Langage C

Wieslaw Zielonka
zielonka@irif.fr

les caractères

```
char c, d, e;
c = 'c';
d = '\0';
e = '\n';
```

char est un type entier.

signed char char unsigned char

La variable char peut-être utilisée comme une variable entière.

Est-ce que char est un entier signé (signed char) ou non signé (unsigned char)?
Cela dépend de l'implémentation ou de l'architecture de l'ordinateur.

En C toujours `sizeof(char) == 1` (char occupe un octet).

codes ASCII

Dec	Hex	char		Dec	Hex	char	Dec	Hex	char	Dec	Hex	char
0	00	NUL	*caract. null*	32	20	espace	64	40	@	96	60	'
1	01	SOH		33	21	!	65	41	A	97	61	a
2	02	STX		34	22	"	66	42	B	98	62	b
3	03	ETX		35	23	#	67	43	C	99	63	c
4	04	EOT		36	24	$	68	44	D	100	64	d
5	05	ENQ		37	25	%	69	45	E	101	65	e
6	06	ACK		38	26	&	70	46	F	102	66	f
7	07	BEL	*bell*	39	27	‘	71	47	G	103	67	g
8	08	BS	*backspace*	40	28	(	72	48	H	104	68	h
9	09	TAB	*tabul. horiz*	41	29	)	73	49	I	105	69	i
10	0A	LF	*line feed*	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	*Tabul. vertic*	43	2B	+	75	4B	K	107	6B	k
12	0C	FF		44	2C	,	76	4C	L	108	6C	l
13	0D	CR	*retour ligne*	45	2D	-	77	4D	M	109	6D	m
14	0E	SO		46	2E	.	78	4E	N	110	6E	n
15	0F	SI		47	2F	/	79	4F	O	111	6F	o
16	10	DLE		48	30	0	80	50	P	112	70	p
17	11	DC1		49	31	1	81	51	Q	113	71	q
18	12	DC2		50	32	2	82	52	R	114	72	r
19	13	DC3		51	33	3	83	53	S	115	73	s
20	14	DC4		52	34	4	84	54	T	116	74	t
21	15	NAK		53	35	5	85	55	U	117	75	u
22	16	SYN		54	36	6	86	56	V	118	76	v
23	17	ETB		55	37	7	87	57	W	119	77	w
24	18	CAN		56	38	8	88	58	X	120	78	x
25	19	EM		57	39	9	89	59	Y	121	79	y
26	1A	SUB		58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	*escape*	59	3B	;	91	5B	[	123	7B	{
28	1C	FS		60	3C	<	92	5C	\	124	7C	|
29	1D	GS		61	3D	=	93	5D	]	125	7D	}
30	1E	RS		62	3E	>	94	5E	^	126	7E	~
31	1F	US		63	3F	?	95	5F	_	127	7F	DEL

les caractères

```
char x, y;
x = 'c';
y = 99;
```

x et y contiennent la même valeur, le code ASCII du caractère 'c' .

```
x++; y--;
/* x contient maintenant 100 : le code du caractère 'd'
   y contient 98 : le code de caractère 'b' */
```

Utilisation des propriétés de codage ASCII :

```
char x;

x = ... ;

if( 'a' <= x && x <= 'z' ){
   /* x contient une lettre minuscule, les codes de lettres
minuscule entre 'a' et 'z' */

}
Cette condition dépend de codage ASCII. Dans certaines machines
(très très rares) on pourrait avoir d'autres types de codage.
```

les caractères

```
valeur maximale de char :   CHAR_MAX
valeur minimale de char :   CHAR_MIN
```

```
Les  macro-constantes CHAR_MAX et CHAR_MIN
 sont définies dans <limits.h>

Dans le même fichier <limits.h> d'autres macro-constantes :

 INT_MAX, INT_MIN, UINT_MAX, LONG_MAX, LONG_MIN, etc.
```

