บทที่ 5

Programming Standard

5.1 บทนำ

<u>ความจำเป็นของมาตรฐานโปรแกรม</u>

มาตรฐานของโปรแกรมเป็นสิ่งจำเป็นอย่างหนึ่งที่เราจะต้องคำนึงถึง ลักษณะของมาตรฐานโปรแกรมจะประกอบด้วยสิ่งต่อไปนี้

(1) ease of program design

(2) ease of program maintenance

(3) efficiency of computer resource utilization

5.2 เกณฑ์การพิจารณาในการเขียนโปรแกรม

การเขียนโปรแกรมขึ้นมาใช้งานนั้น เราจะมีเกณฑ์ในการพิจารณาดังนี้คือ

(1) ความถูกต้องของการแก้ปัญหา

(2) ความชาญฉลาดในการเลือกใช้ algorithm ที่มีประสิทธิภาพ

(3) การใช้เวลาถอดรหัสโปรแกรมน้อยที่สุด

(4) การประมวลผลอย่างรวดเร็ว

(5) การใช้พื้นที่ในสมองเครื่องอย่างประหยัด

(6) ง่ายในการตรวจสอบ และการทำความเข้าใจ

เกณฑ์การพิจารณาเหล่านี้จะขึ้นอยู่กับแต่ละสถานะการณ์ซึ่งแตกต่างกันไป เช่น ในบางสถานะการณ์ผู้เขียนโปรแกรมอาจจะคำนึงถึง ปัจจัยเรื่องการใช้หน่วยความจำของเครื่องเป็นอันดับหนึ่ง ส่วนปัจจัยอื่นๆเป็นอันดับรองลงไปในขณะที่ผู้เขียนโปรแกรมบางคนอาจจะคำนึงถึงความรวดเร็วในการทำงานเป็นอันดับสำคัญ ดังนั้นการจะยึดถือปัจจัยใดเป็นปัจจัยสำคัญนั้นก็จะแตกต่างกันไป

5.3 มาตรฐานของการออกแบบโปรแกรมมีรูปแบบดังนี้ คือ

(1) Use Standard Algorithm

พยายามเลือกใช้ algorithm ที่โปรแกรมเมอร์ด้วยกันนิยมเลือกใช้ในการแก้ปัญหาแต่ละรูปแบบ เช่นการเลือก algorithm ที่เป็นมาตรฐานที่ใช้กันทั่วไป

เช่นในการ search, sort เป็นต้น เพราะการใช้ common technique เหล่านี้ จะช่วยให้การบำรุงดูแลรักษาโปรแกรมต่อไปในอนาคต จะเป็นลักษณะที่เรียกว่า free maintenance นั่นก็คือใครก็ได้จะสามารถมาบำรุงดูแลรักษาได้ไม่เจาะจงต้องเป็นผู้เขียนโปรแกรมคนเดิม

(2) Use standard structure patterns

เมื่อมาถึงขณะนี้คิดว่าเราคงจะพอเข้าใจแนวคิดของ module design ได้แล้ว ดังนั้นการออกแบบโปรแกรมของแต่ละ module ย่อยก็จะยึดหลักการเดียวกัน คือ มีการใช้ mainline เพื่อที่จะประสานงานในการเรียก subroutine หรือ procedure ต่าง ๆ มาใช้งานในกิจกรรมต่าง ๆ กัน โดยที่รูปแบบของการดำเนินการในลักษณะเช่นนี้จะแตกต่างกันไปในแต่ละภาษาเช่น ภาษา COBOL จะใช้ section ทำหน้าที่ของ module entity หรือใน FORTRAN ก็จะใช้การ Call Subroutine เป็นต้น

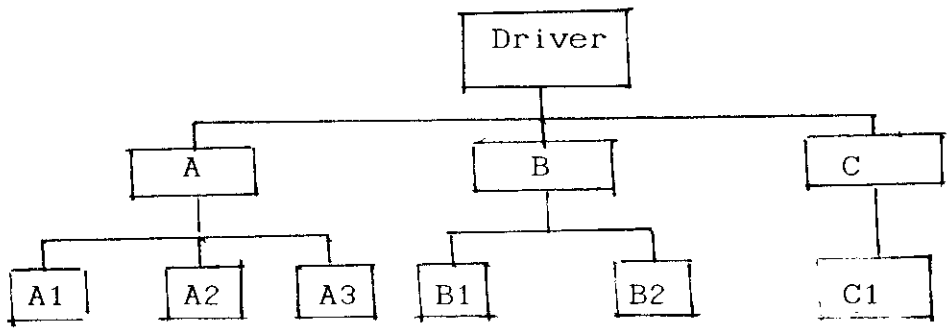
(3) Use standard Control mechanisms

ให้เลือกใช้ Control flow ซึ่งอยู่ในรูปแบบของโปรแกรมแบบโครงสร้าง (Structure programming) ซึ่งหมายถึงเลือกรูปแบบที่เป็นโครงสร้างเข้ามาทดแทนโปรแกรมแบบเดิมๆซึ่งเคยใช้กัน (มักจะเป็น program ที่มีคำสั่ง goto มาก)

หลักการออกแบบโปรแกรมทั่วไป (design Strategies)

แบ่งออกเป็น 3 แบบ คือ

1. **Top Down Design**

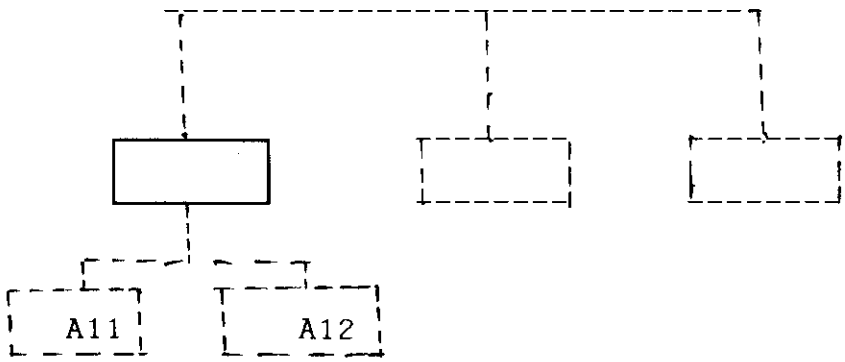


step refinement

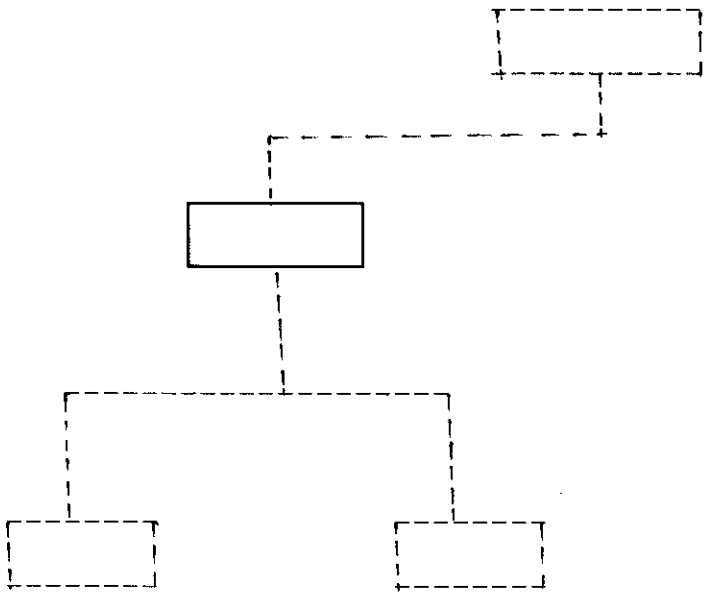
การอธิบาย PDL or Pseudo Progam

```
Driver : procedure
set up task A;
call A;
do while (condition)
  set up task B;
  call B;
  endl
set up task C;
call C;
end Driver.
```

