第5章 语句

5.1 简单语句

1. 复合语句 *compound statment* 也被称作块 *block* 。块不以分号作为结束。

5.3 条件语句

5.3.1 `if` 语句

1. 悬垂 `else` *(dangling else)* 问题：C++规定 `else` 与离他最近的尚未匹配的 `if` 匹配，从而消除程序的二义性。

5.3.2 `switch` 语句

1. 如果某个 `case` 标签匹配成功，将从该标签开始往后顺序执行所有 `case` 分支，除非程序显示地终端了这以过程，否则直到 `switch` 的结尾处才会停下来。
2. 一般不要省略 `case` 分支最后的 `break` 语句。如果没写 `break` 语句，最好加一段注释说清楚程序的逻辑。
3. 不允许跨过变量的初始值语句直接跳转到该变量作用域的另一个位置。
4. 如果需要为某个 `case` 分支定义并初始化一个变量，应该把变量定义在块内，从而确保后面的所有 `case` 标签都在变量的作用域之外。

5.4 迭代语句

5.4.3 范围 `for` 语句

1. 循环中预存了 `end()` 的值。一旦序列中添加（删除）元素， `end()` 的值就可能变的无效。

5.5 跳转语句

5.5.3 goto 语句

1. 从 goto 语句无条件跳转到同一个函数内的另一条语句。
2. 不要在程序中使用 goto 语句，这会让程序既难理解又难修改。
3. 标签标识符可以和程序中其他实体的标识符使用同一个名字而不互相干扰。

5.6 try 语句块和异常处理

5.6.1 throw 表达式

1. 程序的异常检测部分使用 throw 表达式引发一个异常。其中表达式的类型就是抛出的异常类型。

5.6.2 try 语句块

1. throw 和 try 示例：

```
Sales_item item1, item2;
cin >> item1 >> item2;
try
{
    if(item1.isbn() != item2.isbn())
        throw runtime_error("Data must refer to same ISBN");
    cout << item1 + item2 << endl;
}
catch (runtime_error err)
{
    cout << err.what()
         << "\nTry Again? Enter y or n" << endl;
}

// 如果代码抛出异常，则 catch 输出的是：
// Data must refer to same ISBN
// Try Again? Enter y or n
```

2. 程序抛出异常时，会沿着程序的执行路径逐层回退，直到找到适当类型的 catch 子句为止。如果最终没能找到任何匹配的 catch 子句，程序会转到名为 terminate 的标准库函数，程序会非正常退出。
3. 如果一段程序没有 try 语句块且发生了异常。

4. 编写异常安全的代码非常困难，不在本书的讨论范围内。

5.6.3 标准异常

1. 异常类型只定义了一个名为 `what()` 的成员函数，返回值是一个指向 C 风格字符串的 `const char*` 。该字符串的目的是提供关于异常的一些文本信息。
2. 如果异常类型有一个字符串初始值，则 `what()` 返回该字符串。对于其他无初始值的异常安全类型来说， `what()` 返回的内容由编译器决定。