<Adv C & App/>

Advanced C Programming And It's Application

String I: Character and String Basic

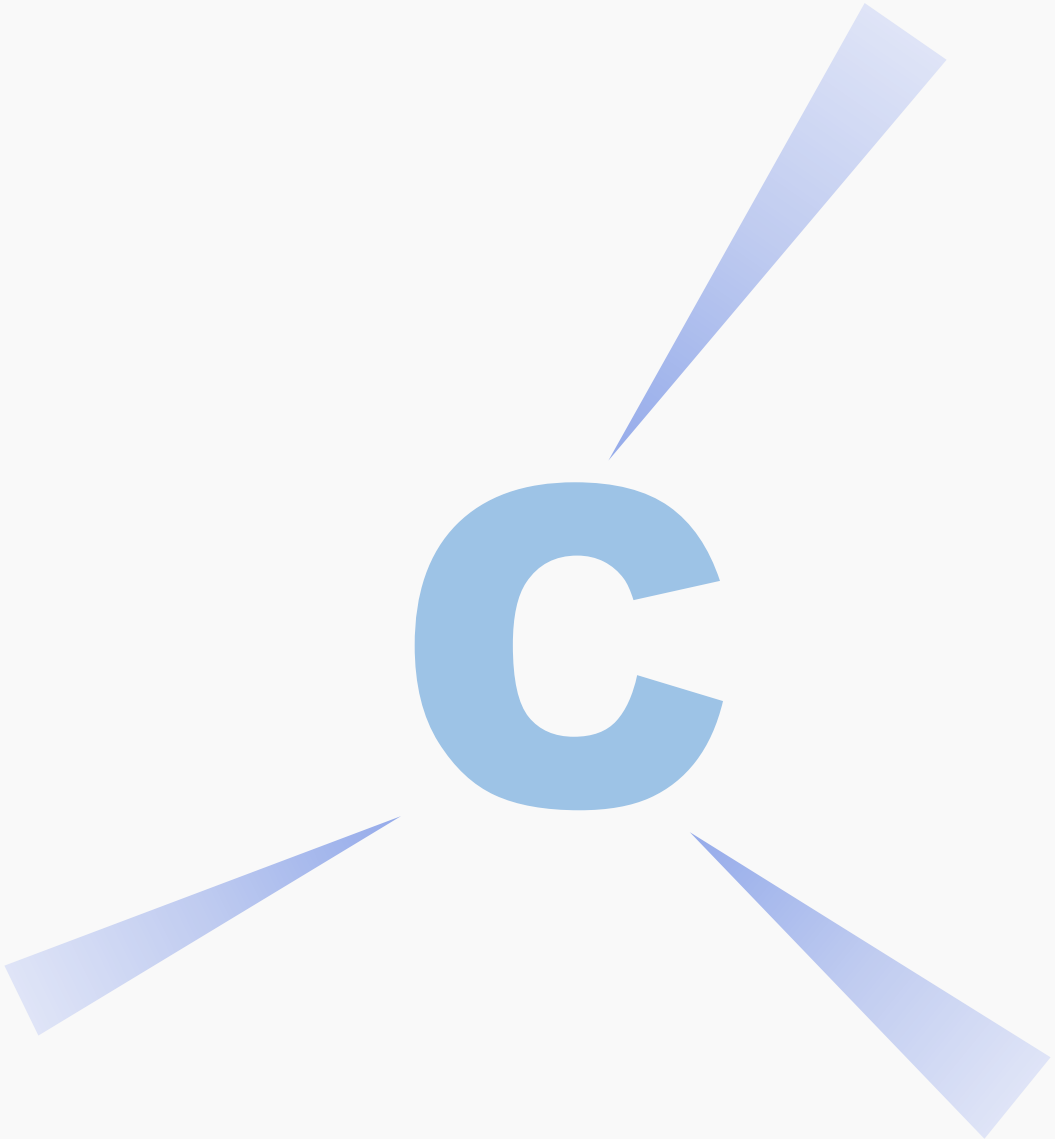
Assistant Prof. Chan, Chun-Hsiang

Department of Artificial Intelligence, Tamkang University

Dec. 08, 2021

</ Adv C & App >

大綱



字元

在第一堂課的時候，我們介紹過字元，但那時我們並沒有深入去介紹這個資料型態。在今天的課程會比較詳細一點，去來介紹字元的一些特性。

首先，字元的宣告使用的文字char，一樣要搭配一個變數名稱，不同的是給定的值，必須使用單引號括弧起來，例如:

```
// refer to Ex 10-1 Declare a Char Variable
char c1 = 'a';
char c2 = 'b';
```

字元

再來提到的是我們要怎麼輸出跟讀取字元?

輸出的時候我們需要使用要的格式化符號為「%c」；同理，讀取的時候也是需要使用「%c」，例如:

Lab 10-1:

利用Ex 10-1實做看看，輸入以下測試資料會得到甚麼結果? 請解釋輸出結果的原因。

(1) a (2) 1 (3) ab (4) 1234。

```
#include <stdio.h>
int main(){
        /*Ex 10-1: Read and Print a Char */
        printf("Ex 10-1: Read and Print a Char\n");
        char c;
        printf("Please enter a char:\n");
        scanf("%c", &c);
        printf("The char is %c\n", c);
}
```

特殊字元

特殊字元其實在我們之前的課程中，大家就使用了很多次。

Expression	Meaning	Expression	Meaning
' '	space	'\a'	alarm
'\t'	tab	'\b'	backspace
'\n'	newline	'\\'	backslash

