

CS199 Computer programming



Spring 2018

Lecture 6

Loops

Objectives

In this chapter, you will:

- Learn about repetition (looping) control structures
- Explore how to construct and use deterministic and non-deterministic loops
- Examine `Exit` and `cycle` statements
- Discover how to form and use nested control structures

Loops (Iterative Constructs)

- Many applications require certain operations to be carried out more than once. Such situations require repetition in control flow.
- Loops allow a code segment to be executed many times.
- Loop construct in FORTRAN
 - Deterministic do loop – repeat a fixed number of times
 - Non-deterministic do loops – repeat until some criterion is met.

Deterministic do loop

- **Deterministic do** loop is used when the number of times to be repeated is fixed/known

- **syntax**

```
do index = istart, iend[, incr]
  statement block
```

```
end do
```

- The index must be a named scalar integer variable
- istart, iend, incr can be constants, variables, or expression of integer.
- If incr is not specified, incr = 1.
- incr can be positive or negative

- **semantics**

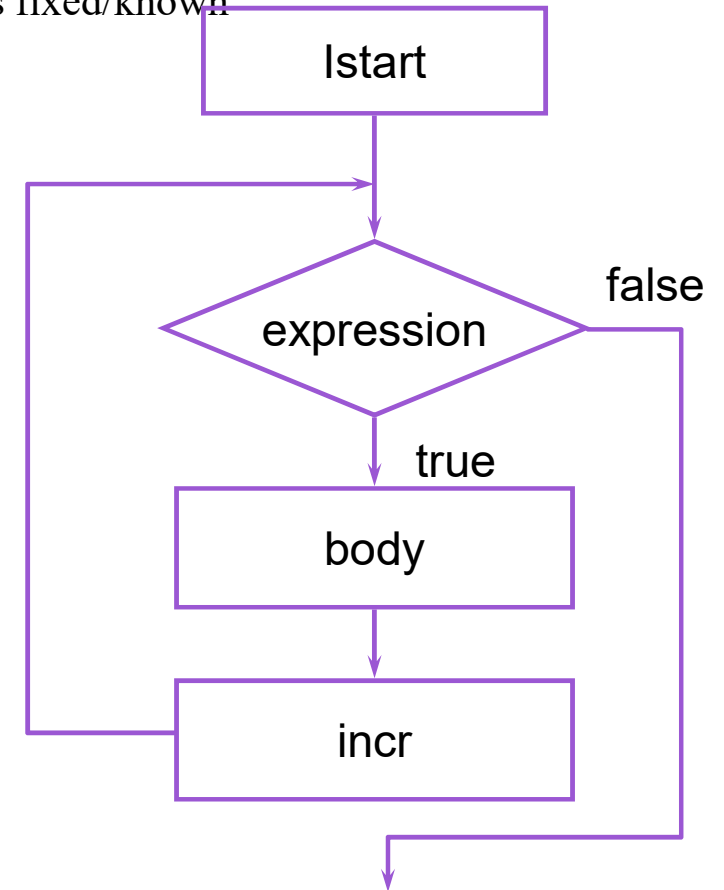
- execute **istart**
- evaluate **expression**, if true: iterate
- iteration:
 - execute **body**
 - execute **incr**
 - repeat expression evaluation

