第13章

输出格式

在前几章中，我们已经多次使用函数 `setw()` 来设定输出字段的宽度，让输出数据能够上下对齐，更容易了解。本章将仔细介绍其它设定输出格式的语法 .

13.1 使用格式操作符(manipulators)设定输出格式

设定数值的输出格式

计算结果的输出不仅要正确，也应有适当的格式。例如，要显示三组相关的数据：

```
38.7654139          38.7654
3.6854226           3.6854
378.255484          378.2555
```

很显然的，右边的编排方式比较容易判读和对照。

一般而言，数值的输出格式包括：

1. 靠左或靠右对齐
2. 要以一般的浮点数表示法还是科学表示法显示
3. 占多少字段 (亦即每笔数据占有的宽度)
4. 小数点以下的有效位数

等四个特性。

以 cout 为输出目标的数值可以使用格式操作符(manipulators) 来设定数值的格式。例如：

```
cout << "The answer is: " << setw( 5 ) << x ;
```

式中的 setw(5) 就是一个典型的用来设定宽度的格式操作符。如果没有其它设定的话，x 值就会被放置在宽度为 5 个字段的空间内，且靠右对齐。

表 13.1.1 列出常用的格式操作符。使用时，必须使用预处理指令在程序开头处加上头文件 <iomanip>:

```
#inculde <iomanip>
```

事实上，<iomanip> 包括了<iostream>，所以如果已经在程序内含入了<iomanip>，则 <iostream> 可以省略。

表 13.1.1　常用的格式操作符(需要头文件 <iomanip>)

格式操作符	效　果
setw(n)	将字段宽度设定为 n
setprecision(n)	将浮点精度设为 n 位数。 不管是一般表示法或科学表示法，实数的小数点后 n 位后的位数，以四舍五入处理，如果数字不够，则以 0 补足。对于整数则没有作用。如果没指定精度，则默认值相当于 setprecision(6)
fixed	使用一般的浮点数表示法
scientific	使用科学表示法
showpoint	此指令对于整数没有作用。对于浮点数而言，则预设 6 个有效位数，且一定会显示小数点(例如 1.0 将显示为 1.0 而不是 1)： 1．如果没有设定 ios::fixed 如果整数部分大于 6 位数，则以科学表示法表示，否则使用一般浮点数表示。例如 2645132.8 将表示为 2.64513e+06 2．如果有设定 ios::fixed 不用科学表示法表示
showpos	如果为正数，正数之前加上+ 号
left	在字段内靠左对齐
right	在字段内靠右对齐
dec	以十进制法表示
oct	以 8 进位法表示
hex	以 16 进位法表示

这些格式操作符的设定都有保持性，除非再次更动，否则以本程序内最近的设定为准。但有一个例外：宽度的设定 setw() 必须在每个送至 cout 的数值之前使用才有效。

此外，格式操作符，例如：

```
fixed
```

也可以写成

```
setiosflags(ios::fixed)
```

这里 ios::fixed 这类包括在 setiosflags() 括号内的表达式称为标记(flags)，运算符“::”称为范围运算符(the scope operator)或范围确认运算符(the scope resolution operator)。不过，使用这个语法时，在预处理指令之后的 using 声明必须写成

```
using std::setiosflags;
using std::ios;
```

而不是

```
using std::fixed;
```

其它格式操作符也是同样的做法，依此类推。这里 std 为 standard 的缩写，用来称呼 C++ 的标准名称空间 (namespace)。

下面是一些实例（见表 13.1.2）：

表 13.1.2

Manipulator	输 出
cout << setw(2) << 3 << endl ;	| 3|
cout << setw(5) << 158 << endl ;	| 158|
cout << setw(5) << 69.72 << endl ;	|69.72|
cout << setw(5) << left << 876 << endl ;	|876 |
cout << setw(6) << fixed << setprecision(2) << 133.456 << endl ;	|133.46|

setw() 所设定的宽度包括整数部分的位数加上小数点，以及小数点之后由 setprecision 所设定的有效位数，如果总位数大于宽度时，setw()的设定将不执行，所以不致因为 setw() 而导致输出错误。

表 13.1.2 是由程序 Format_1.cpp 所产生。注意，为了清楚显示数据所占用的字段，我们在数据前后都加了字符 '|' 作为标志。此外，setw()不能置于 '|' 之前，否则没有作用。

范例程序　文件 Format_1.cpp

```
// Format_1.cpp

#include <iomanip>
using std::cout;
using std::endl;
using std::setw;
using std::setiosflags;
using std::ios;
using std::setprecision;


int main()
{
  cout  << '|' << setw(2) << 3
        << '|' << endl ;
  cout  << '|' << setw(5) << 158
        << '|' << endl ;
  cout  << '|' << setw(5) << 69.72
        << '|' << endl ;
  cout  << setiosflags(ios::left)
        << '|' << setw(5) << 876
        << '|' << endl ;
  cout  << setiosflags(ios::fixed)
        << setprecision(2)
        << '|' << setw(6) << 133.456
        << '|' << endl ;
  return 0;
}
```

