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 <iostream>
2  using namespace std;
3  int main(){
4      char ch,charin;
5      ch='a';
6      cout<<ch<<endl;
7      return 0;
8  }
```

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 charin;

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      char charin;
5      cin>>charin;
6      cout<<charin<<endl;
7      return 0;
8  }
```

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 <iostream>
2  using namespace std;
3  int main()
4  {
5      string words="hello";
6      cout<<words<<endl;
7      return 0;
8  }
```

這樣就會將 “hello” 字串複製到名為 words的字串陣列中。

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words 是一個字串，。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。

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      string words;
6      cin>>words;
7      cout<<words<<endl;
8      return 0;
9  }
```

輸入：
Hello
輸出：
Hello

字元與ASCII

ASCII TABLE

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

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

```c
#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

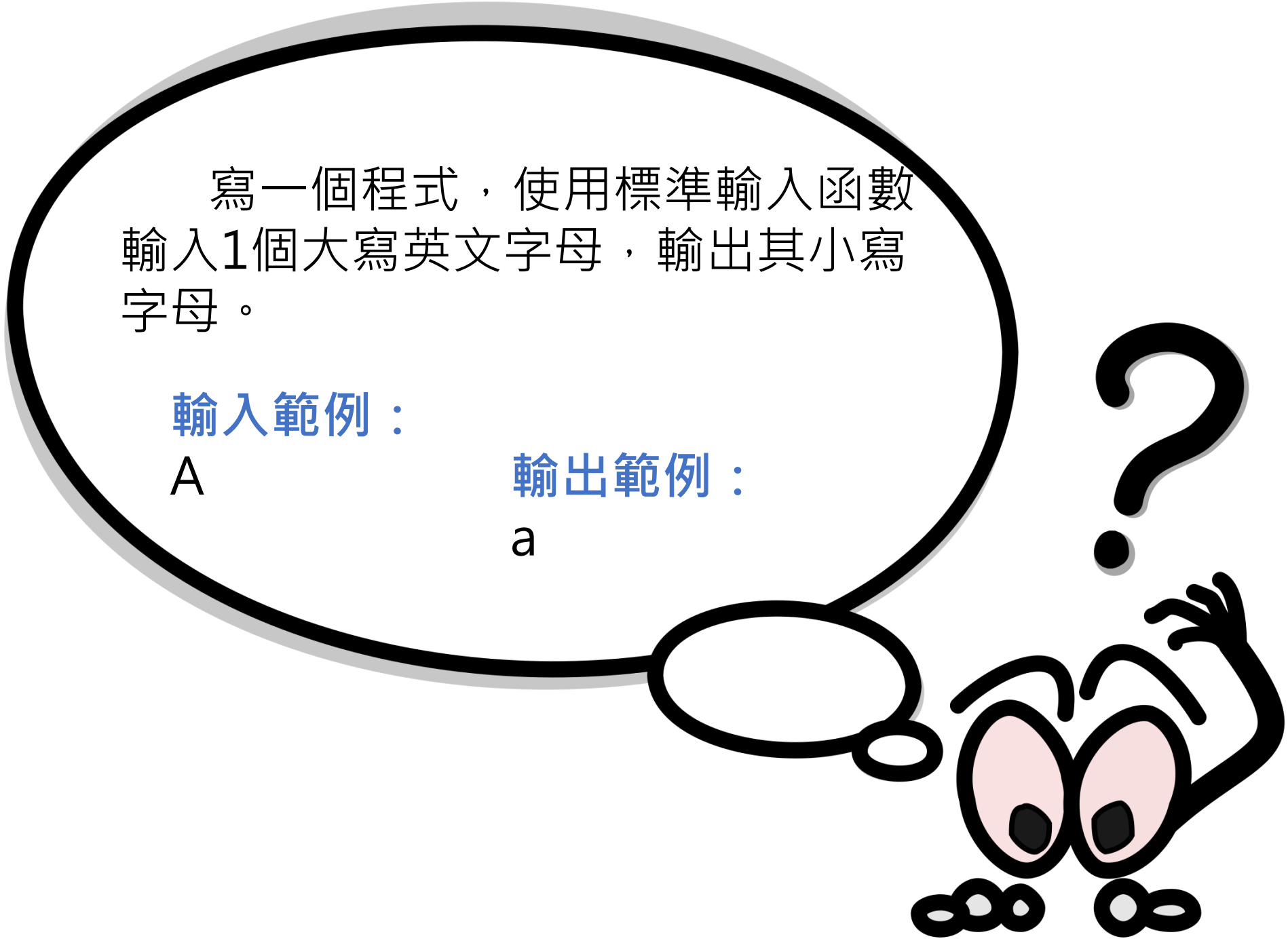
寫一個程式，使用標準輸入函數輸入1個大寫英文字母，輸出其小寫字母。

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。