Lab 10-2:

設計一個程式，將分別使用上述所有的特殊字元。

字元比較

我們在教數值的時候，有提到說可以利用==或是!=，來判斷是否為相同或不同的數值；再者，也可以利用大於小於的方式，比較數字的大小。

那個同樣的道理也可以在字元裡面實做出來，只是意義上有一點不同:

```
/*Ex 10-3: Char Comparison */
printf("Ex 10-3: Char Comparison\n");
char c1 = 'a', c2 = 'a', c3 = 'A', c4 = '9';
if (c1==c2){
        printf("c1 (%c) and c2 (%c) are equal!\n", c1, c2);
} else{
        printf("c1 (%c) and c2 (%c) are different!\n", c1, c2);
}
if (c1>c3){
        printf("c1 (%c) is larger than c3 (%c)!\n", c1, c3);
} else if (c1==c3){
        printf("c1 (%c) and c3 (%c) are equal!\n", c1, c3);
} else{
        printf("c1 (%c) is smaller than c3 (%c)!\n", c1, c3);
}
if (c1>c4){
        printf("c1 (%c) is larger than c4 (%c)!\n", c1, c4);
} else if (c1==c4){
        printf("c1 (%c) and c4 (%c) are equal!\n", c1, c4);
} else{
        printf("c1 (%c) is smaller than c4 (%c)!\n", c1, c4);
}
```

字元比較

透過Ex 10-3，我們可以知道 a > A and a > 9。

那麼究竟 A 跟 9 誰比較大呢?

Lab 10-3:

設計一個程式，證明[a-z]、[A-Z]與[0-9]之間的大小順序為何?

字元比較

還記得我們在很久之前的課程，要大家利用ASCII編碼印出Hello嗎？ 其實當時我們看的編碼對應表就有答案了!

順序依序為:

1. **0-9**
2. **A-Z**
3. **a-z**

二進位	十進位	十六進位	圖形	二進位	十進位	十六進位	圖形	二進位	十進位	十六進位	圖形
0010 0000	32	20	(space)	0100 0000	64	40	@	0110 0000	96	60	`
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r
0011 0011	51	33	3	0101 0011	83	53	S	0111 0011	115	73	s
0011 0100	52	34	4	0101 0100	84	54	T	0111 0100	116	74	t
0011 0101	53	35	5	0101 0101	85	55	U	0111 0101	117	75	u
0011 0110	54	36	6	0101 0110	86	56	V	0111 0110	118	76	v
0011 0111	55	37	7	0101 0111	87	57	W	0111 0111	119	77	w
0011 1000	56	38	8	0101 1000	88	58	X	0111 1000	120	78	x
0011 1001	57	39	9	0101 1001	89	59	Y	0111 1001	121	79	y
0011 1010	58	3A	:	0101 1010	90	5A	Z	0111 1010	122	7A	z
0011 1011	59	3B	;	0101 1011	91	5B	[	0111 1011	123	7B	{
0011 1100	60	3C	<	0101 1100	92	5C	\	0111 1100	124	7C	|
0011 1101	61	3D	=	0101 1101	93	5D	]	0111 1101	125	7D	}
0011 1110	62	3E	>	0101 1110	94	5E	^	0111 1110	126	7E	~
0011 1111	63	3F	?	0101 1111	95	5F	_	0111 1111	127	7F	DEL (delete)

字元比較

如果仔細觀察右表，每個符號都可以對應到十進位一個數字；換句話說，當我們做+/-的運算符號時，也可以同時更新我們要用的字元。

```
#include <stdio.h>
int main(){
        /*Ex 10-4: Char Iteration */
        printf("Ex 10-4: Char Iteration\n");
        int i;
        char c = 'a';
        for (i=0; i<26; i++){
                printf("%c\t", c+i);
        }
}
```

```
Ex 10-4: Char Iteration
a   b   c   d   e   f   g   h   i   j   k   l   m   n   o   p   q   r   s   t   u   v   w   x   y   z
```

二進位	十進位	十六進位	圖形	二進位	十進位	十六進位	圖形	二進位	十進位	十六進位	圖形
0010 0000	32	20	(space)	0100 0000	64	40	@	0110 0000	96	60	`
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r
0011 0011	51	33	3	0101 0011	83	53	S	0111 0011	115	73	s
0011 0100	52	34	4	0101 0100	84	54	T	0111 0100	116	74	t
0011 0101	53	35	5	0101 0101	85	55	U	0111 0101	117	75	u
0011 0110	54	36	6	0101 0110	86	56	V	0111 0110	118	76	v
0011 0111	55	37	7	0101 0111	87	57	W	0111 0111	119	77	w
0011 1000	56	38	8	0101 1000	88	58	X	0111 1000	120	78	x
0011 1001	57	39	9	0101 1001	89	59	Y	0111 1001	121	79	y
0011 1010	58	3A	:	0101 1010	90	5A	Z	0111 1010	122	7A	z
0011 1011	59	3B	;	0101 1011	91	5B	[	0111 1011	123	7B	{
0011 1100	60	3C	<	0101 1100	92	5C	\	0111 1100	124	7C	|
0011 1101	61	3D	=	0101 1101	93	5D	]	0111 1101	125	7D	}
0011 1110	62	3E	>	0101 1110	94	5E	^	0111 1110	126	7E	~
0011 1111	63	3F	?	0101 1111	95	5F	_	0111 1111	127	7F	DEL (delete)

getchar & putchar

之前我們都是使用scanf以及printf進行讀取使用者資訊以及印出資訊。

但其實對於字元來說，還有另一個方法可以做這件事情，那就是 getchar 與 putchar。

putchar	印出指定的字元。
getchar	取得第一個字元。

```
/*Ex 10-5: putchar, getchar */
char c4gc;
printf("Ex 10-5: putchar, getchar\n");
printf("Plz enter a word: \n");
c4gc = getchar();
putchar(c4gc); //print the variable c4gc
putchar('\n'); //newline
```

Result:

```
Ex 10-5: putchar, getchar
Plz enter a word:
abc
a
```

getchar & putchar

既然他是一次只能讀一個，如果我多寫幾次或是利用迴圈是否可以得到更多的字元呢?

```
/*Ex 10-6: putchar, getchar */
char c4gc1, c4gc2;
printf("Ex 10-6: putchar, getchar\n");
printf("Plz enter a word: \n");
c4gc1 = getchar();
c4gc2 = getchar();
putchar(c4gc1); //print the variable c4gc1
putchar(c4gc2); //print the variable c4gc2
putchar('\n'); //newline
```