les tests de catégories de caractères

les fonctions suivantes testent les propriétés de caractères et retournent une valeur int différente de 0 si le caractère passe le test et 0 sinon.
Le paramètre est un int mais sera transformé en char.

```
#include <ctype.h>

isalpha(c)    c est une lettre
iscntrl(c)    caractère de contrôle
isdigit(c)    un chiffre décimal
isalnum(c)    équivalent à isalpha(c) || isdigit(c)  est vrai
isgraph(c)    le caractère imprimable sauf espace
isprint(c)    caractère imprimable y compris l'espace
islower(c)    lettre minuscule
isupper(c)    lettre majuscule
isspace(c)    caractère blanc (whitespace) : ' ' espace, '\t' tabulation,
              '\n' newline, '\r' carriage return, '\v' vertical tab.

ispunct(c)    caractère imprimable différent de l'espace,
              des lettres et des chiffres
isxdigit(c)   un chiffre hexadécimal
isblank(c)    un caractère séparant les mots : ' ' et  '\t' (et peut-être
d'autres)
```

conversion de lettres

```
#include <ctype.h>
```

`int tolower(int c)` - convertit c en minuscule (si la conversion impossible, c'est-à-dire c n'est pas une lettre majuscule alors la fonction retourne c).

`int toupper(int c)` - convertit c en majuscule (ou retourne c si la conversion impossible)

exemple

lire les caractères à l'entrée standard mais ne sauvegarder que les lettres transformées en majuscules.

```
  char tab[NUM];

  int c;
  int i=0;

  while( i< NUM && ( c = getchar() ) != EOF ){
    if( isalpha(c) ){
      tab[i++] = toupper(c);      /* tab[i]=toupper(c); i++; */
    }
}
tab[i]='\0'; /* pour transformer en chaîne de caractères */
printf("%s\n",tab);
```

`int getchar(void)` lit un caractère et le transforme en int .
getchar() retourne EOF si l'entrée fermée par ctrl-D.

caractères

De préférence ne pas assumer que les caractères sont codés en ASCII.

par exemple pour vérifier si `char x` est un chiffre on préfère

```
isdigit(x)
```

au lieu de

```
'A' <= x &&  x <= 'Z'
```

chaînes de caractères

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

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

Oui: les chaînes de caractères représentées par le pointeur char * pointant vers le premier caractère d'une suite de caractères.

Une convention (définition):

> *une chaîne de caractères est une suite de caractères qui termine par le caractère `nul '\0'`.*

```
Le caractère '\0' est un caractère dont tous les
bits sont 0, ce n'est pas le caractère '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 ne fait pas partie de la chaîne**, c'est un marquer de la fin de la chaîne
- la longueur d'une chaîne c'est le nombre de caractères avant le caractère nul '\0'
- une chaîne de caractère ne contient jamais de caractère nul '\0'
- en anglais on parle de **null-terminated string**

suite de caractères :

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

chaînes de caractères versus suites de caractères

Si on a un pointeur

char *

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

Impossible en regardant les caractères à partir de l'adresse char *.

A priori quand nous avons une fonction

type_retour fonct(char *x, ...)

c'est la documentation qui doit indiquer si char * est une chaîne de caractères ou juste un pointeur vers une suite de caractères quelconques.

Mais presque toujours un paramètre de type char * est une chaîne de caractères dans le sens de C.

chaîne de caractères

Si le prototype d'une fonction a une forme

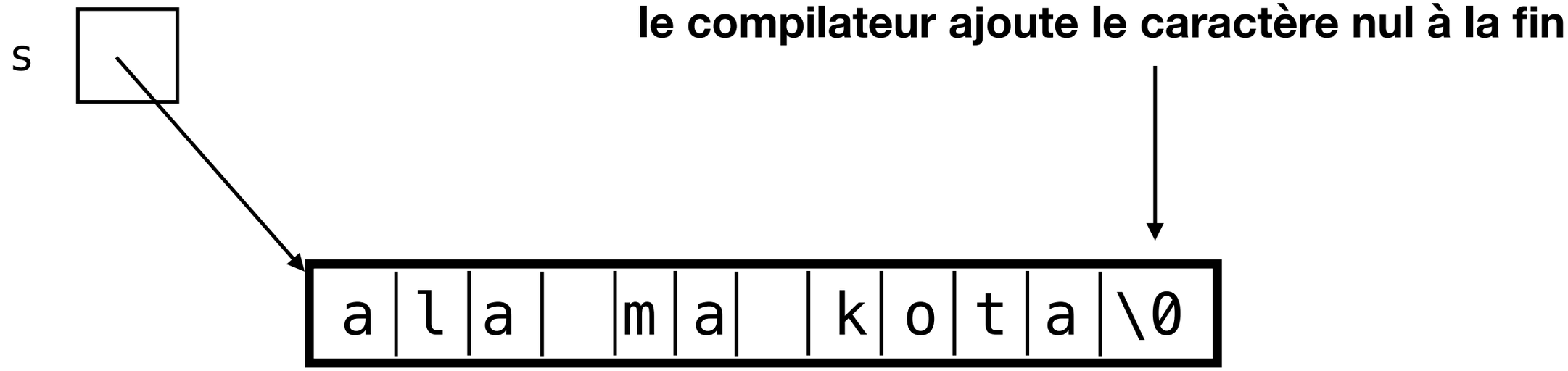
```
type_retour f(char *s, ...)
```

et la description indique que s doit être une "chaîne de caractères" (null-terminated 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 est indéfini (la fonction calcule n'importe quoi comme le résultat ou l'exécution s'arrête à cause d'exception).

chaînes de caractères

