

C++ Programming

Chapter 14 Files and Streams

Zheng Guibin
(郑贵滨)

目录

CONTENT

- 14. 1 C++ 输入/输出类层次结构
- 14. 2 打开文件
- 14. 3 文件出错检查
- 14. 4 单字符的I/O和文件末尾检测
- 14. 5 向文件末尾添加数据
- 14. 6 从文件中读取行
- 14. 7 随机存取
- 14. 8 对象I/O
- 14. 9 二进制I/O



14.1 The C++ input/output class hierarchy

C++ 输入/输出类层次结构

◆ 从键盘读取数据，在屏幕上显示信息/结果

- 程序把来自键盘的数据存储在计算机的内存中。
- 当程序终止时，内存中的数据就会丢失，这样每次运行程序的时候都要重新输入这些数据。

◆ 文件输入/输出流

- 文件使用外部存储设备，例如用硬盘和U盘来存储数据。
- 他们都是永久性的存储设备，保存的数据在程序终止时不会丢失。



14.1 C++ 输入/输出类层次结构

- ◆ C++ 没有内置的输入输出 (I/O) 命令。在 C++ 中，I/O 命令被包含在类库中。

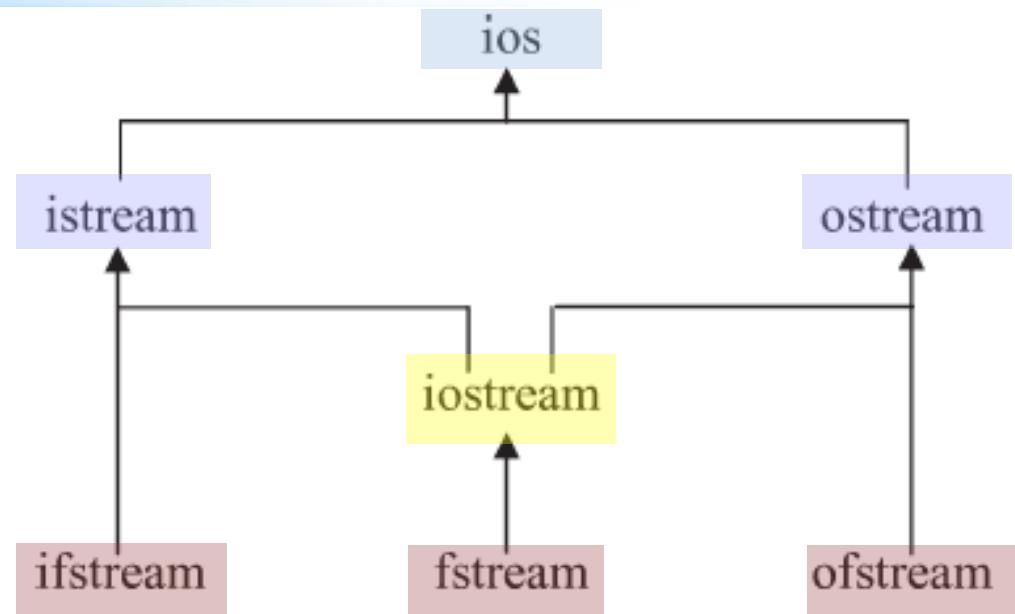


Figure 14.1 I-O Class hierarchy

基类 **ios** 中有一些数据成员是用来描述状态的，例如输入/输出流对象是否处于打开状态，以及是否已经到达文件的末尾等。



14.1 C++ 输入/输出类层次结构

- ◆ 派生类 **istream** 向基类添加了一些从流中读取数据的函数。
 - 它通过重载运算符 **>>** 来从流中读取内置数据类型 (char, int, float etc.) 的数据，从而提供基本的输入处理操作。
 - 流 **cin** 是派生类 **istream** 的对象，通常和键盘输入相关联。
- ◆ 派生类 **ostream** 中包含的一些成员函数可用于向流中输出数据。
 - 它通过重载运算符 **<<** 向流中输出内置数据类型的数据，从而提供了基本的输出处理操作。
 - 流 **cout** 是派生类 **ostream** 的对象，通常和屏幕输出相关联。

14.1 C++ 输入/输出类层次结构

- ◆ ***ifstream*** 是从**istream**派生的类，用于创建输入文件对象。
- ◆ ***ofstream*** 是从**ostream**派生的类，用于创建输出文件的对象。
- ◆ ***fstream*** 是用于创建文件对象的类，既能用于输入也能用于输出。

iostream	Includes the definition of <ul style="list-style-type: none">• <code>ios</code>, <code>istream</code>, <code>ostream</code>, and <code>iostream</code> classes• stream objects <code>cin</code>, <code>cout</code>, <code>clog</code> and <code>cerr</code>• stream manipulators <code>endl</code>, <code>ws</code>, <code>dec</code>, <code>hex</code> and <code>oct</code>.
<code>f</code> stream	<ul style="list-style-type: none">• Includes the definition of the <code>ifstream</code>, <code>ofstream</code>, and <code>fstream</code> classes.