Result:

```
Ex 10-6: putchar, getchar
Plz enter a word:
abc
ab
```

Char Built-in Function #include <ctype.h>

int isalnum(int c)	檢查是否為數字或字母。
int isalpha(int c)	檢查是否為字母。
int isdigit(int c)	檢查是否為十進位數字。
int islower(int c)	檢查是否為小寫字母。
int isupper(int c)	檢查是否為大寫字母。
int isspace(int c)	檢查是否為空白。
int tolower(int c)	將大寫字母轉為小寫字母。
int toupper(int c)	將小寫字母轉為大寫字母。

Char Built-in Function

```c
#include <stdio.h>
#include <ctype.h>
int main(void)
{
        /*Ex 10-7: Char Built-in Function */
        char c1 = '1', c2 = 'a', c3 = 'A';
        int i = 10, j = 'B';
        printf("Ex 10-7: Char Built-in Function\n");
        printf("[isalnum test] i=10    %d\n", isalnum(i));
        printf("[isalnum test] c1='1' %d\n", isalnum(c1));
        printf("[isalnum test] j='B'   %d\n\n", isalnum(j));
        printf("[isalpha test] i=10    %d\n", isalpha(i));
        printf("[isalpha test] c1='1'  %d\n\n", isalpha(c1));
```

```
Ex 10-7: Char Built-in Function
[isalnum test] i=10   0
[isalnum test] c1='1' 4
[isalnum test] j='B'  1

[isalpha test] i=10   0
[isalpha test] c1='1' 0
```

Char Built-in Function

```
    printf("[isdigit test] i=10   %d\n", isdigit(i));
    printf("[isdigit test] c1='1' %d\n\n", isdigit(c1));
    printf("[islower test] c2='a' %d\n", islower(c2));
    printf("[islower test] c3='A' %d\n\n", islower(c3));
    printf("[isupper test] c2='a' %d\n", isupper(c2));
    printf("[isupper test] c3='A' %d\n\n", isupper(c3));
    printf("[isspace test] ' '    %d\n", isspace(' '));
    printf("[isspace test] '\t'   %d\n\n", isspace('\t'));
    printf("[tolower test] c3='A' %c\n", tolower(c3));
    printf("[toupper test] c2='a' %c\n", toupper(c2));
}
```

```
[isdigit test] i=10   0
[isdigit test] c1='1' 1

[islower test] c2='a' 2
[islower test] c3='A' 0

[isupper test] c2='a' 0
[isupper test] c3='A' 1

[isspace test] ' '    8
[isspace test] '    '   8

[tolower test] c3='A' a
[toupper test] c2='a' A
```

<char2int/>

Char to Int

如果今天我們讀檔案的時候是用字元做讀取，那如何將字元轉成整數呢?

(1) atoi() // stdlib.h

(2) (int) char

```
Ex 10-8: Char to Int
by using atoi() => 9
by using (int)c => 57
by using (int)c-48 => 9
```

```c
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    /*Ex 10-8: Char to Int */
    printf("Ex 10-8: Char to Int\n");
    char c='9';
    int m1, m2, m3;
    m1 = atoi(&c);
    printf("by using atoi() => %d\n", m1);
    m2 = (int)c;
    printf("by using (int)c => %d\n", m2);
    m3 = (int)c-48;
    printf("by using (int)c-48 => %d\n", m3);
}
```

0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:

</char2int>

字串

字串就是由好幾個字元所構成的陣列(矩陣)，且最後一個字元是以空字元「'\0'」做結尾。

字串有自己的模組需要先import進來 → #include <string.h>

還記得陣列的宣告方式嗎? 就是用 → [大小] 的方式來做!

