Basic Input/Output

The example programs of the previous sections provided little interaction with the user, if any at all. They simply printed simple values on screen, but the standard library provides many additional ways to interact with the user via its input/output features. This section will present a short introduction to some of the most useful.

前面几节的示例程序几乎没有与用户进行交互。它们只是在屏幕上打印简单的值，但是标准库通过其输入/输出特性提供了许多其他与用户交互的方法。本节将简要介绍一些最有用的方法。

C++ uses a convenient abstraction called *streams* to perform input and output operations in sequential media such as the screen, the keyboard or a file. A stream is an entity where a program can either insert or extract characters to/from. There is no need to know details about the media associated to the stream or any of its internal specifications. All we need to know is that streams are a source/destination of characters, and that these characters are provided/accepted sequentially (i.e., one after another).

c++使用一种叫做流的方便抽象来执行连续媒体(如屏幕、键盘或文件)的输入和输出操作。流是一个实体，程序可以在其中插入或提取字符。不需要知道与流关联的媒体或其任何内部规范的细节。我们只需要知道流是字符的源/目标，并且这些字符是按顺序提供/接受的(即，一个接一个)。

The standard library defines a handful of stream objects that can be used to access what are considered the standard sources and destinations of characters by the environment where the program runs:

标准库定义了一些流对象，可以用来访问程序运行的环境中被认为是标准源和字符目标的内容:

stream	description
`cin`	standard input stream
`cout`	standard output stream
`cerr`	standard error (output) stream
`clog`	standard logging (output) stream

We are going to see in more detail only `cout` and `cin` (the standard output and input streams); `cerr` and `clog` are also output streams, so they essentially work like `cout`, with the only difference being that they identify streams for specific purposes: error messages and logging; which, in many cases, in most environment setups, they actually do the exact same thing: they print on screen, although they can also be individually redirected.

我们只会更详细地了解' cout '和' cin '(标准输出和输入流);' cerr '和' clog '也是输出流，所以它们本质上像' cout '一样工作，唯一的区别是它们为特定的目的识别流:错误消息和日志;在很多情况下，在大多数环境设置中，它们实际上做的是相同的事情:它们在屏幕上打印，尽管它们也可以单独重定向。

Standard output (cout)

On most program environments, the standard output by default is the screen, and the C++ stream object defined to access it is `cout`.

在大多数程序环境中，默认的标准输出是屏幕，而定义来访问它的c++流对象是' cout '。

For formatted output operations, `cout` is used together with the *insertion operator*, which is written as `<<` (i.e., two "less than" signs).

对于格式化输出操作，' cout '与*插入操作符*一起使用，插入操作符*写成' << '(即两个"小于"符号)。

```
cout << "Output sentence"; // prints Output sentence on screen
cout << 120;               // prints number 120 on screen
cout << x;                 // prints the value of x on screen
```

The `<<` operator inserts the data that follows it into the stream that precedes it. In the examples above, it inserted the literal string `Output sentence`, the number `120`, and the value of variable `x` into the standard output stream `cout`. Notice that the sentence in the first statement is enclosed in double quotes (`"`) because it is a string literal, while in the last one, `x` is not. The double quoting is what makes the difference; when the text is enclosed between them, the text is printed literally; when they are not, the text is interpreted as the identifier of a variable, and its value is printed instead. For example, these two sentences have very different results:

' << '操作符将后面的数据插入到前面的流中。在上面的例子中，它将字符串' Output sentence '、数字' 120 '和变量' x '的值插入到标准输出流' cout '中。注意，第一个语句中的句子是用双引号括起来的(' " ')，因为它是字符串字面量，而在最后一个语句中，' x '不是。双引号是区别所在;当文本在它们之间被包围时，文本将按字面意思打印;如果不是，则将文本解释为变量的标识符，并打印变量的值。例如，这两个句子有非常不同的结果:

```
cout << "Hello";  // prints Hello
cout << Hello;    // prints the content of variable Hello
```

Multiple insertion operations (<<) may be chained in a single statement:

多个插入操作(<<)可以链接在一个语句中:

```
cout << "This " << " is a " << "single C++ statement";
```

This last statement would print the text `This is a single C++ statement`. Chaining insertions is especially useful to mix literals and variables in a single statement:

最后一条语句将输出文本“This is a single c++ statement”。链接插入对于在单个语句中混合文字和变量特别有用:

```
cout << "I am " << age << " years old and my zipcode is " << zipcode;
```

Assuming the age variable contains the value 24 and the zipcode variable contains 90064, the output of the previous statement would be:

假设年龄变量包含值24，邮政编码变量包含90064，前面语句的输出将是:

```
I am 24 years old and my zipcode is 90064
```

What cout does not do automatically is add line breaks at the end, unless instructed to do so. For example, take the following two statements inserting into `cout`:

cout不会自动做的是在末尾添加换行符，除非有命令这样做。例如，下面两个语句插入' cout ':

```
cout << "This is a sentence.";
cout << "This is another sentence.";
```

The output would be in a single line, without any line breaks in between. Something like:

