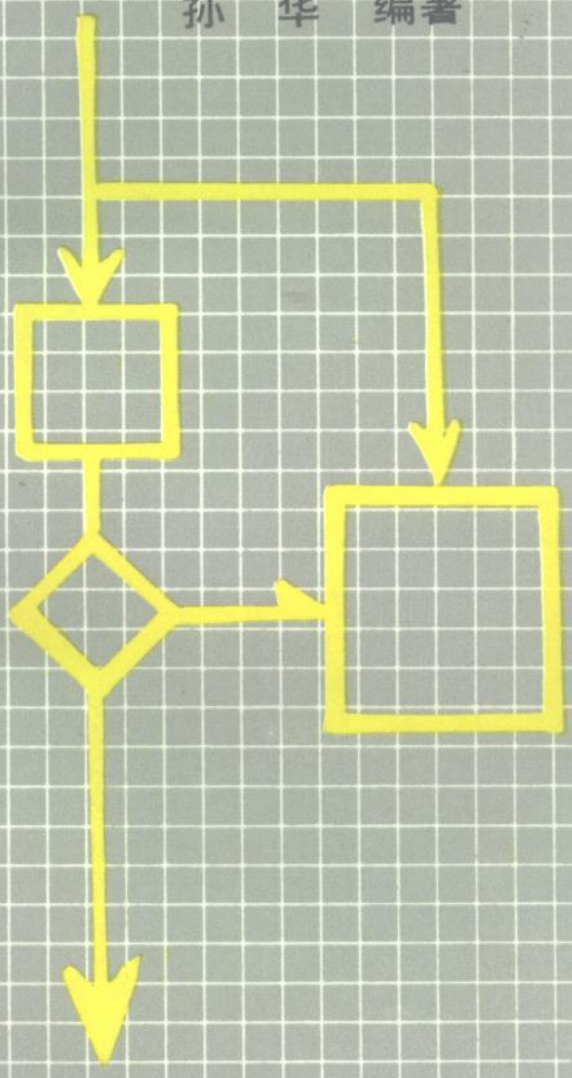
实用程序设计方法（中级）

TIANJIN
中德培训中心

实用程序设计方法（中级）

孙 华 编著



中国科学技术出版社

实用程序设计方法

（中 级）

孙 华 编著

中国科学技术出版社

内 容 提 要

本书从实用的角度出发，介绍了有关的程序设计方法，并不涉及程序语言各个语句，而是对程序设计过程中可能遇到的主要问题加以描述，以帮助程序设计的初学者在实际工作中应用。

本书可供程序设计的初学者阅读、应用，亦可做为教材。

(京)新登字175号

实用程序设计方法

（中级）

孙 华 编著

*

责任编辑：勇 夫

封面设计：王 福

技术设计：王守祯

*

中国科学技术出版社出版（北京海淀区魏公村白石桥路32号）

新华书店北京发行所发行 各地新华书店经售

中国科学院印刷厂印刷

开本：787×1097毫米 1/16 印张：9.75 字数：222千字

1993年3月第一版 1993年3月第一次印刷

印数：1—2 120册 定价：5.70元

ISBN 7-5046-0782-7/TP·30

前　言

计算机热早已席卷中华大地，数以千万计的人至少学习过一门计算机语言，但是他们之中能够自己编写程序的人却少之又少。仅仅学过一门语言就着手编写程序，往往会遇到许多令人困惑的问题，要经过许多失败，才能逐渐摸索出一些经验。

实际上许多失败是不必要的，计算机技术发展至今天，对于程序设计已经有着比较成熟的一整套原则、方法。但是一般的计算机语言课程只讲授各种语句的格式与用法，对程序设计本身讲得很少，而一些专门的书籍又往往理论性太强，不适于非计算机专业的广大读者。

因此，我们将初学者在编程时可能遇到的各种问题，可能涉及的原则、方法收集并组织在一起，以通俗、实用的方式讲述。所有论述均由实例引出，并配有大量的习题。

程序设计具有很强的实践性，仅仅阅读本书不会有很大的效果，读者应该通过上机编制并调试有关的例题、习题，通过编程实践掌握各章的内容。自己动手，这是掌握程序设计的唯一途径!

