CSC 201 Week Five Lecture Note

FORTRAN LANGUAGE

CLASSES OF DATA

Computer programs, regardless of the language in which they are written, are designed to manipulate data of some kinds. CONSTANTS and VARIABLES, and their terms are used in FORTRAN in almost the same sense as in mathematics.

CONSTANTS

A constant is a quantity whose value does not change during program execution. It may be of numeric, characters, or logical type.

INTEGER CONSTANTS - A is integer constant is literally an integer, i.e., a whole number without a decimal point. It may be zero or any positive or negative value.

RULES FOR FORMING INTEGER NUMBERS

1. No decimal point
2. No fractional values.
3. Used as counter, exponents and identification number.
4. Results of integer arithmetic reduced to next-lower integer.
5. In general, the maximum magnitudes is 2^{31}- 1 a 10 digit number (depending on computers but some used 9 digits)
6. Remainders resulting from division are lost or truncated.

   14/3 value is 4

   5/6 value is 0

   19/5 value is 3
7. No commas are to be included in a string of digits.
8. Spaces with a constant are allowed but their use is discouraged as it is error prone
9. The minus sign must precede a negative constant, a plus sign is optional and an unsigned constant is considered positive.

Example:- 25 0 -7 +15274

But the following are not valid integer constants.

18.0	Contains a decimal point
- .284	Contains a decimal point
10,235	Contains a comma
--7	Not preceded by a single algebra
12345678995	Too large.

REAL CONSTANT – Also called a floating points constant, is a constant with a decimal point and may have a fractional part.

Examples: 18.3 -163.0 42.0 4.0125 (they are all valid)

While the followings are invalid

1,465.3	Contains a comma
-56	Contains no decimal point.

RULES FOR FORMING REAL CONSTANT

1. Decimal point must be included
2. Should not be used as an exponent for integer numbers.
3. Remainders resulting from division are not lost.

4.Ø/3.Ø	Value is 1.3333333
4.Ø/5.Ø	Value is Ø.8ØØØØØ
17.Ø/5.Ø	Value is 3.4ØØØØØ
Ø.5e2/ Ø.2E1	Value is 25. ØØØØØ

Other types of constant are double precision and complex but they are not treated here.

VARIABLES

A variable name, or simply variable, is a name used to identify data stored in a memory location whose contents may change during program execution.

Integer Variable Name:– A variable beginning with any of the letters I, J, K, L, M, N is assumed to be an INTEGER Variable.

Real-Variable Names: – A variable beginning with any letter except I, J, K, L, M or N is assumed to be a REAL Variable.

RULES FOR FORMING VARIABLE NAMES

1. First character must be a letter
2. Subsequent letters may be any combination of letters and digits (other characters cannot be used).
3. Variable names must not exceed six characters (for most computers)
4. Integer-Variable names must begin with letters l to N only.
5. Real variable names must begin with any other letter (A-H or O-Z)
6. Blank spaces may be inserted to improve readability and do not count as one of the six allowable characters.
7. The same name must not be used for more than one variable in a given program.
8. Before a variable name can be used in computation, it must be assigned a numerical value by an assignment statement, READ statements, or a DATA statement.
9. Term that are part of the FORTRAN vocabulary e.g. READ, WRITE, REAL, IF, (reserved words should not be used as variable names.

	Integer variable names	Real Variable names
Designator	I, J, K, L, M,N	A-H, O to Z
	NUMBERS NUM NO	RADIUS RAD R
	MAN 1 MOL M	AREA ARC A
	ITEMS JOHN J	CIRCUM CIR C
	NUM 1 LAGOS KILO2	TIM T1 T2

The following are invalid variable name.

KONTAGORA	exceeds 6 characters
READ	FORTRAN Reserved word
KILO –1	Contains – (hyphen)
TIME .T	Contains special character not a letter.

OPERATIONS

Numeric data can be manipulated using arithmetic operations, character data using concatenation and substring operations, and logical data using logical data using logic operation. When referring to a combination of variables and constants together with operation symbols, we use the phrase expression.

Arithmetic Operation

The following six arithmetic operations are available in FORTRAN.

1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exponentiation
6. Negation

The first five arithmetic operations are all binary operations in the sense that two operations are required while the last operation is called Unary operation because only one operand is required. E.g. To indicate the negation of a variable X write.

-X or (-X).

When two constants or variable of the same type are combined, using one of the four arithmetic operations (+, -, *, /) the result will be of the same type as the operands.

