Università di Pisa - Facoltà di Ingegneria

Notes for the lectures on FORTRAN Programming

a.a. 09/10

Parte I

Ing. Nicola Forgione

Dipartimento di Ingegneria Meccanica, Nucleare e della Produzione

E-mail: nicola.forgione@ing.unipi.it; tel. 0502218057

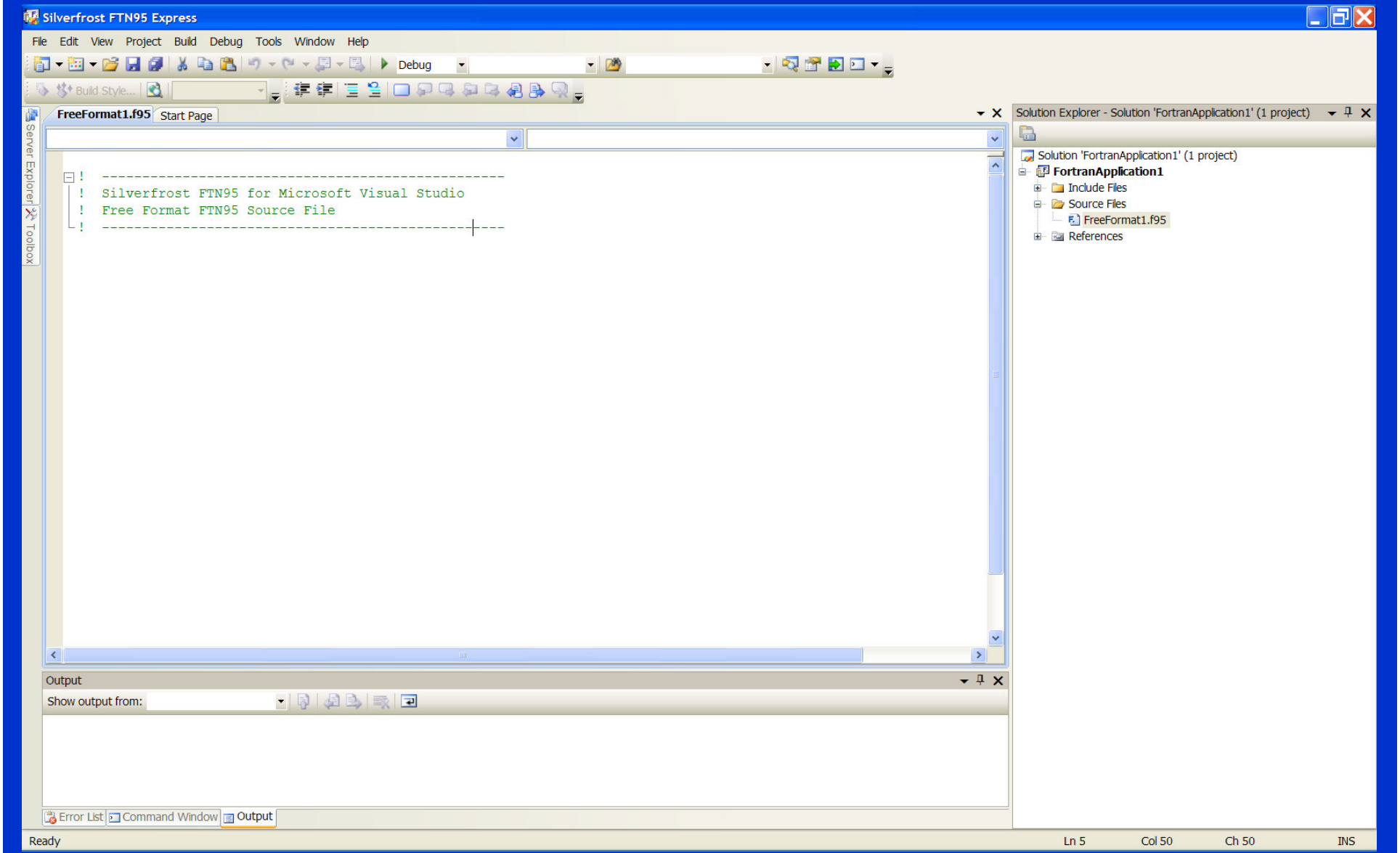
Development of Fortran Standard

- **FORTRAN is the acronym of mathematical FORmula TRANslation System**
- ***Salford compiler is a free FORTRAN 95 compiler for non-commercial use*:**