- **example**

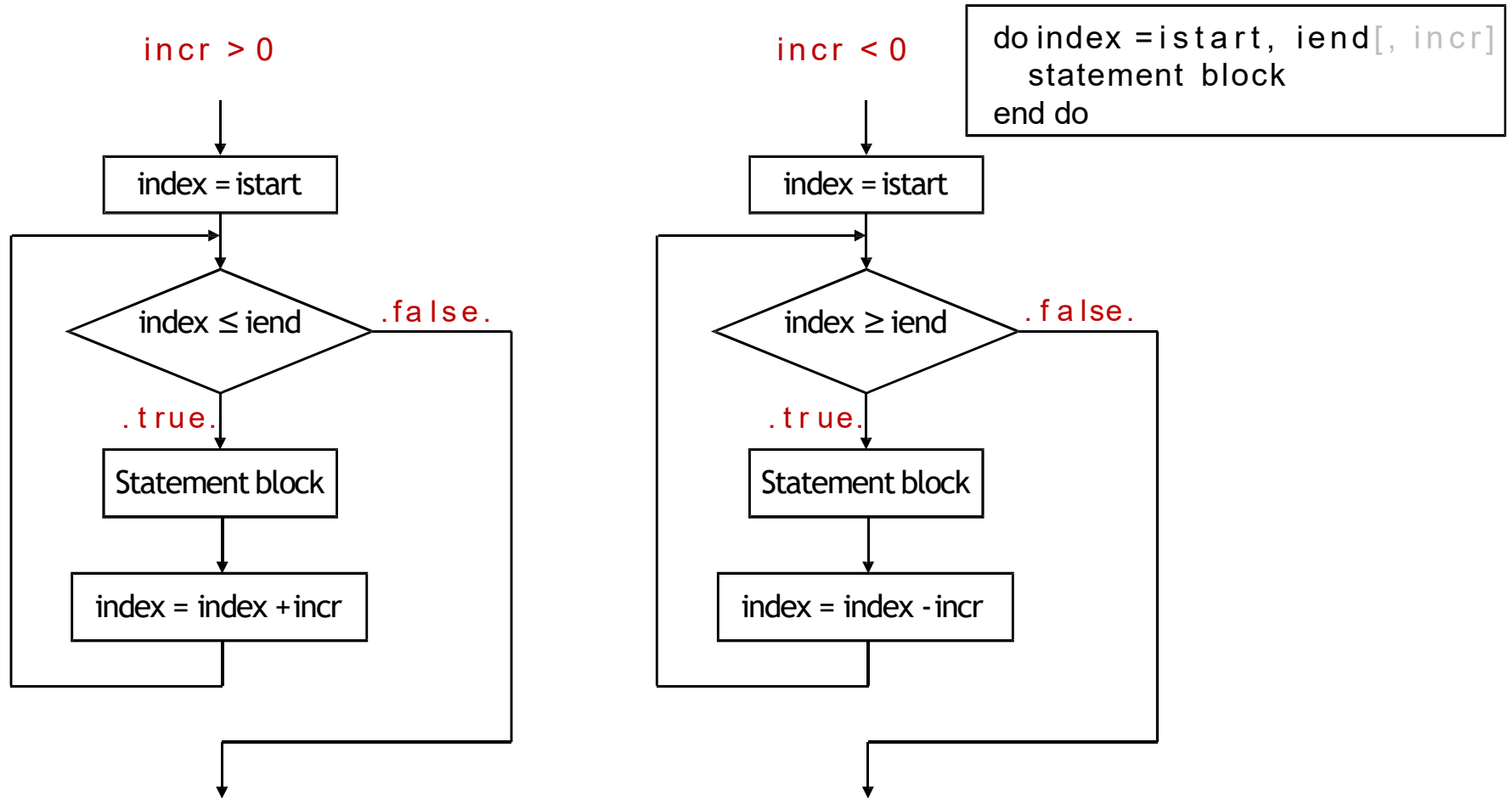
```
do i= 1, 5,2
  x=x+1
End do
```

```
Loop iterates
3 times as
i=1 , 3, 5
```

```
X=1
X=2
X=3
```



