

程序规划方法漫谈

程序匠人原创

Wang1jin 收藏.

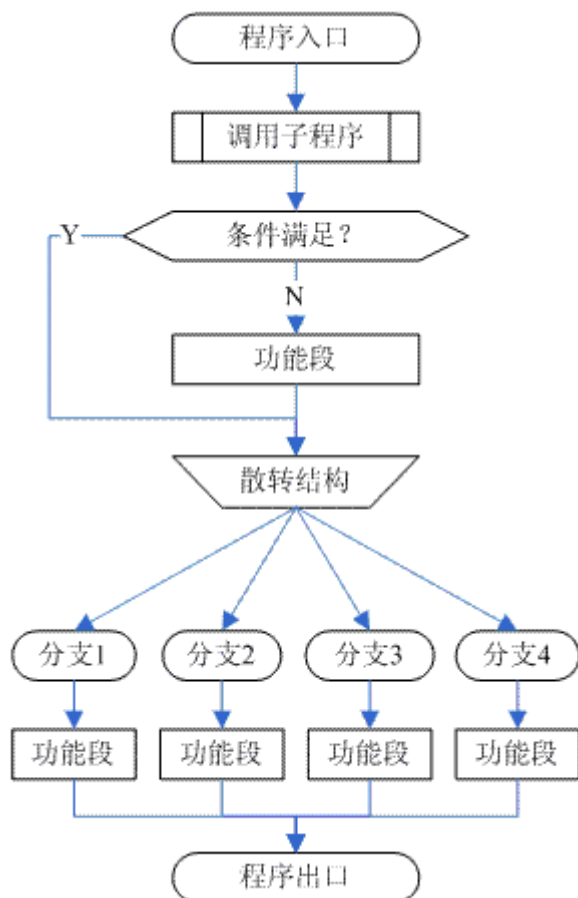
交流论坛: <http://bbs.cepark.com/>

推荐网站: <http://www.cepark.com>

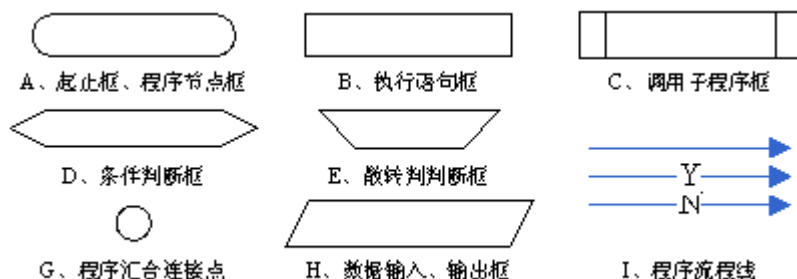
个人博客: <http://wang1jin.cepark.com>

一、前言 “程序设计”的真谛是什么？许多初学者的理解是“写代码”。但是，在匠人看来，把“程序设计”理解为“写代码”，就像把“电路设计”理解为“画 PCB”一样。新手们苦恼的问题是，他们只会“写代码”。他们一接到新的项目，总是在第一时间就爬到键盘上去敲代码。新手们的精力总是比较旺盛，他们加班加点，两天就把所有代码敲完。然后他们会用十倍或几十倍以上的时间去调试，中间伴随着几次三番的推倒重来。最后，他们交出一个勉强能跑的程序。这种程序，外行乍一看，觉得还行；内行乍一看，却是吓出一身冷汗！这也许不能怪新手们，因为他们的老师还没有来得及教会他们“程序设计”的一些方法。他们甚至还没有学会写注释，就已经毕业了。于是他们只能在毕业后的工作中，去完成这段本该在学校里完成的修炼。要说到程序设计，最重要的一种方法，就是“多思考”。偏偏这又是最难手把手地教的。在此，匠人介绍一些设计时比较常用方法给大家。我们可以借助这些方法来对程序进行更高效、更多维的规划。

二、程序流程图 1、从一个简单的流程图说起 我们先来看看这个图（参见图 1.1：一个程序流程图例子）。许多人都很熟悉，它的名称叫“流程图”，或者“程序流程图”。流程图是一种传统的算法表示法，程序流程图是人们对解决问题的方法、思路或算法的一种描述。它利用图形化的符号框来代表各种不同性质的操作，并用流程线来连接这些操作。 2、流程图的作用 流程图简单直观，应用广泛，功能卓越。在程序的规划阶段，通过画流程图，可以帮助我们理清程序思路。尤其是在非结构化的汇编语言中，流程图的重要性不言而喻。在程序的调试、除错、升级、维护过程中，作为程序的辅助说明文档，流程图也是很高效便捷的。另外，在团队的合作中，流程图还是程序员们相互交流的重要手段。阅读一份简明扼要的流程图，比阅读一段繁杂的代码更加易于理解。