所以字串 = 字元陣列，字元的宣告是用 → char；因此，字串宣告就會是 char [大小]。

e.g., char [50];

字串變數宣告

再看一次定義：
字串就是由好幾個字元所構成的陣列(矩陣)，且最後一個字元是以空字元「'\0'」做結尾。

```
Ex 10-9: Declare a String
str1[5] = hello<0x14><0x1f>
str2[6] = hello
str3[50]  = hello
str4[50]  =
```

```c
#include <stdio.h>
#include <string.h>
int main(){
        /*Ex 10-9: Declare a String */
        printf("Ex 10-9: Declare a String\n");
        char str1[5] = "hello";
        char str2[6] = "hello";
        char str3[50] = "hello";
        char str4[50] = "";
        printf("str1[5] = %s\n", str1);
        printf("str2[6] = %s\n", str2);
        printf("str3[50]  = %s\n", str3);
        printf("str4[50]  = %s\n", str4);
}
```

Length, copy

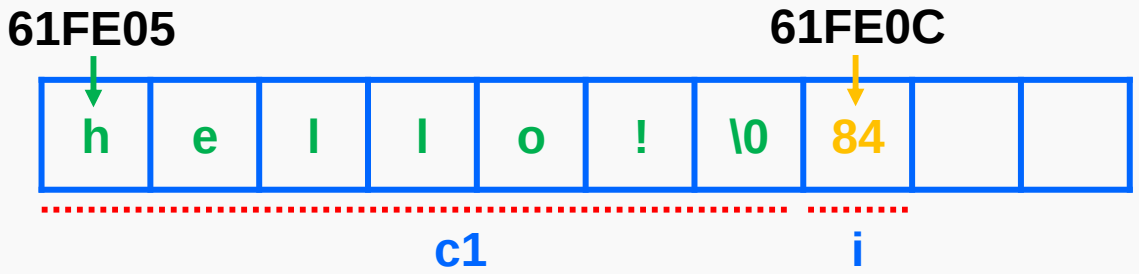
在這個部分會介紹三個常用的string函數:

1. strlen() :: 用來計算該字串變數的長度，不包含最後的空白字元。
2. strcpy(var, string) :: 用來複製字串string，指到變數var裡面。
3. strncpy(var, string, count) :: 用來複製需要的字元數count的字串string，指到變數var裡面。

*不過複製字串的時候，會遇到一些問題...

strlen & strcpy

```
Ex 10-10: Length & Copy
The original string: hello! (length: 6; size: 7; int i: T)
RAM address: 000000000061FE0C 000000000061FE05
The strcpy's string: A-Wond (length: 6; size: 7; int i: T)
RAM address: 000000000061FE0C 000000000061FE05
```




```
/*Ex 10-10: Length & Copy */
int i = 84;
char c1[7] = "hello!";
printf("Ex 10-10: Length & Copy\n");
printf("The original string: %s (length: %d; size: %d; int i: %c)\n", c1,
        strlen(c1), sizeof(c1), i);
printf("RAM address: %p %p\n", &i, &c1);
strcpy(c1, "A-Wond");
printf("The strcpy's string: %s (length: %d; size: %d; int i: %c)\n", c1,
        strlen(c1), sizeof(c1), i);
printf("RAM address: %p %p\n", &i, &c1);
```

strncpy

```
/*Ex 10-11: strncpy */
char c1[4] = "abc";
char c2[7] = "hello!";
printf("Ex 10-11: strncpy\n");
printf("The original string: %s (length: %d; size: %d; c1: %s)\n", c2,  strlen(c2), sizeof(c2), c1);
printf("RAM address: %p %p\n", &c1, &c2);

strncpy(c2, "A-Wonder", 6);
printf("[count= 6] The strncpy's string: %s (length: %d; size: %d; c1: %s)\n", c2, strlen(c2),
        sizeof(c2), c1);
strncpy(c2, "A-Wonder", 7);
printf("[count= 7] The strncpy's string: %s (length: %d; size: %d; c1: %s)\n", c2, strlen(c2),
        sizeof(c2), c1);
```

```
The original string: hello! (length: 6; size: 7; c1: abc)
RAM address: 000000000061FE0C 000000000061FE05
[count= 6] The strncpy's string: A-Wond (length: 6; size: 7; c1: abc)
[count= 7] The strncpy's string: A-Wondeabc (length: 10; size: 7; c1: abc)
```