```
char *s ;

s = "ala ma kota";
```




```
printf("%c",s[4]);          OK, imprime  m
```

La chaîne qui apparaît explicitement dans le text du programme est stockée dans la mémoire non modifiable du programme.

```
s[4] = 'z';
```

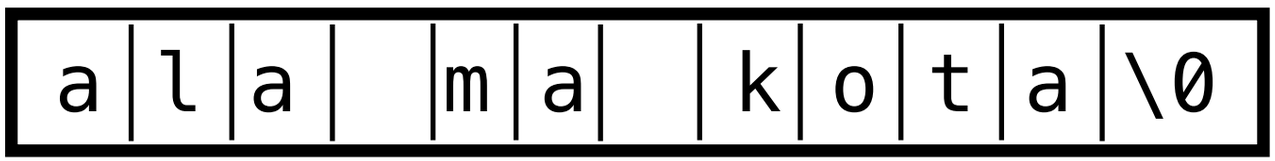
erreur d'exécution (exception)

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'.

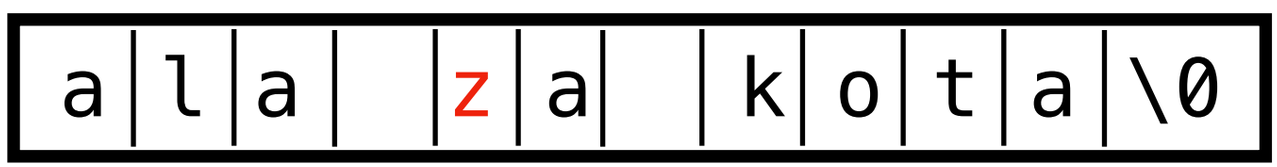
tab