3、流程图的基本画法 按道理，画流程图应该是每个程序员的基本功。匠人惊讶的是，居然有那么多人不会或不屑于画流程图。在这里，匠人罗列出一些流程图中常用的符号（参见图 1.2：常用的流程图符号）。细心的读者会发现，这里给出的一些流程图符号与教科书上的有点出入。比如说符号 D（条件判断框），书上给出的一般都是四角菱形的，而不是匠人推荐使用的六角菱形。匠人在实践中发现，容纳同样多的文字，六角菱形比四角菱形可以节省更多的空间。这也就意味着我们可以在同样大小的幅面内画出更多的内容。因此，除非是您公司里有明文规定必须使用四角菱形，否则就让教科书见鬼去吧。另一个不同点，就是如果程序中要调用一个子程序，那么最好给这个子程序一个特别的符号，就像符号 C（调用子程序框）。这样做的好处是可以更有利于阅读。



4、画流程图软件 匠人推荐用 Visio 软件来画流程图。这款软件功能非常强大，而画流程图只是它众多功能中的一个。您只需新建一个 Visio 文件，点击菜单“文件”->“形状”->“流程图”->“基本流程图”，就可以得到许多现成的流程图符号。在 Visio 画好的流程图，可以很方便地复制到 Word 环境中。并且可以在 Word 中进一步进行修改编辑。当然，如果您只是偶然画流程图，也可以用 Word 或 Excel 软件的画图功能来实现。它们一样可以画流程

图，只是没有那么专业罢了。（未完待续）

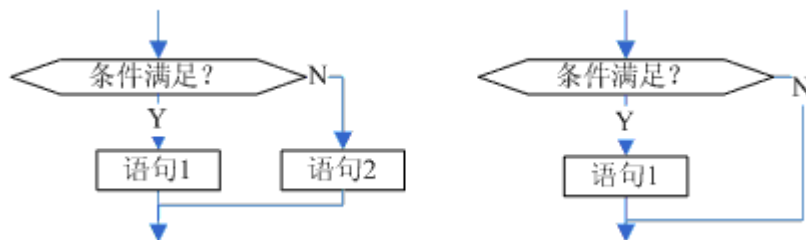
5、流程图的结构化 早期的非结构化语言中都有类似“goto”的语句。它允许程序从一个地方直接跳到另一个地方去。而随着 C 语言的盛行，对程序的结构化要求，必然在流程图中得到体现。经过研究，人们发现任何复杂的程序算法，都可以分解为顺序、选择（分支）和循环这三种基本结构。基本结构之间可以并列、嵌套，但不允许交叉跳转。我们构造一个算法的时候，也仅以这三种结构为构成单位，并遵守三种基本结构的规范。如果说“goto”是孙悟空的“筋斗云”，那么“结构化”就是“如来神掌”。也就是说，不管你如何翻腾，也不能从一个结构直接跳转到另一个结构的内部去。呵呵！这就是结构化编程的要求。它的好处就是结构清晰，易于正确性验证，易于纠错。既然整个算法都是由三种基本结构组成的，那么，我们只要掌握这三种结构的流程图画法，就可以画出任何算法的流程图，无往而不利了。

（1）顺序结构顺序结构是简单的线性结构，每条语句按顺序执行（参见图 1.4：顺序结构）。太简单了，实在没啥好说。



（2）选择（分支）结构

（2）选择（分支）结构选择（分支）结构是对某个给定条件进行判断，条件为“真”（满足）或为“假”（不满足）时，分别执行不同的程序语句。当条件不满足时，有时需要执行一些语句，而有时可能什么都不做，由此分化出两种形态。（参见图 1.5：选择（分支）结构）



散转（Switch）结构

对于简单的选择（if）结构，条件判断的结果只有 Yes 和 No 两种。而在更复杂的选择结构中，比如说对某个表达式的值进行多重条件判断，结果就会有許多。假设这个表达式可能 =0、1、2、3、或者溢出，那么结果就有 5 个分支，我们可以用 4 个选择结构来实现这个流程图（参见图 1.6：多重选择（分支）结构）。当然，我们也可以用一个散转（Switch）结构来画（参见图 1.7：散转（Switch）结构）。

