2.2 變數與輸入

像 `1`, `24`, `3.14` 這樣的數，我們稱為 **字面常數**(literal constant)，它的值是固定不變的。

另外像我們在數學代數中用到的 `x`, `y`, `z` 等，則稱為 **變數** ，它的值可以改變。

在電腦程式中，變數是很重要的。它可以用來儲存輸入的資料，計算中的數值，表示某個狀態等。

使用 cin 輸入資料

下面這段程式在執行之後，會先詢問你的年齡，在你輸入年齡並按下 [Enter] 後，輸出 `"You are xx years old."`。這個 `xx` 會是你輸入的值。

```
int age;

cout << "How old are you?";
cin >> age;
cout << "You are " << age << " years old." << endl;
```

標準輸入

在程式中出現的 `cin` 是用來由 **標準輸入(standard input)** 將資料讀入電腦，我們可以把它想像成和 `cout` 相反的流向。

一般來說標準輸入指的是鍵盤的輸入，而輸入的值必須被存放到電腦裡，供後續運算和使用。

變數

我們可以把變數想像成是一塊有名字的記憶體。但是它除了有名字之外還有 **型別** ，一個型別為整數(integer)的變數，裡面只能放整數；型別為字串(string)的變數，裡面只能放字串。

宣告

在上面程式碼的第一行 `int age;` ，是在 **宣告(declare)** 這個變數。每一個變數在使用前都必須先宣告，明確指出變數的型別和名字。

一個典型的變數宣告，長這個樣子。

型別 變數名;

例如：

int age;

如果不宣告就使用變數，會發生什麼事呢？

```
#include <iostream>

using namespace std;

int main()
{
    cout << "How old are you?";
    cin >> age;
```

```
    cout << "You are " << age << " years old." << endl;

    return 0;
}
```

在上面這個程式的第 8 行，我們想使用 age 這個變數。但是往前看卻沒有看到這個變數的宣告。

這個程式在編譯時，編譯器(compiler)會發出如下的錯誤訊息。

```
main.cpp: In function ‘int main()’:
main.cpp:8:12: error: ‘age’ was not declared in this scope
    8 |     cin >> age;
      |            ^~~
```

編譯器的錯誤訊息都是英文的，但是同學們一定要學會看錯誤訊息，看懂錯誤訊息可以讓你很快抓到重點，把錯誤修正。

在這段錯誤息裡

- 第一行是告訴我們，它抓到錯誤的位置在 `main.cpp` 這個檔案裡的 `int main()` 函數裡。
- 第二行可以看到精確的位置，第 8 列(row)，第 12 行(column)。所以到第 8 列第 12 個字元的位置，你就可以看到這個錯誤訊息描述的 'age'。
- 繼續往下看 `error` 表示這是個「錯誤」，不修正它程式就無法成功編譯執行。
- 之後則是錯誤的描述 `‘age’ was not declared in this scope`，它說「這個 'age' 沒有在這個範圍內宣告」(它有在下面把位置標記給你看)

> 稍後我們還會看到編譯器給出 `warring` 也就是「警告」的狀況，這是編譯器發現某處可能有問題，但程式依然可以完成編譯並執行。

看懂錯誤訊息後，下一步就是修正它。我們在前面補上宣告即可。

```
#include <iostream>

using namespace std;

int main()
{
    int age; // <-- 我們在這裡補上宣告
    cout << "How old are you?";
    cin >> age;
    cout << "You are " << age << " years old." << endl;

    return 0;
}
```

變數的資料型別和名字

前面提到了兩個重點

- 變數在使用前要 **宣告(declare)**
- 宣告時要明定其 **「型別」** 和 **「名字」**

變數名命規則

我們可以自行命名每一個變數，但是必須遵守以下規則：

1. 變數名稱的第一個字元必須是底線 `_` 或英文字母 `A~Z, a~z`
2. 除了第一個字元外，變數名稱只能由底線 `_` 、英文字母 `A~Z, a~z` 和數字 `0~9` 組成。

