**课程实践作业四**

**一 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.

**二 译文**

**6.2.3 当事情变的艰难的时候**

肯尼迪总统的父亲约瑟夫·帕特里克·肯尼迪曾告诫他的孩子们：“当事情变的艰难的时候，坚强的人会继续前行。”他从来没有调试过程序。但是在调试程序的过程中，这句话也能给我们带来一些实用的启示。

•寻找最常见的错误，例如你是否忽略了以下几点：

1. 将参数传递给函数的时候弄错了顺序
2. 拼错了一个关键词，例如你本应该输入大写字母却输入了小写字母
3. 忘记初始化变量
4. 检测到写的程序是令两个浮点型数值相等（例如a==b）但实际运行结果却是两个数值约等（记住，浮点型的运算跟你在学校学习的算数是不一样的。）
5. 当你想比较两个对象是否相等时（如：id(L1) == id(L2)),），你写的代码实际是在比较两个数值是否相等（如：用表达式L1 == L2比较2个列表)
6. 忘记了一些内置函数有副作用
7. 如果你忘记了“（）”，会使对对象的引用变成对函数的调用
8. 无意中创建了别名
9. 或者在你经常出错的地方犯一些典型的错误

•别再问为什么这个程序不是像我们所想的那样运行，而是应该问问自己为什么程序会是这个样子。这应该是一个很容易回答的问题，而且这很有可能就是我们找到如何很好地解决这个问题的方法所迈出的第一步。

•要记住，错误不会总是出现在你所认为的地方，如果确实如此的话，你早就应该发现它了。实践的过程中，想要找到错误所在就得考虑错误最不可能出现在哪里。正如福尔摩斯所说，“排除所有其他可能，剩下的那个就一定是真相。”

•要试着向别人解释问题所在，我们都会存在思虑不全的情况，当你试图向别人阐述观点的时候你通常会意识到自己忽略了什么。解决问题的一个很好的方法就是向别人解释错误为什么不会出现在某个地方。

•不要完全相信你读到的任何东西，尤其是文献资料。代码可不会完全按照书里的见解运行。

•偶尔停止调试程序并写一些文档，这会帮助你从不同的角度思考问题解决问题。

•放下手头的工作明天再试，这可能意味着你修复错误的时间会比一直坚持来得晚，但是你可以节省很多时间。也就是说，我们可以用延时来换取效率。（同学们，这是一个督促我们早点开始学习程序设计的好理由。）

**6.2.4 当你找到问题所在**

当你认为自己已经找到了程序中的问题时，那种想要立刻开始测试和修复程序的欲望是难以抗拒的。然而，慢一点进行往往会是更好的。要记住，我们的目的不是修复一个错误，而是更快更有效的使我们的程序没有错误存在。

问问自己这个错误是否能够解释我们观察到的所有现象，又或者那只是冰火山一角而已。如果是后者，我们最好小心的对待这个错误以使它与其他的变化相适应。例如，你已经发现了这个错误是由一个意外发生变化的列表引起的，你可以（通过建立一个副本）局部的避免这个问题，或者你可以考虑使用元组来代替列表（因为元组是不变的），也许这样可以消除代码中其他地方的同种错误。

在做任何改变之前，要努力思考一下改正这个错误之后可能引起的其他问题。它会破坏其他部分的结构吗？它会使程序变得更加复杂吗？它能使代码的其它部分更顺畅吗？

要确保你的代码可以回到最初始的状态。当你意识到你所做的一系列修改反而使你离目标越来越远，但是你却无法回到最初的状态是最令人感到沮丧的了。磁盘的空间是充足的，要记得利用它储存你的原始数据。

最后，如果仍然有一些无法解释的错误，你应该思考逐个的寻找并且修改错误是否是正确的途径，也许你能找到更好的方法组织程序，或者使用一些更简单的算法来得到正确的结果。

**三 心得体会**

想要学好编程，实践远比理论学习来的有用，实践方可出真知。在学习编程的过程中我们应该在了解理论知识后就立刻去实践，这样我们才能对理论掌握得更加扎实，理解的更加深入。而且这样才能将理论学习和实践更好地结合在一起。编写代码就像做英语阅读一样，需要有相应的所谓“语感”，因此我们在学习一门新的程序语言的时候，我们要常做练习，即使是照着现有的示例在打一遍，运行一次也会大有收获。然而在实践的过程中我们总会遇到各种各样的问题，在阅读这部分的文章后，我得到了很大的启示。

当事情变的艰难的时候，坚强的人会继续前行。这句话说的很有道理，而且在我们学习程序设计的过程中也有很大的帮助。学习一门程序语言不是一蹴而就的事情，我们应该沉下心来慢慢摸索。

首先，当我们在编写程序的过程中遇到问题的时候，我们需要有耐心，勤于思考，而且我们应该多注意细节，编程是容不得一点错误的，也许我们只是少打了一个括号或者是一个等号，在程序语言中都会有很大影响。其次，我们要学会从多个角度考虑问题，我们不能总以为错误只会出现自我们所认为的地方。我们得摆脱固有思维，也许自己觉得最不可能出现问题的地方往往是解决问题的关键所在。再次，我们要学会总结，同样的错误不能总犯，一直盲目前进却不知道总结经验绝对不是一种好的学习方法。同时我们得学会停下来，并不是一路向前冲就能最快的解决问题，有时候让自己休息一下，放松一会，自己的思维或许就会发生改变，那时候在看待原来的问题就会发现一切都没有那么复杂。

在以后学习编程的过程中，只要多多注意上述几点，想信我一定能对程序设计有更多新的认识并且掌握的更加扎实。