输出将在一行中，中间没有任何换行符。就像这样:

```
This is a sentence.This is another sentence.
```

To insert a line break, a new-line character shall be inserted at the exact position the line should be broken. In C++, a new-line character can be specified as `\n` (i.e., a backslash character followed by a lowercase `n`). For example:

要插入换行符，必须在应该换行的确切位置插入新行字符。在c++中，一个新行字符可以指定为' \n '(即，一个反斜杠字符后跟一个小写的' n ')。例如:

```
cout << "First sentence.\n";
cout << "Second sentence.\nThird sentence.";
```

This produces the following output:

这个程序会这样输出

`First sentence.`

`Second sentence.`

`Third sentence.`

Alternatively, the `endl` manipulator can also be used to break lines. For example:

或者，' endl '操纵器也可以用来断行。例如:

```
cout << "First sentence." << endl;
cout << "Second sentence." << endl;
```

This would print:

`First sentence.`

`Second sentence.`
The `endl` manipulator produces a newline character, exactly as the insertion of `'\n'` does; but it also has an additional behavior: the stream's buffer (if any) is flushed, which means that the output is requested to be physically written to the device, if it wasn't already. This affects mainly *fully buffered* streams, and `cout` is (generally) not a *fully buffered*stream. Still, it is generally a good idea to use `endl` only when flushing the stream would be a feature and `'\n'` when it would not. Bear in mind that a flushing operation incurs a certain overhead, and on some devices it may produce a delay.

Standard input (cin)

In most program environments, the standard input by default is the keyboard, and the C++ stream object defined to access it is `cin`.

在大多数程序环境中，默认的标准输入是键盘，而定义来访问它的c++流对象是' cin '。

For formatted input operations, `cin` is used together with the extraction operator, which is written as `>>` (i.e., two "greater than" signs). This operator is then followed by the variable where the extracted data is stored. For example:

对于格式化的输入操作，' cin '与提取操作符一起使用，提取操作符写成' >> '(即两个大于号)。该操作符后面跟着存储提取数据的变量。例如:

```
int age;
cin >> age;
```

The first statement declares a variable of type `int` called `age`, and the second extracts from `cin` a value to be stored in it. This operation makes the program wait for input from `cin`; generally, this means that the program will wait for the user to enter some sequence with the keyboard. In this case, note that the characters introduced using the keyboard are only transmitted to the program when the ENTER (or RETURN) key is pressed. Once the statement with the extraction operation on `cin` is reached, the program will wait for as long as needed until some input is introduced.

第一个语句声明了一个名为“age”的类型为“int”的变量，第二个语句从“cin”中提取一个值存储在其中。这个操作使程序等待来自' cin '的输入;通常，这意味着程序将等待用户用键盘输入某个序列。在本例中，请注意，只有当按下ENTER(或RETURN)键时，使用键盘引入的字符才会传输到程序。一旦到达对' cin '进行提取操作的语句，程序将等待所需的时间，直到引入一些输入。

The extraction operation on `cin` uses the type of the variable after the `>>` operator to determine how it interprets the characters read from the input; if it is an integer, the format expected is a series of digits, if a string a sequence of characters, etc.

对' cin '的提取操作使用' >> '操作符后的变量类型，以确定它如何解释从输入读取的字符;如果它是一个整数，则期望的格式是一系列数字，如果字符串是一系列字符，等等。

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

As you can see, extracting from `cin` seems to make the task of getting input from the standard input pretty simple and straightforward. But this method also has a big drawback. What happens in the example above if the user enters something else that cannot be interpreted as an integer? Well, in this case, the extraction operation fails. And this, by default, lets the program continue without setting a value for variable `i`, producing undetermined results if the value of `i` is used later.

如您所见，从' cin '提取似乎使从标准输入获取输入的任务变得非常简单和直接。但这种方法也有很大的缺点。在上面的例子中，如果用户输入了其他不能被解释为整数的东西，会发生什么?在这种情况下，营救行动失败了。默认情况下，这样程序就可以在不为变量i设置值的情况下继续执行，如果以后使用变量i的值，就会产生不确定的结果。

This is very poor program behavior. Most programs are expected to behave in an expected manner no matter what the user types, handling invalid values appropriately. Only very simple programs should rely on values extracted directly from `cin` without further checking. A little later we will see how *stringstreams* can be used to have better control over user input.
Extractions on `cin` can also be chained to request more than one datum in a single statement:

这是非常糟糕的程序行为。不管用户类型是什么，大多数程序都按照预期的方式运行，并适当地处理无效值。只有非常简单的程序才应该依赖于直接从' cin '中提取的值，而不需要进一步检查。稍后，我们将看到如何使用*stringstreams*更好地控制用户输入。
对' cin '的提取也可以被链接起来，以在单个语句中请求多个数据:

```
cin >> a >> b;
```

This is equivalent to:

```
cin >> a;
cin >> b;
```

In both cases, the user is expected to introduce two values, one for variable `a`, and another for variable `b`. Any kind of space is used to separate two consecutive input operations; this may either be a space, a tab, or a new-line character.

