Fortran Programming

Dr. Ugur GUVEN

What is FORTRAN?

- Fortran stands for FORMULA Translation as it is a programming language that is used for scientific and engineering calculations
- Although original FORTRAN was very weak in its visual aspects, the new FORTRAN 2008 is completely a visual FORTRAN.
- Your programming codes are translated and compiled into machine language. Machine language is made up of 0 and 1 and that is the only terminology that a computer understands.

Binary and HexaDecimal Codes

Numbers in a computer's memory must therefore be represented in *binary code*, where each bit in a sequence stands for a successively higher power of 2. The decimal numbers 0 to 15, for example, are coded in binary as follows:

Decimal	Binary	Hexadecimal	Decimal	Binary	Hexadecimal
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

A *byte* is the amount of computer memory required for one character, and is eight bits long. Since each bit in a byte can be in two possible states, this gives 2^8, i.e. 256, different combinations.

Hexadecimal code (see table) is often used because it is more economical than binary. Each hexadecimal digit stands for a power of 16. E.g.

Sample Numerical Fortran Program

```
! Vertical motion under gravity
IMPLICIT NONE
REAL, PARAMETER :: G = 9.8 ! acceleration due to
gravity
REAL S ! displacement (metres)
REAL T ! time
REAL U ! initial speed (metres/sec)
PRINT*, ' Time Displacement'
PRINT*
U = 60
T = 6
S = U * T - G / 2 * T ** 2
PRINT*, T, S
END PROGRAM Vertical
```

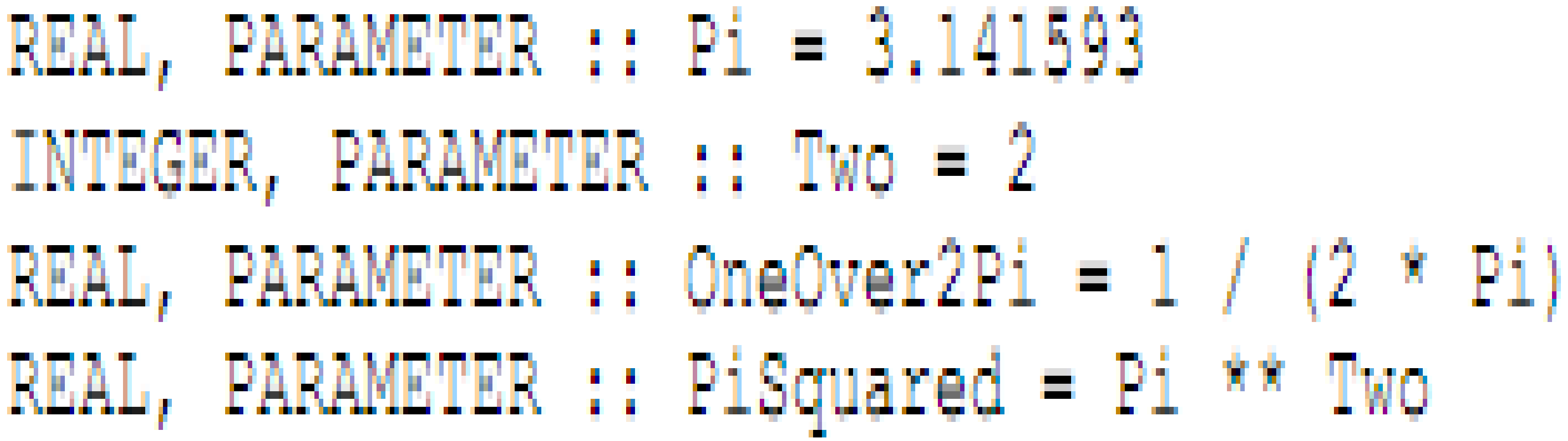
Definition of Variables in Fortran

- It is important for you to define the constants and variables in the program.
- If the value must remain the same throughout the program, that data should be defined as constant.
- Variables can be defined as Integer, Real or Character.
- Due to Implicit rule, variables I to N are reserved for integers, so you should use IMPLICIT NONE statement if you don't want to use these variables as Integers.

```
INTEGER X
REAL INTEREST, A, B
CHARACTER LETTER
REAL :: A = 1
```

Declared Constants

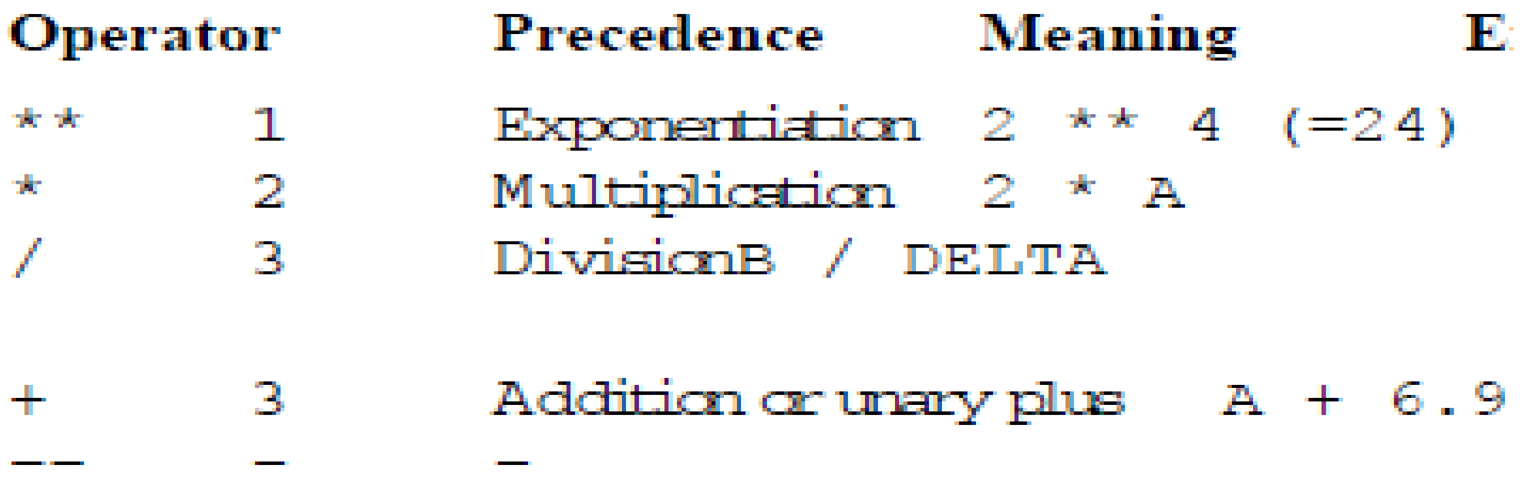
- In FORTRAN, you can declare a constant that can stay the same throughout the program. Any attempt to change it would generate an error message



```
REAL, PARAMETER :: Pi = 3.141593
INTEGER, PARAMETER :: Two = 2
REAL, PARAMETER :: OneOver2Pi = 1 / (2 * Pi)
REAL, PARAMETER :: PiSquared = Pi ** Two
```

Mathematical Operations in FORTRAN

- Mathematical Operations are carried out according to their algebraic priorities. This means that first inside the parentheses are evaluated and then the multiplication and divisions are carried out. Last the additions and subtractions are carried out.



