第 三 章

字串處理

第一節　字元與字串

一、 字元與 ASCII

C++ 中的**字元** (Character) char 其實是儲存一個 0 到 255 的整數，在電腦中有一個**符號表**，每個符號都有他各自的編號。輸出字元時，計算機就會自動將 char 裡面的整數去查符號表，印出對應符號，這個表格我們稱爲 **ASCII 碼**。雖然 char 印出來是符號，但實際上儲存的是整數。

ASCII 碼網路上都能查到，在這邊只提到幾個重要的觀念就好，各位讀者可以邊看網路上查到的 ASCII 碼邊看接下來的內容。

字元表示方法是用單引號包上一個符號，如：'0'。每個符號都有一個編號，例如 '0' 的編號是 48，其他常用的字元編號如表 3.1。

字元	'0'	'A'	'a'	'␣'	'\n'
編號	48	65	97	32	10
備註				空白	換行字元

表 3.1: 常用字元編號

(一) 特殊字元

有些字元因爲沒辦法直接用單引號包住符號的方式表示，就會用「倒斜線+字元」來代表，表 3.2 是一些常見的特殊字元。

字元	意義	備註
'\n'	換行字元	
'\t'	Tab	
'\r'	迴車鍵	Windows 系統中以 \r\n 代表換行
'\''	單引號	
'\"'	雙引號	
'\0'	空字元	用來代表字串的結束
'\\'	倒斜線	倒斜線被用做跳脫字元，因此要用兩個倒斜線表示

表 3.2: 常用特殊字元

(二) 常用技巧：字元判斷

在 ASCII 碼中，有三個區塊是連續的：

- 數字區塊：'0' 到 '9'
- 大寫字母區塊：'A' 到 'Z'
- 小寫字母區塊：'a' 到 'z'

因爲 char 實際上是存整數，所以可以用大於小於來判斷，程式碼 3.1 可以判斷一個字元是否是數字字元。

```
1 bool myIsDigit(char ch) {
2   return '0' <= ch && ch <= '9';
3 }
```

程式碼 3.1: 判斷數字字元

在 <cctype> 中有一些可以判斷字元類型的函式，常用的如表 3.3，留意這些函式傳進去的參數型態就直接是 int。

不過臨時記不得這些函式，筆者建議自己寫一個，不難寫。

(三) 常用技巧：計算數字

用同樣的概念，可以把一個數字字元「轉換」成 int 的 0。字元 '0' 的編號爲 48，因此我們將 '0' - 48 就可以取得實際的 int 值，但這個方法稍嫌笨重，因爲我們

函式	範圍	意義
int isalnum(int)	A 到 Z、a 到 z、0 到 9	判斷是否爲英文字母或數字字元
int isalpha(int)	A 到 Z、a 到 z	判斷是否爲英文字母
int isdigit(int)	0 到 9	判斷是否爲數字字元
int islower(int)	a 到 z	判斷是否爲小寫字母
int isupper(int)	A 到 Z	判斷是否爲大寫字母

表 3.3: <cctype> 常用函式

必須記得 '0' 的編號，在 C++ 中有提供字元相減的方法，於是我們可以寫爲程式碼 3.2，將一個數字字元減去 '0'。

```
1 int charToNumber(char ch) {
2   return ch - '0';
3 }
```

程式碼 3.2: 數字字元轉換爲 int

同樣的方法也可以用在具有連續區間的大寫字母、小寫字母。

二、 C++ 字串與 C 字串

C++ 的字串爲 string 物件，需要引入標頭檔 <string>。這裡要講 C 字串，C 字串事實上就是**字元陣列**，宣告如程式碼 3.3。

```
1 char str[110];
```

程式碼 3.3: C 字串宣告

使用 C 語言的字串時，就有很多東西要注意：第一、字串本身是用 '\0' 來當結尾，**注意**！不是阿拉伯數字的 '0' 而是 ASCII 碼爲 0 的 '\0' (表 3.2)，代表「字串的結尾」。

```
1 char str[10] = "bird";
```

程式碼 3.4: C 字串

例如程式碼 3.4 的字串爲 "bird"，事實上儲存情形如圖 3.1。

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
b	i	r	d	\0	?	?	?	?	?

