华章科技

深刻解读　精彩注释

Essential C++
（注释版）

Essential C++

（美） Stanley B. Lippman 著　　徐旭铭 金萍华 注释

- 《C++ Primer》作者编著的入门经典
- 众多专家推荐的C++初级读物
- 深刻解读　精彩注释

机械工业出版社
China Machine Press

Essential C++
（注释版）

Essential C++

（美） Stanley B. Lippman 著　　徐旭铭 金萍华 注释

机械工业出版社
China Machine Press

本书版权登记号：图字：01-2010-1535

图书在版编目（CIP）数据

Essential C++（注释版）/（美）斯坦利（Lippman, S. B.）著；徐旭铭，金萍华注释. —北京：机械工业出版社，2010.8

（经典重读）

书名原文：Essential C++

ISBN 978-7-111-31221-5

Ⅰ. E…　Ⅱ. ① 斯…　② 徐…　③ 金…　Ⅲ. C语言－程序设计－英文　Ⅳ. TP312

中国版本图书馆CIP数据核字（2010）第129159号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：秦　健

北京京师印务有限公司印刷

2010年10月第1版第1次印刷

186mm × 240mm · 18.75印张

标准书号：ISBN 978-7-111-31221-5

定价：49.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991；88361066

购书热线：（010）68326294；88379649；68995259

投稿热线：（010）88379604

读者信箱：hzjsj@hzbook.com

序 言

本书是Stanley继《C++ Primer》后的又一本经典著作。它的定位和《C++ Primer》颇为不同,《C++ Primer》是一本巨细靡遗的教科书，而《Essential C++》的目的是为了让你能够迅速了解C++这门语言以便展开工作，所以一切都是从实际出发，讲求的是效率。

注解一本书，特别是这样一本精简版教程的难点就在于取舍和细节程度（Stanley自己也在前言里提到了这两点），书中剔除了很多C++的高级特性，也回避了很多细节内容。因此我们也要时刻提醒自己这些高阶内容不属于本书的范畴，不应该在书中提到它们。在讲解的时候，书中主要关注的地方是语言的“正确”用法，而C++的灵活性又导致了实际工作中存在很多不那么“正规”的用法，这是我们在注解里用笔墨较多的一个方面，主要是提醒你这里有陷阱，那么写会有什么效果等。另外由于C和C++之间千丝万缕的关系，在碰到遗留代码的时候，往往会看到很多C风格或者是“传统”C++风格的代码，这些则是注解关注的另一个方面。最后就是一些扩展内容，遵循本书的原则，我们也不做过多的展开，只是列出了可以参考的读物，让有兴趣的读者能自行查阅。

在注解过程中，我们查阅了很多其他书籍，并在相关的知识点上给出了参考读物。在此要感谢这些好书的作者们，是他们让我们的工作轻松了不少。

最后，感谢华章公司编辑的支持和辛劳，让我们可以看到一本又一本的好书。

徐旭铭

前　言

好吧，这本书实在是非常薄。我的意思是，这太叫人意外了。我的《C++ Primer》算上索引、标题和感谢一共有1237页。而这本书才这么点儿——在拳击术语里，这算是“羽量级”的。

那么第一个问题就是这到底是怎么回事呢？这里还真有一个故事可以讲。

多年以来我一直在询问迪士尼动画里的每一个人，期望他们能参与到制作工作中来。我央求了很多导演和管理层人士——老实说，甚至连Mickey都有问过。我想其中的一部分原因是好莱坞以及大银幕的那种魔力。而另一方面则是我不但拥有计算机科学学位，同时还是艺术类的硕士，而制作电影对我个人来讲似乎是可以兼得的。不过我和管理人员说的是我需要一些实际的制作经验来提供有用的工具。身为一名编译器编写者，其实我就是自己最重要的用户之一。当连你自己也对自己的软件感到诸多不满的时候，就不会对那些批评感到抗拒和不公了。

《狂想曲2000》里火鸟组曲的电脑特效指导对我能否加入他的团队颇有兴趣。不过作为一项考验，他要求我先编写一个工具来读取一个场景中迪士尼摄像机的原始资料，并生成一个可以插入到Houdini动画包中的摄像机节点。我用C++实现了这个工具。他们很满意，于是我就加入啦。

