



# Langage C – Cours 5

Lélia Blin

[lelia.blin@irif.fr](mailto:lelia.blin@irif.fr)

2023 - 2024

# Les chaines de caractères

# Les littéraux chaînes de caractères

- En C, il n'y a pas à proprement parlé un type chaîne de caractères.
- Mais il y a des littéraux chaînes de caractères, par exemple :

```
"Hello\n", "Blabla blabla", "1234\n256"
```

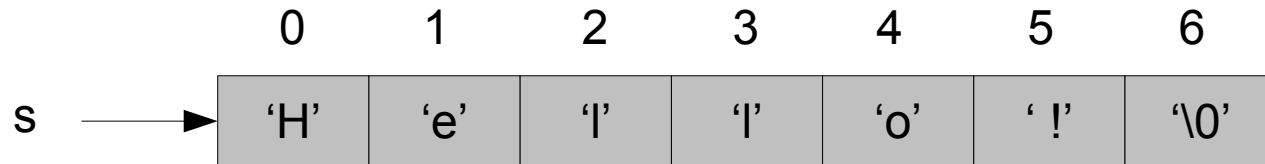
- Moralement ils correspondent à un tableau de caractères dont le dernier élément est **le caractère spécial '\0'**.
- Ces littéraux chaînes de caractères ne peuvent pas être changés tels quels, ils sont 'read-only'
  - On verra par la suite qu'en fait on peut changer l'intérieur de chaînes de caractères, mais il faudra faire attention d'où sont stockés ces chaînes.
  - Le compilateur ne détecte pas de telles modifications interdites
  - Si on essaie de les modifier tel quel, on aura à l'exécution une erreur **bus error**

# Les littéraux chaînes de caractères

- En pratique, pour les stocker dans une variable, on peut faire :

```
char *s="Hello!"
```

- Dans ce cas, le pointeur s pointe vers une zone de la mémoire **read-only** (non modifiable)
- Cette zone est une zone de 7 caractères dont le dernier caractère est '\0'



**NON MODIFIABLE**

- Attention :** le caractère '\0' est différent du caractère '0'
- En revanche, on peut lire les éléments de la zone

# Exemple

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

int main () {
    char *test="abcd";
    for (char *p=test;p<test+4;++p){
        printf("%c ",*p);
    }
    printf("\n");
    return EXIT_SUCCESS;
}
```

string-literal.c

# Exemple

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

int main () {
    char *test="abcd";
    *(test+1)='f';
    printf("%s \n",test);
    return EXIT_SUCCESS;
}
```

string-literal2.c

Génère à l'exécution une error : **bus error**  
-> modification d'un endroit de la mémoire non autorisé

# Exemple

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

void change(char *st){
    *st='E';
}

int main () {
    char *test="abcd";
    change(test);
    for (char *p=test;p<test+4;++p){
        printf("%c ",*p);
    }
    printf("\n");
    return EXIT_SUCCESS;
}
```

string-literal3.c

**Génère à l'exécution une error : bus error**  
**-> modification d'un endroit de la mémoire non autorisé**

# Utilisation de tableaux

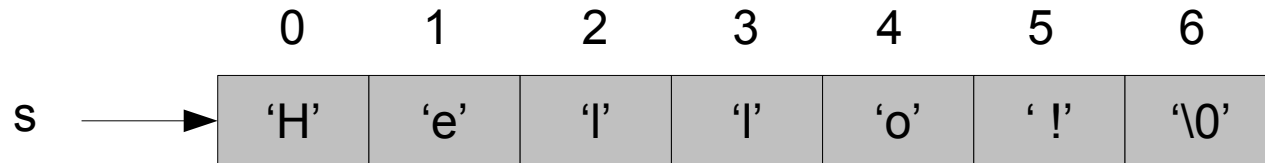
- On peut aussi stocker un littéral chaîne de caractères dans un tableau :

```
char s[7]="Hello!";
```

- Ou :

```
char s[]="Hello!";
```

- Il faut faire attention que le tableau soit assez grand et laisser de la place à la fin pour le caractère '\0'
- En pratique, la chaîne de caractères (de longueur L) est copiée dans les cases d'un tableau de taille L+1 et dans la (L+1)-ème case, le caractère '\0' est aussi copié.
- Cette zone est une zone de 7 caractères dont le dernier caractère est '\0'



- Avantage :** on peut modifier les cases du tableau
- Inconvénient :** s indique tableau et donc son utilisation est restreint (on ne peut pas faire d'affectation par exemple)



# Exemple

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

int main () {
    char test[5]="abcd";
    *test='E';
    *(test+1)='F';
    for (char *p=test;p<test+4;++p){
        printf("%c ",*p);
    }
    printf("\n");
    return EXIT_SUCCESS;
}
```

string-tab.c

# Tableaux de chaînes de caractères

- On peut faire des tableaux de chaînes de caractères :

```
char *s[3]={"Hello!","World","Monde"};
```

- Dans ce cas, chaque case du tableau contient un 'pointeur' vers une chaîne de caractères non modifiable
- Quand on le passe en argument à une fonction on aura cette fois ci-bien le type `char **` car il s'agit bien d'un tableau de pointeurs !