```
printf("%c",tab[4]);          OK, imprime  m
```

```
tab[4] = 'z';                 OK,
```

tab



<string.h>

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

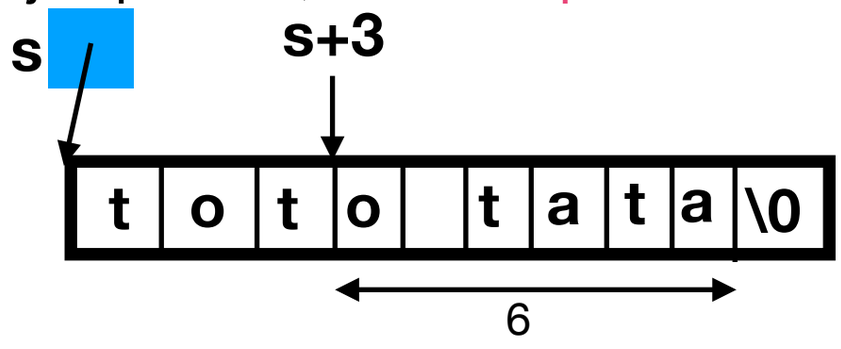
Leurs noms commencent par str.

size_t strlen(char *s)

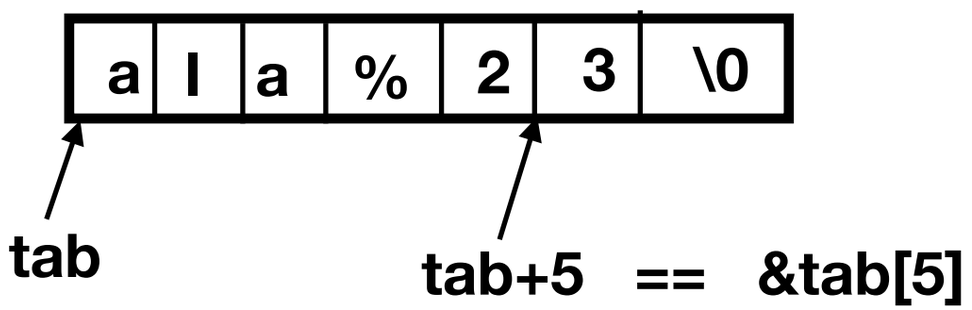
```
size_t strlen(char *s)
```

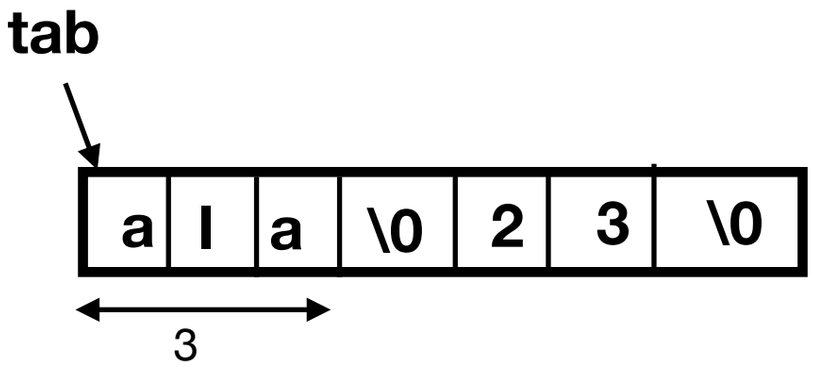
retourne la longueur de la chaîne s (le nombre de caractère jusqu'à '\0', sans compter '\0'.

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

size_t t = strlen(s);  la valeur de t == 9

t = strlen(s+3);       la valeur de t == 6
```




```
char tab[] = "ala%23";

t = strlen( &tab[0] );    t vaut 6

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

tab[3] = '\0';

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





exemple : implémenter strlen()soi même

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

La fonction strlen() **ne peut pas** vérifier si s est une chaîne de caractères.

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

```
size_t k = strlen(s);
```

cette appel à strlen() soit provoque une exception soit renvoie un résultat aléatoire

int strcmp(char *s, char *t)

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

la fonction retourne

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

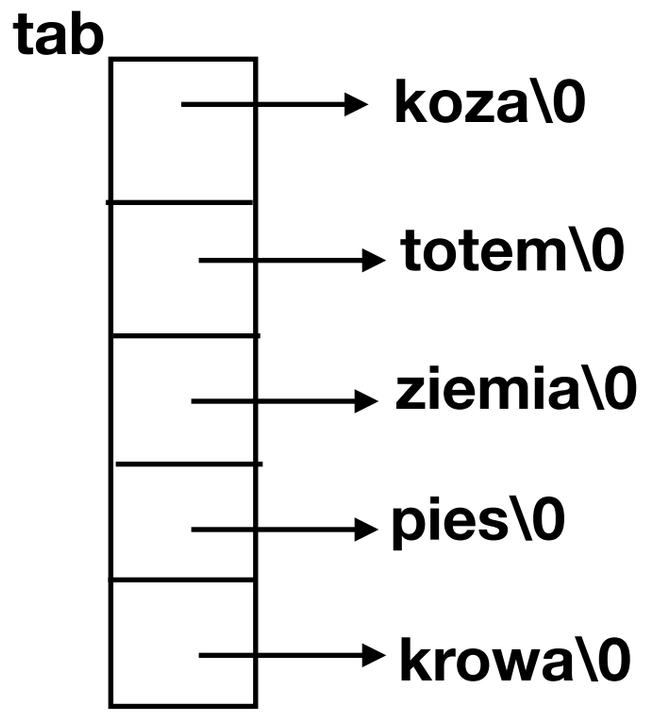
Implémentation possible de `strcmp() :`

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

exemple

Trouver et afficher le mot le plus grand dans l'ordre lexicographique dans un vecteur de mots.