一进团队（感谢Jinko和Chyuan），他们就要我用Perl重写这个工具。因为其他的TD都不是职业程序员，他们只会用Perl、Tcl之类的语言。（TD是电影行业里的术语，指的是技术指导。我是部门的软件TD。我们还有一位灯光TD Mira、一位模型TD Tim和好几位电脑特效动画师Mike、Steve和Tonyal。）而且，我还得尽快把它搞完，因为我们要验证一些概念测试，因为导演Paul、Gaetan和特效总监Dave都等着把这个结果交给大老板Peter呢。别急，慢慢来，你懂我的意思的，不过……

这就让我有点犯难了。要是用C++的话我对自己的编程速度还是很自信的。可是我不懂Perl啊。我想，好吧，既然这样我就找本书读一下。不过它不可以太厚了，至少现在不行，我没那么多时间。它最好不用告诉我太多细节，虽然我知道我应该理解所有的内容，不过以后再说吧。毕竟结果才是最重要的：导演需要的是验证概念，艺术家们需要的是可以用来证明概念的插件，而制片人——咳咳，她恨不得一天有48个小时。我不要最好的Perl经典——只要一份能够正确引领我前进，同时不要过分偏离正轨的教程就行了。

我选的是Randal Schwartz的《Learning Perl》。它让我能着手开始工作，而且我要说这本书非常好读。嗯……好吧，任何电脑书都很好读的。它省略了很多重要的东西，不过那个时侯我也不需要了解所有的东西——我只要让我的Perl脚本能运行就可以了。

最终这让我很遗憾地意识到对任何希望学习C++的人来说，《C++ Primer》第3版没有办法

负担起类似的角色。它已经变得太庞大了。虽然我觉得它是一本好书——特别是在第3版里请到Josée Lajoie来合著以后。不过对于这种快速学习C++语言的要求来说，这本书太过繁复了。这就是我决定编写本书的原因。

你可能会想，可是C++不是Perl啊。没错。可是本书也不是《Learning Perl》。它讲的是如何学习C++。真正的问题在于，怎样在删掉将近一千页之后仍然能自称是有价值的教科书呢？

1) **细节程度**。在计算机图形学里，细节程度指的是一幅图像渲染的尖锐程度。屏幕左上角骑在马背上的匈奴人需要能看清眼睛头发的脸、五点钟方向的阴影，以及服饰等细节。而远在后面的匈奴人就不需要关心这种程度的细节了。同样，本书的细节程度也将大幅降低。在我看来，《C++ Primer》在运算符重载上的讨论是目前为止最详细的，可读性也是最强的（我这么说不是自卖自夸，这一节的内容是Josée写的）。但是它花去了整整46页的篇幅来讨论和举例。这里我只用了两页而已。

2) **语言核心**。在《C++ Report》担任编辑的时候，我曾经说过这样一句话，杂志编辑有一半的时间都花在决定哪些内容是要删掉的。本书也是一样。这本书的内容围绕一系列的编程问题展开。并在解决问题的过程中不断引入语言特性。我没有需要动用多重继承或者虚拟继承才能解决的问题，所以这里就不会讨论它们了。不过为了实现迭代器类，我必须引入嵌套类型。类转换操作符很容易被误用，解释起来也相当复杂。所以我决定不在这里讨论它们。这样的例子还有很多。展示语言特性的选择和顺序都是可以批评的。这是我的选择，同时也是我的责任。

3) **范例数量**。《C++ Primer》里包含了几百页详细讲解的范例，包括一个面向对象的文本查询系统和将近半打完整实现的类。虽然本书同样也是围绕代码展开的，但是范例的数量远不及《C++ Primer》的丰富。为了弥补这一点，附录A里提供了所有程序习题的答案。就像我的编辑Deborah Lafferty说的："如果你想教得快一点，唾手可得的答案对于强化学习是很有帮助的。"

本书的结构

本书包含了七章和两个附录。第1章通过编写一个小型的交互程序，给出了预定义语言的描述。它涵盖的内容有内建数据类型、预定义操作符、vector和string类、条件和循环语句，以及负责输入输出的iostream库。我在这一章里引入vector和string类是因为我强烈建议读者使用它们而不是内建数组或是C风格的字符串。

第2章解释了如何设计和使用函数，并介绍了C++里函数的各种风格：内联函数、重载函数、模板函数，以及指向函数的指针。

第3章涵盖了所谓的标准模板库（STL）：一组容器类，例如vector、list、set和map，以及作用于那些容器上的泛型算法，例如`sort()`、`copy()`和`merge()`。附录B按照字母顺序列出了最常用的泛型算法并就如何使用给出了范例。