操作结果

```
| 3|
|  158|
|69.72|
|876  |
|133.46|
```

下列程序片断可以很整齐地显示宽度为 13，且小数点之后有效位数为 2 的值：

```
cout << setw(13)
     << fixed
     << showpoint
     << setprecision ( 2 ) << x << endl ;
```

这些格式的设定只用来改变输出的形式，对于计算机内部的数值表示和运算完全没有影响。当然，如果这些输出的数值又被其它程序读入，则其精度就会对下一轮的运算有决定性的作用。

在以下范例程序 Format_2.cpp 里，我们进一步示范了上述的格式设定指令，这些指令有别于 Format_1.cpp。也就是说，在这个语法下，预处理指令之后的 using 指令需要逐一列举格式操作符，比较麻烦，但主程序内的指令较简单。

范例程序 文件 Format_2.cpp

```
// Format_2.cpp
#include <iomanip>
using std::cout;
using std::endl;
using std::setw;
using std::hex;
using std::fixed;
```

```
using std::showpoint;
using std::scientific;
using std::setprecision;

// ---  主程序 -------------------------------
main()
{
  cout  << hex << '|' << 14 << '|' << endl;

  cout  << scientific
        << setprecision(2)
        << '|'
        << setw(5)
        << 46218.542 << '|' << endl;

  cout  << setprecision (4)
        << fixed     << showpoint
        << '|'       <<  setw(13)
        << 64.7766   << '|'  <<  endl ;
}
```

操作结果

```
|e|
|4.62e+04|
|      64.7766|
```

13.2　输出格式设定间的交互作用

setw()和 setprecision()，showpoint，fixed/scientific 等设定对于输出格式有很复杂的交互作用，我们以下面两个数值为例：

```
t1 = 253.0
t2 = 0.123456789
```

分别代入各种输出格式设定的排列组合，得到表 13.2.1：

表 13.2.1

标记	setprecision(2)		setprecision(5)	
	t1	t2	t1	t2
(不指定)	2.5e+02	1.23e-0	253	1.23457e-01
showopint	2.5e+02	1.23e-01	253.00	1.23457e-01
showpoint fixed	253.00	0.12	253.00000	0.12346
showpoint scientific	2.53e+02	1.23e-01	2.53000e+02	1.23457e-01

观察上表，可以推论得到以下的规则：

1. 如果不指定 showpoint，且原来的数值为整数 (小数点之后没有数值)，则以整数显示。
2. 如果指定 showpoint，但不指定 fixed 或 scientific，则 setprecision 所代表的意义不同。如果 precision 足够，则使用 fixed，否则使用 scientific notation。
   在以 fixed 的方式显示的情况下，有效位数为小数点前后位数的和。如果以 scientific 的方式显示，则指数为负值时有效位数只表示小数点之后的位数，指数为正值时则为小数点前后位数的和。
3. 如果同时指定了 fixed 或 scientific，则 setprecision 表示的有效位数为小数点之后的位数。

表 13.2.1 的结果可以由程序 Format_3_Short.cpp 的执行获得。这个程序同时也示范了上述格式设定的语法。(为节省篇幅，这个程序只出示了有代表性的部分，完整程序详见本书所附 CD-ROM 中的程序 Format_3.cpp。)

范例程序　文件 Format_3_Short.cpp

```
// Format_3_Short.cpp
#include <iomanip>
using std::cin;
using std::cout;
using std::endl;
using std::setw;
// --- 主程序 ---------------------------------
main()
{
  double t1=253.0;
  double t2=0.123456789;

    cout << " (1)The value of t1 is :  "
         << std::setprecision(2)
         << t1 << endl;
    cout << " (2)The value of t1 is :  "
         << std::setprecision(3)
         << std::showpoint
         << t1 << endl;
    cout << " (3)The value of t1 is :  "
         << std::setprecision(5)
         << std::showpoint
         << std::fixed
         << t1 << endl;
    cout << " (4)The value of t1 is :  "
         << std::setprecision(5)
         << std::showpoint
         << std::scientific
         << t1 << endl;
}
```

操作结果

```
(1)The value of t1 is :  2.5e+02
(2)The value of t1 is :  253.
(3)The value of t1 is :  253.00000
(4)The value of t1 is :  2.53000e+02
```

13.3 三种格式设定语法之比较

Format_3.cpp 的格式设定语法与 Format_1.cpp 和 Format_2.cpp 都不相同，我们在这里作一个具体的比较。例如，以下三组语句的效果是完全相同的：

