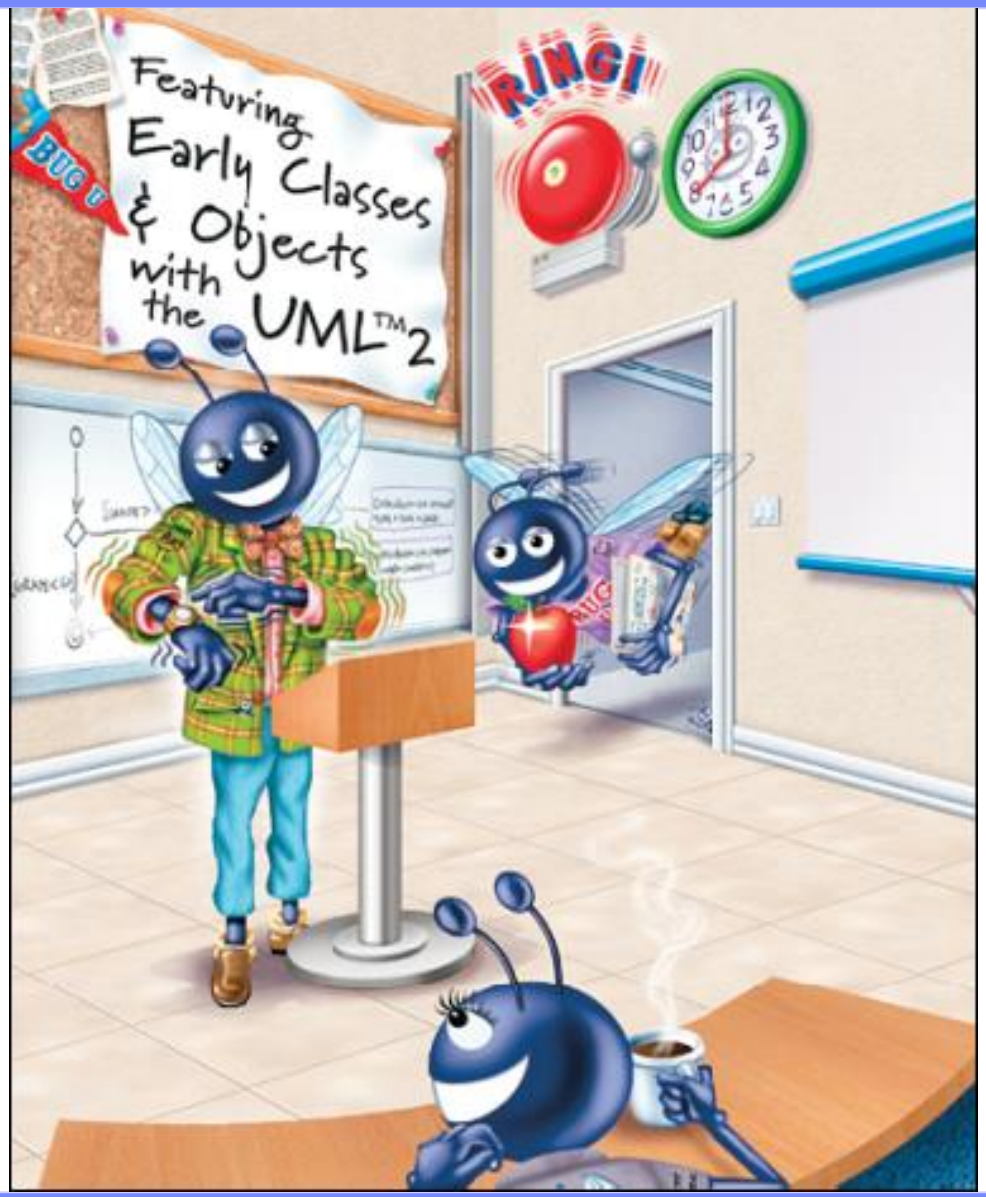


Lecture 12:
类与对象

第十五讲 输入输出流

学习目标：

- 使用C++面向对象的输入/输出流
- 能够格式化输入和输出
- I/O流类的层次结构
- 使用流操纵符
- 控制对齐和内容填充
- 判断输入/输出操作的成功与失败
- 把输出流连到输入上

12.1 简介

- C++ 标准库 input/output 功能
  - 许多I/O特性是面向对象:重载、引用等
  - 类型安全的I/O
    - ✓I/O 操作对数据类型敏感
    - ✓类型不匹配的数据不能“暗中”通过系统
  - 实现对用户自定义类型的I/O操作
    - ✓通过重载流插入符”<<”和流提取运算符”>>”来实现用户自定义类型数据的IO操作

12.2 流

- C++的I/O是以字节流的形式实现的，流实际上就是一个字节序列
  - 输入
    - ✓ 字节从输入设备(如键盘、磁盘、网络连接等)流向内存
  - 输出
    - ✓ 字节从内存流向输出设备(如显示器、打印机、磁盘、网络连接等)。
  - I/O花费的时间要比处理器处理数据的时间长得多

- 低层次“Low-level”， unformatted I/O
  - 只在设备和内存之间传输一些字节
  - 以单个字节为单位
  - 高速度，大容量
  - 使用起来不太方便
- 高层次“High-level”， formatted I/O
  - 若干个字节组合成有意义的单位
    - ✓ 如整数、浮点数、字符、字符串以及用户自定义类型的数据
  - 适合于大多数情况下的输入输出，但在处理大容量的I/O时不是很好

12.2.1 传统流与标准流

- C++ 传统流库
  - 允许输入/输出char字符（一个字节大小）
  - ASCII 字符集
  - Uses single bytes
  - 只能表示有限的字符集
- Unicode 字符集
  - 包括了世界主要的商用语言、数学符号等
  - www.unicode.org

- C++ 标准流库
  - 对Unicode字符进行I/O操作
  - 重新设计了传统C++流类
    - ✓ 利用类模板特化分别处理char和wchar_t字符类型
      - ◈ wchar_t存储Unicode字符

12.2.2 iostream 库的头文件

- <iostream>
  - 声明了所有I/O流操作所需的基础服务。
  - 定义了cin, cout, cerr 和 clog
  - 提供了非格式化和格式化的I/O服务
- <iomanip>

  声明了对于带参数化流操纵符的格式化I/O有用的服务
- <fstream>

  声明了用户控制的文件处理服务

12.2.3输入/输出流类和对象

- Iostream库提供了许多模板来处理通常的I/O操作
  - basic_istream
    - ✓ 支持输入流操作
  - basic_ostream
    - ✓ 支持输出流操作
  - basic_iostream
    - ✓ 支持输入和输出流操作

- iostream库提供了一组typedef为类模板特化提供别名
  - 为预定义的数据类型声明一个同义词
    - ✓ Example
      - ◈ typedef Card *CardPtr;
        - ◇ CardPtr 就是Card *的同义词
  - 更短更易理解的标识符