Deterministic do loop

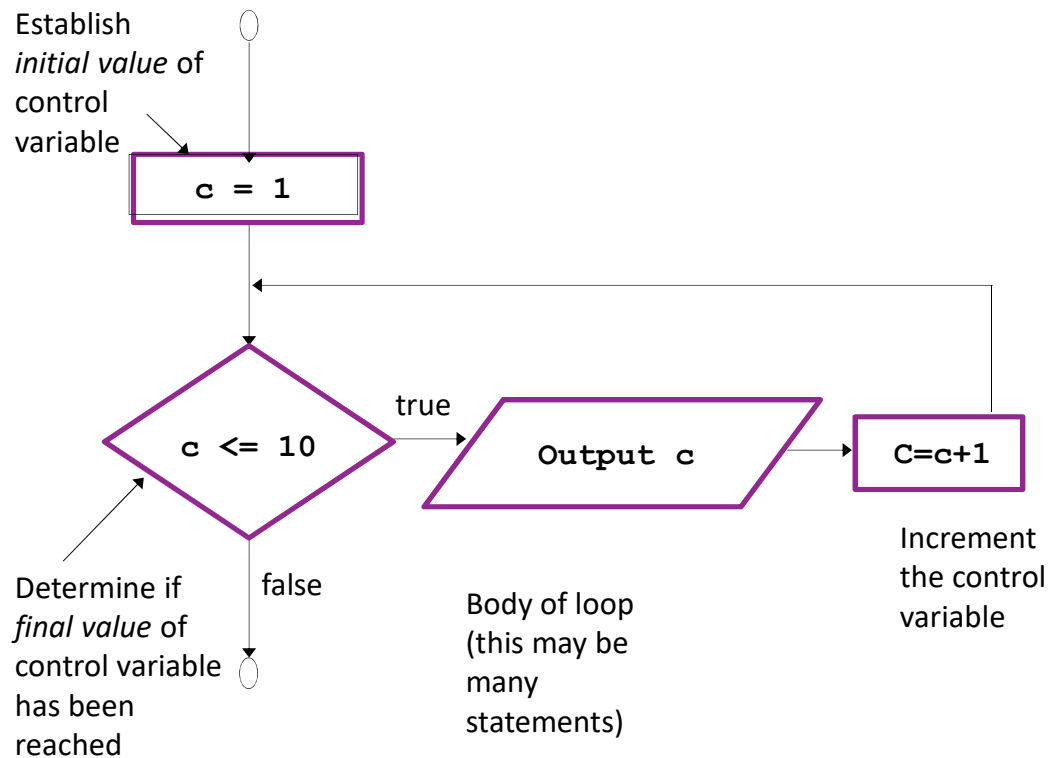


Examples Using the Deterministic do Structure

- Vary control variable from 1 to 30 in increments of 2
DO i = 1, 30, 2
... ! i takes the values 1,3,5,7,...,29 , 15 iterations
END DO
- Vary control variable from 30 to 1 in decrements of -2
DO j = 30, 1, -2
... ! j takes the values 30,28,26,...,2 , 15 iterations
END DO
- There is no iterations
DO k = 30, 1, 2
... ! 0 iterations , loop skipped
END DO
- Vary control variable from 1 to 30 in steps of 1
DO l = 1, 30
... ! l takes the values 1,2,3,...,30 , 30 iterations
END DO

Example 1

- Print the integer numbers 1 to 10 on screen using for loop statement



Program counter

! function main begins program execution

integer :: c

! Initialization, expression and incrementing
! are all included in the Do structure header.

Do c = 1 , 10 , 1

write (*,*) c

End do

end

```
Select Project Console
1
2
3
4
5
6
7
8
9
10
Press any key to continue . . .
```

Factorial example

$n!$ (n factorial) is defined as the product of all the integers from 1 to n .

$$n! = 1 * 2 * 3 * \dots * n$$

Example: $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$

```
integer :: no, f=1, n
```

```
write (*,*) "Enter positive integer:"  
read (*,*) no
```

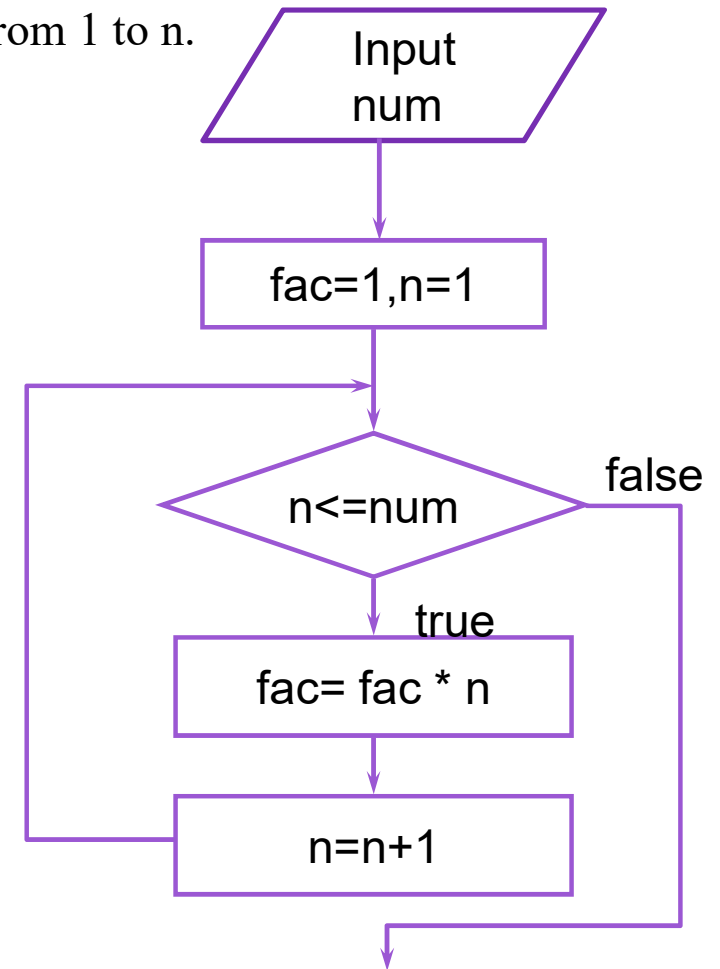
```
do n=1,no,1
```

```
    f=f*n
```

```
end do
```

```
write(*,*) "The factorial of ", no, " is ", f
```

```
end
```