# Exemple

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

void f(char **t){
    for(size_t i=0;i<3;++i){
        printf("%s\n",*(t+i));
    }
}

int main () {
    char *test[3]={"abcd","blabla","hello"};
    f(test);
    return EXIT_SUCCESS;
}
```

string-tab-tab.c

# Exemple

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

void f(char **t){
    for(size_t i=0;i<3;++i){
        printf("%s\n",*(t+i));
    }
}

int main () {
    char *test[3]={"abcd","blabla","hello"};
    *(*test)='C';
    f(test);
    return EXIT_SUCCESS;
}
```

string-tab-tab2.c

Génère à l'exécution une error : **bus error**

# En pratique

- On utilisera pour manipuler des chaînes de caractères
  - des littéraux chaînes de caractères
    - En se rappelant qu'ils sont non modifiables
  - Des tableaux de caractères
  - Des zones dans le tas mémoire allouées dynamiquement contenant des caractères
- Il faudra faire attention au point suivant :
  - **Quand on encode/manipule une chaîne de caractères, on suppose toujours que**
    - **Elle ne contient pas le caractère '\0'**
    - **Elle se termine par le caractère '\0'**
- Toutes les fonctions manipulant des chaînes de caractères prennent en compte ces hypothèses et ont un comportement non spécifiés si les hypothèses ne sont pas respectés
- Un tableau de caractères quelconque peut lui très bien contenir plusieurs occurrences de '\0' mais il ne faut pas le considérer comme une chaîne

# Exemple

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

int main () {
    char tab[4]={'A','\0','B','\0'};
    printf("%s \n",tab);
    char tab2[4]="ABC";
    printf("%s \n",tab2);
    return EXIT_SUCCESS;
}
```

tab-vs-string.c

```
>../tab-vs-string
A
ABC
```

**Dans le printf,%s attend une chaîne de caractères et s'arrête donc pour tab au premier '\0'**

# Manipulation

- La plupart des fonctions de manipulation de chaînes de caractères se trouvent dans **<string.h>**
- Certaines fonctions modifient la chaîne donnée entrée et d'autres non
- Pour les fonctions qui ne modifient pas la chaîne dans la signature de la fonction on a le mot clef const
- Par exemple, fonction pour connaître la taille du chaîne :
  - **size\_t strlen(const char \*s);**
- Cette fonction renvoie le nombre de caractère dans la chaîne sans prendre en compte le dernier caractère '\0'

```
char *s="Hello!"  
size_t taille=strlen(s);
```

- Dans taille on aura la valeur 6.

# Exemple

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

int main () {
    char *st="ABRA CADA BRA!";
    size_t lo=strlen(st);
    printf("%s Taille : %zu\n",st,lo);
    lo=strlen(st+4);
    printf("%s Taille : %zu\n",st+4,lo);
    char st2[]="ABCD";
    lo=strlen(st2);
    printf("%s Taille : %zu\n",st2,lo);
    *(st2+2)='\0';
    lo=strlen(st2);
    printf("%s Taille : %zu\n",st2,lo);
    return EXIT_SUCCESS;
}
```

string-long.c



# Reprogrammer strlen

```
size_t longueur(char *);

int main () {
    char *st="ABRA CADA BRA!";
    size_t lo=longueur(st);
    printf("%s Taille : %zu\n",st,lo);
    return EXIT_SUCCESS;
}

size_t longueur(char *s){
    size_t i=0;
    while(*s){
        ++s;
        ++i;
    }
    return i;
}
```

string-long2.c

# Copie de zone mémoire

## Rappel du dernier cours

- On peut copier des zones mémoire, et on a deux fonctions (elle sont **dans <string.h>**) :
  - `void *memmove(void *dst, const void *src, size_t len)`
  - `void *memcpy(void *dst, const void *src, size_t len)`
- Elles copie toutes les deux len octets de src vers dst
- **Les deux zones pointées doivent être allouées et de la bonne taille** (inférieure ou égale à len)
- Elles renvoient toutes les deux dst
- Différence :
  - Pour memcpy, les deux zones pointées par dst et src ne doivent pas se chevaucher !

