第12章

数据流与文件的存取

如果程序所处理的数据只能写在原始程序内部，或以交互的方式由键盘逐一输入，则功能将很有限。本章探讨如何从文件读取数据，以及将处理后的数据存入文件的方法。

12.1 数据流 (stream)

在讨论如何存取文件前，我们先以一个最简单的程序回顾一下标准的输入输出管道，cout 和 cin：

范例程序 文件 BasicIO.cpp

```
// BasicIO.cpp
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main ()
{
  int N;
  cout  << "请输入一个整数:"  << endl;
  cin   >> N;
  cout  << "您输入的是"      <<  N << endl;
  return 0;
}
```

操作结果

```
请输入一个整数:
4563
您输入的是 4563
```

在程序 BasicIO.cpp 里，cout 是联结程序和显示器的管道，而 cin 是联结程序和键盘的管道。它们由下述预处理指令以及 using 声明：

```
#include <iostream>
using std::cin;
```

```
using std::cout;
```

而在程序一开始执行时便自动由编译器产生。

我们把单向的数据流通管道称为数据流（stream）。以面向对象（object-oriented）的观点而言，cin 是一个输入数据流对象（input stream object），而 cout 是一个输出数据流对象（output stream object）。所谓“对象”（object），其原文有“个体”、“对象”等多种含义，如今“对象”似乎已经是个通用的名词。

我们已熟悉各种定义变量的语句，例如

```
int N;
```

它的意义代表“在内存中设定一个数据类型为 int 且名叫 N 的变量，这个变量即将在本程序中使用”。同样的，对象的数据类型称为类（class）。类是一种广义的数据类型，它可以由程序写作者自行定义，而且由类定义的“对象”不仅包括数据，而且还包括与对象进行交互所需要的各种函数。我们将在第三篇中继续探讨如何使用类去定义对象的各种语法，目前先暂时接受上述 class 和 object 的基本概念。

事实上 <iostream> 里面包括了很多数据流类（stream classes）的定义，而 cin 和 cout 分别属于 istream 和 ostream 两个类。特别的是，我们不需要定义 cin 和 cout，因为它们是最常用的对象。

驱动程序

以硬件的观点而言，要完成数据的显示和输入尚需“驱动程序”的配合。驱动程序（drivers）是操作系统（例如 Windows 和 Linux）的一部分，用来与硬件（譬如键盘和显示器）沟通，以处理硬件装置与内存之间的数据流动，其中包括传输速度的协调等细节。其关系如图 12.1.1 所示：

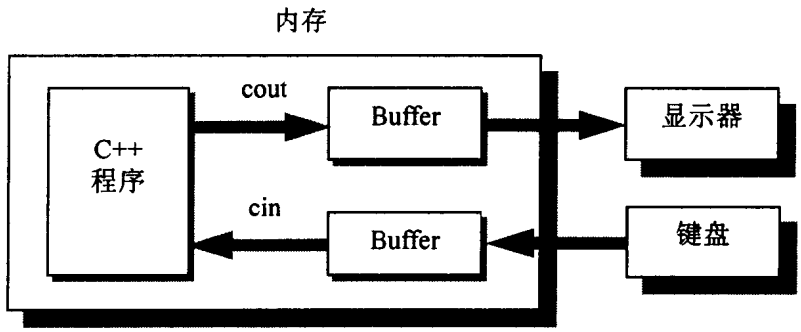

图 12.1.1 数据流（stream）与硬件之间的关系图

面向对象的基本语法

类(Class) 内部所定义的各种函数称为成员函数（member functions）或方法 (methods). 成员函数又分为 public 和 private 两种，public 成员函数是与对象进行交互与沟通的接口，适用于所有由同一个类所定义的对象. 我们将在第 18 章中仔细检讨这些名词所代表的意义，在这里我们要简略介绍的是调用成员函数的语法.

