國立屏東大學 資訊工程學系 C++程式設計入門教材

12. 字串

字串(String)可以說是我們最熟悉、同時卻也最為陌生的主題。打從一開始的Hello C++!程式，我們就已經在程式中使用了字串，但直到現在，本章才要開始為你詳細地說明字串。這是因為在C++語言中的字串是由在記憶體中一塊連續空間中的多個字元所組成，這牽涉到了陣列與指標等主題，所以一直到現在才有足夠的基礎為你介紹字串。本章將從字串常值(String Literal)開始、就字串變數、字串的輸入與輸出、相關的函式、字串陣列以及命令列引數等主題逐一進行介紹。另外，做為一個物件導向的程式語言C++也幫我們事件設計好了一個string類別，本章也將簡介其使用方法。

12.1 字串常值

所謂的字串(String)是指一些字元的集合，例如“Hello”這個用雙引號框起來的字元集合就是一個字串。在C++語言中，這種用雙引號框起來的字串又稱為字串常值(String Literals)，我們已經在過去許多範例中使用過。每個在程式中的字串常值，將會自動配置到某塊連續的記憶體空間，其中存放組成該字串的多個字元以及一個特殊的字元'\0'。'\0'通常又被稱做「空字元(Null character)」，放在字串末端做為結尾。

要特別、特別注意的是，字串常值的內容是不可以被更改的!!!

請參考figure 1，其中顯示了一個包含有“Hello”字串的記憶體空間，並以'\0'做為字串的結束。

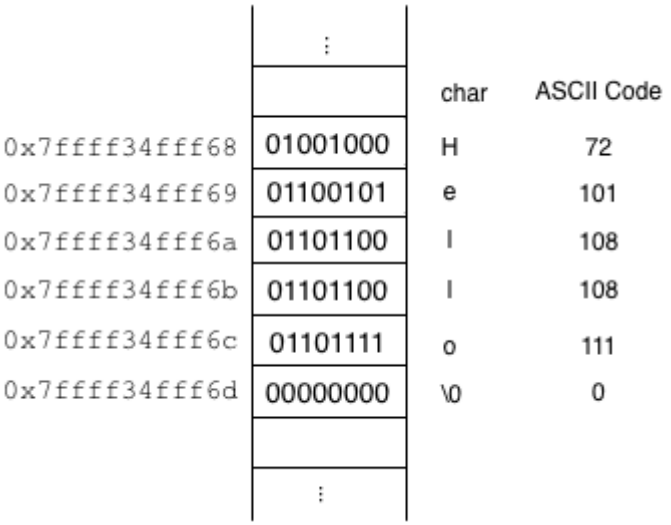

Fig. 1: 在記憶體中的字串常值

當然，你也可以figure 2來思考這個在記憶體中的字串常值。

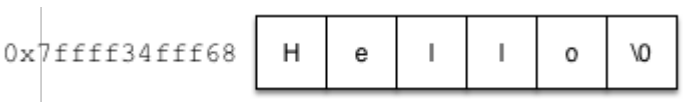

Fig. 2: 在記憶體中的字串常值

下面這個程式，以指標來檢視儲存在記憶體中的字串常值，請將它加以編譯並執行看看其結果為何。

```
#include <iostream>
using namespace std;

int main()
{
  char *p;
  char *q;
  int i;

  p="Hello";
  q=p;
  cout << p << endl;

  for(i=0;i<6;i++)
  {
    cout << *q << " = " << (int)*q << " at " << (void *)q << endl;
    q++;
  }
  return 0;
}
```

上面這個程式使用一個迴圈逐一地將字串裡的每個字元、其對應的ASCII數值，以及其所配置到的記憶體位址加以輸出。然而為了要能夠將資料以正確的型態輸出，所以在此使用了兩次的強制型態轉換：