2. Bottom Up Design



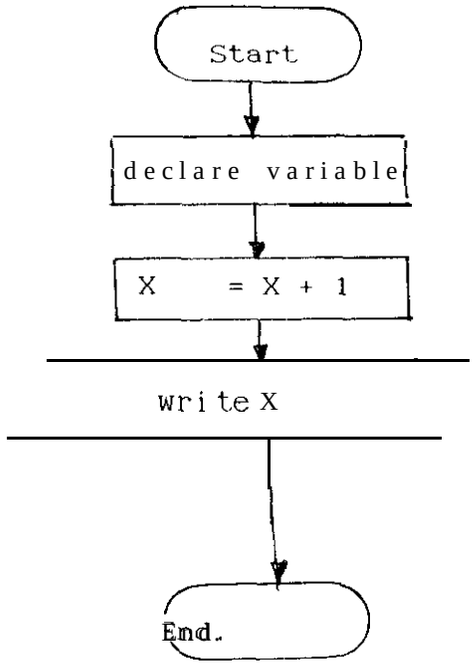
3. Sandwich



ผังโปรแกรมแบบโครงสร้างจะประกอบด้วยรูปแบบ ดังนี้

a. sequence

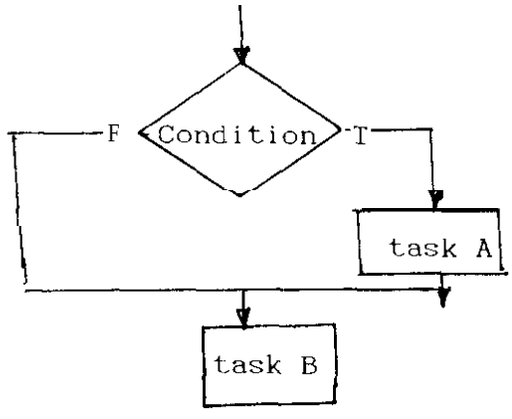
ซึ่งจะแสดงให้ดูด้วยผังโปรแกรมและโปรแกรมลำลอง ดังนี้




```
start
    declare variable
    let x = x+1
    Write X
end of program.
```

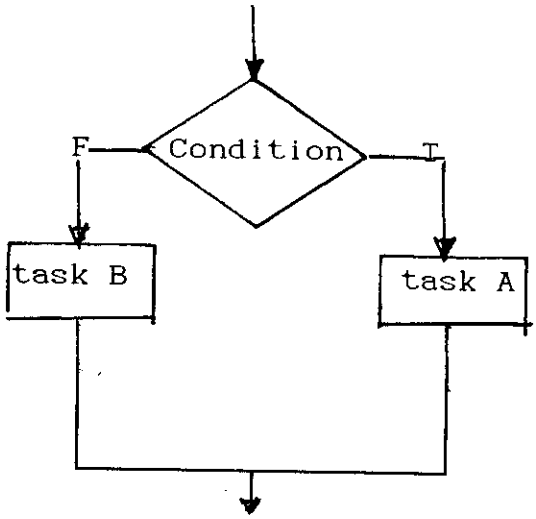
b. selection

b.1 if then




```
if (condition) true
    then do task A
end of if
do task B
```

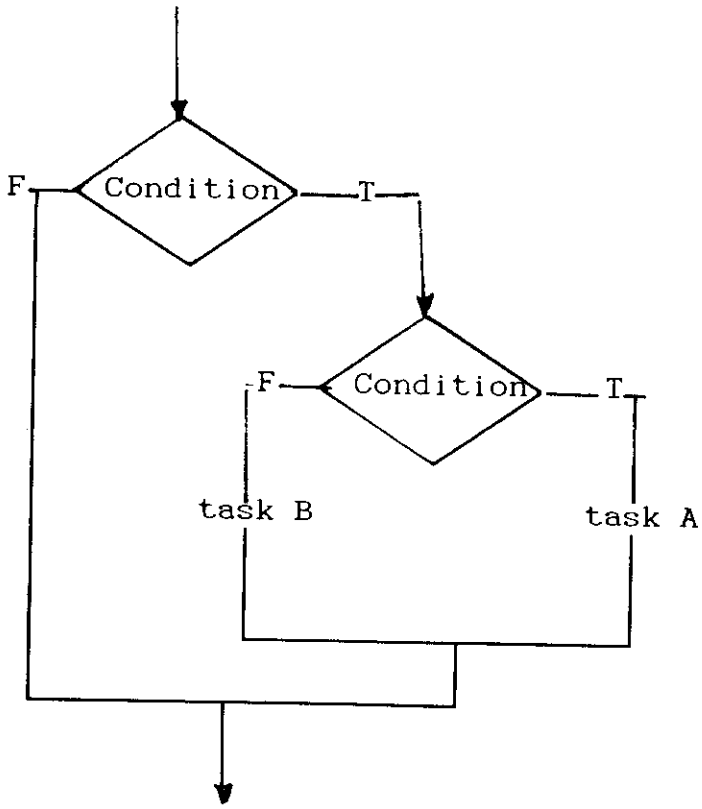
b.2 if - then - else




```
if (condition true)
then do task A
else do task B
end of if
```