使用这些成员函数的语法有些特别. 例如，我们固然可以使用下列的语句输入一个字符：

```
char Ch;
cin >> Ch;
```

我们也可以使用下列的语法完成相同的动作：

```
char Ch;
cin.get(Ch);
```

在这个语句里，句点“.”称为成员运算符号（member operator），它的前面是对象的名称，而其后是成员函数的名称，它们的关系如图 12.1.2 所示：

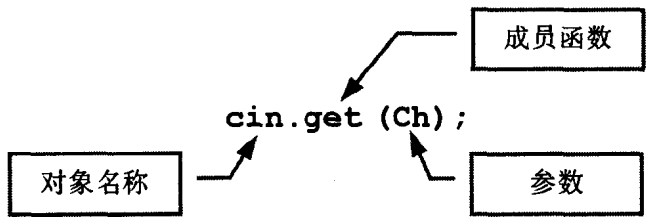

图 12.1.2　调用对象 cin 的成员函数 get()

也就是说，使用成员函数时必须在它的前面写出其所附属对象的名称.

12.2　文件的存取

当程序需要进行文件的存取时，使用的也是上节所介绍的“数据流”（stream）之语法和概念，但有以下三点主要的差异：

(1) 程序开头要加入的头文件是 <fstream>，也就是说，需要使用预处理指令

```
#include <fstream>
```

以包含必要的类声明.

(2) 必须明确地声明和定义读写文件所需的管道(亦即数据流)，而数据流的代号可以随意命名，只要符合 C++ 的命名规则即可.
要读取文件时，需要定义 ifstream 类的输入型文件数据流（input file stream）.
例如：

```
ifstream FileInput;
```

定义了名为 FileInput 的输入型文件数据流.
要写入文件时，则需定义 ofstream 类的输出型文件数据流（output file stream）. 例如：

```
ofstream FileOutput;
```

定义了名为 FileOutput 的输出型文件数据流。

在这两个例子里，FileInput 和 FileOutput 都是数据流的名称，我们可以在 C++的命名规范下自由为它们取名。

(3) 读写文件前，需要先执行将文件“开启”（open）的动作。读写完毕后，最好执行将文件“关闭”（close）的动作。因为每个系统能同时开启的文件数目有限，随手关闭不再读写的文件可以节省系统的资源，并提高执行效率。

所谓“文件”指的是在一个名称之下，内含有数据的储存单位。文件通常储存于硬盘、磁盘、光盘、磁带或 MO 等媒体上。文件和程序、数据流之间的关系如图 12.2.1 所示：

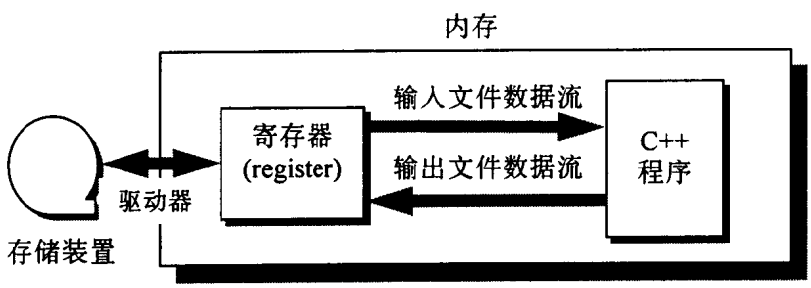

图 12.2.1 文件、数据流和程序之间的关系图

一旦文件被开启，则数据存取的语法和先前 cin，cout 的用法相同。例如：

```
FileInput  >> x;
FileOutput << x << endl
```

分别为“将数据从文件数据流 **FileInput** 读入变量 x”以及“将变量 x 中的数值送进文件数据流 **FileOutput**”。这个时候，程序内部所直接使用的是各个“已经联结到特定文件的文件数据流”，而不再提及文件的名称。

例如，要将数据流 **FileInput** 联结到文件 FileA.txt，则在文件开启时将语句写成：

```
FileInput.open("FileA.txt");
```

这里使用的是上述对象的“成员函数”调用语法，而文件名是以“字符串”的形式作为函数的参数。文件关闭时，则不用再写出文件名称，只需写成：

```
FileInput.close();
```

即可。一旦执行了关闭的语句之后，数据流 **FileInput** 和文件 FileA.txt 的联结关系即告结束。有需要的话，还可以再用同一个数据流来联结另一个文件。

在以下的完整范例程序 TaxFile.cpp 里，我们从文件 Income.txt 将收入的数值读到变量 Income 里，再调用函数 CalculateTax() 以计算税额 Tax，最后把税额的数值输出到显示器，并同时存到文件 Tax.txt 里。函数 CalculateTax()改写自 4.4 节的程序 Tax.cpp。