- <iostream> 库中的typedef
  - istream
    - ✓ 代表了basic_istream允许char字符输入的类模板特化
  - ostream
    - ✓ 代表了basic_ostream允许char字符输出的类模板特化
  - iostream
    - ✓ 代表了basic_stream允许char字符输入/输出的类模板特化

- I/O流模板层次
  - basic_istream和basic_ostream 派生自同一个类模板 basic_ios
  - basic_iostream 派生自 basic_istream 和 basic_ostream
    - ✓ 使用的是多重继承
- 流操纵符重载
  - 流插入运算符
    - ✓ 左移运算符 (<<)重载为流插入运算符来实现流的输出
  - 流提取运算符
    - ✓ 右移运算符 (>>)重载为流提取运算符来实现流的输入

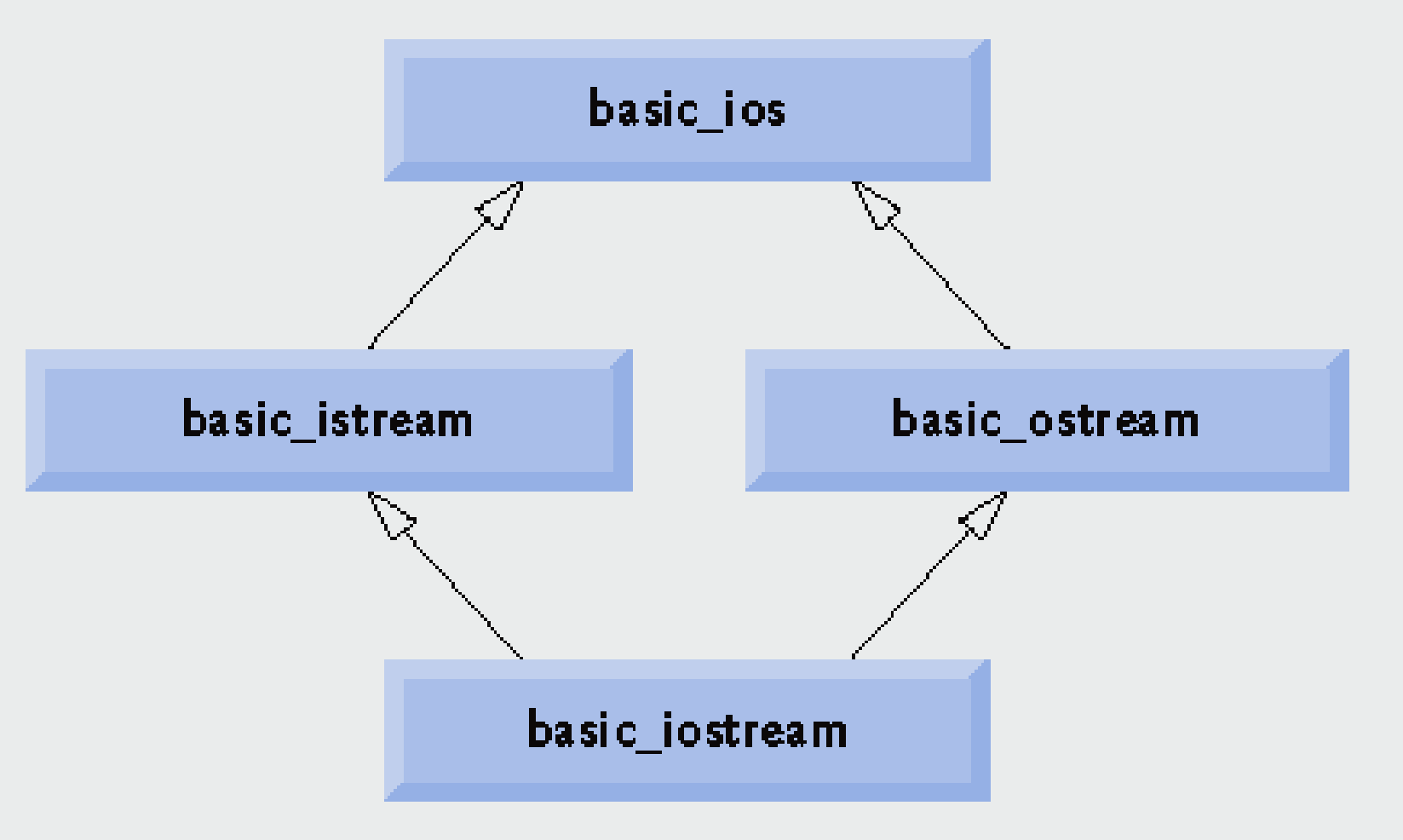


Fig. 15.1| Stream-I/O template hierarchy portion.

- 标准流对象
  - istream 实例
    - ✓ cin
      - ◈ 被连接到标准输入设备，通常是键盘
  - ostream 实例
    - ✓ cout
      - ◈ 被连接到标准输出设备，通常是显示器
    - ✓ cerr
      - ◈ 被连接到标准错误设备
      - ◈ 无缓冲——立刻显示
    - ✓ clog
      - ◈ 被连接到标准错误设备
      - ◈ 带缓冲——直到缓冲区满或被清空

- 文件处理模板
  - basic_ifstream
    - ✓ 用于文件输入
    - ✓ 派生自basic_istream
  - basic_ofstream
    - ✓ 用于文件输出
    - ✓ 派生自basic_ostream
  - basic_fstream
    - ✓ 用于文件输入和输出
    - ✓ 派生自basic_iostream
- typedef specializations
  - ifstream, ofstream and fstream

