c
#include <stdio.h>

int main(void){
  char *s="ala ma kota";

  s[1]++;  /* remplacer l par le caractère suivant m  */

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

U:--- **chaine_caracterees.c** All (C/*l Abbrev)

COURS06 — -bash ▸ Emacs-arm64-12 — 80×10

```
(base) COURS06 $ gcc -Wall -g chaine_caracterees.c
(base) COURS06 $ ./a.out
Bus error: 10
(base) COURS06 $
```

tentative de modification de caractère s[1] échoue

chaine_caracterees2.c

```c
#include <stdio.h>

int main(void){
  const char *s="ala ma kota";

  s[1]++;  /* remplacer l par le caractère suivant m  */

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

```
U:---   chaine_caracterees2.c    All   (C/*l Abbrev)
```

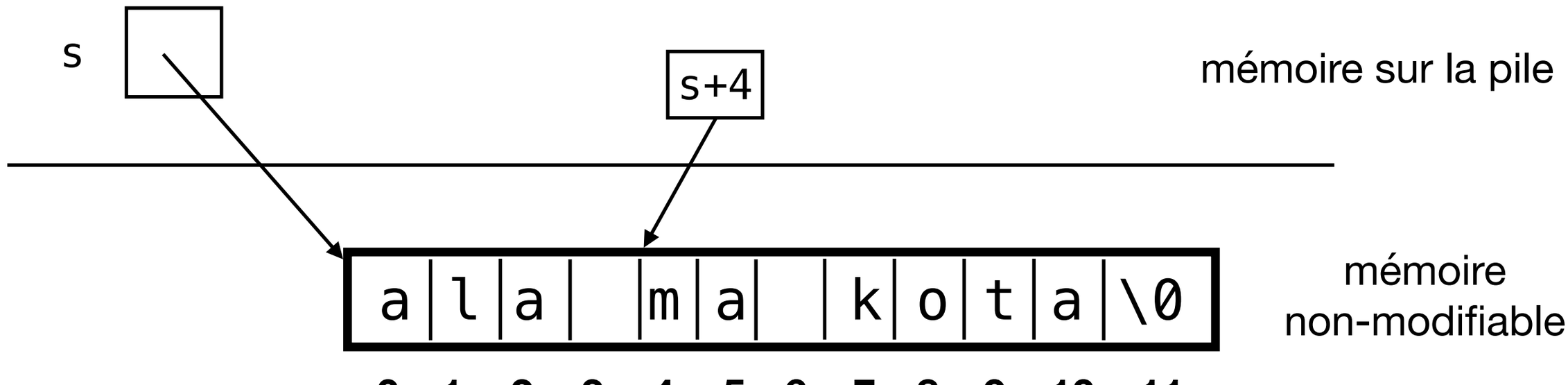
COURS06 — -bash ▸ Emacs-arm64-12 — 80×10

```
(base) COURS06 $ gcc -Wall -g -pedantic chaine_caracterees2.c
chaine_caracterees2.c:6:7: error: read-only variable is not assignable
  s[1]++;  /* remplacer l par le caractère suivant m  */
  ~~~~^
1 error generated.
(base) COURS06 $
```

const   char *s = "ala ma kota";
/* const indique que la chaîne n'est pas modifiable */