这个程序完整地示范了如何声明数据流，如何开启和关闭文件，以及如何在文件里存取数据的细节。

范例程序　文件 TaxFile.cpp

```
// TaxFile.cpp

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

#include <fstream>
using std::ifstream;
using std::ofstream;

// ---函数 CalculateTax() 的声明----
float CalculateTax(float);

// ---主程序------------------------
```

```
int main()
{
  // 以字符串的方式声明文件名
  char* FileNameIn  = "Income.txt";
  char* FileNameOut = "Tax.txt";
  float Income, Tax;

  ifstream FileInput;
  ofstream FileOutput;
  FileInput.open ( FileNameIn  );
  FileOutput.open( FileNameOut );

  if (!FileInput)
   {
      cout << "文件: "        << FileNameIn
           << " 开启失败!"    << endl; exit(1);
   }
  if (!FileOutput)
   {
      cout << "文件: "        << FileNameOut
           << " 存盘失败!"    << endl; exit(1);
   }

  FileInput >> Income;
  Tax = CalculateTax(Income);

  cout  << "您要缴的综合所得税是: "
        << Tax << " 元";

  FileOutput << "您要缴的综合所得税是: "
        << Tax << " 元";

  FileInput.close();    // 关闭输入文件
  FileOutput.close();   // 关闭输出文件
  return 0;
}
```

```
// ---函数 CalculateTax() 的定义-----
// 改写自 4.4 节“嵌套 if-else 语句”
// 程序 Tax.cpp 以计算税额

float CalculateTax(float GIncome)
{
  float Tax;

   if (GIncome < 0.0)
       cout << "您输入的综合所得净额没有意义!\n";
   else if (GIncome <  330000.0)
       Tax = GIncome * 0.06;
   else if (GIncome <  890000.0)
       Tax = GIncome * 0.13 - 23100;
   else if (GIncome < 1780000.0)
       Tax = GIncome * 0.21 - 94300;
   else if (GIncome < 3340000.0)
       Tax = GIncome * 0.3 - 254500;
   else
       Tax = GIncome * 0.4 - 588500;

  return Tax;
}
```

操作结果

```
您要缴的综合所得税是: 50715.2 元
```

讨 论

如果下列指令能够正确执行，则表示文件开启成功

```
FileInput.open( FileNameIn );
FileOutput.open( FileNameOut );
```

这时，**FileInput** 和 **FileOutput** 都会传回 true（整数 1）。如果开启失败（通常是因为文件名不对，文件不存在，或是因为储存装置容量不足），则传回 false（整数 0）。因此，可以用下列指令检查，以便在文件开启失败时，将无法开启的文件名称输出提醒使用者，并关闭程序：

```
if (!FileInput)
{
    cout    << "文件: "          << FileNameIn
            << " 开启失败!"      << endl; exit(1);
}
if (!FileOutput)
{
    cout    << "文件: "          << FileNameOut
            << " 存盘失败!"      << endl; exit(1);
}
```

此外，函数 exit(1) 要求操作系统立即中止此程序的执行，并回到操作系统。使用此函数需要头文件 <iomanip>。

12.3 文件的存取模式

在文件数据流和文件之间建立联结关系时，除了“输出”和“输入”，两个主要不同模式的区别外，还有下列几种不同的模式：

1. 就文件本身而言，如果文件不存在，要不要产生一个新的文件，如果文件已经存在，要不要将其取代。亦即有 create（创造）和 replace（取代）两种模式。

2. 就文件的内容而言，从数据流输入的数据要接在原来的内容之后，还是要取代原来的内容。亦即有 append (附加)和 replace (取代) 两种不同的模式。

另外一种检查打开文件有没有失败的语法

在 12.2 节的范例程序 TaxFile.cpp 里，我们通过检查数据流 FileInput 和 FileOutput，以了解打开文件（open a file）的动作有没有失败。为了能够在打开文件时，能够涵盖如果失败时应该如何让程序发出警告信息，以及如何让程序正常终止等动作，还有另外一种写法，可以将打开文件部分的语句改写如下：

```
FileInput.open (FileNameIn, ios::nocreate);
if (FileInput.fail())
{
  cout << "文件: "        << FileNameIn
       << " 开启失败!"    << endl;
  exit(1);
}
```

