Initiation à l'algorithmique
Les chaînes de caractères

Mohamed MESSABIHI

mohamed.messabihi@gmail.com

Université de Tlemcen
Département d'informatique
1ère année MI

https://sites.google.com/site/informatiquemessabihi/

Le type Char

- Le type char permet de stocker des nombres allant de -128 à 127.
- Un caractère est une variable de type Char qui prend 1 octet.
- Comme la mémoire ne peut stocker que des nombres, on a inventé une table qui fait la conversion entre les nombres et les lettres. Ainsi par exemple que le nombre 65 équivaut à la lettre A.
- En C, on peut travailler sur un caractère à partir de son numéro dans la table ASCII, il suffit d'écrire cette lettre entre apostrophes, comme ceci : 'A'. 'A' sera donc remplacé par la valeur correspondante : 65.

Exemple

```c
int main()
{
    char lettre = 'A';
    printf("%d\n", lettre);
    return 0;
}
```

Le type Char

- Le type char permet de stocker des nombres allant de -128 à 127.
- Un caractère est une variable de type Char qui prend 1 octet.
- Comme la mémoire ne peut stocker que des nombres, on a inventé une table qui fait la conversion entre les nombres et les lettres. Ainsi par exemple que le nombre 65 équivaut à la lettre A.
- En C, on peut travailler sur un caractère à partir de son numéro dans la table ASCII, il suffit d'écrire cette lettre entre apostrophes, comme ceci : 'A'. 'A' sera donc remplacé par la valeur correspondante : 65.

Exemple

```
int main()
{
    char lettre = 'A';
    printf("%d\n", lettre);
    return 0;
}
```

Le type Char

- Le type char permet de stocker des nombres allant de -128 à 127.
- Un caractère est une variable de type Char qui prend 1 octet.
- Comme la mémoire ne peut stocker que des nombres, on a inventé une table qui fait la conversion entre les nombres et les lettres. Ainsi par exemple que le nombre 65 équivaut à la lettre A.
- En C, on peut travailler sur un caractère à partir de son numéro dans la table ASCII, il suffit d'écrire cette lettre entre apostrophes, comme ceci : 'A'. 'A' sera donc remplacé par la valeur correspondante : 65.

Exemple

```c
int main()
{
    char lettre = 'A';
    printf("%d\n", lettre);
    return 0;
}
```

Le type Char

- Le type char permet de stocker des nombres allant de -128 à 127.
- Un caractère est une variable de type Char qui prend 1 octet.
- Comme la mémoire ne peut stocker que des nombres, on a inventé une table qui fait la conversion entre les nombres et les lettres. Ainsi par exemple que le nombre 65 équivaut à la lettre A.
- En C, on peut travailler sur un caractère à partir de son numéro dans la table ASCII, il suffit d'écrire cette lettre entre apostrophes, comme ceci : 'A'. 'A' sera donc remplacé par la valeur correspondante : 65.

Exemple

```
int main()
{
    char lettre = 'A';
    printf("%d\n", lettre);
    return 0;
}
```

Le type Char

- Le type char permet de stocker des nombres allant de -128 à 127.
- Un caractère est une variable de type Char qui prend 1 octet.
- Comme la mémoire ne peut stocker que des nombres, on a inventé une table qui fait la conversion entre les nombres et les lettres. Ainsi par exemple que le nombre 65 équivaut à la lettre A.
- En C, on peut travailler sur un caractère à partir de son numéro dans la table ASCII, il suffit d'écrire cette lettre entre apostrophes, comme ceci : 'A'. 'A' sera donc remplacé par la valeur correspondante : 65.

Exemple

```
int main()
{
    char lettre = 'A';
    printf("%d\n", lettre);
    return 0;
}
```

Table ASCI

- La table ASCII est un tableau de 256 caractères, numérotés de 0 à 255.
- La plupart des caractères « de base » sont codés entre les nombres 0 et 127.

Le code ASCII

American Standard Code for Information Interchange

ASCII control characters

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

ASCII printable characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

theASCIIcode.com.ar

Extended ASCII characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ò	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

Table ASCI

- La table ASCII est un tableau de 256 caractères, numérotés de 0 à 255.
- La plupart des caractères « de base » sont codés entre les nombres 0 et 127.

Le code ASCII

American Standard Code for Information Interchange

ASCII control characters

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

ASCII printable characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

theASCIIcode.com.ar

Extended ASCII characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ò	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	▀	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

Lire et afficher un caractère

- On peut demander à l'utilisateur d'entrer une lettre en utilisant le %c dans un scanf (c comme caractère).
- Pour afficher un caractère on doit également utiliser le symbole %c :

Exemple

```c
int main()
{
    char lettre = 0;

    scanf("%c", &lettre);
    printf("%c\n", lettre);

    return 0;
}
```

Lire et afficher un caractère

- On peut demander à l'utilisateur d'entrer une lettre en utilisant le %c dans un scanf (c comme caractère).
- Pour afficher un caractère on doit également utiliser le symbole %c :

Exemple

```c
int main()
{
    char lettre = 0;

    scanf("%c", &lettre);
    printf("%c\n", lettre);

    return 0;
}
```

Lire et afficher un caractère

- On peut demander à l'utilisateur d'entrer une lettre en utilisant le %c dans un scanf (c comme caractère).
- Pour afficher un caractère on doit également utiliser le symbole %c :

Exemple

```c
int main()
{
    char lettre = 0;

    scanf("%c", &lettre);
    printf("%c\n", lettre);

    return 0;
}
```

Les chaînes de caractères

Une chaîne de caractères n'est rien d'autre qu'un tableau de type char.

Adresse	Valeur
18000	'S'
18001	'a'
18002	'l'
18003	'u'
18004	't'

Le symbole de fin de chaîne de caractères

Une chaîne de caractère doit impérativement contenir un caractère spécial à la fin de la chaîne, appelé « caractère de fin de chaîne ». Ce caractère s'écrit '\0'.

Adresse	Valeur
18000	'S'
18001	'a'
18002	'l'
18003	'u'
18004	't'
18005	'\0'

Le symbole de fin de chaîne de caractères

- Le caractère '\0' permet tout simplement d'indiquer la fin la chaîne.
- Par conséquent, pour stocker le mot « Salut » (qui comprend 5 lettres) en mémoire, il ne faut pas un tableau de 5 char, mais de 6 !
- Chaque fois que vous créez une chaîne de caractères, vous allez donc devoir penser à prévoir de la place pour le caractère de fin de chaîne. Il faut toujours toujours ajouter un bloc de plus dans le tableau pour stocker ce caractère '\0', c'est impératif !
- Oublier le caractère de fin '\0' est une source d'erreurs impitoyable du langage C.

Le symbole de fin de chaîne de caractères

- Le caractère '\0' permet tout simplement d'indiquer la fin la chaîne.
- Par conséquent, pour stocker le mot « Salut » (qui comprend 5 lettres) en mémoire, il ne faut pas un tableau de 5 char, mais de 6 !
- Chaque fois que vous créez une chaîne de caractères, vous allez donc devoir penser à prévoir de la place pour le caractère de fin de chaîne. Il faut toujours toujours ajouter un bloc de plus dans le tableau pour stocker ce caractère '\0', c'est impératif !
- Oublier le caractère de fin '\0' est une source d'erreurs impitoyable du langage C.

Le symbole de fin de chaîne de caractères

- Le caractère '\0' permet tout simplement d'indiquer la fin la chaîne.
- Par conséquent, pour stocker le mot « Salut » (qui comprend 5 lettres) en mémoire, il ne faut pas un tableau de 5 char, mais de 6 !
- Chaque fois que vous créez une chaîne de caractères, vous allez donc devoir penser à prévoir de la place pour le caractère de fin de chaîne. Il faut toujours toujours ajouter un bloc de plus dans le tableau pour stocker ce caractère '\0', c'est impératif !
- Oublier le caractère de fin '\0' est une source d'erreurs impitoyable du langage C.

Le symbole de fin de chaîne de caractères

- Le caractère '\0' permet tout simplement d'indiquer la fin la chaîne.
- Par conséquent, pour stocker le mot « Salut » (qui comprend 5 lettres) en mémoire, il ne faut pas un tableau de 5 char, mais de 6 !
- Chaque fois que vous créez une chaîne de caractères, vous allez donc devoir penser à prévoir de la place pour le caractère de fin de chaîne. Il faut toujours toujours ajouter un bloc de plus dans le tableau pour stocker ce caractère '\0', c'est impératif !
- Oublier le caractère de fin '\0' est une source d'erreurs impitoyable du langage C.

Le symbole de fin de chaîne de caractères

Grâce au caractère '\0' :

- vous n'aurez pas à retenir la taille de votre tableau car il indique que le tableau s'arrête à cet endroit.
- vous pourrez passer votre tableau de char à une fonction sans avoir à ajouter à côté une variable indiquant la taille du tableau.
- cela n'est valable que pour les chaînes de caractères (c'est-à-dire le type char*, qu'on peut aussi écrire char[]).
- pour les autres types de tableaux, vous êtes toujours obligés de retenir la taille du tableau quelque part.

Le symbole de fin de chaîne de caractères

Grâce au caractère '\0' :

- vous n'aurez pas à retenir la taille de votre tableau car il indique que le tableau s'arrête à cet endroit.
- vous pourrez passer votre tableau de char à une fonction sans avoir à ajouter à côté une variable indiquant la taille du tableau.
- cela n'est valable que pour les chaînes de caractères (c'est-à-dire le type char*, qu'on peut aussi écrire char[]).
- pour les autres types de tableaux, vous êtes toujours obligés de retenir la taille du tableau quelque part.

Le symbole de fin de chaîne de caractères

Grâce au caractère '\0' :

- vous n'aurez pas à retenir la taille de votre tableau car il indique que le tableau s'arrête à cet endroit.
- vous pourrez passer votre tableau de char à une fonction sans avoir à ajouter à côté une variable indiquant la taille du tableau.
- cela n'est valable que pour les chaînes de caractères (c'est-à-dire le type char*, qu'on peut aussi écrire char[]).
- pour les autres types de tableaux, vous êtes toujours obligés de retenir la taille du tableau quelque part.