在这两种情况下，用户都需要引入两个值，一个用于变量' a '，另一个用于变量' b '。任意一种空间用于分隔两个连续的输入操作;这可以是空格、制表符或换行符。

cin and strings

The extraction operator can be used on `cin` to get strings of characters in the same way as with fundamental data types:

提取操作符可以用于' cin '获取字符串，方法与基本数据类型相同:

```
string mystring;
cin >> mystring;
```

However, `cin` extraction always considers spaces (whitespaces, tabs, new-line...) as terminating the value being extracted, and thus extracting a string means to always extract a single word, not a phrase or an entire sentence.

然而，' cin '提取总是将空格(空格、制表符、换行符……)视为所提取值的结束，因此提取字符串意味着始终提取单个单词，而不是短语或整个句子。

To get an entire line from `cin`, there exists a function, called `getline`, that takes the stream (`cin`) as first argument, and the string variable as second. For example:

要从' cin '获取整行，存在一个名为' getline '的函数，它将流(' cin ')作为第一个参数，将字符串变量作为第二个参数。例如:

```
// cin with strings
#include <iostream>
#include <string>
using namespace std;
```

```
int main ()
{
  string mystr;
  cout << "What's your name? ";
  getline (cin, mystr);
  cout << "Hello " << mystr << ".\n";
  cout << "What is your favorite team? ";
  getline (cin, mystr);
  cout << "I like " << mystr << " too!\n";
  return 0;
}
```

Notice how in both calls to `getline`, we used the same string identifier (`mystr`). What the program does in the second call is simply replace the previous content with the new one that is introduced.

注意，在对' getline '的两次调用中，我们使用了相同的字符串标识符(' mystr ')。程序在第二个调用中所做的只是简单地用引入的新内容替换先前的内容。

The standard behavior that most users expect from a console program is that each time the program queries the user for input, the user introduces the field, and then presses ENTER (or RETURN). That is to say, input is generally expected to happen in terms of lines on console programs, and this can be achieved by using `getline` to obtain input from the user. Therefore, unless you have a strong reason not to, you should always use `getline` to get input in your console programs instead of extracting from `cin`.

大多数用户期望控制台程序的标准行为是，每当程序向用户查询输入时，用户引入字段，然后按ENTER(或RETURN)。也就是说，在控制台程序中，输入通常是按行进行的，这可以通过使用“getline”获取用户的输入来实现。因此，除非你有很强的理由不这样做，否则你应该总是使用' getline '在控制台程序中获取输入，而不是从' cin '提取。

stringstream

The standard header `<sstream>` defines a type called `stringstream` that allows a string to be treated as a stream, and thus allowing extraction or insertion operations from/to strings in the same way as they are performed on `cin` and `cout`. This feature is most useful to convert strings to numerical values and vice versa. For example, in order to extract an integer from a string we can write:

标准头文件' '定义了一个名为' stringstream '的类型，该类型允许将字符串作为流处理，因此允许提取或插入字符串操作，就像对' cin '和' cout '执行操作一样。这个特性对于将字符串转换为数值非常有用，反之亦然。例如，为了从字符串中转换整数，我们可以这样写:

```
string mystr ("1204");
int myint;
stringstream(mystr) >> myint;
```

This declares a `string` with initialized to a value of `"1204"`, and a variable of type `int`. Then, the third line uses this variable to extract from a `stringstream` constructed from the string. This piece of code stores the numerical value `1204` in the variable called `myint`.

这声明了一个初始化为值为1204的字符串和一个类型为int的变量。然后，第三行使用这个变量从字符串构造的' stringstream '中提取。这段代码将数值' 1204 '存储在名为' myint '的变量中。

```
// stringstreams
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main ()
{
  string mystr;
  float price=0;
  int quantity=0;

  cout << "Enter price: ";
  getline (cin,mystr);
  stringstream(mystr) >> price;
  cout << "Enter quantity: ";
  getline (cin,mystr);
  stringstream(mystr) >> quantity;
  cout << "Total price: " << price*quantity << endl;
  return 0;
}
```

In this example, we acquire numeric values from the *standard input* indirectly: Instead of extracting numeric values directly from `cin`, we get lines from it into a string object (`mystr`), and then we extract the values from this string into the variables `price` and `quantity`. Once these are numerical values, arithmetic operations can be performed on them, such as multiplying them to obtain a total price.

在这个例子中，我们间接地从标准输入中获取数值:我们不是直接从cin中提取数值，而是从cin中提取行到一个字符串对象(mystr)中，然后从这个字符串中提取值到变量price和quantity中。一旦这些值是数值，就可以对它们执行算术运算，例如将它们相乘以获得总价。

With this approach of getting entire lines and extracting their contents, we separate the process of getting user input from its interpretation as data, allowing the input process to be what the user expects, and at the same time gaining more control over the transformation of its content into useful data by the program.

使用这种方法的整个线路和提取它们的内容,我们分开的过程从其解释用户输入数据,允许用户输入过程预计,同时获得更多的控制其内容转换成有用的数据的程序。