strncpy

```
The original string: hello! (length: 6; size: 7; c1: abc)
RAM address: 000000000061FE0C 000000000061FE05
[count= 6] The strncpy's string: A-Wond (length: 6; size: 7; c1: abc)
[count= 7] The strncpy's string: A-Wondeabc (length: 10; size: 7; c1: abc)
[count= 8] The strncpy's string: A-Wonderbc (length: 10; size: 7; c1: rbc)
[count= 9] The strncpy's string: A-Wonder (length: 8; size: 7; c1: r)
[count=10] The strncpy's string: A-Wonder (length: 8; size: 7; c1: r)
[count=11] The strncpy's string: A-Wonder (length: 8; size: 7; c1: r)
```

```
strncpy(c2, "A-Wonder", 8);
printf("[count= 8] The strncpy's string: %s (length: %d; size: %d; c1: %s)\n", c2,
        strlen(c2), sizeof(c2), c1);
strncpy(c2, "A-Wonder", 9);
printf("[count= 9] The strncpy's string: %s (length: %d; size: %d; c1: %d)\n", c2,
        strlen(c2), sizeof(c2), c1);
strncpy(c2, "A-Wonder", 10);
printf("[count=10] The strncpy's string: %s (length: %d; size: %d; c1: %d)\n", c2,
        strlen(c2), sizeof(c2), c1);
strncpy(c2, "A-Wonder", 11);
printf("[count=11] The strncpy's string: %s (length: %d; size: %d; c1: %d)\n", c2,
        strlen(c2), sizeof(c2), c1);
```

copy

char *strcpy(char *dest, const char *src) 複製字符串src指向到dest。
char *strncpy(char *dest, const char *src, size_t n) 副本最多n個字符的字符串src指向到dest。

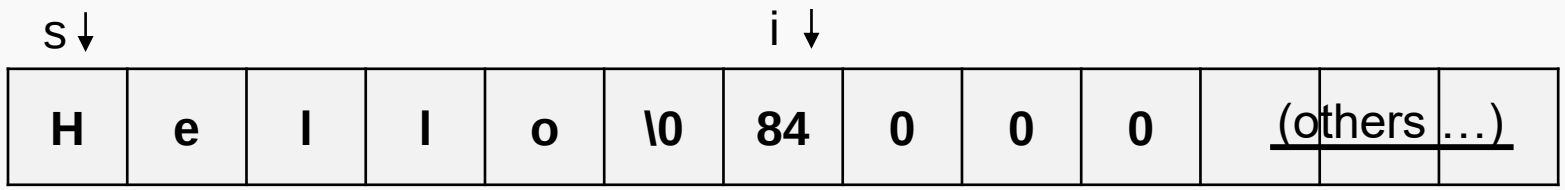
那甚麼時候會出現undefined behavior呢?

strcpy	strncpy
(1) 緩衝區溢位 (2) dest和src有重疊的部分 (3) 如果dest指到的不是一個字元陣列 (4) 如果src指向的字串沒有以'\0'結尾	(1) 緩衝區溢位 (2) dest和src有重疊的部分 (3) 如果src或dest其中一個指到的不是一個字元陣列 (4) 如果dest不夠長 (比count短) (5) 如果src指向的字串沒有以'\0'結尾