本书讲述的是与程序设计有关的原则、方法，不涉及具体的计算机及语言。但本书以 BASIC 语言做为实例使用的语言。这至少有两个优点，第一绝大多数读者都学过，第二凡是它可做的事情用其它语言做起来更加简单。

本书的适用对象是所有学过计算机语言，又准备自己动手编写程序的有志者。目前许多中、高等学校(非计算机专业)都在酝酿进一步开拓计算机课程，迫切需要合适的教材，本书也是这方面的一个尝试。每年一次进行的程序员国家统一考试，程序设计是其中的难点，我们希望认真读完本书的读者将不再视其为难点。

本教材已选为中德培训中心的教材。天津中德培训中心是中华人民共和国与德意志联邦共和国在职业教育方面最大的一个合作项目。她的主要任务是：为全国的企业培训和造就机械、电子方面的高级技术工人和技术干部。

天津中德培训中心　**孙华**

目　录

第一章　几种BASIC 语言及与 Pascal 语言的比较

本章对比较了苹果机 BASIC(Applesoft)，PC BASIC、True BASIC 及 Pascal 语言在控制语句方面的不同。目的是希望读者通过对比，能更好地领会新语句、新结构的优越性。即使对于学过 Pascal 语言的人来说，这一点也仍然重要。我们在教学中就曾发现一位刚学过Pascal 语言的学生仍试图用 if 和 goto 语句(象在 Applesoft 中一样)去构筑 while 型循环。

没有学过 True BASIC 或 Pascal 语言的读者可略过本章。

此后本书再提到"BASIC 语言"，除非特别声明，一般指 Applesoft，其结论往往也适用于 PC BASIC。

§ 1.1 APPLESOFT 中的基本语句

基本 BASIC 语句共有 17 条，各种版本的 BASIC 语言均包括这 17 条语句。随着计算机科学的发展，有些语句的形式有所改进，有些语句被新的语句替代。通过比较这些形式上的差异，可以发现不少问题。本章重点考虑的是控制语句，因为程序的流程主要由它们所决定。

APPLESOFT 中基本语句　　表 1.1

语　句	功　能	实　例
① READ	从数据区读数	20 READ A , X1, B $
② DATA	在数据区中存放数据	30 DATA −1, 2.07, "SUB"
③ RESTORE	恢复数据区起始位置	50 RESTORE
④ LET	计算并赋值	10 LET X1=X2+2 : B$="BEQ"
⑤ INPUT	由键盘即时输入数据	40 INPUT X1, X2, X3
⑥ PRINT	打印数或字符串	50 PRINT "SUM=" ; S
⑦ REM	注释	45 REM SUB 1
⑧ DIM	说明数组的大小	60 DIM A(5), B(3, 2), C$(9)
⑨ DEF	定义用户函数	40 DEF FN A(X)=X*X−5*X+100 (只允许一个参数)
⑩ STOP	使程序暂停	60 STOP
⑪ END	程序终止	80 END
⑫ GOTO	转移	15 GOTO 30
⑬ IF	条件转移	20 IF A<1 AND B=5 THEN 100(THEN 后语句亦可)
⑭ FOR	循环设置与执行	60 FOR N=1 TO 15 STEP 2
⑮ NEXT	循环出口	70 NEXT N
⑯ GOSUB	调用子程序	50 GOSUB 100
⑰ RETURN	由子程序返回调用程序	150 RETURN

ON GOTO语句虽不属基本语句，由于我们的需要也列于下。

⑱ ON GOTO	控制转移(多分支)	50 ON K GOTO 100, 200, 300

以下依次给出 PC BASIC, True BASIC 中基本语句的形式。为便于对比，各语句的编号是一致的。最后，列表给出BASIC与PASCAL之间主要语句的对比。有些细节予以忽略。