- (int) 將char字元轉換為int整數，如此就可以輸出其對應的ASCII數值
- (void *)將char字元指標轉換為一個記憶體位址

(void *)是什麼？
(void *)是強制的型態轉換，用來將一個數值視為指標，但是是不具型態的指標 — 意即，我們不知道(或不想知道)該記憶體位址裡所存放的到底是什麼樣型態的資料。由於此處我們想要輸出的是 “ 那個記憶體位址 ”，而不是 “ 在那個記憶體位址裡面的東西 ”，所以其型態並不重要，我們它強制轉型為(void *) — 無型態指標，也就是一個記憶體位址。另外，如果不做型態轉換的話，cout會把 “ 餵 ” 給它的記憶體位址視為字串，它會將裡面的字元逐一輸出直到遇到'\0'為止。

此程式的執行結果如下：

```
Hello
H = 72 at 0x100f5fef0
e = 101 at 0x100f5fef1
l = 108 at 0x100f5fef2
l = 108 at 0x100f5fef3
```

```
o = 111 at 0x100f5fef4
 = 0 at 0x100f5fef5
```

你可以從最後一行的輸出結果看到'\0'的存在… 等等我沒看到啊！

> 因為'\0'是一個不可視字元，所以當然用 “ 眼睛 ” 是看不到的，但是用 “ 心 ” 看，有沒有看到我們所印出的ASCII數值？沒錯，'\0'的整數值為0。

當然，經過上述的說明，可能你已經想到也可以使用陣列來處理字串，請參考下面的程式：

```
#include <iostream>
using namespace std;

int main()
{
  int i;

  for(i=0;i<6;i++)
  {
           // 1                              2                                3
    cout << "Hello"[i] << " = " << (int)"Hello"[i] << " at " << (void
*)&"Hello"[i] << endl;
  }
  return 0;
}
```

前述程式中“Hello”被當成一個陣列來使用。其中第11行出現過三次“Hello”字串常值。原則上，在程式碼中，每出現一次字串常值，編譯器就會自動為其配置一個適當的記憶體空間；可是若是內容完全相同的字串常值，編譯器也會有智慧地不再重覆地配置空間。

12.2 字串變數

過去從C語言開始，就有兩種方式來宣告並處理字串變數：其一為陣列，其二為指標，直到後來的C++語言仍然可以使用。

嚴格來講，其實不應該稱為字串 “ 變數 ”，因為所謂變數存放的是單一數值，字串是由多個字元值所組成，並不符合變數的定義。正確的說法應該是 “ 字元陣列 ” 或 “ 字元指標 ”，本章後續將繼續加以說明。不過，放輕鬆一點，很多時候大家還是習慣用比較通俗的說法~~ 稱呼為字串變數的人還是蠻多的啦！

若我們要宣告一個可儲存或操作含有10個字元的字串變數str那麼我們可以使用下面的宣告:

```
char str[11];
```

這是因為考慮到字串必須以'\0'結尾。通常，我們會利用下面的方法來宣告一個字串：

```
#define STR_LEN 80
char str[STR_LEN + 1];
```

我們也可以在宣告字串變數的同時，給定其初始值，例如：

```
char str[6] = "Hello";
```

就如同figure 2所顯示的一樣，C++語言的編譯器會自動在其後幫我們增加一個'\0'做為結束的標記。上述的宣告等同於下列的程式碼：

```
char str[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

但是，如果我們的陣列宣告過大，所給的初始值不夠時又會如何呢?

> C++語言的編譯器會在其不足處皆補上'\0'，因此，以下兩行程式碼是等價的：

```
char str[7] = "Hello";
char str[7] = {'H', 'e', 'l', 'l', 'o', '\0', '\0'};
```

如果情況反過來，初始值超過陣列的大小那又會如何？在這種情況下，C++無法幫我們在字串結束處補上'\0'，因此有可能在未來的操作上出現問題，請參考下面的程式：