(1) 在 Format_1.cpp 中的语法

```
cout << setiosflags(ios::fixed||ios::showpoint)
     << setprecision(2) << t1 << endl;
cout << setiosflags(ios::hex) << 26;
```

(2) 在 Format_2.cpp 中的语法

```
cout << fixed     <<  showpoint
     << setprecision(2) << t1 << endl;
cout << hex << 26;
```

(3) 在 Format_3.cpp 中的语法

```
cout << std::fixed << std::showpoint
     << std::setprecision(2) << t1 << endl;
cout << std::hex << 26;
```

而且与上述语法配合的预处理指令之后的 using 指令也不相同：

(1) 要配合：

```
#include <iomanip>
using std::cout;
using std::endl;
using std::setiosflags;
```

```
using std::ios;
using std::setprecision;
```

(2) 要配合：

```
#include <iomanip>
using std::cout;
using std::endl;
using std::fixed;
using std::showpoint;
using std::setprecision;
using std::hex;
```

(3) 要配合 (最简短)：

```
#include <iomanip>
using std::cout;
using std::endl;
```

必须分辨清楚。

但是，如果不考虑名称可能冲突的问题 (我们将在第 16 章“名称空间”中进一步讨论)，则上述三种写法的预处理指令之后的 using 指令都可以简单地写成

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

这个时候，显然第二种写法 (在 Format_2.cpp 中的语法)最简洁。

13.4　文件储存格式的设定

文件的储存与 cout 的用法相同，都是使用文件数据流 （file stream）的概念，只是把 cout 以输出文件数据流取代而已。因此，所有的输出格式设定语法可以直接套用文件的储存格式。

通常只有在储存大量数据时才需要讲究输出格式，因此，我们在以下的范例程序SaveMatrix.cpp中将以矩阵 (matrix) 的储存为例。为了使程序更具结构性，方便将相同的程序片段使用于不同的问题，我们进一步将矩阵的储存封装成函数 RecMatrix()。

此外，为了避免矩阵的列数太多，同一行数字无法一次呈现完毕，以致在显示时转到下一行影响阅读，我们在每一行的前面都加上该行的编号以兹区别。

范例程序 文件 SaveMatrix.cpp

```
// SaveMatrix.cpp
#include <iomanip>
#include <cstring>
#include <fstream>
using namespace std;
const int M = 4;
const int N = 5;
// --- 函数 RecMatrix () 的声明 ---------------------
void RecMatrix (char *, double [][N], int, int);
// --- 主程序 ---------------------------------------
int main ()
{
  double Matrix[M][N];
  char *FileName   = "Record.txt";
    for (int i = 0; i < M; i++)
      for(int j = 0; j < N ; j++)
        Matrix[i][j]= (i+j*j+0.5)/(i+j+2);
  char Ch;
  cout << "你要将矩阵存在 " << FileName
       << " 中吗? (Y/N)"   << endl;
  cin  >> Ch;
  if (Ch == 'Y' || Ch == 'y')
    RecMatrix (FileName, Matrix, M, N);
  else
    cout << "没有存盘。" << endl;
  return 0;
}
```

```
// --- 函数 RecMatrix () 的定义 ----------------------
void  RecMatrix (char *FileNameOut,
      double A[][N], int M, int N)
{
  ofstream FileOutput;
  FileOutput.open( FileNameOut, ios::out);
  if (!FileOutput)
     {cout << "文件: "     << FileNameOut
           << " 存盘失败!"  << endl; exit(1);}
  FileOutput << setprecision(4)  << right
             << showpoint        << fixed;
  for (int i = 0; i < M; i++)
    {
     FileOutput  << "第 " << i+1 << " 列: ";
     for(int j = 0; j < N ; j++)
       FileOutput  << setw(8)
                   << A[i][j] << " " ;
     FileOutput    << endl;
    }
  FileOutput.close();
  cout << "成功存于文件 "
       << FileNameOut << " 内." << endl;
}
```

操作结果　执行 SaveMatrix.exe 在显示器上会看到下列信息：

```
你要将矩阵存在 Record.txt 中吗? (Y/N)
y
成功存于文件 Record.txt 内.
```

文件 Record.txt 的内容是：

```
第 1 行:   0.2500   0.5000   1.1250   1.9000   2.7500
第 2 行:   0.5000   0.6250   1.1000   1.7500   2.5000
第 3 行:   0.6250   0.7000   1.0833   1.6429   2.3125
第 4 行:   0.7000   0.7500   1.0714   1.5625   2.1667
```