For example, the sum, difference and product of the integers 8 and 6 will be the integers 14, 2 and 48 respectively. Integer division in FORTRAN, say, evaluation 15/4 yields instead of 3.75, 5/6 yields Ø and –5/2 yields –2 (these are all integral part of the quotient. The fractional part of the quotient is deleted simply because the results are real in value.

It is also possible to combine an integer quantity with a real quantity, using the four arithmetic operations (+, -, *, /). Whenever one of the operands of operations is an integer and the other real, the integer is automatically converted to its real equivalent, and the result is of real type. Thus,

5. + 4 = 5. + 4. = 9.

5. + 3 = 5.* 3. = 15

5./4 = 5./4 = 1.25

Such operations involving different types of numeric operands are called mixed –mode operations.

The manner in which exponentiation is performed depends upon the type of the exponent. E.g.

2* *3 = 2* 2*2 = 4 * 2 = 8

1.5 * *2 = 1.5 * 1.5 = .25

(-3.Ø) * * 2 = (-3.Ø) * (-3.Ø) = 9. Ø

But, if the exponents is a real number the exponentiation of any constant/number, integer or real, is computed using logarithms and hence, since logarithms of negative values are not defined, the negative number raised to a real power is undefined. Thus, even through

(-3.Ø) * * 2 yields the value 9. Ø, (-3.Ø) * * 2 .Ø – exp (2. Ø log (-3.Ø)) is undefined, since log (-3.Ø) is not defined.

PRECEDENCE – This specifies the order in which the arithmetic operations are to be carried out.

ORDER

1st parentheses

2nd function

3rd Exponentiation

4th Multiplication/Division (any one met first)

5th Addition/Subtraction (any one encountered first)

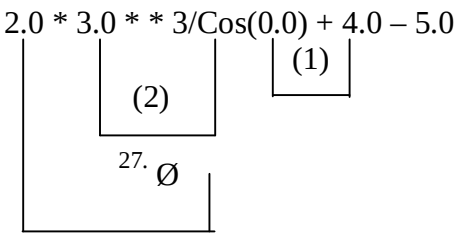
Unary plus and minus on the same level as binary addition and subtraction, that is, -2.2* *2 means – (2.2 * * 2) which yields –4.84

Examples.

Mathematical expression: $\frac{ab^3}{Cos^C} + d - e$

FORTRAN expression: A* B * * 3/ cos (c) + D – E

Assuming: A = 2.0, B = 3.0, c = 0.0, D = 4.0 E= 5.0



Mathematical expression: $\frac{a + b}{(c + d)\ \mathrm{Sin}^2\ (PI/b)}$

FORTRAN EXPRESSION: (A + B) / (C + D) * SIN (PI/B)**2

Assuming: A= 4.0, B = 6.0, C = 2.0, D = 3.0, PI = 3.141593

Then, the expression is evaluated in FORTRAN as

(4.0 + 6.0) / (2.0 + 3.0) * SIN (3.141593/6.0) **2

(a) X^{23} (b) $(X^2)^3$

FORTRAN expression (a) X * * 2 * * 3 (b) (X * * 2) * * 3

Let X = 5.0

The FORTRAN expression will be:

(a) 5.0 * * 2 * * 3

(b) (5. Ø * * 2) * * 3

CONCATENATION

This is a binary operation. This operation is denoted by the symbol // (double slash) and can be used to combine two, or more, character strings, and /or character variable e.g.

‘UN’ // ‘AAB’

Produces the character string

UNAAB

Again, if CHARACTER * 7 MODEL (here the 7 indicates 7 spaces)

And MODEL is set to 'IBM – 360'

Then MODEL // 'com'// "PUTER"

Yield the character string

'IBM – 360 COMPUTER'

SUBSTRING

This is a binary operation requiring only one operand. This is performed on character strings to extract a sequence of consecutive characters from a string. Such a sequence is called a substring of the given string.

The general form is

STRING (i:j)

Where i and j are positive integer constants, variables, or expressions such that i < j. The default value for i and j are 1 and the last position in the character string respectively.

Examples

CHARACTER * ll COURSE

And set course to 'MATHEMATICS', then

COURSE (:5) will yield 'MATHS'

And COURSE (6:) WILL YIELD 'MATICS'

Again,

COURSE (M: M + 2) if M is assigned the value 3, has the value

'THE'

Care must be taken to ensure that i is not greater than the number of characters in the character string. The value for j should be such that the sequence of characters from i through j does not extend beyond the last character in the character string.

