DO loops
in Fortran 90 Programming Language

Course Coordinator: Dr. Anil Kumar Yadav

Course: Fortran 90 Programming Language

Text book

- Computer Programming in Fortran 90 and 95 by V. Rajaraman

Outline

- Need of loops
- What is loops?
- Forms of Do loops
- Examples

Example

program 1: Finding the sum of digits of a number

```
!THE MOD FUNCTION RETURNS
LEAST SIGNIFICANT
!DIGITOF n
digit1 =MOD(n,1 0)
n=n/10
digit2=MOD(n,10)
n=n/10
digit3=MOD(n,10)
n=n/10
digit4=MOD(n,10)
n=n/10
digit5=n
sum=digit1+digi2+digit3+digit4+digit5
PRINT *,"sum of digits = ",sum
END PROGRAM
```

- Using MOD function and defining n as integer.
- Finding digit and number repeated 5 times

LOOP

- Loops are computer program which are used for repeated execution of similar things.
- As in previous program digit extraction and division of number was repeated 4 times for 5 digit number.

DO LOOP

- Do Command

```
DO
Block of statements
END DO
```

Observe that there is no way to leaving the loop.
Statements is stoppable forcefully only by switching of the computer.
No exist condition to leave the loop.

```
DO
IF(n==0)exit
Block of statements
END DO
```

Exit Condition from loop

Summing digits by DO loop

```
EXAMBPLE: summing of digit using do loop
!PROGRAM FOR SUMMING DIGITS
!USE OF DO LOOP
PROGRAM sum_digit
IMPLICIT NONE
INTEGER::n,number,digit,sum=0
PRINT*,"type number"
READ*,number
n=number
DO
  IF(n==0)EXIT
    digit=MOD(n,10)
    sum=sum+digit
    n=n/10
END DO
PRINT*,"number=",number,"sum of
digit=",sum
END
```

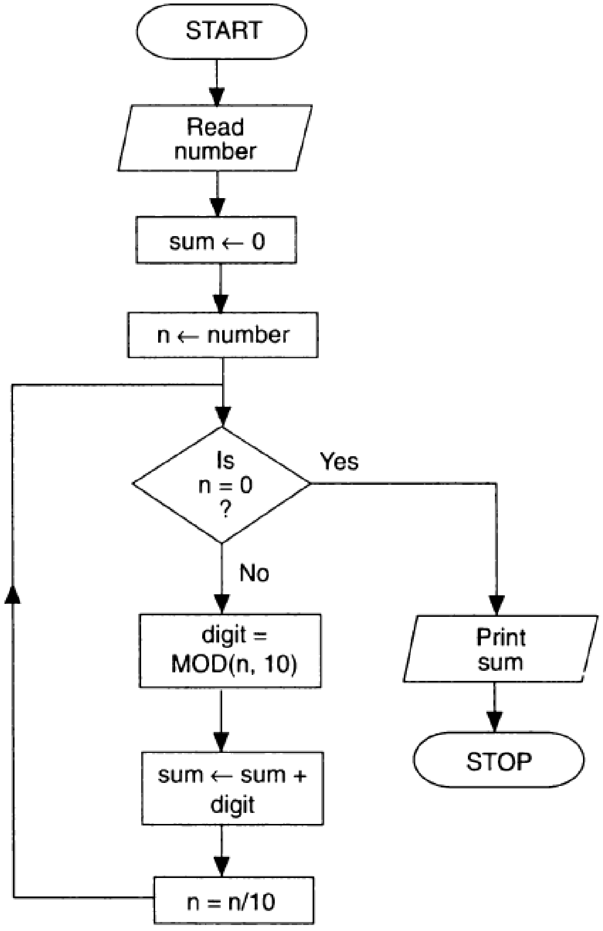


Fig. 7.1 Flowchart to sum digits of a number.

Example: Reverse order of a number

EXAMBPLE: PROGRAM FOR REVERSE ORDER OF A GIVEN NUMBER

```
!PROGRAM FOR REVERSE THE ORDER OF A GIVEN NUMBER
PROGRAM rev_order
IMPLICIT NONE
INTEGER::number,n,digit,sum1=0
PRINT*,"type the number"
READ*,number
n=number
DO
  IF(n==0)EXIT
  digit=MOD(n,10)
  n=n/10
  sum1=sum1*10+digit
END DO
PRINT*,"given number=",number,"number in reverse order",sum1
END PROGRAM
```

General form of DO Loop

The general form of the block DO loop is:

```
DO
    block of statements-1
IF (logical expression) EXIT
    block of statements-2
END DO
```

