**作业四**

03013435 郭子

**题目要求**

将Guttag, John的《Introduction to Computation and Programming Using Python》中的：

6.2.3 When the Going Gets Tough

6.2.4 And When You Have Found “The” Bug

翻译为中文。

要求：

1. 翻译重在“意达”，翻译中能体现自己理解的“意”为佳；
2. 翻译以后根据自己编程体验，写体会。
3. 翻译文档内容：英文原文、翻译和编程体验
4. **英文原文**

**6.2.3 When the Going Gets Tough**

Joseph P. Kennedy, father of President Kennedy, reputedly instructed his children, “When the going gets tough, the tough get going.” But he never debugged a piece of software. This subsection contains a few pragmatic hints about what do when the debugging gets tough.

• Look for the usual suspects. E.g., have you

o Passed arguments to a function in the wrong order,

o Misspelled a name, e.g., typed a lowercase letter when you should have typed an uppercase one,

o Failed to reinitialize a variable,

o Tested that two floating point values are equal (==) instead of nearly equal (remember that floating point arithmetic is not the same as the arithmetic you learned in school),

o Tested for value equality (e.g., compared two lists by writing the expression L1 == L2) when you meant object equality (e.g., id(L1) == id(L2)),

o Forgotten that some built-in function has a side effect,

o Forgotten the () that turns a reference to an object of type function into a function invocation,

o Created an unintentional alias, or

o Made any other mistake that is typical for you.

• Stop asking yourself why the program isn’t doing what you want it to. Instead, ask yourself why it is doing what it is. That should be an easier question to answer, and will probably be a good first step in figuring out how to fix the program.

• Keep in mind that the bug is probably not where you think it is. If it were, you would probably have found it long ago. One practical way to go about deciding where to look is asking where the bug cannot be. As Sherlock Holmes said, “Eliminate all other factors, and the one which remains must be the truth.”

• Try to explain the problem to somebody else. We all develop blind spots. It is often the case that merely attempting to explain the problem to someone will lead you to see things you have missed. A good thing to try to explain is why the bug cannot be in certain places.

• Don’t believe everything you read. In particular, don’t believe the documentation. The code may not be doing what the comments suggest.

• Stop debugging and start writing documentation. This will help you approach the problem from a different perspective.

• Walk away, and try again tomorrow. This may mean that bug is fixed later in time than if you had stuck with it, but you will probably spend a lot less of your time looking for it. That is, it is possible to trade latency for efficiency. (Students, this is an excellent reason to start work on programming problem sets earlier rather than later!)

**6.2.4 And When You Have Found “The” Bug**

When you think you have found a bug in your code, the temptation to start coding and testing a fix is almost irresistible. It is often better, however, to slow down a little. Remember that the goal is not to fix one bug, but to move rapidly and efficiently towards a bug-free program.

Ask yourself if this bug explains all the observed symptoms, or whether it is just the tip of the iceberg. If the latter, it may be better to think about taking care of this bug in concert with other changes. Suppose, for example, that you have discovered that the bug is the result of having accidentally mutated a list. You could circumvent the problem locally (perhaps by making a copy of the list), or you could consider using a tuple instead of a list (since tuples are immutable), perhaps eliminating similar bugs elsewhere in the code.

Before making any change, try and understand the ramification of the proposed “fix.” Will it break something else? Does it introduce excessive complexity? Does it offer the opportunity to tidy up other parts of the code?

Always make sure that you can get back to where you are. There is nothing more frustrating than realizing that a long series of changes have left you further from the goal than when you started, and having no way to get back to where you started. Disk space is usually plentiful. Use it to store old versions of your program.

Finally, if there are many unexplained errors, you might consider whether finding and fixing bugs one at a time is even the right approach. Maybe you would be better off thinking about whether there is some better way to organize your program or some simpler algorithm that will be easier to implement correctly.

1. **翻译**

**6.2.3 当前路变得艰难的时候**

据说肯尼迪总统的父亲约瑟夫·P·肯尼迪在教育他的孩子的时候说：“当事情变得艰难的时候,坚强者继续前行。”但他从来没有调试过一个软件，下面的段落给出了能帮助你在程序调试陷入困境时，找出问题所在的有效提示：