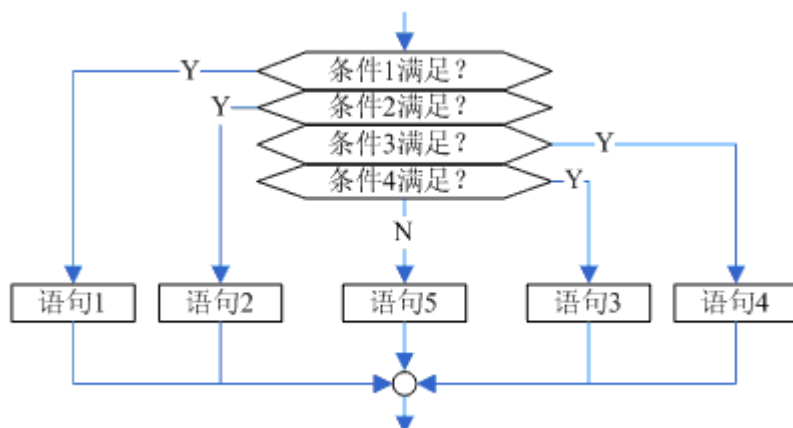
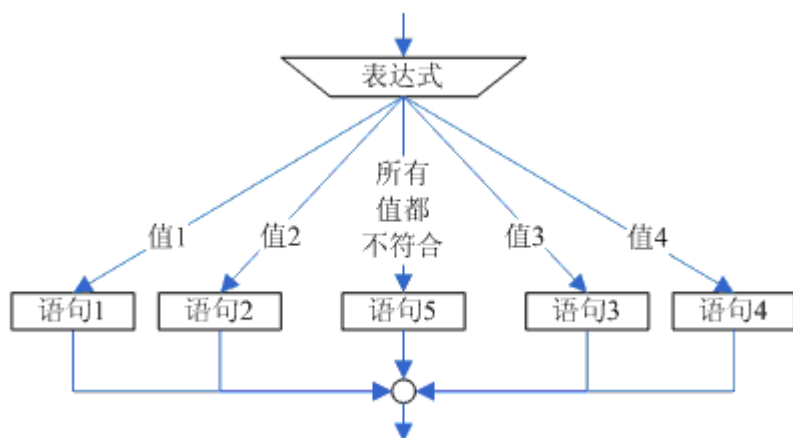


图 1.7: 散转 (Switch) 结构



(3) 循环结构 循环结构的常见形态, 包括当型 (While) 结构和直到型 (Do-While) 结构。当型 (While) 循环结构中, 是在每一轮循环的开始处执行条件判断, “当”条件满足则继续执行循环体中的语句, 否则跳出循环。(参见图 1.8: 当型 (While) 循环结构)。直到型 (Do-While) 循环结构中, 是先执行循环体中的程序语句, 然后再在循环结束处进行条件判断, 如果条件满足则继续开始新一轮循环, 周而复始; “直到”条件不满足时跳出循环。(参见图 1.9: 直到型 (Do-While) 循环结构)。由于直到型循环结构是先执行语句, 后进行条件判断, 也就是说, 在直到型循环结构中, 循环体中的语句起码会被执行 1 次