§1.2 PC BASIC中的基本语句

为简便起见，凡是与前面所列(APPLESOFT)相同的地方就不再讨论，以下仅讨论主要不同点。

一、⑨ DEF 定义用户函数

允许多个参数，例:

```
10 DEF FNMUD (X,Y)=X-(INT(X/Y)*Y)
```

二、⑬ IF 条件转移

增加了两分支 IF 语句:

IF〈逻辑表达式〉THEN { 〈行号〉 / 〈语句序列〉 } ELSE { 〈行号〉 / 〈语句序列〉 }

三、⑭ FOR 循环设置与执行
⑮ NEXT 循环出口

这两个语句仍然保留。又提供了 while 循环，形式如下:

```
WHILE 条件
   ⋮
(循环体)
WEND
```

功能是只要条件为真，就再一次执行循环体。

四、通过编辑可以使程序清单显示出程序嵌套的层次，如:

```
140 FOR I=1 TO 4
150     C=1 : A=1
160   FOR K=1 TON
170     C=C * K : A=A * K
180   NEXT K
190   Y1=Y1+A/C
200 NEXT I
```

§1.3 True BASIC 中的基本语句:

一、True BASIC 简介:

它是 BASIC 创造人 John G. Kemeny 和 Thomas E. Kurt 等人 1985 年的作品。严格遵照美国国家 BASIC 标准，是一个典型的结构化程序设计语言。由于它有良好的移植性，已在许多微型机上实现。以下针对的是 IBM PC 上的 True BASIC。国内已有汉化的版本。

True BASIC 不仅保留了 BASIC 语言的易学、简洁等特点，而且符合"结构化程序设计"的理论。它提供了一些新的控制结构，与公认的结构程序设计语言Pascal 风格相近。

其作者声称: True BASIC 开拓了BASIC 的新纪元，是真正的BASIC。

二、与原有 BASIC 语言的主要差异:

PC BASIC 已较APPLESOFT 有些改进，这在前面已讨论过。因此，这里的差异是将 True BASIC 与前者加以比较。

1. 与其它 BASIC 版本的兼容问题:

True BASIC 较老版本有很大的改进。但是为了兼容，用户需要将在其它BASIC 版本上写成的程序转换为 True BASIC程序。为此，True BASIC 的作者给予了相应的考虑。

对于兼容，主要考虑的是: 所有早期的控制语句都在 True BASIC 中保留。这样，如果你使用了下面语句中的任何一个，你的程序行就得使用行号。这些语句是 GOTO，GOSUB，IF ……THEN〈行号〉，ON GOTO，ON GOSUB。

True BASIC 规定: 若程序中有一行编号，那么所有的行都要编号。行号的规定和执行顺序与老版本是一致的。

True BASIC 所提供的新的控制语句可以完全替代以上那几条语句。这样，我们编写程序时就可以不使用那些易于造成混乱的旧的控制语句。在新的程序中，没有行号。

以下分别讨论各个不同点。

2. ④ LET 计算并赋值

a. LET 不得省略。　　b. 允许多变量赋值。

```
如: let   e=2.718282
    let   i,j=2
```

第二句相当于 I=2 : J=2。

3. ⑨ DEF 定义用户函数

除了单行函数外，又增加了多行函数。即允许用多行语句来表示一个函数，这时，要用 end def 结尾。

例:

```
def ×××$(n)
```

```
let  S$=""
for  i=1 To n
  let  S$=S$ & "×"
next i
let   ×××$=S$
end def
```