Le symbole de fin de chaîne de caractères

Grâce au caractère '\0' :

- vous n'aurez pas à retenir la taille de votre tableau car il indique que le tableau s'arrête à cet endroit.
- vous pourrez passer votre tableau de char à une fonction sans avoir à ajouter à côté une variable indiquant la taille du tableau.
- cela n'est valable que pour les chaînes de caractères (c'est-à-dire le type char*, qu'on peut aussi écrire char[]).
- pour les autres types de tableaux, vous êtes toujours obligés de retenir la taille du tableau quelque part.

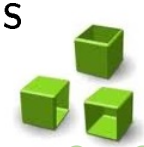
Création et initialisation de la chaîne

- Si on veut initialiser notre tableau chaine avec le texte « Salut », on peut utiliser la méthode manuelle mais peu efficace :

```c
int main()
{
    char chaine[6];

    chaine[0] = 'S';
    chaine[1] = 'a';
    chaine[2] = 'l';
    chaine[3] = 'u';
    chaine[4] = 't';
    chaine[5] = '\0';
    // Affichage de la chaine grace au %s du printf
    printf("%s", chaine);
    return 0;
}
```

- Pour afficher une chaine de caractère, il faut utiliser le symbole%s (s comme string, qui signifie « chaîne » en anglais) dans la fonction printf.



Création et initialisation de la chaîne

- En tapant entre guillemets la chaîne que vous voulez mettre dans votre tableau, le compilateur C calcule automatiquement la taille nécessaire. C'est-à-dire qu'il compte les lettres et ajoute 1 pour placer le caractère '\0'.
- Il écrit ensuite une à une les lettres du mot « Salut » en mémoire et ajoute le '\0' comme on l'a fait nous-mêmes manuellement.

```c
int main()
{
    char chaine[] = "Salut"; // La taille du tableau chaine
        est automatiquement calculee

    printf("%s", chaine);

    return 0;
}
```

- Il y a toutefois un défaut : ça ne marche que pour l'initialisation ! Vous ne pouvez pas écrire plus loin dans le code : chaine = "Salut" ;

Création et initialisation de la chaîne

- En tapant entre guillemets la chaîne que vous voulez mettre dans votre tableau, le compilateur C calcule automatiquement la taille nécessaire. C'est-à-dire qu'il compte les lettres et ajoute 1 pour placer le caractère '\0'.
- Il écrit ensuite une à une les lettres du mot « Salut » en mémoire et ajoute le '\0' comme on l'a fait nous-mêmes manuellement.

```c
int main()
{
    char chaine[] = "Salut"; // La taille du tableau chaine
        est automatiquement calculee

    printf("%s", chaine);

    return 0;
}
```

- Il y a toutefois un défaut : ça ne marche que pour l'initialisation ! Vous ne pouvez pas écrire plus loin dans le code : chaine = "Salut" ;

Création et initialisation de la chaîne

- En tapant entre guillemets la chaîne que vous voulez mettre dans votre tableau, le compilateur C calcule automatiquement la taille nécessaire. C'est-à-dire qu'il compte les lettres et ajoute 1 pour placer le caractère '\0'.
- Il écrit ensuite une à une les lettres du mot « Salut » en mémoire et ajoute le '\0' comme on l'a fait nous-mêmes manuellement.

```c
int main()
{
    char chaine[] = "Salut"; // La taille du tableau chaine
        est automatiquement calculee

    printf("%s", chaine);

    return 0;
}
```

- Il y a toutefois un défaut : ça ne marche que pour l'initialisation ! Vous ne pouvez pas écrire plus loin dans le code : chaine = "Salut" ;

Création et initialisation de la chaîne

- En tapant entre guillemets la chaîne que vous voulez mettre dans votre tableau, le compilateur C calcule automatiquement la taille nécessaire. C'est-à-dire qu'il compte les lettres et ajoute 1 pour placer le caractère '\0'.
- Il écrit ensuite une à une les lettres du mot « Salut » en mémoire et ajoute le '\0' comme on l'a fait nous-mêmes manuellement.

```c
int main()
{
    char chaine[] = "Salut"; // La taille du tableau chaine
        est automatiquement calculee

    printf("%s", chaine);

    return 0;
}
```

- Il y a toutefois un défaut : ça ne marche que pour l'initialisation ! Vous ne pouvez pas écrire plus loin dans le code : chaine = "Salut" ;

Lire une chaîne de caractères

- On peut enregistrer une chaîne entrée par l'utilisateur via un scanf, en utilisant là encore le symbole %s.
- Seul problème : on ne sait pas combien de caractères l'utilisateur va entrer.
- Il va falloir créer un tableau de char très grand, suffisamment grand !

```
int main()
{
    char prenom[100];

    printf("Comment t'appelles-tu ? ");
    scanf("%s", prenom);
    printf("Salut %s, je suis heureux de te rencontrer !",
        prenom);

    return 0;
}
```

Lire une chaîne de caractères

- On peut enregistrer une chaîne entrée par l'utilisateur via un scanf, en utilisant là encore le symbole %s.
- Seul problème : on ne sait pas combien de caractères l'utilisateur va entrer.
- Il va falloir créer un tableau de char très grand, suffisamment grand !

```c
int main()
{
    char prenom[100];

    printf("Comment t'appelles-tu ? ");
    scanf("%s", prenom);
    printf("Salut %s, je suis heureux de te rencontrer !",
        prenom);

    return 0;
}
```

Lire une chaîne de caractères

- On peut enregistrer une chaîne entrée par l'utilisateur via un scanf, en utilisant là encore le symbole %s.
- Seul problème : on ne sait pas combien de caractères l'utilisateur va entrer.
- Il va falloir créer un tableau de char très grand, suffisamment grand !

```
int main()
{
    char prenom[100];

    printf("Comment t'appelles-tu ? ");
    scanf("%s", prenom);
    printf("Salut %s, je suis heureux de te rencontrer !",
        prenom);

    return 0;
}
```

Fonctions de manipulation des chaînes

- Afin de nous aider un peu à manipuler les chaînes, on nous fournit dans la bibliothèque **string.h** un ensemble de fonctions dédiées aux calculs sur des chaînes.

- Pensez donc à inclure #include <**string.h**> en haut des fichiers .c où vous en avez besoin.

- Si vous ne le faites pas, l'ordinateur ne connaîtra pas ces fonctions car il n'aura pas les prototypes, et la compilation plantera.

Fonctions de manipulation des chaînes

- Afin de nous aider un peu à manipuler les chaînes, on nous fournit dans la bibliothèque **string.h** un ensemble de fonctions dédiées aux calculs sur des chaînes.

- Pensez donc à inclure #include <**string.h**> en haut des fichiers .c où vous en avez besoin.

- Si vous ne le faites pas, l'ordinateur ne connaîtra pas ces fonctions car il n'aura pas les prototypes, et la compilation plantera.

Fonctions de manipulation des chaînes

- Afin de nous aider un peu à manipuler les chaînes, on nous fournit dans la bibliothèque **string.h** un ensemble de fonctions dédiées aux calculs sur des chaînes.

- Pensez donc à inclure #include <**string.h**> en haut des fichiers .c où vous en avez besoin.

- Si vous ne le faites pas, l'ordinateur ne connaîtra pas ces fonctions car il n'aura pas les prototypes, et la compilation plantera.

strlen : calculer la longueur d'une chaîne

strlen est une fonction qui calcule la longueur d'une chaîne de caractères (sans compter le caractère '\0').

Exemple

```
int main()
{
    char chaine[] = "Salut";
    int longueurChaine = 0;

    // On recupere la longueur de la chaine dans
        longueurChaine
    longueurChaine = strlen(chaine);

    // On affiche la longueur de la chaine
    printf("La chaine %s fait %d caracteres de long", chaine
        , longueurChaine);

    return 0;
}
```

strlen : calculer la longueur d'une chaîne

```c
int longueurChaine(const char* chaine);
int main()
{
    char chaine[] = "Salut";
    int longueur = 0;
    longueur = longueurChaine(chaine);
    printf("La chaine %s fait %d caracteres de long", chaine
        , longueur);
    return 0;
}
int longueurChaine(const char* chaine)
{
    int nombreDeCaracteres = 0;
    char caractereActuel = 0;
    do
    {
        caractereActuel = chaine[nombreDeCaracteres];
        nombreDeCaracteres++;
    }while(caractereActuel != '\0');
    nombreDeCaracteres--;
    return nombreDeCaracteres;
}
```

strcpy : copier une chaîne dans une autre

La fonction strcpy (comme « string copy ») permet de copier une chaîne à l'intérieur d'une autre.

```c
int main()
{
    char chaine[] = "Texte", copie[100] = {0};
    strcpy(copie, chaine); // On copie "chaine" dans "copie"
    printf("chaine vaut : %s\n", chaine);
    printf("copie vaut : %s\n", copie);
    return 0;
}
```

Cette fonction prend deux paramètres :

- copie : c'est un pointeur vers un char* (tableau de char). C'est dans ce tableau que la chaîne sera copiée ;
- chaine : c'est un pointeur vers un autre tableau de char. Cette chaîne sera copiée dans copie.

La fonction renvoie un pointeur sur copie, ce qui n'est pas très utile. En général, on ne récupère pas ce que cette fonction renvoie.

strcpy : copier une chaîne dans une autre

La fonction strcpy (comme « string copy ») permet de copier une chaîne à l'intérieur d'une autre.

```c
int main()
{
    char chaine[] = "Texte", copie[100] = {0};
    strcpy(copie, chaine); // On copie "chaine" dans "copie"
    printf("chaine vaut : %s\n", chaine);
    printf("copie vaut : %s\n", copie);
    return 0;
}
```

Cette fonction prend deux paramètres :

- copie : c'est un pointeur vers un char* (tableau de char). C'est dans ce tableau que la chaîne sera copiée ;
- chaine : c'est un pointeur vers un autre tableau de char. Cette chaîne sera copiée dans copie.

La fonction renvoie un pointeur sur copie, ce qui n'est pas très utile. En général, on ne récupère pas ce que cette fonction renvoie.

strcpy : copier une chaîne dans une autre

La fonction strcpy (comme « string copy ») permet de copier une chaîne à l'intérieur d'une autre.

```c
int main()
{
    char chaine[] = "Texte", copie[100] = {0};
    strcpy(copie, chaine); // On copie "chaine" dans "copie"
    printf("chaine vaut : %s\n", chaine);
    printf("copie vaut : %s\n", copie);
    return 0;
}
```

Cette fonction prend deux paramètres :

- copie : c'est un pointeur vers un char* (tableau de char). C'est dans ce tableau que la chaîne sera copiée ;
- chaine : c'est un pointeur vers un autre tableau de char. Cette chaîne sera copiée dans copie.