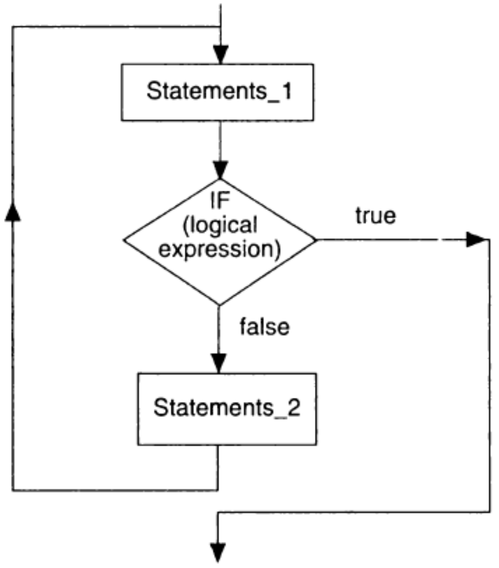


Fig. 7.2 Flowchart of block DO loop.

The DO command orders that the *block of statements enclosed by DO and END DO is to* be executed again and again as long as the *logical expression is false. When the logical expression becomes true the program jumps to the statement next to END DO.* In this case the DO loop will be executed as long as the *logical expression is false. The programmer should ensure that the logical* expression will become *true so that control leaves the loop.*

EXAMBPLE: Average height of boys and girls of the class

```
!PROGRAM TO FIND AVERAGE HEIGHT OF BOYS AND GIRLS IN THE CLASS
PROGRAM avg_height
IMPLICIT NONE
INTEGER::roll_no,total_boys=0,total_girls=0,sex_code !sex code '1'for boys and '0'for girls
REAL::height,avg_boys,avg_girls,sum1=0.0,sum2=0.0
DO
  PRINT*,"type the value of roll_no,sex_code,height"
  READ*,roll_no,sex_code,height