Non-deterministic do loops

- repeat until some criterion is met
- There are two constructs
 - Using **if (....) exit**
 - Using **do while (...)**

- **Syntax**

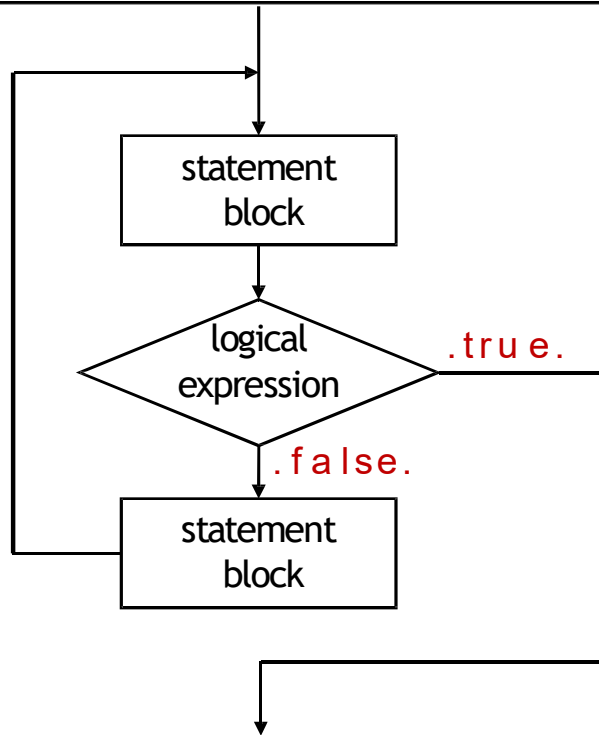
if (....) exit	do while (...)
do if (logical_expression) exit end do	do while (logical_expression) end do

- **semantics**

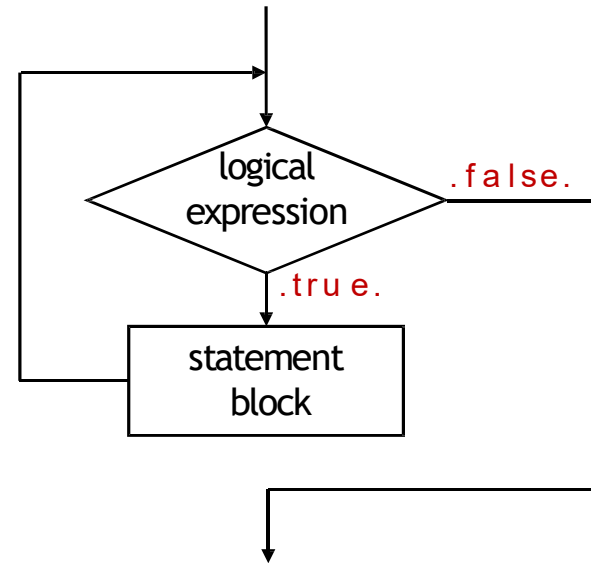
- if **expression** is true then execute **body**
 - **body** is either a single statement or a block
 - *iteration*: single execution of body
- iterate until **expression** evaluates to false
- Avoid Infinite loop by including statements in loop body that assure exit condition is eventually **false**

Non-deterministic do loops

```
do  
  .....  
  if (logical_expression) exit  
  .....  
end do
```



```
do while (logical_expression)  
  .....  
end do
```



WHILE LOOP could result in infinite looping.

exit and cycle Statements

- The exit and cycle statements alter the flow of control
- **exit** statement
 - Causes immediate exit from **loop**
 - Program execution continues with first statement after structure
- **cycle** statement
 - Skips remainder of loop body
 - Proceeds with next iteration of loop