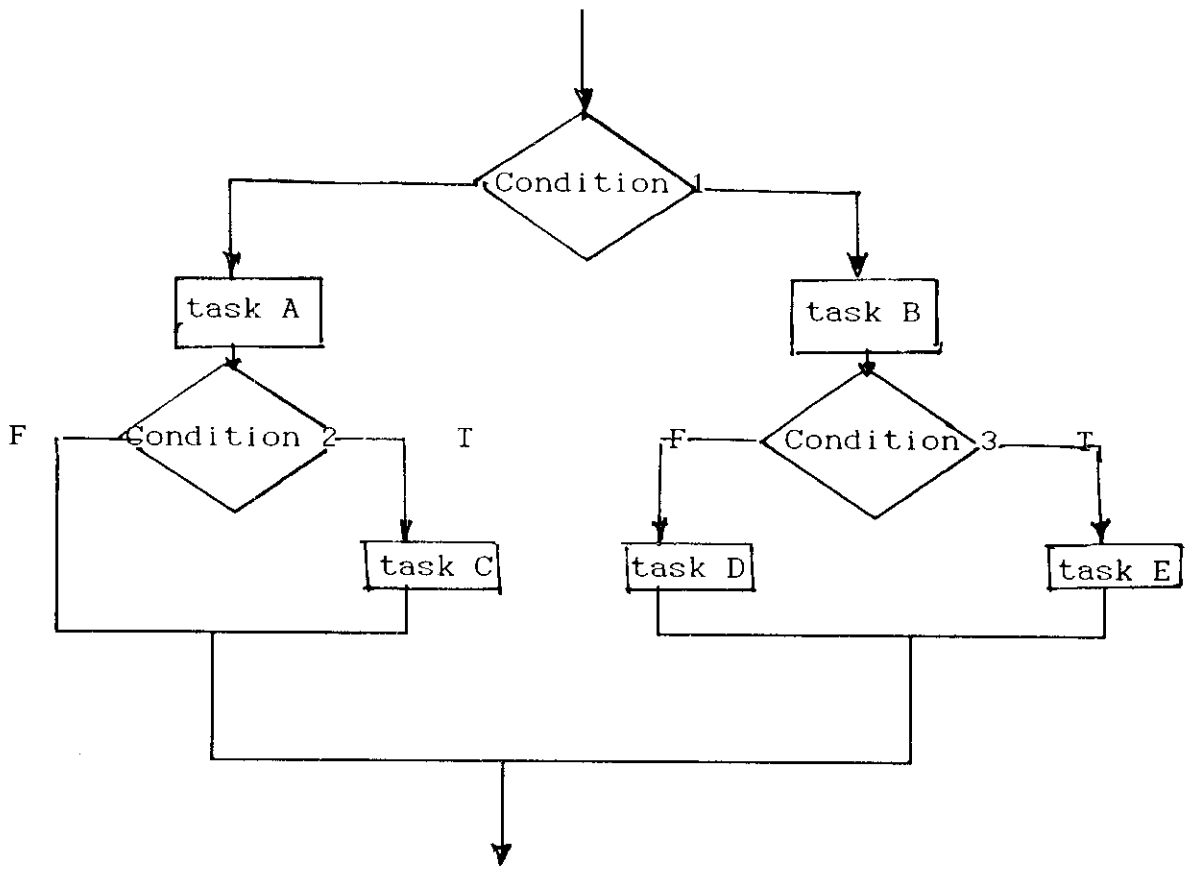
b.3 nested if

ตัวอย่างที่ 1



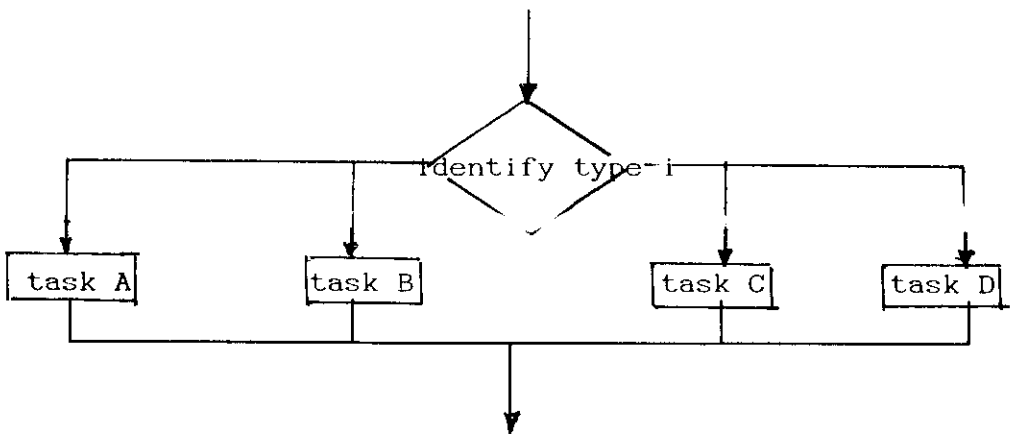
```
if (condition 1 true)
   then do if (condition 2 true)
                then do task A
                else do task B
             end of if (condition 2)
end if (condition 1)
```

ตัวอย่างที่ 2



```
if (Condition 1 true)
     then do 1
                task B
                if (Condition 3 true)
                   then do task E
                   else do task D
                end of if Condition 3
              end of do 1
     else
              do 2
               task A
                if (Condition 2 true)
                   then do task C
                end of if (Condition 2)
  end of if (Condition 1)
```

b.4 Case เป็นกรณีที่ใช้ได้ในบางภาษาเท่านั้น เช่น pascal, C แต่ในบางภาษายังไม่สามารถใช้ได้

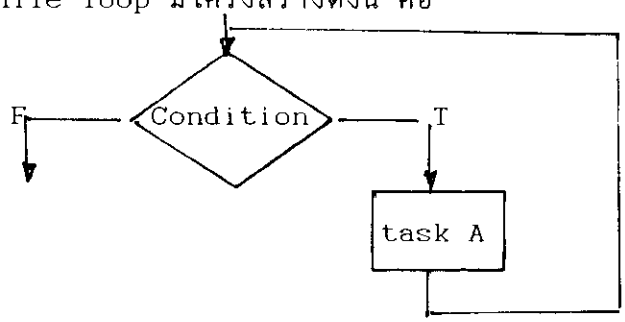


```
Case of type i
        i = 1 do task A
        i = 2 do task B
        i = 3 do task C
        i = 4 do task D
 end of Case
```

c. Repetition or Iteration

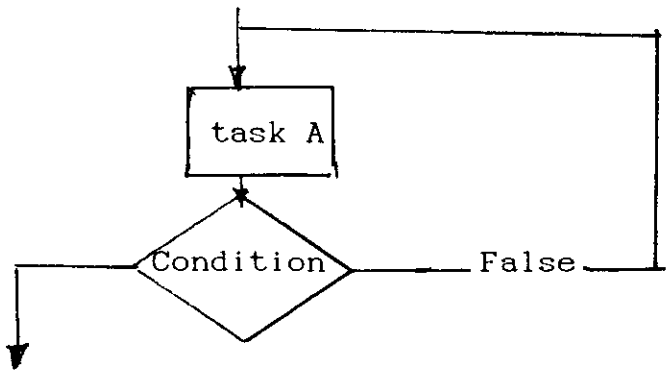
การวนเพื่อทำกิจกรรมซ้ำ ๆ นั้น เราจะมี Control flow ที่เป็นแบบโครงสร้างอยู่ 3 รูปแบบ คือ

c.1 while loop มีโครงสร้างดังนี้ คือ



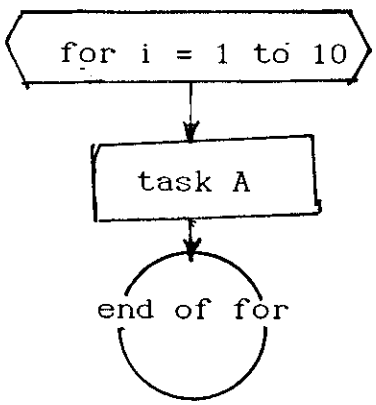

```
while (condition true) do
     begin
          task A
     end (while loop)
```

c.2 Repeat




```
repeat
        begin
            task A;
        end
  until (condition true)
```