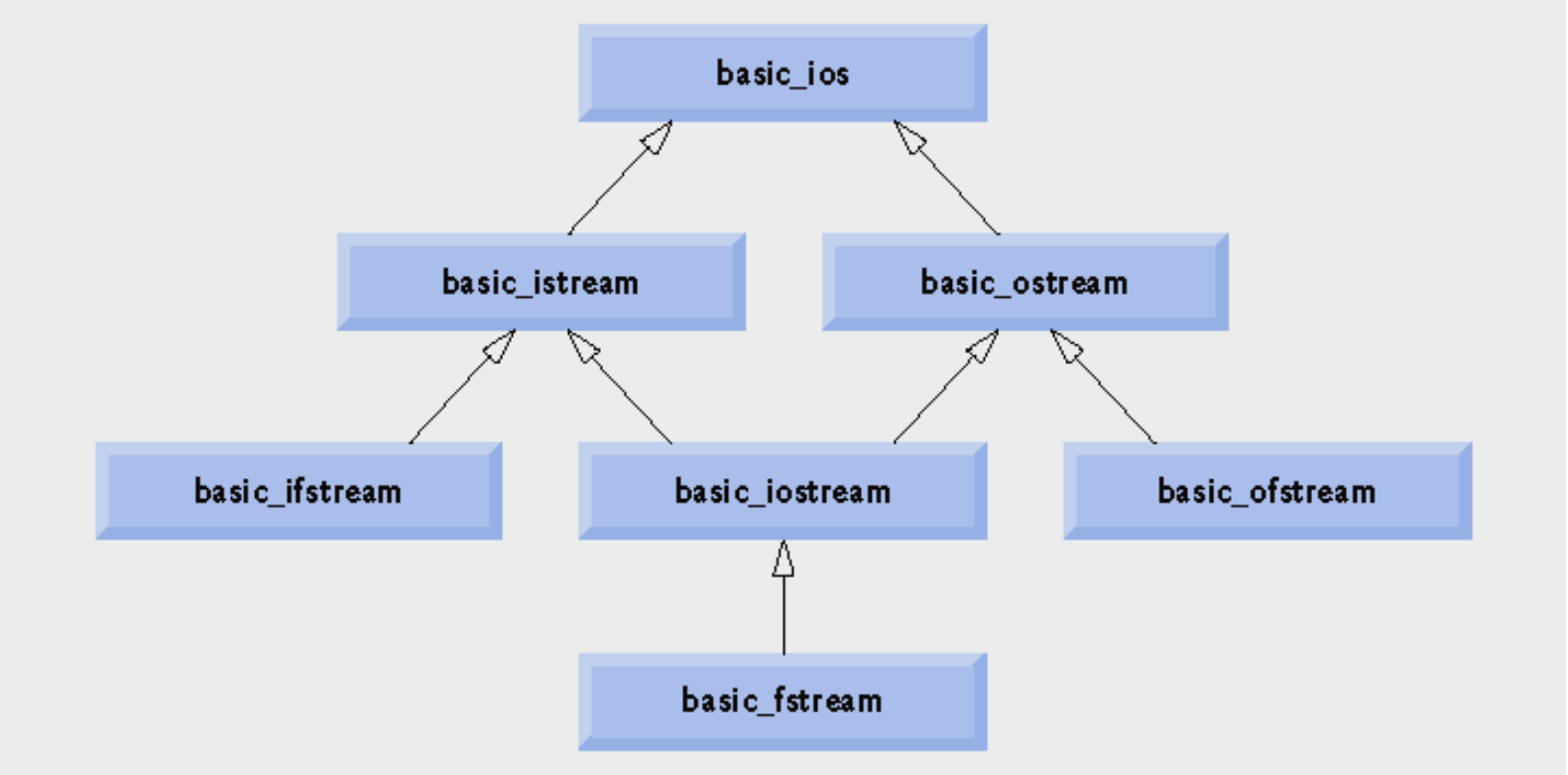


Fig. 15.2 | Stream-I/O template hierarchy portion showing the main file-processing templates.

12.3 输出流

- ostream 提供了格式化和非格式化的输出功能
  - 标准数据类型
  - 字符输出
  - 非格式化的输出
  - 整数输出
  - 浮点数输出
  - 指定宽度数据的输出
  - 区域填充
  - 科学计数法输出

12.3.1 char * 变量的输出

- 输出 char * (字符型指针)
  - C++能够自动判断数据类型，对比C是个优点，但有时会产生一些问题，如：
    char *ptr , 如何输出Ptr的地址?
  - 显然，不可以使用 <<
    - ✓ 已经被重载为将char*作为空字符结尾的字符串来打印
  - 解决
    - ✓ 将char * 强制转换为 void *
  - 地址是以16进制形式打印的

```cpp
// Fig. 15.3: Fig15_03.cpp
// Printing the address stored in a char * variable.
#include <iostream>
using std::cout;
using std::endl;

int main()
{
   char *word = "again";

   // display value of char *, then display value of char *
   // static_cast to void *
   cout << "Value of word is: " << word << endl
      << "Value of static_cast< void * >( word ) is: "
      << static_cast< void * >( word ) << endl;
   return 0;
} // end main
```

```
Value of word is: again
Value of static_cast< void * >( word ) is: 00428300
```

Cast the **`char *`** to a **`void *`**

Address prints as a hexadecimal (base-16) number

12.3.2 使用成员函数put进行字符输出

- Ostream的成员函数 put
  - 输出一个字符
  - 给ostream对象返回一个引用
    - 可以级联（串联）使用
  - 可以用代表一个ASCII值的数字表达式调用
  - Examples
    - cout.put('A');
    - cout.put('A').put('\n');
    - cout.put(65);

12.4 输入流

- istream 输入功能
  - 流提取运算符 (重载>>)
    - ✓ 跳过空白字符（空格，制表符和换行符）
    - ✓ 给接收到所提取的信息的流对象istream返回一个引用
  - 若引用被用做判断条件，将隐式调用流重载的void * 强制转换运算符函数
    - ✓ 转化为非空指针 (true)或空指针 (false)
      - ◈ 基于最后输入操作的成功与否
        - ◇ 试图越过流的末尾进行读取操作时

- 状态位
  - ✓ 控制流的状态
  - ✓ failbit
    - ◈ 当输入错误数据类型
  - ✓ badbit
    - ◈ 当流提取操作失败

12.4.1 get 和 getline 成员函数

- istream 成员函数 get
  - 不带参数
    - ✓ 从指定的输入流中读取(输入)一个字符，并将该字符作为函数调用的值返回
      - ◇ 包括空白字符和非图形字符
    - ✓ 当遇到输入流中的文件结束符时，返回EOF
  - 带一个字符引用参数
    - ✓ 读取输入流中的下一个字符(包括空白字符),并将这个字符存在其引用的字符参数内
    - ✓ 给调用它的istream对象返回一个引用

- 带3个参数：一个字符数组，一个数组长度，和一个分隔符 (默认值为'\n')
  - ✓ 读取并存储字符到字符数组中
  - ✓ 或者在读取比指定的最大字符数少一个字符后结束，或者在遇到分隔符时结束
    - ◈ 分隔符不会放到字符数组中，仍然会保留在输入流中
  - ✓ 空字符会被插入到字符数组中作为结束符号

- istream 成员函数 eof
  - 输出值为0(false)表明没有在cin中遇到文件结束符 end-of-file
  - 输出值为1(true)表明在cin中遇到文件结束符 end-of-file

```cpp
// Fig. 15.4: Fig15_04.cpp
// Using member functions get, put and eof.
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main()
{
   int character; // use int, because char cannot represent EOF

   // prompt user to enter line of text
   cout << "Before input, cin.eof() is " << cin.eof() << endl
      << "Enter a sentence followed by end-of-file:" << endl;

   // use get to read each character; use put to display it
   while ( ( character = cin.get() ) != EOF )
      cout.put( character );
```