chaînes de caractères

```
char *s ;

s = "ala ma kota";
```



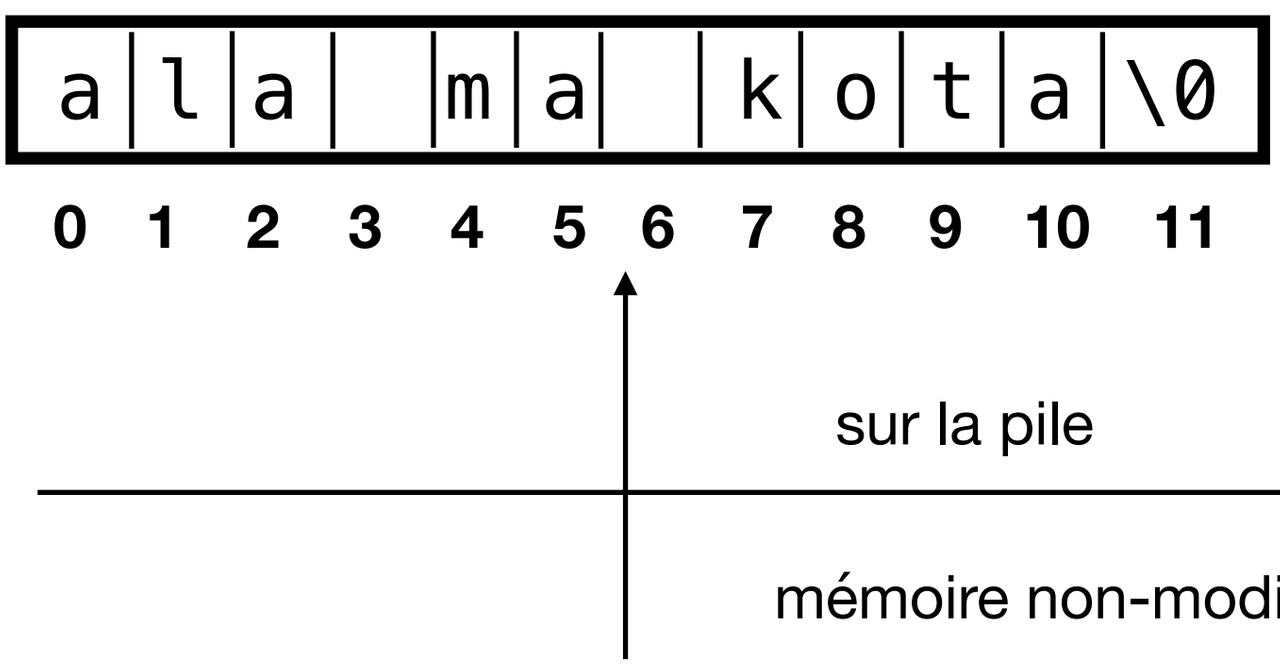

```
printf("%c", s[4]);          OK, écrit :  m
printf("%s", s) ;            écrit  :  ala ma kota
printf("%s", s+4);           écrit  :  ma kota
printf("%s", s+9);           écrit  :  ta
```

initialiser un tableau de char avec une chaîne de caractères

```
char tab[] = "ala ma kota";
```

tab[] est un tableau initialisé avec les caractères **recopiés** à partir de la chaîne à droite, y compris le caractère '\0'.