Operator	Precedence	Meaning	E
**	1	Exponentiation	2 ** 4 (=24)
*	2	Multiplication	2 * A
/	3	DivisionB / DELTA	
+	3	Addition or unary plus	A + 6.9
--	-	-	

Order of Operations in FORTRAN

- Hence, the following order is followed:

1) Parentheses

2) Exponential Calculations

2) Multiplication & Division (Carried out from left to right)

3) Normal Addition and Subtraction

Sample Operations in FORTRAN

- Hence, the following samples hold true:

a) (2+3)*4 = 20

b) 2*5*6/20 = 3

c) 4*5**2-10 = 90

d) 2+5*6/10 =5

e) (5*2)**2/4 = 25

Integer Division in FORTRAN

- Make sure that in divisions, you don’t mix Integers and Real numbers as Integer division will yield different results as compared to Real Division.

```
10 / 3          evaluates to 3
  19 / 4        evaluates to 4
  4 / 5          evaluates to 0
by zero)
  - 8 / 3       evaluates to -2
3 * 10 / 3      evaluates to 10
10 / 3 * 3      evaluates to 9
```

Input in FORTRAN

- You can input data by directly defining them in FORTRAN or you can use the READ statement so that the user can input his or her data.

```
BALANCE = 1000
RATE = 0.09
```

```
READ*, BALANCE, RATE
```

The general form of the READ* statement is

```
READ*, list
```

where *list* is a list of variables separated by commas.

Read Statement in FORTRAN

The statements

```
READ*,  A
READ*,  B,  C
READ*,  D
```

with the input records

```
1  2  3
4
7  8
9  10
```

have the same effect as the assignments

```
A  =  1
B  =  4
C  =  7
D  =  9
```

Read Statement in FORTRAN

- So, as you can see, Read statement reads the input line by line for each Read statement.
- If there are two variables in a Read statement and 3 values on a line, the read statement will only read those first two values.

Reading from a File

- Suppose you have more then one value that you need to input. Inputting 10 or more data every time that you run the program can be very annoying.
- One way to overcome this is by creating an ASCII Text file and then inputting these values from the data file. Thus, every time that you run the program the data from the file will be inputted. (Such as the vertices of an airfoil or temperature gradients in a rocket nozzle)

Reading from a File

- Lets say that you have created a text file called Data.txt with the following values
- 3 4 5 (The values have no comma to separate them)
- You have to use the Open statement to input these values to the program

```
OPEN( 1, FILE = 'DATA' )
READ(1, *) A, B, C
```

- So, as you can see the first number 1 denotes the number of the file and the ‘DATA’ in the first line denotes the name of the file.

Output in FORTRAN

- The best way to output in FORTRAN is by using the PRINT statement. It allows you to output to the screen your private statements, the solution of a numeric expression or any variable.

The general form is

PRINT*, *list*

Output in FORTRAN

- Thus any statement that you want to be printed exactly must be in quotes or apostrophes
- Any variable for which you want the value can be written directly.
- You can write a constant directly

```
PRINT*, "The square root of", 2, 'is', SQRT( 2.0 )
```

```
The square root of  2 is 1.41
```

Output in FORTRAN

- A = 2, B= 3, D= 4, Name = Richard
- PRINT*, A, B, C

  2 3 4
- PRINT*, “My name is ”, Name

  My name is Richard
- PRINT*, 2

  2
- PRINT*, A, “is less than “, 5

  2 is less than 5

Outputting to a Printer

- If you want to output the result to a printer then you use the WRITE statement instead of the PRINT statement.
- WRITE*, “This statement is written on a printer”

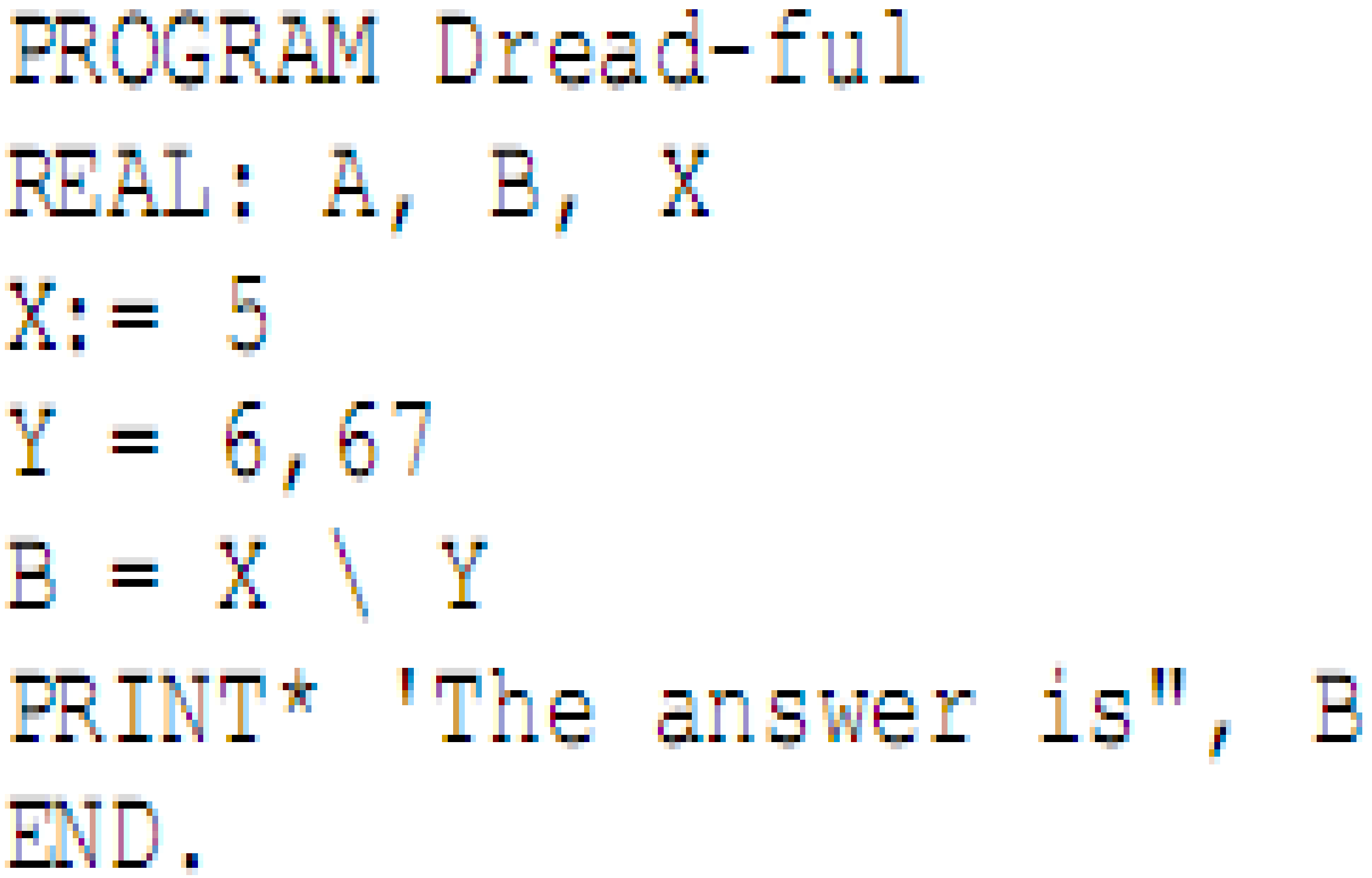
Putting Comments in FORTRAN

- It is always best for you to write some comments on a program line, so that you will be able to remember important points when you have to go back to your program
- The comments are put after an exclamation mark (!) and the program doesn’t read the line behind the !
- **! This is a remark line and it is not read by the compiler**