14.2 打开文件 (Opening a file)

◆ 打开文件

- 先要创建一个适当的类的实例
ifstream in; //in 是一个输入文件对象
ofstream out; // out 是一个输出文件对象
- After在创建了一个适当的类的实例之后，必须打开该文件对象，将其和存储在硬盘或者其他存储设备的文件相关联。
in.open("in.dat"); // in 和in.dat相关联
out.open("out.dat"); // out 和out.dat相关联
- 创建文件对象的实例和打开文件可以合并为一条语句来完成，这可以通过使用类 *ifstream* 和 *ofstream* 的构造函数来实现。

*ifstream in("in.dat");
ofstream out("out.dat");*



14.2 打开文件 (Open)

```

3 #include <iostream>
4 #include <fstream>
5 #include <string>
6 using namespace std ;
7 main()
8 {
9     char c = 'A' ;
10    int i = 1 ;
11    string s = "Hello" ;
12
13    ofstream out ;           // Create a
14    out.open( "file.txt" ) ; // open file
15
16    // Write some data to the file using <
17    out << s << ' ' << c << ' ' << i << endl ;
18
19    out.close() ; // Close the output file
20
21    ifstream in( "file.txt" ) ; // Use co
22
23    // Read data from the file using >>
24    in >> s >> c >> i ;
25
26    in.close() ; // Close the input file.
27    cout << "Data read from file:" ;
28    cout << s << ' ' << c << ' ' << i << endl ;
29 }

```

- 第14行创建了一个输出文件对象的实例out。
- 第15行把out 和一个外部文件file.txt关联，并且打开这个文件用于处理操作。如果该文件不存在，那么默认情况下，将在程序所在的同一文件夹(目录)下创建这个文件。
- 第18行按照将数据输出到流对象cout中同样的方式，使用<<将数据输出到文件流中。
- 第21行使用ifstream类的构造函数来打开file.txt文件，准备进行输入操作。
- 第23行读取文件的方法和从键盘读入数据的方法类似，唯一不同的就是用in代替了cin。

14.3 文件出错检查 (File error checking)



◆ 可能的错误

- 输出设备上没有可用的空间
- 试图打开读取的文件不存在

◆ *ios* 类中包含一些成员函数可以用来检查这类错误。

- 因为 ifstream 和 ofstream 继承自 ios, 所以这两个类的对象也能使用这些成员函数。

打开一个文件时先检查错误是一个
好习惯!



14.3 文件出错检查 (File error checking)

Table 14.2 Some member functions of `ios`

Member Function	Return Value
<code>fail()</code>	Returns the Boolean value <code>true</code> if the operation failed; otherwise the Boolean value <code>false</code> is returned.
<code>good()</code>	Returns the Boolean value <code>true</code> if the operation succeeded; otherwise the Boolean value <code>false</code> is returned. This is the opposite of <code>fail()</code> .
<code>eof()</code>	Returns the Boolean value <code>true</code> if the end of the file has been reached; otherwise the Boolean value <code>false</code> is returned.

- ◆ 程序例P14B: 使用成员函数`fail()`来检查打开文件时可能发生的错误。
 - 返回0值表示成功, 返回非0值表示出现了一个错误。



14.3 文件出错检查 (File error checking)

◆ Program Example P14B

```
3 #include <iostream>
4 #include <fstream>
5 #include <string>
6 using namespace std ;
7
8 main()
9 {
10    char c = 'A' ;
11    int i = 1 ;
12    string s = "Hello" ;
13    ofstream out ;
14
15    out.open( "file.txt" ) ;
16    if ( out.fail() ) // Has the file failed to open?
17    {
18        cerr << "Failure to open file.txt for output" << endl ;
19        return 1 ;
20    }
```

测试：打开一个不存在的盘符上的文件



14.3 文件出错检查 (File error checking)

◆ 程序例P14B

```
21 // Write some data to the file using <<
22 out << s << ' ' << c << ' ' << i << endl ;
23 out.close() ;
24
25 ifstream in( "file.txt" ) ;
26 if ( in.fail() )
27 {
28     cerr << "Failure to open file.txt for input" << endl ;
29     return 1 ;
30 }
31 // Read data from the file using >>
32 in >> s >> c >> i ;
33 in.close() ; // Close the input file.
34 cout << "Data read from file:" ;
35 cout << s << ' ' << c << ' ' << i << endl ;
36 return 0 ;
37 }
```



14.4 单字符的I/O 和文件末尾的检测

- ◆ 从文件中读取数据时，必须检测是否达到了文件的末尾，以便结束对文件的处理操作。
 - 可以使用 *ios* 类的成员函数 *eof()* 来检测文件的末尾，*ifstream* 类也从 *ios* 类继承了这个函数。
- ◆ 程序例P14C
 - 演示了在复制文本文件时函数 *eof()* 的作用。
 - 类 *istream* 的成员函数 *get()* 用于从输入文件中读取一个字符。
 - 类 *ofstream* 的成员函数 *put()* 则负责将字符写入到输出文件中。



14.4 单字符的I/O 和文件末尾的检测

```
1 // Program Example P14C
2 // Program to demonstrate member functions eof, get and put.
3 #include <iostream>
4 #include <fstream>
5 using namespace std ;
6
7 main()
8 {
9     char c ;
10    int count ;
11
12    ifstream in( "file.txt" ) ; // Open the input file.
13    if ( in.fail() )
14    {
15        cerr << "Open failure on file.txt" << endl ;
16        return 1 ;
17    }
18
19    ofstream out( "file.bak" ) ; // Open output file.
```