身为一名C++程序员，你的主要工作就是编写类以及面向对象的类层次体系。第4章介绍了

C++类机制的设计和使用，它让你能创建针对应用程序领域的数据类型。例如我在梦工厂担任咨询工作的时候，我们就设计了可以进行四通道图像合成等工作的类。第5章解释了如何扩展类的设计，在面向对象的类层次体系里支持一系列相关的类。例如，我们通过继承和动态绑定定义了一个合成所需的层次体系，而不是八个各自孤立的图像合成类。

类模板是第6章的主题。类模板是一种对类的描述，它允许你参数化一个或多个类型或者值。例如vector类就可以参数化它要包含的元素类型。而buffer类则不但可以参数化其元素类型，还可以参数化它的大小。这一章将围绕一个二叉树模板类的实现展开。

最后，第7章展示了如何使用C++的异常处理机制以及如何让它融入到现有的标准库异常体系中去。附录A给出了编程习题的答案。而附录B则为最常见的泛型算法提供了代码范例和讨论。

关于源代码

本书给出的完整源代码以及习题答案都可以在Addison Wesley Longman的网站（http://www.informit.com/store/product.aspx?isbn=0201485184）上下载。所有的代码都在Visual C++ 5.0（Intel C++编译器）和Visual C++ 6.0（微软C++编译器）下通过测试。你在自己的系统上编译的时候可能会需要稍微做一点修改。如果是这样，请把你做的修改发给我(slippman@objectwrite.com)，我会把它们和你的名字一起加在答案代码里。(注意本书并没有展示完整的源代码。)

感谢

这里要特别感谢《C++ Primer》第3版的合作者Josée Lajoie。一直以来她都给予了极大的支持，不仅是因为她对本书草稿所给出的深刻评注，更是因为她毫无保留的鼓励。同时我还要特别感谢Dave Slayton，他以一支尖锐的绿色铅笔审阅了本书的全部内容和代码范例，以及Steve Vinoski对本书草稿极富爱心但又坚决的批评。

特别感谢Addison-Wesley的编辑团队：编辑Deborah Lafferty从一开始就支持着这个项目，审稿编辑Betsy Hardinger对本书的可读性作出了相当大的贡献，而产品经理John Fuller则领导我们把手稿变成装订成册的书本。

在撰写此书的过程中，我还有一些独立咨询顾问的工作，同时肩负《Essential C++》和一些能够（合理）体谅我的客户。所以我要感谢Colin Lipworth、Edwin Leonard以及Kenneth Meyer的耐心和信赖。

寻找更多的信息

这里我要厚着脸皮地说，介绍C++语言最好的两本书分别是Lippman和Lajoie合著的《C++

Primer》以及Stroustrup的《The C++ Programming Language》，两本书都有第3版[⊖]。在本书中我会就更深入的主题向你推荐它们来阅读。下面是本书中引用到的书目。(更详细的参考资料可以在《C++ Primer》和《The C++ Programming Language》里找到。)

[LIPPMAN98] Lippman, Stanley, and Josée Lajoie, C++ *Primer, 3rd Edition*, Addison Wesley Longman, Inc., Reading, MA (1998) ISBN 0-201-82470-1.

[LIPPMAN96a] Lippman, Stanley, *Inside the* C++ *Object Model*, Addison Wesley Longman, Inc., Reading, MA (1996) ISBN 0-201-83454-5.

[LIPPMAN96b] Lippman, Stanley, Editor, C++ *Gems*, a SIGS Books imprint, Cambridge University Press, Cambridge, England (1996) ISBN 0-13570581-9.

[STROUSTRUP97] Stroustrup, Bjarne, *The* C++ *Programming Language, 3rd Edition*, Addison Wesley Longman, Inc., Reading, MA (1997) ISBN 0-201-88954-4.

[SUTTER99] Sutter, Herb, *Exceptional* C++, Addison Wesley Longman, Inc., Reading, MA (2000) ISBN 0-201-61562-2.