Call **eof** member function before end-of-file is reached

while loop terminates when **get** member function returns **EOF**

```
19
20    // display end-of-file character
21    cout << "\nEOF in this system is: " << character << endl;
22    cout << "After input of EOF, cin.eof() is " << cin.eof() << endl;
23    return 0;
24 } // end main
```

```
Before input, cin.eof() is 0
Enter a sentence followed by end-of-file:
Testing the get and put member functions
Testing the get and put member functions
^Z

EOF in this system is: -1
After input of EOF, cin.eof() is 1
```

Display **character**, which currently contains the value of **EOF**

Call **eof** member function after end-of-file is reached

End-of-file is represented by <*ctrl*>-*z* on Microsoft Windows systems, <*ctrl*>-*d* on UNIX and Macintosh systems.

```cpp
// Fig. 15.5: Fig15_05.cpp
// Contrasting input of a string via cin and cin.get.
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main()
{
   // create two char arrays, each with 80 elements
   const int SIZE = 80;
   char buffer1[ SIZE ];
   char buffer2[ SIZE ];

   // use cin to input characters into buffer1
   cout << "Enter a sentence:" << endl;
   cin >> buffer1;

   // display buffer1 contents
   cout << "\nThe string read with cin was:" << endl
      << buffer1 << endl << endl;

   // use cin.get to input characters into buffer2
   cin.get( buffer2, SIZE );

   // display buffer2 contents
   cout << "The string read with cin.get was:" << endl
      << buffer2 << endl;
   return 0;
} // end main
```

Use stream extraction with **cin**

Call three-argument version of member function **get** (third argument is default value **'\n'**)

```
Enter a sentence:
Contrasting string input with cin and cin.get

The string read with cin was:
Contrasting

The string read with cin.get was:
 string input with cin and cin.get
```

Stream extraction operation reads up to first white-space character;
剩余的部分依然在输入队列中，会被下一个cin读入

get member function reads up to the delimiter character **'\n'**

- 成员函数getline
  - 类似第三个版本的get成员函数，在末尾插入空字符
  - 从流中移除分隔符，即读入后丢弃
  - 带3个参数：一个字符数组，一个数组长度，和一个分隔符 (默认值为'\n')
    - ✓ 读取并存储字符到字符数组中
    - ✓ 或者在读取比指定的最大字符数少一个字符后结束，或者在遇到分隔符时结束
      - ◇ 分隔符不会放到字符数组中，会从输入流中移除
    - ✓ 输入数组的末尾插入一个空字符

```
1  // Fig. 15.6: Fig15_06.cpp
2  // Inputting characters using cin member function getline.
3  #include <iostream>
4  using std::cin;
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    const int SIZE = 80;
11    char buffer[ SIZE ]; // create array of 80 characters
12
13    // input characters in buffer via cin function getline
14    cout << "Enter a sentence:" << endl;
15    cin.getline( buffer, SIZE );
16
17    // display buffer contents
18    cout << "\nThe sentence entered is:" << endl << buffer << endl;
19    return 0;
20 } // end main
```

Call member function **getline**

```
Enter a sentence:
Using the getline member function

The sentence entered is:
Using the getline member function
```

12.4.2 istream的成员函数peek, putback and ignore

- istream 成员函数 ignore
  - 读取并丢弃一定数量的字符或者是遇到指定分隔符时停止
    - ✓ 默认丢弃一个字符
    - ✓ 默认的分隔符为 EOF
- istream 成员函数 putback
  - 将先前使用get函数从输入流里获得的字符再放回到流中
- istream 成员函数 peek
  - 返回输入流中的下一个字符，但不将它从流里去除

12.4.3 类型安全 I/O

- C++ 提供类型安全的I/O
  - 重载<< 和 >>运算符来接收各种指定类型的数据项
    - ✓ 如果没有为用户自定义类型重载运算符<<和>>,试图输入或输出一个该用户自定义类型的对象的内容，那么编译器就会报错
  - 如果遇到意料之外的数据类型，各种相应的错误位就会别设置
    - ✓ 用户可以通过检测错误位来判断I/O操作是否成功
  - 使程序“保持在控制之下”

12.5 使用read, write 和 gcount的非格式化的 I/O

- istream 成员函数 read
  - 把一定量的字节写入字符数组
  - 如果读取的字符个数少于指定的数目，可以设置标志位 failbit
- istream成员函数 gcount
  - 返回最近一次输入操作所读取的字符个数
- ostream成员函数 write
  - 把一定量的字节从字符数组中输出

```
1  // Fig. 15.7: Fig15_07.cpp
2  // Unformatted I/O using read, gcount and write.
3  #include <iostream>
4  using std::cin;
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    const int SIZE = 80;
11    char buffer[ SIZE ]; // create array of 80 characters
12
13    // use function read to input characters into buffer
14    cout << "Enter a sentence:" << endl;
15    cin.read( buffer, 20 );
16
17    // use functions write and gcount to display buffer characters
18    cout << endl << "The sentence entered was:" << endl;
19    cout.write( buffer, cin.gcount() );
20    cout << endl;
21    return 0;
22 } // end main
```

read 20 bytes from the input stream to **buffer**

write out as many characters as were read by the last input operation from **buffer** to the output stream

```
Enter a sentence:
Using the read, write, and gcount member functions
The sentence entered was:
Using the read, writ
```

12.6 流操纵符简介

- 流操纵符功能
  - 设置域宽
  - 设置精度
  - 设置和取消格式状态
  - 设置域的填充字符
  - 刷新流
  - 向输出流中添加新行并刷新流
  - 在输出流中添加一个空字符并跳过输入流中的空白

12.6.1 整型流的基数：dec，oct，hex and setbase

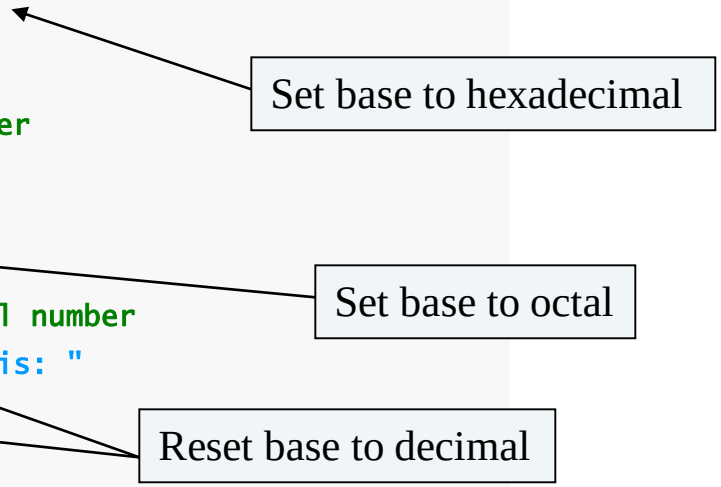
- 通过插入操纵符，更改流中整数的基数
  - hex 16进制
  - oct 8进制
  - dec 10进制
  - setbase 参数化操纵符
    - ✓ 使用一个整数参数: 10, 8 or 16
    - ✓ 将基数设置为10，8或16进制
    - ✓ 需要头文件 <iomanip>
  - 粘性设置
    - ✓ 该设置一直有效，直到改变为另一种基数

```
1  // Fig. 15.8: Fig15_08.cpp
2  // Using stream manipulators hex, oct, dec and setbase.
3  #include <iostream>
4  using std::cin;
5  using std::cout;
6  using std::dec;
7  using std::endl;
8  using std::hex;
9  using std::oct;
10
11 #include <iomanip>
12 using std::setbase;
13
```