  IF(roll_no==0)EXIT
  IF(sex_code==1)then
    sum1=sum1+height
    total_boys=total_boys+1
  ELSE IF(sex_code==0)then
    sum2=sum2+height
    total_girls=total_girls+1
  ELSE
    PRINT*,"error in sex code"
  END IF
END DO
 avg_boys=sum1/total_boys
 avg_girls=sum2/total_girls
 PRINT*,"total no of boys=",total_boys,"average_boys_height=",avg_boys
 PRINT*,"total no of girls=",total_girls,"average_girls_height=",avg_girls
END
```

COUNT CONTROL DO LOOP

The general form of the count controlled DO loop is:

```
DO count=initial value, final value, increment
    block of statements
END DO
```

Where count, initial value, final value and increment are integer variable names.

Another valid form for the DO loop is

```
DO count=initial value, final value
    block of statements
END DO
```

In the above form *increment* is assumed to be 1.

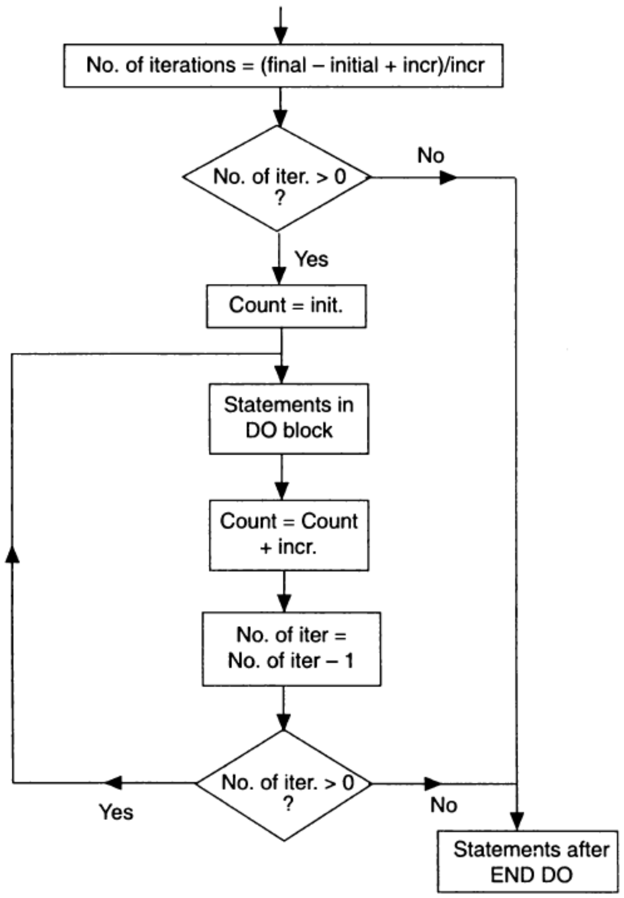
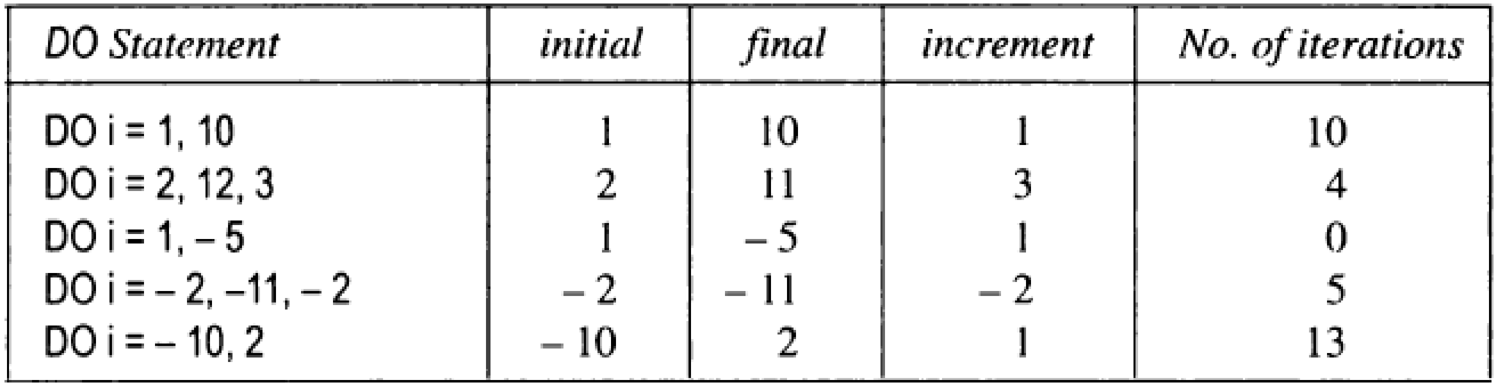