这里 **FileInput.open()** 的第二个参数 **ios::nocreate** 表示文件的开启模式是“只开启旧文件，如果文件不存在也不另开新文件”。而 **FileInput.fail()** 则用来检查 **FileInput.open()** 有没有执行成功，如果失败，其值为 ture（整数 1），将在发出文件开启失败的警告信息后执行“exit(1);”，以终止程序。

如果文件数据流 **FileInput** 在程序中只跟一个文件联结，则打开文件的动作可以结合数据流的声明语句，而进一步写成：

```
ifstream FileInput (FileName, ios::nocreate);
```

参数 nocreate 又称为模式指示参数 (mode indicator)。

我们将常用的模式指示参数整理列在表 12.3.1 中。

表 12.3.1 常用的模式指示参数（mode indicator）

模式指示参数	打开文件模式
ios::in	将文件开启成输入模式
ios::out	将文件开启成输出模式，如果已有同名旧文件则将其取代
ios::app	开启为 append（添加）模式
ios::ate	移到所开启文件的结尾处
ios::binary	将开启的文件内容视为二进制代码.（预设模式为“文字”）
ios::trunc	如果文件已经存在，则将其内容清除
ios::nocreate	只用来开启旧文件作为输出之用，如果文件不存在就放弃
ios::noreplace	只用来开启新文件作为输出之用，如果已有旧文件就放弃

我们在数据流的声明时，以 **ifstream** 和 **ofstream** 两种类 (classes) 来区别该数据流的方向。例如

```
ifstream FileInput;
ofstream FileOutPut;
```

另外一种写法是统一使用 **fstream** 类，数据流的方向则在其后的模式指示参数中再指定。例如：

```
fstream FileInput;
fstream FileOutPut;
FileInput.open(FileName, ios::in);
FileOutput.open(FileName, ios::out);
```

此外，还可以结合模式指示参数，以进一步决定“如果要开启以供输入的文件不存在”的处理方式，使用“或”（也就是 or）的语法：

```
FileInput.open(FileName, ios::in | ios::nocreate);
```

EOF

每一个文件结束的地方都有一个结束记号 EOF (为 end-of-file 的缩写)，它在 Windows 和 DOS 操作系统都是使用第 26 个 ASCII 字符 ‘Ctrl-Z’。为了避免与字符混淆，在某些操作系统内 EOF 的值设为 –1，EOF 为头文件 <fstream> 内定义的常数，它在文件内的地

位相当于结束符号'\0'在字符串内的地位.

以下范例程序 ConvFile.cpp 使用本节介绍的语法进行读文件与存盘的操作。首先从文件 Original.txt 逐一读进字母，再经 <cctype> 中的函数 toupper()将字母转换为大写字母后，逐一存进文件 Converted.txt 中。

范例程序　文件 ConvFile.cpp

```
// ConvFile.cpp
#include <iomanip>
using std::cin;
using std::cout;
using std::endl;
using std::setw;
#include <cctype>
using std::isupper;
using std::islower;
using std::isdigit;
using std::isspace;
using std::ispunct;
using std::toupper;

#include <fstream>
using std::fstream;
using std::ios;

char* FileNameIn  = "Original.txt";
char* FileNameOut = "Converted.txt";

// --- 函数 Sort()的声明 ----
int Sort(char X);

// ---  主程序------------------------
int main()
 {
   char C;
```

```
    fstream FileInput(FileNameIn, ios::in);
    if (!FileInput)
      {cout << "文件: "  << FileNameIn
       << "开启失败! "   << endl; exit(1);}
    fstream FileOutput(FileNameOut, ios::out);
    if (!FileOutput)
      {cout<< "文件: " << FileNameOut
       << "保存失败! " << endl; exit(1);}
    while ((C=FileInput.get()) != EOF)
    {
      switch (Sort(C))
      {
        case 1:
          FileOutput << char(toupper(C));
          break;
        case 0:  case 2:
        case 3:  case 4:
          FileOutput << C ;
          break;
        case 5:
          FileOutput << "Other" << endl;
          break;
        default:
          cout << "程序有问题! " << endl;
      }
    }
  FileOutput.close();
  FileInput.close();
  cout   << "成功存于文件 " << FileNameOut
         << " 内."          << endl;
  return 0;
}

// ---函数 Sort() 的定义---------
int Sort(char X)
{
   if (isupper(X))
```

```
        return 0;
      else if (islower(X))
        return 1;
      else if (isdigit(X))
        return 2;
      else if (isspace(X))
        return 3;
      else if (ispunct(X))
        return 4;
      else
        return 5;
  }
```