Parameterized stream manipulator **`setbase`** is in header file **`<iomanip>`**

```
14 int main()
15 {
16    int number;
17
18    cout << "Enter a decimal number: ";
19    cin >> number; // input number
20
21    // use hex stream manipulator to show hexadecimal number
22    cout << number << " in hexadecimal is: " << hex
23       << number << endl;
24
25    // use oct stream manipulator to show octal number
26    cout << dec << number << " in octal is: "
27       << oct << number << endl;
28
29    // use setbase stream manipulator to show decimal number
30    cout << setbase( 10 ) << number << " in decimal is: "
31       << number << endl;
32    return 0;
33 } // end main
```

```
Enter a decimal number: 20
20 in hexadecimal is: 14
20 in octal is: 24
20 in decimal is: 20
```



12.6.2 浮点精度（precision, setprecision）

- 控制浮点数的精度
  - 小数点右边的位数
  - setprecision 参数化流操纵符
  - precision 成员函数
    - ✓ 无参数调用时，返回当前的精度设置
  - 粘性设置

```cpp
// Fig. 15.9: Fig15_09.cpp
// Controlling precision of floating-point values.
#include <iostream>
using std::cout;
using std::endl;
using std::fixed;

#include <iomanip>
using std::setprecision;

#include <cmath>
using std::sqrt; // sqrt prototype

int main()
{
   double root2 = sqrt( 2.0 ); // calculate square root of 2
   int places; // precision, vary from 0-9

   cout << "Square root of 2 with precisions 0-9." << endl
      << "Precision set by ios_base member function "
      << "precision:" << endl;

   cout << fixed; // use fixed-point notation

   // display square root using ios_base function precision
   for ( places = 0; places <= 9; places++ )
   {
      cout.precision( places );
      cout << root2 << endl;
   } // end for
```

Use member function **precision** to set **cout** to display **places** digits to the right of the decimal point

```
31
32    cout << "\nPrecision set by stream manipulator "
33       << "setprecision:" << endl;
34
35    // set precision for each digit, then display square root
36    for ( places = 0; places <= 9; places++ )
37       cout << setprecision( places ) << root2 << endl;
38
39    return 0;
40 } // end main
```

Use parameterized stream manipulator **`setprecision`** to set **`cout`** to display **`places`** digits to the right of the decimal point

```
Square root of 2 with precisions 0-9.
Precision set by ios_base member function precision:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562

Precision set by stream manipulator setprecision:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562
```

12.6.3 域宽（width, setw）

- width设置当前的域宽(即输入输出的字符数)并返回以前设置的域宽
  - (针对ostream) 输出值所占的字符位数
    - ✓ 如果输出值的宽度比设置的域宽小，插入填充字符进行填充
    - ✓ 如果宽度比设置的域宽大,显示数据并不会被截断，系统会输出所有位。
  - (针对 istream)可以输入的最大字符数
    - ✓ 对字符数组输入时，读入的最大字符数比指定宽度小1个字符因为必须在输入的字符串中插入空字符

- 成员函数width 的基类是 ios_base
  - ✓ 设置域宽
  - ✓ 返回先前的域宽
    - ◈ width 进行无参调用时，返回当前域宽
- Setw参数化流操纵符
  - ✓ 设置域宽Sets the field width
- 非粘性

```cpp
// Fig. 15.10: Fig15_10.cpp
// Demonstrating member function width.
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main()
{
   int widthValue = 4;
   char sentence[ 10 ];

   cout << "Enter a sentence:" << endl;
   cin.width( 5 ); // input only 5 characters from sentence

   // set field width, then display characters based on that width
   while ( cin >> sentence )
   {
      cout.width( widthValue++ );
      cout << sentence << endl;
      cin.width( 5 ); // input 5 more characters from sentence
   } // end while

   return 0;
} // end main
```

```
Enter a sentence:
This is a test of the width member function
This
   is
     a
   test
      of
      the
      widt
           h
        memb
            er
          func
           tion
```

常见编程错误：宽度设置只适用于下一次的输入与输出；之后的宽度被隐式设置为0. 宽度设置没有粘性，假定设置一次就能适用于以后所有的输出是逻辑错误。

12.6.4 用户自定义输出流操纵符

- 程序员可以创建自己的流操纵符
  - 输出流操纵符
    - ✓ 返回类型和参数必须是ostream &类型的。

```cpp
// Fig. 15.11: Fig15_11.cpp
// Creating and testing user-defined, nonparameterized
// stream manipulators.
#include <iostream>
using std::cout;
using std::flush;
using std::ostream;

// bell manipulator (using escape sequence \a)
ostream& bell( ostream& output )
{
   return output << '\a'; // issue system beep
} // end bell manipulator

// carriageReturn manipulator (using escape sequence \r)
ostream& carriageReturn( ostream& output )
{
   return output << '\r'; // issue carriage return
} // end carriageReturn manipulator

// tab manipulator (using escape sequence \t)
ostream& tab( ostream& output )
{
   return output << '\t'; // issue tab
} // end tab manipulator
```

```
26
27 // endLine manipulator (using escape sequence \n and member
28 // function flush)
29 ostream& endLine( ostream& output )
30 {
31    return output << '\n' << flush; // issue endl-like end of line
32 } // end endLine manipulator
33
34 int main()
35 {
36    // use tab and endLine manipulators
37    cout << "Testing the tab manipulator:" << endLine
38       << 'a' << tab << 'b' << tab << 'c' << endLine;
39
40    cout << "Testing the carriageReturn and bell manipulators:"
41       << endLine << ".....";
42
43    cout << bell; // use bell manipulator
44
45    // use carriageReturn and endLine manipulators
46    cout << carriageReturn << "-----" << endLine;
47    return 0;
48 } // end main
```

```
Testing the tab manipulator:
a       b       c
Testing the carriageReturn and bell manipulators:
-----.....
```

12.7 流的格式状态和流操纵符

- 在I/O流操作过程中，流操纵符可以指定各种格式
  - 所有流操纵符都属于类ios_base

Stream Manipulator	Description
`skipws`	Skip white-space characters on an input stream. This setting is reset with stream manipulator `noskipws`.
`left`	Left justify output in a field. Padding characters appear to the right if necessary.
`right`	Right justify output in a field. Padding characters appear to the left if necessary.
`internal`	Indicate that a number's sign should be left justified in a field and a number's magnitude should be right justified in that same field (i.e., padding characters appear between the sign and the number).
`dec`	Specify that integers should be treated as decimal (base 10) values.
`oct`	Specify that integers should be treated as octal (base 8) values.
`hex`	Specify that integers should be treated as hexadecimal (base 16) values.

