4. 建立子程序的步骤

程序设计语言（PDL）

> 类似于伪代码，不依赖于特定语言。描述基本框架和结构，要在写代码之前就想好和写出基本的骨架来，既清晰，也方便后续修改。

- PDL 可以使评审工作变得更容易。不必检查源代码就可以评审详细设计。它可以使详细评审变得很容易，并且减少了评审代码本身的工作。
- PDL 可以帮助实现逐步细化的思想。从结构设计工作开始，再把结构设计细化为 PDL，最后把 PDL 细化为源代码。这种逐步细化的方法，可以在每次细化之前都检查设计，从而可以在每个层次上都可以发现当前层次的错误，从而避免名影响下一层次的工作。
- PDL 使得变动工作变得很容易。几行 PDL 改起来要比一整页代码容易得多。你是愿意在蓝图上改一条线还是在房屋中拆掉一堵墙？在软件开发中差异可能不是这样明显，但是，在产品最容易改动的阶段进行修改，这条原则是相同的。项目成功的关键就是在投资最少时找出错误，以降低改措成本。而在 PDL 阶段的投资就比进行完编码、测试、调试的阶段要低得多，所以尽早发现错误是很明智的。
- PDL 极大地减少了注释工作量。在典型的编码流程中，先写好代码，然后再加注释。而在 PDL 到代码的编码流程中，PDL 本身就是注释，而我们知道，从代码到注释的花费要比从注释到代码高得多。
- PDL 比其它形式的设计文件容易维护。如果使用其它方式，设计与编码是分隔的，假如其中一个有变化，那么两者就毫不相关了。在从 PDL 到代码的流程中，PDL 语句则是代码的注释，只要直接维护注释，那么关于设计的 PDL 文件就是精确的。

设计子程序

步骤：

- 给子程序命名
- 考虑效率
- 研究算法和数据结构
- 编写PDL
- 编写工作应该从抽象到具体，一个子程序最抽象的部分便是最开始的注释部分，如果感到编写抽象说明时有困难，应该想到可能是某一环节出了问题。
- 考虑数据
- 检查PDL，人们往往更愿意检查一个只有几行的PDL，而不愿意看好多行的子程序

  要确认对子程序做什么和将怎样做已经有了清楚透彻的了解。如果在 PDL 这一层次上对这点还没有概念上的了解，那么在编码阶段了解它的机会还有多少呢？如果连你都理解不了它的话，又有谁会理解呢？
- 逐步细化

子程序编码

把PDL当注释，在每一行注释下面编码。

在每一行注释下面填上代码。在每一行 PDL 注释语句下面填上代码。这有点像给报纸排版。首先画好轮廓线，然后再把每一篇文章填到空格中，每一个 PDL 注释行都相当于给代码画的轮廓线，而代码相当于文章。同文学文章的长度一样，代码的长度也是由它所要表达的内容多少决定的。程序的质量则取决于它的设计思想的侧重点和巧妙程度。

检查子程序

> 这说的完全就是我……

业余爱好者与职业程序员之间的最大区别就是迷信还是理解。在这里，“迷信”这个词指并不是指在月圆之夜产生各种错误或使你毛骨悚然的一段程序。它指的是你对代码的感觉代替对代码的理解。如果你总是认为编译程序或者硬件系统有故障，那说明你还处在迷信阶段。只有 5% 的错误是由编译程序、硬件或者是操作系统引起的（Brown and sampson, 1973, Ostrand and Weyuher, 1984）。进入理解境界的程序员总是怀疑自己的工作，因为他们知道 95% 的错误出自这里。要理解每一行编码的意义，并且要明白为什么需要它。没有仅仅因为有效便是正确的东西。如果你不知道为什么它是有效的，那么往往它是无效的，只不过你没有发现罢了。

> 也有点像我……

但是，如果晚一些开始编译，将会获得许多收益。其中的一个主要原因是，一旦开始编译，那么你脑袋里的秒表便开始嘀嗒作响了，在第一次编译之后，你就开始不停地想：下次编译一定让它全对。结果，在这种“就只再编译一次”的压力下，作了许多匆忙的、更易产生错误的修改，反而浪费了更多的时间。所以，在确信子程序是正确的之前，不要急于开始编译。

本书的重点之一，就是想告诉读者如何避免陷入把各种代码拼凑到一起，通过试运行检验它是否有效的怪圈。而在确信程序是正确的之前，就匆忙开始编译，恰恰是陷入了这种怪圈。如果你还没有进入这个怪圈，那最好还是当确信程序正确之后再开始编译。

5. 高质量子程序特点

子程序是具有单一功能的可调用的函数或过程。

【模块】

模块的内聚性准则，与单个子程序的内聚性准则一样，都是十分简单的。一个模块应该提供一组相互联系的服务。

模块与程序其它部分间的耦合标准与子程序间的耦合标准也是类似的。模块应被设计成可以提供一整套功能，以便程序的其它部分与它清楚地相互作用。

信息隐蔽，每一个模块的最大特点都是通过设计和实现，使它对其它模块保密。模块的作用是将自己的信息隐蔽起来以保卫自己的隐私权。另一个称谓是“封装”。一个模块应该像是一座冰山，你只看到它的一角，而它其余7/8的部分则 藏在水下。常见需要隐含的信息：容易被改动的区域，复杂的数据，复杂的逻辑，在编程语言层次上的操作。

【设计】

软件结构设计、高层次模块设计和实现细节设计。设计的层次：1、划分成子系统，2、划分成模块，3、划分成子程序，4、子程序内部的设计。自顶向下分解&自底向上合成。设计是一个复杂、险恶、启发的过程。好的设计都是通过迭代逼近得到的、结构化设计比较适合于小规模的子程序组合，面向对象设计更适合于小规模的子程序组合，同时对于功能变化可能性比数据大的问题也是较适用的。

【数据】

数据结构在创建阶段能带来的收益大小，在某种程度上是由它对创建前的高层次工作影响大小决定的。好的数据结构所带来的收益往往是在需求分析和结构设计阶段体现出来的。为了尽可能地利用好的数据结构带来的收益,应在需求分析和结构设计阶段就定义主要数据结构。建立自己的数据类型，以增加程序的可变动性。表驱动法是一种编程模式，从表里面查找信息而不使用逻辑语句（if和case）。表提供了一种复杂逻辑和继承结构的替代方案。

【变量】

作用域准则：尽可能减小作用域，把对某一变量的引用集中放置。应尽量使变量成为局部或模块的，避免使用全局变量。使每个变量有且仅有一个功能。 阅读代码的次数远远多于编码，所以命名变量应该是阅读方便而不是编写方便
复杂数据结构：使用结构化数据来表明数据间的关系，使用结构化数据来简化对成块数据的操作，使用结构化数据来简化参数表，使用结构化数据来降低维护工作量。恰当的对数据进行结构化，可以使程序更简单、更容易理解也更容易维护。可以使用表来代替复杂的逻辑结构。

【顺序程序语句】

顺序语句指导：组织代码使它们间的依赖关系明显，子程序的名字应当清楚地表明依赖关系，使用子程序参数使依赖关系明显，注明不明确的依赖关系。
与顺序无关的程序语句：使代码能由上读到下，不要到处转移。使用一变量局部化。使变量存活时间尽可能短。相关语句组织在一起。