Sample Program

```
PROGRAM Vertical
! Vertical motion under gravity

IMPLICIT NONE

REAL, PARAMETER :: G = 9.8 ! acceleration due to gravity
REAL S           ! displacement (metres)
REAL T           ! time
REAL U           ! initial speed (metres/sec)

PRINT*, ' Time    Displacement'
PRINT*
U = 60
T = 6
S = U * T - G / 2 * T ** 2
PRINT*, T, S

END PROGRAM Vertical
```

Find the Errors in the Syntax



```
PROGRAM Dread-ful
REAL: A, B, X
X:= 5
Y = 6,67
B = X \ Y
PRINT* 'The answer is", B
END.
```

Summary

- Fort ran statements may be up to 132 characters long and may s tar t anywhere on the line .
- All statements , except assignments , s tar t with a keyword.
- A Fort ran token is a sequence of characters forming a label , keyword, name, constant , operator
- or separator.
- Blanks should be used to improve readability, except inside keywords and names.
- Comments may be type after the exclamation! They should be used liberally to descr ibe variables and to explain how a program works .
- A statement with & as i t s last non-blank character will be continued onto the next line .
- There are five intrinsic data types: integer, real , complex, logical and character .

Summary

- Numeric express ions may be formed from constants and variables with the five numeric
- Intrinsic opera t ors , which operate according to strict rules of precedence .
- Decimal parts are truncated when integers are divided, or when integers are assigned to reals .
- Numeric assignment computes the value of a numeric express ion and assigns it to a real or
- integer variable .
- Groups of variables may be given initial values in a DATA statement .
- PRINT* is used to pr in t (display) output .
- READ* is used to input data from the keyboard while a program is running.
- Data may also be read from an external f i le (e .g . a disk f i le) .

Exercises

3.9 What are the values of X and A (both real) after the following program section has been executed?

```
A = 0
I = 1
X = 0
A = A + I
X = X + I / A
A = A + I
X = X + I / A
A = A + I
X = X + I / A
A = A + I
X = X + I / A
```

Exercises

- State , giving reasons , which of the following are not For t ran variable names:
-
- (a) A2 (b) A.2 (c) 2A (d) 'A'ONE
-
- (e) AONE (f) X_1
- (g) MiXedUp (h) Pay Day
- (i) U.S.S.R. (j) Pay_Day
- (k) min*2 (l) PRINT

Exercises

- Translate the following expressions into FORTRAN

$$ax^2 + bx + c = 0$$

$$\Phi(x) = 0.5 - r\left(at + bt^2 + ct^3\right)$$

$$\left(-0.5x^2\right)/\sqrt{2\pi}$$

$$t = 1/\left(1 + 03326x\right)$$

Exercises

2.8 There are eight pints in a gallon, and 1.76 pints in a litre. The volume of a tank is given as 2 gallons and 4 pints. Write a program which reads this volume in gallons and pints and converts it to litres. (Answer: 11.36 litres)

2.9 Write a program to calculate petrol (gas) consumption. It should assign the distance travelled (in kilometres) and the amount of petrol used (in litres) and compute the consumption in km/litre as well as in the more usual form of litres per 100 km. Write some helpful headings, so that your output looks something (?) like this:

```
Distance          Litres used        Km/L    L/100Km
```

Exercises

- Please write a program to calculate the following statement. Ask the user for inputting X and the number of terms that there should be in your function.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Solution Method for Exercises

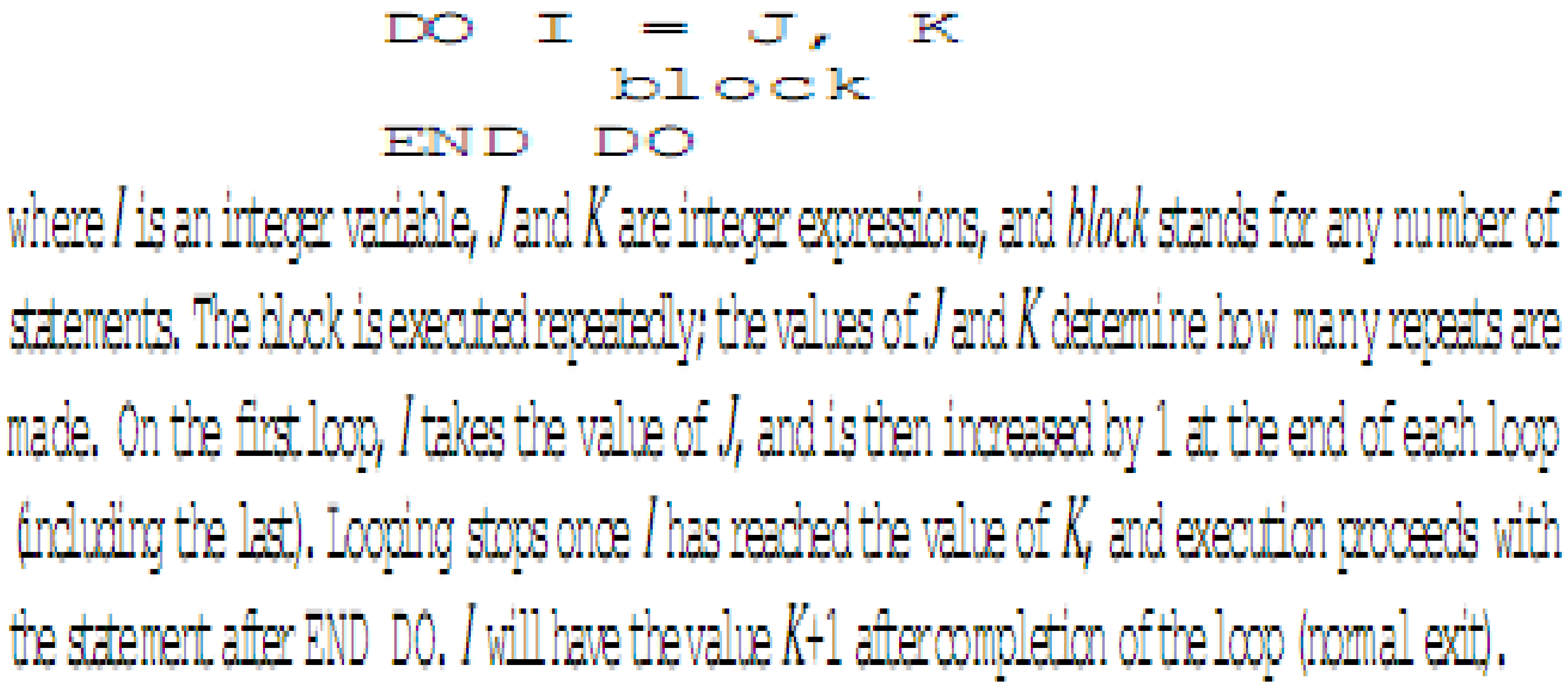
- **Write Source Code in your Fortran Environment**
  - Define variables and constants
  - Input the necessary info from users with Read statement
  - Process the data using appropriate formulas
  - Output the Results
  - Use Remarks after Exclamation Point
- **Use Compiler to Test for Syntax Errors**
- **Use Builder to Test for Logic Errors**
- **Execute the Program**