4. ⑯ GOSUB　调用子程序

⑰ RETURN　由子程序返回调用程序

True BASIC 保留了这两个语句，但这时程序要有行号。我们可以使用更好的可替代它们的语句。

子程序形式：

```
sub  程序名(参数1，参数2，…)
  ……
  ……
  ……
 end  sub
```

子程序调用：

```
call 程序名(表达式1，表达式2,…)
```

5. ⑫ GOTO　转移

前面已提到，保留该语句，但此时程序应有行号。使用新的控制语句可以避免使用 GOTO 语句。

6. ⑬ IF　条件转移

PC BASIC 中有 IF－THEN－ELSE 结构，但是它们都放在一行中。True BASIC 提供了多行的 IF－THEN－ELSE 结构，这时要用 end if 表示结构的结束。形式如下：

```
if   条件   THEN
     ……
     ……  } 块1
     ……
else
     ……
     ……  } 块2
     ……
end if
```

单行的 IF－THEN－ELSE 可以进行嵌套，如：

```
25  IF x>y THEN PRINT "x>y" ELSE IF y>x
    THEN PRINT "y>x" ELSE PRINT "x =y"
```

这种形式不宜处理较复杂的情况，也不易看出嵌套的层次。

在 True BASIC 中可以利用多行的 IF－THEN－ELSE 进行嵌套。如：

```
if guess<1 then
  print "Must be at least 1."
else
  if guess>6 then
```

```
    print "Can't be more than 6."
  else
    if guess<answer then
      print "Too Low."
    else
      if guess >answer then
        print "Too high."
      else
        print "Correct !!!"
        stop
      end if
      end if
      end if
      end if
```

这种形式仍较繁，有多个 end if 要一、一与前面对应。 True BASIC 提供了 else if 语句，从而可用更加优雅、易读的形式写出上例:

```
if guess<1 then
  print "Must be at least 1."
else if guess>6 then
  print "Can't be more than 6."
else if guess<answer then
  prmt "Too low."
else if guess>answer then
  print "Too high. "
else
  print "Correct !!!"
end if
```

只有一个end if。 按顺序判定每个条件。 第一个条件满足时，执行 print "Must be at least !" 不满足则判定"guess>6"是否满足。 如条件都不满足时执行 else 后面的 print "Correct !!!"。

7. ⑭ FOR 循环设置与执行
   ⑮ NEXT 循环出口

这两个语句仍然存在。 但是 True BASIC 又提供了另一种循环结构: DO 循环。形式如下:

```
do <……>
……
……   } 循环体
……
loop <……>
```

在 do，loop 的后面可以出现while〈条件〉或 until〈条件〉，可以两者都出现，可以只出现一个，也可以都不出现。 每次执行循环体前要检查 do 后面的条件。 每次执行循环

体后，要检查 loop 后面的条件。 当 while 的伴随条件为真时就重复执行循环结构。 当 until 的伴随条件为真时，就停止执行循环体。这样就可以根据条件的不同情况使循环体的执行出现以下几种可能：根本没有执行循环，只执行一次，执行多次或无限循环下去。

无论哪一种循环，我们常常需要从循环的中间转出来。 True BASIC 提供了专门的转出循环语句：exit for（exit do）。它们可放在 FOR 循环(DO 循环)的循环体的中间，一般与 if 语句连用，当执行到它们时立即转到 next 语句(loop 语句)后面的语句。

8. ⑱ ON GOTO 控制转移(多分支)

保留该语句，但此时程序应有行号。 采用新的控制语句可以避免使用它。

可替 代它的新的控制语句是 select case 语句，形式如下：

```
select case 表达式
  case 检测1，检测 2，…
       ……
       …… } 块1
       ……
  case 检测3，检测 4，… …
       ……
       …… } 块2
       ……
     ……
  case else
       ……
       …… } 块E
       ……
end select
```

以end select 结束。 case else 只能放在最后。 case 中的检测必须与 select case中的表达式类型一致。

该语句通过多个 case 构成多个分支。 执行时 True BASIC 首先给select case 中的表达式赋值。 然后按照先后次序与各检测进行比较，如与某一检测匹配就执行相应的 case 语句后的语句块，然后跳到 end select 后面的语句继续执行。 如果没有匹配的检测，就执行 case else 后面的语句块。