  http://www.silverfrost.com/52/ftn95/ftn95_express.aspx

Year	*Version*	*Note*
1956	FORTRAN	
1958	FORTRAN II/III	
1962	FORTRAN IV	
1966	FORTRAN 66	Standard ANSI (American Standard Association)
1978	FORTRAN 77	
1992	FORTRAN 90	
1997	FORTRAN 95	
2004	FORTRAN 2003	Standard ISO
2010?	FORTRAN 2008	Standard ISO

FTN95 Compiler



Counting DO Loop: DO - CONTINUE

The DO construct controls the repeated execution of a block of statements

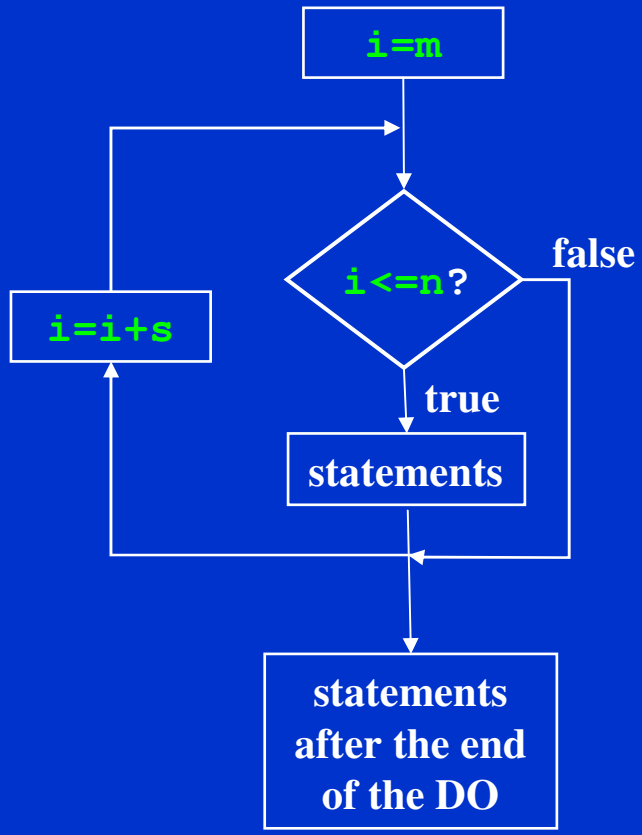
Syntax in FORTRAN 77:

- **Form 1**

```
DO lab var = in-val, fin-val, step
   statements
lab     CONTINUE
```

- **Form 2 (if the step is = to 1)**

```
DO lab var = in-val, fin-val
   statements
lab     CONTINUE
```



where: `var` is a variable of type INTEGER, called

loop variable or loop index;

`in-val` specify the initial value of `var`;

`fin-val` specify the final value of `var`;

`step` specify the step-size value of `var`.

- **The do-loop variable must never be changed by other statements within the loop**

Counting DO Loop: DO - END DO

Syntax in FORTRAN 95

- Form 1

```
DO var = in-val, fin-val, step
   statements
END DO
```

- Form 2 (if the step is = to 1)

```
name: DO var = in-val, fin-val
   statements
END DO name
```

where: `var` is a variable of type INTEGER, called

loop variable or loop index;

`in-val` specify the initial value of `var`;

`fin-val` specify the final value of `var`;

`step` specify the step-size value of `var`.

