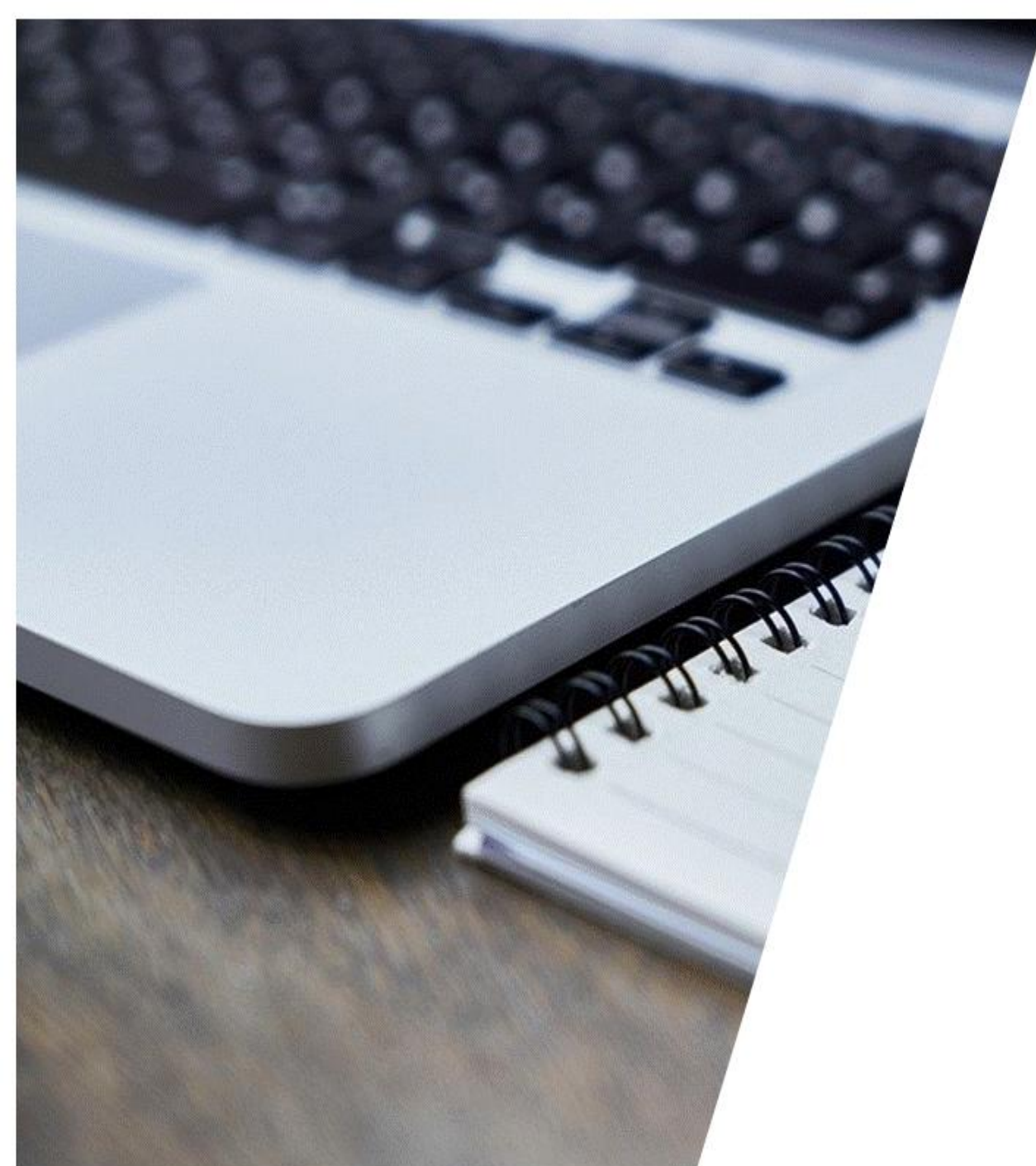




C 語言

字元.字串與數



字元型資料：包含字元常數與字元變數

字元常數

字元常數是用單引號括起來的一個字元。

例如：

'a'、'b'、'='、'+'、'？'

都是合法字元常數。

```
1 #include <stdio.h>
2 int main() {
3     char ch;
4     ch='a';
5     printf("%c\n", ch);
6     return 0;
7 }
```

資料型態	型別	格式符號
整數	int	%d
浮點數	float	%f
字元	char	%c

字元型資料：包含字元常數與字元變數

字元變數用來存儲字元，即單一字元（如：'a'）。
字元變數的類型說明符是char。字元變數類型定義的格式和書寫規則都與整型變數相同。

例如：

```
char a,b;  
char text;
```

```
#include <stdio.h>  
  
int main(){  
    char text;  
    scanf("%c",&text);  
    printf("%c\n",text);  
    return 0;  
}
```

資料型態	型別	格式符號
整數	int	%d
浮點數	float	%f
字元	char	%c

每個字元變數被分配一個位元組的記憶體空間，
因此只能存放一個字元

字串型資料：包含字串常數與字串變數

字串常數是指在程式中直接使用的字串值，例如："hello"。在 C 語言中，字串常數是以雙引號括起來的一連串字元。當我們在程式碼中使用字串常數時，編譯器會將這些字串常數存儲在記憶體中的唯讀區域（read-only memory），因此無法修改字串常數的值。