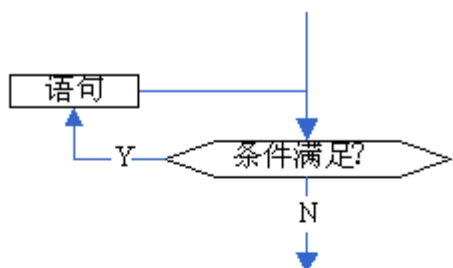


图 1.8 当型 (While) 循环结构

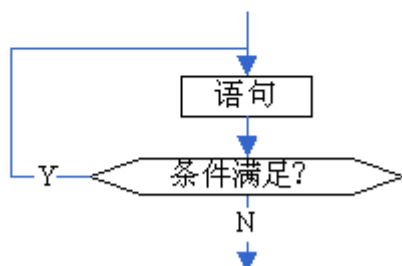
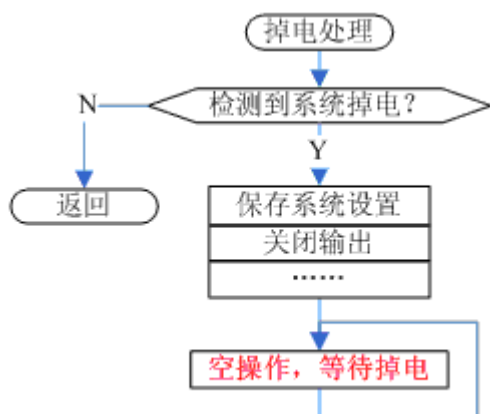
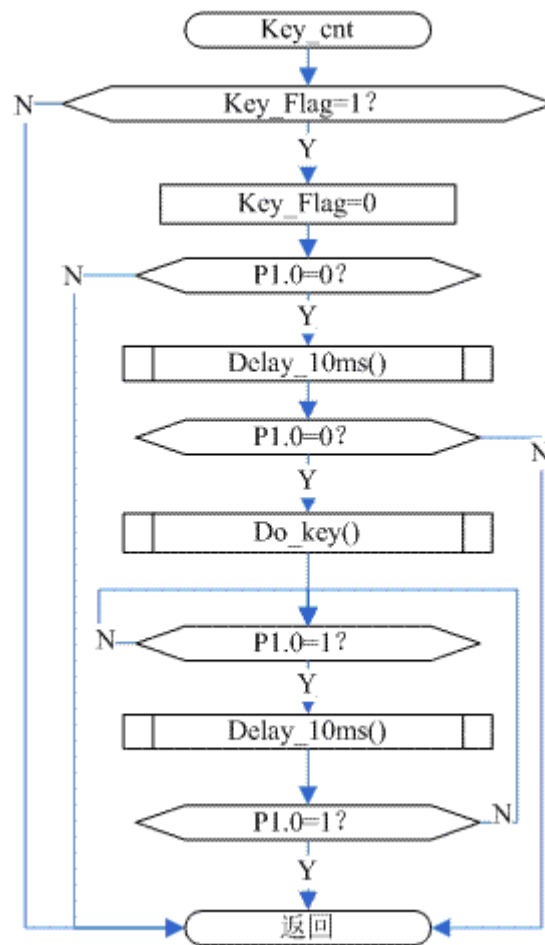


图 1.9 直到型 (Do-While) 循环结构

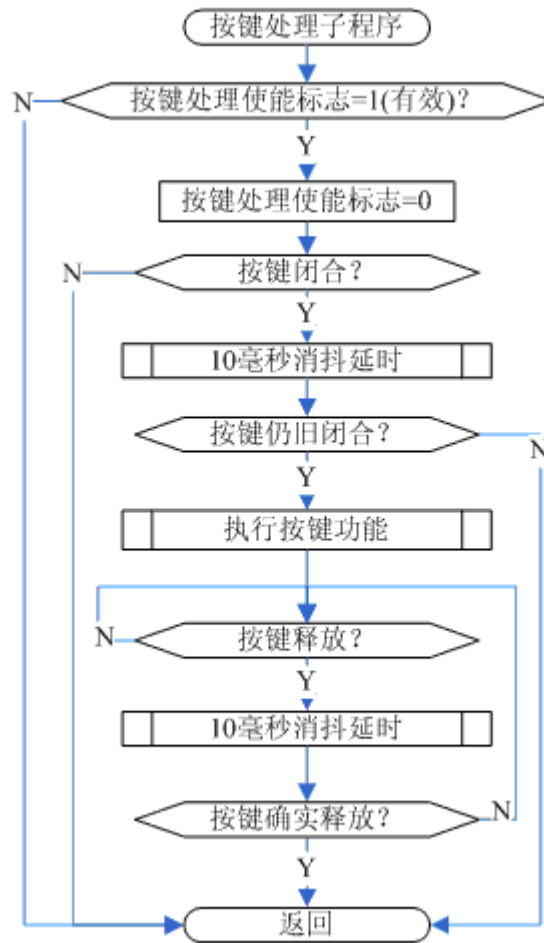
当条件判断的结果恒为“真”(Yes)时,循环就永远不会退出了。我们称之为“死循环”。主程序(主函数)就是一个常见的死循环的例子。一般情况下,我们要避免死循环的产生。因为这样会导致其它任务被挂起——除非我们有意想那么做。举例说明:比如在掉电处理程序中,当我们检测到系统掉电信号后,需要预先保存好系统设置、并关闭输出,然后执行一个空操作的死循环。这个时候,我们有意要把所有任务挂起,让系统无牵无挂地、专心致志地“等死”(这就是“安乐死”,呵呵!)