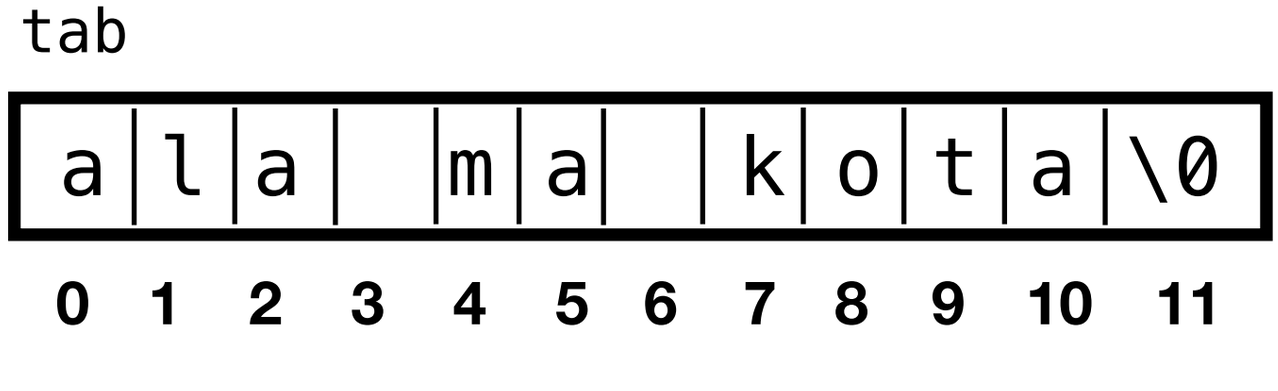
tab



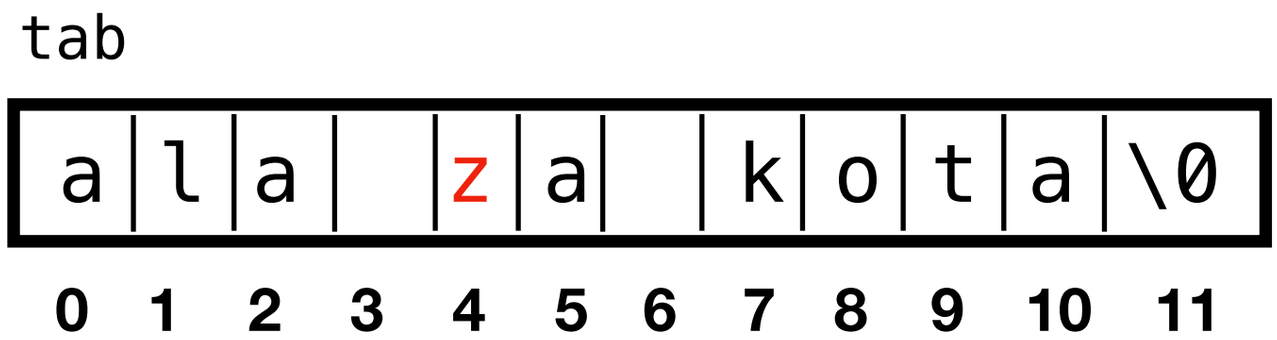
le tableau tab sur la pile est initialisé avec la chaîne copiée depuis la mémoire non-modifiable, tab contient 12 caractères

initialiser un tableau de char avec une chaîne de caractères

```
char tab[] = "ala ma kota";
```




```
printf("%c",tab[4]);          OK, affiche : m
tab[4] = 'z';                 OK,
```




```
printf("%s",tab+4);      affiche : za kota
```

<string.h>

Les prototypes de fonctions de traitement des chaînes de caractères se trouvent dans <string.h>

Leur nom commence par str.

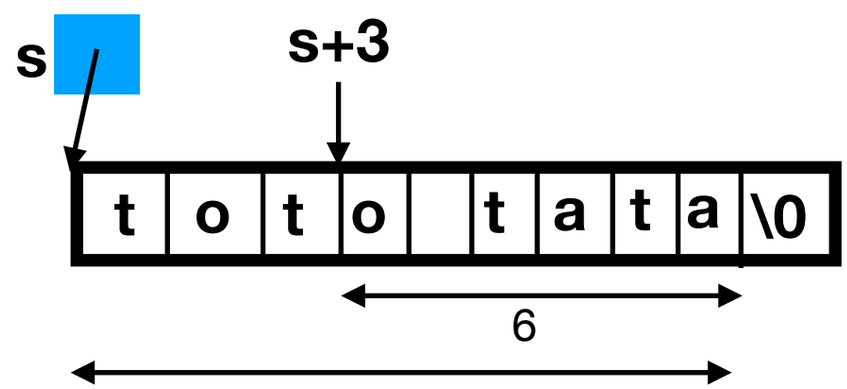
size_t strlen(const char *s)