- The do-loop variable must never be changed by other statements within the loop

```
loopy: DO i = 1, 30, 2
 ... ! i is 1,3,5,7,...,29
 ... ! 15 iterations
END DO loopy
```

```
DO l = 1,30
 ... ! i = 1,2,3,...,30
 ... ! 30 iterations
END DO
```

```
DO j = 30, 1, -2
 ... ! j is 30,28,26,...,2
 ... ! 15 iterations
END DO
```

```
DO k = 30, 1, 2
 ... ! 0 iterations
 ... ! loop skipped
END DO
```

DO – WHILE Statement

- The DO-WHILE statement executes the block of statements while a specified condition remains true

Syntax

```
DO lab WHILE (expr)
  statements
lab     statement (or END DO)
```

```
CHARACTER(1) input
input = ' '
DO WHILE ((input .NE. 'n') .AND. (input .NE. 'y'))
   WRITE (*, '(A)') 'Enter y or n: '
   READ (*, '(A)') input
END DO
```

where: `lab` (optional) is a label specifying an executable statement in the same program unit;

`expr` is a scalar logical expression enclosed in parentheses.

The following examples show required and optional **END DO** statements:

Required

```
DO WHILE (I .GT. J)
  ARRAY(I,J) = 1.0
  I = I - 1
END DO
```

Optional

```
   DO 10 WHILE (I .GT. J)
     ARRAY(I,J) = 1.0
     I = I - 1
10 END DO
```

CYCLE and EXIT Statements

- The CYCLE statement interrupts the current execution cycle of the innermost (or named) DO construct and a new iteration cycle of the DO construct can begins

Syntax

`CYCLE[name]`

```
DO I =1, 10
  A(I) = C + D(I)
  IF (D(I) < 0) CYCLE     ! If true, the next statement is omitted
  A(I) = 0                ! from the loop and the loop is tested again.
END DO
```

where: `name` (optional) is the name of the DO construct

- The EXIT statement terminates execution of the innermost (or named) DO construct

Syntax

`EXIT[name]`

where: `name` (optional) is the name

of the DO construct

```
i = 0
DO
  i = i + 1
  IF (i .GT. 100) EXIT
  PRINT*, "I is", i
END DO
! if i>100 control jumps here
PRINT*, "Loop finished. I now equals", i
```

Named and Nested Loops

- DO loops can have a name (only from FORTRAN 90) and EXIT and/or CYCLE statements can be made to refer to a particular loop through its loop-name

```
LOOP_A : DO I = 1, 15
  N = N + 1
  IF (N > I) EXIT LOOP_A
END DO LOOP_A
```

```
0|      outa: DO
1|       inna: DO
2|        ...
3|        IF (a.GT.b) EXIT outa  ! jump to line 9
4|        IF (a.EQ.b) CYCLE outa ! jump to line 0
5|        IF (c.GT.d) EXIT inna  ! jump to line 8
6|        IF (c.EQ.a) CYCLE      ! jump to line 1
7|       END DO inna
8|      END DO outa
9|      ...
```

IF-Arithmetic statement

- Conditionally transfers control to one of three statements through their corresponding labels, based on the value of an arithmetic expression (it is an obsolescent feature in Fortran 90).

  Syntax

  ```
  IF (expr) lab1, lab2, lab3
  ```

where: `expr` is a scalar numeric expression of type integer or real (enclosed in parentheses);

`lab1`, `lab2`, `lab3` are the labels of valid branch target statements; all the three labels are required, but they do not need to refer to three different statements; the same label can appear more than once in the same arithmetic IF statement.

If the Value of *expr* is:	Control Transfers To:
Less than 0	Statement *label1*
Equal to 0	Statement *label2*
Greater than 0	Statement *label3*

IF-Arithmetic statement

- The following example transfers control to statement 50 if the real variable THETA is less than or equal to the real variable MU. Control passes to statement 100 only if THETA is greater than MU.

```
IF (THETA-MU) 50,50,100
```

- The following example transfers control to statement 400 if the value of the integer variable N is even. It transfers control to statement 200 if the value is odd.

```
IF (N/2*2-N) 200,400,200
```

- The following statement transfers control to statement 100 for N<5, to statement 200 for N=5, and to statement 300 for N>5:

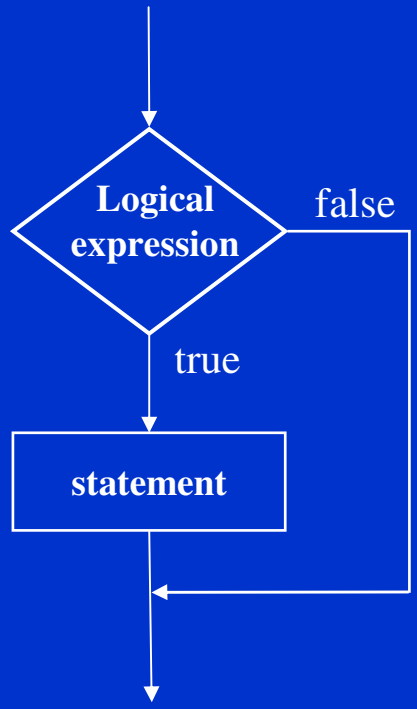
```
IF (N-5) 100, 200, 300
```

IF-Logical statement

- **Executes one statement based on the value of a logical expression. (This statement was called a logical IF statement in FORTRAN 77)**

  Syntax

  `IF (expr) stmt`



where: `expr` is a scalar logical expression

(enclosed in parentheses);

`stmt` is an executable Fortran statement.

For example,

```
IF (x .GT. y) Maxi = x
```

means 'if `x` is greater than `y` then set `Maxi` to be equal to the value of `x`'.

More examples,

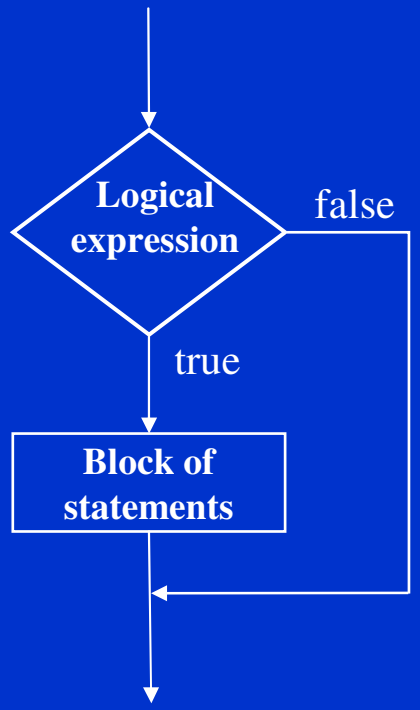
```
IF (a*b+c <= 47) Boolie = .TRUE.
IF (i .NE. 0 .AND. j .NE. 0) k = 1/(i*j)
IF (i /= 0 .AND. j /= 0) k = 1/(i*j) ! same
```

IF-Construct

- **IF-THEN-ENDIF**: executes one block of statements depending on the value of a logical expression.

Syntax

```
IF (expr) THEN
   block
ENDIF
```



where: `expr` is a scalar logical expression (enclosed in parentheses);

`block` is a sequence of more statements.

Examples:

```
IF (x.GT.0) THEN
   y=sqrt(x)
ENDIF
```

```
IF((x>=0).AND.(y>=0)) THEN
   z=sqrt(x)+sqrt(y)
   w=z+5
ENDIF
```

IF-Construct

- **IF-THEN-ELSE-ENDIF: executes one block of statements if the logical expression is true otherwise executes another block of statements .**