14.4 单字符的I/O 和文件末尾的检测

```

20     if ( out.fail() )
21     {
22         cerr << "Open failure on file.bak" << endl ;
23         return 1 ;
24     }
25
26     in.get( c ) ; // Read the first character in the file.
27     count = 1 ;
28     while( !in.eof() ) // Loop while not end of file.
29     {
30         out.put( c ) ; // Write the character.
31         in.get( c ) ; // Read the next character.
32         count++ ;
33     }
34
35     in.close() ;
36     out.close() ;
37
38     cout << "Copy completed. "
39             << count - 1 << " characters copied." << endl ;
40     return 0 ;
41 }
```

- 第 26 行程序读取输入文件的第一个字符
- 第 28 到 33 行 继续写入一个字符中，然后读取下一个来自输入文件，直到检测到到达了输入文件的末尾



14.4 单字符的I/O 和文件末尾的检测

```

20     if ( out.fail() )
21     {
22         cerr << "Open failure on file.bak" << endl ;
23         return 1 ;
24     }
25
26     in.get( c ) ; // Read the first character in the file.
27     count = 1 ;
28     while ( in.get( c ) )      Loop while not end of file. •28行的简化
29     {
30         out.put( c ) ; // Write the character.
31         read the next character.
32         count++ ;
33     }
34
35     in.close() ;
36     out.close() ;
37
38     cout << "Copy completed. "
39             << count - 1 << " characters copied." << endl ;
40     return 0 ;
41 }
```



14.4 单字符的I/O 和文件末尾的检测

◆ 程序例P14E

- 这种检测文件末尾的方法同样可以应用于返回流引用的成员函数。
- 使用流提取运算符 `>>`
- 这个程序假设每个单词之间用一个或者多个空白字符来分隔。空白字符会被`>>`忽略掉。

```
4 #include <iostream>
5 #include <iomanip>
6 #include <string>
7 using namespace std ;
8
9 main()
10 {
11     string word ;
12     int word_count = 0 ;
```



14.4 单字符的I/O 和文件末尾的检测

```

14     ifstream in( "words.txt" ) ;
15     if ( in.fail() )
16     {
17         cerr << "Open failure on words.txt" << endl ;
18         return 1 ;
19     }
20
21     while ( in >> word ) // Read a word while not end of file.
22     {
23         cout << word << endl ;
24         word_count++ ;
25     }
26
27     in.close() ;
28     cout << "Number of words read = " << word_count << endl
29     return 0 ;
30 }
```

•第 21 行到第25 行的while循环将文件内容连续显示在屏幕上，直到文件尾为止。

•到达文件尾以后，while循环关闭，并显示文件中的单词。



14.5 向文件末尾添加数据

- ◆ 为了向一个文件的末尾添加数据，在打开文件的时候必须指定文件的打开模式。
- ◆ 文件的打开模式在类**ios** 中定义。



14.5 向文件末尾添加数据

Table 14.3 File modes

Mode	Meaning
<code>ios::app</code>	Opens a file for appending - additional data is written at the end of the file.
<code>ios::ate</code>	Start <u>at the end</u> of the file when a file is opened.
<code>ios::binary</code>	Opens a file in binary mode (default is text mode).
<code>ios::in</code>	Opens a file for input (default for <code>ifstream</code> objects).
<code>ios::out</code>	Opens a file for output (default for <code>ofstream</code> objects).
<code>ios::trunc</code>	Truncates (deletes) the file contents if the file already exists.
<code>ios::nocreate</code>	Do not create a new file. Open fails if the file does not already exist. For output files only.
<code>ios::noreplace</code>	Do not replace an existing file. Open fails if the file already exists. For output files only.

14.5 向文件末尾添加数据

- ◆ 可以通过使用按位或运算符 (|) 联合使用多种打开模式。
 - 打开一个文件既可用作输入，也可用作输出

```
fstream in_out( "file.txt", ios::in | ios::out );
```
 - 默认打开模式：
 - *ifstream* 类对象的默认打开模式是 **ios:: in**
 - *ofstream* 类对象的默认打开模式是 **ios::out**
- ◆ 程序例P14F：演示了把“**This line is added to the end of the file**”这一行添加到文件file.txt中。



14.5 向文件末尾添加数据

◆ 程序例P14F:演示

```
3 #include <iostream>
4 #include <fstream>
5 using namespace std ;
6
7 main()
8 {
9     ofstream out( "file.txt", ios::app ) ;
10    if ( out.fail() )
11    {
12        cerr << "Open failure on file.txt" << endl ;
13        return 1 ;
14    }
15    out << "This line is added to the end of the file" << endl
16    out.close() ;
17    return 0 ;
18 }
```



14.6 从文件中读取行

◆ *getline()*

- 使用函数`getline()`可以从文件读入一整行数据到C/C++字符串中。
- ◆ 程序 P14G使用C++ 字符串存储一行数据。

```
1 // Program Example P14G
2 // Program to demonstrate reading a file line by line
3 // using a C++ style string.
4 #include <iostream>
5 #include <fstream>
6 using namespace std ;
7
8 main()
9 {
10     string cpp_string ; // A C++ style string.
11     int line_number = 0 ;
```



14.6 从文件中读取行

◆ 程序例P14G...续

```
13  ifstream in( "p14g.cpp" ) ; // Open this program file.  
14  if ( in.fail() )  
15  {  
16      cerr << "Open failure on p14g.cpp" << endl ;  
17      return 1 ;  
18  }  
19  // Read each line into a C++ string,  
20  // and display the line number and the string.  
21  while ( getline( in, cpp_string ) )  
22  {  
23      cout << ++line_number << ":" << cpp_string << endl ;  
24  }  
25  in.close() ;  
26  return 0 ;  
27 }
```