Fig. 15.12 | Format state stream manipulators from <iostream>. (Part 1 of 2)

Stream Manipulator	Description
`showbase`	Specify that the base of a number is to be output ahead of the number (a leading `0` for octals; a leading `0x` or `0X` for hexadecimals). This setting is reset with stream manipulator `noshowbase`.
`showpoint`	Specify that floating-point numbers should be output with a decimal point. This is used normally with `fixed` to guarantee a certain number of digits to the right of the decimal point, even if they are zeros. This setting is reset with stream manipulator `noshowpoint`.
`uppercase`	Specify that uppercase letters (i.e., `X` and `A` through `F`) should be used in a hexadecimal integer and that uppercase `E` should be used when representing a floating-point value in scientific notation. This setting is reset with stream manipulator `nouppercase`.
`showpos`	Specify that positive numbers should be preceded by a plus sign (+). This setting is reset with stream manipulator `noshowpos`.
`scientific`	Specify output of a floating-point value in scientific notation.
`fixed`	Specify output of a floating-point value in fixed-point notation with a specific number of digits to the right of the decimal point.

Fig. 15.12 | Format state stream manipulators from <iostream>. (Part 2 of 2)

12.7.1 尾数零和小数点（showpoint）

- 流操纵符 showpoint
  - 强制要求浮点数的输出必须带小数点和尾数零
    - ✓ Example
      - ◈ 79.0 输出显示为 79.0000而不是 79
  - noshowpoint

```
1  // Fig. 15.13: Fig15_13.cpp
4  #include <iostream>
7  using std::showpoint;
8
9  int main()
10 {
12    cout << "Before using showpoint" << endl
13       << "9.9900 prints as: " << 9.9900 << endl
14       << "9.9000 prints as: " << 9.9000 << endl
15       << "9.0000 prints as: " << 9.0000 << endl << endl;

18    cout << showpoint
19       << "After using showpoint" << endl
20       << "9.9900 prints as: " << 9.9900 << endl
21       << "9.9000 prints as: " << 9.9000 << endl
22       << "9.0000 prints as: " << 9.0000 << endl;
23    return 0;
24 } // end main
```

```
Before using showpoint
9.9900 prints as: 9.99
9.9000 prints as: 9.9
9.0000 prints as: 9

After using showpoint
9.9900 prints as: 9.99000
9.9000 prints as: 9.90000
9.0000 prints as: 9.00000
```

12.7.2 对齐（left, right and internal）

- 使域对齐
  - 操纵符left
    - ✓ 使域左对齐
    - ✓ 在右边填充字符
  - 操纵符right
    - ✓ 使域右对齐
    - ✓ 在左边填充字符
  - 操纵符internal
    - ✓ 数字的符号左对齐
      - ◈ Showpos流操纵符显示的基数
    - ✓ 数值部分右对齐
    - ✓ 在中间填充字符

```
1  // Fig. 15.14: Fig15_14.cpp
2  // Demonstrating left justification and right justification.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::left;
7  using std::right;
8
9  #include <iomanip>
10 using std::setw;
11
12 int main()
13 {
14    int x = 12345;
15
16    // display x right justified (default)
17    cout << "Default is right justified:" << endl
18       << setw( 10 ) << x;
19
20    // use left manipulator to display x left justified
21    cout << "\n\nUse std::left to left justify x:\n"
22       << left << setw( 10 ) << x;
23
24    // use right manipulator to display x right justified
25    cout << "\n\nUse std::right to right justify x:\n"
26       << right << setw( 10 ) << x << endl;
27    return 0;
28 } // end main
```

```
Default is right justified:
     12345

Use std::left to left justify x:
12345

Use std::right to right justify x:
     12345
```

```cpp
// Fig. 15.15: Fig15_15.cpp
// Printing an integer with internal spacing and plus sign.
#include <iostream>
using std::cout;
using std::endl;
using std::internal;
using std::showpos;

#include <iomanip>
using std::setw;

int main()
{
   // display value with internal spacing and plus sign
   cout << internal << showpos << setw( 10 ) << 123 << endl;
   return 0;
} // end main
```

```
+       123
```

12.7.3 内容填充（fill，setfill）

- 指定对齐域的填充字符
  - 成员函数 fill
    - ✓ 指定填充字符
    - ✓ 如没指定，空格符进行填充
    - ✓ 返回设定之前的填充字符
  - 流操纵符setfill
    - ✓ 指定填充字符

```
1  // Fig. 15.16: Fig15_16.cpp
2  // Using member function fill and stream manipulator setfill to change
3  // the padding character for fields larger than the printed value.
4  #include <iostream>
5  using std::cout;
6  using std::dec;
7  using std::endl;
8  using std::hex;
9  using std::internal;
10 using std::left;
11 using std::right;
12 using std::showbase;
13
14 #include <iomanip>
15 using std::setfill;
16 using std::setw;
17
18 int main()
19 {
20    int x = 10000;
21
22    // display x
23    cout << x << " printed as int right and left justified\n"
24       << "and as hex with internal justification.\n"
25       << "Using the default pad character (space):" << endl;
26
27    // display x with base
28    cout << showbase << setw( 10 ) << x << endl;
29
```

```
30    // display x with left justification
31    cout << left << setw( 10 ) << x << endl;
32
33    // display x as hex with internal justification
34    cout << internal << setw( 10 ) << hex << x << endl << endl;
35
36    cout << "Using various padding characters:" << endl;
37
38    // display x using padded characters (right justification)
39    cout << right;
40    cout.fill( '*' );
41    cout << setw( 10 ) << dec << x << endl;
42
43    // display x using padded characters (left justification)
44    cout << left << setw( 10 ) << setfill( '%' ) << x << endl;
45
46    // display x using padded characters (internal justification)
47    cout << internal << setw( 10 ) << setfill( '^' ) << hex
48       << x << endl;
49    return 0;
50 } // end main
```

```
10000 printed as int right and left justified
and as hex with internal justification.
Using the default pad character (space):
     10000
10000
0x    2710

Using various padding characters:
*****10000
10000%%%%%
0x^^^^2710
```

12.7.4整型流的基数（dec，oct，hex，showbase）

- 流插入的整数基数
  - 流操纵符 dec, hex and oct
- 流提取时的整数基数
  - 0开头的整数
    - ✓ 八进制数
  - 0x或0X
    - ✓ 十六进制数
  - All other integers
    - ✓ 其余按十进制数