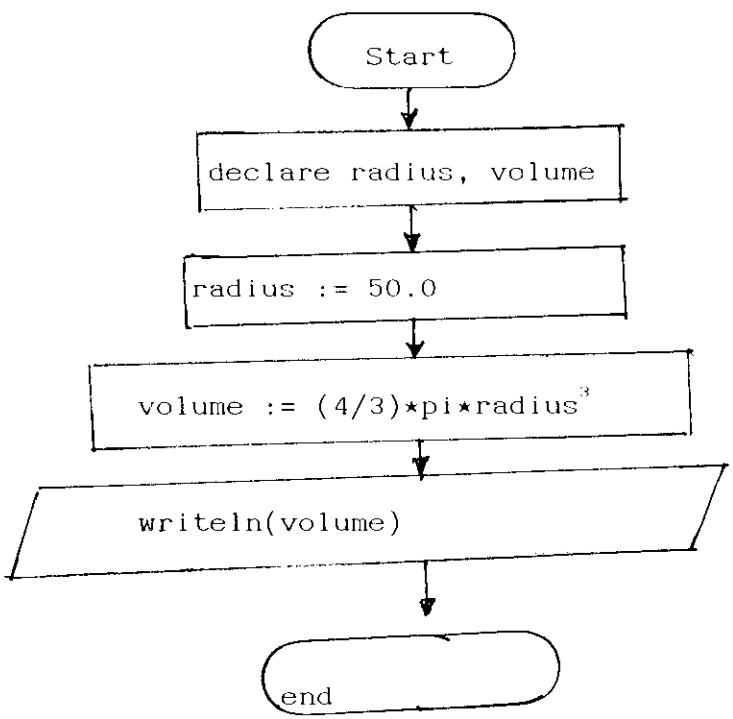
c.3 for - loop



```
for i = 1 to 10 do
    begin
        task A
    end of for
```

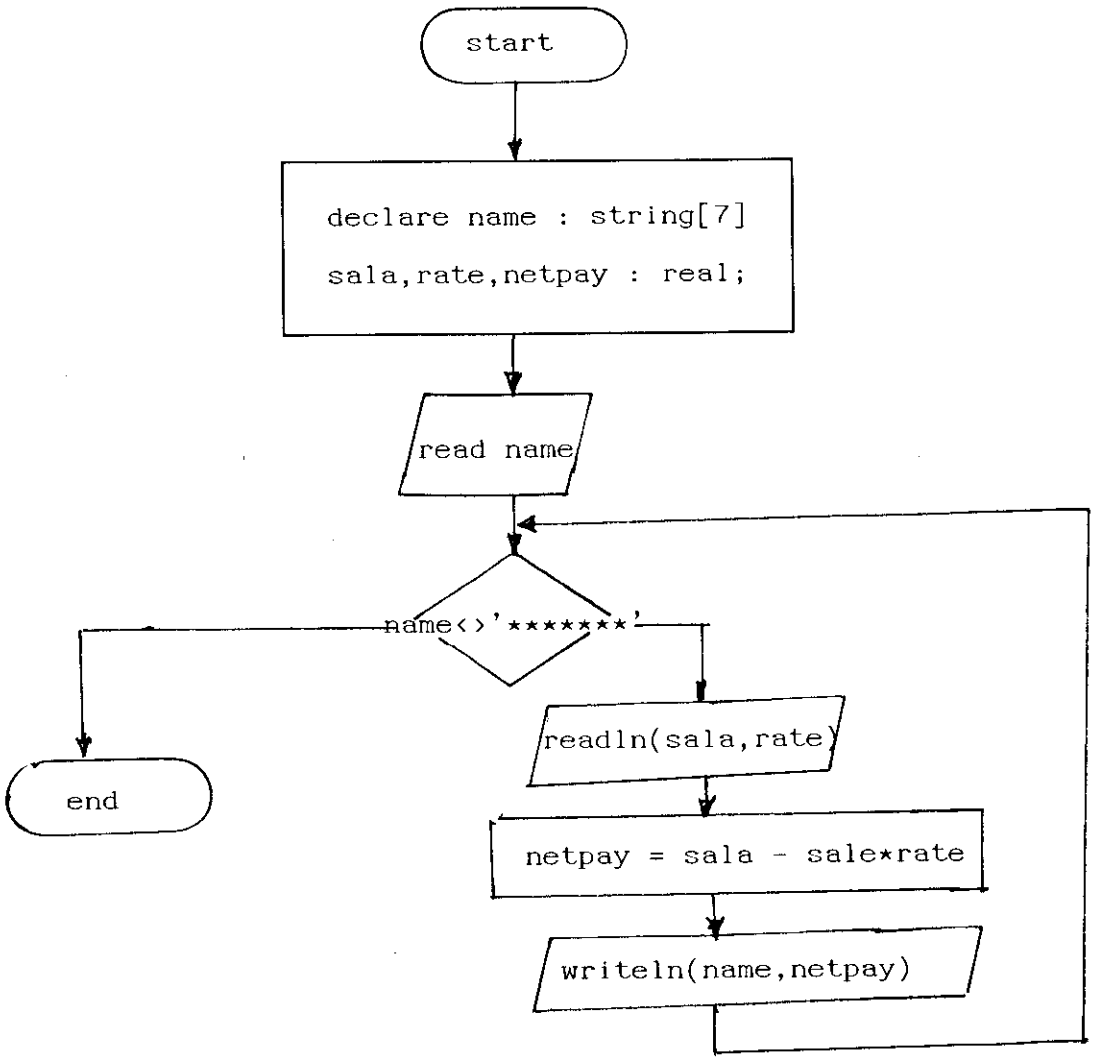
โครงสร้างการวนทั้ง 3 แบบนี้มักจะมีปรากฏอยู่ในหลาย ๆ ภาษายกเว้นรูปแบบของ repeat - until อาจจะมีปรากฏเฉพาะในบางภาษาเท่านั้น ซึ่งในกรณีที่ไม่มี repeat until ให้ใช้งาน ก็ไม่แปลกอะไรเพราะเราสามารถทดแทนได้ด้วยคำสั่ง While loop ตัวอย่างของการออกแบบโปรแกรมและการเขียนโปรแกรมโดยใช้ภาษา pascal เป็นเครื่องมือ

ตัวอย่างที่ 1



```
program ex1(output)   ;
const pi = 3.14159265  ;
var
    radius, volume : real ;
begin
    radius := 50.0;
    volume := (4.0/3.0)*pi*radius*radius*radius;
    writeln(volume)
end.
```