14.6 从文件中读取行

◆ 程序例P14H: 使用C字符串存储。

```
3 // using a C-style string.  
4 #include <iostream>  
5 #include <fstream>  
6 using namespace std ;  
7  
8 main()  
9 {  
10    char c_string[81] ; // A C-style string, maximum size is 80.  
11    int line_number = 0 ;  
12  
13    ifstream in( "p14h.cpp" ) ;  
14    if ( in.fail() )  
15    {  
16        cerr << "Open failure on p14h.cpp" << endl ;  
17        return 1 ;  
18    }
```



14.6 从文件中读取行

◆ 程序例P14H...续

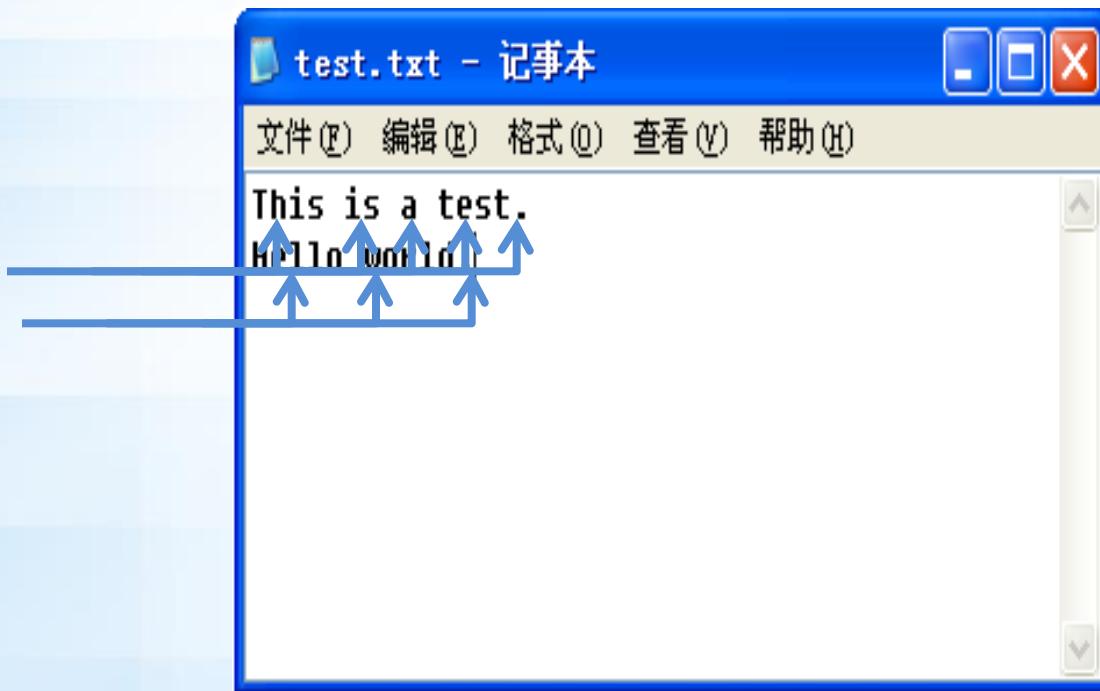
```
19 // Read each line into a C-string
20 // and display the line number and the string.
21 while( in.getline( c_string, 81 ) )
22 {
23     cout << ++line_number << ':' << c_string << endl ;
24 }
25 in.close() ;
26 return 0 ;
27 }
```



14.7 随机存取 (Random access)

◆ 顺序文件处理

- 在顺序文件处理过程中, 对数据项的读/写是一个接一个进行的。



14.7 随机存取 (Random access)

◆ 随机存取或直接存取

- 允许在文件中随意定位, 在文件的任何位置读写数据。
- 文件中的位置信息保存在文件位置标记(FPM)中。
- *seekg()* (meaning ‘seek get’) 用来为输入文件设定FPM, 指示下一个要读取的数据的位置。
 - *istream* 类的成员函数
 - 需要两个实参

Table 14.4 Offset values

<code>ios::beg (=0)</code>	The offset is from the beginning of the stream.
<code>ios::cur (=1)</code>	The offset is from the current position in the stream.
<code>ios::end (=2)</code>	The offset is from the end of the stream.

14.7 随机存取 (Random access)

- ◆ 例：打开一个包含字符A到J的文件

```
istream in( "letters.dat" ) ;
```

```
in.seekg( 4, ios::beg ) ;
```

or in.seekg(4) ;

```
in.seekg( 1, ios::cur ) ;
```

```
in.seekg( -3, ios::cur ) ;
```

```
in.seekg( 0, ios::beg ) ;
```

```
in.seekg( 0, ios::end ) ;
```



FPM = 10



14.7 随机存取 (Random access)

◆ **seekp()**

- 在输出文件中用于设置FPM的函数是seekp()。

◆ **tellg()**

- 返回输入流中FPM的当前值

◆ **tellp()**

- 返回输出流中FPM的当前值



14.7 随机存取 (Random access)

程序例14.1

```
1 // Program Example P14.1
2 // Simple demonstration of random file access.
3 #include <iostream>
4 #include <fstream>
5 using namespace std ;
6
7 main()
8 {
9     char c ;
10    int file_pos, last_pos, offset ;
11
12    ifstream in( "letters.txt" ) ;
13    if ( in.fail() )
14    {
15        cerr << "Open failure on letters.txt" << endl ;
16        return 1 ;
17 }
```