La fonction renvoie un pointeur sur copie, ce qui n'est pas très utile. En général, on ne récupère pas ce que cette fonction renvoie.

strcpy : copier une chaîne dans une autre

La fonction strcpy (comme « string copy ») permet de copier une chaîne à l'intérieur d'une autre.

```c
int main()
{
    char chaine[] = "Texte", copie[100] = {0};
    strcpy(copie, chaine); // On copie "chaine" dans "copie"
    printf("chaine vaut : %s\n", chaine);
    printf("copie vaut : %s\n", copie);
    return 0;
}
```

Cette fonction prend deux paramètres :

- copie : c'est un pointeur vers un char* (tableau de char). C'est dans ce tableau que la chaîne sera copiée ;
- chaine : c'est un pointeur vers un autre tableau de char. Cette chaîne sera copiée dans copie.

La fonction renvoie un pointeur sur copie, ce qui n'est pas très utile. En général, on ne récupère pas ce que cette fonction renvoie.

strcpy : copier une chaîne dans une autre

La fonction strcpy (comme « string copy ») permet de copier une chaîne à l'intérieur d'une autre.

```c
int main()
{
    char chaine[] = "Texte", copie[100] = {0};
    strcpy(copie, chaine); // On copie "chaine" dans "copie"
    printf("chaine vaut : %s\n", chaine);
    printf("copie vaut : %s\n", copie);
    return 0;
}
```

Cette fonction prend deux paramètres :

- copie : c'est un pointeur vers un char* (tableau de char). C'est dans ce tableau que la chaîne sera copiée ;
- chaine : c'est un pointeur vers un autre tableau de char. Cette chaîne sera copiée dans copie.

La fonction renvoie un pointeur sur copie, ce qui n'est pas très utile. En général, on ne récupère pas ce que cette fonction renvoie.

strcat : concaténer 2 chaînes

Cette fonction ajoute une chaîne à la suite d'une autre. On appelle cela la concaténation.

```c
int main()
{
    char chaine1[100] = "Toto ", chaine2[] = "Loulou";

    strcat(chaine1, chaine2); // On concatene chaine2 dans
        chaine1
    printf("chaine1 vaut : %s\n", chaine1);
    printf("chaine2 vaut toujours : %s\n", chaine2);
    return 0;
}
```

Supposons que l'on ait les variables suivantes :

- chaine1 = "Toto "
- chaine2 = "Loulou"

Si on concatène chaine2 dans chaine1, alors chaine1 vaudra "Toto Loulou". Quant à chaine2, elle n'aura pas changé et vaudra donc toujours "Loulou". Seule chaine1 est modifiée.

strcat : concaténer 2 chaînes

Cette fonction ajoute une chaîne à la suite d'une autre. On appelle cela la concaténation.

```c
int main()
{
    char chaine1[100] = "Toto ", chaine2[] = "Loulou";

    strcat(chaine1, chaine2); // On concatene chaine2 dans
        chaine1
    printf("chaine1 vaut : %s\n", chaine1);
    printf("chaine2 vaut toujours : %s\n", chaine2);
    return 0;
}
```

Supposons que l'on ait les variables suivantes :

- chaine1 = "Toto "
- chaine2 = "Loulou"

Si on concatène chaine2 dans chaine1, alors chaine1 vaudra "Toto Loulou". Quant à chaine2, elle n'aura pas changé et vaudra donc toujours "Loulou". Seule chaine1 est modifiée.

strcat : concaténer 2 chaînes

Cette fonction ajoute une chaîne à la suite d'une autre. On appelle cela la concaténation.

```c
int main()
{
    char chaine1[100] = "Toto ", chaine2[] = "Loulou";

    strcat(chaine1, chaine2); // On concatene chaine2 dans
        chaine1
    printf("chaine1 vaut : %s\n", chaine1);
    printf("chaine2 vaut toujours : %s\n", chaine2);
    return 0;
}
```

Supposons que l'on ait les variables suivantes :

- chaine1 = "Toto "
- chaine2 = "Loulou"

Si on concatène chaine2 dans chaine1, alors chaine1 vaudra "Toto Loulou". Quant à chaine2, elle n'aura pas changé et vaudra donc toujours "Loulou". Seule chaine1 est modifiée.

strcmp : comparer 2 chaînes

La fonction strcmp compare 2 chaînes entre elles

```c
int main()
{
    char chaine1[] = "Texte de test", chaine2[] = "Texte de
        test";
    if (strcmp(chaine1, chaine2) == 0)
    {
        printf("Les chaines sont identiques\n");
    }
    else
    {
        printf("Les chaines sont differentes\n");
    }
    return 0;
}
```

Il est important de récupérer ce que la fonction renvoie. En effet, strcmp renvoie :

- 0 si les chaînes sont identiques ;
- une autre valeur (positive ou négative) si les chaînes sont différentes.

strcmp : comparer 2 chaînes

La fonction strcmp compare 2 chaînes entre elles

```c
int main()
{
    char chaine1[] = "Texte de test", chaine2[] = "Texte de
        test";
    if (strcmp(chaine1, chaine2) == 0)
    {
        printf("Les chaines sont identiques\n");
    }
    else
    {
        printf("Les chaines sont differentes\n");
    }
    return 0;
}
```

Il est important de récupérer ce que la fonction renvoie. En effet, strcmp renvoie :

- 0 si les chaînes sont identiques ;
- une autre valeur (positive ou négative) si les chaînes sont différentes.

Précédence alphabétique des caractères

- La précédence des caractères dans l'alphabet d'une machine est dépendante du code de caractères utilisé.
  ...,0,1,2,...,9,...,A,B,C,...,Z,...,a,b,c,...,z,...
- Les symboles spéciaux (' ,+ ,- ,/ ,{ ,] ,...) et les lettres accentuées (é ,è ,à ,û ,...) se trouvent répartis autour des trois grands groupes de caractères (chiffres, majuscules, minuscules).
- Leur précédence ne correspond à aucune règle d'ordre spécifique
- On peut déduire une relation de précédence 'est inférieur à' sur l'ensemble des caractères. Ainsi, on peut dire que '0' est inférieur à 'Z' et noter '0' $<$ 'Z'.
- Car le code du caractère '0' (ASCII : 48) est inférieur au code du caractère 'Z' (ASCII : 90).

Précédence alphabétique des caractères

- La précédence des caractères dans l'alphabet d'une machine est dépendante du code de caractères utilisé.
  ...,0,1,2,...,9,...,A,B,C,...,Z,...,a,b,c,...,z,...

- Les symboles spéciaux (' ,+ ,- ,/ ,{ ,] ,...) et les lettres accentuées (é ,è ,à ,û ,...) se trouvent répartis autour des trois grands groupes de caractères (chiffres, majuscules, minuscules).

- Leur précédence ne correspond à aucune règle d'ordre spécifique

- On peut déduire une relation de précédence 'est inférieur à' sur l'ensemble des caractères. Ainsi, on peut dire que '0' est inférieur à 'Z' et noter '0' $<$ 'Z'.

- Car le code du caractère '0' (ASCII : 48) est inférieur au code du caractère 'Z' (ASCII : 90).

Précédence alphabétique des caractères

- La précédence des caractères dans l'alphabet d'une machine est dépendante du code de caractères utilisé.
  ...,0,1,2,...,9,...,A,B,C,...,Z,...,a,b,c,...,z,...

- Les symboles spéciaux (' ,+ ,- ,/ ,{ ,] ,...) et les lettres accentuées (é ,è ,à ,û ,...) se trouvent répartis autour des trois grands groupes de caractères (chiffres, majuscules, minuscules).

- Leur précédence ne correspond à aucune règle d'ordre spécifique

- On peut déduire une relation de précédence 'est inférieur à' sur l'ensemble des caractères. Ainsi, on peut dire que '0' est inférieur à 'Z' et noter '0' $<$ 'Z'.

- Car le code du caractère '0' (ASCII : 48) est inférieur au code du caractère 'Z' (ASCII : 90).

Précédence alphabétique des caractères

- La précédence des caractères dans l'alphabet d'une machine est dépendante du code de caractères utilisé.
  ...,0,1,2,...,9,...,A,B,C,...,Z,...,a,b,c,...,z,...

- Les symboles spéciaux (' ,+ ,- ,/ ,{ ,] ,...) et les lettres accentuées (é ,è ,à ,û ,...) se trouvent répartis autour des trois grands groupes de caractères (chiffres, majuscules, minuscules).

- Leur précédence ne correspond à aucune règle d'ordre spécifique

- On peut déduire une relation de précédence 'est inférieur à' sur l'ensemble des caractères. Ainsi, on peut dire que '0' est inférieur à 'Z' et noter '0' $<$ 'Z'.

- Car le code du caractère '0' (ASCII : 48) est inférieur au code du caractère 'Z' (ASCII : 90).

Précédence alphabétique des caractères

- La précédence des caractères dans l'alphabet d'une machine est dépendante du code de caractères utilisé.
  ...,0,1,2,...,9,...,A,B,C,...,Z,...,a,b,c,...,z,...

- Les symboles spéciaux (' ,+ ,- ,/ ,{ ,] ,...) et les lettres accentuées (é ,è ,à ,û ,...) se trouvent répartis autour des trois grands groupes de caractères (chiffres, majuscules, minuscules).

- Leur précédence ne correspond à aucune règle d'ordre spécifique

- On peut déduire une relation de précédence 'est inférieur à' sur l'ensemble des caractères. Ainsi, on peut dire que '0' est inférieur à 'Z' et noter '0' $<$ 'Z'.

- Car le code du caractère '0' (ASCII : 48) est inférieur au code du caractère 'Z' (ASCII : 90).