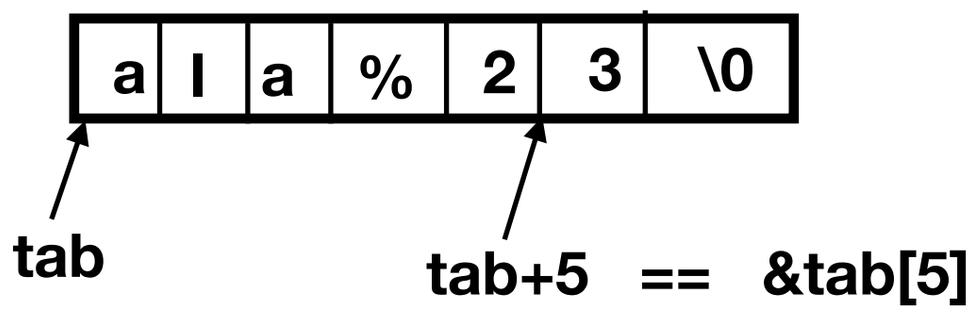
`const :` la fonction ne modifie pas la chaîne pointé par `s`

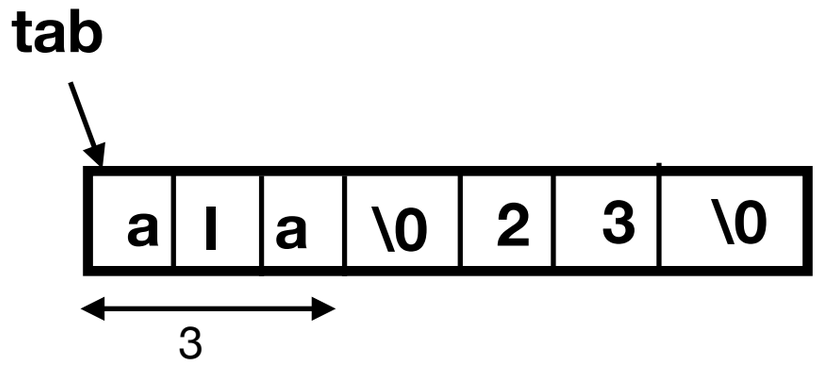
strlen retourne la longueur de la chaîne s, le nombre

de caractère jusqu'à '\0', sans compter '\0'.

```
char *s = "toto tata";

size_t t = strlen(s);  // t == 9

t = strlen(s+3);       // t == 6

char tab[] = "ala%23";

t = strlen( tab );    //t == 6

t = strlen( &tab[5] );    //t == 1

tab[3] = '\0';

t = strlen( tab );     // t == 3
```







exemple : implémenter strlen()soi même

```
size_t strlen(const char *s){
    size_t i;
    for( i=0; *s != '\0'; s++, i++ )
       ;
    return i;
}
```

la boucle peut être écrite comme :

```
for( i=0 ; *s ; s++, i++ )     //pourquoi c'est équivalent ?
      ;
```

Notez que la fonction strlen() ne peut pas vérifier si s un une chaîne de caractères.
Si une suite de caractères à l'adresse s ne termine pas avec le caractère nul alors
la fonction continue le parcours sur les octets qui ne sont plus dans votre suite.

```
char s[]={'a','b','c','d'};

size_t k = strlen(s);  /*  s n'est pas un string,
                        * le résultat indéfini */
```

Tableau de chaînes de caractères ?

```
char *tab[] = {"koza", "totem", "ala",
               "ziemia", "pies", "krowa"};
size_t n = sizeof(tab)/sizeof(tab[0]);
```