14.7 随机存取 (Random access)

◆ 程序例P14I

```
19 // Find the position of the last character in the file.  
20 in.seekg( 0, ios::end ) ;  
21 last_pos = in.tellg() ;  
22  
23 // Loop until user enters a 0.  
24 do  
25 {  
26     cout << "Enter the file position (0 to end) " ;  
27     cin >> file_pos ;  
28     // Is this a valid position?  
29     if ( file_pos > last_pos || file_pos < 0 )  
30         cout << "Invalid position. Enter a value between 1 &  
31             << last_pos << endl ;
```



14.7 随机存取 (Random access)

```

32     if ( file_pos > 0 && file_pos <= last_pos )
33     {
34         // The offset is 1 less than the position.
35         offset = file_pos - 1 ;
36         // Go directly to the character's offset.
37         in.seekg( offset, ios::beg ) ;
38         // Read the character and display.
39         in.get( c ) ;
40         cout << "Character at position " << file_pos
41             << ", offset " << offset
42             << " is "<< c << " ASCII "
43             << static_cast<int>( c ) << endl ;
44     }
45 }
46 while ( file_pos != 0 ) ;
47 in.close() ;
48 return 0 ;
49 }
```

注意：第43行使用强制类型转换将变量c的值转换成和它等价的整数即ASCII码值。

```

Enter the file position (0 to end) 3
Character at position 3, offset 2 is C ASCII 67
Enter the file position (0 to end) 0
```

14.8 对象I/O (Object I/O)

- ◆ 在程序P10G中通过重载运算符 >> 和<< 使类 time24 具有了键盘输入和屏幕输出的功能。
- ◆ 如果对time24类不做修改，那么就需要使用磁盘文件来代替键盘和屏幕进行输入和输出。
- ◆ 程序例P14J
 - 打开一个文件，并且对来自这个文件的time24 对象进行读写。



14.8 对象I/O (Object I/O)

```

1 // Program Example P14J
2 // Demonstration of object I/O.
3 #include <iostream>
4 #include <fstream>
5 #include "time24.h"
6 #include "time24.cpp"
7 using namespace std ;
8
9 main()
10 {
11     time24 t1( 1, 2, 3 ) ;
12     time24 t2( 10, 10, 10 ) ;
13
14     fstream in_out( "times.dat", ios::in|ios::out ) ;
15     if ( in_out.fail() )
16     {
17         cerr << "error on opening times.dat" << endl ;
18         return 1 ;
19     }

```

程序P10H

- 第5行中的文件time24.h 包含time24 类声明。
- 第6行中的文件time24.cpp包含time24 类的成员函数。
- 第14行打开文件 times.dat 作为fstream 对象用来输入输出



14.8 对象I/O (Object I/O)

◆ 程序例P14J...续

```

21 // Write objects to the file.
22 in_out << t1 << t2 ;
23
24 // Rewind the file to the start.
25 in_out.seekg( 0 ) ;
26
27 // Read objects back and display on the screen.
28 in_out >> t1 >> t2 ;
29 cout << t1 << t2 ;
30 in_out.close() ;
31 return 0 ;
32 }
```

01:02:03
10:10:10



- 第22行将 time24 的对象 t1 和t2写入到文件中。
- 第 28行 读取来自文件的对象，第29行显示这些对象。

14.9 二进制I/O (Binary I/O)

◆ C++文件的类型:

- 文本文件 (或者ASCII文件)
 - 数值型数据以字符的ASCII码形式存储
- 二进制文件
 - 数值型数据以二进制形式存储
- 例 short int n = 123;
 - 短整型变量n在内存中占两个字节
 - 把变量n的值存储在文本文件中需要3个字节的内存。

Character :

ASCII value in decimal:

ASCII value in binary:

'1'	'2'	'3'
49	50	51
00110001	00110010	00110011

14.9 二进制I/O (Binary I/O)

◆ ASCII 文件

- 每一位数字都要占用一个字节的存储空间。

◆ 二进制文件

- 一个数据的每一位数字并不占用单独的存储单元，而是把整个数据作为一个二进制数来存储。

- 例：

- n=123
- n=1234: 相同大小的存储空间

0000100	10010010
---------	----------

对于 ASCII 码文件来说，则需要再增加一个字节来存储额外的数字4。



14.9.1 对象连续写入二进制文件

◆ **write()**

- 带有两个实参的*istream* 类的成员函数
- 用于以二进制格式写一个对象
- 第一个实参是存储对象的内存区域的地址，第二个实参是这个内存区域的大小。

◆ 对象必须存储在一块连续的内存区域中

- 对象不能有指向动态分配的内存的指针数据成员。
- 类不能有任何*string* 数据成员，因为*string* 类使用一个指针数据成员。



14.9 二进制I/O (Binary I/O)

◆ 程序例P14K

```
1 // Program Example P14K
2 // Demonstration of writing to a binary file.
3 #include <iostream>
4 #include "stock.h"
5 #include "stock.cpp"
6 using namespace std ;
7
8 main()
9 {
10    // Initialise an array of stock objects.
11    class stock stationery[ 10 ] =
12    { stock( 1, "Pencil", 68 ),
13      stock( 2, "Pen", 30 ),
14      stock( 3, "Highlighter", 90 ),
15      stock( 4, "Eraser", 24 ),
16      stock( 5, "Pencil Sharpener", 5 ),
```