各个检测必须互不相同，不可能同时选择两个不同的语句块去执行。

显然，select case 语句较 ON GOTO（及ON GOSUB)更加醒目、整齐、灵活。

从某种意义上说，select case 语句类似前面介绍的 IF－THEN－ELSEIF，都是从几个不同的选择中，每次执行其中之一。 但是，从形式上 select case 更加简洁和优雅。

9. 全局变量与局部变量：

在旧版本的 BASIC 语言中变量都是全局变量，即使使用子程序也仍是如此。 例如：某子程序中有变量 A1，在该子程序之外也有一个变量 A1，那么 BASIC 认为它们是同一个变量。 显然，当程序较大时这很不方便，也易于将变量名搞错。

True BASIC 提供了外部子程序与外部函数以解决这一问题。 外部子程序与外部函数所用的变量均局部于本子程序或函数，称为局部变量。 如果是局部变量，那么即使它们与子程序或函数外面的变量重名，也互不影响，True BASIC 认为它们是不同的变量。

§1.4 旧的BASIC与Pascal在程序的控制语句与结构方面的对比

列表给出:

表1.2

	BASIC	PASCAL
赋值	LET J=1 J=1	J:=1
转移	GOTO 100	goto 100
复合	I=1 : J=1	begin I:=1; J:=2; end
分支	IF J>1 THEN 100 IF J>1 THEN I=1 IF J>1 THEN I=1 ELSE I=2	if J>1 then goto 100 if J>1 then I:=1 if J>1 then I:=1 else I:=2
分支	10 ON I GOTO 20, 40, 60 20 J=3 30 GOTO 70 40 J=1 50 GOTO 70 60 J=2	case I of 1: J:=3; 2: J:=1; 3: J:=2 end
循环	10 FOR I=1 TO 10 20 PRINT "WHEE" 30 NEXT I	for I:=1 To 10 do writeln ('whee')

循环	10 I=1
	20 IF I>10 THEN 60
	30 PRINT "WHEE"
	40 I=I+1
	50 GOTO 20
	60 REM
		I:=1;
		while I<=10 do
		begin
		writeln ('whee');
		I:=I+1
		end

循环	10 I=1 20 PRINT "WHEE" 30 I=I+1 40 IF I<=10 THEN 20 50 REM	I:=1; repeat writeln ('whee'); I:=I=1 until I>10

子程序	1 GOTO 100
	10 REM SUB
	20 PRINT "ALL THE WAY HOME"
	30 RETURN
	40 REM
	100 REM PROGRAM STARTS HERE
	110 GOSUB 10
	120 END
		program Stars Below (input, output);
		procedure Write All;
		writeln ('All the way home');
		end;
		begin
		Writeln
		end

【习 题】：

1. BASIC 语言的基本语句有哪几条?
2. 你使用的 BASIC 语言中有哪些控制语句?
3. PC BASIC 在基本语句方面的改进有哪些?
4. True BASIC 在基本语句方面的改进有哪些?
5. True BASIC 的改进大不大? 它所追求的是什么风格? 谈谈你的看法。
6. 按照表 1.2 所列的项目，将True BASIC 与PASCAL 进行比较。

第二章 结构化程序

§2.1 问题的提出

一、历史背景:

在早期，由于计算机硬件价格贵，性能差(速度慢、存储量小)，人们在编制程序时总是尽量少占用内存、尽可能缩短运行的时间。 程序员们千方百计的追求程序的"技巧"，以便减少哪怕是一条语句、一个存储单元。 但是往往越"巧妙"的程序，越难以被人读懂。

计算机技术的迅速发展，使硬件的价格大大下降，速度与容量成千百倍地提高，程序的效率已不是主要的问题。 另一方面，计算机的使用日益深广，人们所编制的软件越来越庞大、复杂，程序的编制、修改、完善日益困难，耗资巨大。 沿用旧的方法已难以处理更加复杂的程序。