Loops in Programming

- When you need something repeated in a program, then you will need to create loops to repeat certain lines of codes.
- These loops can be conditional loops or they can be unconditional loops.
- In a conditional loop, a repetition will take place only if a condition is satisfied.
- In an unconditional loop, the repetition will take place regardless of conditions

Do Loop

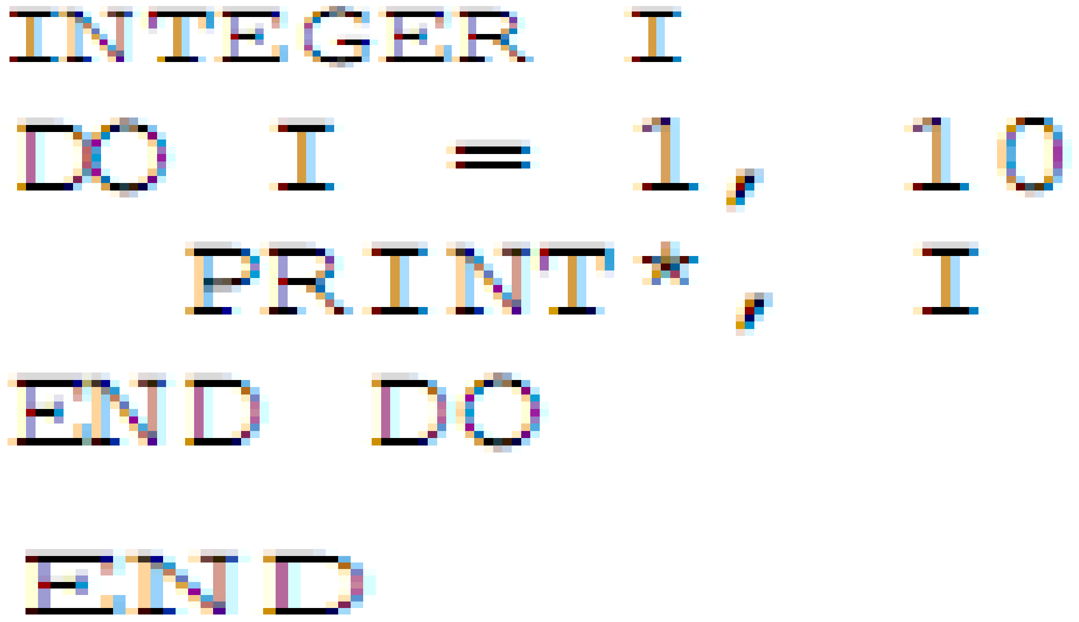
- Do Loop is one of the most powerful loops in FORTRAN or in any other programming language.
- The typical Do statement looks something like this:



```
DO I = J, K
     block
END DO
```

where *I* is an integer variable, *J* and *K* are integer expressions, and *block* stands for any number of statements. The block is executed repeatedly; the values of *J* and *K* determine how many repeats are made. On the first loop, *I* takes the value of *J*, and is then increased by 1 at the end of each loop (including the last). Looping stops once *I* has reached the value of *K*, and execution proceeds with the statement after END DO. *I* will have the value *K*+1 after completion of the loop (normal exit).

Do Loop Example



```
INTEGER I
DO I = 1, 10
   PRINT*, I
END DO
END
```

Do Loop Example

- With the DO Loop, You can Go Forward or Backwards. The statements will be executed until the expressions in the Do Loop is satisfied and exhausted.
- DO I = 10 , 7, -1

  PRINT*, I + 1

  END DO
- The above code segment will output:

  11

  10

  9

  8

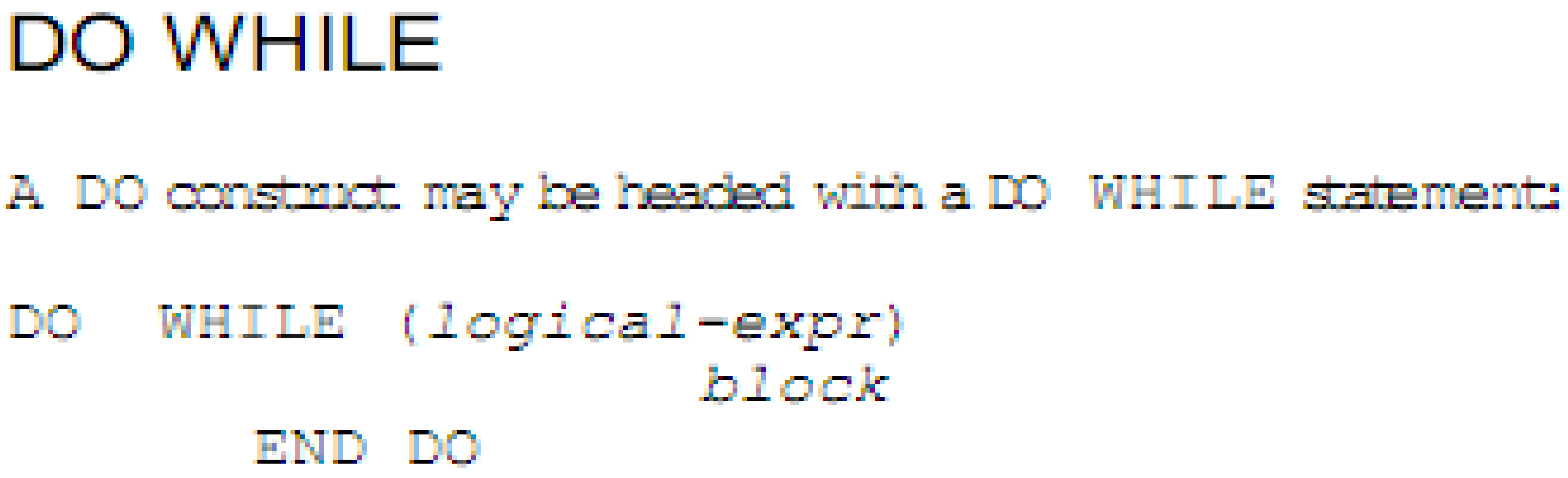
Do Loop Example

- The last number in the DO statement denotes how many steps are incremented. For example:

-
  ```
  Do M = 4, 12, 2
  print*, M
  end do
  ```
- The output will be:
  ```
  4
  6
  8
  10
  12
  ```

Conditional DO Loop (DO WHILE)

- If you only want something executed repeatedly as long as a condition holds true, then you must use the Do While Construct



DO WHILE

A DO construct may be headed with a DO WHILE statement:

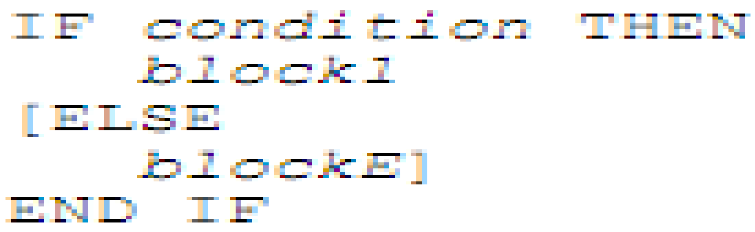
```
DO  WHILE (logical-expr)
                    block
      END DO
```