Cycle vs. Exit

- **CYCLE** statement

```
do i = 1, 5  
  if(i == 3) cycle  
  write(*,*) i  
end do
```

the output will be 1, 2, 4, 5

- **EXIT** statement

```
do i = 1, 5  
  if(i == 3) exit  
  write(*,*) i  
end do
```

the output will be 1, 2

Example 3

! print 1 to 100 using a do loop using if

Integer :: c=1 ! Declaration and
!initialization of control variable c

do

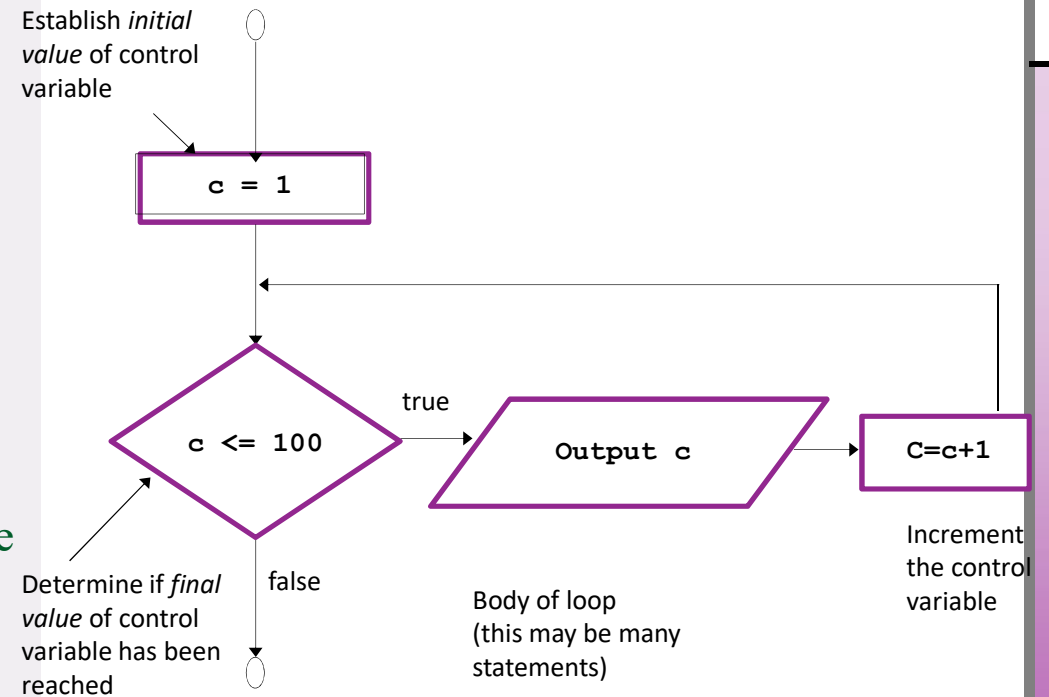
if (c > 100) exit ! while c is <=100

write (*,*) c

c=c+1 !add one to the c every
!time the loop repeats.