- 合法的變數名稱例子，如：`Age`, `age`, `length`, `_name`, `id1246`
- 不合法的變數名稱例子，如：`369city` (開頭不能是數字), `my#name` (變數名稱不能用 #)

注意！ C++裡的變數名稱是有區分大小寫的，也就是 `Age` 和 `age`，會被視為 2 個不同的變數。

常用的基本資料型別

名稱	關鍵字	大小	範圍	備註
字元	char	1 Byte	$0 \sim 255$	• ASCII • 如：'a', '@'
整數	int	4 Byte	$-2^{31} \sim 2^{31}-1$	如：12, -65
無號整數	unsigned int	4 Byte	$0 \sim 2^{32}-1$	如：23, 656372
單精確浮點數	float	4 Byte		• IEEE 754 • 如：3.14, 5.0
雙精確浮點數	double	8 Byte		• IEEE 754 • 如：3.14, 5.0
字串	string			如："Peter"

資料型別牽涉到資料如何被儲存到記憶體裡，以及記憶體中的資料要如何被解讀。

避免誤用或使用不合適的型別

在下面這個用半徑計算圓周長的程式裡，我們宣告了一個名為 pi 的整數(int)型別變數，但是卻把一個浮點數 3.14 放入這個變數裡。

```
    int r;
    cout << "請輸入半徑 r:";
    cin >> r;

    int pi;
    pi=3.14;
    cout << "半徑為 " << r << " 的圓，其周長為 " << 2*pi*r << endl;
```

編譯時沒有出現任何錯誤訊息，執行結果如下。

```
請輸入半徑 r:1
半徑為 1 的圓，其周長為 6
```

如果把第 5 行的 pi 宣告成浮點數

```
    double pi;
```

執行結果則為

```
請輸入半徑 r:1
半徑為 1 的圓，其周長為 6.28
```

修改前因為 int 無法儲存小數的值，所以在 `pi=3.14;` 這裡發生了一些狀況。

3.14 本來是個浮點數，因為被指定(assign)到 int 型別的變數 pi，所以隱式轉型(implicit casting)為 int，喪失精確度變成了 3，過程中沒有出現任何錯誤訊息。如果我們在寫一個需要精確到小數以下 2 位的程式時，誤用 int 型別的變數來儲存數值，那麼結果可能會造成重大的損失。

如果是連喪失精確度轉型都做不到的狀況呢？

```
#include <iostream>

using namespace std;

int main()
{
    int name;

    name = "Peter";  // <--我們在這裡把字串放入整數型別的變數裡
    cout << "Hello, " << name << endl;

    return 0;
```

```
}
```

會出現錯誤訊息。

```
main.cpp: In function ‘int main()’:
main.cpp:9:12: error: invalid conversion from ‘const char*’ to ‘int’ [-fpermissive]
    9 |     name = "Peter";
      |            ^~~~~~~
      |            |
      |            const char*
```

重點在這句 `invalid conversion from ‘const char*’ to ‘int’`

不可以把 `const char*` 轉型為 `int`，因為「字串」沒辦法轉換成「整數」。

連續讀取多個值

cout 可以像這樣串接，一次輸出多組資料。

```
cout << "I am " << 16 << " years old.";
```

cin 也可以串接，讀取多個輸入值。

```
string name;
int age;

cin >> name >> age;
cout << name << " is " << age << " years old.";
```

原則上判斷輸入的斷句是在 `[空白]` 或 `[Enter]`

程式執行後，你可以輸入 `Peter` `[空格]` `16` `[Enter]`，或是輸入 `Peter` `[Enter]` `16` `[Enter]`

第一個輸入的值("Peter")會被放入變數 name 裡，第二個輸入的值(16)會被放入變數 age 裡。

```
Peter is 16 years old.
```

Revision #50
★Created 2 May 2023 23:58:22 by huihui
Updated 4 March 2024 12:08:42 by huihui