Fig. 7.3 Flowchart of a count controlled DO loop.

Number of iterations=(final value - initial value + increment)/increment

Table 7.1 Calculating Number of Iterations of DO Loop



DO Statement	*initial*	*final*	*increment*	*No. of iterations*
DO i = 1, 10	1	10	1	10
DO i = 2, 12, 3	2	11	3	4
DO i = 1, – 5	1	– 5	1	0
DO i = – 2, –11, – 2	– 2	– 11	– 2	5
DO i = – 10, 2	– 10	2	1	13

Example: Tabulation of celsius to fahrenheit conversion

```
!USE OF COUNT CONTROLLED LOOP TO TABULATE CELSIUS TO
!FAHRENHEIT CONVERSION
PROGRAM temp_conv
IMPLICIT NONE
INTEGER :: initial_celsius,final_celsius,celsius
REAL :: fahrenheit
PRINT *, "Type initial and final celsius values"
READ * ,initial_celsius,final_celsius
PRINT *,"Celsius Fahrenheit"
DO celsius=initial_celsius,final_celsius
fahrenheit = 1.8*REAL(celsius) + 32.0
PRINT *,celsius," ",fahrenheit
END DO
PRINT *,"End of conversion"
END PROGRAM temp_conv
```

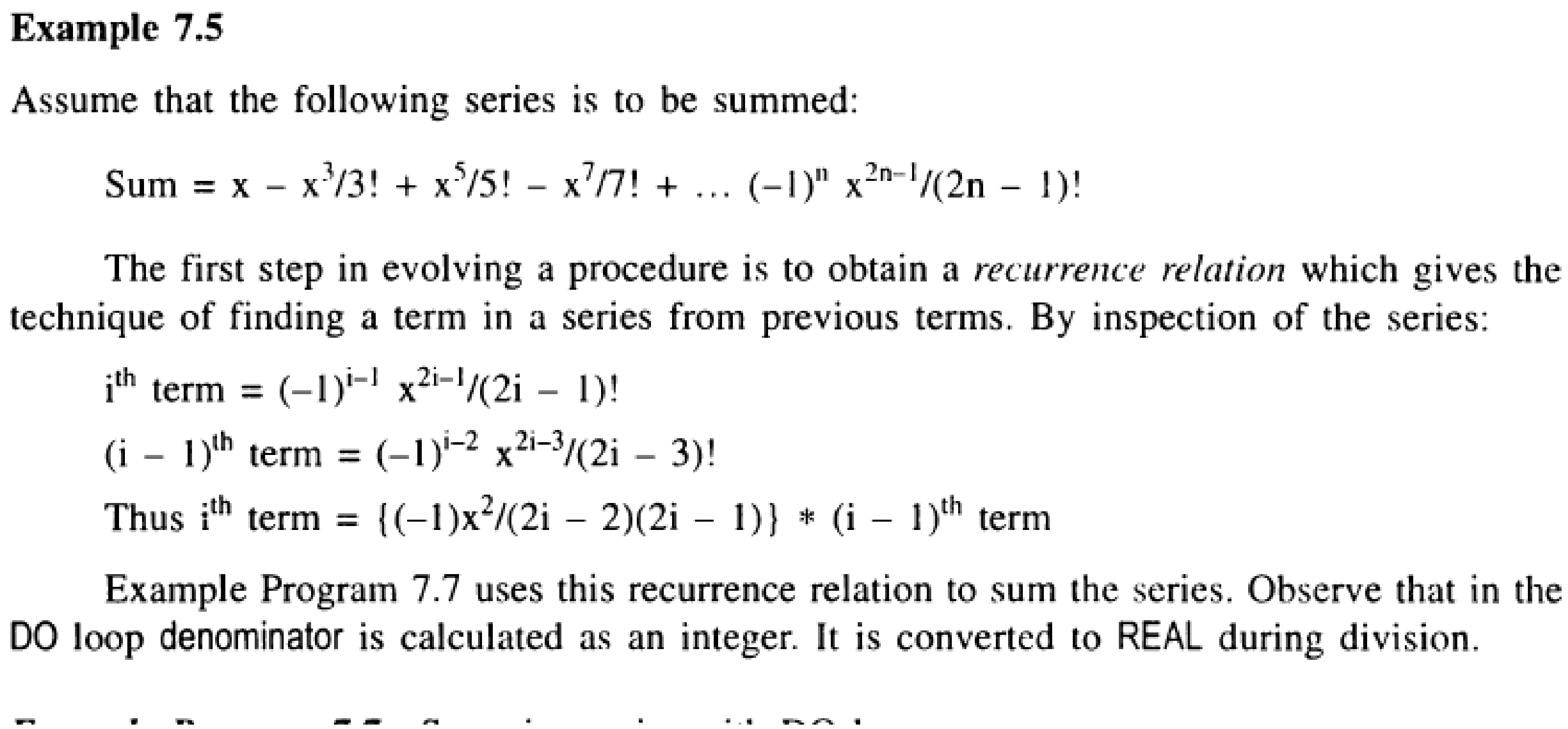
Finding negative integer

IF(number>=0)cycle

If condition is true then it transfers control to the DO statement.

```
!DATA IS A LIST OF INTEGERS
!REQUIRED TO FIND SERIAL NO.OF NEGATIVE INTEGERS
PROGRAM find_negative
IMPLICIT NONE
INTEGER :: serial,number,m,count_negative=0
PRINT *, "Type no. of integers"
READ *,m
DO serial=1 ,m
PRINT *,"Type integer"
READ * ,number
PRINT *,"Number =",number
IF(number >= 0) CYCLE
count_negative = count_negative + 1
PRINT *,"Serial =",serial," ",number
END DO
PRINT *,"Number of negative numbers =",count_negative
END PROGRAM find_negative
```

Summing Series with DO loop



Example 7.5

Assume that the following series is to be summed:

$$\text{Sum} = x - x^3/3! + x^5/5! - x^7/7! + \ldots (-1)^n x^{2n-1}/(2n - 1)!$$

The first step in evolving a procedure is to obtain a *recurrence relation* which gives the technique of finding a term in a series from previous terms. By inspection of the series:

i^{th} term $= (-1)^{i-1} x^{2i-1}/(2i - 1)!$

$(i - 1)^{th}$ term $= (-1)^{i-2} x^{2i-3}/(2i - 3)!$

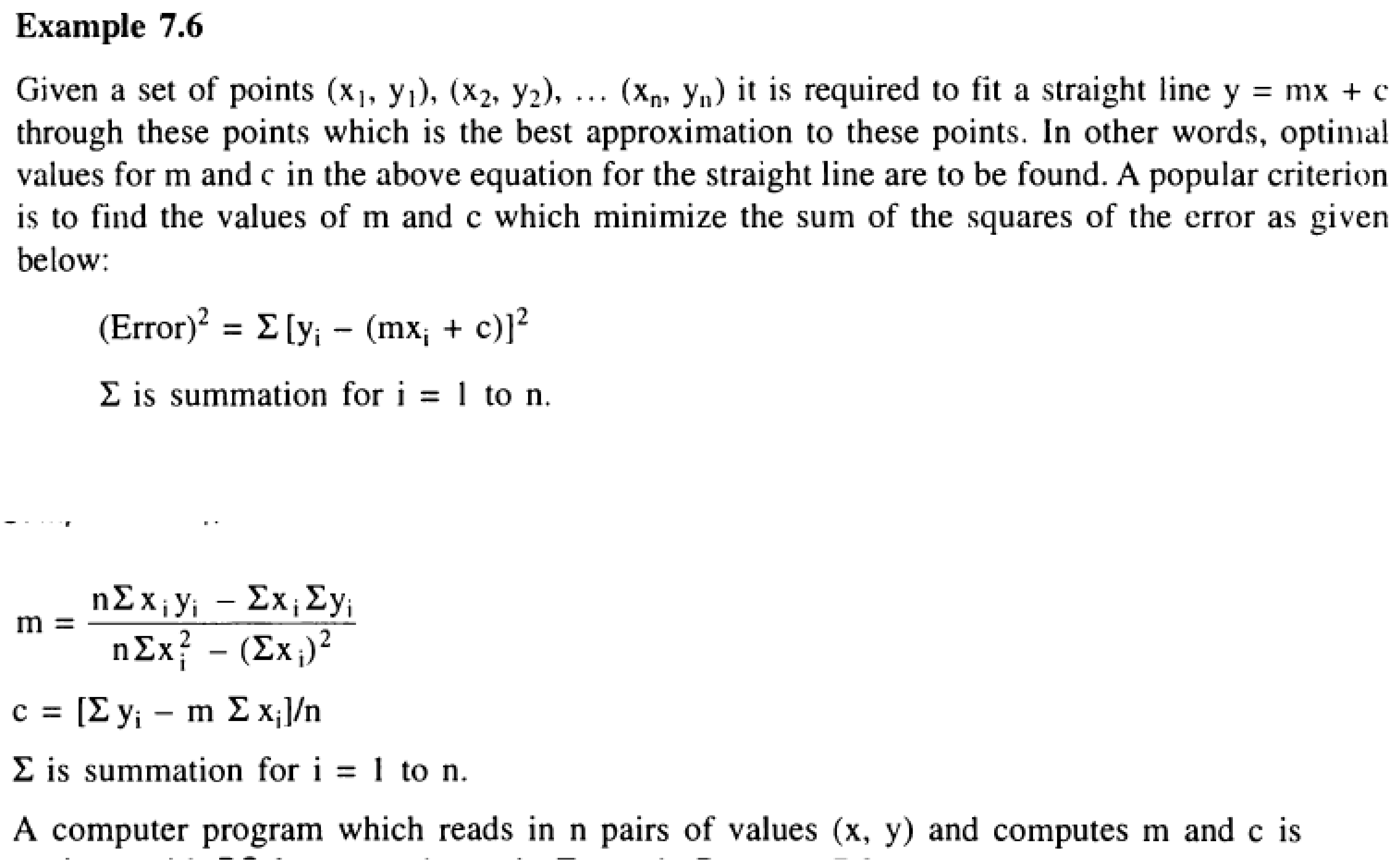
Thus i^{th} term $= \{(-1)x^2/(2i - 2)(2i - 1)\} * (i - 1)^{th}$ term

Example Program 7.7 uses this recurrence relation to sum the series. Observe that in the DO loop denominator is calculated as an integer. It is converted to REAL during division.

EXAMPLE: Finding sum of series

```
!SUMMING OF SERIES WITH DO LOOP
PROGRAM sum_series
IMPLICIT NONE
REAL :: x,term,sum
INTEGER :: i,n,denominator
PRINT *, "Type values of x and n"
READ *,x,n
PRINT *,"x =",x," n =",n
sum = x
term = x
DO i=2,n
denominator=(2*i-2)*(2*i-1)
term=term*(-x)*(x)/REAL(denominator)
sum=sum + term
END DO
PRINT *,"SL!m =",sum
END PROGRAM sum_series
```

Least Square fitting parameters



Example 7.6

Given a set of points (x_1, y_1), (x_2, y_2), … (x_n, y_n) it is required to fit a straight line $y = mx + c$ through these points which is the best approximation to these points. In other words, optimal values for m and c in the above equation for the straight line are to be found. A popular criterion is to find the values of m and c which minimize the sum of the squares of the error as given below:

$$(\text{Error})^2 = \Sigma [y_i - (mx_i + c)]^2$$

Σ is summation for i = 1 to n.