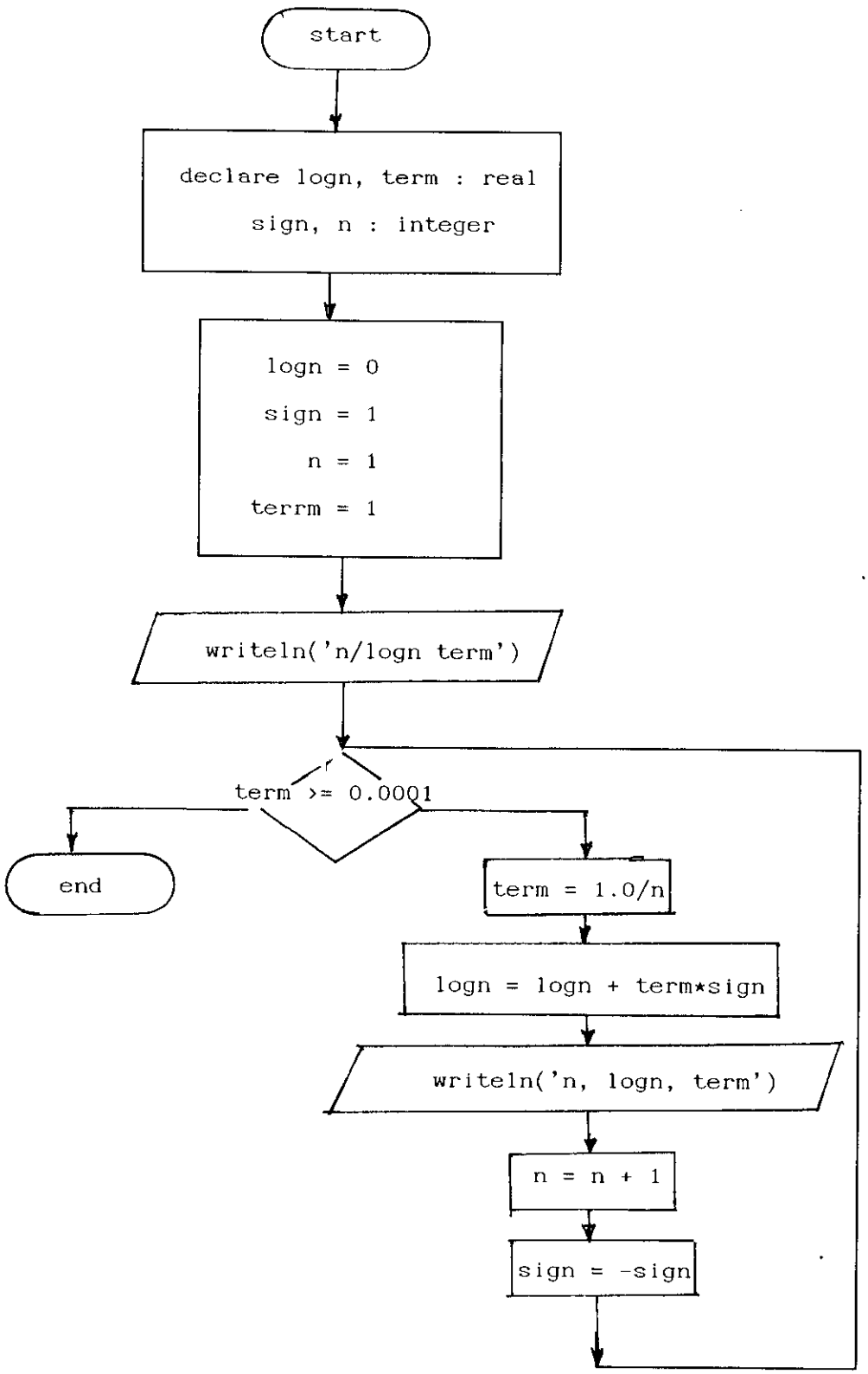
ตัวอย่างที่ 2



```
program ex2(input,output);
uses CRT;
var
     name : string[7];
     sala, rate, netpay : real;
begin
     read(name);
     while(name<>'*******') do
     begin
        readln(sala,rate);
        netpay := sala - sala*rate;
        writeln(name, netpay);
        read(name)
     end {end of while}
end. {end of main}
```

ตัวอย่างที่ 3

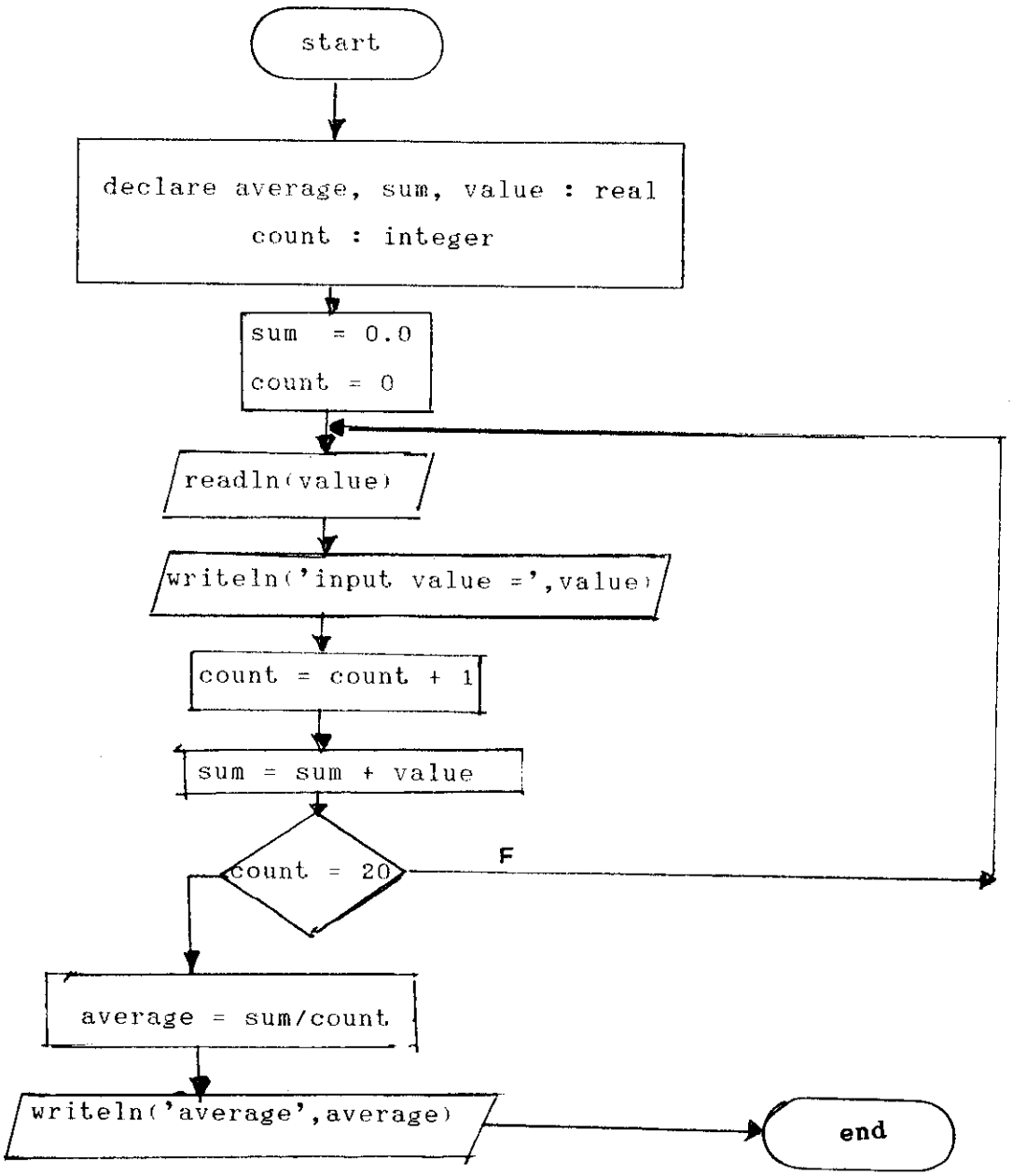
จงเขียนโปรแกรมคำนวณหาค่า ln 2 โดยที่ ln 2 = 1 - 1/2 + 1/3 - 1/4 + 1/5 - ...