# Exemple

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

int main () {
    char *st="ABRA CADA BRA!";
    char *st2=" MAGIE!!!";
    char*st3=malloc((strlen(st)+strlen(st2)+1)*sizeof(char));
    memcpy(st3,st,strlen(st)*sizeof(char));
    memcpy(st3+strlen(st),st2,(strlen(st2)+1)*sizeof(char));
    printf("%s\n",st3);
    return EXIT_SUCCESS;
}
```

string-concat.c

# Exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

char *concat_tab(char **,size_t n);

int main () {
    char *st[3]={"ABRA","CADA","BRA!"};
    char *st2=concat_tab(st,3);
    printf("%s\n",st2);
    return EXIT_SUCCESS;
}

/*Fonction qui concatene les chaines*/
/*D'un tableau de chaine de caracteres de taille n*/
char *concat_tab(char **t,size_t n){
    size_t taille=0;
    for(char **tmp=t;tmp<t+n;++tmp){
        taille+=strlen(*tmp);
    }
    char *res=malloc((taille+1)*sizeof(char));
    assert(res!=NULL);
    size_t pos=0;
    for(char **tmp=t;tmp<t+n;++tmp){
        memcpy(res+pos,*tmp,strlen(*tmp));
        pos+=strlen(*tmp);
    }
    *(res+taille)='\0';
    return res;
}
```

# La fonction strcat

- La fonction
  - `char *strcat(char *restrict s1, const char *restrict s2);`
- Elle concatène la chaîne de caractères s2 à la fin de la chaîne de caractères s1 (en mettant le '\0' à la fin de la concaténation)
- Renvoie le pointeur s1
- Il faut que les deux chaînes se terminent par '\0'
- Il faut qu'il y ait assez de place dans la zone pointée par s1
- **ATTENTION** aux débordement en utilisant cette fonction (si il n'y a pas assez de place dans s1 par exemple)
- Sa variante
  - `char *strncat(char *restrict s1, const char *restrict s2, size_t n);`
  - Copie au plus n caractères de s2 et ajoute '\0' ensuite

# Exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

int main () {
    char *s="HELLO";
    char *s2=" BOB ";
    char *s3="OR WORLD!";
    size_t taille=strlen(s)+strlen(s2)+strlen(s3);
    char *st=malloc((taille+1)*sizeof(char));
    *st='\0';
    strcat(st,s);
    strcat(st,s2);
    strcat(st,s3);
    printf("%s\n",st);
}
```

string-strcat.c

# Exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

int main () {
    char *s="HELLO";
    char *s2=" BOB ";
    char *s3="OR WORLD!";
    size_t taille=7;
    char *st=malloc((taille+1)*sizeof(char));
    *st='\0';
    strncat(st,s,2);
    strncat(st,s2,3);
    strncat(st,s3,2);
    printf("%s\n",st);
}
```

string-strncat.c

# Copie de chaînes de caractères

- La fonction
  - `char *strcpy(char *dst, const char *src);`
- Elle copie la chaîne pointée par src dans dst, y compris le caractère de fin de chaîne '\0'
- Elle renvoie le pointeur dst
- Il faut que dst pointe vers une zone allouée modifiable
- Il faut qu'il y ait assez de place dans la zone pointée par dst
- Sa variante
  - `char *strncpy(char * dst, const char * src, size_t len);`
  - Copie au plus len caractères de s2
  - Si src à moins de len caractères, elle complète en rajoutant des '\0'
  - Sinon, il se peut que la chaîne copiée n'est pas de caractères de fin de chaîne



# Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

int main () {
    char *st=malloc(100*sizeof(char));
    strcpy(st,"HELLO");
    printf("%s\n",st);
    return EXIT_SUCCESS;
}
```

string-strcpy.c

# Exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

int main () {
    char *st=malloc(100*sizeof(char));
    strcpy(st,"HELLO");
    printf("%s\n",st);
    strcpy(st,"On met une phrase longue");
    printf("%s\n",st);
    strncpy(st,"HELLO",3);
    printf("%s\n",st);
    strcpy(st,"HELLO");
    printf("%s\n",st);
    return EXIT_SUCCESS;
}
```

string-strcpy2.c

```
>../string-strcpy2
HELLO
On met une phrase longue
HELmet une phrase longue
HELLO
```

# Comparaison de chaînes de caractères

- Pour tester si deux chaînes de caractères sont égales :
  - `int strcmp(const char *s1, const char *s2);`
- Cette fonction renvoie 0 si les chaînes pointées par s1 et s2 sont égales
- Si les deux chaînes sont différentes, elle va renvoyer :
  - un entier strictement positif si s1 est strictement plus grande que s2
  - un entier strictement négatif si s1 est strictement plus petite que s2
- Qu'est ce que veut dire s1 strictement plus grande que s2 ?
  - On prend l'ordre lexicographique (en gros l'ordre du dictionnaire)
  - Si les caractères de s1 sont a1 a2 ... ak et les caractères de s2 sont b1 ... bn alors s1 > s2 si et seulement si
    - Il existe i entre 1 et k tel que :
      - soit i=n+1 et a1=b1,..., a(i-1)=b(i-1)
      - soit i<=n et ai>bi et a1=b1,..., a(i-1)=b(i-1)
- La comparaison entre caractères est celle du dictionnaire (ie 'a'<'b'<'c'...) et les lettres majuscules sont plus **petites** que les lettres minuscules.

# Exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

int main () {
    char *st1=malloc(30*sizeof(char));
    char *st2=malloc(100*sizeof(char));
    strcpy(st1,"HELLO");
    strcpy(st2,"AB");
    int diff=strcmp(st1,st2);
    if(diff==0){
        printf("%s égale %s\n",st1,st2);
    }else if(diff>0){
        printf("%s plus grande que %s\n",st1,st2);
    }else if(diff<0){
        printf("%s plus petite que %s\n",st1,st2);
    }
    return EXIT_SUCCESS;
}
```

string-strcmp.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

int main () {
    char *st1=malloc(30*sizeof(char));
    char *st2=malloc(30*sizeof(char));
    strcpy(st1,"");
    strcpy(st2,"");
    while(strcmp(st1,"QUIT")!=0){
        puts("Entrez chaine 1");
        char *ret=fgets(st1,29,stdin); //lit 29 caractère de stdin (peut mettre le \n)
        assert(ret!=NULL);
        if(st1[strlen(st1)-1]=='\n'){
            st1[strlen(st1)-1]='\0';
        }

        if(strcmp(st1,"QUIT")==0){
            break;
        }
        puts("Entrez chaine 2");
        ret=fgets(st2,29,stdin);
        assert(ret!=NULL);
        if(st2[strlen(st2)-1]=='\n'){
            st2[strlen(st2)-1]='\0';
        }

        int diff=strcmp(st1,st2);
        if(diff==0){
            printf("%s égale %s\n",st1,st2);
        }else if(diff>0){
            printf("%s plus grande que %s\n",st1,st2);
        }else if(diff<0){
            printf("%s plus petite que %s\n",st1,st2);
        }
    }
    return EXIT_SUCCESS;
}

```

string-strcmp2.c

# Traduction

- Pour traduire une chaîne de caractères en un entier
  - `int atoi(const char *str);`
- Cette fonction renvoie un entier
- Elle ne prend pas en compte les caractères d'espacement ' ', '\n', '\r' au début
- Elle s'arrête au premier caractère qui n'est pas un chiffre

# sprintf

- La fonction :
  - `int sprintf(char * restrict str, const char * restrict format, ...);`
- Même principe que printf sauf qu'elle met le résultat dans la chaîne str

```
char *s=malloc(30*sizeof(char));  
int x=10 ;  
sprintf(s,"L'entier %d",x);
```

- Renvoie une valeur négatif en cas de problème

# Exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

int main () {
    char *s=malloc(30*sizeof(char));
    int x=10;
    int r=sprintf(s,"L'entier %d",x);
    assert(r>=0);
    printf("%s\n",s);
    return EXIT_SUCCESS;
}
```

string-sprintf.c



# Fonction de test sur les caractères

Name	Meaning	C locale	Extended
<b>islower</b>	Lowercase	'a' ... 'z'	Yes
<b>isupper</b>	Uppercase	'A' ... 'Z'	Yes
<b>isblank</b>	Blank	'_', '\t'	Yes
<b>isspace</b>	Space	'_', '\f', '\n', '\r', '\t', '\v'	Yes
<b>isdigit</b>	Decimal	'0' ... '9'	No
<b>isxdigit</b>	Hexadecimal	'0' ... '9', 'a' ... 'f', 'A' ... 'F'	No
<b>isctrl</b>	Control	'\a', '\b', '\f', '\n', '\r', '\t', '\v'	Yes
<b>isalnum</b>	Alphanumeric	<b>isalpha</b> (x)    <b>isdigit</b> (x)	Yes
<b>isalpha</b>	Alphabet	<b>islower</b> (x)    <b>isupper</b> (x)	Yes
<b>isgraph</b>	Graphical	(! <b>isctrl</b> (x)) && (x != '_')	Yes
<b>isprint</b>	Printable	! <b>isctrl</b> (x)	Yes
<b>ispunct</b>	Punctuation	<b>isprint</b> (x) && !( <b>isalnum</b> (x)    <b>isspace</b> (x))	Yes

.....

Pris de Modern C de Jens Gustedt

**renvoient 0 si le test échoue et une valeur  
différente sinon**