14.9 二进制I/O (Binary I/O)

◆ 程序例P14K

```

17     stock( 6, "Pocket Folder", 50 ),
18     stock( 7, "Paper Tie", 300 ),
19     stock( 8, "Glue Stick", 30 ),
20     stock( 9, "Box File", 35 ),
21     stock( 10, "Note Book", 97 )
22
23     int object_size = sizeof( stock ) ;
24
25 // Open the stock file for binary output.
26 ofstream stock_file( "stock.dat", ios::binary ) ;
27 if ( stock_file.fail() )
28 {
29     cerr << "Open failure on stock.dat" << endl ;
30     return 1 ;
31 }
```

第23行计算了函数 ***write()*** 的第二个实参,即内存块的字节数。



14.9 二进制I/O (Binary I/O)

•第41行的函数**write()**把**stock**对象看做是一整块儿的内存字节或字符，无需转换为**ASCII**码就可以直接从内存复制到磁盘文件中。

```

32
33 // Write each object to the output file.
34 for ( int i = 0 ; i < 10 ; i++ )
35 {
36     cout << "Writing Stock Code "
37         << stationery[ i ].get_code() << ' '
38         << stationery[ i ].get_description()
39         << " quantity = " << stationery[ i ].get_quantity()
40         << endl ;
41     stock_file.write( reinterpret_cast<char *>( &stationery[ i ] ),
42                         object_size ) ;
43 }
44 stock_file.close() ;
45 return 0 ;
46 }
```



14.9 二进制I/O (Binary I/O)

```
Writing Stock Code 1 Pencil quantity = 68
Writing Stock Code 2 Pen quantity = 30
Writing Stock Code 3 Highlighter quantity = 90
Writing Stock Code 4 Eraser quantity = 24
Writing Stock Code 5 Pencil Sharpener quantity = 5
Writing Stock Code 6 Pocket Folder quantity = 50
Writing Stock Code 7 Paper Tie quantity = 300
Writing Stock Code 8 Glue Stick quantity = 30
Writing Stock Code 9 Box File quantity = 35
Writing Stock Code 10 Note Book quantity = 97
```



14.9 二进制I/O (Binary I/O)

- ◆ 函数write()的第一个实参是一个指向字符串的指针。
 - 对象被看做是字符块儿，所以第i个stock对象的地址 $\&stationery[i]$ 被强制转换成指向字符串的指针。

`reinterpret_cast<char*>(&stationery[i])`

- *reinterpret_cast* 允许从任意指针类型转换到其他任意指针类型。
- ◆ 函数write()的第二个实参是内存块儿的字节数。



14.9 二进制I/O (Binary I/O)

◆ stock.h 文件

```
1 #if !defined STOCK_H
2 #define STOCK_H
3
4 // Header for a simple stock class.
5 #include <fstream>
6 using namespace std ;
7
8 class stock
9 {
10 public:
11     stock() ;
12     // Purpose: Default constructor.
13
14     stock( int code, char desc[ ], int quantity ) ;
15     // Purpose    : Constructor
16     // Parameters: stock code, description and stock quantity.
17     //                  : Note that the description is a C-string.
```



14.9 二进制I/O (Binary I/O)

◆ stock.h 文件

```
19 void set_quantity( int quantity ) ;
20 // Purpose : Assign a value to the stock quantity.
21 // Parameter: The quantity to be assigned.
22
23 const int get_code () const ;
24 // Purpose: Inspector function to return the stock code.
25
26 const char* get_description() const ;
27 // Purpose: Inspector function to return the stock description.
28 //           : Note that description is a C-string.
29
30 const int get_quantity () const ;
31 // Purpose: Inspector function to return the stock quantity.
32
33 void binary_write( ofstream& os );
34 // Purpose : Write stock object to a binary file.
35 // Parameter: os - File output stream.
```



14.9 二进制I/O (Binary I/O)

◆ stock.h 文件

```
37     bool binary_read( ifstream& is ) ;
38     // Purpose : Read stock object from a binary file.
39     // Parameter: is - File input stream.
40     // Returns : true if end of file, otherwise false is returned.
41
42 private:
43     int stock_code ;
44     char description[21] ;
45     int quantity_in_stock ;
46 } ;
47
48 #endif
```



14.9 二进制I/O (Binary I/O)

◆ stock.cpp 文件

```
1 #if !defined STOCK_CPP
2 #define STOCK_CPP
3
4 // Member function definitions for stock class.
5 #include "stock.h"
6 #include <cstring>
7 stock::stock()
8 {
9     stock_code = 0 ;
10    quantity_in_stock = 0 ;
11    description[0] = '\0' ;
12 }
13
14 stock::stock( int code, char desc[ ], int quantity )
15 {
16     stock_code = code ;
17     strcpy( description, desc ) ;
18     quantity_in_stock = quantity ;
19 }
```



14.9 二进制I/O (Binary I/O)

◆ stock.cpp 文件

```
21 void stock::set_quantity ( int quantity )
22 {
23     quantity_in_stock = quantity ;
24 }
25
26 const int stock::get_code () const
27 {
28     return stock_code ;
29 }
30
31 const char* stock::get_description() const
32 {
33     return description ;
34 }
35
36 const int stock::get_quantity() const
37 {
38     return quantity_in_stock ;
39 }
```



14.9 二进制I/O (Binary I/O)