在宣告字串陣列時，我們可以直接將字串常數賦值給陣列，例如：

```
1 #include <stdio.h>
2 int main() {
3     char ch[ ]="hello";
4     printf("%s\n", ch);
5     return 0;
6 }
```

這樣就會將 "hello" 字串複製到名為 str 的字符陣列中。請注意，字符陣列的大小要足夠大以容納整個字串，包括**結尾的 null 字元 '\0'**。

資料型態	型別	格式符號
整數	int	%d
浮點數	float	%f
字元	char	%c
字串	char[]	%s

字串型資料：包含字串常數與字串變數

字串變數：下面的例子宣告ch 是一個字串，長度為 255個字元。我們在宣告字串的時候要注意它的長度，以免位數不夠造成程式錯誤。事實上，**每個字串後面都有一個 '\0' 的字元**。

由於 s 本身就是這個字元陣列的位址，所以在 scanf 裡我們不需要加上 & 符號。

```
1 #include <stdio.h>
2 int main()
3 {
4     char ch[255];
5     scanf("%s", ch);
6     printf("%s\n", ch);
7     return 0;
8 }
```

資料型態	型別	格式符號
整數	int	%d
浮點數	float	%f
字元	char	%c
字串	char[]	%s

輸入：
Hello
輸出：
Hello

為什麼輸入字串時不需要 & 符號?

在 C 語言中，`scanf("%s", ch);` 這裡的 `ch` 代表的是字元陣列 `ch` 的起始記憶體位址。

陣列名稱的特性：當您直接使用陣列名稱（例如 `ch`）時，它會自動轉換成指向陣列第一個元素的指標。換句話說，`ch` 本身就已經代表了 `&ch[0]`，也就是 `ch` 陣列第一個元素的記憶體位址。

`scanf` 的運作方式：`scanf` 函數需要知道要將讀取到的資料儲存到哪個記憶體位置。對於字串（`%s`）來說，它需要的是一個字元陣列的起始位址，以便將讀取到的字元依序存入陣列中，直到遇到空白字元或達到陣列的容量上限。

因此，當您寫 `scanf("%s", ch);` 時，`ch` 已經是陣列的起始位址，不需要再額外加上 `&` 符號。如果加上 `&` 符號，變成 `scanf("%s", &ch);`，那就會變成傳遞整個陣列的位址，而不是陣列第一個元素的位址，這會導致程式行為未定義，甚至可能出錯。

簡單來說：

`ch` -> 陣列 `ch` 的起始位址 (也就是 `&ch[0]`)