Précédence lexicographique des chaînes de caractères

- La **précédence lexicographique** pour les chaînes de caractères suit l'ordre du dictionnaire et est définie de façon récurrente :
  - La chaîne vide "" précède lexicographiquement toutes les autres chaînes.
  - La chaîne A = "$a_1 a_2 a ... a_p$" (p caractères) précède lexicographiquement la chaîne B = "$b_1 b_2 ... b_m$" (m caractères) si l'une des deux conditions suivantes est remplie :
    1. 'a1' < 'b1'
    2. 'a1' = 'b1' et "$a_2 a_3 ... a_p$" précède lexicographiquement "$b_2 b_3 ... b_m$"
- Exemples :
  1. "ABC" précède "BCD" car 'A'<'B'
  2. "ABC" précède "B" car 'A'<'B'
  3. "Abc" précède "abc" car 'A'<'a'
  4. "ab" précède "abcd" car "" précède "cd"
  5. " ab" précède "ab" car ' '<'a'
- le code ASCII de ' ' est 32, et le code ASCII de 'a' est 97

Précédence lexicographique des chaînes de caractères

- La **précédence lexicographique** pour les chaînes de caractères suit l'ordre du dictionnaire et est définie de façon récurrente :
  - La chaîne vide "" précède lexicographiquement toutes les autres chaînes.
  - La chaîne A = "$a_1 a_2 a ... a_p$" (p caractères) précède lexicographiquement la chaîne B = "$b_1 b_2 ... b_m$" (m caractères) si l'une des deux conditions suivantes est remplie :
    1. 'a1' < 'b1'
    2. 'a1' = 'b1' et "$a_2 a_3 ... a_p$" précède lexicographiquement "$b_2 b_3 ... b_m$"
- Exemples :
  1. "ABC" précède "BCD" car 'A'<'B'
  2. "ABC" précède "B" car 'A'<'B'
  3. "Abc" précède "abc" car 'A'<'a'
  4. "ab" précède "abcd" car "" précède "cd"
  5. " ab" précède "ab" car ' '<'a'
- le code ASCII de ' ' est 32, et le code ASCII de 'a' est 97

Précédence lexicographique des chaînes de caractères

- La **précédence lexicographique** pour les chaînes de caractères suit l'ordre du dictionnaire et est définie de façon récurrente :
  - La chaîne vide "" précède lexicographiquement toutes les autres chaînes.
  - La chaîne A = "$a_1 a_2 a ... a_p$" (p caractères) précède lexicographiquement la chaîne B = "$b_1 b_2 ... b_m$" (m caractères) si l'une des deux conditions suivantes est remplie :
    1. 'a1' < 'b1'
    2. 'a1' = 'b1' et "$a_2 a_3 ... a_p$" précède lexicographiquement "$b_2 b_3 ... b_m$"
- Exemples :
  1. "ABC" précède "BCD" car 'A'<'B'
  2. "ABC" précède "B" car 'A'<'B'
  3. "Abc" précède "abc" car 'A'<'a'
  4. "ab" précède "abcd" car "" précède "cd"
  5. " ab" précède "ab" car ' '<'a'
- le code ASCII de ' ' est 32, et le code ASCII de 'a' est 97

Précédence lexicographique des chaînes de caractères

- La **précédence lexicographique** pour les chaînes de caractères suit l'ordre du dictionnaire et est définie de façon récurrente :
  - La chaîne vide "" précède lexicographiquement toutes les autres chaînes.
  - La chaîne A = "$a_1 a_2 a ... a_p$" (p caractères) précède lexicographiquement la chaîne B = "$b_1 b_2 ... b_m$" (m caractères) si l'une des deux conditions suivantes est remplie :
    1. 'a1' < 'b1'
    2. 'a1' = 'b1' et "$a_2 a_3 ... a_p$" précède lexicographiquement "$b_2 b_3 ... b_m$"
- Exemples :
  1. "ABC" précède "BCD" car 'A'<'B'
  2. "ABC" précède "B" car 'A'<'B'
  3. "Abc" précède "abc" car 'A'<'a'
  4. "ab" précède "abcd" car "" précède "cd"
  5. " ab" précède "ab" car ' '<'a'
- le code ASCII de ' ' est 32, et le code ASCII de 'a' est 97

Précédence lexicographique des chaînes de caractères

- La **précédence lexicographique** pour les chaînes de caractères suit l'ordre du dictionnaire et est définie de façon récurrente :
  - La chaîne vide "" précède lexicographiquement toutes les autres chaînes.
  - La chaîne A = "$a_1 a_2 a ... a_p$" (p caractères) précède lexicographiquement la chaîne B = "$b_1 b_2 ... b_m$" (m caractères) si l'une des deux conditions suivantes est remplie :
    1. 'a1' < 'b1'
    2. 'a1' = 'b1' et "$a_2 a_3 ... a_p$" précède lexicographiquement "$b_2 b_3 ... b_m$"
- Exemples :
  1. "ABC" précède "BCD" car 'A'<'B'
  2. "ABC" précède "B" car 'A'<'B'
  3. "Abc" précède "abc" car 'A'<'a'
  4. "ab" précède "abcd" car "" précède "cd"
  5. " ab" précède "ab" car ' '<'a'
- le code ASCII de ' ' est 32, et le code ASCII de 'a' est 97

Précédence lexicographique des chaînes de caractères

- La **précédence lexicographique** pour les chaînes de caractères suit l'ordre du dictionnaire et est définie de façon récurrente :
  - La chaîne vide "" précède lexicographiquement toutes les autres chaînes.
  - La chaîne A = "$a_1 a_2 a ... a_p$" (p caractères) précède lexicographiquement la chaîne B = "$b_1 b_2 ... b_m$" (m caractères) si l'une des deux conditions suivantes est remplie :
    1. 'a1' < 'b1'
    2. 'a1' = 'b1' et "$a_2 a_3 ... a_p$" précède lexicographiquement "$b_2 b_3 ... b_m$"
- Exemples :
  1. "ABC" précède "BCD" car 'A'<'B'
  2. "ABC" précède "B" car 'A'<'B'
  3. "Abc" précède "abc" car 'A'<'a'
  4. "ab" précède "abcd" car "" précède "cd"
  5. " ab" précède "ab" car ' '<'a'
- le code ASCII de ' ' est 32, et le code ASCII de 'a' est 97

Précédence lexicographique des chaînes de caractères

- La **précédence lexicographique** pour les chaînes de caractères suit l'ordre du dictionnaire et est définie de façon récurrente :
  - La chaîne vide "" précède lexicographiquement toutes les autres chaînes.
  - La chaîne A = "$a_1 a_2 a ... a_p$" (p caractères) précède lexicographiquement la chaîne B = "$b_1 b_2 ... b_m$" (m caractères) si l'une des deux conditions suivantes est remplie :
    1. 'a1' < 'b1'
    2. 'a1' = 'b1' et "$a_2 a_3 ... a_p$" précède lexicographiquement "$b_2 b_3 ... b_m$"
- Exemples :
  1. "ABC" précède "BCD" car 'A'<'B'
  2. "ABC" précède "B" car 'A'<'B'
  3. "Abc" précède "abc" car 'A'<'a'
  4. "ab" précède "abcd" car "" précède "cd"
  5. " ab" précède "ab" car ' '<'a'
- le code ASCII de ' ' est 32, et le code ASCII de 'a' est 97

Précédence lexicographique des chaînes de caractères

- La **précédence lexicographique** pour les chaînes de caractères suit l'ordre du dictionnaire et est définie de façon récurrente :
  - La chaîne vide "" précède lexicographiquement toutes les autres chaînes.
  - La chaîne A = "$a_1 a_2 a ... a_p$" (p caractères) précède lexicographiquement la chaîne B = "$b_1 b_2 ... b_m$" (m caractères) si l'une des deux conditions suivantes est remplie :
    1. 'a1' < 'b1'
    2. 'a1' = 'b1' et "$a_2 a_3 ... a_p$" précède lexicographiquement "$b_2 b_3 ... b_m$"
- Exemples :
  1. "ABC" précède "BCD" car 'A'<'B'
  2. "ABC" précède "B" car 'A'<'B'
  3. "Abc" précède "abc" car 'A'<'a'
  4. "ab" précède "abcd" car "" précède "cd"
  5. " ab" précède "ab" car ' '<'a'
- le code ASCII de ' ' est 32, et le code ASCII de 'a' est 97

Précédence lexicographique des chaînes de caractères

- La **précédence lexicographique** pour les chaînes de caractères suit l'ordre du dictionnaire et est définie de façon récurrente :
  - La chaîne vide "" précède lexicographiquement toutes les autres chaînes.
  - La chaîne A = "$a_1 a_2 a ... a_p$" (p caractères) précède lexicographiquement la chaîne B = "$b_1 b_2 ... b_m$" (m caractères) si l'une des deux conditions suivantes est remplie :
    1. 'a1' < 'b1'
    2. 'a1' = 'b1' et "$a_2 a_3 ... a_p$" précède lexicographiquement "$b_2 b_3 ... b_m$"
- Exemples :
  1. "ABC" précède "BCD" car 'A'<'B'
  2. "ABC" précède "B" car 'A'<'B'
  3. "Abc" précède "abc" car 'A'<'a'
  4. "ab" précède "abcd" car "" précède "cd"
  5. " ab" précède "ab" car ' '<'a'
- le code ASCII de ' ' est 32, et le code ASCII de 'a' est 97

Exemples d'utilisation de la précédence lexicographique

En tenant compte de l'ordre alphabétique des caractères, on peut contrôler le type du caractère (chiffre, majuscule, minuscule).

```
int main()
{
        char C=' ';
        C=getchar();
        if (C>='0' && C<='9') printf("Chiffre\n", C);
        if (C>='A' && C<='Z') printf("Majuscule\n", C);
        if (C>='a' && C<='z') printf("Minuscule\n", C);
// Il est facile, de convertir des lettres majuscules dans
    des minuscules:
        if (C>='A' && C<='Z') C = C-'A'+'a';
//ou vice-versa:
        if (C>='a' && C<='z') C = C-'a'+'A';
        putchar(C);

return 0;
}
```

strchr : rechercher un caractère

La fonction strchr recherche un caractère dans une chaîne. Elle renvoie un pointeur vers le premier caractère qu'elle a trouvé. Elle renvoie NULL sinon.

```c
int main()
{
    char chaine[] = "Texte de test", *suiteChaine = NULL;
    suiteChaine = strchr(chaine, 'd');
    if (suiteChaine != NULL) // Si on a trouve quelque chose
    {
        printf("Voici la fin de la chaine a partir du
            premier d : %s", suiteChaine);
    }
    return 0;
}
```

La fonction prend 2 paramètres :