```
#include <iostream>
using namespace std;

int main()
{
  char str[5]="Hello";
  cout << str;
  return 0;
}
```

請試試看編譯後執行的結果如何，再試著將str[5]改成str[4]看看會發生什麼事。有些版本的C++語言編譯器會幫我們處理好這個問題，但有的不會，最好的解決方法是不要犯這種錯誤，或是改用以下的宣告方法，讓C++語言的編譯器自動幫我們處理好陣列的大小問題：

```
char str[] = "Hello";
```

我們也可以用指標的方式，來進行字串變數的宣告，例如：

```
char *strP = "World";
```

但是要注意，此處所宣告的char字元指標strP將會指向“World”字串常值。請參考下面的程式：

```
#include <iostream>
using namespace std;

int main()
{
  char strA[]="Hello";
  char *strP = "World";

  cout << strA << " " << strP << endl;

  strA[0]='h';

  *strP='w';

  cout << strA << " " << strP << endl;

  return 0;
}
```

以char strA[]=“Hello”宣告的字串，其“Hello”只是被當成是陣列的初始值，編譯器還是在記憶體內建立了一個連續的空間，用以存放在該字串中的字元；可是char *strP=“World”;在編譯的過程中，它會先在記憶體內產生一個“World”的字串常值，然後讓指標strP指向該字串常值所在的地方。由於字串常值是不可更改內容的，因此未來strP='w';的操作會在執行時發生錯誤。

12.3 字串的輸出

在C++語言裡，你可以使用cout輸出串流，並搭配相關的函式或串流操控子來進行字串的輸出，請參考以下的程式：

```
#include <iostream>
using namespace std;
```

```
#include <iomanip>

int main()
{
  char str[] = "Hello World";
  cout << str << endl;
  cout.fill('.');
  cout << setw(20) << str << endl;
  cout << left << setw(20)  << str << endl;
  return 0;
}
```

12.4 字串的輸入

我們可以cin取得使用者輸入的字串，請參考下例：

```
#include <iostream>
using namespace std;

int main()
{
  char name[20];

  cout << "Please input your name: ";

  cin >> name;
  cout << "Hello, " << name << endl;
  return 0;
}
```

此程式的執行結果如下：

```
[14:32 user@ws home] ./a.out
Please input your name: Jack
Hello, Jack
[14:32 user@ws home]
```

看起來沒問題，但如果我們輸入的名字中含有空白字元，其結果將不正確，例如：

```
[14:32 user@ws home] ./a.out
Please input your name: Jack Lin
Hello, Jack
[14:32 user@ws home]
```

注意到了嗎？那個空白字元及其後的Lin都不見了。其實cin是istream的一個物件，你可以使用它的函式來取得字串，例如get()或getline()它們可以取得使用者的輸入直到遇到換行為止，我們以cin.getline()為例，改寫上述程式如下：

```
#include <iostream>
using namespace std;

int main()
{
  char name[20];

  cout << "Please input your name:";

  cin.getline(name, 20);

  cout << "Hello, " << name <<  endl;
  return 0;
}
```

使用getline()方法需要兩個引數，第一個是存放輸入的字串，第二個則為字串的長度(最後一個字元存放'\0')。要注意getline()會將最後的換行字元從緩衝區讀入，但不會放在輸入結果內get()則會將該換行字元留在緩衝區內，因此下一次的輸入將有可能遇到問題。請參考下面的程式：

```
#include <iostream>
using namespace std;

int main()
{
  char firstname[20];
  char lastname[20];

  cout << "Please input your first name: ";
  cin.get(firstname, 20);

  cout << "Please input your last name: ";
  cin.get(lastname, 20);

  cout << "Hello, " << firstname << " " << lastname <<  endl;
  return 0;
}
```

測試看看，上面這個程式會遇到什麼問題？ 要解決這個問題，可以在第一個cin.get()後再加上一個cin.get()將緩衝區裡的換行字元清除(或是使用以前用過的ignore()函式)：