◆ stock.cpp 文件

```
41 void stock::binary_write( fstream& os )
42 {
43     os.write( reinterpret_cast<char *>( this ), sizeof( *this ) ) ;
44 }
45
46 bool stock::binary_read( fstream& is )
47 {
48     is.read( reinterpret_cast<char *>( this ), sizeof( *this ) );
49     return !is.eof() ;
50 }
51
52 #endif
```



14.9 二进制I/O (Binary I/O)

◆ 14.9.2 对象从二进制文件中读取

- **read()**

- *istream* 类的成员函数
- 从二进制文件中将对象的数据读到内存中
- 函数*read()*的实参和函数*write()*相同
- 程序例P14L: 演示了从二进制文件中读取数据

```
1 // Program Example P14L
2 // Demonstration of serial reading of a binary file.
3 #include <iostream>
4 #include <iomanip>
5 #include "stock.h"
6 #include "stock.cpp"
7 using namespace std ;
```



14.9 二进制I/O (Binary I/O)

◆ 程序例P14L

```
9 main()
10 {
11     stock stock_record ;
12
13     // Open the stock file for binary input.
14     ifstream stock_file( "stock.dat", ios::binary ) ;
15     if ( stock_file.fail() )
16     {
17         cerr << "Open failure on stock.dat" << endl ;
18         return 1 ;
19     }
20
21     int object_size = sizeof( stock ) ;
22
23     cout << "Code Description Quantity" << endl ;
24     cout << left ; // Left justify data.
```



14.9 二进制I/O (Binary I/O)

◆ 程序例P14L

```

25 // Read and display stock records until eof.
26 while( stock_file.read
27     ( reinterpret_cast<char *>( &stock_record ),
28      object_size ) )
29 {
30     cout << setw(5) << stock_record.get_code()
31     << setw(20) << stock_record.get_description()
32     << stock_record.get_quantity() << endl ;
33 }
34 stock_file.close() ;
35 return 0 ;
36 }
```

Code	Description	Quantity
1	Pencil	68
2	Pen	30
3	Highlighter	90
4	Eraser	24
5	Pencil Sharpener	5
6	Pocket Folder	50
7	Paper Tie	300
8	Glue Stick	30
9	Box File	35
10	Note Book	97



14.9 二进制I/O (Binary I/O)

◆ 14.9.3 二进制 I/O 作为类的成员函数

- 如果将读写函数的细节隐藏在类的成员函数中，那么使用二进制I/O就比较容易了。

◆ stock.cpp

```
void stock::binary_write( fstream& os ) const
{
    os.write( reinterpret_cast<char *>( this ), sizeof( *this ) );
}
```

• 内置指针**this** 是一个指向调用成员函数的对象的指针即**stationery[i]**

sizeof(*this) 等同于

sizeof(stationery[i])

- Example: P14K

```
41     stock_file.write( reinterpret_cast<char *>( &stationery[i] ),
42                         object_size ) ;
```



```
stationery[i].binary_write( stock_file ) ;
```



14.9 二进制I/O (Binary I/O)

◆ stock.cpp

```
bool stock::binary_read( fstream& is )
{
    is.read( reinterpret_cast<char *>( this ), sizeof( *this ) );
    return !is.eof();
}
```

除从输入流中读取对象外，函数
binary_read()将在到达文件末尾时返回
false，否则返回**true**。

- Example: P14L

```
26     while( stock_file.read
27             ( reinterpret_cast<char *>( &stock_record ),
28             object_size ) )
```



```
while ( stock_record.binary_read( stock_file ) )
```



14.9 二进制I/O (Binary I/O)

◆ Example: P14L

```
14    ifstream stock_file( "stock.dat", ios::binary ) ;
```



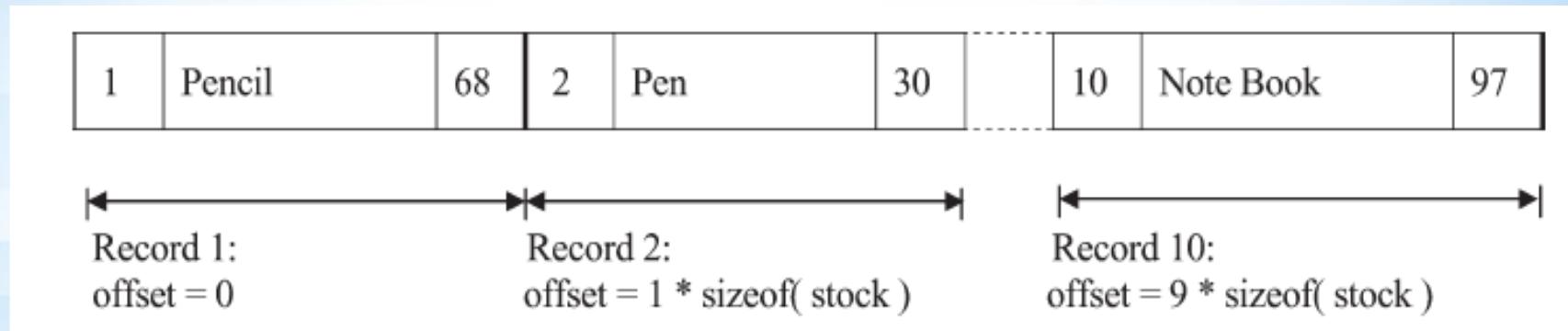
```
fstream stock_file( "stock.dat", ios::binary | ios::in ) ;
```