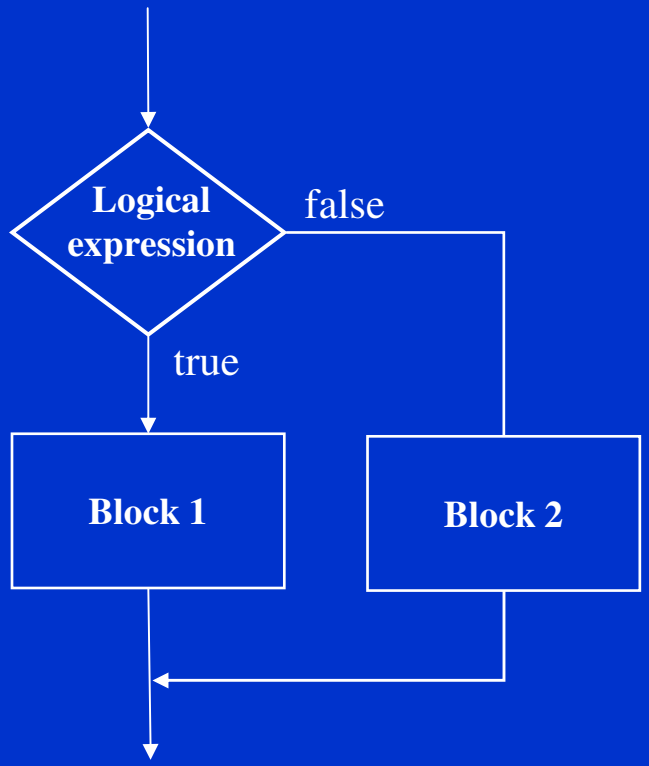
Syntax

```
IF (expr) THEN
   block1
ELSE
   block2
ENDIF
```

where: `expr` is a scalar logical expression (enclosed in parentheses);

`block1` is the first sequence of statements.

`block2` is the second sequence of statements.




```
IF (x>=0) THEN
   y=sqrt(x)
ELSE
   y=exp(x)-1
ENDIF
```

IF-Construct

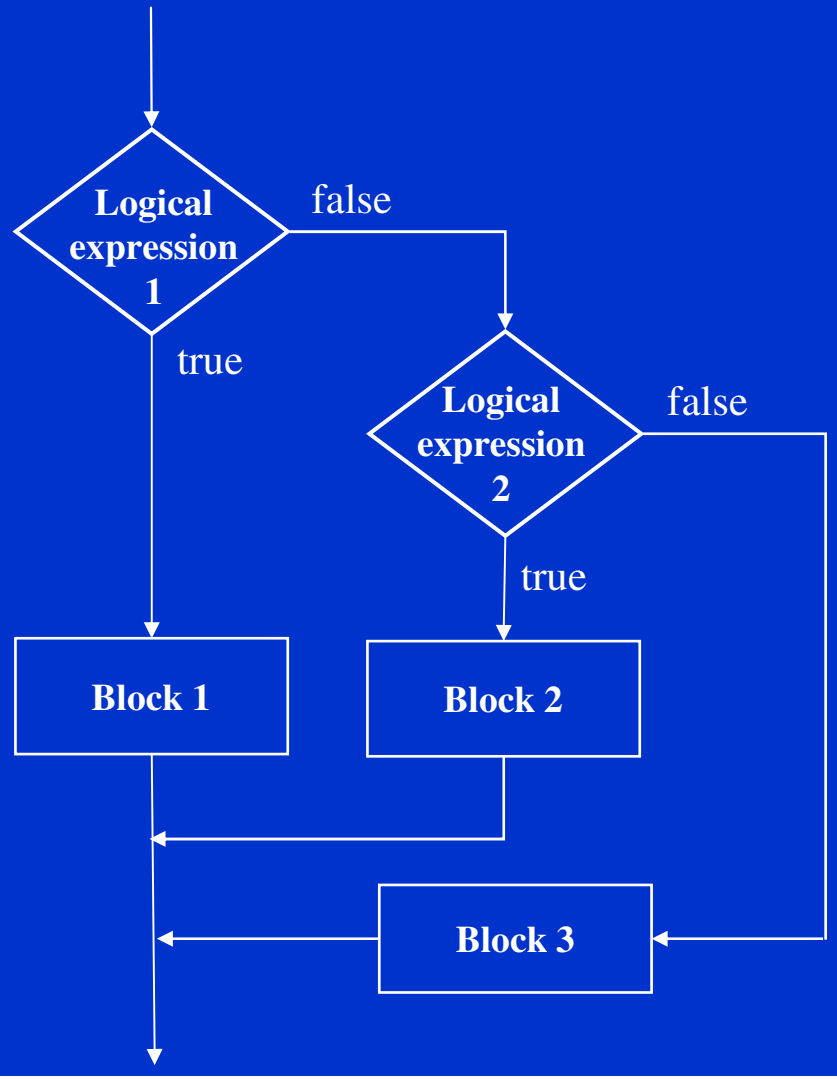
- **IF-THEN-ELSEIF-ELSE-ENDIF: executes one block of statements (block1) if the logical expression (expr1) is true otherwise executes another block of statements (block 2) if the corresponding logical expression (expr2) is true otherwise executes another block of statements (block3).**