6、流程图的人性化 程序是写给机器看的,因此必须严格遵循编程语言的规范,否则机器就无法解析执行。而流程图则是写给人看的,因此应当尽可能地人性化。让我们对比一下这两个流程图(参见图 1.11:“没人性”的流程图和图 1.12:人性化的流程图)。它们描述的程序功能是一样的。前者更简洁,但是却给阅读者带来了思维障碍,“Key_Flag”是什么标志?“P1.0”又是什么功能的 IO 口?阅读者为了了解这些问题,不得不放下流程图去查阅相关的文档。阅读思路的连贯性被打断了。很显然,我们更喜欢第二个流程图。这才是无障碍的阅读。不要吝啬那点打字的功夫,磨刀不误砍柴工。



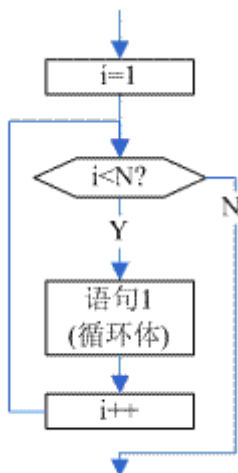
非人性化流程



人性化流程图.

额外说说 for 循环语句

说到循环结构, 还要额外说说 for 循环语句。因为今天在论坛里有 fzu_csc 网友问怎么只介绍了 while 循环和 do- while 循环, 却没有介绍 for 循环。让我们先举个例子, 先看看下面这个 for 循环程序: for(i=1; i<=N) 时, 结束循环。可以看出, for 循环其实就是一个当型循环结构 (参见图 1.11: for 循环例子)。



7、流程图的简化一个繁琐的流程图，是画流程图的人的噩梦，同时也是阅读者所不愿意见到的。因此，我们在画流程图时，不必拘泥于规则和形式，一板一眼。在适当的地方进行简化，不但可以节省精力和幅面，而且可以让流程图更易于阅读。让我们对比下面这两个图（参见图 1.14：繁琐的流程图和图 1.15：简洁的流程图）。这是同一个程序的两种流程图表达方法。程序的功能，是用查询法对脉冲输入口上输入的脉冲计数，当计数满 100 次后进行一些处理并退

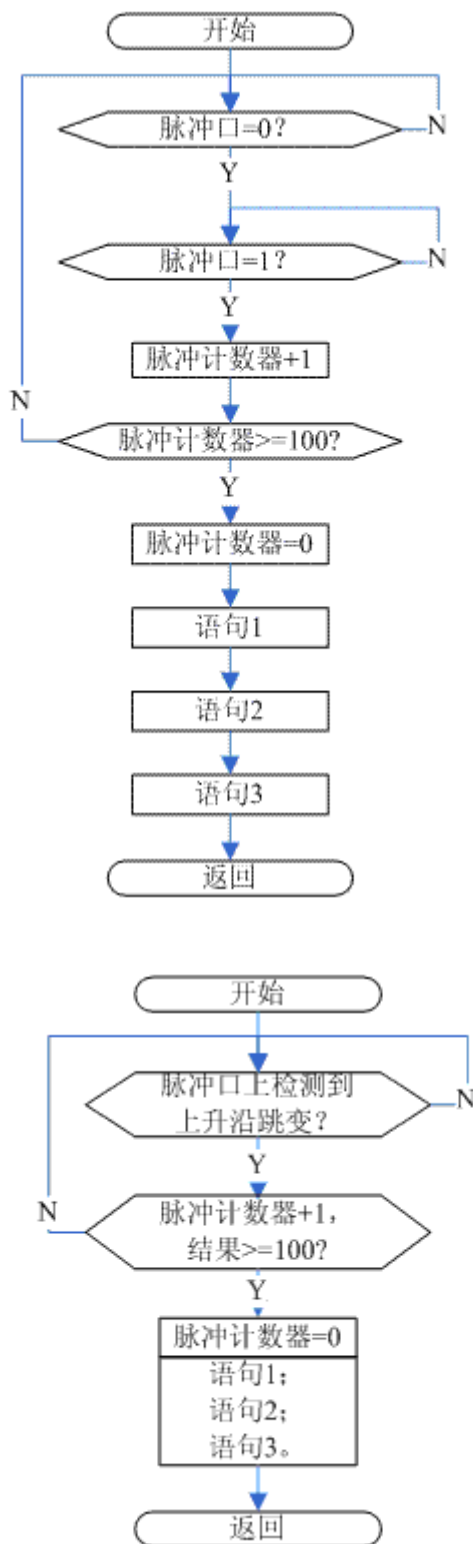
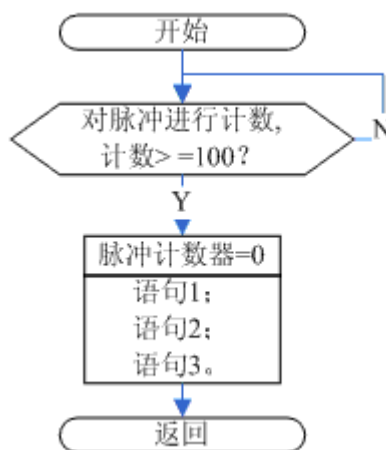