$$m = \frac{n\Sigma x_i y_i - \Sigma x_i \Sigma y_i}{n\Sigma x_i^2 - (\Sigma x_i)^2}$$

$$c = [\Sigma y_i - m\,\Sigma x_i]/n$$

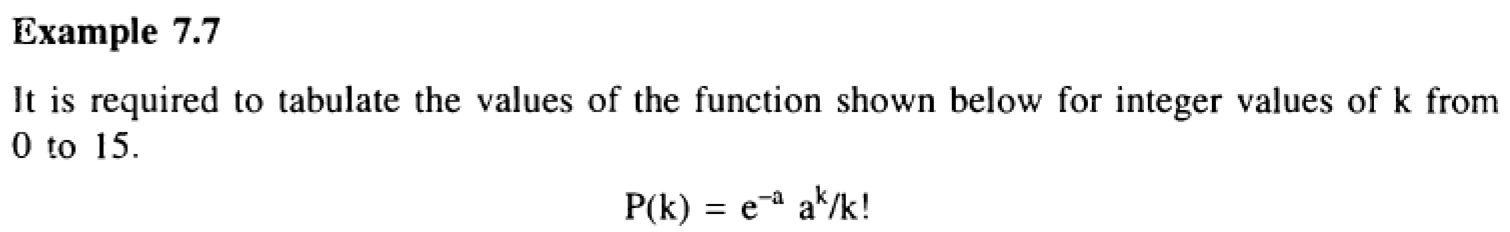
Σ is summation for i = 1 to n.

A computer program which reads in n pairs of values (x, y) and computes m and c is

EXAMPLE: Finding sum of series

```
!x,y COORDINATES .THE STRAIGHT LINE IS y=mx+c
PROGRAM straight_line
IMPLICIT NONE
INTEGER :: i,n
REAL :: sum_x=0,sum_y=0,sum_xy=0,sum_xsq=0,x,y,numerator,denominator,m,c
!READ THE NUMBER OF POINTS n
PRINT *, "Type no. of points"
READ *,n
DO i=1,n
PRINT *, "Type values of x and y"
READ *,x,y
PRINT *,"x =",x,"Y =",y
sum_x = sum_x + x
sum_y = sum_y + y
sum_xy = sum_xy + x*y
sum_xsq = sum_xsq + x*x
END DO
numerator = REAL(n)*sum_y - sum_x*sum_y
denominator = REAL(n)*sum_xsq - sum_x*sum_x
m = numerator/denominator
c = (sum_y - m*sum_x)/REAL(n)
PRINT *,"Equation of straight line is"
PRINT *,"y =",m,"x +",c
END PROGRAM straight_line
```

Program for Poisson function



Example 7.7

It is required to tabulate the values of the function shown below for integer values of k from 0 to 15.

$$P(k) = e^{-a}\, a^{k}/k!$$

P(k)=exp(-a) a/k a(k-1)/(k-1)!=a/k P(k-1)

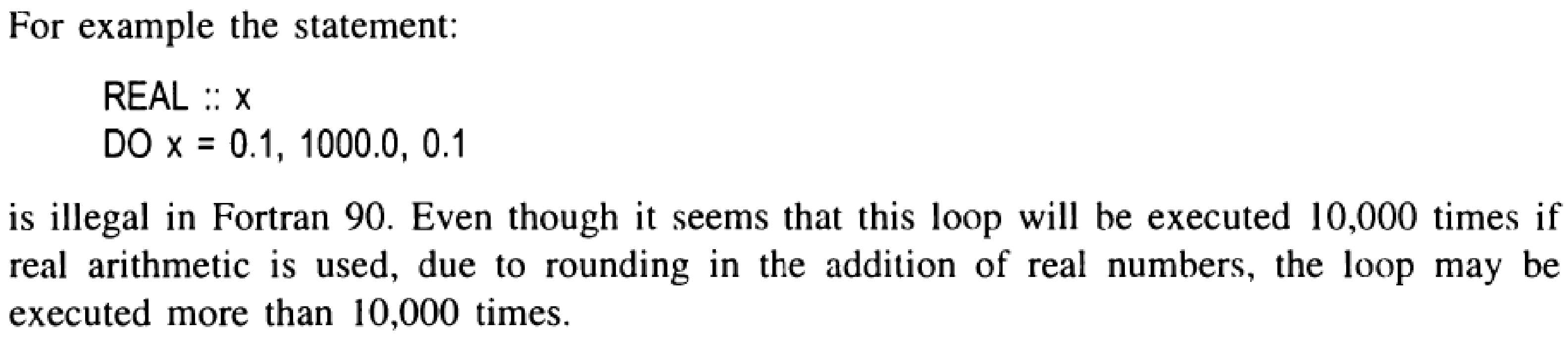
The function exp(-a) which is independent of k is computed outside the DO loop and only the terms dependent on k are computed inside the loop.

EXAMPLE: Tabulation of Poisson function

```
!POISSON FUNCTION TABULATION
PROGRAM poisson
IMPLICIT NONE
INTEGER:: k
REAL :: a,pois
PRINT *, "Type value of a"
READ *,a
PRINT *,"a= ",a
pois = EXP(-a)
k=0
PRINT *,"k poisson(k)"
PRINT *,k," ",pois
DO k=1,15
pois = pois*a/REAL(k)
PRINT *,k," ",pois
END DO
END PROGRAM poisson
```

Rules to be remembered in writing DO Loops

- Rule 1: The DO loop indices should not be reals. Only integers are allowed.