圖 3.1: 字串就是字元陣列

因爲 '\0' 被當作「字串的結尾」(很重要所以還要再說一次)，所有 C 字串函數的操作都會跟這個有關。

三、 C 字串函數

C 字串函數都在 <cstring> 底下，以下解說幾個常用的 C 字串函數，如程式碼 3.5。

```
size_t strlen(const char *str);
int    strcmp(const char *str1, const char *str2);
char*  strcpy(char* dest, const char* src);
char* strncpy(char* dest, const char* src, size_t num);
char*  strcat(char* dest, const char* src);
char* strncat(char* dest, const char* src, size_t num);
```

程式碼 3.5: 常用的字串函數

(一) strlen 函數

strlen 函數可以知道一個 C 字串的長度，參數就是傳一個字串指標，這個函數就是從一開始的位址往後掃，直到碰到 '\0' 爲止，並回傳長度值。

```
strlen("bird");
```

程式碼 3.6: strlen 範例

程式碼 3.6 回傳的長度值爲 4。當然我們也可以對字串變數做 strlen，程式碼 3.7 中 num 字串的長度爲 10。

```cpp
char num[110] = "0123456789";
strlen(num);
```

程式碼 3.7: strlen 範例

此外，strlen 是**函式**，它和 C++ 字串的 str.size() 不同的地方在於：每次呼叫 strlen 就會重新計算一次字串長度，像程式碼 3.8a 的寫法非常浪費時間，每次迴圈都會重新呼叫一次 strlen。

```cpp
for (i = 0; i < strlen(str); i++)
{
  // do something
}
```

(a) 直觀寫法

```cpp
int len = strlen(str);
for (i = 0; i < len; i++)
{
  // do something
}
```

(b) 較好的寫法

程式碼 3.8: 注意 strlen 的用法

但很多時候字串的長度沒有改變，因此會浪費很多時間，比較好的做法是用一個變數儲存長度，再下去執行迴圈，如程式碼 3.8b。

補充，雖然現代的編譯器大多會去辨別這一情況加以優化，但盡量不要依賴編譯器，最好從一開始養成好習慣，才不會被坑。

(二) strcmp 函數

strcmp 函數用來比較兩個 C 字串，參數爲兩個字串指標。它會從兩個字串一開始的字元逐個比較，比較到**其中一個**爲 '\0' 爲止，strcmp 的功能和 C++ 字串之間用 ==、>、< 來比較的功能類似。回傳值分成三類：

- 等於 0：代表兩個字串相等。
- 大於 0：代表 str1 的字典順序大於 str2，通常是回傳 1。
- 小於 0：代表 str1 的字典順序小於 str2，通常是回傳 -1。

(三) strcpy 函數

strcpy 函數就是把 src 字串複製到 dest 字串，複製的原理和上述函式類似：就是從第一個字元開始複製，直到遇到 '\0' 爲止。接著對 dest 最尾端補 '\0'。strcpy 相當於在 C++ 中 dest = src。

使用 strcpy 時需要特別**注意**，strcpy 函數**不會檢查** dest 的長度，換句話說，如果 src 比 dest 長，strcpy 會持續複製直到複製完爲止，這對程式而言是非常危險的，因爲寫出字串的範圍很有可能會寫到很重要的記憶體，然後**電腦就炸了**。

其他較爲安全的替代方案，就是使用 strncpy 函數，strncpy 的第三個參數代表**最多**複製幾個字元，不過要注意 strncpy 只負責複製字元，複製完後**不會補** '\0'。

(四) strcat 函數

strcat 函數主要是把 src 字串接到 dest 字串後面，它是從 dest 字串的**第一個** '\0' 開始串接，因此也和 strcpy 一樣可能會造成寫出記憶體的問題，對應的函數是 strncat。strcat 函數相當於 C++ 字串中，直接做 dest += src。

其它有關字串的函數還有：strstr、strchr、strrchr、strtok、strspn 等，這些函數在此不贅述。

四、 字串轉換

實作上，C++ 字串操作上來得 C 字串**容易** (有運算子多載)，但因爲 C++ 字串是物件，因此操作 C 字串會比 C++ 字串來得快。我們可以選擇混合使用這兩種字串，在不難處理又快速時我們選擇 C 字串；而在 C 字串比較難解決的事情時，我們可以利用方便的 C++ 字串來處理。

(一) C++ 字串轉 C 字串

如果要將一個 C++ 字串轉成 C 字串，我們利用 <string> 中的一個成員函式 c_str()。

```
1 string str = "bird";
2 cout << str.c_str() << endl;
```