start
declare logn, term : real
sign, n : integer
logn = 0
sign = 1
n = 1
terrm = 1
writeln('n/logn term')
term >= 0.0001
end
term = 1.0/n
logn = logn + term*sign
writeln('n, logn, term')
n = n + 1
sign = -sign

```
program ex3(output);
  var
       logn, term : real;
       sign, n : integer;
  begin
       logn := 0;
       sign := 1;
          n := 1;
       term := 1;
       writeln('n  logn  term');
       while(term >= 0.0001)
            do
              begin
                 term := 1.0/n;
                 logn := logn + term*sign;
                 writeln(n, logn, term);
                 n := n+1;
                 sign := -sign
       end {end of while}
  end.
```

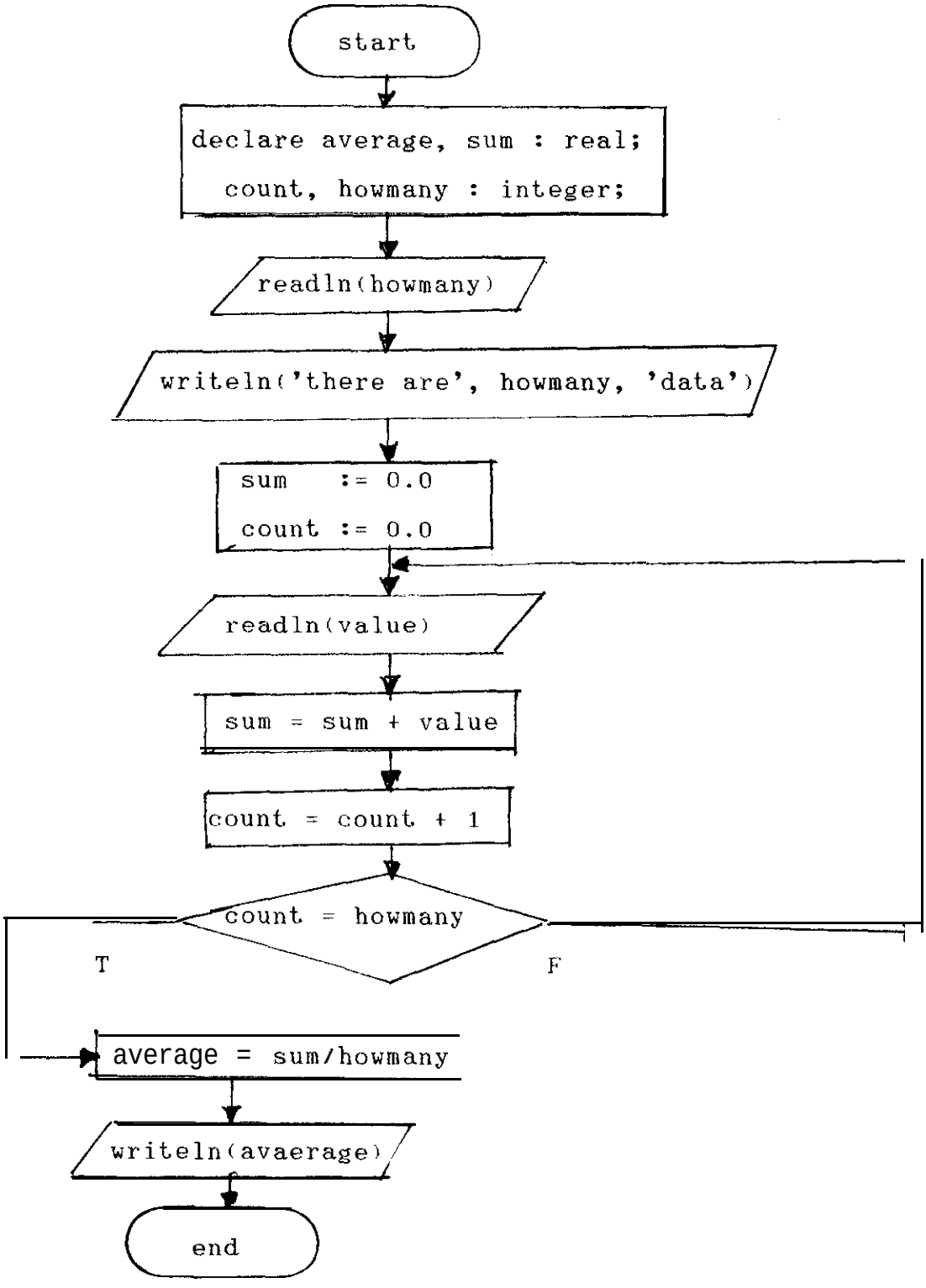
ตัวอย่างที่ 4

โปรแกรมหาค่าเฉลี่ยของตัวเลข โดยที่แต่ละจำนวนจะรับจากแป้นพิมพ์จนกว่าจะครบ 20 จำนวนที่ต้องการ



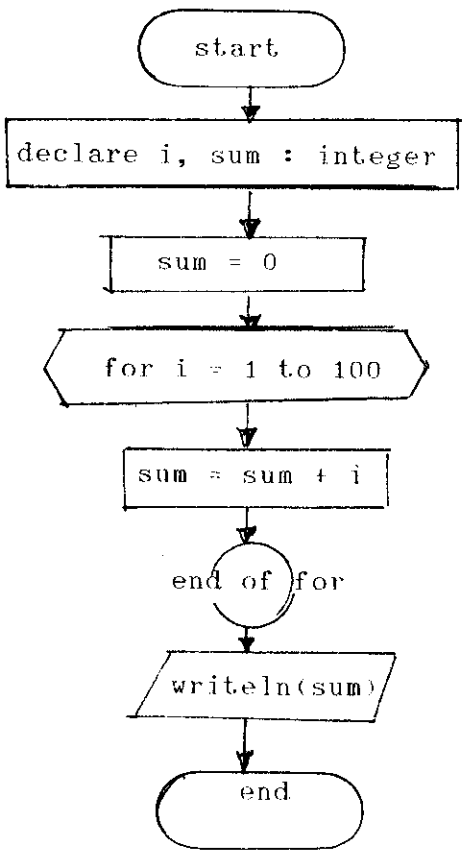
```
program ex4(input, output)
    {program to compute the average of real values read from
keyboard}
    var
         average : real;
         sum : real;
         value : real;
    begin
         sum := 0.0;
         count := 0;
         repeat
              readln(value);
              writeln('input value =', value : 15:5);
              count := count + 1;
              sum := sum + value;
         until(count = 20); {condition}
         average := sum/count;
         writeln('average =', average : 15:3)
    end.
```