这时编写程序的主要限制是人们自身对程序的理解，而不是计算机的硬件特性。 因此，为了提高程序的可靠性和编写程序的效率，必须提高程序的易读性，追求简洁清晰的风格，必须摒弃因人而异的手工业方式，寻找一套公认的规范与方法。

1965 年计算机科学家 Dijkstra 建议把 GOTO 语句从编程语言中删除。 1968 年他在著名的"GOTO 通信"中进一步指出利用一定的程序结构技术会使程序更易编、易读、易改，从而更容易成为正确的程序。 "GOTO"通信发表以后，引起了极大的轰动，导致一场大辩论。 赞成去掉 GOTO 的人认为它转来转去，是造成程序复杂的主要原因，而程序流程复杂又导致难编、难读、难调、难改，从而也就难于保证程序的正确性。 争论的结果远远超出了 GOTO 语句本身。 人们开始研究程序设计的方法，注意到程序的结构。最后比较一致的看法是: 问题在于按照一定的规则保持程序的结构，而不在于是否使用 GOTO 语句。

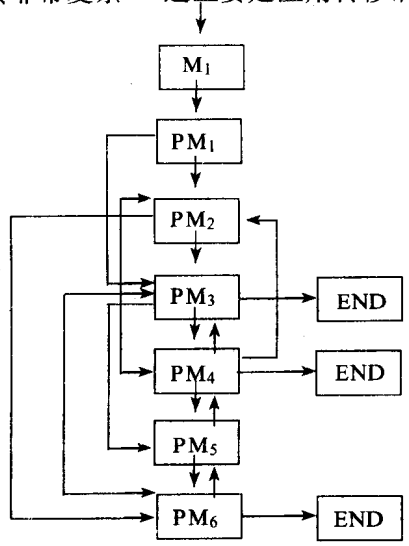
目前，程序设计已发展出许多成熟的方法。 "结构化程序设计"就是其中的一种，它是指利用自顶向下，逐步求精，模块化等方法编制结构化程序。 结构化程序由若干"基本结构"按一定的规则构成，因此简洁清晰。

请读者注意，本章讲的"结构化"与后面讲的"模块化"是有区别的。 以写一篇文章来比喻。 "结构化"规定，每一段另起一行，前面空两格，句号、逗号不得在每行开头，引号必须配对，等等。 总之，对文章的格式作一定规定，使之清晰、易读，"模块化" 要解决的是文章的内容，整篇文章分几部分，每部分又包括多少自然段，划分的根据是各部分的实际意义。 两者是有联系的，但显然各有侧重。

二、结构化程序的优点:

下面是一个程序的示意图，方框中是一组语句。 现在，假定在 PM5 框中发现一个错误。 要予以改动。 请问这一改动会影响到哪些块? 我们不知道具体的程序，难以给出明

确的回答。 但是从图上给出的程序流程可以看出。 这一改动可能造成的影响极难预料。这样的程序谁敢轻易保证它是正确的？此程序是典型的所谓“耗子窝”程序，它的流程来回跳转非常复杂。 这主要是滥用转移语句造成的。



在 BASIC 语言中，一般是按语句标号顺序执行的，这时控制流程像一条从上到下的直线，如：

```
10  A=1
20  B=2
30  C=2
40  PRINT A, B, C
```

当遇到控制语句(如 IF 语句， GOTO 语句，GOSUB 语句。 循环语句等)时，就会改变控制流程。如果我们随心所欲地使用控制语句(特别是GOTO语句)就会使程序转来转去，无法看清程序逻辑，难以了解程序完成什么功能，怎样完成。

再者，这使从程序的某点到另一点有许多可能的路径，要跟踪控制流程几乎是不可能的。 比如上图，假定程序执行到 PM_6 框，但是它是从 PM_5，还是从 PM_2 或 PM_3 转来的？ 如果它是从 PM_2 转来的，那么又是通过哪条路到达 PM_2 的？ 在读程序时， 经常是读了最初的几句就得跳转到后面读几句， 又返回到中间再读几句，周转 4 ～ 5 次以后，就会忘记从哪里开始，以及为什么在现在这个地方？ 这时，即使认定程序是正确的，在没有周全考虑对其它地方的程序逻辑是否会产生意外影响的情况下，也不能对某点进行修改。