程式碼 3.9: C++ 字串轉 C 字串

要注意的是，c_str() 產生一個**唯讀**的 C 字串。

(二) C 字串轉 C++ 字串

正確將 C 字串轉成 C++ 字串的方法則是將 C 字串丟進 C++ 字串中，如程式碼 3.10。

```
1 char str1[110] = "bird";
2 string str2 = str1;
3 cout << str2.size() << endl;
4 cout << str2 << endl;
```

程式碼 3.10: C 字串轉 C++ 字串

五、 字串練習

可能有些人看完上面的敘述後，可能還不是很理解這些函數的用途。要理解 C 字串的用法不難，但如果要完全掌握住這些函數，還需要配合指標的觀念，以下問題，筆者就不寫上解答，留給讀者思考、討論。

1. 假設現在有一個字串

```
1 char str1[110] = "abcdefghijklmnopqrstuvwxyz";
```

(a) 它的長度爲何？

(b) 用 sizeof 和 strlen 有何不同呢？

(c) 請問下面的語句和上面又有什麼差異呢？

```
1 strlen("abcdefghijklmnopqrstuvwxyz");
2 sizeof("abcdefghijklmnopqrstuvwxyz");
```

2. 有一字串

```
1 char str2[110] = "bird";
2 char str3[4];
```

(a) 我們可以用 strcpy(str3, str2) 嘛？

(b) 如果不行，我們使用 strncpy(str3, str2, 4); 來避免超出記憶體呢？試著印出來觀察看看。

3. 有一字串

```
char str4[110] = "cat\0bird";
```

(a) 這個字串的長度爲何？

(b) strlen(str4 + 4) 的回傳值爲何？strlen(str4 + 8) 呢？

(c) 若我們使用 strcpy(str4, "dog")，會得到什麼結果？此時執行下面語句會有什麼反應呢？

```
cout << str4 << "␣" << str4 + 4 << endl;
```

(d) 若我們對原先的 str4 執行 strcat(str4, "dog")，有得到你預期的結果嘛？

(e) 要是 strcat(str4 + 4, "dog") 呢？

4. 假設有一 C++ 字串和一 C 字串

```
string str5 = "bird";
char str6[110];
```

要怎樣將 str5 複製給 str6 呢？

5. 現有兩字串

```
char str7[110] = "cat\0bird";
char str8[110];
```

試比較 strcpy(str8, str7); 和 memcpy(str8, str7, sizeof(str7)); 的不同。

第二節　輸入與輸出

一、 格式字串

(一) scanf 和 printf

C++ 中常使用的輸入輸出是 cin 和 cout，有時候 cin 和 cout 的速度並不能滿足我們的需求，這時候就需要使用 C 語言本身的輸入輸出。

C 語言的輸入輸出在 <cstdio> 內，輸入的函數爲 scanf，而輸出的函數爲 printf。

注意！C 語言輸入輸出是**函式**，這兩個函式的用法和原本 C++ 的相比較爲繁瑣，但也有比較方便的地方。

```
1 int  scanf(const char *format, ...);
2 int printf(const char *format, ...);
```

程式碼 3.11: scanf 和 printf

scanf 和 printf 的參數中，後面 ... 稱爲**不定參數**，代表參數的數量是不固定的，決定參數的數量是靠前面的 format 字串決定，這個字串我們稱爲**格式字串**。

格式字串可以像我們平常輸出字串一樣作法，如程式碼 3.12。

```
1   printf("Hello␣world!\n");
```

程式碼 3.12: 輸出字串

記得 endl 屬於 C++ 當中的換行，在此須使用換行字元 '\n' 來換行。輸出整數變數，在格式字串中我們用 "%d" 來代表，每個 "%d" 都代表著一個整數，如程式碼 3.13。

程式碼 3.13a 直接印出數字，也可以寫爲 printf("3\n");，程式碼 3.13b 會印出變數 x 的值，印出的位置在 "%d" 處。

輸入的話，因爲我們是呼叫函數，若要改到變數值就需要使用傳址呼叫，因此輸入一個整數寫爲程式碼 3.14。

```c
printf("%d\n", 3);
```

(a) 印出常數

```c
int x = 3;
printf("%d\n", x);
```

(b) 印出 int 變數

程式碼 3.13: 印出整數

```c
int x;
scanf("%d", &x);
```

程式碼 3.14: 輸入一個整數

同樣地，unsigned int、long long、unsigned long long、float、double 都有對應的的格式，如表 3.4。

型態	對應格式	備註
unsigned int	"%u"	
long long	"%lld"	windows 環境下可能會用 "%I64d"
unsigned long long	"%llu"	windows 環境下可能會用 "%I64u"
float	"%f"	
double	"%lf"	printf 時須用 "%f"
char	"%c"	
char[]	"%s"	C++ 的字串 string 沒辦法直接使用 scanf、printf