- chaine : la chaîne dans laquelle la recherche doit être faite ;
- caractereARechercher : le caractère que l'on doit rechercher dans la chaîne.

strchr : rechercher un caractère

La fonction strchr recherche un caractère dans une chaîne. Elle renvoie un pointeur vers le premier caractère qu'elle a trouvé. Elle renvoie NULL sinon.

```
int main()
{
    char chaine[] = "Texte de test", *suiteChaine = NULL;
    suiteChaine = strchr(chaine, 'd');
    if (suiteChaine != NULL) // Si on a trouve quelque chose
    {
        printf("Voici la fin de la chaine a partir du
            premier d : %s", suiteChaine);
    }
    return 0;
}
```

La fonction prend 2 paramètres :

- chaine : la chaîne dans laquelle la recherche doit être faite ;
- caractereARechercher : le caractère que l'on doit rechercher dans la chaîne.

strchr : rechercher un caractère

La fonction strchr recherche un caractère dans une chaîne. Elle renvoie un pointeur vers le premier caractère qu'elle a trouvé. Elle renvoie NULL sinon.

```c
int main()
{
    char chaine[] = "Texte de test", *suiteChaine = NULL;
    suiteChaine = strchr(chaine, 'd');
    if (suiteChaine != NULL) // Si on a trouve quelque chose
    {
        printf("Voici la fin de la chaine a partir du
            premier d : %s", suiteChaine);
    }
    return 0;
}
```

La fonction prend 2 paramètres :

- chaine : la chaîne dans laquelle la recherche doit être faite ;
- caractereARechercher : le caractère que l'on doit rechercher dans la chaîne.

strchr : rechercher un caractère

La fonction strchr recherche un caractère dans une chaîne. Elle renvoie un pointeur vers le premier caractère qu'elle a trouvé. Elle renvoie NULL sinon.

```
int main()
{
    char chaine[] = "Texte de test", *suiteChaine = NULL;
    suiteChaine = strchr(chaine, 'd');
    if (suiteChaine != NULL) // Si on a trouve quelque chose
    {
        printf("Voici la fin de la chaine a partir du
            premier d : %s", suiteChaine);
    }
    return 0;
}
```

La fonction prend 2 paramètres :

- chaine : la chaîne dans laquelle la recherche doit être faite ;
- caractereARechercher : le caractère que l'on doit rechercher dans la chaîne.

strchr : rechercher un caractère

La fonction strrchr recherche un caractère dans une chaîne. Elle renvoie un pointeur vers le dernier caractère qu'elle a trouvé. Elle renvoie NULL sinon.

```c
int main()
{
    char chaine[] = "Texte de test", *suiteChaine = NULL;
    suiteChaine = strrchr(chaine, 'e');
    if (suiteChaine != NULL) // Si on a trouve quelque chose
    {
        printf("Voici la fin de la chaine a partir du
            premier d : %s", suiteChaine);
    }
    return 0;
}
```

La fonction prend 2 paramètres :

- chaine : la chaîne dans laquelle la recherche doit être faite ;
- caractereARechercher : le caractère que l'on doit rechercher dans la chaîne.

strchr : rechercher un caractère

La fonction strrchr recherche un caractère dans une chaîne. Elle renvoie un pointeur vers le dernier caractère qu'elle a trouvé. Elle renvoie NULL sinon.

```
int main()
{
    char chaine[] = "Texte de test", *suiteChaine = NULL;
    suiteChaine = strrchr(chaine, 'e');
    if (suiteChaine != NULL) // Si on a trouve quelque chose
    {
        printf("Voici la fin de la chaine a partir du
            premier d : %s", suiteChaine);
    }
    return 0;
}
```

La fonction prend 2 paramètres :

- chaine : la chaîne dans laquelle la recherche doit être faite ;
- caractereARechercher : le caractère que l'on doit rechercher dans la chaîne.

strchr : rechercher un caractère

La fonction strrchr recherche un caractère dans une chaîne. Elle renvoie un pointeur vers le dernier caractère qu'elle a trouvé. Elle renvoie NULL sinon.

```
int main()
{
    char chaine[] = "Texte de test", *suiteChaine = NULL;
    suiteChaine = strrchr(chaine, 'e');
    if (suiteChaine != NULL) // Si on a trouve quelque chose
    {
        printf("Voici la fin de la chaine a partir du
            premier d : %s", suiteChaine);
    }
    return 0;
}
```

La fonction prend 2 paramètres :

- chaine : la chaîne dans laquelle la recherche doit être faite ;
- caractereARechercher : le caractère que l'on doit rechercher dans la chaîne.

strchr : rechercher un caractère

La fonction strrchr recherche un caractère dans une chaîne. Elle renvoie un pointeur vers le dernier caractère qu'elle a trouvé. Elle renvoie NULL sinon.

```
int main()
{
    char chaine[] = "Texte de test", *suiteChaine = NULL;
    suiteChaine = strrchr(chaine, 'e');
    if (suiteChaine != NULL) // Si on a trouve quelque chose
    {
        printf("Voici la fin de la chaine a partir du
            premier d : %s", suiteChaine);
    }
    return 0;
}
```

La fonction prend 2 paramètres :

- chaine : la chaîne dans laquelle la recherche doit être faite ;
- caractereARechercher : le caractère que l'on doit rechercher dans la chaîne.

strstr : rechercher une chaîne dans une autre

Cette fonction recherche la première occurrence d'une chaîne dans une autre chaîne.

```
int main()
{
    char *suiteChaine;

    // On cherche la premiere occurrence de "test" dans "
        Texte de test" :
    suiteChaine = strstr("Texte de test", "test");
    if (suiteChaine != NULL)
    {
        printf("Premiere occurrence de test dans Texte de
            test : %s\n", suiteChaine);
    }

    return 0;
}
```

Elle renvoie, comme les autres, un pointeur quand elle a trouvé ce qu'elle cherchait. Elle renvoie NULL si elle n'a rien trouvé.

strstr : rechercher une chaîne dans une autre

Cette fonction recherche la première occurrence d'une chaîne dans une autre chaîne.

```
int main()
{
    char *suiteChaine;

    // On cherche la premiere occurrence de "test" dans "
        Texte de test" :
    suiteChaine = strstr("Texte de test", "test");
    if (suiteChaine != NULL)
    {
        printf("Premiere occurrence de test dans Texte de
            test : %s\n", suiteChaine);
    }

    return 0;
}
```

Elle renvoie, comme les autres, un pointeur quand elle a trouvé ce qu'elle cherchait. Elle renvoie NULL si elle n'a rien trouvé.

sprintf : écrire dans une chaîne

Cette fonction ressemble énormément au printf que vous connaissez mais, au lieu d'écrire à l'écran, sprintf écrit dans une chaîne !

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char chaine[100];
    int age = 15;
    // On ecrit "Tu as 15 ans" dans chaine
    sprintf(chaine, "Tu as %d ans !", age);
    // On affiche chaine pour verifier qu'elle contient bien
        cela :
    printf("%s", chaine);
    return 0;
}
```

Elle renvoie, comme les autres, un pointeur quand elle a trouvé ce qu'elle cherchait. Elle renvoie NULL si elle n'a rien trouvé.

sprintf : écrire dans une chaîne

Cette fonction ressemble énormément au printf que vous connaissez mais, au lieu d'écrire à l'écran, sprintf écrit dans une chaîne !

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char chaine[100];
    int age = 15;
    // On ecrit "Tu as 15 ans" dans chaine
    sprintf(chaine, "Tu as %d ans !", age);
    // On affiche chaine pour verifier qu'elle contient bien
        cela :
    printf("%s", chaine);
    return 0;
}
```

Elle renvoie, comme les autres, un pointeur quand elle a trouvé ce qu'elle cherchait. Elle renvoie NULL si elle n'a rien trouvé.

Quelques fonctions utiles de <stdio.h>

1. **getchar** : qui lit le prochain caractère du fichier d'entrée standard stdin (le clavier).
2. **putchar(c)** : transfère le caractère c vers le fichier de sortie standard stdout (l'écran).
3. **puts (s)** : écrit la chaîne de caractères désignée par s sur stdout (l'écran) et provoque un retour à la ligne.
4. **gets(s)** : lit une ligne de de caractères du clavier et la copie à l'adresse indiquée par s. Le retour à la ligne final est remplacé par le symbole de fin de chaîne '\0'.

```
void main()
{
    char chaine[100];
    char C=' ';
        gets(chaine);
        puts(chaine);
        C=getchar();
        putchar(C);
}
```

Quelques fonctions utiles de <stdio.h>

1. **getchar** : qui lit le prochain caractère du fichier d'entrée standard stdin (le clavier).
2. **putchar(c)** : transfère le caractère c vers le fichier de sortie standard stdout (l'écran).
3. **puts (s)** : écrit la chaîne de caractères désignée par s sur stdout (l'écran) et provoque un retour à la ligne.
4. **gets(s)** : lit une ligne de de caractères du clavier et la copie à l'adresse indiquée par s. Le retour à la ligne final est remplacé par le symbole de fin de chaîne '\0'.

```
void main()
{
    char chaine[100];
    char C=' ';
        gets(chaine);
        puts(chaine);
        C=getchar();
        putchar(C);
}
```

Quelques fonctions utiles de <stdio.h>

1. **getchar** : qui lit le prochain caractère du fichier d'entrée standard stdin (le clavier).
2. **putchar(c)** : transfère le caractère c vers le fichier de sortie standard stdout (l'écran).
3. **puts (s)** : écrit la chaîne de caractères désignée par s sur stdout (l'écran) et provoque un retour à la ligne.
4. **gets(s)** : lit une ligne de de caractères du clavier et la copie à l'adresse indiquée par s. Le retour à la ligne final est remplacé par le symbole de fin de chaîne '\0'.

```
void main()
{
    char chaine[100];
    char C=' ';
        gets(chaine);
        puts(chaine);
        C=getchar();
        putchar(C);
}
```

Quelques fonctions utiles de <stdio.h>

1. **getchar** : qui lit le prochain caractère du fichier d'entrée standard stdin (le clavier).
2. **putchar(c)** : transfère le caractère c vers le fichier de sortie standard stdout (l'écran).
3. **puts (s)** : écrit la chaîne de caractères désignée par s sur stdout (l'écran) et provoque un retour à la ligne.
4. **gets(s)** : lit une ligne de de caractères du clavier et la copie à l'adresse indiquée par s. Le retour à la ligne final est remplacé par le symbole de fin de chaîne '\0'.

```
void main()
{
    char chaine[100];
    char C=' ';
        gets(chaine);
        puts(chaine);
        C=getchar();
        putchar(C);
}
```

Quelques fonctions de <stdlib.h>

Conversion de chaînes de caractères en nombres

- **atoi(s)** :retourne la valeur numérique représentée par <s> comme int
- **atol(s)** :retourne la valeur numérique représentée par <s> comme long
- **atof(s)** : retourne la valeur numérique représentée par <s> comme double.

