本章小结

在本章，你已经学习了如何使用 < string > 类库，利用该类库你可以编写出字符串操作函数，并且不用担心底层表示的细节问题。本章的重点包括：

- < string > 类库提供了一个string类，它用于代表一个字符序列。尽管Cpp语言也包括一个更原始的类型去维护对于C语言的兼容性，但在编写程序时最好使用string类。
- 如果你使用>>提取操作符读取字符串数据，输入将会在第一个空白字符处停止。如果你想从用户那里读取一整行文本，使用C + + 标准库提供的getline函数会更好。
- string类中提供的最常用方法显示在表3-1中。因为string是一个类，方法都使用接收方语法来代替传统的函数形式。因此，为了获取存储在str变量中字符串的长度，你需要调用str.length () 。
- ·string类提供的几种方法破坏性地修改了接收字符串。给予用户自由的权限去使用这些能够改变一个对象内部状态的方法，这使得对象的完整性更难以保护。因此，本书中的程序将最大限度地减少这些方法的使用。
- **string类使用操作符重载以简化许多常见的字符串操作。对于字符串来说，最重要的操作符是 + （连接）操作符、 [] （选择）操作符和关系操作符。**
- 循环遍历一个字符串中字符的标准模式是：

```
for (int i = 0;i < str.length();i++) {
    body of loop that manipulates str[i]...
}

for (auto i : str) {
    body of loop that manipulates str[i]...
}
```

- < cctype > 库提供了几种处理单个字符的函数。其中最重要的函数显示在表3-2中。
- 通过连接以增长一个字符串的标准模式是：

```
string str="";

for (whatever loop header line fits the application){
    str += the next substring or character;
}
```

字符串的常见操作与cctype库（对单个字符的操作）

表 3-1 <string> 库中常见的方法

字符串操作	
str_1 + str_2	连接字符串 str_1 和 str_2，返回一个连接后的新字符串。其中 str_1 或 str_2 可以替换为字符类型，但不允许替换为数字类型
str += str_2	将字符串 str_2 的拷贝添加到 str 的末尾。C++ 语言重载了这个操作符，使得 str_2 可以为字符类型
str_1 == str_2 str_1 != str_2 str_1 < str_2 str_1 <= str_2 str_1 > str_2 str_1 >= str_2	这些操作符用于比较字符串 str_1 和 str_2。比较标准参照**字典序**（lexicographic order），字典序由字符 ASCII 码值定义
str[k]	返回字符串 str 索引位置 k 上的字符，[] 操作符并不检测 k 是否在其合法的范围内
读字符串内容方法	
str.**length**()	返回字符串 str 中字符的个数
str.**at**(k)	返回字符串 str 中索引位置 k 的字符。与 [] 操作符相反，如果 k 超出了其合法值范围，则 at 方法会产生异常
str.**substr**(pos, n)	返回一个新的字符串，该字符串是从 str 的 pos 位置开始，包含 n 个字符或直到 str 字符串末尾的子串。该方法的第二个参数是可选的，若无参数 n，子串总是会延伸至 str 字符串的末尾
str.**compare**(str_2)	比较接收方字符串 str 和 str_2，若两个字符串相等，则返回一个整数 0；如果 str 字典序在 str_2 之前，则返回一个负数；如果 str 字典序在 str_2 之后，返回一个正数。因为 C++ 重载了关系操作符，因此，程序员很少明确地调用 compare 方法
str.**find**($pattern$, pos)	在接收方字符串 str 中，从 pos 位置开始查找 $pattern$ 子串，若找到，则函数返回 $pattern$（可以是一个字符或一个字符串）所出现的第一个索引值；如果 $pattern$ 没有出现，find 返回常量 string::npos。方法的第 2 个参数是可选的；如果不提供第 2 个参数，则 find 方法会从字符串头开始搜索
修改接收方字符串内容方法	
str.**erase**(pos, n)	从 str 的 pos 处开始向后删除 n 个字符
str.**insert**(pos, str_2)	从 str 的 pos 处开始插入 str_2 的拷贝
str.**replace**(pos, n, str_2)	以 str_2 替换 str 中从 pos 处开始的 n 个字符
用 C++ 创建 C 风格的字符串方法	
string($carray$)	返回一个与 $carray$ 字符串相同字符的 C++ 字符串
string(n, ch)	返回一个包含 n 个 ch 字符的 C++ 字符串
str.**c_str**()	返回一个与 str 内容同样的 C 风格字符数组