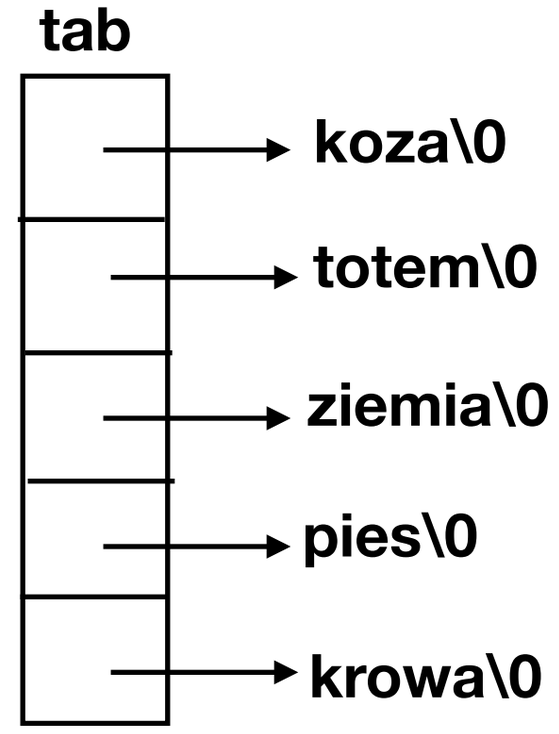


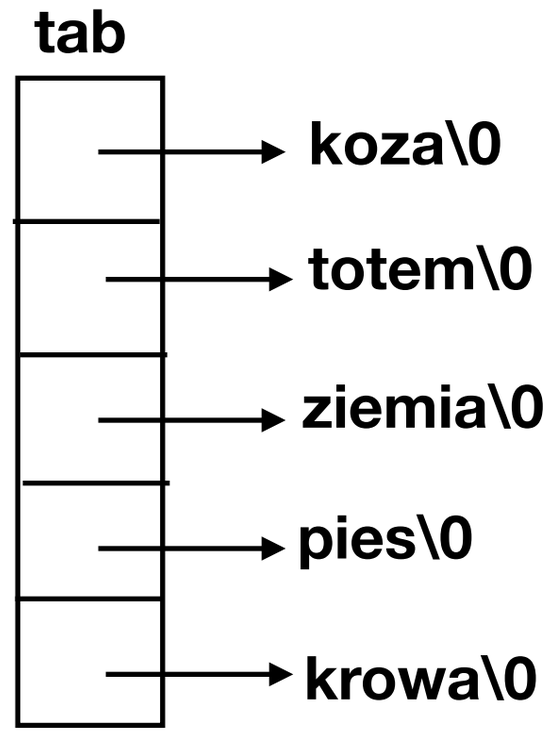
Tableau de chaînes de caractères ne contient pas les chaînes de caractères.

On devait parler de "tableau de pointeurs vers des chaînes de caractères".

Concatener les chaînes de caractères

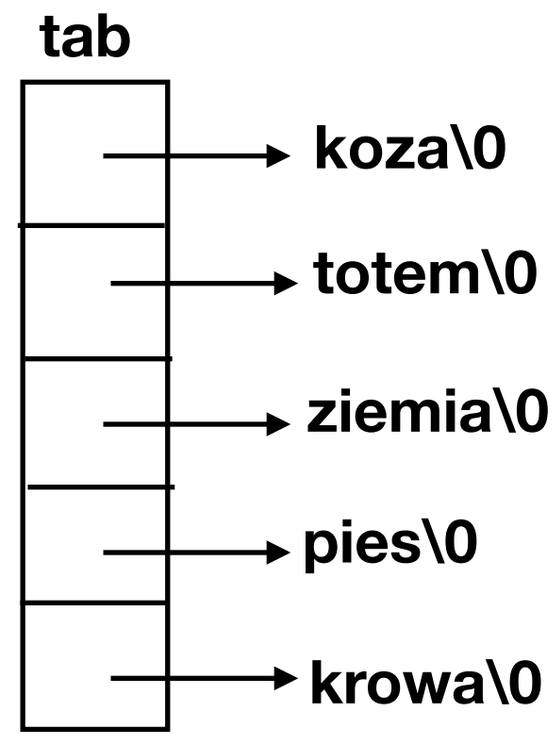
```
char *tab[] = {"koza", "totem", "ala",
               "ziemia", "pies", "krowa"};
size_t n = sizeof(tab)/sizeof(tab[0]);
size_t len;
for( size_t i = 0; i < n ; i++ )
   len += strlen( tab[i] );

/* la mémoire pour le résultat de
 * concaténation : la somme de longueurs + 1 */

char *conc = malloc( len + 1 );
assert(conc != NULL );
```

pour le caractère '\0'