```
char *tab[ ] = {"koza", "totem", "ala",

                "ziemia", "pies", "krowa"};

  size_t len = sizeof(tab)/sizeof(tab[0]);

  int m = 0;

for( int i = 1; i < len; i++){

   if( strcmp(tab[i],tab[m]) > 0 )

     m = i;

}

printf("%s\n",tab[m]);
```



vecteur de strings est un tableau de pointeurs char *

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

`char *strcpy(char *dest, 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;

s = "toto";

char *dest = malloc( strlen(s) + 1 );

strcpy(dest, s);
```

Très important :
n'oubliez +1 pour avoir la place pour le caractère nul '\0'

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

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

Copie au plus n octets 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 qu'il est possible après l'appel à `strncpy` dest ne pointe pas vers une chaîne qui ne termine avec '\0'.

```
char *source =  "abcdef";  /* la chaîne source */
char tab[4];

char mot[8];

strncpy(tab, source, 4);

/* tab contient maintenant 'a','b','c','d' */

strncpy(mot, source, 8);

/* mot contient maintenant 8 caractères :   a b c d \0 \0 \0 \0  */
```

encore quelques fonctions : chaînes de caractères

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

concatène src à la suite de dest, retourne s.

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 = "dog";
strcat(x,y);
```

débordement de mémoire

CORRECT :

```
char *x = "chien";
char *y = "dog";
char *z = malloc( strlen(x) + strlen(y) + 1);
strcpy(z, x); //copier x dans z
strcat(z, y); //concatener y à la suite
```

exemple strcat()

```
//concatener tous les mots d'un vecteur mots
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;
}
```

exemple strcat()

Utilisation de la fonction concatener()

```
int main(){

   char *mots[ ] ={ "lama", "chien", "chat", "chevre"};

   //calculer les nombres de mots
    size_t len = sizeof(mots)/sizeof(mots[0]);

    char *res = concatener( len, mots);

....

}
```

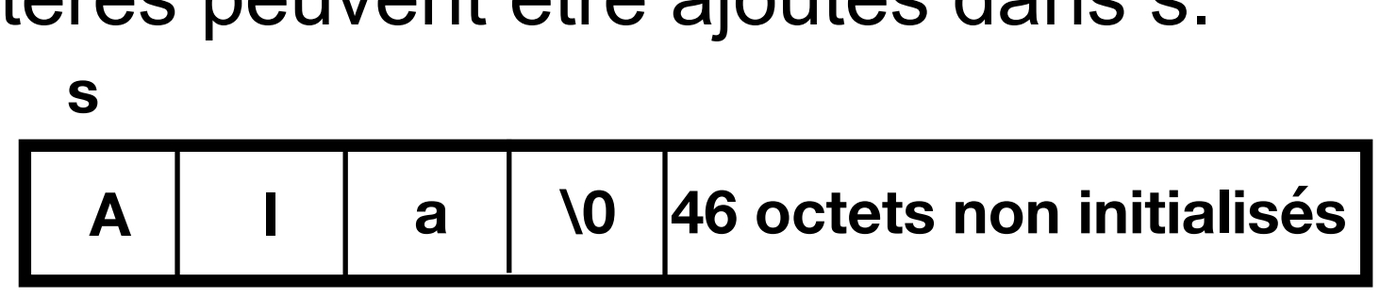
encore quelques fonctions : chaînes de caractères

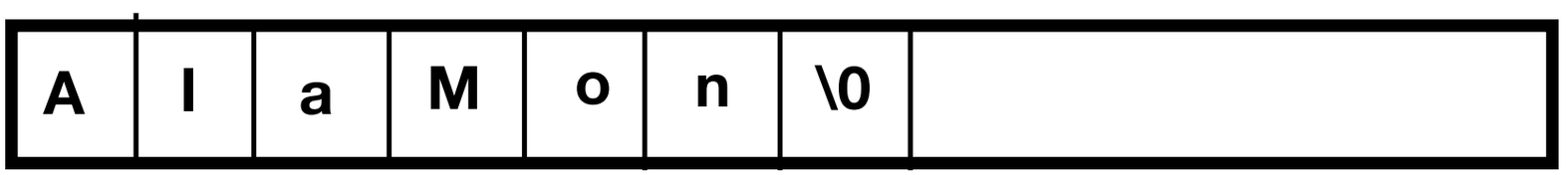
```
char *strncat(char *s, char *t, size_t n)
```

`strncat()` copie au plus n caractères à la suite de la chaîne s (en supprimant '\0' terminant s). La copie s'arrête avec '\0' dans la chaîne t. Le caractère nul est ajouté à la fin.

Donc au total n+1 caractères peuvent être ajoutés dans s.