将`stock_file`按照函数`stock::binary_read()`的要求变成
fstream 对象。



14.9.4 二进制文件随机存取

- 由程序 P14K 创建的文件结构如下图所示：



- 相对文件

- 存货的代码和存货记录在文件中的位置有关,
 - 存货代码1为第1条记录,
 - 存货代码2为第2条记录,
 - 存货代码3为第3条记录,
 -

一个存货记录在文件中的偏移量可以用存货代码值减去1，然后乘以存货记录的字节大小来计算。

14.9.4二进制文件随机存取

◆ 程序例 P14M: 演示了二进制文件的随机存取读入。

```
1 // Program Example P14M
2 // Program to demonstrate random access reading of a binary file.
3 #include <iostream>
4 #include <iomanip>
5 #include "stock.h"
6 #include "stock.cpp"
7 using namespace std ;
8
9 // Non-class standalone functions.
10 int input_stock_code() ;
11 // Purpose: Function to input a stock code from the keyboard.
12 // Returns: The stock code entered.
13
14 void update_stock_record( int stock_code, fstream& stock_file )
15 // Purpose    : Update the stock quantity.
16 // Parameters: The stock code and the output stream.
```



14.9.4 二进制文件随机存取

◆ 程序例 P14M

```
18 main()
19 {
20     int stock_code, max_stock_code ;
21     stock stock_record ;
22
23     // Open the stock file for binary input and output.
24     fstream stock_file( "stock.dat",
25                         ios::binary | ios::in | ios::out) ;
26     if ( stock_file.fail() )
27     {
28         cerr << "Open failure on stock.dat" << endl ;
29         return 1 ;
30     }
31
32     // Find the number of records in the file.
33     stock_file.seekg( 0, ios::end ) ;
34     max_stock_code = stock_file.tellg() / sizeof( stock ) ;
```



14.9.4二进制文件随机存取

当文件中的数据更新完毕之后，显示整个文件的内容。首先在第49行将FPM移到文件的起始位置，然后通过第50行至55行的程序显示每一个存货记录。

程序例P14M

```

36 // Input a stock code and update the stock quantity, code 0 ends.
37 do
38 {
39     stock_code = input_stock_code() ;
40     if ( stock_code > 0 && stock_code <= max_stock_code)
41         update_stock_record( stock_code, stock_file ) ;
42 }
43 while ( stock_code != 0 ) ;
44
45 // Display the updated file.
46 cout << "Code Description          Quantity" << endl ;
47 cout << left ; // Left justify data.
48 // Read and display stock records until end of the file.
49 stock_file.seekg( 0 ) ;
50 while ( stock_record.binary_read( stock_file) )
51 {
52     cout << setw(5) << stock_record.get_code()
53             << setw(20)<< stock_record.get_description()
54             << stock_record.get_quantity() << endl ;
55 }
```



14.9.4二进制文件随机存取

◆ 程序例P14M

```
57     stock_file.close() ;
58
59 }
60
61 int input_stock_code()
62 {
63     int code ;
64     cout << "Enter a stock code (0 to end ): " ;
65     cin >> code ;
66     return code ;
67 }
68
69 void update_stock_record( int stock_code, fstream& stock_file )
70 {
71     int quantity ;
72     int offset ;
73     stock stock_record ;
```



14.9.4二进制文件随机存取

◆ 程序例P14M

```

75 // Place the file position marker just before the record.
76 offset = ( stock_code -1 ) * sizeof( stock_record ) ;
77 stock_file.seekg( offset, ios::beg ) ;
78
79 // Read the record.
80 stock_record.binary_read( stock_file ) ;
81
82 // Get the new quantity.
83 cout << "Enter Stock Quantity for Stock Code: "
84     << stock_code << ", "
85     << stock_record.get_description() << ":" ;
86 cin >> quantity ;
87
88 // Update the record.
89 stock_record.set_quantity( quantity ) ;
90 stock_file.seekp( offset, ios::beg ) ;
91 stock_record.binary_write( stock_file ) ;
92 }
```

与输入的存货代码相对应的存货记录可以通过计算该记录在文件中的偏移量来获取，即将文件位置标记（FPM）移到该记录的起始位置，然后读取该记录。



编程陷阱 (Programming Pitfalls)

◆ 1. 文件名通常都会包含\

- 例如, *c:\newfile.dat*
- 为了打开一个这样的文件, 必须使用一个额外的\:

```
istream in("c:\\newfile.dat");
```

- 如果没有这个额外的\, 那么 **n** 将会被解释为转义序列中的换行。Linux 和 UNIX 在文件说明中使用正斜杠 /, 所以不会出现这个问题。

◆ 2. istream类的成员函数 *read()*

- 第一个实参是一个指向内存单元的字符指针。
- 内存单元必须足够大以便能够存储所有被读入的数据。

编程陷阱 (Programming Pitfalls)

◆ 3. 默认打开模式

- *ifstream* 类对象的默认打开模式是 **is::in**
- *ofstream* 类对象的默认打开模式是 **in::out**.
- *fstream* 类对象没有默认的打开模式，所以打开一个fstream 对象时，必须指定文件的打开方式。

```
fstream in_out("file.txt", ios::in|ios::out);
```

◆ 4. 使用按位或者运算符 |, 可以将多种文件打开模式联合使用，但不能使用逻辑或运算符 ||。



Q & A



Two hands, one in a blue shirt and one in a dark blue shirt, are shown in a handshake. The background is white.

Thank You!