图 1.14：繁琐的流程图

图 1.15: 简洁的流程图 很显然, 第一个图占用了更多的幅面, 却并不易于理解。而第二个图在经过简化处理后, 节省了空间, 程序思路反而表达得更加清晰。 我们甚至可以进一步抛弃那些细微末节 (参见图 1.16: 更简洁的流程图)。反正这个流程图是给人看的, 只要看



得懂程序思路, 能简化何乐而不为呢?

图 1.16: 更简洁的流程图 简单的才是有效的。适当的简化, 是人性化的体现。

总结一下简化的方法: (1) 合并那些相互关联的条件判断。比如: 当“条件 1”满足, 且“条件 2”满足执行某个动作。我们可以把这两个条件判断合并后, 放入同一个条件判断框中 (参见图 1.17: 分支结构)。 (2) 省去一些不必要的流程线。对于不会产生歧义的顺序结构流程, 可以把那些箭头省略。节省空间, 甚至可以把一些语句合并在一个执行语句框里面。(参见图 1.18: 顺序结构的简化)

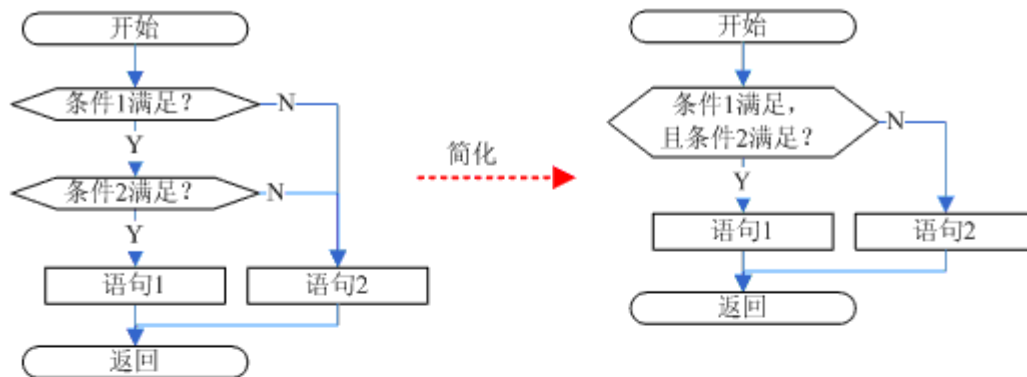


图 1.17：分支结构的简化

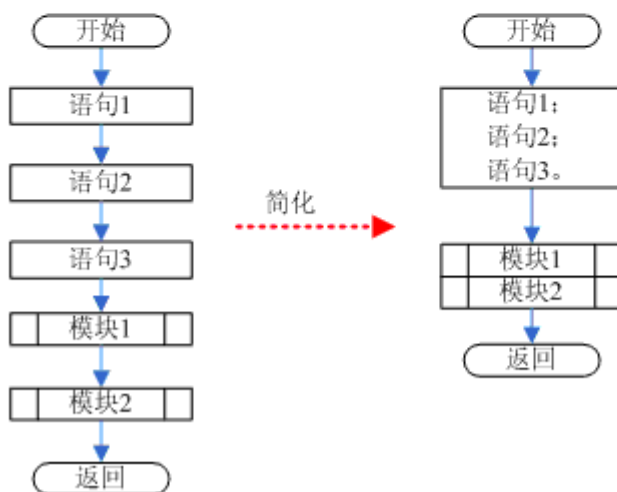
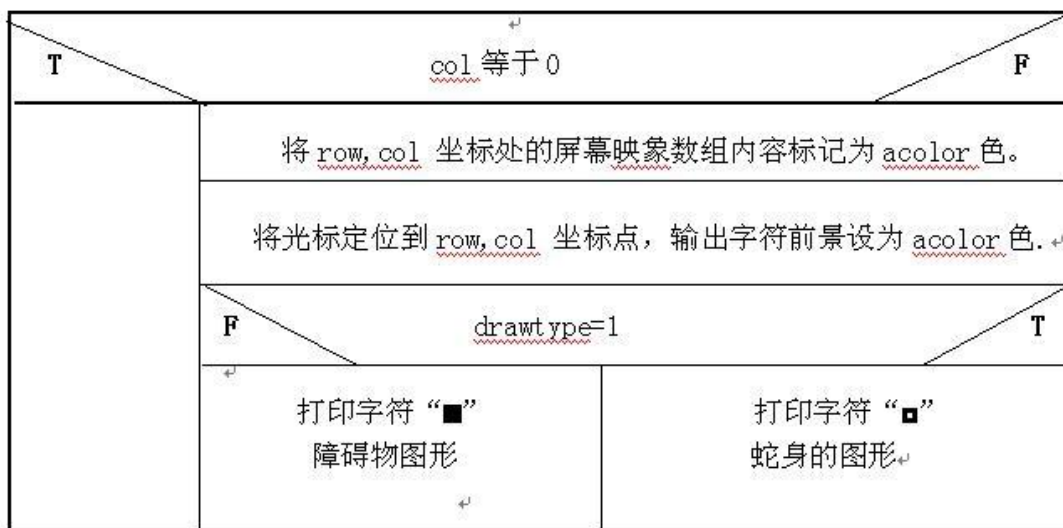