```
cin.get(firstname, 20);
cin.get();
```

或是

```
cin.get(firstname, 20).get();
```

下面這個程式解決了上述問題：

```
#include <iostream>
using namespace std;

int main()
{
  char firstname[20];
  char lastname[20];

  cout << "Please input your first name: ";
  cin.get(firstname, 20).get();

  cout << "Please input your last name: ";
  cin.get(lastname, 20);

  cout << "Hello, " << firstname << " " << lastname <<  endl;
  return 0;
}
```

再考慮下一個程式：

```
#include <iostream>
using namespace std;

int main()
{
  char firstname[20];
  char lastname[20];
  int lucky;

  cout << "Please input a lucky number: ";
  cin >> lucky;

  cout << "Please input your first name: ";
  cin.get(firstname, 20).get();

  cout << "Please input your last name: ";
  cin.get(lastname, 20);

  cout << "Hello, " << firstname << " " << lastname <<  endl;
  return 0;
}
```

執行看看，會發生什麼結果？為解決此問題，我們可以在取得字串前，先以一個cin.get()來將前一個輸入數字時所遺留下來的換行清除掉，請參考下例:

```
#include <iostream>
using namespace std;

int main()
{
  char firstname[20];
  char lastname[20];
  int lucky;

  cout << "Please input a lucky number: ";
  (cin >> lucky).get();
//  cin >> lucky;
//  cin.get();

  cout << "Please input your first name: ";
  cin.get(firstname, 20).get();

  cout << "Please input your last name: ";
  cin.get(lastname, 20);

  cout << "Hello, " << firstname << " " << lastname <<  endl;
  return 0;
}
```

12.5 C語言的字串處理函式

在C語言的函式庫中，提供了許多與字串操作相關的函式，我們也可以在C++語言裡使用它們。由於在C語言裡，那些字串處理函式是定義於string.h標頭檔中，所以在C++語言使用時，記得要使用#include <cstring>將其載入。

我們在本節列舉部份常用的字串處理函式:

- char *strcpy(char *s1, const char *s2);

將字串s2的內容複製到字串s1中，且複製完成的新字串也會傳回。

```
#include <iostream>
using namespace std;
#include <cstring>

int main()
{
```

```
  char str1[10], *str2;

  strcpy(str1, "abcd");

  cout << str1 << endl;;

  str2=strcpy(str1, "hello");

  cout << str2 << endl;

  return 0;
}
```

- char *strncpy(char *s1, const char *s2, size_t n); 將s2字串中前n個字元複製到s1中，其中size_t其實就是int型態。
- size_t strlen(const char *s); 傳回字串s的長度(不包含null character)。
- char *strcat(char *s1, const char *s2); 將字串s2的內容串接於s1之後。
- int strcmp(const char *s1, const char *s2); 比較字串s1與s2的內容，傳回值取決於s1與s2的內容：
  - 傳回0，若s1與s2相同
  - 傳回>0的值，若s1>s2
  - 傳回<0的值，若s1<s2

strcmp比較兩個字串的方法，就是一般字典在排列英文字的方法，我們稱之為Lexicographic order。若s1>s2，則：

- s1語s2前面i個字元相同，但s1的第i+1個字元小於s2。例如“abc”小於“abd”。或 “ abc”小於“bcd”。
- s1的內容與s2相同，但s1的長度小於s2。例如“abc”小於“abcd”。

至於每個字元的比較基準則是以ACSII編碼為依據：

- 數字小於字母
- 大寫字母小於小寫字母
- 空白小於字母與數字

關於更多的字串處理函式，可以參考其它相關書籍或網站。

另外，還有一些C語言的函式可用以將字串轉換為其它型態的數值，例如定義在stdlib.h裡的atoi()函式可以將字串轉成整數，請參考以下的程式：

```
#include <iostream>
using namespace std;
#include <cstdlib>

int main(void)
{
    cout << atoi("123") << endl;
    cout << atoi("45.678") << endl;
    cout << atoi("abc") << endl;
    cout << atoi("5+2") << endl;
    return 0;
```