表 3.4: scanf 和 printf 格式表

因爲 % 在 scanf 和 printf 當中作爲跳脫字元，印出 % 要使用 "%%"。

和 cin 類似，scanf 中也不會用空白和換行，如程式碼 3.15。

```c
scanf("%d%d%d", &a, &b, &c);
```

程式碼 3.15: scanf 範例

(二) 字元讀取問題

讀取字元時，cin 和 scanf 的行爲會不一致。

```
1 char a, b;
2 cin >> a >> b;
3 scanf("%c%c", &a, &b);
4 printf("\'%c\' \'%c\'\n", a, b);
```

程式碼 3.16: cin 和 scanf 行爲不一致

程式碼 3.16 第 2 行和第 3 行中可以嘗試輸入「1 2」，可以發現 cin 會跳過空白，可是 scanf 並不會。

(三) scanf_s 函數

VC++ 在讀取字串時很多時候會被擋下，因爲 scanf 讀取字串不會檢查長度，會有安全問題。scanf_s 函式在讀取字串時要多傳一個參數，作爲最多讀取的長度，程式碼 3.17 爲 scanf_s 的一個範例。

```
1 char str[5];
2 scanf_s("%s", str, 4);
```

程式碼 3.17: scanf_s 的範例

程式碼 3.17 中，因爲 str 字串最後要有 '\0' 字元，因此最多只能讀 4 個字元。

到目前爲止，可看出 scanf 和 printf 在用法上比 cin 和 cout 繁瑣，除了在效能上的優勢外，還有什麼其他優點呢？

(四) 進位輸出

C++ 中整數提供八進位、十進位、十六進位的輸出方法，分別是使用 std::oct 、std::dec 、 std::hex 這三種，如程式碼 3.18a。

程式碼 3.18b 是對應的版本，分別使用 "%o"、"%d"、"%x"。如要將十六進位印爲大寫，cout 須加上 std::uppercase，printf 則使用 "%X"。

(五) iomanip 格式

在 C++ 中，標頭檔 <iomanip>，專門做輸入輸出的處理。表 3.5 列出 <iomanip> 中常用的**串流操縱符 (Stream manipulator)**，也就是用來串在 cin、cout 的東西。

```cpp
int a = 11;
cout << oct << a << endl;
cout << dec << a << endl;
cout << hex << a << endl;
```

(a) cout 版本

```cpp
int a = 11;
printf("%o\n", a); // 13
printf("%d\n", a); // 11
printf("%x\n", a); // b
```

(b) printf 版本

程式碼 3.18: 輸出進制比較

操縱符	作用
std::setprecision	設定精準度
std::setw	設定輸出寬度
std::setfill	設定填充字元

表 3.5: <iomanip> 常用操縱符

std::setprecision 通常是用來限定輸出數字的精準度，例如程式碼 3.19，setprecision 的參數爲 5，因此會保留 5 位有效位數。

```cpp
double f = 3.14159;
cout << setprecision(5) << f << endl; // 3.1415
cout << fixed << setprecision(5) << f << endl; // 3.14159
```

程式碼 3.19: 設定有效位數

當 setprecision 加上 std::fixed 的話，會變作印出小數點後 n 位，也就是**四捨五入**，如程式碼 3.19 第 3 行。

std::setw 代表輸出的寬度，例如程式碼 3.20，輸出的結果爲 "␣␣␣16"。

```cpp
int a = 16;
cout << setw(5) << a << endl;
```

程式碼 3.20: 設定輸出寬度

std::setfill 可以設定空白處的填充字元，傳入的參數是一個字元，通常和 std::setw 混用，如程式碼 3.21。

```
cout << setw(5) << setfill('x') << 16 << endl;
```

程式碼 3.21: 設定填充字元

可以試試看以下程式碼，在此不贅述。

- `cout << setw(3) << 55688 << endl;`
- `cout << setfill('x') << setw(5) << left << 16 << endl;`
- `cout << setfill('x') << setw(5) << right << 16 << endl;`
- `cout << setfill('x') << setw(5) << internal << -16 << endl;`

printf 格式 printf 本身就有內建和 <iomanip> 相似的功能，比如我們可以設定輸出寬度、印出小數點後 n 位、填充前導零等，如程式碼 3.22。

```
printf("%5d\n", 16); // 設定寬度
printf("%.3f", 3.14159); // 小數點後 3 位
printf("%05d\n", 16); // 前導 0，結果爲 00016
```