- 流操纵符 showbase
  - 要求整数的基数被输出
    - ✓ 默认输出是十进制
    - ✓ 0开头的是八进制整数
    - ✓ 0x或0X开头的是十六进制整数
  - 流操纵符noshowbase可以取消showbase的设定

```cpp
// Fig. 15.17: Fig15_17.cpp
// Using stream manipulator showbase.
#include <iostream>
using std::cout;
using std::endl;
using std::hex;
using std::oct;
using std::showbase;

int main()
{
   int x = 100;

   // use showbase to show number base
   cout << "Printing integers preceded by their base:" << endl
      << showbase;

   cout << x << endl; // print decimal value
   cout << oct << x << endl; // print octal value
   cout << hex << x << endl; // print hexadecimal value
   return 0;
} // end main
```

```
Printing integers preceded by their base:
100
0144
0x64
```

12.7.5 浮点数、科学计数法和定点小数计数法（scientific, fixed）

- 流操纵符 scientific
  - 要求浮点数以科学计数法的格式输出
- 流操纵符fixed
  - 要求浮点数以指定小数位数的形式显示
    - ✓ 可以使用成员函数 precision 或流操纵符 setprecision
- 不是用 scientific 或 fixed
  - 浮点数的值决定浮点数的输出格式

```
1  // Fig. 15.18: Fig15_18.cpp
2  // Displaying floating-point values in system default,
3  // scientific and fixed formats.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7  using std::fixed;
8  using std::scientific;
9
```

```
10 int main()
11 {
12     double x = 0.001234567;
13     double y = 1.946e9;
14
16     cout << "Displayed in default format:" << endl
17          << x << '\t' << y << endl;
18
20     cout << "\nDisplayed in scientific format:" << endl
21          << scientific << x << '\t' << y << endl;
22
24     cout << "\nDisplayed in fixed format:" << endl
25          << fixed << x << '\t' << y << endl;
26     return 0;
27 } // end main
```

```
Displayed in default format:
0.00123457      1.946e+009

Displayed in scientific format:
1.234567e-003   1.946000e+009

Displayed in fixed format:
0.001235        1946000000.000000
```


12.7.6 大写/小写控制（uppercase）

- 流操纵符 uppercase
  - 在输出十六进制整数时输出大写字母X，也可以使十六进制整数中的字母以大写A-F形式显示
  - 在输出科学计数格式的浮点数时输出大写E
  - 这些字母默认情况下是小写的
  - 取消 uppercase 设置可以使用nouppercase

```cpp
// Fig. 15.19: Fig15_19.cpp

#include <iostream>

using std::hex;
using std::showbase;
using std::uppercase;

int main()
{
   cout << "Printing uppercase letters in scientific" << endl
      << "notation exponents and hexadecimal values:" << endl;

   cout << uppercase << 4.345e10 << endl
      << hex << showbase << 123456789 << endl;
   return 0;
} // end main
```

```
Printing uppercase letters in scientific
notation exponents and hexadecimal values:
4.345E+010
0X75BCD15
```

12.7.7 指定布尔格式（boolalpha）

- 流操纵符 boolalpha
  - 使bool变量以“true”或“false”形式输出
    - ✓ 默认情况下 bool变量输出是 0 或 1
  - noboolalpha 设置bools 变量显示0 或 1
  - Bool是粘性设置

```
1  // Fig. 15.20: Fig15_20.cpp
2  // Demonstrating stream manipulators boolalpha and noboolalpha.
3  #include <iostream>
4  using std::boolalpha;
5  using std::cout;
6  using std::endl;
7  using std::noboolalpha;
8
9  int main()
10 {
11    bool booleanValue = true;
12
13    // display default true booleanValue
14    cout << "booleanValue is " << booleanValue << endl;
15
16    // display booleanValue after using boolalpha
17    cout << "booleanValue (after using boolalpha) is "
18       << boolalpha << booleanValue << endl << endl;
```

```
19
20    cout << "switch booleanValue and use noboolalpha" << endl;
21    booleanValue = false; // change booleanValue
22    cout << noboolalpha << endl; // use noboolalpha
23
24    // display default false booleanValue after using noboolalpha
25    cout << "booleanValue is " << booleanValue << endl;
26
27    // display booleanValue after using boolalpha again
28    cout << "booleanValue (after using boolalpha) is "
29       << boolalpha << booleanValue << endl;
30    return 0;
31 } // end main
```

```
booleanValue is 1
booleanValue (after using boolalpha) is true

switch booleanValue and use noboolalpha

booleanValue is 0
booleanValue (after using boolalpha) is false
```

12.7.8 通过成员函数flags设置和重置格式状态

- 成员函数 flags
  - 不带参数
    - ✓ 将当前的格式设置为以fmtflags数据类型返回
      - ◇ 表示了格式状态
  - 带一个 fmtflags 参数
    - ✓ 将格式状态转换为参数指定的格式状态
    - ✓ 返回之前的状态设定
  - 原始设定值可能根据不同系统有所不同
  - Fmtflags是属于类ios_base

```cpp
// Fig. 15.21: Fig15_21.cpp
// Demonstrating the flags member function.
#include <iostream>
using std::cout;
using std::endl;
using std::ios_base;
using std::oct;
using std::scientific;
using std::showbase;

int main()
{
   int integerValue = 1000;
   double doubleValue = 0.0947628;

   // display flags value, int and double values (original format)
   cout << "The value of the flags variable is: " << cout.flags()
      << "\nPrint int and double in original format:\n"
      << integerValue << '\t' << doubleValue << endl << endl;

   // use cout flags function to save original format
   ios_base::fmtflags originalFormat = cout.flags();
   cout << showbase << oct << scientific; // change format

   // display flags value, int and double values (new format)
   cout << "The value of the flags variable is: " << cout.flags()
      << "\nPrint int and double in a new format:\n"
      << integerValue << '\t' << doubleValue << endl << endl;

   cout.flags( originalFormat ); // restore format
```

Save the stream's original format state

Restore the original format settings

```
31
32    // display flags value, int and double values (original format)
33    cout << "The restored value of the flags variable is: "
34       << cout.flags()
35       << "\nPrint values in original format again:\n"
36       << integerValue << '\t' << doubleValue << endl;
37    return 0;
38 } // end main
```

```
The value of the flags variable is: 513
Print int and double in original format:
1000    0.0947628

The value of the flags variable is: 012011
Print int and double in a new format:
01750   9.476280e-002

The restored value of the flags variable is: 513
Print values in original format again:
1000    0.0947628
```

12.8 流的错误状态

- eofbit
  - 在遇到文件尾end-of-file时背设置
  - 使用成员函数eof 判断
    - ✓ 如果遇到文件尾返回true
    - ✓ 否则返回false
- failbit
  - 当在流中发生格式错误
    - ✓ 例如要求输入整数时，流中却是非数字字符
    - ✓ 这些字符不会丢失，仍然保留在流中
  - 使用成员函数fail 判断
  - 通常这种严重错误时可以恢复的