Concatener les chaînes de caractères

```
len = 0;
for( size_t i = 0; i < n ; i++ ){
   size_t dl = strlen( tab[i] );
   memmove( conc + len, tab[i], dl );
   len += dl;
}

/* pour terminer la chaîne ajouter '\0'
 * à la fin */

conc[ len ] = '\0';
```



conc

k	o	z	a	t	o	t	e	m	z	i	e	m	i	a	p	i	e	s	k	r	o	w	a	\0

conversion string -> nombre

convertir un chaîne de caractères en nombre

```
#include <stdlib.h>

double atof(const char *s)   convertit s en double
int    atoi(const char *s)                en int
long   atol(const char *s)                en long
long long atoll(const char *s)            en long long
```

Ces fonctions ne prennent pas en compte les caractères d'espacement (dans le sens de `isspace()` (espace, '\n', '\t') au début de la chaîne et arrêtent la conversion quand elles rencontrent un caractère qui n'est pas un chiffre.

```
char *s = "    \n23abc  ";
int i = atoi( s );          /* i == 23 */

char t[] = "  \n  -54.89  mld";
double d = atof( t );   /* d == -54.89 */

int j = atoi("  \n  a45bc");  /* j == 0 */
```

l'ordre de dictionnaire

`int strcmp(const char *s, const char *t)`

`strcmp()` retourne :

- une valeur < 0 si s inférieur à t dans l'ordre de dictionnaire
- == 0 si s égal à t dans l'ordre lexicographique
- une valeur > 0 si s supérieur à t dans l'ordre de dictionnaire

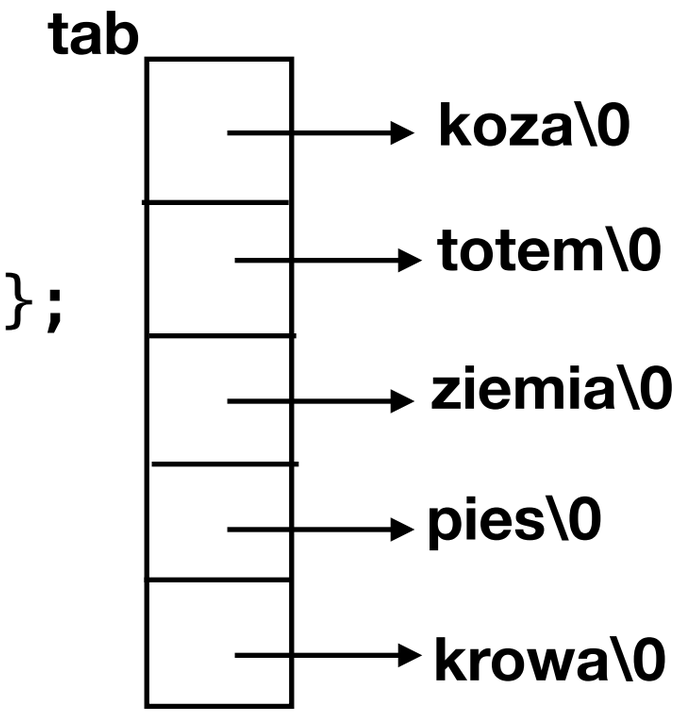
Implémentation possible de strcmp() :

```
int strcmp(const char * s,const char *t){
    while( *s != '\0' && *t != '\0' && *s == *t){
          s++; t++;
    }
    if( *s < *t)
         return -1;
    else if( *s == *t )
         return 0;
    return 1;
}
```

Tableau de chaînes de caractères ?