```
}
```

其輸出結果為：

```
123
45
0
5
```

除了atoi()函式外C語言還提供了atof()函式與atol()等函式，用以將字串轉換為浮點數與長整數，這些都是定義在stdlib.h標頭檔裡，你可以視需要加以使用。

最後，再讓我看一個範例：

```
#include <iostream>
using namespace std;
#include <cstring>

int main()
{
  char *str1="Hello";
  char *str2=" World";
  char str3[20];

  cout << "str1=" << str1 << endl;
  cout << "str2=" << str2 << endl;

  strcat(str3, str1);
  strcat(str3, str2);

  cout << "The length of str1 is " << strlen(str1) << endl;
  cout << "str1+str2= " << str3 << endl;
  return 0;
}
```

12.6 字串與函式呼叫

字串也可以做為函式設計時的引數或傳回值，以引數為例，下面的程式示範了傳入一個字串，計算並傳回其中包含空白字元的個數：

```
#include <iostream>
using namespace std;
```

```
int countSpace(const char s[])
{
  int count=0, i;
  for(i=0;s[i]!='\0';i++)
  {
    if(s[i]==' ')
    {
      count++;
    }
  }
  return count;
}

int main()
{
  char str[]="This is a test.";

  cout << "There are " << countSpace(str) << " space(s) in the string." <<
endl;
  return 0;
}
```

要注意在宣告函式時const char s[]表示該引數為一個字串。更明確來說，這個引數所傳入的是一個字串所在的記憶體位址，在main()函式中呼叫時，我們以countSpace(str)將str這個字串的位址傳入即可。其中的const保證了所傳入的值不可被更改。

我們也可以使用指標的方式，來設計相同的程式，請參考下例：

```
#include <iostream>
using namespace std;

int countSpace(const char *s)
{
  int count=0;

  for(;*s!='\0';s++)
  {
    if(*s==' ')
    {
      count++;
    }
  }
  return count;
}

int main()
{
  char str[]="This is a test.";
```

```
  cout << "There are " << countSpace(str)<< " space(s) in the string." <<
endl;
  return 0;
}
```

我們也可以讓字串做為函式的傳回值，但要注意的是，此情況下僅能以char *做為函式的傳回值，不能以char []做為函式傳回值 — 因為C++不允許函式傳回多個數值：

```
#include <iostream>

char *getMessage(int i)
{
  if(i==0)
    return "Welcome!";
  else
    return "Hello!";
}

int main()
{
  char *str;
  cout << getMessage(0) << endl;

  str=getMessage(1);
  cout << str << endl;
  return 0;
}
```

12.7 字串陣列

我們也可以設計一個陣列來存放多個字串，例如：

```
#include <iostream>
using namespace std;

int main()
{
  int i;

  char month[][9] = { "January", "February", "March", "April",
                      "May", "June", "July", "August",
                      "September", "October", "November", "December"};
  for(i=0;i<12;i++)
  {
```

```
    cout << month[i] << endl;
  }
  return 0;
}
```

其實，這個程式有一個錯誤存在，請試著找出問題所在並加以更正。

我們也可以用指標的陣列來存放這些字串：

```
#include <iostream>
using namespace std;

int main()
{
  int i;