Substring names may be used to form character expressions just as character variables are used. They may be concatenated with other character values as in

X (NAME (i:i). LT. 'A') PRINT * 'ILLEGA NAME)

LOGICAL OPERATIONS

The logical variables, or constants may be combined, using the three basic logical operations, AND., OR., and .NOT.

Examples:

A. .LT. B .AND. C .GT. D

i.e. Is A less than B and is C greater than D?. The .AND. requires that both conditions be satisfied for the expression to be true. The .OR. operator is used as follows.

E .GE. F .OR. G .LE. (H – X)

i.e is E greater than or equal to F or is G less than or equal to the value of H minus X ? The .OR. means that when either condition is true, the expression is true.

An example of an INCORRECT logical expression is

A. .AND. B .LT. C

because logical operations . AND. and .OR. relate logical expressions, not individual variable name.

Logical Expressions

Logical Expression	Meaning
A. LT. B	Is A less than B?
C. EQ. (D/E)	Is C equal to the quotient of D divided by E
X .GT. (R + S)	Is X greater than the Sum of R and S.

8.3 FUNCTIONS

FORTRAN provides a number of program modules, or built-in functions that performs such mathematical functions. These built-in functions are termed INTRINSIC FUNCTIONS. The use of an intrinsic function is specified by writing the function name followed by the expression to be operated upon (the argument) inside a set of parentheses. e.g.

The FORTRAN expression of

$X = \sqrt{y}$	is X = SQRT (Y)
$X = \lvert Y - a \rvert$	is X = ABS|Y – A|
$X = e^{y+a}$	is X = EXP(Y + A)

X = max(a,b,c) is X = AMAXI(A,B, C)

FORTRAN STATEMENT

ASSIGNMENT STATEMENT

This is used to assign values to variables and has the form

Variable = expression.

Where expression may be a constant, another variable to which a value has previously been assigned, or a formula which the computer can evaluate.

The 'equals' i.e. '=' in the assignment statement is not interpreted the same sense as in mathematics but must be read as "is assigned". Thus A =B is not the same as B = A since the first assigns B to A while the latter assigns A to B leaving B unchanged.

This can be interpreted as A ⟵ B and B ⟵ A respectively.

Consider the assignment Statement

SOLA = SOLA + BUNMI

This instructs the computer to add to the value of the variables SOLA the value of the variable BUNMI and assign the result as the new value of the variable SOLA. If SOLA and BUNMI contains.

SOLA	BUNMI
8.0	16.0

SOLA = SOLA + BUNMI

SOLA = 8.0 + 16.0

SOLA = 24.0

SOLA
24.0

The former content of SOLA (i.e. 8.0 is destroyed after the assignment statement given way to the new value 24.0.

The variable to be assigned a value must appear on the left of thee equal sign and that a legal expression appears on the right of the equal sign in the assignment statement.

The following are invalids.

18 = M	Variable M is on the right hand instead of the left hand
X + 4.3 = 3.14	Numeric expression should not appear to the left of equal sign.
STRING = 4118	The number 4118 is an illegal expression
A = B = 7	B = 7 is an illegal expression
C = '3' * '9'	'3' * '9' is an illegal expression
STRING = 10	numeric value 10 is assigned to a character
M = '10'	a character constant '10' is assigned to a numeric variable

ASSIGNMENT TO NUMERIC VARIABLES

When assignment statement is used to assign value to a numeric variable, it must take care of the type of the numeric variable and the value assigned to the variable.

If an integer-value expression is assigned to a real variable, the value is converted to a real constants and then assigned to the variable e.g. if N = 10

```
Y = 3
X = (N + 5) / 5.0
```

Assigns the real constant 3.0 to X and real constant 3.0 to Y

If a real expression is assigned to an integer variable, the fractional part is truncated and the integer part is assigned to the variable. E.g. If X = 7.41 and I, J, K are all integer variables, then,

```
I = 3.14159
J = X/3.
K = 1./3. + 1./3
```

Would assign the integer constants 3, 2 and 0 to the variables I, J and K respectively.

ASSIGNMENT TO CHARACTER VARIABLES

It must take care of the length associated with it. e.g.

```
CHARACTER* 6 STRNGA, STRNGB* M .TRVN, PAD
```

STRNGA = ‘VRMILA’

STRNGB = STRNGA // ‘AGRAWAL’

From here, there are 6 character (spaces/ length) assigned to STRNGA, TRVN, and PAD while STRNGB is assigned 14 characters.