提取字符串中的子串

在连接较短字符串以形成更长的字符串运算后，你经常需要先做相反的工作：将一个字符串分解成数个短片段。一个较长字符串的一部分称为其子串（substring）。

string类提供了一个带有两个参数的方法substr，**其中第一个参数为提取字符串子串的起始位置，第二个参数为欲提取的子串字符个数。**

- 调用函数str.substr (start, n) ，将从start指定的索引位置开始在str中提取一个n个字符的子串。例如，如果str包含字符串 " hello, world”，调用如下方法：

  `str.substr(1,3)`

  将返回包含3个字符的子串“ell”。由于C++中的下标从0开始，在索引位置1处为字符`e`。

- 在substr方法中第二个参数是可选的，如果第二个参数省略，substr返回一个从指定位置开始持续到该字符串结尾的子串。因此，如以下调用方法：

  `str.substr(7)`

  将返回字符串 " world " 。

- 类似地，如果n是已知的，但指定的开始位置后面少于n个字符，substr只返回原始字符串指定位置到字符串结尾的子串。

下面的函数调用将返回参数str的后半段子串，如果str的长度为奇数，则其子串将包含最中间的字符。

```
string secondHalf(string str) {
    return str.substr(str.length() / 2);
}
```

```
#include <iostream>
#include <string>

int main()
{
    std::string a = "0123456789abcdefghij";

    // count is npos, returns [pos, size())
    std::string sub1 = a.substr(10);
    std::cout << sub1 << '\n';

    // both pos and pos + count are within bounds, returns [pos, pos + count)
    std::string sub2 = a.substr(5, 3);
    std::cout << sub2 << '\n';

    // pos is within bounds, pos + count is not, returns [pos, size())
    std::string sub4 = a.substr(a.size() - 3, 50);
    // this is effectively equivalent to
    // std::string sub4 = a.substr(17, 3);
    // since a.size() == 20, pos == a.size() - 3 == 17, and a.size() - pos == 3

    std::cout << sub4 << '\n';

    try
    {
        // pos is out of bounds, throws
        std::string sub5 = a.substr(a.size() + 3, 50);
        std::cout << sub5 << '\n';
    }
    catch (const std::out_of_range& ex)
    {
```

```
        std::cout << ex.what() << '\n';
    }
}
```

可能的输出如下：

```
abcdefghij
567
hij
basic_string::substr: __pos (which is 23) > this→size() (which is 20)
```

`cctype`: C Character Type Library.