End do ! when c is >100 then the loop will terminate

End



Example 3

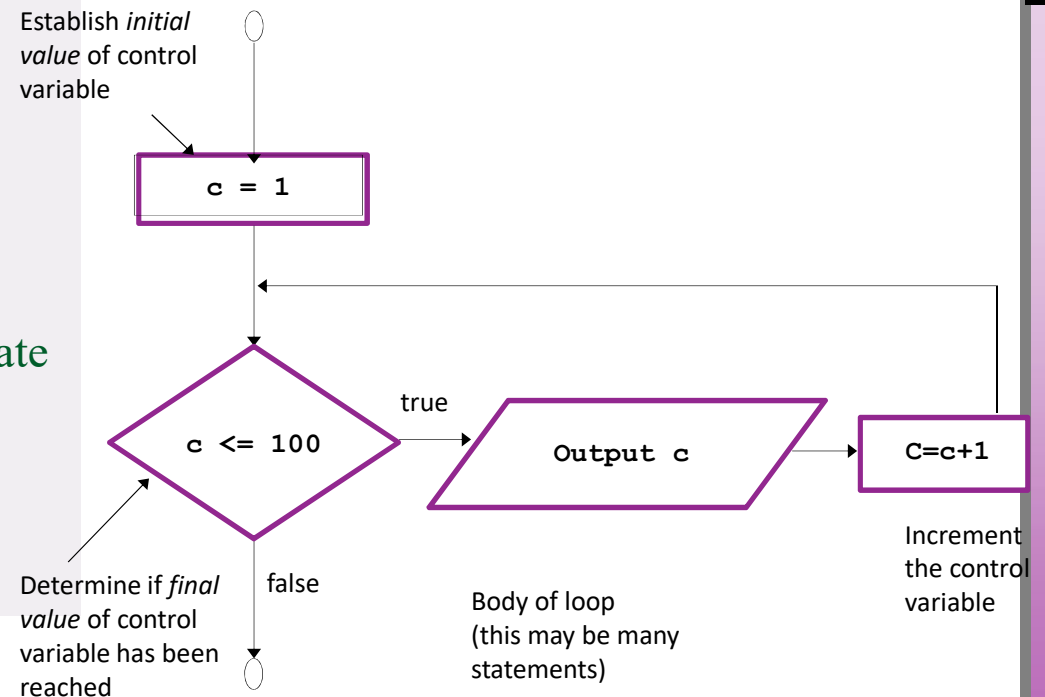
! print 1 to 100 using a while loop

Integer :: c=1 ! Declaration and
!initialization of control variable c

do while (c<=100) ! while c <=100
 write (*,*) c
 c=c+1 !add one to the c every
 !time the loop repeats.

End do ! when c is >100 then the loop will terminate

End



Example 4

- Find the maximum positive number

Integer :: v =0, m =0

do while(v /= -1)

Write (*,*) "Enter a positive integer", "(-1 to stop):"

read (*,*) v

if(v > m) **then**

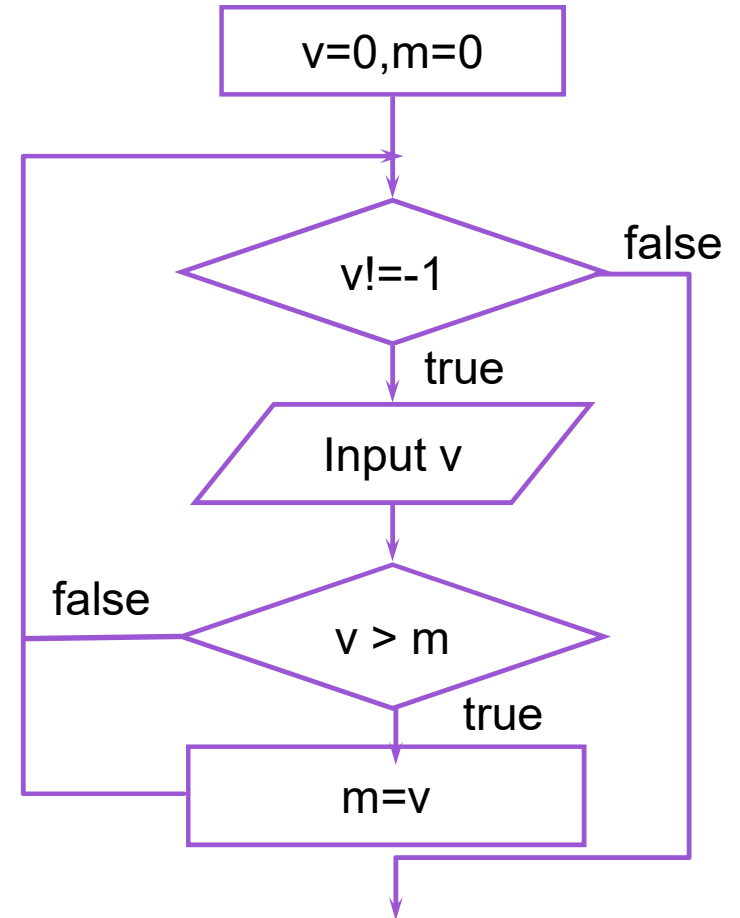
m= v;