排版约定

本书的文字字体设为10.5 pt. Palatino。程序文字和语言关键字则设为8.5 pt. `lucida`。函数定义为它们的名字加上C++函数调用操作符（`()`）。因此，`foo`表示的是一个程序对象，而`bar()`表示的就是一个程序函数了。类名的字体也同样设为Palatino。

⊖ 《C++程序设计语言》在国内有中文译本。另外，Bjarne Stroustrup还有另一本以C++为主题的新作(http://stroustrup.com/Programming。中文版正在翻译中)，这本是针对新人的教科书。另外，本书读者群并非彻底的编程初学者，而是具备一定编程经验，希望快速上手开始进行C++编程的程序员。——译者注

Contents

1

Basic C++ Programming

这一章介绍编写一个C++程序的基本要素，包括基本类型、各种运算、各种控制语句、基本的复合类型、指针、数组以及最常见的数据结构等。

In this chapter, we evolve a small program to exercise the fundamental components of the C++ language. These components consist of the following:

1. A small set of data types: Boolean, character, integer, and floating point.
2. A set of arithmetic, relational, and logical operators to manipulate these types. These include not only the usual suspects, such as addition, equality, less than, and assignment, but also the less conventional increment, conditional, and compound assignment operators.
3. A set of conditional branch and looping statements, such as the `if` statement and `while` loop, to alter the control flow of our program.
4. A small number of compound types, such as a pointer and an array. These allow us, respectively, to refer indirectly to an existing object and to define a collection of elements of a single type.
5. A standard library of common programming abstractions, such as a string and a vector.

1.1 How to Write a C++ Program

We've been asked to write a simple program to write a message to the user's terminal asking her to type in her name. Then we read the name she enters, store the name so that we can use it later, and, finally, greet the user by name.

OK, so where do we start? We start in the same place every C++ program starts — in a function called `main()`. `main()` is a user-implemented function of the following general form:

```
int main()
{
   // our program code goes here
}
```

见书后第1章注释1

`int` is a C++ language keyword. *Keywords* are predefined names given special meaning within the language. `int` represents a built-in integer data type. (I have much more to say about data types in the next section.)

A *function* is an independent code sequence that performs some computation. It consists of four parts: the return type, the function name, the parameter list, and the function body. Let's briefly look at each part in turn.

return 0; 表示程序正确返回，非零值表示出错。这是惯例。

The *return type* of the function usually represents the result of the computation. `main()` has an integer return type. The value returned by `main()` indicates whether our program is successful. By convention, `main()` returns 0 to indicate success. A nonzero return value indicates something went wrong.

函数名和变量名都可以任意取，只是不能使用C++关键字而已。因此有时候也称它们为保留字。

The *name* of a function is chosen by the programmer and ideally should give some sense of what the function does. `min()` and `sort()`, for example, are pretty good function names. `f()` and `g()` are not as good. Why? Because they are less informative as to what the functions do.

`main` is not a language keyword. The compilation system that executes our C++ programs, however, expects a `main()` function to be defined. If we forget to provide one, our program will not run.

见书后第1章注释2

The *parameter list* of a function is enclosed in parentheses and is placed after the name of the function. An empty parameter list, such as that of `main()`, indicates that the function accepts no parameters.

The parameter list is typically a comma-separated list of types that the user can pass to the function when the function is executed. (We say that the user has *called*, or *invoked*, a function.) For example, if we write a function `min()` to return the smaller of two values, its parameter list would identify the types of the two values we want to compare. A `min()` function to compare two integer values might be defined as follows:

```
int min( int val1, int val2 )
{
   // the program code goes here ...
}
```

The *body* of the function is enclosed in curly braces (`{}`). It holds the code sequence that provides the computation of the function. The double forward slash (`//`) represents a comment, a programmer's annotation on some aspect of the code. It is intended for readers of the program and is discarded during compilation. Everything following the double forward slash to the end of the line is treated as a comment.

输入输出是一个很大的话题，足以单独写一本书。不过在这里我们暂时不关心其中的细节。

Our first task is to write a message to the user's terminal. Input and output are not a predefined part of the C++ language. Rather, they are supported by an *object-oriented class hierarchy* implemented in C++ and provided as part of the C++ standard library.

有了类机制，编程语言的表达能力就能大大增强。稍后章节里会详细讲解。

A class is a user-defined data type. The class mechanism is a method of adding to the data types recognized by our program. An object-oriented class hierarchy defines a family of related class types, such as terminal and file input, terminal and file output, and so on. (We have a lot more to say about classes and object-oriented programming throughout this text.)

C++ predefines a small set of fundamental data types: Boolean, character, integer, and floating point. Although these provide a foundation for all our programming, they are not the focus of our programs. A camera, for example, must have a location in space, which is generally represented by three floating point numbers. A camera also has a viewing orientation, which is also represented by three floating point numbers. There is usually an aspect ratio describing the ratio of the camera viewing width to height. This is represented by a single floating point number.