一．寻找常见的错误。例如，你有没有：

1.将参数传递给函数的时候顺序出错；

2.拼错一个名称，例如，当你应该键入一个大写的字母时，输入了小写字母；例如，输入小写字母时，当你应该键入一个大写的

3.未能重新初始化变量；

4.测试两个浮点值是否完全相等而不是约等于（记住，浮点运算和你在学校里所学的运算是不一样的）；

5.在你想要两个对象相等时（例如：id（L1）==id（L2）），检查它们的数值是否相等（例如：通通过表达式L1==L2来比较两列数据）；

6.忘记一些内置函数所具有的副作用；

7.忘记“（）”会将一个对函数类型对象的引用转变成函数的调用；

8.创建一个没有定义过的对象；

9.犯一些你自己常犯的错误。

二．不要再问自己为什么程序不是按照你所想的在运行，相反，问问你自己为什么程序像现在这样。 那会是一个更加容易回答的问题。并且，这可能将会是找出修复这个问题的办法中最好的第一步

三．记住，错误也可能并没有出现在你认为它应该出现的地方。如果它真的出现在了那儿，那么你应该早就找到了它。一个决定从哪里开始寻找错误的最有实用性的方法是问自己错误最不可能出现在哪里。就像福尔摩斯所说，“消除所有其他的可能，剩下的那一个就是真相。”

四．尝试向别人解释问题。我们都有盲点。往往在向别人阐述自己遇到的问题时，你会发现自己所忽视的东西。最好的方法便是试着去解释为什么错误出现在了你意料之外的地方。

五．不相信你阅读到的所有东西。特别是文献资料。程序可能不会像意见中所说的那样工作。

六．停止调试并开始编写文档。这将帮助你从不同的角度看问题的方法。

七．先离开,明天再试一次。这可能意味着问题会被解决得更晚，但是你可能会花更少的时间来寻找问题的所在。即是说，我们也许可以用等待来换取效率。(同学们，这是一个很好的早一些而不是更晚去开始课程练习题程序设计的理由！

)

**6.2.4 当你找到错误所在的时候**

当你认为你在你的代码中发现了一个错误，那种想要马上就开始编码和测试修改的冲动是难以抗拒的。但是，放慢一些往往是更好的选择。请记住，我们的目标是不是为了修复一个错误，而是要迅速有效地迈向一个没有错误的程序。

问问自己，如果这个错误是否解释了所有观察到的异常，或者它是否是只冰山一角。如果是后者，那么你最好在考虑修改这个错误的时候要保持它和其它的变化相一致。假定这样一个例子，你发现错误是由于不小心改变了一个列表导致的。你可以（也许是通过将列表备份）局部地绕过这个问题，或者你可以考虑用元组替代列表（因为元组是不可变的），这样也许就消除了代码中其他地方类似的错误。

在进行任何更改之前，尝试去思考“修复”所带来的衍生问题。这将会破坏其他地方吗？是否会导致程序过于复杂？它是不是能够为整理代码的其他部分提供机会？

始终要确保你能回到你所处的地方。没有什么比认识到一系列的改变使你离目标越来越远时，而你却没有办法再次回到你最开始的地方更加让人沮丧的了。磁盘空间通常很充足。用它来存储旧版本的程序。

最后，如果有许多不明原因的错误，你应该思考一下逐个找到错误并修改是不是正确的方法。也许你会更好地思考是否有更好的方法来组织你的程序或一些简单的算法，将更容易实现正确的结果。

1. **编程体验**

通过这次作业对两小节英文文章的翻译，我有了很多感触与收获。因为自己目前还没有用python写过很多程序，大一时的c++学得也不是很好，所以我的感触更多一些。

翻译工作本身并不算是特别大的难题但是比较“繁”，绝大多数句子并不复杂可以读懂，其中有一些专业词汇和表达方式，还有一些词在编程语境下的意思和平时也不大一样，还有一些比较长的句子需要考虑结构顺序，这些都花了我一定的精力与时间。理解单词含义，并且通过恰当的表达讲其组成一句句完整通顺，让人可以读懂的话,这次作业使我的英语翻译能力得到了一次很好的锻炼。

接下来就该说说这两段文章所讲述的关于编程方面的问题了，对于第一篇文章，“当事情变得艰难的时候,坚强者继续前行。”这句话对我还是蛮受用的，编程本身就是在不断的错误中摸索，最后找到一套解决问题的方法。还有就是提示中的第一点寻找常见的错误，里面的九个小点确实是在自己过去的学习过程中经常会犯的，大的地方会注意，往往是小错误困扰时间更多。

第二篇文章告诉了我们在发现错误时的正确处理态度:”我们的目标是不是为了修复一个错误，而是要迅速有效地迈向一个没有错误的程序。”这句话十分富有哲理，编程犯错太正常不过，不要执着与错误本身，而是应该对整个程序进行思考和把握后再做决定，尝试不同的思路，甚至可以在错误很多的时候考虑更换一种算法，也许就会有不一样的结果。