程式碼 3.22: printf 格式範例

scanf 格式 學習 scanf 最有價值的是他可以對輸入的格式做設定，比如要讀入一個時間的格式「hh:mm」，cin 一般需要多使用一個變數來讀入 ':'，如程式碼 3.23a。

```
int hh, mm;
char ch;
cin >> hh >> ch >> mm;
```

(a) 用 cin 輸入

```
int hh, mm;
scanf("%d:%d", &hh, &mm);
```

(b) 用 scanf 輸入

程式碼 3.23: 比較 cin 與 scanf 差異

scanf 可以直接設定格式，省去多使用一個變數的麻煩，如程式碼 3.23b。

(六) 回傳值

scanf 也可以使用在 0 尾版、n 行版等迴圈中，不難，在此集中講解 EOF 版。

當 scanf 讀到檔尾時，會回傳一個常數 EOF (數值通常是 -1)，因此 EOF 版就會寫爲程式碼 3.24。

```
while (scanf("%d", &n) != EOF) {
  // do something ...
}
```

程式碼 3.24: EOF 版

二、 標準 I/O

(一) 簡介

(二) 行讀取

(三) 字元讀取

(四) 輸入輸出優化

三、 檔案 I/O

(一) 開檔讀檔

(二) fscanf 和 fprintf

(三) freopen

四、 字串 I/O

(一) sscanf 和 sprintf

(二) stringstream

C++ 中也有提供字串 I/O，稱爲 stringstream 類別，在 <sstream> 標頭檔裡面，用法與 cin、cout 差不多，如程式碼 3.25。

程式碼 3.25 會印出 "Hello␣world"! 字串，再來看看 stringstream 怎麼解決 sscanf 遇到迴圈無法解決的事，如程式碼 3.26。

```cpp
stringstream ss;
string str;
ss << "Hello world!"; // 將東西塞進 stringstream
ss >> str; // 丟到字串
cout << str << endl; // Hello world!
```

程式碼 3.25: stringstream 基本用法

```cpp
int a;
stringstream ss;
string str = "1 2 3 4 5";
ss << str;
for (int i = 0; i < 5; i++) {
  ss >> a;
  cout << a << endl;
}
```

程式碼 3.26: 可以用迴圈來讀取

```cpp
while (ss >> a) {
  // do something
}
```

程式碼 3.27: stringstream 碰到 EOF

第三節　字串技巧

一、 善用 index

二、 回文

三、 二維問題

四、 子字串

五、 其他

第四節　字串應用

一、 羅馬數字

對大多數的人而言，這並不是一個陌生的主題。羅馬數字用一些特別的字母來當做某個數字，如表 3.6。

數字	1	5	10	50	100	500	1000
符號	I	V	X	L	C	D	M

表 3.6: 羅馬數字

羅馬數字系統遵守兩個原則：

- **加法原則**：透過累加符號遞增數字。例如：數字 1 寫爲「I」、數字 2 寫爲「II」、3 表示爲「III」，以此類推。
- **減法原則**：爲簡化書寫，4 不寫作「IIII」，而是用「$5-1$」寫爲「IV」。

總結來說，兩種規則的區分如下：如果較小的數寫在較大的數的右邊，則爲加法；反之則爲減法。例如：11 爲 $10+1$，表示爲 XI；9 被當作 $10-1$，表示爲 IX。

除此之外，還有兩個小細節：

- **統一書寫規則**：羅馬人規定個位數由個位數決定、十位數由十位數決定，以此類推。例如，99 雖然可以視爲 $100-1$，但羅馬數字會統一看做 $90+9$，也就是 XC (90) 和 IX (9)，寫爲 XCIX。
- 符號不超過**三個**：4 會寫爲 IV 而非 IIII，9 被寫爲 IX 而非 XIIII。因此，不管怎麼湊，羅馬數字會在 3999 以內 (在此不討論更大數的表示法)。

(一) 阿拉伯數字轉羅馬數字

羅馬數字之中，個位數的符號表示個位數、十位數用來表示十位數，這之間不會混用，因此我們用「**位數**」來觀察較合適。在此我們可以用表 3.7 去觀察出個位、十位、百位的規律：