ตัวอย่างที่ 5 จะเป็นไปในลักษณะคล้ายกับโปรแกรมในตัวอย่างที่ 4 จะแตกต่างกันก็ตรงที่ว่าโปรแกรมนี้จะไม่ทราบว่ามีเลขกี่จำนวนที่จะนำมาหาค่าเฉลี่ยดังนั้นเราจึงกำหนดตัวแปร howmany ขึ้นมา 1 ตัวเพื่อรับจำนวนเลขที่จะหาค่าเฉลี่ยทั้งหมดจากแป้นพิมพ์ เช่นถ้าเราป้อน howmany เข้าไป 100 ก็หมายความว่าโปรแกรมนี้จะหาค่าเฉลี่ยของเลขจำนวนจริง 100 จำนวน โดยการใช้วิธีเช่นนี้จะมีข้อดีก็คือทำให้โปรแกรมนี้ยืดหยุ่นดีกว่าโปรแกรมตัวอย่างที่ 4 ที่ยึดติดอยู่กับค่า 20จำนวน ในการทดสอบเงื่อนไขของrepeat-until


start
declare average, sum : real;
count, howmany : integer;
readln(howmany)
writeln('there are', howmany, 'data')
sum := 0.0
count := 0.0
readln(value)
sum = sum + value
count = count + 1
count = howmany
T
F
average = sum/howmany
writeln(avaerage)
end

```
program ex5(input, output);
{find the average of a set of a real data value 0. the number of
values is specified on the first value which accept from keyboard}
var
     average : real;
     count : integer;
     howmany : integer;
     sum : real;
begin
     readln(howmany);
     writeln('there are',howmany,'data value');
     sum := 0.0;
     count := 0;
repeat
     readln(value);
     writeln('data value =',value);
     sum := sum + value;
     count := count + 1;
until (count = howmany);
     average := sum/howmany;
     writeln('the average is =', average : 15:3)
end.
```

ตัวอย่างที่ 6 การหาค่าผลบวกเลขอนุกรมชุดหนึ่งจำนวน 100 เทอม โดยเลขชุดดังกล่าวมีลักษณะดังนี้คือ 1,2, 3, ... , 100 ตัวอย่างนี้เราจะใช้คำสั่ง for ด้วยเหตุเพราะเราทราบถึงจำนวนครั้งของการวน loop คือ 100 ครั้ง




```
program ex6(output);
var i, sum : integer;
begin
     sum := 0
     for i := 1 to 100 do
          begin
               sum := sum + i
          end;
     writeln('sum =',sum)
end.
```

ที่จริงในส่วนของ program คือ for นั้นเราสามารถจะละทิ้งคำว่า begin และend ในส่วนต้นและปลายของ block for ได้ ซึ่งจะเป็นรูปแบบดังนี้

```
:
for i := 1 to 100 do
     sum := sum + i;
     writeln('sum =',sum)
end.
```

แต่ด้วยเหตุที่เก็บ begin กับ end ไว้เพื่อช่วยแยกแยะให้คนอ่านโปรแกรมสามารถเข้าใจได้ง่ายว่า block ของ for นั้นเริ่มที่ใดและสิ้นสุดที่ใด ซึ่งถ้าเราเปรียบเทียบกับวิธีที่ 2 จะพบว่าแบบที่ 2 ซึ่งแม้ว่าจะทำงานในลักษณะเดียวกันแต่การอ่านทำความเข้าใจลำบากกว่าบางคนอาจจะเข้าใจค่า block ของ for นั้นคลุมถึงคำสั่ง writeln('sum =', sum) ด้วยซึ่งความจริงไม่ใช่

ตัวอย่างที่ 7 โปรแกรมหาค่า กำลังสองและกำลังสี่ของเลข 1 ถึง 5

```
program ex7(output);
var i, j ,k : integer;
begin
          writeln('square and fourth powers');
          writeln;
          for i := 1 to 5 do
               begin
                    j := i*i;
                    k := j*j;
                    writeln(i, j, k)
               end
```

ตัวอย่างที่ 8 โปรแกรมพิมพ์ข้อความว่า I am a boy จะกี่บรรทัดก็ได้ตามค่าที่รับจากแป้นพิมพ์

```
program ex8(input, output);
var j, final : integer;
begin
     write('Enter Final Value');
     readln(final);
          for j := 1 to final do
               writeln('I am a boy');
          writeln('The End')
end.
```

โปรแกรมนี้จะเห็นได้ว่าประโยคที่ทำงานใน for loop นั้นมีเพียงประโยคเดียวคือ writeln('The End') นั้นอยู่นอก loop for

ตัวอย่างที่ 9

```
program ex9(output);
var i : integer;
begin
     for i := 1 to 5 do
          writeln('I =', i);
          writeln('Final = ',i)
end.
```

ผลของปฏิบัติงานของโปรแกรมนี้จะเป็นรูปแบบนี้

```
Output
     i = 1
     i = 2
     i = 3
     i = 4
     i = 5
final = 5
```

โปรแกรมตัวอย่างที่ 9 นี้ถ้าเราเกิดเขียนผิดโดยความไม่เข้าใจ เช่นไปเปลี่ยนค่า i ดังส่วนหนึ่งของโปรแกรมต่อไปนี้

```
     :
     for i := 1 to 10 do
          begin
               writeln('i = ',i);
               i := 2; {mistake}
          end;
          :
```

ผลที่ปรากฏออกมาจากการปฏิบัติงาน จะมีรูปแบบดังนี้คือโปรแกรมจะวนไม่รู้จบสาเหตุเนื่องจากค่าของ i ไม่เป็นไปตามกฎกติกาของ for loop

เพื่อให้เข้าใจถึงกรณีผิดพลาดในการใช้คำสั่งบางอย่างใน for loop จะให้ตัวอย่างที่ 10 ต่อไปนี้

ตัวอย่างที่ 10

```
 program ex10(output);
 var i : integer;
 begin
          for i := 1 to 10 do
     begin
               writeln('i = ',i);
               readln(i)
     end
end.
```

สมมติว่า ข้อมูลที่เราป้อนเข้าไปคือ 3, 6, 9 และ 12 จะทำให้ output ปรากฏดังนี้คือ

```
i = 1
3
i = 4
6
i = 7
9
i = 10
12
i = 13
:
```

ซึ่งในสภาพดังกล่าวก็คือเกิด error กับตัวแปร i ซึ่งเป็นตัวควบคุม loop นั่นเอง

ต่อไปนี้จะเป็นตัวอย่างที่ดูเสมือนว่าจะเกิด error แต่จริง ๆ แล้วไม่เกิด แต่เราก็ไม่ควรทำเพราะทำให้เกิดความสับสนกับผู้อ่าน อย่าลืมว่า "ก่อนที่จะนำโปรแกรมไปให้เครื่องทำงานนั้น คนจะต้องอ่านโปรแกรมเสียก่อน ดังนั้นถ้าโปรแกรมใดที่มนุษย์อ่านไม่รู้เรื่องก็ควรจะหลีกเลี่ยง"