Syntax

```
IF (expr1) THEN
   block1
ELSE IF (expr2) THEN
   block2
ELSE
   block3
ENDIF
```

Both ELSE and ELSEIF are optional

```
IF (x>=0) THEN
   y=sqrt(x)
ELSE IF (x>-10)
   y=exp(x)-1
ELSE
   y=-1
ENDIF
```



IF-Construct

- **IF-THEN-ELSEIF-ELSE-ENDIF: examples**

```
IF (i .EQ. 0) THEN
 PRINT*, "I is Zero"
ELSE IF (i .GT. 0) THEN
 PRINT*, "I is greater than Zero"
ELSE
 PRINT*, "I must be less than Zero"
ENDIF
```

- **We can also have more ELSEIF branches inside an IF-construct**

```
IF (x .GT. 3) THEN
 CALL SUB1
ELSEIF (x .EQ. 3) THEN
 A = B*C-D
ELSEIF (x .EQ. 2) THEN
 A = B*B
ELSE
 IF (y .NE. 0) A=B
ENDIF
```

Named and Nested IF-Construct

- **In FORTRAN 90/95 all the IF-constructs can be named and nested.** The names may be used once per program unit and are intended to make much more clear the program.

```
outa: IF (a .NE. 0) THEN
        PRINT*, "a /= 0"
        IF (c .NE. 0) THEN
          PRINT*, "a /= 0 AND c /= 0"
        ELSE
          PRINT*, "a /= 0 BUT c == 0"
        ENDIF
      ELSEIF (a .GT. 0) THEN outa
        PRINT*, "a > 0"
      ELSE outa
        PRINT*, "a must be < 0"
      ENDIF outa
```

```
outa: DO i = 1,n
 inna: DO j = 1,m
         ...
       IF (X == 0) EXIT
         ...
       IF (X < 0) EXIT outa
         ...
       IF (X > 10) CYCLE inna
         ...
       IF (X > 100) CYCLE outa
         ...
     END DO inna
    END DO outa
```

SELECT-CASE-Construct

- **SELECT CASE-CASE-END SELECT: transfers program control to a selected block of statements according to the value of a controlling expression.**

Syntax

```
SELECT CASE (expr)
CASE (value1)
   block1
CASE (value2)
   block2
...
CASE DEFAULT
   block3
END SELECT
```

```
SELECT CASE (I)
 CASE(1);   Print*, "I==1"
 CASE(2:9); Print*, "I>=2 and I<=9"
 CASE(10);  Print*, "I>=10"
 CASE DEFAULT; Print*, "I<=0"
END SELECT
```

where: `expr` is a scalar expression of type integer, logical or character (enclosed in parentheses); evaluation of this expression results in a value called the *case index*;

`value` is one or more scalar integer, logical, or character initialization expressions (enclosed in parentheses). Each case-value must be of the same type and kind parameter as expr.

SELECT-CASE-Construct

```
GET_ANSWER: SELECT CASE (cmdchar)
CASE ('0')
   WRITE (*, *) "Must retrieve one to nine files"
CASE ('1':'9')
   CALL RetrieveNumFiles (cmdchar)
CASE ('A', 'a')
   CALL AddEntry
CASE ('D', 'd')
   CALL DeleteEntry
CASE ('H', 'h')
   CALL Help
CASE DEFAULT
   WRITE (*, *) "Command not recognized; please use H for help"
END SELECT GET_ANSWER
```

Example 1

The roots of a quadratic equation $ax^2 + bx + c = 0$ can be expressed as follows:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

In order to use the square root, $b^2 - 4ac$ must be positive.