數字	1	2	3	4	5	6	7	8	9
符號	I	II	III	IV	V	VI	VII	VIII	IX
數字	10	20	30	40	50	60	70	80	90
符號	X	XX	XXX	XL	L	LX	LXX	LXXX	XC
數字	100	200	300	400	500	600	700	800	900
符號	C	CC	CCC	CD	D	DC	DCC	DCCC	CM

表 3.7: 羅馬數字找規律

可以得到以下結果：

- 個位、十位、百位之間符號不同，但「**格式**」相同，也就是只差在 IXC 用 VLD、XCM 來代換。既然格式相同，我們就可以試圖用迴圈簡化其結果。
- 若只觀察個位數，可發現 1、2、3 的符號恰好是重複符號。除此之外，1、2、3 和 6、7、8 之間只差一個「5」的符號。這代表我們可以用 `if` 把「5」的符號判斷掉，扣掉後用 1、2、3 的方式處理。
- 4 和 9 如果要取巧可能會比較困難，平常練習可以思考如何修改，若思考時間不夠建議直接用特殊判斷處理掉。

根據上面的討論，我們利用迴圈判斷的同時，我們需要利用陣列來儲存我們想要的東西，程式碼 3.28 是一個很簡便的實現方法。

```
1 #define L 3
2 string Nine[L] = {"CM", "XC", "IX"};
3 string Four[L] = {"CD", "XL", "IV"};
4 string Five[L] = {"D", "L", "V"};
5 string One[L]  = {"C", "X", "I"};
6 int    Mod[L]  = {100, 10, 1};
7 int    Mod5[L] = {500, 50, 5};
```

程式碼 3.28: 儲存符號、餘數

程式碼 3.28 簡單紀錄每一次判斷的餘數 (`Mod` 和 `Mod5` 陣列)，並且會對應到 `Five`、`One` 這兩個陣列做處理。此外，`Nine`、`Four` 陣列是針對 4 和 9 直接例外處理。

程式碼 3.29 是對應的處理方法，可以看到第 1 行針對千位數做判斷，在迴圈中第 5 行、第 7 行也是根據先前的討論，優先處理 4 和 9 的情形 (可以想一下這兩個判斷爲什麼不能反過來)。

(二) 羅馬數字轉阿拉伯數字

至於如何把羅馬數字換回阿拉伯數字？由剛剛的表格知道，除了 4 和 9 的羅馬數字是兩個字元 (由於**減法規則**)，其他都可以拆成一個字元來看待，因此我們可以藉由**記錄目前**掃到的羅馬數字，來判斷下個羅馬數字該使用加法規則還是減法規則。

```
string roman = "";
for (;n >= 1000 && n; n -= 1000)
  roman += "M"; // 千位數例外判斷
for (int i = 0; i < L; i++) { // 百位、十位、個位數
  if (n / Mod[i] == 9) // 特殊判斷 9
    roman += Nine[i], n -= 9 * Mod[i];
  if (n / Mod[i] == 4) // 特殊判斷 4
    roman += Four[i], n -= 4 * Mod[i];
  while (n >= Mod5[i]) // 判斷 5 以上
    roman += Five[i], n -= Mod5[i];
  while (n >= Mod[i])  // 剩下的部分
    roman += One[i],  n -= Mod[i];
}
```

程式碼 3.29: 阿拉伯數字轉羅馬數字

二、 字串和數字轉換

(一) 字串轉數字

一開始有提到，把一個數字字元轉換成數字的方法，若現在有一個數字字串，例如 "123"，要怎麼做呢？

程式碼 3.2 的概念可以繼續延伸，我們可以對每個字元減掉 '0' 之後再乘上對應的值，最後加總就會轉換成對應數字。

$$\begin{aligned} 123 &= 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 \\ &= (\text{'1'} - \text{'0'}) \times 10^2 + (\text{'2'} - \text{'0'}) \times 10^1 + (\text{'3'} - \text{'0'}) \times 10^0 \end{aligned}$$

實作上我們會從高位的字元往後做，以節省計算 10^n 的時間。

如程式碼 3.30，我們把剛剛的算式做轉換，可以得到如下算式：

$$123 = ((0 \times 10 + 1) \times 10 + 2) \times 10 + 3$$

除此之外，也有些人會從個位數開始做，只是這樣就要額外變數來紀錄 10 的次方。

```
int MyAtoI(string str)
{
  int res = 0, i;
  for (i = 0; i < str.size(); i++)
  {
    res *= 10;
    res += str[i] - '0';
  }
  return res;
}
```

程式碼 3.30: 從高位數開始做

(二) 數字轉字串

(三) 進位變換

三、 習題

索 引