由于字符串是由字符组成的，处理字符串中的单个字符而不是整个字符串的操作就显得非常必要。

`<cctype>`库提供了处理字符串中字符的各种函数，其中最常见的函数如表3–2所示。

表 3-2　在 <cctype> 库中的部分函数

检验字符类型的判断函数	
isalpha(*ch*)	如果 *ch* 是一个字母字符，则返回 **true**
isupper(*ch*)	如果 *ch* 是一个大写字母字符，则返回 **true**
islower(*ch*)	如果 *ch* 是一个小写字母字符，则返回 **true**
isdigit(*ch*)	如果 *ch* 是一个数字（'0' ～ '9'），则返回 **true**
isxdigit(*ch*)	如果 *ch* 是一个十六进制数字（'0' ～ '9'，'A' ～ 'F'，'a' ～ 'f'），则返回 **true**
isalnum(*ch*)	如果 *ch* 是一个**字母数字混合的**（alphanumeric）字符，则返回 **true**。一个字母数字意味着它要么是一个字母，要么是一个数字
ispunct(*ch*)	如果 *ch* 是一个标点符号，返回 **true**
isspace(*ch*)	如果 *ch* 是一个空白字符，则返回 **true**。' '（空格字符）、'\t'、'\n'、'\f'、'\v' 和 '\r' 都被认为是空白字符
isprint(*ch*)	如果 *ch* 是任意可打印的字符，则返回 **true**
大小写转换函数	
toupper(*ch*)	返回 *ch* 对应的大写字母（若 *ch* 不是一个字母则为 *ch* 本身）
tolower(*ch*)	返回 *ch* 对应的小写字母（若 *ch* 不是一个字母则为 *ch* 本身）

遗留的C风格字符串

`string str = "hello" + "," + "world";`

是不可以的，　在拼接操作符的两边必须有一方是C++ string，不可以都是字符串字面值。

编写字符串应用程序 ♥♥

回文识别

回文（palindrome）是指其字母排列正序与倒序均一致的词语，例如单词“level”或“noon”。

本节的目的是编写一个判断函数以检测一个字符串是否属于回文。调用isPalindrome（"level"）应该返回 true；调用isPalindrome（"xyz"）应返回 false。

```
bool isPalindrome(string str){
    int len = str.length();
    for (int i = 0; i < len/2; i++){
        if (str[i] != str[len - i - 1])
            return false;
    }
    return true;
}
```

一个更简洁的实现是使用reverse()

```
bood isPalindrome(string str){
    return str == reverse(str);
}
```

读起来几乎和英语一样流畅：如果一个字符串和它相反顺序的字符串相等，则它是一个回文。

尤其是当你正在学习编程时，致力于程序的简洁性比关注其执行效率更为重要。鉴于现在计算机的速度，牺牲几个机器周期来使程序更易于理解是值得的。

将英语翻译成儿童黑话

为了使你更多地了解如何实现对字符串操作的应用，本节列举了一个 C++ 程序，它读取用户输入的一行文字，然后将这行文字的每个单词从英语翻译成儿童黑话（Pig Latin），这是一种拼凑的语言，世界上说英语的大部分孩子都对其熟知。在儿童黑话中，单词是通过应用下面的规则从它们对应的英语单词中构造而成：