Program Example: Square Rooting with Newton

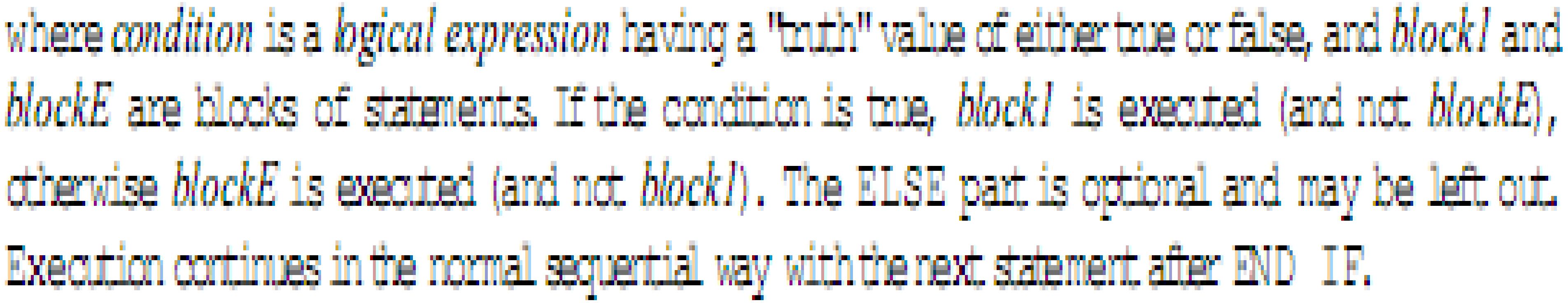
```fortran
PROGRAM Newton
! Square rooting with Newton

IMPLICIT NONE
REAL    A                ! number to be square rooted
INTEGER I                ! iteration counter
REAL    X                ! approximate square root of A

WRITE( *, 10, ADVANCE = 'NO' ) 'Enter number to be square rooted: '
   10  FORMAT( A )
READ*, A
PRINT*
X = 1                    ! initial guess (why not?)

DO I = 1, 6
  X = (X + A / X) / 2
  PRINT*, X
ENDDO

PRINT*
PRINT*, 'Fortran 90''s value:', SQRT( A )

END
```

Conditional Statements: IF-THEN-ELSE

- Conditional statements allow for you to branch the programming depending upon the condition set forth in the conditional statement
- **IF** (Condition Satisfied) **THEN** (Do This) **ELSE** (Do This)



```
IF condition THEN
    block1
[ELSE
    blockE]
END IF
```



where *condition* is a *logical expression* having a "truth" value of either true or false, and *block1* and *blockE* are blocks of statements. If the condition is true, *block1* is executed (and not *blockE*), otherwise *blockE* is executed (and not *block1*). The ELSE part is optional and may be left out. Execution continues in the normal sequential way with the next statement after END IF.

IF - THEN - ELSE

The condition may be formed from numeric expressions with the *relational operators*, such as <, <=, == (equals) and /= (not equals), and from other logical expressions with the *logical operators*, such as .NOT., .AND. and .OR.. These are all discussed fully with the most general form of IF

```
IF (Final >= 50) THEN
   PRINT*, CRM, ExmAvg, Final, 'PASS'
ELSE
   PRINT*, CRM, ExmAvg, Final, 'FAIL'
END IF
```

IF THEN Statement Examples

```
• IF (X < A) Then
        A=A+1
  Else A=A-1
• IF (X <= A) THEN
        A=A*B
  Else A=A/B
• IF (X >= A) THEN
        X=4
  Else X=3
```

IF Constructs in General

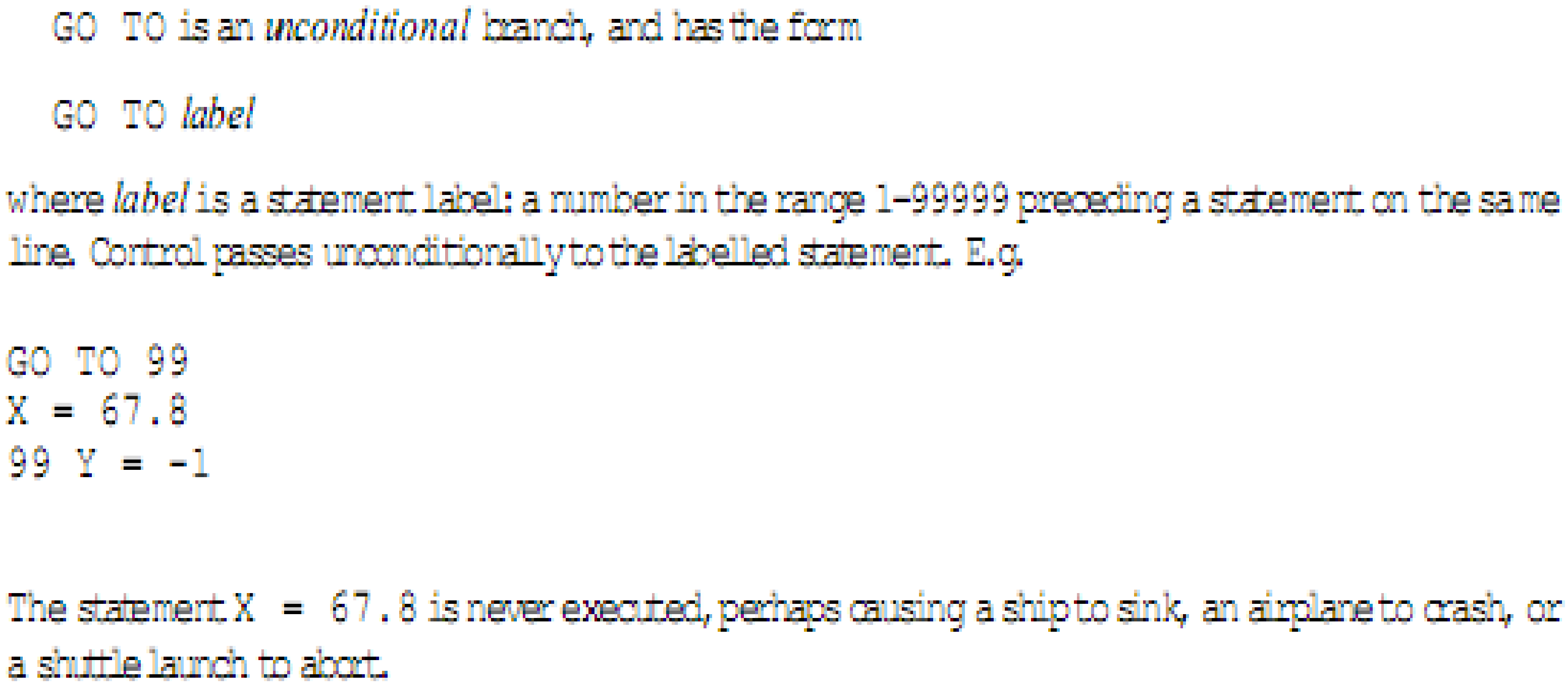
```
IF (logical-expr1) THEN
             block1
     ELSE IF (logical-expr2) THEN
                   block2
     ELSE IF (logical-expr3) THEN
                   block3
     ...
     ELSE
                   blockE
     END IF
```

If *logical-expr1* is true the statements in *block1* are executed, and control passes to the next statement after END IF. If *logical-expr1* is false, *logical-expr2* is evaluated. If it is true the statements in *block2* are executed, followed by the next statement after END IF. If none of the logical expressions is true, the statements in *blockE* are executed. Clearly, the logical expressions should be arranged so that only one of them can be true at a time.

