

Langage C

Wieslaw Zielonka
zielonka@irif.fr

chaînes de caractères : rappel

`size_t strlen(const char *s)`

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

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

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

`double atof(const char *s)`

`int atoi(const char *s)`

`long atol(const char *s)`

`long long atoll(const char *s)`

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

`dest` : un pointeur vers une chaîne de caractères

concatène la chaîne pointée par `src` à la suite de `dest` et retourne `dest`

Attention : il faut suffisamment de mémoire à l'adresse `dest` pour les deux chaînes `s` et `t`. La fonction n'ajoute pas de mémoire supplémentaire dans `dest`.

INCORRECT :

```
char *x = "chien";  
char *y = "chat";  
strcat(x,y);
```

```
/* incorrect x pointe vers  
une zone de mémoire  
non-modifiable */
```

INCORRECT :

```
char x[] = "chien" ;  
char *y = "chat" ;  
  
strcat(x,y);
```

```
/* le tableau x contient assez  
* de place pour la chaîne "chien"  
* mais pas de place pour y ajouter  
* encore 4 lettres de "chat",  
* débordement */
```

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

```
char *x = "chien";  
char *y = "chat";  
char *z = malloc( strlen(x) + strlen(y) + 1);  
strcpy(z, x); /*copier x dans z, strcat() ne s'applique pas  
              * parce que z ne pointe pas vers une chaîne  
              * de caractères */  
strcat(z, y); /* concatener y à la suite */
```

une autre solution :

```
char *x = "chien"; char *y = "chat";  
char *z = malloc( strlen(x) + strlen(y) + 1);  
/* mettre une chaîne vide à l'adresse z */  
*z = 0 ; /* équivalent: *z='\0' ; z[0]='\0'; z[0]=0; */  
strcat(z, x); /* concatener x à la suite de la chaîne vide */  
strcat(z, y); /* concatener y */
```

exemple strcat()

/* vecteur de

```
char *concatener(size_t n, char *mots[]){
```

```
    //calculer la somme de longueurs de tous les mots
```

```
    size_t longueur = 0;
```

```
    for( size_t i = 0; i < n; i++){
```

```
        longueur += strlen(mots[i]);
```

```
    }
```

```
    //Allouer la mémoire pour le résultat, n'oubliez pas le caractère nul
```

```
    char *p = malloc( longueur + 1 );
```

```
    p[0]='\0'; //pourquoi initialiser p comme un string vide ?
```

```
    for( size_t i = 0 ; i < n; i++ ){
```

```
        strcat(p, mots[i]);
```

```
    }
```

```
    return p;
```

```
}
```

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

`strncat()` copie depuis l'adresse `src` au plus `n` caractères à la suite de la chaîne à l'adresse `dest` (en supprimant `'\0'` terminant `dest`). La copie s'arrête avec `'\0'` dans la chaîne `src`. Le caractère nul est ajouté à la fin.

```
char s[50] = "Ala";  
char *t = "Monique";  
strncat(s, t, 3);
```

`s` initialement

A	l	a	\0	46 octets non initialisés
---	---	---	----	---------------------------

`s` après `strncat(s, t, 3)`

A	l	a	M	o	n	\0	
---	---	---	---	---	---	----	--

```
char s[50] = "Ala";  
char *t = "Monique";  
strncat(s, t, 20);
```

`s` initialement

A	l	a	\0	46 octets non initialisés
---	---	---	----	---------------------------

`s` après `strncat(s, t, 20)`

A	l	a	M	o	n	i	q	u	e	\0	
---	---	---	---	---	---	---	---	---	---	----	--

**Décoder les chaînes de
caractères avec
sscanf, scanf, fscanf**

décoder une chaîne de caractères

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

```
int scanf( const char *format, ... )  
int fscanf( FILE *stream, const char *format, ... )  
int sscanf( char *s, const char *format, ... )
```

Les trois fonctions décodent une chaîne de caractères selon le format.

scanf() – la chaîne de caractère à décoder est lue sur l'entrée standard (terminal)

fscanf() – la chaîne de caractère est lue dans un fichier

sscanf() – la chaîne de caractères à décoder est pointé par s

Ces fonctions retournent le nombre de conversions.

Les trois points ... **doivent être remplacés par des adresses.**

(Ce sont des fonctions à nombre variable d'arguments).


```
char *s=" 12 -15.5 ";
int a; double b;

sscanf( s, "%d %lf", &a, &b);
/* a == 12 b==-15.5 */
```

```
char *s = " 12 aa -32.1" ;
int a; double b;

int i=sscanf( s, "%d %lf", &a, &b);

/* a == 12, i == 1; b ne change pas, décodage s'arrête sur
 * la première lettre a */
```

```
int a; double b;

char *s = " -4 chat -11.0definition ";
#define LEN 1024
char tab[LEN];

sscanf( s, " %d %1024s %lf", &a, tab, &b);
/* a == -4 tab contient "chat" b == -11 */
```

```
int a; double b;
#define LEN 1024
char tab[LEN];

char *s = " -44  giraffe-11.0  definition ";

int  i = sscanf( s, " %d %1024s %lf", &a, tab, &b);

/* a == -44,  tab contient "giraffe-11.0", i == 2
 * b ne change pas */
```

les règles pour scanf, sscanf, fscanf

- une suite d'espaces dans le format correspond à n'importe quelle suite d'espaces (pas forcément de la même longueur), espace dans le sens isspace()
- les espaces au début sont ignorées
- %s correspond à la plus longue suite de caractères sans espace (dans le sens de isspace)
- les formats sont parfois différents que pour printf, par exemple %lf pour un double dans (s,f)scanf

lire des entiers depuis le terminal

```
int len = 1024;
/* tableau pour stocker les entiers lus */
int *tab = malloc( sizeof( int[ len ] ));
assert( tab != NULL );
int i;

for( i = 0; i < len; i++ ){
    int k = scanf("%d", &tab[i]);
    if( k == EOF || k == 0 )
        break;
}

/* i entiers dans tab, la lecture termine
   - quand on tape Ctrl-D, scanf retourne EOF
   - quand on tape un caractère différent de l'espace et
     qui ne fait pas partie d'un nombre.
   Les entiers à l'entrée séparés par n'importe quel nombre
   d'espaces, de caractères '\n', '\t' */
```

**Transformer les données en
chaîne de caractères avec
printf, fprintf, sprintf**

construire une chaîne de caractères

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

```
int printf( const char *format, ... )
```

```
int fprintf( FILE *stream, const char *format, ... )
```

```
int sprintf( char *s, const char *format, ... )
```

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

Les trois fonctions construisent une chaîne de caractères selon le format.

printf() – écrit la chaîne construite sur la sortie standard (terminal)

fprintf() – écrit la chaîne de caractère dans un fichier

sprintf() – écrit la chaîne construite à l'adresse s

Ces fonctions retournent le nombre de caractères écrits (sans compter '\0' à la fin).

Les trois points ... **doivent être remplacés par des expressions dont le type correspond** .
(Ce sont des fonctions à nombre variable d'arguments).

conversion en chaîne de caractères

```
#define LEN 1024  
char t[LEN];
```

```
int i = 35;  
double d = -44.2346 ;
```

```
char *s = "elephant giraffe";
```

```
snprintf( t, LEN, "%+d  %s = %6.2f\n", i, s, d);
```

Après l'exécution de ce code t contient la chaîne de caractères :

```
" +35  elephant giraffe = -44.23\n"
```