Règles générales pour la conversion :

- Les espaces au début d'une chaîne sont ignorés
- Il n'y a pas de contrôle du domaine de la cible
- La conversion s'arrête au premier caractère non convertible
- Pour une chaîne non convertible, les fonctions retournent zéro

Quelques fonctions de <stdlib.h>

Conversion de chaînes de caractères en nombres

- **atoi(s)** :retourne la valeur numérique représentée par <s> comme int
- **atol(s)** :retourne la valeur numérique représentée par <s> comme long
- **atof(s)** : retourne la valeur numérique représentée par <s> comme double.

Règles générales pour la conversion :

- Les espaces au début d'une chaîne sont ignorés
- Il n'y a pas de contrôle du domaine de la cible
- La conversion s'arrête au premier caractère non convertible
- Pour une chaîne non convertible, les fonctions retournent zéro

Quelques fonctions de <stdlib.h>

Conversion de chaînes de caractères en nombres

- **atoi(s)** :retourne la valeur numérique représentée par <s> comme int
- **atol(s)** :retourne la valeur numérique représentée par <s> comme long
- **atof(s)** : retourne la valeur numérique représentée par <s> comme double.

Règles générales pour la conversion :

- Les espaces au début d'une chaîne sont ignorés
- Il n'y a pas de contrôle du domaine de la cible
- La conversion s'arrête au premier caractère non convertible
- Pour une chaîne non convertible, les fonctions retournent zéro

Quelques fonctions de <stdlib.h>

Conversion de chaînes de caractères en nombres

- **atoi(s)** :retourne la valeur numérique représentée par <s> comme int
- **atol(s)** :retourne la valeur numérique représentée par <s> comme long
- **atof(s)** : retourne la valeur numérique représentée par <s> comme double.

Règles générales pour la conversion :

- Les espaces au début d'une chaîne sont ignorés
- Il n'y a pas de contrôle du domaine de la cible
- La conversion s'arrête au premier caractère non convertible
- Pour une chaîne non convertible, les fonctions retournent zéro

Quelques fonctions de <stdlib.h>

Conversion de chaînes de caractères en nombres

- **atoi(s)** :retourne la valeur numérique représentée par <s> comme int
- **atol(s)** :retourne la valeur numérique représentée par <s> comme long
- **atof(s)** : retourne la valeur numérique représentée par <s> comme double.

Règles générales pour la conversion :

- Les espaces au début d'une chaîne sont ignorés
- Il n'y a pas de contrôle du domaine de la cible
- La conversion s'arrête au premier caractère non convertible
- Pour une chaîne non convertible, les fonctions retournent zéro

Quelques fonctions de <stdlib.h>

Conversion de chaînes de caractères en nombres

- **atoi(s)** :retourne la valeur numérique représentée par <s> comme int
- **atol(s)** :retourne la valeur numérique représentée par <s> comme long
- **atof(s)** : retourne la valeur numérique représentée par <s> comme double.

Règles générales pour la conversion :

- Les espaces au début d'une chaîne sont ignorés
- Il n'y a pas de contrôle du domaine de la cible
- La conversion s'arrête au premier caractère non convertible
- Pour une chaîne non convertible, les fonctions retournent zéro

Quelques fonctions de <stdlib.h>

Conversion de chaînes de caractères en nombres

- **atoi(s)** :retourne la valeur numérique représentée par <s> comme int
- **atol(s)** :retourne la valeur numérique représentée par <s> comme long
- **atof(s)** : retourne la valeur numérique représentée par <s> comme double.

Règles générales pour la conversion :

- Les espaces au début d'une chaîne sont ignorés
- Il n'y a pas de contrôle du domaine de la cible
- La conversion s'arrête au premier caractère non convertible
- Pour une chaîne non convertible, les fonctions retournent zéro

Quelques fonctions de <stdlib.h>

Conversion de chaînes de caractères en nombres

- **atoi(s)** :retourne la valeur numérique représentée par <s> comme int
- **atol(s)** :retourne la valeur numérique représentée par <s> comme long
- **atof(s)** : retourne la valeur numérique représentée par <s> comme double.

Règles générales pour la conversion :

- Les espaces au début d'une chaîne sont ignorés
- Il n'y a pas de contrôle du domaine de la cible
- La conversion s'arrête au premier caractère non convertible
- Pour une chaîne non convertible, les fonctions retournent zéro

Quelques fonctions de <stdlib.h>

Conversion de nombres en chaînes de caractères

- **itoa** (n_int, s, b)
- **ltoa** (n_long, s, b)
- **ultoa** (n_uns_long, s, b)

Chacune de ces trois procédures convertit son premier argument en une chaîne de caractères qui sera ensuite attribuée à <s>. La conversion se fait dans la base <b>.

- n_int est un nombre du type int
- n_long est un nombre du type long
- n_uns_long est un nombre du type unsigned long
- s est une chaîne de caractères longueur maximale de la chaîne : 17 resp. 33 byte
- b est la base pour la conversion (2 ... 36)

Quelques fonctions de <stdlib.h>

Conversion de nombres en chaînes de caractères

- **itoa** (n_int, s, b)
- **ltoa** (n_long, s, b)
- **ultoa** (n_uns_long, s, b)

Chacune de ces trois procédures convertit son premier argument en une chaîne de caractères qui sera ensuite attribuée à <s>. La conversion se fait dans la base <b>.

- n_int est un nombre du type int
- n_long est un nombre du type long
- n_uns_long est un nombre du type unsigned long
- s est une chaîne de caractères longueur maximale de la chaîne : 17 resp. 33 byte
- b est la base pour la conversion (2 ... 36)

Quelques fonctions de <stdlib.h>

Conversion de nombres en chaînes de caractères

- **itoa** (n_int, s, b)
- **ltoa** (n_long, s, b)
- **ultoa** (n_uns_long, s, b)

Chacune de ces trois procédures convertit son premier argument en une chaîne de caractères qui sera ensuite attribuée à <s>. La conversion se fait dans la base <b>.

- n_int est un nombre du type int
- n_long est un nombre du type long
- n_uns_long est un nombre du type unsigned long
- s est une chaîne de caractères longueur maximale de la chaîne : 17 resp. 33 byte
- b est la base pour la conversion (2 ... 36)

Quelques fonctions de <stdlib.h>

Conversion de nombres en chaînes de caractères

- **itoa** (n_int, s, b)
- **ltoa** (n_long, s, b)
- **ultoa** (n_uns_long, s, b)

Chacune de ces trois procédures convertit son premier argument en une chaîne de caractères qui sera ensuite attribuée à <s>. La conversion se fait dans la base <b>.

- n_int est un nombre du type int
- n_long est un nombre du type long
- n_uns_long est un nombre du type unsigned long
- s est une chaîne de caractères longueur maximale de la chaîne : 17 resp. 33 byte
- b est la base pour la conversion (2 ... 36)

Quelques fonctions de <stdlib.h>

Conversion de nombres en chaînes de caractères

- **itoa** (n_int, s, b)
- **ltoa** (n_long, s, b)
- **ultoa** (n_uns_long, s, b)

Chacune de ces trois procédures convertit son premier argument en une chaîne de caractères qui sera ensuite attribuée à <s>. La conversion se fait dans la base <b>.

- n_int est un nombre du type int
- n_long est un nombre du type long
- n_uns_long est un nombre du type unsigned long
- s est une chaîne de caractères longueur maximale de la chaîne : 17 resp. 33 byte
- b est la base pour la conversion (2 ... 36)

Quelques fonctions de <stdlib.h>

Conversion de nombres en chaînes de caractères

- **itoa** (n_int, s, b)
- **ltoa** (n_long, s, b)
- **ultoa** (n_uns_long, s, b)

Chacune de ces trois procédures convertit son premier argument en une chaîne de caractères qui sera ensuite attribuée à <s>. La conversion se fait dans la base <b>.

- n_int est un nombre du type int
- n_long est un nombre du type long
- n_uns_long est un nombre du type unsigned long
- s est une chaîne de caractères longueur maximale de la chaîne : 17 resp. 33 byte
- b est la base pour la conversion (2 ... 36)

Quelques fonctions de <stdlib.h>

Conversion de nombres en chaînes de caractères

- **itoa** (n_int, s, b)
- **ltoa** (n_long, s, b)
- **ultoa** (n_uns_long, s, b)

Chacune de ces trois procédures convertit son premier argument en une chaîne de caractères qui sera ensuite attribuée à <s>. La conversion se fait dans la base <b>.

- n_int est un nombre du type int
- n_long est un nombre du type long
- n_uns_long est un nombre du type unsigned long
- s est une chaîne de caractères longueur maximale de la chaîne : 17 resp. 33 byte
- b est la base pour la conversion (2 ... 36)

Quelques fonctions de <stdlib.h>

Conversion de nombres en chaînes de caractères

- **itoa** (n_int, s, b)
- **ltoa** (n_long, s, b)
- **ultoa** (n_uns_long, s, b)

Chacune de ces trois procédures convertit son premier argument en une chaîne de caractères qui sera ensuite attribuée à <s>. La conversion se fait dans la base <b>.

- n_int est un nombre du type int
- n_long est un nombre du type long
- n_uns_long est un nombre du type unsigned long
- s est une chaîne de caractères longueur maximale de la chaîne : 17 resp. 33 byte
- b est la base pour la conversion (2 ... 36)

Quelques fonctions de <stdlib.h>

Conversion de nombres en chaînes de caractères

- **itoa** (n_int, s, b)
- **ltoa** (n_long, s, b)
- **ultoa** (n_uns_long, s, b)

Chacune de ces trois procédures convertit son premier argument en une chaîne de caractères qui sera ensuite attribuée à <s>. La conversion se fait dans la base <b>.