For example the statement:

```
REAL :: x
DO x = 0.1, 1000.0, 0.1
```

is illegal in Fortran 90. Even though it seems that this loop will be executed 10,000 times if real arithmetic is used, due to rounding in the addition of real numbers, the loop may be executed more than 10,000 times.

- Rule 2: Enclosed within a DO loop there may be other DO loops. That is. the DO to END DO blocks of latter DOs must be enclosed within the DO to END DO block of the first one, A set of DOs satisfying this rule is called nested DOs.

Outer loop: DO i = 1, 10

Inner loop: DO j = 2, 20

END DO inner loop

END DO outer loop

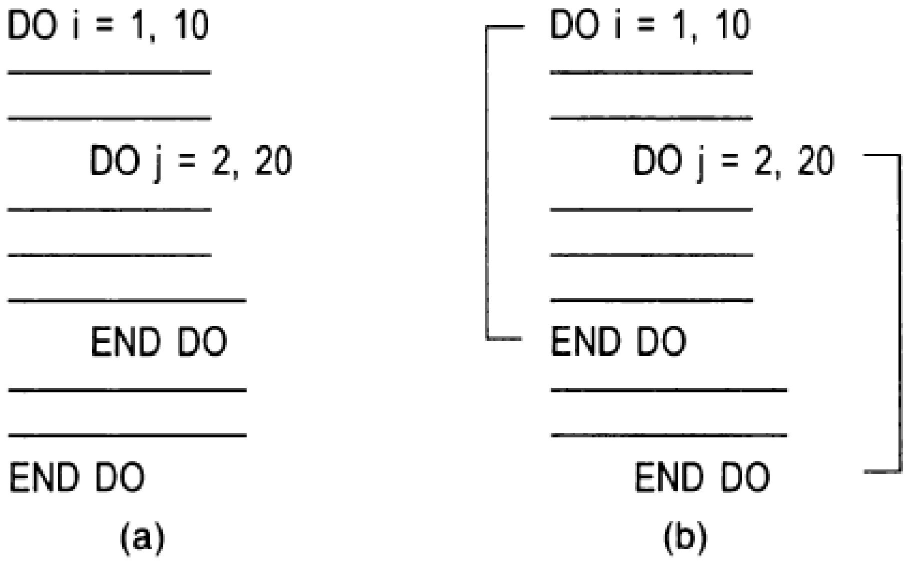
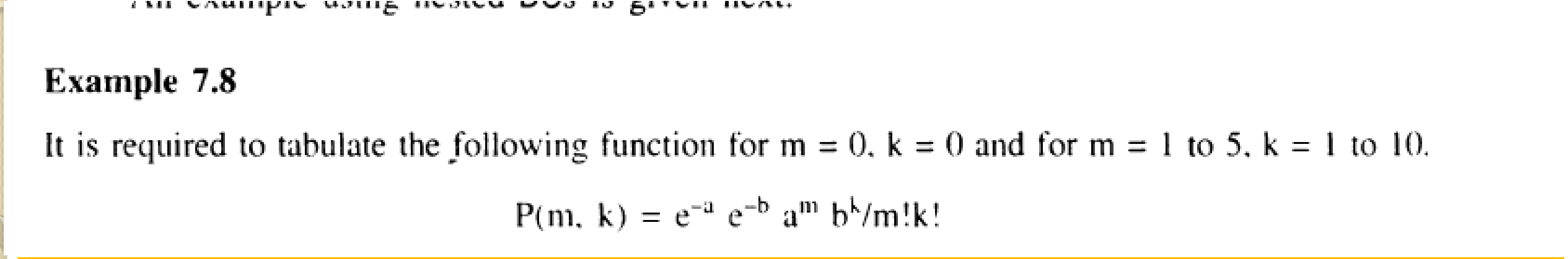


Fig. 7.4 Legal and illegal nested DO loops.



An example using nested DOs is given next.