```
char s[50] = "Ala";

chat *t = "Monique";

strncat(s, t, 3);      //le vecteur   s contient maintenant
```





encore quelques fonctions : chaînes de caractères

Chercher un caractère dans une chaîne de caractère :

```
char * strchr(const char *s, int c)

char * strrchr(const char *s, int c)
```

`strchr()` retourne l'adresse de la première occurrence du caractère c dans s ou NULL si s ne contient pas c

`strrchr()` retourne l'adresse de la dernière occurrence du caractère c dans s ou NULL si s ne contient pas c

Chercher un string dans un string :

```
char *strstr(const char *src, const char *sub)
```

la fonction retourne le pointeur vers la première occurrence du string `sub` dans le string `src` (ou NULL si sub n'apparaît pas dans src).

strtok()

`char *strtok(char *s, const char *t)`

décompose s en sous-chaînes : lexèmes (tokens). La chaîne `s` est modifiée par les appels à `strtok()`.

Le premier appel se fait avec s différent de NULL.

La fonction trouve le premier lexème composé de caractères n'appartenant pas à `t` et elle remplace le caractère suivant le lexème par `'\0'` et retourne le pointeur vers le lexème.

Dans les appels suivant on mettra s égal à NULL et la fonction retourne le pointeur vers le lexème suivant.

La fonction retourne NULL s'il n'y a plus de lexèmes.

Il est possible de changer t à chaque appel.

```
char u[]="toto tuti tata?";
char *e=" ?";
char *q=strtok(u,e);
if( !q )
    return 0;
do{
    printf("\"%s\"\n", q);
    q=strtok(NULL,e);
}while(q);
printf("\"%s\"\n",u);
```

les printf dans la boucle affichent :
"toto"
"tuti"
"tata"

Notez qu'à la fin u est modifié et le printf après la boucle affiche "toto"

Notez aussi que le string u doit être modifiable.

strtok()

Souvent c'est plus facile d'écrire votre propre parseur que

utiliser strtok()

convertir un chaîne de char en nombre

```
#include <stdlib.h>

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

Ces fonctions ne prennent pas en compte des caractères d'espacement au début et arrêtent la conversion quand elles rencontrent un caractère qui ne fait pas partie d'un nombre.

```
int i = atoi("  \n  23nkl");   /* i <-  23 */
double d = atof("  \ntoto23  ");  /* d <-  0  */

Dans le deuxième exemple la recherche de nombre s'arrête
sur le caractère t.
double e = atof("    34.99abc ");  /* e <- 34.99 */
```

les paramètres de main()

```
int main(void){  } ou

int main(){ }


int main( int argc, char *argv[] ){ }

équivalent à

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

argc est le nombre d'éléments de vecteurs argv.

quel est le contenu de vecteur argv[]   ?

les paramètres de main()

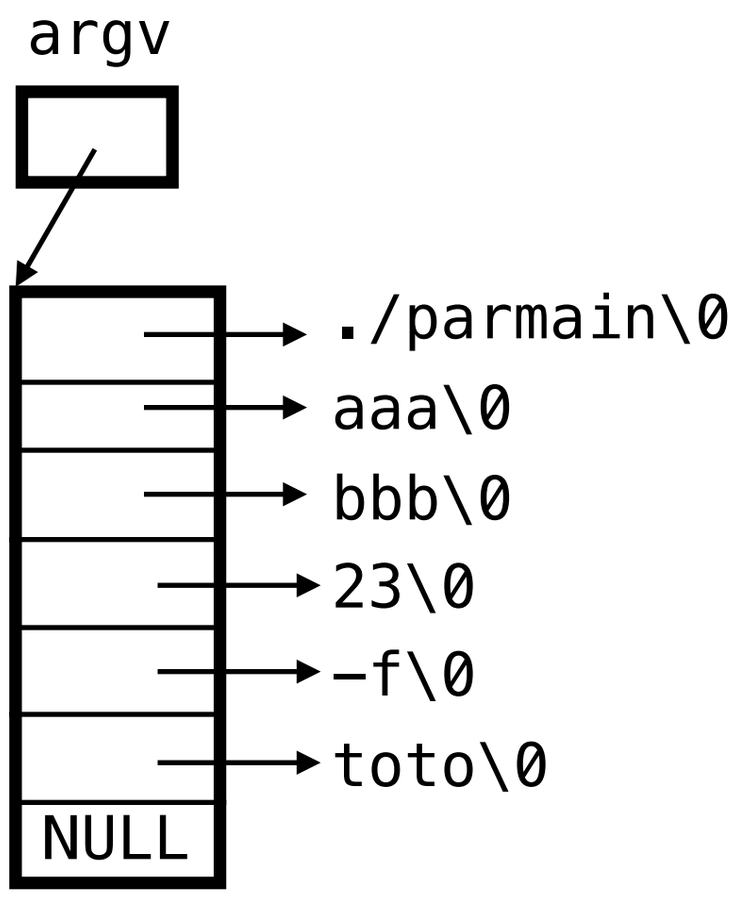
```
/*  parmain.c   */
int main(int argc, char *argv[]){