There may be any number of ELSE IFs (or none at all), but there may be no more than one ELSE.

GOTO Statement

- If you want your program to branch off in another direction without executing some of these statements, then the best way to proceed would be to use the GOTO statement



`GO TO` is an *unconditional* branch, and has the form

`GO TO` *label*

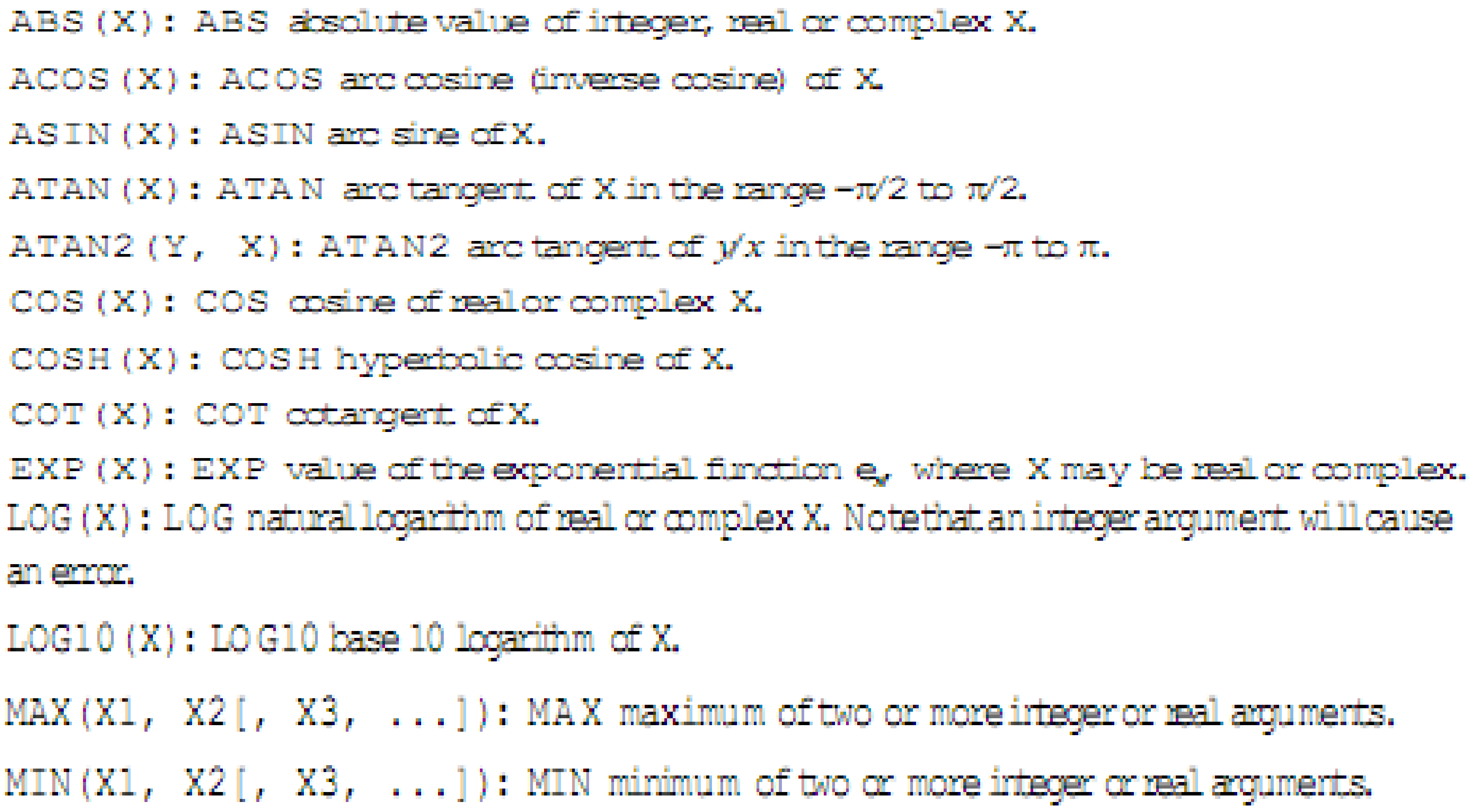
where *label* is a statement label: a number in the range 1–99999 preceding a statement on the same line. Control passes unconditionally to the labelled statement. E.g.

```
GO TO 99
X = 67.8
99 Y = -1
```

The statement `X = 67.8` is never executed, perhaps causing a ship to sink, an airplane to crash, or a shuttle launch to abort.

INTRINSIC FUNCTIONS

- FORTRAN has several functions that you can use for calculating ready made functions.



ABS(X): ABS absolute value of integer, real or complex X.

ACOS(X): ACOS arc cosine (inverse cosine) of X.

ASIN(X): ASIN arc sine of X.

ATAN(X): ATAN arc tangent of X in the range $-\pi/2$ to $\pi/2$.

ATAN2(Y, X): ATAN2 arc tangent of y/x in the range $-\pi$ to π.

COS(X): COS cosine of real or complex X.

COSH(X): COSH hyperbolic cosine of X.

COT(X): COT cotangent of X.

EXP(X): EXP value of the exponential function e^x, where X may be real or complex.

LOG(X): LOG natural logarithm of real or complex X. Note that an integer argument will cause an error.

LOG10(X): LOG10 base 10 logarithm of X.

MAX(X1, X2[, X3, ...]): MAX maximum of two or more integer or real arguments.

MIN(X1, X2[, X3, ...]): MIN minimum of two or more integer or real arguments.

INTRINSIC FUNCTIONS

NINT(X [,KIND]): NINT *nearest* integer to X, e.g. NINT(3.9) returns 4, while NINT(-3.9) returns -4.

REAL(X [,KIND]): REAL function converts integer, real or complex X to real type, e.g. REAL(2)/4 returns 0.5, whereas REAL(2/4) returns 0.0.

SIN(X): SIN sine of real or complex X.

SINH(X): SINH hyperbolic sine of X.

SQRT(X): SQRT square root of real or complex X.

TAN(X): TAN tangent of X.

TANH(X): TANH hyperbolic tangent of X.

Exercises

3.1 Translate the following into Fortran statements:

(a) Add 1 to the value of I and store the result in I.

(b) Cube I, add J to this, and store the result in I.

(c) Set G equal to the larger of the two variables E and F.

(d) If D is greater than zero, set X equal to minus B.

(e) Divide the sum of A and B by the product of C and D, and store the result in X.

Exercises

3.2 If C and F are Celsius and Fahrenheit temperatures respectively, the formula for conversion from Celsius to Fahrenheit is $F = 9C/5 + 32$.

(a) Write a program which will ask you for the Celsius temperature and display the equivalent Fahrenheit one with some sort of comment, e.g.

```
The Fahrenheit temperature is: ...
```

Try it out on the following Celsius temperatures (answers in parentheses): 0 (32), 100 (212), -40 (-40!), 37 (normal human temperature: 98.6).

(b) Change the program to use a DO loop to compute and write the Fahrenheit equivalent of Celsius temperatures ranging from 20° to 30° in steps of 1°.