On the most primitive level, that is, a camera is represented as seven floating point numbers, six of which form two x,y,z coordinate tuples. Programming at this low level requires that we shift our thinking back and forth from the manipulation of the camera abstraction to the corresponding manipulation of the seven floating point values that represent the camera in our program.

The class mechanism allows us to add layers of type abstraction to our programs. For example, we can define a Point3d class to represent location and orientation in space. Similarly, we can define a Camera class containing two Point3d class objects and a floating point value. We're still representing a camera by seven floating point values. The difference is that in our programming we are now directly manipulating the Camera class rather than seven floating point values.

见书后第1章注释3

The definition of a class is typically broken into two parts, each represented by a separate file: a *header file* that provides a declaration of the operations supported by the class, and a *program text file* that contains the implementation of those operations.

见书后第1章注释4

To use a class, we include its header file within our program. The header file makes the class known to the program. The standard C++ input/output library is called the *iostream* library. It consists of a collection of related classes supporting input and output to the user's terminal and to files. To use the iostream class library, we must include its associated header file:

```
#include <iostream>
```

To write to the user's terminal, we use a predefined class object named `cout` (pronounced *see out*). We direct the data we wish `cout` to write using the output operator (<<), as follows:

```
cout << "Please enter your first name: ";
```

This represents a C++ program *statement*, the smallest independent unit of a C++ program. It is analogous to a sentence in a natural language. A statement is terminated by a semicolon. Our output statement writes the string literal (marked by double quotation marks) onto the user's terminal. The quotation marks identify the string; they are not displayed on the terminal. The user sees

```
Please enter your first name:
```

Our next task is to read the user's input. Before we can read the name the user types, we must define an object in which to store the information. We define an object by specifying the data type of the object and giving it a name. We've already seen one data type: `int`. That's hardly a useful way of storing someone's name, however! A more appropriate data type in this case is the standard library string class:

```
string user_name;
```

This defines `user_name` as an object of the string class. The definition, oddly enough, is called a *declaration statement*. This statement won't be accepted, however, unless we first make the string class known to the program. We do this by including the string class header file:

```
#include <string>
```

To read input from the user's terminal, we use a predefined class object named `cin` (pronounced *see in*). We use the input operator (`>>`) to direct `cin` to read data from the user's terminal into an object of the appropriate type:

```
cin >> user_name;
```

The output and input sequence would appear as follows on the user's terminal. (The user's input is highlighted in bold.)

```
Please enter your first name: anna
```

All we've left to do now is to greet the user by name. We want our output to look like this:

```
Hello, anna ... and goodbye!
```

I know, that's not much of a greeting. Still, this is only the first chapter. We'll get a bit more inventive before the end of the book.

To generate our greeting , our first step is to advance the output to the next line. We do this by writing a newline character literal to `cout`:

见书后第1章注释5

```
cout << '\n';
```

A character literal is marked by a pair of single quotation marks. There are two primary flavors of character literals: printing characters such as the alphabet (`'a'`, `'A'`, and so on), numbers, and punctuation marks (`';'`, `'-'`, and so on), and nonprinting characters such as a newline (`'\n'`) or tab (`'\t'`). Because there is no literal representation of nonprinting characters, the most common instances, such as the newline and tab, are represented by special two-character sequences.

Now that we've advanced to the next line, we want to generate our `Hello`:

```
cout << "Hello, ";
```

Next, we need to output the name of the user. That's stored in our string object, `user_name`. How do we do that? Just the same as with the other types:

```
cout << user_name;
```

Finally, we finish our greeting by saying goodbye (notice that a string literal can be made up of both printing and nonprinting characters):

```
cout << " ... and goodbye!\n";
```

In general, all the built-in types are output in the same way — that is, by placing the value on the right-hand side of the output operator. For example,

```
cout << "3 + 4 = ";
cout << 3 + 4;
cout << '\n';
```

generates the following output:

```
3 + 4 = 7
```

As we define new class types for use in our applications, we also provide an instance of the output operator for each class. (We see how to do this in Chapter 4.) This allows users of our class to output individual class objects in exactly the same way as the built-in types.