1．如果单词中不含元音字母，不作任何翻译，这意味着儿童黑话的单词和原单词一样。

2．如果单词以元音字母开始，则翻译出的儿童黑话包括原单词加上其后缀way。

3．如果单词以辅音字母开始，提取辅音字符子串直到遇见第一个元音字母，移动收集的辅音字母到单词的结尾，然后添加后缀ay，这样就形成了翻译后的儿童黑话。

line2piglatin将行分解成单词，而word2piglatin将单词按照规则转变。

PIGLATIN.cpp

```
/*
 * Function: lineToPigLatin
 * Usage: string translation = lineToPigLatin(line);
```

```
 * ------------------------------------------------
 * Translates each word in the line to Pig Latin, leaving all other
 * characters unchanged.  The variable start keeps track of the index
 * position at which the current word begins.  As a special case,
 * the code sets start to -1 to indicate that the beginning of the
 * current word has not yet been encountered.
 */

string lineToPigLatin(string line) {
   string result;
   int start = -1;
   for (int i = 0; i < line.length(); i++) {
      char ch = line[i];
      if (isalpha(ch)) { // 分词的粒度是以任意非字母元素的
         if (start == -1) start = i;
      }
      else {
         if (start >= 0) {
            result += wordToPigLatin(line.substr(start, i - start));
            start = -1;
         }
         result += ch;
      }
   }
   if (start >= 0) result += wordToPigLatin(line.substr(start));
   return result;
}
```

```
/*
 * Function: wordToPigLatin
 * Usage: string translation = wordToPigLatin(word);
 * ------------------------------------------------
 * Translates a word from English to Pig Latin using the rules
 * specified in the text.  The translated word is returned as the
 * value of the function.
 */

string wordToPigLatin(string word) {
   int vp = findFirstVowel(word);
   if (vp == -1) {
      return word;
   } else if (vp == 0) {
      return word + "way";
   } else {
      string head = word.substr(0, vp);
      string tail = word.substr(vp);
      return tail + head + "ay";
   }
}
```

```
22
23
24  /*
25   * Function: findFirstVowel
26   * Usage: int k = findFirstVowel(word);
27   * ------------------------------------
28   * Returns the index position of the first vowel in word.  If
29   * word does not contain a vowel, findFirstVowel returns -1.
30   */
31
32  int findFirstVowel(string word) {
33     for (int i = 0; i < word.length(); i++) {
34        if (isVowel(word[i])) return i;
35     }
36     return -1;
37  }
38
39  /*
40   * Function: isVowel
41   * Usage: if (isVowel(ch)) . . .
42   * -----------------------------
43   * Returns true if the character ch is a vowel.
44   */
45
46  bool isVowel(char ch) {
47     switch (ch) {
48      case 'A': case 'E': case 'I': case 'O': case 'U':
49      case 'a': case 'e': case 'i': case 'o': case 'u':
50        return true;
51      default:
52        return false;
53     }
54  }
55
```

Strlib.h

3.7 strlib.h 库

正如我多次在本书中所强调的，本章包含在库中的几个函数看起来很完美。一旦你将这些函数应用到实际编程中，你会感觉到在需要实现相同运算的其他应用场合中不使用它们是很浪费的。然而像 wordToPigLatin 这样的函数在其他地方不可能出现，而类似 toUpperCase 和 startsWith 这样的函数你会经常使用。因此为了避免重写和复制粘贴代码，将这些函数放在一个库中是很有意义的，这可以在你需要使用它们时提供极大的便利。

在本书中，Stanford 类库包含了一个接口，接口名为 strlib.h，它提供了如表 3-3 所示的函数。你可以在该表中看到若干函数，它们曾在本章中定义过。在习题中，你将有机会补全包括 endsWith 和 trim 在内的其他定义。表 3-3 中的头四个函数都与将数值转换成字符串形式相关，要求掌握的知识已超出了你目前掌握的 C++ 知识范围，但这只是暂时的。在第 4 章，你将学会如何使用一个新的数据类型，称作 stream 类，它能使这些函数的实现变得更加简单。

表 3-3 strlib.h 接口提供的函数

integerToString(*n*)	将整数 *n* 转换成对应的数字字符串
stringToInteger(*str*)	将数字字符串 *str* 转换成对应的整数
realToString(*d*)	将浮点数 *d* 转换成对应的字符串
stringToReal(*str*)	将实数字符串 *str* 转换成对应的实数值
toUpperCase(*str*)	返回一个新字符串，字符串中的内容是将 *str* 中所有的小写字符转换成它们的大写字符

（续）

toLowerCase(*str*)	返回一个新字符串，字符串中的内容是将 str 中所有的大写字符转换成它们的小写字符
equalsIgnoreCase(s_1, s_2)	在忽略大小写的情况下，如果 s_1 和 s_2 相等，则返回 **true**
startsWith(*str*, *prefix*)	若字符串 *str* 以指定的前缀 *prefix* 开始，其中，*prefix* 可以是一个字符串，也可以是一个字符，则返回 **true**
endsWith(*str*, *suffix*)	若字符串 *str* 以指定的后缀 *suffix* 结束，其中，*suffix* 可以是一个字符串，也可以是一个字符，则返回 **true**
trim(*str*)	从参数字符串 *str* 的开头至结尾处，删除其中所有的空白字符，然后返回一个新的字符串