- n_int est un nombre du type int
- n_long est un nombre du type long
- n_uns_long est un nombre du type unsigned long
- s est une chaîne de caractères longueur maximale de la chaîne : 17 resp. 33 byte
- b est la base pour la conversion (2 ... 36)

Quelques fonctions de <ctype.h>

Les fonctions de classification suivantes fournissent un résultat du type int différent de zéro, si la condition respective est remplie, sinon zéro.

- **isupper(<c)** si <c> est une majuscule ('A'...'Z')
- **islower(c)** si c est une minuscule ('a'...'z')
- **isdigit(c)** si c est un chiffre décimal ('0'...'9')
- **isalpha(c)** si islower(c) ou isupper(c)
- **isalnum(c)** si isalpha(c) ou isdigit(c)
- **isxdigit(c)** si c est un chiffre hexadécimal ('0'...'9' ou 'A'...'F' ou 'a'...'f')
- **isspace(c)** si c est un signe d'espacement (' ', '\t', '\n', '\r', '\f')

Les fonctions de conversion suivantes fournissent une valeur du type int qui peut être représentée comme caractère ; la valeur originale de c reste inchangée :

- **tolower(c)** retourne c converti en minuscule si c est une majuscule
- **toupper(c)** retourne c converti en majuscule si c est une minuscule

Quelques fonctions de <ctype.h>

Les fonctions de classification suivantes fournissent un résultat du type int différent de zéro, si la condition respective est remplie, sinon zéro.

- **isupper(<c)** si <c> est une majuscule ('A'...'Z')
- **islower(c)** si c est une minuscule ('a'...'z')
- **isdigit(c)** si c est un chiffre décimal ('0'...'9')
- **isalpha(c)** si islower(c) ou isupper(c)
- **isalnum(c)** si isalpha(c) ou isdigit(c)
- **isxdigit(c)** si c est un chiffre hexadécimal ('0'...'9' ou 'A'...'F' ou 'a'...'f')
- **isspace(c)** si c est un signe d'espacement (' ', '\t', '\n', '\r', '\f')

Les fonctions de conversion suivantes fournissent une valeur du type int qui peut être représentée comme caractère ; la valeur originale de c reste inchangée :

- **tolower(c)** retourne c converti en minuscule si c est une majuscule
- **toupper(c)** retourne c converti en majuscule si c est une minuscule

Quelques fonctions de <ctype.h>

Les fonctions de classification suivantes fournissent un résultat du type int différent de zéro, si la condition respective est remplie, sinon zéro.

- **isupper(<c)** si <c> est une majuscule ('A'...'Z')
- **islower(c)** si c est une minuscule ('a'...'z')
- **isdigit(c)** si c est un chiffre décimal ('0'...'9')
- **isalpha(c)** si islower(c) ou isupper(c)
- **isalnum(c)** si isalpha(c) ou isdigit(c)
- **isxdigit(c)** si c est un chiffre hexadécimal ('0'...'9' ou 'A'...'F' ou 'a'...'f')
- **isspace(c)** si c est un signe d'espacement (' ', '\t', '\n', '\r', '\f')

Les fonctions de conversion suivantes fournissent une valeur du type int qui peut être représentée comme caractère ; la valeur originale de c reste inchangée :

- **tolower(c)** retourne c converti en minuscule si c est une majuscule
- **toupper(c)** retourne c converti en majuscule si c est une minuscule

Quelques fonctions de <ctype.h>

Les fonctions de classification suivantes fournissent un résultat du type int différent de zéro, si la condition respective est remplie, sinon zéro.

- **isupper(<c)** si <c> est une majuscule ('A'...'Z')
- **islower(c)** si c est une minuscule ('a'...'z')
- **isdigit(c)** si c est un chiffre décimal ('0'...'9')
- **isalpha(c)** si islower(c) ou isupper(c)
- **isalnum(c)** si isalpha(c) ou isdigit(c)
- **isxdigit(c)** si c est un chiffre hexadécimal ('0'...'9' ou 'A'...'F' ou 'a'...'f')
- **isspace(c)** si c est un signe d'espacement (' ', '\t', '\n', '\r', '\f')

Les fonctions de conversion suivantes fournissent une valeur du type int qui peut être représentée comme caractère ; la valeur originale de c reste inchangée :

- **tolower(c)** retourne c converti en minuscule si c est une majuscule
- **toupper(c)** retourne c converti en majuscule si c est une minuscule

Quelques fonctions de <ctype.h>

Les fonctions de classification suivantes fournissent un résultat du type int différent de zéro, si la condition respective est remplie, sinon zéro.

- **isupper(<c)** si <c> est une majuscule ('A'...'Z')
- **islower(c)** si c est une minuscule ('a'...'z')
- **isdigit(c)** si c est un chiffre décimal ('0'...'9')
- **isalpha(c)** si islower(c) ou isupper(c)
- **isalnum(c)** si isalpha(c) ou isdigit(c)
- **isxdigit(c)** si c est un chiffre hexadécimal ('0'...'9' ou 'A'...'F' ou 'a'...'f')
- **isspace(c)** si c est un signe d'espacement (' ', '\t', '\n', '\r', '\f')

Les fonctions de conversion suivantes fournissent une valeur du type int qui peut être représentée comme caractère ; la valeur originale de c reste inchangée :

- **tolower(c)** retourne c converti en minuscule si c est une majuscule
- **toupper(c)** retourne c converti en majuscule si c est une minuscule

Quelques fonctions de <ctype.h>

Les fonctions de classification suivantes fournissent un résultat du type int différent de zéro, si la condition respective est remplie, sinon zéro.

- **isupper(<c)** si <c> est une majuscule ('A'...'Z')
- **islower(c)** si c est une minuscule ('a'...'z')
- **isdigit(c)** si c est un chiffre décimal ('0'...'9')
- **isalpha(c)** si islower(c) ou isupper(c)
- **isalnum(c)** si isalpha(c) ou isdigit(c)
- **isxdigit(c)** si c est un chiffre hexadécimal ('0'...'9' ou 'A'...'F' ou 'a'...'f')
- **isspace(c)** si c est un signe d'espacement (' ', '\t', '\n', '\r', '\f')

Les fonctions de conversion suivantes fournissent une valeur du type int qui peut être représentée comme caractère ; la valeur originale de c reste inchangée :

- **tolower(c)** retourne c converti en minuscule si c est une majuscule
- **toupper(c)** retourne c converti en majuscule si c est une minuscule

Quelques fonctions de <ctype.h>

Les fonctions de classification suivantes fournissent un résultat du type int différent de zéro, si la condition respective est remplie, sinon zéro.

- **isupper(<c)** si <c> est une majuscule ('A'...'Z')
- **islower(c)** si c est une minuscule ('a'...'z')
- **isdigit(c)** si c est un chiffre décimal ('0'...'9')
- **isalpha(c)** si islower(c) ou isupper(c)
- **isalnum(c)** si isalpha(c) ou isdigit(c)
- **isxdigit(c)** si c est un chiffre hexadécimal ('0'...'9' ou 'A'...'F' ou 'a'...'f')
- **isspace(c)** si c est un signe d'espacement (' ', '\t', '\n', '\r', '\f')

Les fonctions de conversion suivantes fournissent une valeur du type int qui peut être représentée comme caractère ; la valeur originale de c reste inchangée :

- **tolower(c)** retourne c converti en minuscule si c est une majuscule
- **toupper(c)** retourne c converti en majuscule si c est une minuscule

Quelques fonctions de <ctype.h>

Les fonctions de classification suivantes fournissent un résultat du type int différent de zéro, si la condition respective est remplie, sinon zéro.

- **isupper(<c)** si <c> est une majuscule ('A'...'Z')
- **islower(c)** si c est une minuscule ('a'...'z')
- **isdigit(c)** si c est un chiffre décimal ('0'...'9')
- **isalpha(c)** si islower(c) ou isupper(c)
- **isalnum(c)** si isalpha(c) ou isdigit(c)
- **isxdigit(c)** si c est un chiffre hexadécimal ('0'...'9' ou 'A'...'F' ou 'a'...'f')
- **isspace(c)** si c est un signe d'espacement (' ', '\t', '\n', '\r', '\f')

Les fonctions de conversion suivantes fournissent une valeur du type int qui peut être représentée comme caractère ; la valeur originale de c reste inchangée :

- **tolower(c)** retourne c converti en minuscule si c est une majuscule
- **toupper(c)** retourne c converti en majuscule si c est une minuscule

Quelques fonctions de <ctype.h>

Les fonctions de classification suivantes fournissent un résultat du type int différent de zéro, si la condition respective est remplie, sinon zéro.

- **isupper(<c)** si <c> est une majuscule ('A'...'Z')
- **islower(c)** si c est une minuscule ('a'...'z')
- **isdigit(c)** si c est un chiffre décimal ('0'...'9')
- **isalpha(c)** si islower(c) ou isupper(c)
- **isalnum(c)** si isalpha(c) ou isdigit(c)
- **isxdigit(c)** si c est un chiffre hexadécimal ('0'...'9' ou 'A'...'F' ou 'a'...'f')
- **isspace(c)** si c est un signe d'espacement (' ', '\t', '\n', '\r', '\f')

Les fonctions de conversion suivantes fournissent une valeur du type int qui peut être représentée comme caractère ; la valeur originale de c reste inchangée :

- **tolower(c)** retourne c converti en minuscule si c est une majuscule
- **toupper(c)** retourne c converti en majuscule si c est une minuscule

Quelques fonctions de <ctype.h>

Les fonctions de classification suivantes fournissent un résultat du type int différent de zéro, si la condition respective est remplie, sinon zéro.

- **isupper(<c)** si <c> est une majuscule ('A'...'Z')
- **islower(c)** si c est une minuscule ('a'...'z')
- **isdigit(c)** si c est un chiffre décimal ('0'...'9')
- **isalpha(c)** si islower(c) ou isupper(c)
- **isalnum(c)** si isalpha(c) ou isdigit(c)
- **isxdigit(c)** si c est un chiffre hexadécimal ('0'...'9' ou 'A'...'F' ou 'a'...'f')
- **isspace(c)** si c est un signe d'espacement (' ', '\t', '\n', '\r', '\f')

Les fonctions de conversion suivantes fournissent une valeur du type int qui peut être représentée comme caractère ; la valeur originale de c reste inchangée :

- **tolower(c)** retourne c converti en minuscule si c est une majuscule
- **toupper(c)** retourne c converti en majuscule si c est une minuscule

Quelques fonctions de <ctype.h>

Les fonctions de classification suivantes fournissent un résultat du type int différent de zéro, si la condition respective est remplie, sinon zéro.