ตัวอย่างที่ 11 ตัวอย่างนี้เป็นเพียงส่วนหนึ่งของโปรแกรมที่ควบคุมด้วย for loop

```
:
 final := 3;
     for i := 1 to final do
          begin
               final := final + 1;
               writeln(i, final)
          end;
     :
```

คำตอบที่ได้จากโปรแกรมนี้จะให้ผลถูกต้องออกมาดังนี้

```
Output
     1 4
     2 5
     3 6
```

สาเหตุที่โปรแกรมนี้ทำงานได้ถูกต้อง ก็เพราะว่า คำสั่ง for นั้น ขณะที่ compile โปรแกรมได้เก็บค่าเริ่มต้นของ i คือ 1 และค่าสิ้นสุดของ i คือ 3 ไว้แล้ว ดังนั้นในขั้นปฏิบัติการซึ่งมีการเปลี่ยนแปลงค่าของ final จึงไม่มีผลต่อโปรแกรม อันนำมาซึ่งความผิดพลาด ถึงแม้ว่าโปรแกรมนี้จะไม่มีที่ผิดพลาดก็จริงแต่เราก็ควรหลีกเลี่ยงในการเขียนเพราะยุ่งยากในการทำความเข้าใจ

<u>สรุป</u> ลักษณะของการใช้คำสั่ง for ในการควบคุม loop ก็คือ

1. ก่อนจะใช้คำสั่งนี้ต้องทราบจำนวนครั้งของการวนใน loop เสียก่อน (ค่าสุดท้ายของตัวแปรควบคุม) ถ้าไม่ทราบจำนวนครั้งในการวนเราจะใช้คำสั่ง for ไม่ได้

2. การใช้คำสั่ง for นั้น ในแต่ละภาษาจะให้ลักษณะแตกต่างกันไป เช่น ภาษาฟอร์แทรน จะขยับค่าของตัวแปรควบคุมได้ในทางบวกเท่านั้น เช่น

```
do 100 i = 1, 100


 100 continue
```

ภาษาปาสคาล อนุญาตให้ใช้การขยับค่าของตัวแปรในลักษณะที่เป็นบวกก็ได้ ลบก็ได้ดังนี้

```
for i := 1 to 100 do
     begin
        :
     end
```

หรือ

```
for i := 100 downto 1 do
     begin


     end
```

ภาษาเบสิค อนุญาตให้ใช้การขยับเพิ่มขึ้นและลดลงของตัวแปรควบคุมได้ และยังให้สิทธิที่ค่าของการขยับจะเป็นเลขทศนิยมได้ด้วย ให้ดูจากตัวอย่างต่อไปนี้

```
50 FOR I = 1 TO 10 STEP .5



100 NEXT I
```

ตัวอย่างของการใช้คำสั่ง for loop ที่มักจะผิด เช่น (ใช้ภาษาปาสคาลเป็นตัวอย่าง)

1. การกำหนดรูปแบบที่ผิดกำหนด

```
for i := 10 to 3 do
     writeln('is anything happen');
:
```

เมื่อมีการเปรียบเทียบ 10 กับ 3 จะพบว่าผิดเงื่อนไขจึงไม่ทำงานใน loop

2. มีการใช้ loop วนไม่รู้จบ เช่น

```
for i := 1 to 10 do
     begin
       :
       i := 1
     end;
```

ส่วนหนึ่งของ for loop นี้จะเกิดอาการของ infinite loop เพราะค่าของ i จะถูกกำหนดให้เป็น 1 อยู่ตลอดเวลา

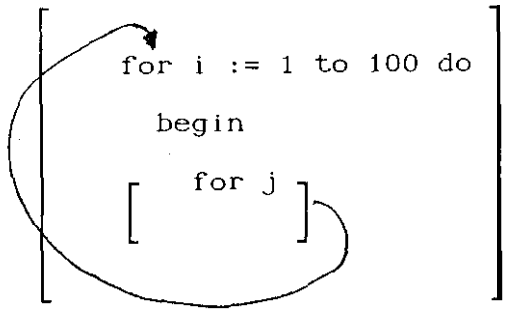
3. การใช้ nested for loop ผิดประเภท เช่น

3.1

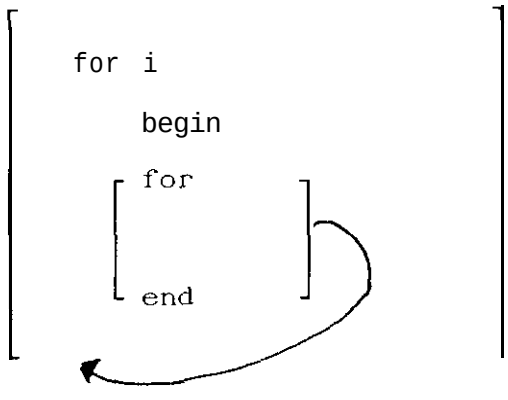
```
10 : for i := 1 to 10 do
          begin
               for j := 1 to 5 do
                    begin

                         if (condition) then go to 10 :
                    end
          end;
```

ลักษณะของตัวอย่างที่ 3.1 จะมีโครงรูปดังนี้



ซึ่งเป็นการผิดกติกาของการวนลูป ในกรณีที่จะออกจากลูปในเพื่อเข้าลูปนอกใหม่ต้องให้ถูกแบบแผนดังรูป ต่อไปนี้



นอกจากนี้การจะมาเข้าใน loop for ก็ต้องเข้าที่ส่วนหัวเท่านั้น จะมาเข้าในส่วนหนึ่งส่วนใดในกลุ่มของคำสั่ง for ไม่ได้ รูปแบบการเรียก for loop มาใช้งานจะเป็นดังนี้

```
program  exfor...

        for i := 1 to 100 do
         begin

        end
```

4. ตัวอย่างโปรแกรมต่อไปนี้จะเป็นโปรแกรมที่ชี้ให้เห็นถึงลักษณะที่ไม่ควรจะใช้กับ for loop

Which Loop Primitive Should I Use?

In this chapter we have introduced three statements for performing iteration in Pascal--the **while, repeat, and for. You** will **need** to choose the appropriate one when you begin writing your programs. In many cases it won't matter , but these two guidedlines can assist you in making the right choice.

1. The for loop is the least flexible statement and should never be used when there are two or more criteria for loop emination. You will get yourself in the following bind.

```
{loop can terminate after 100 iterations or upon encountering a negative value}
for i := 1 to 100 do
begin
    readln(x);
     if x < 0 Then ????
    else
        {process this data value}
end {for loop}
```

What do we put in for the question marks? It will have to be an unconditional branch. If we had selected a while or **repeat , the** loop termination conditions could have been **phrased as**

```
while (count <= 100) and not(x < 0) do
```

2. When choosing between the while and repeat take a carefule look at the null case -- the situation in which there are no data to process The repeat loop will always execute the loop today at least once and may not correctly handle the null case.