- badbit
  - 当发生数据丢失错误时被设置
  - 使用成员函数bad 判断
  - 错误不可修复
- goodbit
  - 如果流中的eofbit, failbit or badbit都没被设置，goodbit被设置
  - 使用成员函数good 判断
- 成员函数 rdstate
  - 返回流的错误状态
    - ✓ 不正确的eofbit, badbit, failbit 和 goodbit
  - 使用单独的成员函数更好

- 成员函数 clear
  - 将流的状态重置
  - 默认参数是 goodbit
  - Examples
    - ✓ cin.clear();
      - Clears cin and sets goodbit
    - ✓ cin.clear(ios::failbit);
      - Sets failbit

```cpp
// Fig. 15.22: Fig15_22.cpp
// Testing error states.
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main()
{
   int integerValue;

   // display results of cin functions
   cout << "Before a bad input operation:"
      << "\ncin.rdstate(): " << cin.rdstate()
      << "\n    cin.eof(): " << cin.eof()
      << "\n   cin.fail(): " << cin.fail()
      << "\n    cin.bad(): " << cin.bad()
      << "\n   cin.good(): " << cin.good()
      << "\n\nExpects an integer, but enter a character: ";

   cin >> integerValue; // enter character value
   cout << endl;

   // display results of cin functions after bad input
   cout << "After a bad input operation:"
      << "\ncin.rdstate(): " << cin.rdstate()
      << "\n    cin.eof(): " << cin.eof()
      << "\n   cin.fail(): " << cin.fail()
      << "\n    cin.bad(): " << cin.bad()
      << "\n   cin.good(): " << cin.good() << endl << endl;
```

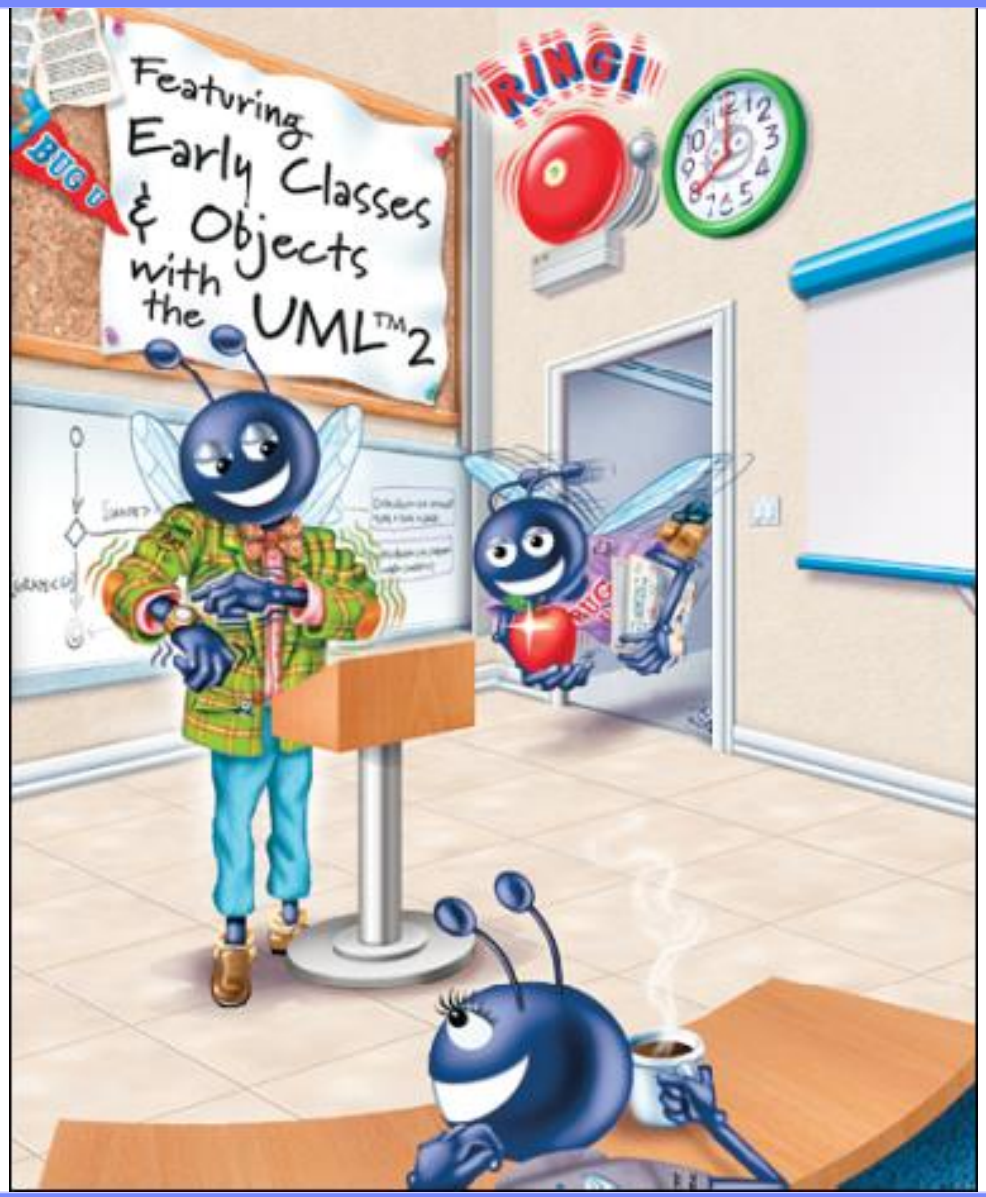
```
31
32    cin.clear(); // clear stream
33
34    // display results of cin functions after clearing cin
35    cout << "After cin.clear()" << "\ncin.fail(): " << cin.fail()
36       << "\ncin.good(): " << cin.good() << endl;
37    return 0;
38 } // end main
```

```
Before a bad input operation:
cin.rdstate(): 0
    cin.eof(): 0
   cin.fail(): 0
    cin.bad(): 0
   cin.good(): 1

Expects an integer, but enter a character: A

After a bad input operation:
cin.rdstate(): 2
    cin.eof(): 0
   cin.fail(): 1
    cin.bad(): 0
   cin.good(): 0

After cin.clear()
cin.fail(): 0
cin.good(): 1
```

- basic_ios 成员函数 operator!
  - 如果badbit位被设置，或是failbit位被设置，返回true
- basic_ios 成员函数 void *
  - 如果badbit位被设置，或是failbit位被设置，返回false

12.9 将输出流连接到输出流

- istream 成员函数 tie
  - 使istream和 ostream操作同步
    - ✓ 确保输出在其接下来的输入操作之前被显示
  - Examples
    - ✓ cin.tie(&cout);
      - ◈ 将标准输入和标准输出连接起来
      - ◈ C++会自动进行这个操作
    - ✓ inputStream.tie(0);
      - ◈ 从输出流上解除对输入流inputStream的连接



Thank you!