Exercises

3.3 Write a program that displays a list of integers from 10 to 20 inclusive, each with its square root next to it.

3.4 Write a program to find and display the sum of the successive integers 1, 2, ..., 100. (Answer: 5050)

3.5 Write a program to find and display the sum of the successive *even* integers 2, 4, ..., 200. (Answer: 10100)

3.6 Ten students in a class write a test. The marks are out of 10. All the marks are entered in an external file MARKS. Write a program which will read all ten marks from the file and find and display the average mark. Try it on the following marks (each on a separate line in the file):

```
5   8   0   10   3   8   5   7   9   4   (Answer: 5.9)
```

3.7 The pass mark for the test in the previous problem is 5 out of 10. Change your program so it uses an IF-THEN to find out how many students passed the test.

Exercises

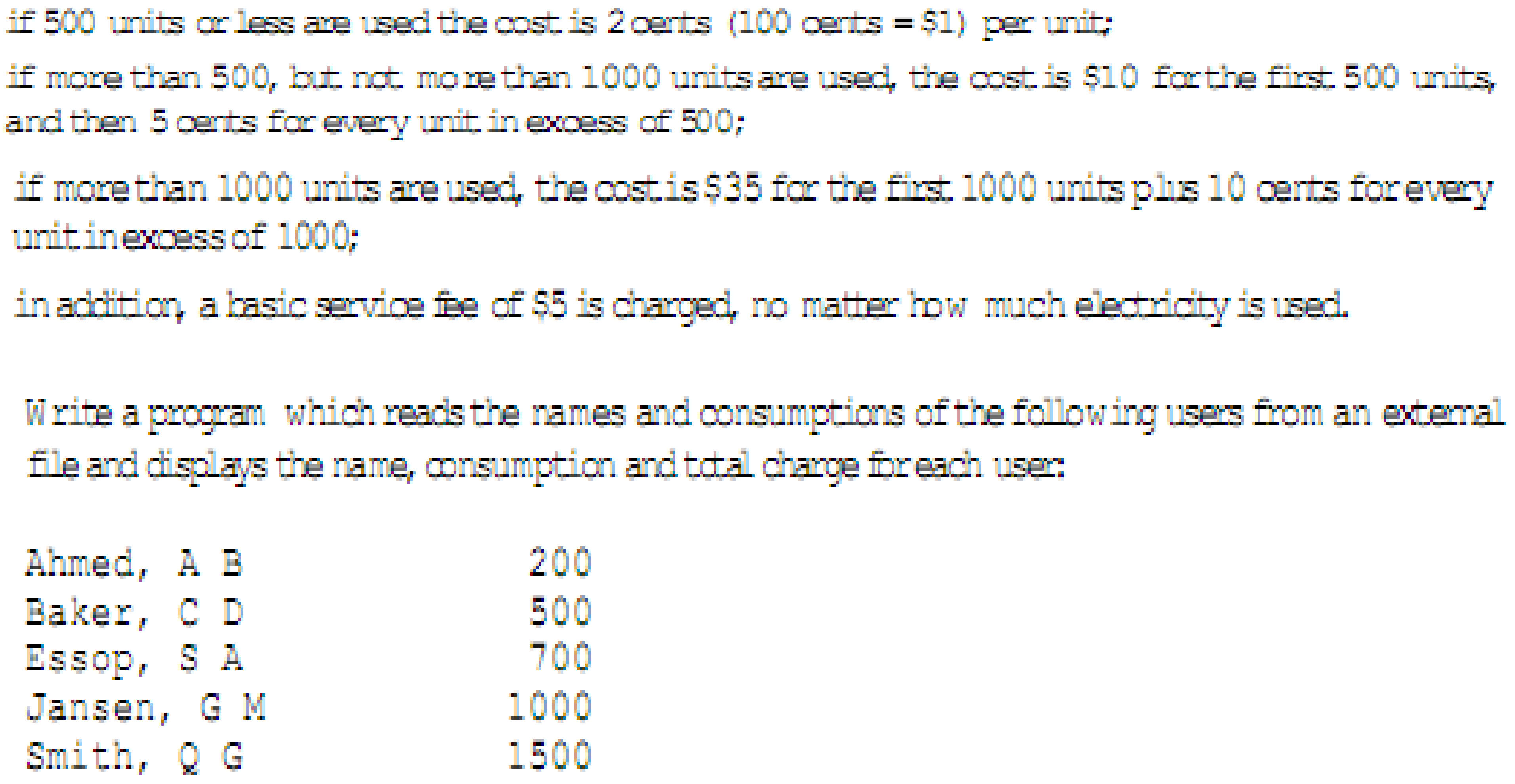
3.16 It has been suggested that the population of the United States may be modelled by the formula

$$P(t) = \frac{197273000}{1 + e^{-0.03134(t - 1913.25)}}$$

where t is the date in years. Write a program to compute and display the population every *ten* years from 1790 to 2000. Use the intrinsic function EXP (X) to compute the exponential e^X.

PROJECT

- Write a program that calculates the electricity bill of a small town residents based on the following conditions



if 500 units or less are used the cost is 2 cents (100 cents = $1) per unit;

if more than 500, but not more than 1000 units are used, the cost is $10 for the first 500 units, and then 5 cents for every unit in excess of 500;

if more than 1000 units are used, the cost is $35 for the first 1000 units plus 10 cents for every unit in excess of 1000;

in addition, a basic service fee of $5 is charged, no matter how much electricity is used.

Write a program which reads the names and consumptions of the following users from an external file and displays the name, consumption and total charge for each user:

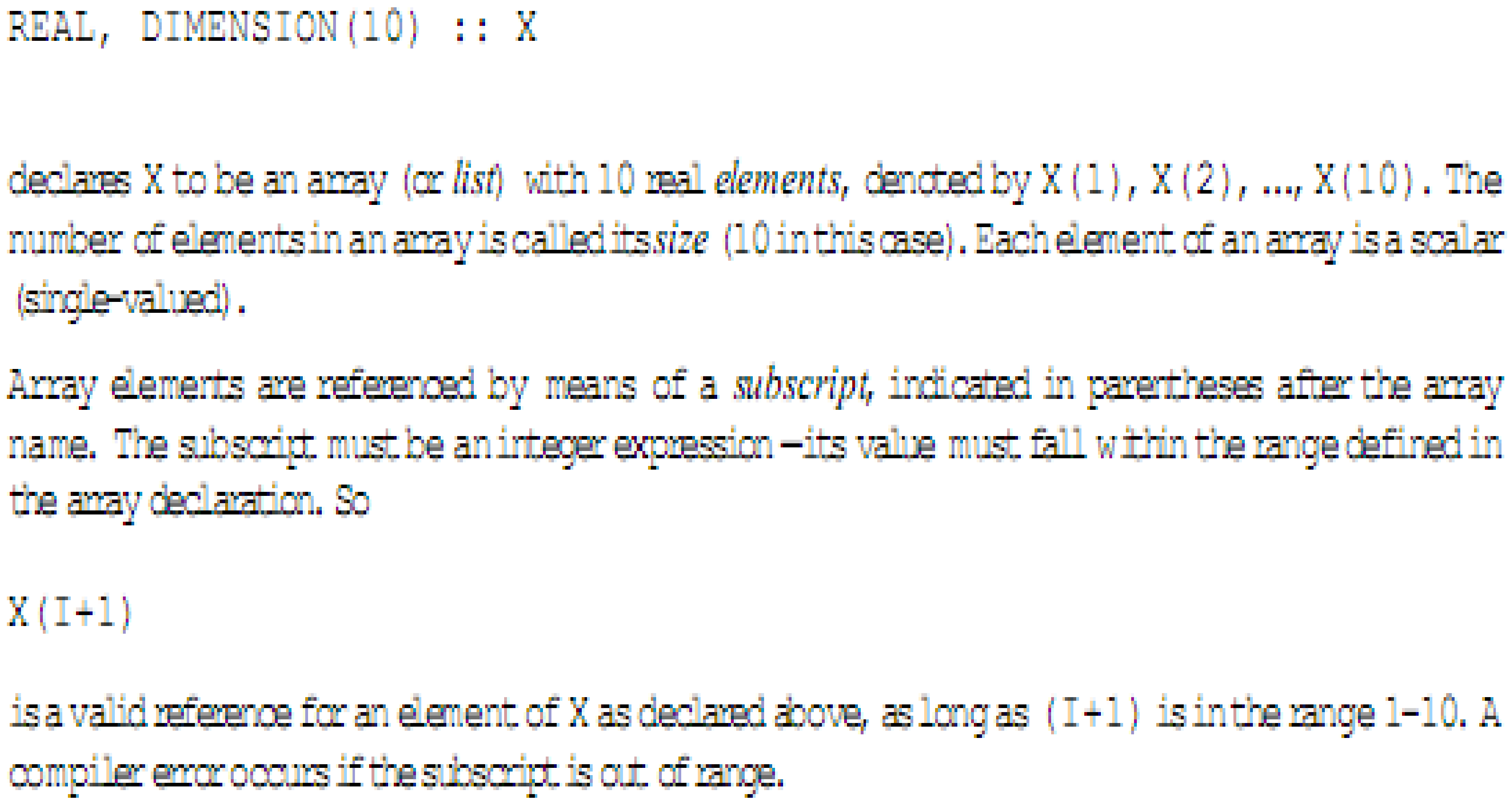
```
Ahmed, A B           200
Baker, C D           500
Essop, S A           700
Jansen, G M         1000
Smith, Q G          1500
```