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

après la compilation nous l'exécutons parmain depuis le terminal :

./parmain aaa bbb 23 -f toto

le programme affichera sur le terminal :

```
./parmain
aaa
bbb
23
-f
toto
```



argv[0] contient le nom du programme (le nom de la commande)

argv[1], argv[2] etc contiennent les paramètres de la commande

exemple

Transformer les paramètres de main en nombres double, faire la somme et imprimer.

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

  if( argc == 1 ){
    fprintf(stderr, "usage: %s nombre [nombres]\n",
            basename(argv[0]) );
    exit(1);
  }
 double s;
 int i;

 for( s=0, i=1; i < argc; i++ )
    s += atof(argv[i]);

  printf("somme = %6.2f\n",s);
  return EXIT_SUCCESS;
}
```

stderr - sortie d'erreur standard

`char *basename(char *s)`
retourne le dernier élément d'un chemin. Exemple : `basename("rep1/rep2/file")` retourne "file"

Sur le terminal on compile et on exécute :

gcc -Wall -g somme.c -o somme
./somme -7.9 12.3 9.6

la commande affiche sur le terminal :

somme = 14.0

valeur de retour de main

```
int main(int arc , char **argv){


     return 0;  //ou exit(0);

}
```

Dans `main()`

`return i;`
aura le même résultat que `exit(i);`
`(sauf si main() est appelé depuis une autre fonction ou récursivement -- oui c'est possible).`

Mais on peut aussi quitter main sans **`exit()`** ni **`return,`** juste parce que on a exécuté la dernière instruction de `main().` Le résultat est le même que da faire `return 0;`
(Mais c'est valable juste pour `int main()` et pour aucune autre fonction qui retourne `int.`)

Comment voir le code de retour depuis le terminal? Tapez

`echo $?`

sur le terminal. Cela affiche la valeur de la variable `? de shell –` le code de retour de la dernière commande exécutée sur le terminal.

décoder une chaîne de caractères avec scanf

```
#include <stdio.h>

int sscanf(char *s, char *format, ...)
```

`sscanf` fait la même chose que `scanf` mais les caractères
à l'entrée sont extraits de la chaîne s et non de l'entrée standard.

Exemple:

```
char *s="12    -15";
int a,b;
sscanf( s, "%d%d",  &a, &b );
```

sscanf met 12 dans a et -15 dans b

conversion en chaîne de caractères

```
#include <stdio.h>

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

`sprintf` fait la même chose que `printf` mais les résultat est mis à l'adresse `s` au lieu de l'écrire sur le terminal, sprintf construit une chaîne de caractères à l'adresse s

les trois codes de format correspondent aux trois expressions

%c %-d %s

```
#define BUF_SIZE 100
  char mot[BUF_SIZE];

  char format[10];
  sprintf(format, "%c%-d%s"  , '%',  BUF_SIZE, "s");
  /* maintenant format contient  la chaîne  "%100s"   */
  size_t k;
  while( ( k = scanf(format, mot) ) != EOF ){

  }
```

le format %-d indique le résultat doit être aligné à gauche

chaînes de caractères littérales dans le texte de programme

```
char *s = "tres tres tres tres tres longue "

          "suite de caractères";
```

équivalent à

```
char *s = "tres tres tres tres tres longue suite de caractères";
```

Le preprocesseur concatène les chaînes de caractères littérales consécutives qui apparaissent dans le texte du programme. Très commode avec printf avec long format :

```
printf(" un très long format  ......                                    "

        " et la suite du même format ...                                  ",

      i, j ,    ...  );
```