- **isupper(<c)** si <c> est une majuscule ('A'...'Z')
- **islower(c)** si c est une minuscule ('a'...'z')
- **isdigit(c)** si c est un chiffre décimal ('0'...'9')
- **isalpha(c)** si islower(c) ou isupper(c)
- **isalnum(c)** si isalpha(c) ou isdigit(c)
- **isxdigit(c)** si c est un chiffre hexadécimal ('0'...'9' ou 'A'...'F' ou 'a'...'f')
- **isspace(c)** si c est un signe d'espacement (' ', '\t', '\n', '\r', '\f')

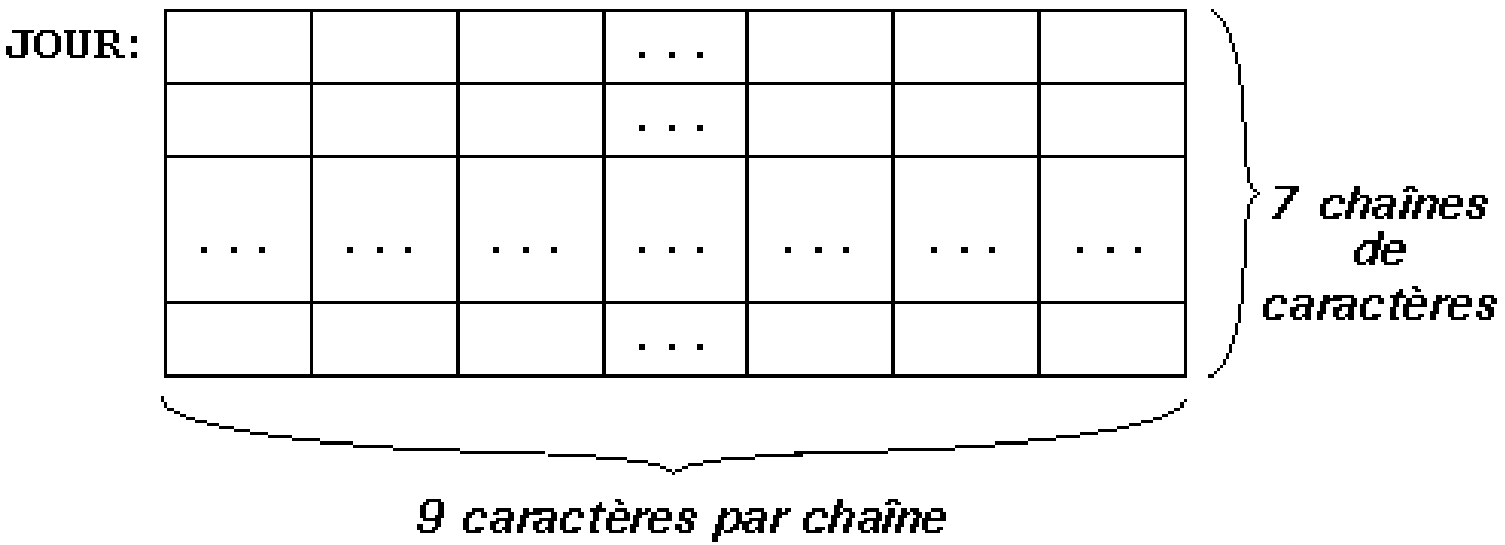
Les fonctions de conversion suivantes fournissent une valeur du type int qui peut être représentée comme caractère ; la valeur originale de c reste inchangée :

- **tolower(c)** retourne c converti en minuscule si c est une majuscule
- **toupper(c)** retourne c converti en majuscule si c est une minuscule

Tableaux de chaînes de caractères

- Un tableau de chaînes de caractères correspond à un tableau à deux dimensions du type char, où chaque ligne contient une chaîne de caractères.

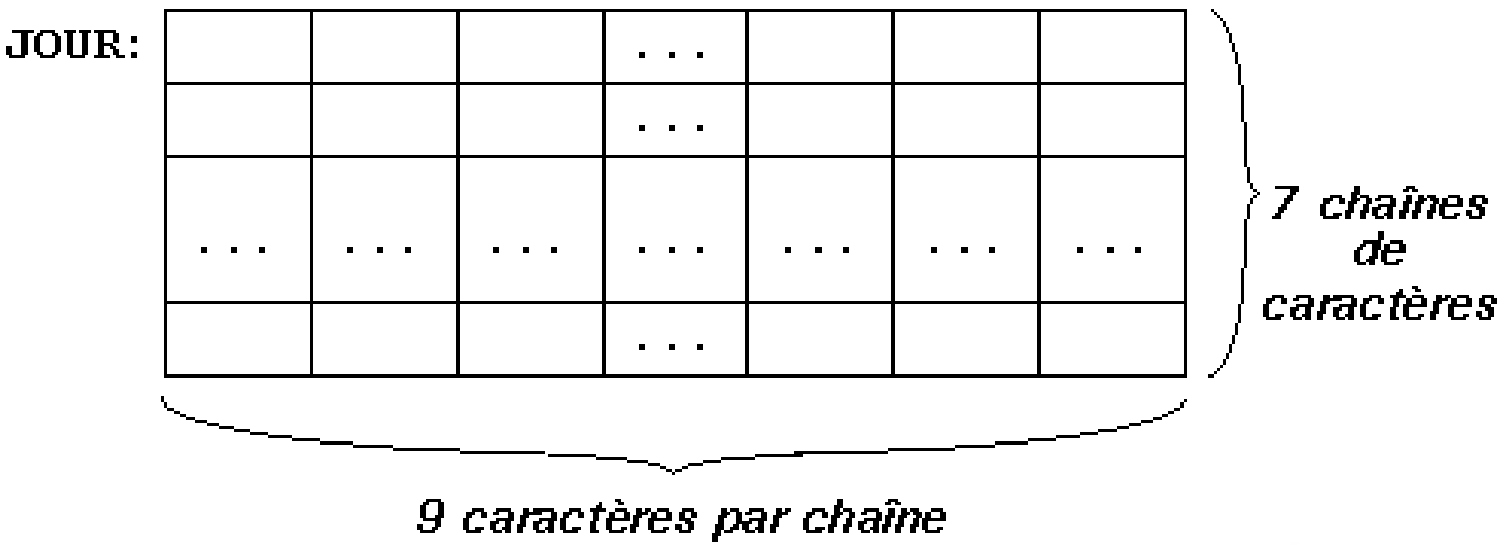
- Déclaration : **char JOUR[7][9] ;** réserve l'espace en mémoire pour 7 mots contenant 9 caractères (dont 8 caractères significatifs).



Tableaux de chaînes de caractères

- Un tableau de chaînes de caractères correspond à un tableau à deux dimensions du type char, où chaque ligne contient une chaîne de caractères.

- Déclaration : **char JOUR[7][9] ;** réserve l'espace en mémoire pour 7 mots contenant 9 caractères (dont 8 caractères significatifs).



Initialisation d'un tableau de chaînes de caractères

- Il est possible d'accéder aux différentes chaînes de caractères d'un tableau, en indiquant simplement la ligne correspondante.

```
char JOUR[7][9]= {"lundi", "mardi", "mercredi",
                  "jeudi", "vendredi",
                  "samedi", "dimanche"};
int I = 2;
printf("Aujourd'hui, c'est %s !\n", JOUR[I]);
```

- **attention :** Des expressions comme JOUR[I] représentent l'adresse du premier élément d'une chaîne de caractères. N'essayez donc pas de 'modifier' une telle adresse par une affectation directe !

- L'attribution d'une chaîne de caractères à une composante d'un tableau de chaînes se fait en général à l'aide de la fonction strcpy. Par exemple : **strcpy(JOUR[6], "Friday");**

Initialisation d'un tableau de chaînes de caractères

- Il est possible d'accéder aux différentes chaînes de caractères d'un tableau, en indiquant simplement la ligne correspondante.

```
char JOUR[7][9]= {"lundi", "mardi", "mercredi",
                  "jeudi", "vendredi",
                  "samedi", "dimanche"};
int I = 2;
printf("Aujourd'hui, c'est %s !\n", JOUR[I]);
```

- **attention :** Des expressions comme JOUR[I] représentent l'adresse du premier élément d'une chaîne de caractères. N'essayez donc pas de 'modifier' une telle adresse par une affectation directe !

- L'attribution d'une chaîne de caractères à une composante d'un tableau de chaînes se fait en général à l'aide de la fonction strcpy. Par exemple : **strcpy(JOUR[6], "Friday") ;**

Initialisation d'un tableau de chaînes de caractères

- Il est possible d'accéder aux différentes chaînes de caractères d'un tableau, en indiquant simplement la ligne correspondante.

```
char JOUR[7][9]= {"lundi", "mardi", "mercredi",
                  "jeudi", "vendredi",
                  "samedi", "dimanche"};
int I = 2;
printf("Aujourd'hui, c'est %s !\n", JOUR[I]);
```

- **attention :** Des expressions comme JOUR[I] représentent l'adresse du premier élément d'une chaîne de caractères. N'essayez donc pas de 'modifier' une telle adresse par une affectation directe !

- L'attribution d'une chaîne de caractères à une composante d'un tableau de chaînes se fait en général à l'aide de la fonction strcpy. Par exemple : **strcpy(JOUR[6], "Friday");**

Accès aux différents éléments d'un tableaux de chaînes de caractères

Lors de la déclaration il est possible d'initialiser toutes les composantes du tableau par des chaînes de caractères constantes :

```
char JOUR[7][9]= {"lundi", "mardi", "mercredi",
                  "jeudi", "vendredi", "samedi",
                  "dimanche"};
```

JOUR:									
	'l'	'u'	'n'	'd'	'i'	'\0'			
	'm'	'a'	'r'	'd'	'i'	'\0'			
	'm'	'e'	'r'	'c'	'r'	'e'	'd'	'i'	'\0'
	...	...	...	...	...	...	...	...	...
	'd'	'i'	'm'	'a'	'n'	'c'	'h'	'e'	'\0'