Trouver le mot le plus grand dans l'ordre lexicographique dans un tableau de mots.

```
char *tab[] = {"koza", "totem", "ala",
                "ziemia", "pies", "krowa"};
size_t n = sizeof(tab)/sizeof(tab[0]);
int i, m = 0;
for( i = 1; i < n; i++){
    if( strcmp(tab[i],tab[m]) > 0 )
      m = i;
}
printf("%s\n", tab[m]);
```



copier une chaîne de caractères

```
char *strcpy(char *dest, const char *src)
```

copie la chaîne `src`, y compris `'\0'`, à l'adresse `dest` et retourne `dest`.

Le bloc de mémoire à l'adresse `dest` doit être suffisamment grand (la fonction `strcpy()` de fait pas d'allocation de mémoire).

```
char s[] = "toto";
char *dest = malloc( strlen(s) + 1 );
strcpy(dest, s);
```

n'oubliez +1 pour avoir la place pour le caractère '\0'

```
char *strncpy(char *dest, const char *src, size_t n)
```

`strncpy()` copie au plus n octet de l'adresse src vers l'adresse dest et retourne dest.

Si src contient moins de n caractères alors après avoir copié src la fonction complète dest avec le caractère '\0' jusqu'à n caractères. Notez que comme le résultat dans dest il est possible d'obtenir une suite de caractères qui ne termine pas avec '\0'.

```
char *source =  "abcdef";  /* la chaîne source */
char tab[4];
strncpy(tab, source, 4);  // tab contient 'a','b','c','d'
char b[9];
strncpy(b, source, 9);  // b contient 'a','b','c','d','e','f','\0','\0','\0'
```

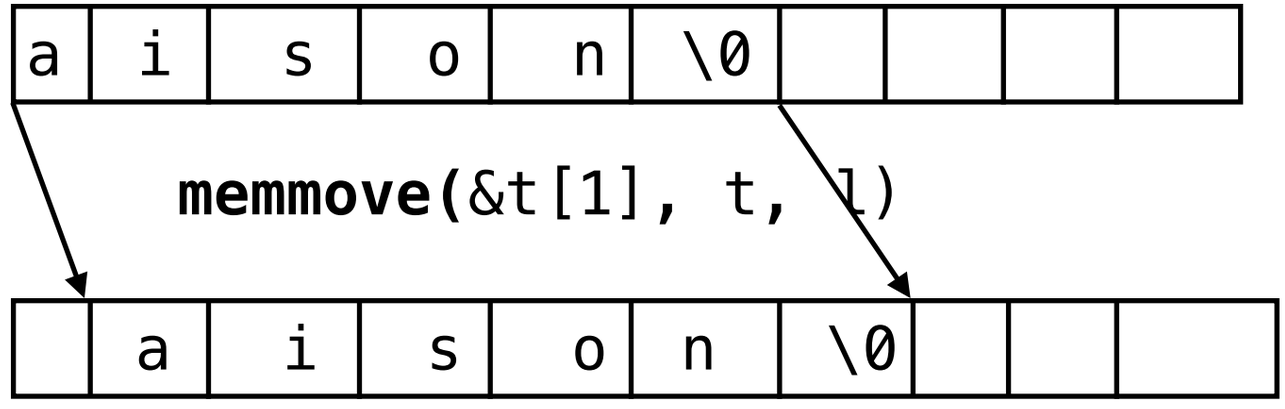
strcpy() peut écrire au delà de la mémoire disponible dans dest : corruption de mémoire.

strncpy() pas de danger de corruption de mémoire. Par contre strncpy() peut produite dans dest une suite de caractères qui n'est pas une chaîne de caractères.

```
char *strncpy(char *dest, const char *src, size_t n)
      char *strcpy(char *dest, const char *src)
```

dest et src ne doivent pas chevaucher:

```
char t[ 10 ] = "aison"; // ajouter la lettre m au début
size_t l = strlen( t ) + 1; // l==6
memmove( &t[1], t, l);  // memmove(t+1, t, l);
t[0] = 'm' ;
```



~~strncpy(&t[1], t, 9);~~ /* incorrect, les zones pontées par
* src et dest chevauchent */