end if

End do

Write (*,*) "The maximum value found is " , m

end



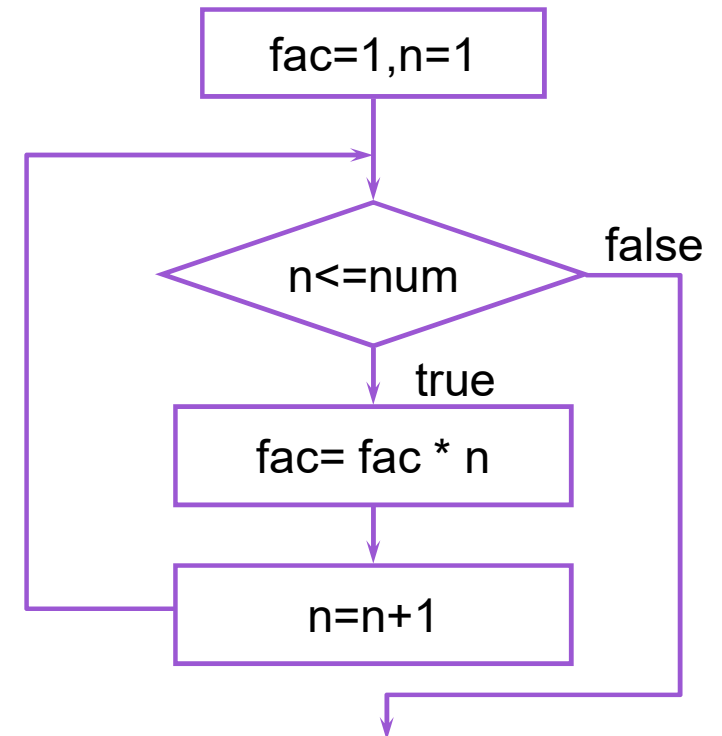
Example 6

$n!$ (n factorial) is defined as the product of all the integers from 1 to n .

$$n! = 1 * 2 * 3 * \dots * n$$

Example: $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$

```
Integer :: num, fac=1, n=1
Write (*,*) "Enter a positive integer:"
read (*,*) num
Do
  if (n > num) exit
      fac = fac * n
      n = n+1
End do
Write (*,*) "The factorial of " , num , " is ", fac
end
```



Choosing the Right Looping Structure

- All loops have their place in FORTRAN
 - If you know or can determine in advance the number of repetitions needed, the **deterministic** loop is the correct choice
 - If you do not know and cannot determine in advance the number of repetitions needed, and it could be zero, use a **non-deterministic** loop

Nesting of Iterative Constructs

- iterative constructs can be *nested*: one iterative construct may be inside the body of another
- When working with nested loops, the outer loop changes only after the inner loop is completely finished
- nesting may be more than two loops deep

- example: • **Nesting loops (loopwithin loop)**

```
do i = 1, 5
  do j = 1, 4
    write(*,*) i, j
  end do
end do
```

- **Named loops**

```
outer: do i = 1, 5
  inner: do j = 1, 4
    write(*,*) i, j
  end do inner
end do outer
```

Output

```
1 1
1 2
1 3
1 4
2 1
2 2
```

loops result in compilation errors

```
do i = 1, 3  
  do j = 1, 4  
    write(*,*) i, j  
  end do
```

Missing end do

```
outer: do i = 1, 3  
  inner: do j = 1, 4  
    write(*,*) i, j  
  end do outer
```

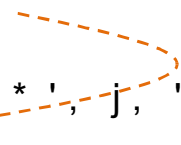
Missing end do inner

```
outer: do i = 1, 3  
  inner: do j = 1, 4  
    write(*,*) i, j  
  end do outer  
end do inner
```

end do for inner and
outer are interchanged

Using CYCLE and EXIT in nested loops:

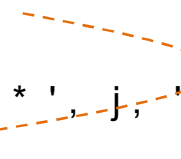
```
program test_cycle_1
integer :: i, j, product
do i = 1, 2
  do j = 1, 3
    if (j==2) cycle
    product = i*j
    write(*,*) i, ' * ', j, ' = ', product
  end do
end do
end program test_cycle_1
```



Output:

```
1 * 1 = 1
1 * 3 = 3
2 * 1 = 2
2 * 3 = 6
```

```
program test_cycle_2
integer :: i, j, product
do i = 1, 2
  do j = 1, 3
    if (j==2) exit
    product = i*j
    write(*,*) i, ' * ', j, ' = ', product
  end do
end do
end program test_cycle_1
```



Output:

```
1 * 1 = 1
2 * 1 = 2
```

Infinite Loops

- Loops that never stop are infinite loops
- The loop body should contain a line that will eventually cause the expression to become false
- Example: Print the odd numbers less than 12

```
integer :: x = 1
do while (x /= 12)

    write (*,*) x
    x = x + 2
end do
```