由片語學習C程式設計

台灣大學資訊工程系劉邦鋒著

台灣大學劉邦鋒老師講授

August 19, 2016

第七單元

浮點數

宣告浮點數

片語 1: float 及 double 變數的宣告方法

```
1 float f;
2 double df;
```

- 浮點數可以表示小數點。 C 程式語言 中有兩種浮點數，float 及 double。
- float 是一般浮點數，通常佔 4 個位元組。
- double 是倍準 (double precision) 浮點數，通常佔 8 個位元組， 具有較高的準確度。

範例程式 2: (sizeof-float-double.c) 計算 float double 大小

```
1 #include <stdio.h>
2 main()
3 {
4   float f;
5   double df;
6   printf("%d\n", sizeof(f));
7   printf("%d\n", sizeof(df));
8 }
```

- 使用 sizeof 得知 float 及 double 所佔的位元組數，

輸出

```
1 4
2 8
```

片語 3: 浮點數 float 的輸出及輸入

```
1 printf("%f\n", f);
2 scanf("%f", &f);
```

片語 4: 倍準浮點數 double 的輸出及輸入

```
1 printf("%f\n", df);
2 scanf("%lf", &df);
```

- 輸出時，浮點數 float 及 倍準浮點數 double 百分號 % 後面一律加 f，因為 printf 會將 float 升級到倍準浮點數 double 再印出。
- 輸入時浮點數在百分號 % 後面加 f， 倍準浮點數加 lf。

範例程式 5: (float-double-io.c) 浮點數輸出入

```
1  #include <stdio.h>
2  main()
3  {
4    float f;
5    double df;
6    scanf("%f", &f);
7    scanf("%lf", &df);
8    printf("%f\n", f);
9    printf("%f\n", df);
10 }
```

輸入

```
1 3.14  4.56
```

輸出

```
1 3.140000
2 4.560000
```

混合類別計算

- 當一個算式同時出現不同類別的變數時 C 程式語言 採用一種**升級**的觀念，就是等級低的會先被升級成等級高的，然後再計算。
- 倍準浮點數 `double` 的等級最高，再來是浮點數 `float`，最後才是整數 `int`。
  - 例如一個整數和一個倍準浮點數做運算，整數會先被升級成倍準浮點數。

範例程式 6: (upgrade.c) 整數升級為倍準浮點數

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int i;
6    int j;
7    double d;
8
9    scanf("%d", &i);
10   scanf("%d", &j);
11   scanf("%lf", &d);
12   printf("%d\n", i / j);
13   printf("%f\n", i / d);
14 }
```

輸入

```
10
4
4.0
```

輸出

```
2
2.500000
```

片語 7: 算式類別轉換

```
1 (type) expression
```

- 除了前述因為算式中出現不同類別而發生的潛在類別轉換之外，有時我們也需要直接將算式的類別做轉換，此時我們就需要使用**轉型** (cast)。
- 只要在算式前加一個用括號包住的類別，就可以將算式轉換為該類別。
- 如果對變數做類別轉換，只是在**算式中**將變數的值轉換為該類別，該變數的實際類別並不會改變。

計算平均分數

- 各科分數的總和在整數 `sum`，科目數量在整數 `count`。
- 如果使用 `sum / count`，因為整數除以整數結果是整數，所以不夠精確。
- 為了增加精確度，可使用 `(double)` 先將 `sum` 轉型為倍準浮點數，此時由於 `/` 的兩個運算元有不同類別，整數 `count` 會被升級為倍準浮點數再做運算，於是結果就有小數點，也比較精確。

範例程式 8: (average.c) 計算平均分數。

```
1  #include <stdio.h>
2  int main()
3  {
4    int count = 0;
5    int sum = 0;
6    int grade;
7    double average;
8
9    scanf("%d", &grade);
10   while (grade >= 0) {
11     sum += grade;
12     count++;
13     scanf("%d", &grade);
14   }
15   average = sum / count;
16   printf("%f\n", average);
17   average = (double) sum / count;
18   printf("%f\n", average);
19   average = (double) (sum / count);
20   printf("%f\n", average);
21 }
```

輸入

```
1 100
2 89
3 97
4 88
5 -1
```

輸出

```
1 93.000000
2 93.500000
3 93.000000
```

計算 e^x

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} \quad (1)$$

- 利用泰勒展開式計算 e^x。
- 使用一個 `for` 迴圈來計算每一次的 $\frac{x^i}{i!}$ 值。分子的部份存在 `x_power`，分母的階乘部份存在 `factorial`。
- 每次迴圈 `x_power` 就再乘一次 `x`，`factorial` 就再乘一次 `i`。
- 計算 `x_power / factorial` 時由於 `x_power` 是浮點數，`factorial` 是整數，此時會發生類別轉換。

範例程式 9: (e-x-float.c) 以 float 計算 e^x

```
1  #include <stdio.h>
2  main()
3  {
4    float x;
5    float e = 1.0;
6    int i;
7    int n = 10;
8    int factorial = 1;
9    float xpower = 1.0;
10
11   scanf("%f", &x);
12   for (i = 1; i <= n; i++) {
13     factorial *= i;
14     xpower *= x;
15     e += xpower / factorial;
16   }
17   printf("%f\n", e);
18 }
```

輸入

```
1 1.5
```

輸出

```
1 4.481686
```

範例程式 10: (e-x-double.c) 改以 double 計算 e^x

```
1  #include <stdio.h>
2  main()
3  {
4    double x;
5    double e = 1.0;
6    int i;
7    int n = 10;
8    int factorial = 1;
9    double xpower = 1.0;
10
11   scanf("%lf", &x);
12   for (i = 1; i <= n; i++) {
13     factorial *= i;
14     xpower *= x;
15     e += xpower / factorial;
16   }
17   printf("%f\n", e);
18 }
```

輸入

```
1 1.5
```

輸出

```
1 4.481687
```

學習要點

我們可以使用 `double` 提升浮點數的計算精密度。

避免溢位

- 使用變數 `factorial` 計算分母的階乘。
- 階乘增加的非常快，當輸入的 `n` 稍大時 變數 `factorial` 就會發生溢位，影響計算結果。
- 改善的方法是不要直接計算分子分母，而是使用一個變數 `term` 記住目前的第 i 項 $\frac{x^i}{i!}$，然後調整成 第 $i+1$ 項 $\frac{x^{i+1}}{(i+1)!}$ 即可。

範例程式 11: (e-x-double-term.c) 不直接計算階乘避免溢位

```
1  #include <stdio.h>
2  main()
3  {
4    double x;
5    double e = 1.0;
6    int i;
7    int n = 20;
8    double term = 1.0;
9
10   scanf("%lf", &x);
11   for (i = 1; i <= n; i++) {
12     term *= (x / i);
13     e += term;
14   }
15   printf("%f\n", e);
16 }
```

輸入

```
1 1.5
```

輸出

```
1 4.481689
```