Arrays

- Arrays are perhaps the most important statements in Fortran.
- Most variables in real life can have more then one component. For example in Fluid Dynamics, a velocity of a particle may have an X, Y and even a Z component.
- Hence, you use arrays to define more then one value to each variable.
- Some example would be V(x,y,z) or T(alpha,beta) etc.

One Dimensional Array

- If a statement has only one dimension, then it is called a one dimensional array



```
REAL, DIMENSION(10) :: X
```

declares X to be an array (or *list*) with 10 real *elements*, denoted by X(1), X(2), ..., X(10). The number of elements in an array is called its *size* (10 in this case). Each element of an array is a scalar (single-valued).

Array elements are referenced by means of a *subscript*, indicated in parentheses after the array name. The subscript must be an integer expression—its value must fall within the range defined in the array declaration. So

```
X(I+1)
```

is a valid reference for an element of X as declared above, as long as (I+1) is in the range 1–10. A compiler error occurs if the subscript is out of range.

Array Boundaries

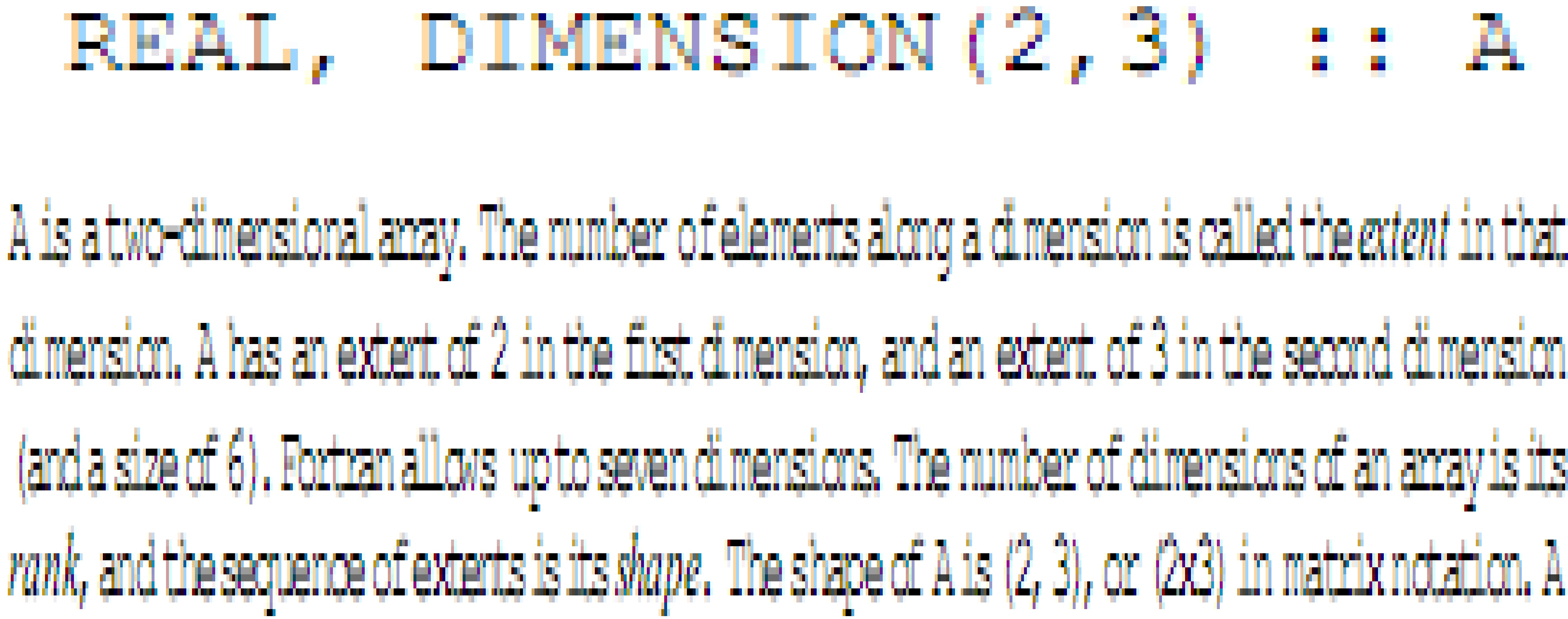
By default arrays have a *lower bound* of 1 (the lowest value a subscript can take). However, you can have any lower bound you like:

```
REAL, DIMENSION(0:100) :: A
```

declares A to have 101 elements, from `A(0)` to `A(100)`. The upper bound *must* be specified; if the lower bound is missing it defaults to 1.

Two Dimensional Array

- If an array has two dimensions, then you can think of it as a matrix. Its two dimensions can be separate from each other. (One dimension may be different in size then other)



```
REAL, DIMENSION(2,3) :: A
```

A is a two-dimensional array. The number of elements along a dimension is called the *extent* in tha dimension. A has an extent of 2 in the first dimension, and an extent of 3 in the second dimensio (and a size of 6). Fortran allows up to seven dimensions. The number of dimensions of an array is its *rank*, and the sequence of extents is its *shape*. The shape of A is (2, 3), or (2x3) in matrix notation. A

Multi-Dimensional Arrays

- In FORTRAN, you can have up to 7 dimensions in an array.
- Let's say that you want to describe the flow property of a particle in 3 dimensions for a grid that has 4 points in the X and Y axis and with 3 points in the Z axis. Then, you would have to state :

  REAL DIMENSION (4,4,3) :: V
- Then, you would write V(i,j,k) or V(1,1,2) or V(i,j,1) or V(i,2,3) etc. to determine the property at a certain grid point.