Example 7.8

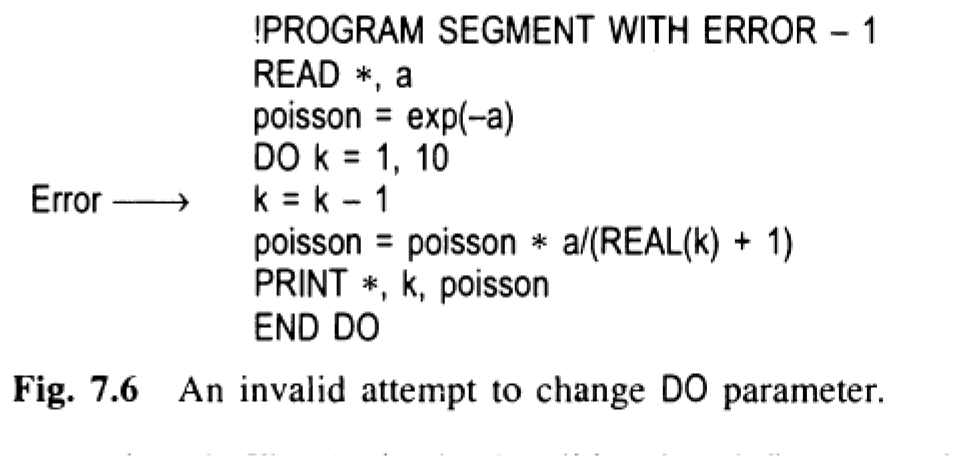
It is required to tabulate the following function for m = 0, k = 0 and for m = 1 to 5, k = 1 to 10.

$$P(m, k) = e^{-a} e^{-b} a^m b^k/m!k!$$

EXAMPLE: Two dimensional Poisson function

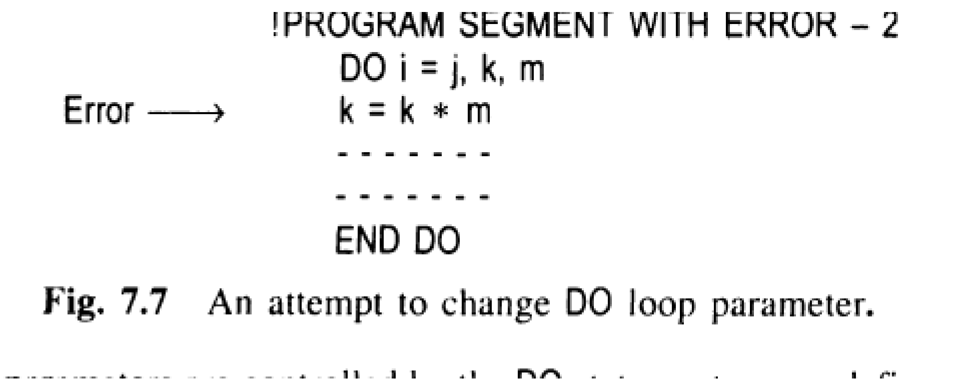
```
!TABULATING TWO DIMENSIONAL POISSON FUNCTION
PROGRAM poisson_2
IMPLICIT NONE
INTEGER :: k,m
REAL:: poisson,a=0.1 ,b=0.1 ,poisson_x
!CALCULATE p(O,O)
poisson = EXP(-a)*EXP(-b)
k = 0;m = 0
PRINT *,"k =",k," m=",m," poisson=",poisson
outer: DO m=1 ,5
poisson = poisson * a/REAL(m)
poisson_x = poisson
inner: DO k=1, 10
poisson_x = poisson_x*b/REAL(k)
PRINT *,"k =",k," m=",m," poisson=",poisson_x
END DO inner
END DO outer
END PROGRAM poisson_2
```

- Rule 3: The DO loop parameters count, initial-value, final-value and increment should not be redefined by statements within the DO loop block.



```
                 !PROGRAM SEGMENT WITH ERROR – 1
                 READ *, a
                 poisson = exp(–a)
                 DO k = 1, 10
Error ——→        k = k – 1
                 poisson = poisson * a/(REAL(k) + 1)
                 PRINT *, k, poisson
                 END DO
```

Fig. 7.6 An invalid attempt to change DO parameter.



```
                 !PROGRAM SEGMENT WITH ERROR – 2
                    DO i = j, k, m
Error ——→           k = k * m
                    - - - - - - -
                    - - - - - - -
                    END DO
```

Fig. 7.7 An attempt to change DO loop parameter.

EXAMPLE: Counting high Marks and finding average

```
!HIGH MARKS AND AVERAGE
PROGRAM marks_90
IMPLICIT NONE
INTEGER ::
roll_no,marks,count=0,high_count=0,sum_marks=0,avg_marks
PRINT *,"List of roll numbers with marks> 90"
DO
READ *,roll_no,marks
IF(roll_no < 0) EXIT
sum_marks = sum_marks + marks
count = count +1
IF(marks <=90) CYCLE
PRINT *,"Roll no =",roll_no," marks =",marks
high_count = high_count +1
ENDDO
avg_marks = sum_marks/count
PRINT *,"No.of students with marks> 90 = ", high_count
PRINT *,"Total no.of students =",count
PRINT *,"Average marks =",avg_marks
END PROGRAM marks_90
```

EXERCISE

- Write program for the problems given at the end of chapter 6 in reference book.

Thank you