操作结果　在显示器上会看到下列信息:

```
成功存于文件 Conerted.txt 内.
文件 Original.txt 的内容是:
    I love you, certainly.
文件 Converted.txt 的内容是:
    I LOVE YOU, CERTAINLY.
```

讨 论

1. 函数 toupper()的使用可以参考 9.3 节程序 CharConv.cpp.
2. 由于函数 toupper() 的输出值是 ASCII 的数值 (整数), 因此要使用表达式

   char(toupper(C))

   将输出值转换成字符 (char). 此外, 我们也示范了 isupper(), islower(), isdigit(), isspace()和 ispunct() 几个函数的用法, 预留了进行其它对比方式的程序结构.

3．我们可以使用下列表达式来检查是否已到文件结尾的地方：

```
(C = FileInput.get())!= EOF
```

其中表达式“C = FileInput.get()”用来将字符读进 char 变量 C，而且这个表达式的值就是 C 的值，因此可以用来和 **EOF** 对比。

12.4 数据的读取与写入

除了使用“<<” 和 “>>” 两种运算符进行数据的读写外，文件数据流 fstream 还提供了一些利于读写动作的成员函数，列举于表 12.4.1 中。为了简化表 12.4 内的说明，我们假设 Ch, S1 和 N 几个变量已分别在事前声明成下列数据类型：

```
char Ch;          char* S1;          int N;
```

表 12.4.1 fstream 的成员函数

成员函数名称	用 途
get(Ch)	从输入数据流读取一个字符
getline(S1,N,'\n')	从输入数据流读取字符，并将其依序置于字符串 S1 中，直到已读取（N-1）个字符，或遇到字符'\n'
peek()	从输入数据流中检查下一个字符，但不将其从数据流中取出
put(Ch)	将一个字符置于输出数据流中
putback(Ch)	在不影响文件内容的情况下，对输入数据流放回一个字符
eof()	如果读取的动作超过 EOF (end-of-file，文件结尾)则函数值为 true
ignore(N)	跳过下面 N 个字符．如果不写参数，而写成 ignore()，表示跳过下个字符

在以下的完整范例程序 RWFiles.cpp 里，我们从文件 Record.txt 将字符串数据和数值数据分别读到字符串数组 Name 和数值数组 Score 里，再单独对 Score 进行计算，最后把字

符串和数值存到文件 Saved.txt 里。

这个程序完整地示范了输入数据流的成员函数 peek()的使用细节。“peek”为“窥视”的意思，在程序中，它用来检查即将读入的字符是否为 EOF，如果尚未到达文件的结尾，就继续依序读入个别数据，并存入相对应的变量。

范例程序　文件 RWFiles.cpp

```
// RWFiles.cpp
#include <iomanip>
using namespace std;
#include <fstream>
using std::fstream;
using std::ios;
char* FileNameIn  = "Record.txt";
char* FileNameOut = "Saved.txt";

// ---主程序------------------------
int main()
 {
   const int MaxNum  = 40;
   const int MaxSize = 20;
   char  Name [MaxNum][MaxSize];
   int   Score[MaxNum];
   fstream FileInput(FileNameIn, ios::in);
   if (!FileInput)
     {cout << "文件: "      << FileNameIn
           << " 开启失败!"  << endl; exit(1);}
   fstream FileOutput(FileNameOut, ios::out);
   if (!FileOutput)
     {cout << "文件: "      << FileNameOut
           << " 存盘失败!"  << endl; exit(1);}

   int Count=0;
   while (FileInput.peek()!= EOF && (Count < MaxNum))
    {
     FileInput >> Name[Count] >> Score[Count];
```