13.5 矩阵和向量间的操作

进行数据分析时，常需要显示矩阵和向量，以及取出矩阵的某一列或某一行并将它存成向量，或将向量存到矩阵的某一列或某一行。本节以完整程序示范这类常用的操作。

在程序 MatrixRow.cpp 中，函数 ShowMatrix () 和 ShowVector () 分别用来显示矩阵和向量，其中函数 ShowMatrix () 基本上使用的是 13.4 节程序 SaveMatrix.cpp 中函数 RecMatrix () 的输出格式设定。在参数列的设定上，可以参考 7.2 和 7.4 两节关于将数组作为参数的介绍。

函数 PickRow() 用来取出矩阵中选定的一整行，并将它存成向量，而函数 SetCol() 用来将向量存到矩阵中选定的某一行。

范例程序 文件 MatrixRow.cpp

```
// MatrixRow.cpp
#include <iomanip>
#include <cstring>
#include <fstream>
using namespace std;

const int M = 4;
const int N = 5;

// --- 函数的声明 ------------------------------------
void  ShowMatrix (double A[][N]);
void  ShowVector (double A[]);
void  PickRow(double A[][N], double B[], int S);
void  SetCol(double A[][N], double B[], int S);

// --- 主程序 ----------------------------------------
main ()
{
  double M2D[M][N];
  double PickV [N];
```

```
    double Data[] = { 1.3, 4.5, 8.32, 45.9, 0.4};
    int PRow = 1;
    int SRow = 2;
      for (int i = 0; i < M; i++)
        for(int j = 0; j < N ; j++)
          M2D[i][j]= (i+j*j+0.5)/(i+j+2);
      cout << "原来的矩阵是 : \n";
      ShowMatrix (M2D);
      cout << "原来的向量 Data 是 : \n";
      ShowVector (Data);
      cout << "将矩阵的第 " << PRow+1
           << " 列取出成向量 PickV: \n";
      PickRow(M2D, PickV, PRow);
      ShowVector (PickV);

      cout << "将矩阵的第 " << SRow+1
           << " 列以向量 Data 取代后,\n 矩阵成为: \n";
      SetCol(M2D, Data, SRow);
      ShowMatrix (M2D);
    return 0;
}

// --- 函数 ShowMatrix() 的定义 ----------------------
void  ShowMatrix (double A[][N])
{
    cout << setprecision(4)  << right
         << showpoint        << fixed;
    for (int i = 0; i < M; i++)
    {
      cout  << "第 "   << i+1 << " 列: ";
      for(int j = 0; j < N ; j++)
         cout << setw(8)
              << A[i][j] << " " ;
      cout << endl;
    }
    cout << endl;
}
```

```
// --- 函数 ShowVector () 的定义 ---------------------
void  ShowVector (double A[])
{
    cout << setprecision(4)  << right
              << showpoint << fixed;
    cout << "向量 :   ";
    for(int j = 0; j < N ; j++)
        cout << setw(8)
             << A[j] << " " ;
    cout << endl;
    cout << endl;
}


// --- 函数 PickRow() 的定义 -------------------------
void PickRow(double A[][N], double B[], int S)
{
  for (int i=0; i < N; i++)
      B[i]= A[S][i];
  return ;
}

// --- 函数 SetCol() 的定义 --------------------------
void SetCol(double A[][N], double B[], int S)
{
  for(int j = 0; j < N ; j++)
    A[S][j]= B[j];
  return;
}
```

操作结果

```
原来的矩阵是 :
第 1 行:   0.2500   0.5000   1.1250   1.9000   2.7500
第 2 行:   0.5000   0.6250   1.1000   1.7500   2.5000
第 3 行:   0.6250   0.7000   1.0833   1.6429   2.3125
第 4 行:   0.7000   0.7500   1.0714   1.5625   2.1667

原来的向量 Data 是 :
向量 :     1.3000   4.5000   8.3200  45.9000   0.4000

将矩阵的第 2 行取出成向量 PickV:
向量 :     0.5000   0.6250   1.1000   1.7500   2.5000

将矩阵的第 3 行以向量 Data 取代后,
矩阵成为:
第 1 行:   0.2500   0.5000   1.1250   1.9000   2.7500
第 2 行:   0.5000   0.6250   1.1000   1.7500   2.5000
第 3 行:   1.3000   4.5000   8.3200  45.9000   0.4000
第 4 行:   0.7000   0.7500   1.0714   1.5625   2.1667
```

13.6　常犯的错误

1. 使用格式设定功能时，忘了使用完整的预处理语句（也可改用“**using namespace std;**”这种较简易的语法。详细内容将在 16.5 节中介绍）：

```
#include <iomanip>
using std::cin;
using std::cout;
using std::endl;
using std::setw;
```

2. 如果一个数值的输出没有指定标记 ios::fixed，而且其绝对值的大小超过 10^5，则数值将会以科学表示法显示。