  char *month[] =   { "January", "February", "March", "April",
                      "May", "June", "July", "August",
                      "September", "October", "November", "December"};
  for(i=0;i<12;i++)
  {
    cout << month[i] << endl;
  }
  return 0;
}
```

figure 3與figure 4分別是這些字串以二維陣列與指標陣列儲存的記憶體示意圖：

	0	1	2	3	4	5	6	7	8	9
0	J	a	n	u	a	r	y	\0	\0	\0
1	F	e	b	r	u	a	r	y	\0	\0
2	M	a	r	c	h	\0	\0	\0	\0	\0
3	A	p	r	i	l	\0	\0	\0	\0	\0
4	M	a	y	\0	\0	\0	\0	\0	\0	\0
5	J	u	n	e	\0	\0	\0	\0	\0	\0
6	J	u	l	y	\0	\0	\0	\0	\0	\0
7	A	u	g	u	s	t	\0	\0	\0	\0
8	S	e	p	t	e	m	b	e	r	\0
9	O	c	t	o	b	e	r	\0	\0	\0
10	N	o	v	e	m	b	e	r	\0	\0
11	D	e	c	e	m	b	e	r	\0	\0

Fig. 3

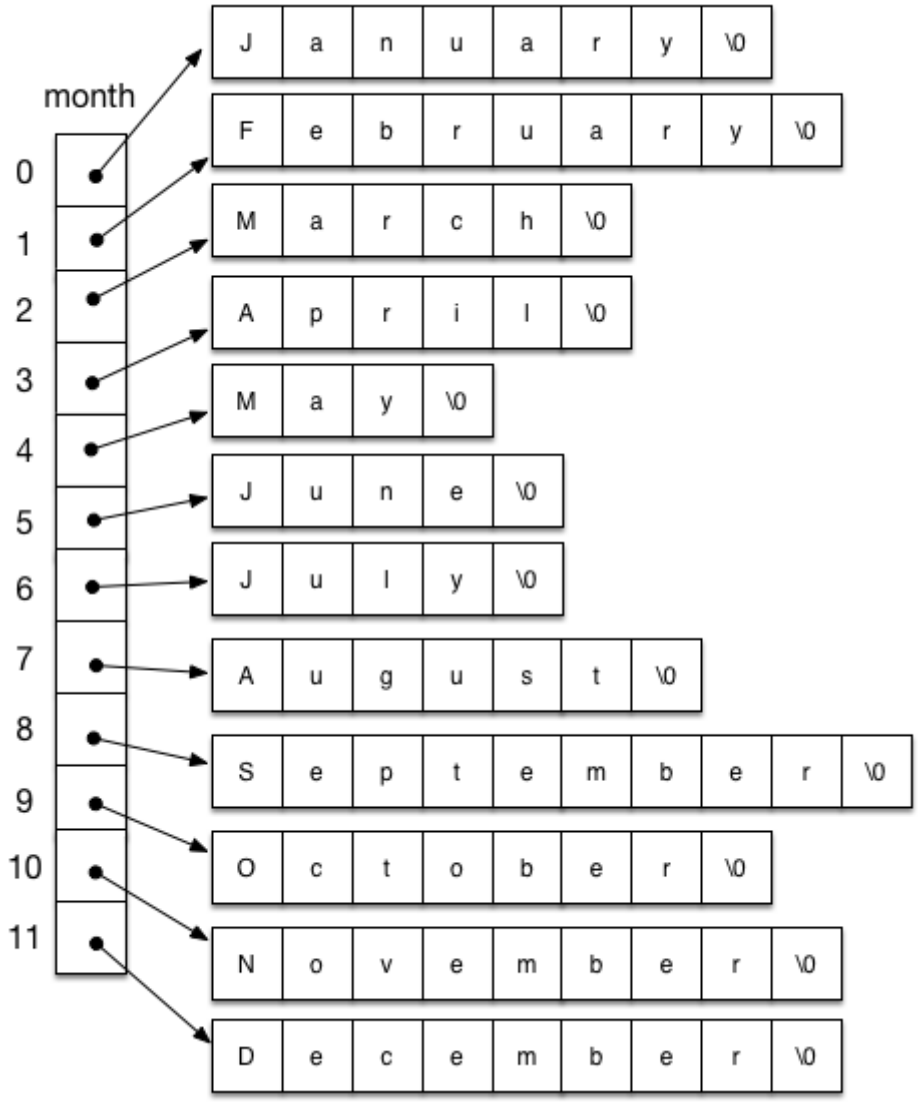

Fig. 4

12.8 命令列引數

所謂的命令列引數(Command-Line Arguments)就是在main()函式中的引數main()雖然是所謂的程式進入點，但它也是一個函式，也可以有引數。不過由於main()函式是由我們在作業系統的命令列透過執行程式而啟動，因此其引數就稱為命令列引數。例如，我們在Linux系統中，可以執行下列的指令：