`&ch` -> 整個陣列 `ch` 的位址

字元與ASCII

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
000	00	0000000	000	[NULL]	048	30	0110000	060	0	096	60	1100000	140	`
001	01	0000001	001	[START OF HEADING]	049	31	0110001	061	1	097	61	1100001	141	a
002	02	0000010	002	[START OF TEXT]	050	32	0110010	062	2	098	62	1100010	142	b
003	03	0000011	003	[END OF TEXT]	051	33	0110011	063	3	099	63	1100011	143	c
004	04	0000100	004	[END OF TRANSMISSION]	052	34	0110100	064	4	100	64	1100100	144	d
005	05	0000101	005	[ENQUIRY]	053	35	0110101	065	5	101	65	1100101	145	e
006	06	0000110	006	[ACKNOWLEDGE]	054	36	0110110	066	6	102	66	1100110	146	f
007	07	0000111	007	[BELL]	055	37	0110111	067	7	103	67	1100111	147	g
008	08	0001000	010	[BACKSPACE]	056	38	0111000	070	8	104	68	1101000	150	h
009	09	0001001	011	[HORIZONTAL TAB]	057	39	0111001	071	9	105	69	1101001	151	i
010	0A	0001010	012	[LINE FEED]	058	3A	0111010	072	:	106	6A	1101010	152	j
011	0B	0001011	013	[VERTICAL TAB]	059	3B	0111011	073	;	107	6B	1101011	153	k
012	0C	0001100	014	[FORM FEED]	060	3C	0111100	074	<	108	6C	1101100	154	l
013	0D	0001101	015	[CARRIAGE RETURN]	061	3D	0111101	075	=	109	6D	1101101	155	m
014	0E	0001110	016	[SHIFT OUT]	062	3E	0111110	076	>	110	6E	1101110	156	n
015	0F	0001111	017	[SHIFT IN]	063	3F	0111111	077	?	111	6F	1101111	157	o
016	10	0010000	020	[DATA LINK ESCAPE]	064	40	1000000	100	@	112	70	1110000	160	p
017	11	0010001	021	[DEVICE CONTROL 1]	065	41	1000001	101	A	113	71	1110001	161	q
018	12	0010010	022	[DEVICE CONTROL 2]	066	42	1000010	102	B	114	72	1110010	162	r
019	13	0010011	023	[DEVICE CONTROL 3]	067	43	1000011	103	C	115	73	1110011	163	s
020	14	0010100	024	[DEVICE CONTROL 4]	068	44	1000100	104	D	116	74	1110100	164	t
021	15	0010101	025	[NEGATIVE ACKNOWLEDGE]	069	45	1000101	105	E	117	75	1110101	165	u
022	16	0010110	026	[SYNCHRONOUS IDLE]	070	46	1000110	106	F	118	76	1110110	166	v
023	17	0010111	027	[ENG OF TRANS. BLOCK]	071	47	1000111	107	G	119	77	1110111	167	w
024	18	0011000	030	[CANCEL]	072	48	1001000	110	H	120	78	1111000	170	x
025	19	0011001	031	[END OF MEDIUM]	073	49	1001001	111	I	121	79	1111001	171	y
026	1A	0011010	032	[SUBSTITUTE]	074	4A	1001010	112	J	122	7A	1111010	172	z
027	1B	0011011	033	[ESCAPE]	075	4B	1001011	113	K	123	7B	1111011	173	{
028	1C	0011100	034	[FILE SEPARATOR]	076	4C	1001100	114	L	124	7C	1111100	174	
029	1D	0011101	035	[GROUP SEPARATOR]	077	4D	1001101	115	M	125	7D	1111101	175	}
030	1E	0011110	036	[RECORD SEPARATOR]	078	4E	1001110	116	N	126	7E	1111110	176	~
031	1F	0011111	037	[UNIT SEPARATOR]	079	4F	1001111	117	O	127	7F	1111111	177	[DEL]
032	20	0100000	040	[SPACE]	080	50	1010000	120	P					
033	21	0100001	041	!	081	51	1010001	121	Q					
034	22	0100010	042	"	082	52	1010010	122	R					
035	23	0100011	043	#	083	53	1010011	123	S					
036	24	0100100	044	\$	084	54	1010100	124	T					
037	25	0100101	045	%	085	55	1010101	125	U					
038	26	0100110	046	&	086	56	1010110	126	V					
039	27	0100111	047	'	087	57	1010111	127	W					
040	28	0101000	050	(	088	58	1011000	130	X					
041	29	0101001	051	)	089	59	1011001	131	Y					
042	2A	0101010	052	*	090	5A	1011010	132	Z					
043	2B	0101011	053	+	091	5B	1011011	133	[					
044	2C	0101100	054	,	092	5C	1011100	134	\					
045	2D	0101101	055	-	093	5D	1011101	135	]					
046	2E	0101110	056	.	094	5E	1011110	136	^					
047	2F	0101111	057	/	095	5F	1011111	137	_					

字元：

在電腦中是以一個八位元的整數來儲存(即 1 Byte)，符號與數字的對應關係我們稱為 ASCII 碼 (American Standard Code for Information Interchange)

字元的本質是數字

```
#include <stdio.h>

int main(){
    char cl;
    scanf( "%c", &cl);
    printf( "%d\n", cl);
    return 0;
}
```

輸入：
a
輸出：
97

- 因為字元的本質是數字，數字本身就可以計算
- 以上的範例，是輸入一個字元，然後印出代表此字元的數字
- a字元的值是97
- 查一下ASCII，是不是也是97呢？

問題

寫一個程式，使用標準輸入函數
輸入1個大寫英文字母，輸出其小寫
字母。

輸入範例：

A

輸出範例：

a



大小寫之間的數值是有規則性的變化的

097	61	1100001	141	a	065	41	1000001	101	A
098	62	1100010	142	b	066	42	1000010	102	B
099	63	1100011	143	c	067	43	1000011	103	C
100	64	1100100	144	d	068	44	1000100	104	D
101	65	1100101	145	e	069	45	1000101	105	E
102	66	1100110	146	f	070	46	1000110	106	F
103	67	1100111	147	g	071	47	1000111	107	G
104	68	1101000	150	h	072	48	1001000	110	H
105	69	1101001	151	i	073	49	1001001	111	I
106	6A	1101010	152	j	074	4A	1001010	112	J
107	6B	1101011	153	k	075	4B	1001011	113	K
108	6C	1101100	154	l	076	4C	1001100	114	L
109	6D	1101101	155	m	077	4D	1001101	115	M
110	6E	1101110	156	n	078	4E	1001110	116	N
111	6F	1101111	157	o	079	4F	1001111	117	O
112	70	1110000	160	p	080	50	1010000	120	P
113	71	1110001	161	q	081	51	1010001	121	Q
114	72	1110010	162	r	082	52	1010010	122	R
115	73	1110011	163	s	083	53	1010011	123	S
116	74	1110100	164	t	084	54	1010100	124	T
117	75	1110101	165	u	085	55	1010101	125	U
118	76	1110110	166	v	086	56	1010110	126	V
119	77	1110111	167	w	087	57	1010111	127	W
120	78	1111000	170	x	088	58	1011000	130	X
121	79	1111001	171	y	089	59	1011001	131	Y
122	7A	1111010	172	z	090	5A	1011010	132	Z

利用運算，就可以做出無窮的變化，例如密碼的編碼也是其中有趣的問題。