图 1.18：顺序结构的简化

4.4 模块 12: `set(row,col,acolor,drawtype)`

用盒图表示如下:



三、N-S 图（盒图）1、N-S 图简介前面讲到流程图的简化时，匠人已经说过：可以把一些不会产生歧义的流程线（箭头）省略。持这种想法的显然不止是匠人一个，据说有两个美国佬走得更远。这两位名叫 I.Nassi 和 B.Shneiderman 的“国际友人”经过潜心研究发现了个规律：既然任何算法都是由前面介绍的三种基本结构组成，那么各基本结构之间的流程线就是多余的。于是他们设计了一种新的算法表示法。用若干个大大小小的框图，通过堆砌或嵌套，来表示程序的流程。这种流程图被以两个美国天才的名字命名为“N-S 图”。（匠人发现天才也得赶早，否则好事都会被别人抢了去！）那一个个框，就像一个个封闭的盒子。所以，N-S 图又被形象地称为“盒图”。N-S 图（盒图）的特性就像盒子一样，结构性很强。由于取消了流程线，象“goto”这样乱跳的语句，也就没有了表达的形式。所以，N-S 图又被人称为是“结构化流程图”。也就是说，对于传统的流程图，结构化编程依赖于程序员的自觉自律；而对于 N-S 图，结构化编程则是由绘图规则来强制保证的。你想不结构化都不行，呵呵。N-S 图除了表示几种标准结构的符号之处，不再提供其他如“流程线”这样的描述符号，这就有效地保证程序的质量。NS 图的另一个优点是形象直观。例如循环的范围、条件语句的范围都是一目了然的，所以容易理解设计意图，为编程、排错、调试、维护都带来了便利。