```
junwu@MBA21 3rd %ls
RevisedProjects    errata        submitted
arts       on-going    website
draft      revision
junwu@MBA21 3rd % ls -l
total 0
drwxr-xr-x  27 junwu  staff   864  3 22 23:54 RevisedProjects
drwxr-xr-x  17 junwu  staff   544  9 20  2021 arts
drwxr-xr-x  35 junwu  staff  1120  7 18  2021 draft
drwxr-xr-x  18 junwu  staff   576  3 15 19:24 errata
drwxr-xr-x   5 junwu  staff   160  6 28 22:40 on-going
drwxr-xr-x   7 junwu  staff   224  7 28  2021 revision
```

```
drwxr-xr-x  35 junwu  staff  1120  6 28 22:45 submitted
drwxr-xr-x  15 junwu  staff   480  7 18  2021 website
junwu@MBA21 3rd %
```

在上述例子中，ls是我們想要執行的指令(也就是一個可執行的程式)，至於 -1 就是命令列引數。 我們自己所撰寫的程式，也可以讓main()函式接收這些引數，並進行後續的程式處理。若要使用命令列引數，main()函式的原型如下：

```
int main(int argc, char *argv[])
```

其中argc為引數的個數，argv為引數的指標陣列。請試著寫出一個程式，將其所接收的命令列引數輸出。

12.9 string類別

不同於C語言，C++提供了一個string類別，我們可以利用此類別進行字串的操作。以string類別進行字串處理，必須先宣告產生string類別的物件，然後才可以使用。請記得在使用string類別前，必須先以#include <string>載入標頭檔。下面的程式顯示一個簡單的例子：

```
#include <iostream>
using namespace std;
#include <string>

int main()
{
  string str;

  cout << "Enter your name: ";
  getline(cin, str);

  cout << "Hi, " << str << "!" << endl;
  cout << "The third letter in your name is " << str[2] << "." << endl;

  return 0;
}
```

要注意的是，我們使用的是在string中的getline()函式，而非cin.getline()，這兩者是不同的，getline()函式的第一個引數為所要取得輸入的資料流物件(也就是透過cin來取得)，第二個引數為所要儲存的字串(string類別的物件)，string類別的物件，也可以被當成是字串陣列，直接以陣列的索引值加以存取。以下我們將詳細說明string類別的字串物件：

12.9.1 字串初始化

其實，在還沒開始學習何謂物件、何謂類別之前，我們可以暫時把string類別視為一種資料型態(當然是比較特別的資料型態)，其字串的宣告與初始化就如同一般的變數宣告一樣：

```
string strName = "Hello";
string strName ("Hello");
string strName = {"Hello"};
string strName {"Hello"};
```

12.9.2 字串操作

string類別的字串物件可以進行各式的操作，包含assignment、concatenation、appending等，請參考以下的程式碼：

```
string str1;
string str2 {"Hello"};
string str3;

str1 = str2; //assignment
str3 = str1 + str2; //assign str3 the joined strings
str1 += str2; //ass str2 to the end of str1
```

另外、string類別的物件還提供許多操作的方法，包含：

- size - 傳回字串的長度
- length - 傳回字串的長度
- clear - 清除字串內容
- empty - 查詢字串內容是否空白
- find - 查詢特定子字串出現在字串中的位置
- substr - 依條件產生子字串
- compare - 比較字串內容
- c_str - 產生傳統C語言的字串

假如要使用上述的方法，則以物件名稱加上 .方法名稱即可，請參考下面的範例：

```
string str1;
getline(cin, str1);
cout << str1.size() << endl;
```

關於string類別更完整的資料，可以至C++ Reference查詢。