Rather than write successive output statements on separate lines, we can concatenate them into one compound output statement:

```
cout << '\n'
     << "Hello, "
     << user_name
     << " ... and goodbye!\n";
```

Finally, we can explicitly end `main()` with the use of a return statement:

```
return 0;
```

其他任何带有返回值的函数都必须在退出时明确返回一个值，main()函数特殊的地方就在于即使省略了return 0;语句也不要紧，编译器只会给出警告而已。正如下面注解所说，编译器会自动为你插入这条语句。

`return` is a C++ keyword. The expression following `return`, in this case 0, represents the result value of the function. Recall that a return value of 0 from `main()` indicates that the program has executed successfully.[1]

Putting the pieces together, here is our first complete C++ program:

```
#include <iostream>
#include <string>
using namespace std; // haven't explained this yet ...

int main()
{
      string user_name;
      cout << "Please enter your first name: ";
      cin >> user_name;
      cout << '\n'
           << "Hello, "
```

[1] If we don't place an explicit `return` statement at the end of `main()`, a `return 0;` statement is inserted automatically. In the program examples in this book, I do not place an explicit `return` statement.

```
              << user_name
              << " ... and goodbye!\n";

         return 0;
    }
```

When compiled and executed, this code produces the following output (my input is highlighted in bold):

```
Please enter your first name: anna
Hello, anna ... and goodbye!
```

There is one statement I haven't explained:

```
using namespace std;
```

见书后第1章注释6

Let's see if I can explain this without scaring you off. (A deep breath is recommended at this point!) Both `using` and `namespace` are C++ keywords. `std` is the name of the standard library namespace. Everything provided within the standard library (such as the string class and the iostream class objects `cout` and `cin`) is encapsulated within the `std` namespace. Of course, your next question is, what is a namespace?

见书后第1章注释7

A *namespace* is a method of packaging library names so that they can be introduced within a user's program environment without also introducing name clashes. (A *name clash* occurs when there are two entities that have the same name in an application so that the program cannot distinguish between the two. When this happens, the program cannot run until the name clash is resolved.) Namespaces are a way of fencing in the visibility of names.

To use the string class and the iostream class objects `cin` and `cout` within our program, we must not only include the string and iostream header files but also make the names within the `std` namespace visible. The *using directive*

```
using namespace std;
```

is the simplest method of making names within a namespace visible. (To read about namespaces in more detail, check out either Section 8.5 of [LIPPMAN98] or Section 8.2 of [STROUSTRUP97].)

所有的习题解答请参考附录A。

Exercise 1.1

Enter the `main()` program, shown earlier. Either type it in directly or download the program; see the Preface for how to acquire the source programs and solutions to exercises. Compile and execute the program on your system.

Exercise 1.2

Comment out the string header file:

```
// #include <string>
```

Now recompile the program. What happens? Now restore the string header and comment out

```
    //using namespace std;
```

What happens?

Exercise 1.3

Change the name of `main()` to `my_main()` and recompile the program. What happens?

Exercise 1.4

Try to extend the program: (1) Ask the user to enter both a first and last name and (2) modify the output to write out both names.

1.2 Defining and Initializing a Data Object

Now that we have the user's attention, let's challenge her to a quiz. We display two numbers representing a numerical sequence and then request our user to identify the next value in the sequence. For example,

```
    The values 2,3 form two consecutive
        elements of a numerical sequence.
    What is the next value?
```

These values are the third and fourth elements of the Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, and so on. A Fibonacci sequence begins with the first two elements set to 1. Each subsequent element is the sum of its two preceding elements. (In Chapter 2 we write a function to calculate the elements.)

If the user enters 5, we congratulate her and ask whether she would like to try another numerical sequence. Any other entered value is incorrect, and we ask the user whether she would like to guess again.

To add interest to the program, we keep a running score based on the number of correct answers divided by the number of guesses.

Our program needs at least five objects: the string class object to hold the name of the user; three integer objects to hold, in turn, the user's guess, the number of guesses, and the number of correct guesses; and a floating point object to hold the user's score.

To define a data object, we must both name it and provide it with a data type. The name can be any combination of letters, numbers, and the underscore. Letters are case-sensitive. Each one of the names `user_name`, `User_name`, `uSeR_nAmE`, and `user_Name` refers to a distinct object.

A name cannot begin with a number. For example, `1_name` is illegal but `name_1` is OK. Also, a name must not match a language keyword exactly. For example, `delete` is a language keyword, and so we can't use it for an entity in our program. (This explains

名字必须是以字母开头，下划线也行，不过从编程风格上来讲，下划线表示的是特殊用途（例如留给库的编写者），而且程序里出现太多带有下划线的名字会大大降低可读性，所以一般都避免使用下划线开头的名字。