2、N-S 图的画法在 N-S 图中，一个算法就是一个大矩形框，框内又包含若干小框。其实只

要掌握 N-S 图的 3 个基本结构画法，即可掌握 N-S 图。匠人在这里简单介绍如下：（1）

顺序结构 顺序结构（参见图 1.19：顺序结构 N-S 图），语句 1、语句 2、语句 3 依次执行。

（2）选择（分支）结构 选择结构（参见图 1.20：选择（分支）结构 N-S 图），条件为真时

执行语句 1，条件为假时执行语句 2。如果条件为假时不需执行任何语句，则对应的执行语

句框中留空。散转（Switch）结构作为选择（分支）结构的一个特例，在 N-S 图方法中较

少被人提及。其实可以用下面这种方法来画（参见图 1.7：散转（switch）结构）。当表达式

结算结果等于某个对应的值时，执行该值下面的语句。（3）循环结构 循环结构的两种常

见形态，包括当型结构（参见图 1.22：当型（while）循环结构 N-S 图）和直到型结构（参

见图 1.23：直到型（do-while）循环结构 N-S 图）。



图 1.19: 顺序结构N-S图

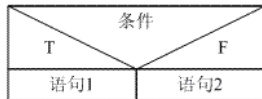


图 1.20: 选择(分支)结构N-S图

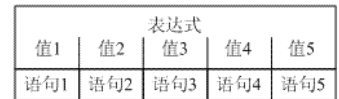


图 1.21: 散转(switch)结构N-S图

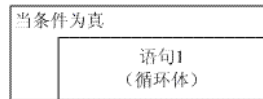


图 1.22: 当型(while)循环结构N-S图

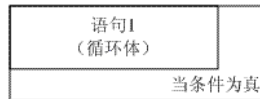
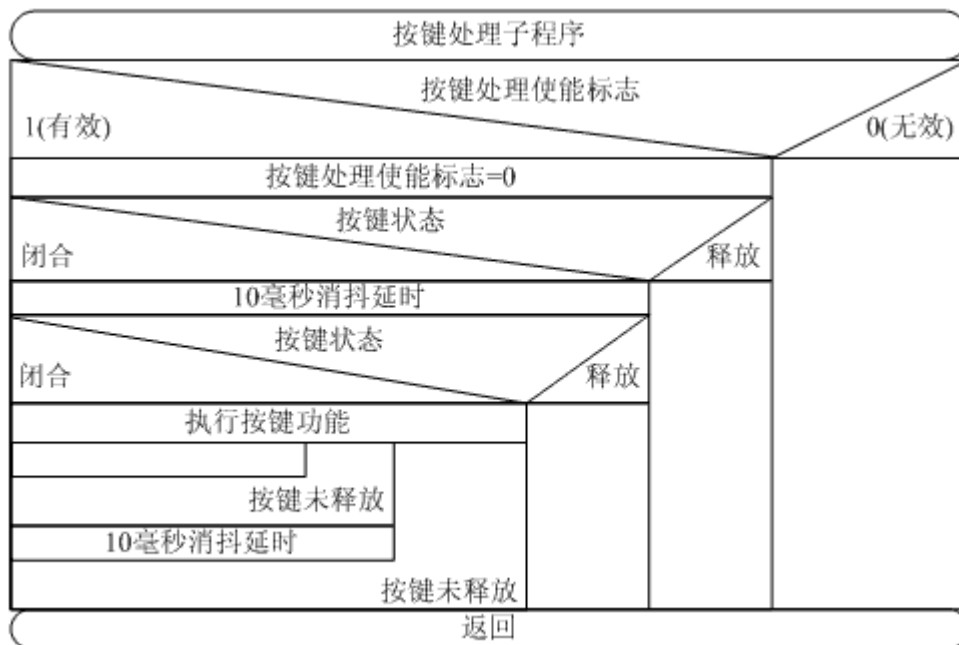


图 1.23: 直到型(do-while)循环结构N-S图

3、N-S 图的实例 我们把前面介绍过的一个“按键处理子程序”的流程图（参见图 1.13：人性化的流程图），用 N-S 图形式再来画一次，如下图（参见图 1.24：按键处理子程序 N-S 图）：



4、N-S 图的软肋 说了半天 N-S 图的花好稻好，那为什么在设计中，还是有许多人却放弃使用 N-S 图，仍旧选择了落后的带箭头的流程图呢？这得说说 N-S 图的软肋。要说这 N-S 图最大的缺点，就是手工画图时，修改起来没有流程图方便。尤其是在分支嵌套层次较多时，就比较难画了。手工画图不便，计算机画图也不便。目前，匠人也没有找到比较适合画 N-S 图的计算机绘图软件。 这些，可能是阻碍 N-S 图进一步推广应用的原因。

最下面给大家介绍几个下载资料的地方:

推荐网站: <http://www.cepark.com/Index.html>
技术交流论坛: <http://bbs.cepark.com/>

51 学习专区:

<http://51.cepark.com/>

USB 学习专区:

<http://usb.cepark.com/>

CAN 学习专区:

<http://can.cepark.com>

AVR 学习专区:

<http://avr.cepark.com/>

FPGA 学习专区:

<http://fpga.cepark.com/>

STM32 学习专区:

<http://stm32.cepark.com/>

ARM 学习专区:

<http://arm.cepark.com/>

DSP 学习专区:

<http://eda.cepark.com/>

PIC 学习专区:

<http://pic.cepark.com/>

DIY 电子制作专区:

<http://diy.cepark.com/>

GPS 学习专区:

<http://gps.cepark.com/>

推荐网站: <http://www.cepark.com/Index.html>

技术交流论坛: <http://bbs.cepark.com/>

GUI 学习专区:

<http://gui.cepark.com/>

EDA 软件学习专区:

<http://eda.cepark.com/>

电源学习专区:

<http://power.cepark.com/>