相反，如果有限制地使用控制语句，只使用有限的几种结构(每个结构只有一个入口、一个出口)。 采用约定的方式构筑程序，就能获得结构良好的程序。这样的程序其整体及各个部分都只有一个入口、 一个出口。并且使得程序中语句排列的静态顺序与程序执行的动态流程基本上是一致的。 如将各个结构视为基本元素 ，那么程序是从上到下无返回地顺序执行的，有如妇女“项链上的一串珍珠”。 程序执行到某个结构时，已完成的在该点之前，未完成的在该点之后。 程序员所读到的程序与所猜测的计算机在做什么之间有良好的对应性。

我们用一个实例来结束本节。

例 2.1.1 求 $Y=\begin{cases} X & X>7.6 \\ X^2 & X=7.6 \\ -X & -10.5\leqslant X<7.6 \\ \sqrt{-X} & X<-10.5 \end{cases}$

```
.....
10  IF X<7.6 THEN 40
20  IF X=7.6 THEN Y=X*X : GOTO 60
30  Y=X
35  GOTO 60
40  IF X<10.5  THEN Y=SQR(-X) : GOTO 60
50  Y=-X
60 PRINT Y
70  END
10
```

这样一个简单的问题，写出的竟是如此难读的程序！现改写如下：

程序2：

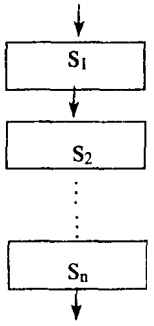
```
.....
10  IF X>7.6 THEN  Y=X:GOTO 50
20  IF X=7.6  THEN Y=X*X:GOTO 50
30  IF (-10.5=<X) AND (X<7.6) THEN Y=-X:GOTO 50
40  Y=SQR(-X)
50  PRINT Y
60  END
```

程序2的流程较程序1简单多了。特别是，如果我们将10～40行视为一个“结构”，那么10～40行的编写就是非常自然的，读程序时亦可以自觉地辨别这一结构。如果进一步将它们用固定的形式给出(如True BASIC中的CASE语句)，甚至可以“一目了然”。

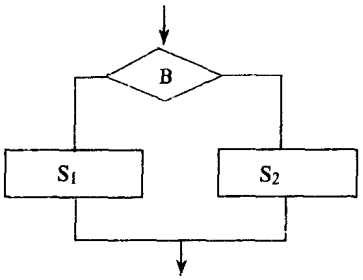
§2.2 程序的控制结构及其实现

一、程序的三种基本控制结构

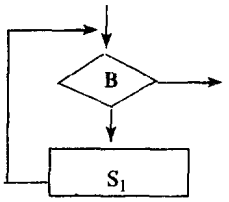
程序的基本控制结构有三种，即顺序结构、选择结构、循环结构。其流程图如下：



顺序结构



选择结构



循环结构(while do)

早在1964年数学家就证明了只要用这三种结构就足以表示出各式各种其它形式的结构。由这三种基本结构构成的程序可以处理任何复杂的问题。结构化程序就可以由这三种结构构成。

以上三种基本结构和结构化程序应满足：

① 一个入口、一个出口。

② 不包含死循环(即没有永远执行不完的循环)。

③ 无死语句。（每一部分都应有一条从入口到出口的路径通过它。）

从流程图上可以看出三种基本结构满足①。单入单出是结构化程序最本质的要求，它保证我们可以将整个程序顺序地划分成一个个结构。

二、不同的程序设计语言对控制结构的实现是不同的

对每一种结构都应当注意两件事，一是它的流程图模式(如上面三种基本结构就是由