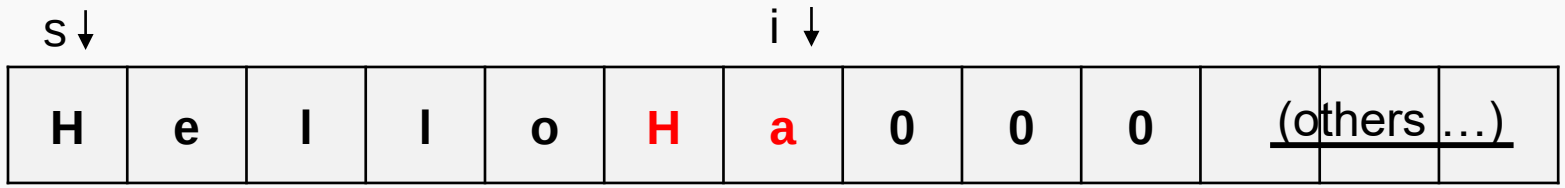
緩衝區溢位

```
int i = 84;
char s[6] = "Hello";
```

那這個時候他們在記憶體中是長甚麼樣子呢?



strcpy(s, "**HelloHa**");



a = 97 (according to ASCII dex table); therefore, i now is 97 not 84.

Lab 10-4:

如果字串是Helloo，請問變數 i 印出來的是甚麼? 為甚麼?

Ref: ASCII Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Source: https://www.asciitable.com/

strcpy & strncpy

Lab 10-5:

從Ex10-11中，我們利用不同長度的複製方式，可以發現到 i 印出來的東西不一樣，那是因為緩衝區溢位的關係。

故此這個練習中，請利用word檔說明，當我要複製一個8字元的字串時，至少需要多大的字串才能完整的接收，且不會有緩衝區溢位的問題。並以Ex10-11的方式，Demo說明最適大小的產生原因。

作業一

建立一個身分證號碼驗證機:

參考資料
https://bit.ly/3siNsqj
https://bit.ly/2UilqyF

身分證字號: 英文字母 + 九位數字

英文字母代表出生地
第一位數字代表性別
第二位至第八位數字為流水號碼
第九位數字為檢查號碼

臺北市	臺中市	基隆市	臺南市	高雄市	新北市	宜蘭縣	桃園市	嘉義市	新竹縣	苗栗縣
A	B	C	D	E	F	G	H	I	J	K
10	11	12	13	14	15	16	17	34	18	19

南投縣	彰化縣	新竹市	雲林縣	嘉義縣	屏東縣	花蓮縣	臺東縣	金門縣	澎湖縣	連江縣
M	N	O	P	Q	T	U	V	W	X	Z
21	22	35	23	24	27	28	29	32	30	33

臺中縣	臺南縣	高雄縣	陽明山管理局
L	R	S	Y
20	25	26	31

作業一

建立一個身分證號碼驗證機:

驗證方式:
假設有一組身分證字號為A169651244 Ref: https://bit.ly/3sjbZvy

字元	A		1	6	9	6	5	1	2	4	4
轉換	1	0									檢查號碼
權重	x1	x9	x8	x7	x6	x5	x4	x3	x2	x1	

$$(1 \times 1 + 0 \times 9 + 1 \times 8 + 6 \times 7 + 9 \times 6 + 6 \times 5 + 5 \times 4 + 1 \times 3 + 2 \times 2 + 4 \times 1) + 4 \times 1$$
$$= (1 + 0 + 8 + 42 + 54 + 30 + 20 + 3 + 4 + 4) + 4$$
$$= 166 + 4$$
$$= 170$$

170為10的倍數; i.e., 170 ➔ 10n。
故此身分證字號為真!

作業一

printf Please input an Taiwanese ID number!

scanf // User input

printf Answer Value

printf "This is correct a Taiwanese ID number!" if correct

printf "This is NOT correct a Taiwanese ID number!" if incorrect

References

https://openhome.cc/Gossip/CGossip/index.html
https://edisonx.pixnet.net/blog/post/35305668
https://www.learn-c.org/
http://tw.gitbook.net/cprogramming/
https://blog.techbridge.cc/2020/05/03/simple-c-language-introduction-tutorial/
https://openhome.cc/Gossip/CGossip/StringLengthCopyCat.html
https://www.huaweicloud.com/articles/12640716.html
https://skylinelimit.blogspot.com/2018/02/c-2.html
https://www.learn-c.org/en/Strings