The values ‘URMILA’ and ‘VRMILA’ ‘AGRAWAL’ are assigned to STRNGA and STRNGB respectively.

Here the declared length of the variables is greater than the length of the value being assigned, values are padded with blanks. Conversely, if the declared length of the variable is less than the length of the value being assigned, the value is truncated to the size of the variable and the leftmost characters are assigned. Thus the statement

PAD = ‘JOY’

Will assign the value ‘JOYbbb’ to the variable PAD, where b denotes blank, and the statement

TRUN = ‘COME – AGAIN’ assigns

The value ‘COME-A‘ to the variable TRUN

INPUT AND OUTPUT STATEMENTS

FORTRAN provides two types of input/output (I/O) statements; formatted and list-directed (Unformatted, or format-free). In the case of formatted, the programmer must specify the format in which the data is input, or, output. But in the case of unformatted, certain predetermined formats which match the types of items in the input/output list are automatically provided by the compiler.

LIST DIRECTED INPUT STATEMENT.

It is of the form

READ *, V_1, V_2, V_3, ... Vn

Where V_1 refers to the first input data item, V_2 to the second and so on. The variable names must be separated by commas. The space between READ and * is optional. Additional spaces may be inserted between a comma and a variable name.

Example:

READ *, ADE, OLU, JOHN.

READ * KOLA

READ* A, B, C or

READ* A

READ* B

READ* C

Another way by which READ statement (without a Format) is coded as

READ (* *) V_1, V_2 ..... Vn

The first asterisk indicates that the data are to be manually keyed in via keyboard. The second asterisk indicates that the data supplied will be unstructured (without FORMAT specification).

V_1, V_2 ..... Vn represents the lists of variable names to be read

E.g. READ (* , *) ADE, OLU

READ (* ,*) OJO

LIST DIRECTED OUTPUT STATEMENT

It is of the form

PRINT *, V_1, V_2 ... Vn

The same rules that apply to READ is also applicable to PRINT.

Another way by which list-Directed WRITE statement can be output is

WRITE (*, *) V_1, V_2, ... , Vn

This is also similar to the READ (*, *) V_1, V_2, ... , Vn.

The first asterisk indicates that the results are to be displayed on the screen. The second asterisk indicates that the compiler will structure the output. The programmer has no control over how the output is structured.

The first output column is reserved for a vertical control character which is always a blank with list-directed output.

Integer numbers are right justified in a 13 column field width, i.e. the number will be placed as far to the right in the field as possible, with any blanks in the field shown to the left of the

number. The 13-column field includes up to 10 digits, the algebraic sign (displayed only when negative), and at least two leading blanks.

Examples:

WRITE (*,*) ADE, OLU

WRIRE (*,*) OJO

STOP AND END

STOP: The STOP statement is used to specify that the program execution is to be brought to a halt. The form of the statement may be.

STOP

Or

STOP CONSTANT

Where constant is an integer constant with five or fewer digits, or a character constant. The use of this word in FORTRAN 77 is optional

END: Every FORTRAN program and subprogram must conclude with an END statement that indicates to the compiler that the end of the program unit has been reached. This statement is of the form:

END

This must be the last statement of the program unit.

There are 3 major differences between these two keywords (END and STOP).

1. STOP is an instruction to the computer to stop the program, where the END statement is an instruction to the compiler that there are no more statements in the program unit.
2. The STOP statement may be used anywhere in a program unit to stop execution and run time. The END statement is used only as the physical end of a program unit.
3. The STOP statement may include a reference number, whereas the END statement does not contain anything but the word END.

RELATIONAL EXPRESSION

A relational expression consists of two arithmetic expressions, or character strings, connected by a relational operator that defines the nature of the condition, or relation to be tested.

Relational Operator in FORTRAN

Relational Operator	Application in FORTRAN	Meaning
.LT	A. LT. D	A is less than B
.LE	A. LE. B	A is less than, or equal to B
.EQ	A. EQ. B	A is equal to B
.NE	A. NE.B	A is not equal to B
.GT	A. GT.B	A is greater than B
.GE	A. GE. B	A is greater than, or equal to, B

The value of a relational expression is one of the logical values. TRUE or FALSE.

NUMBER COMPARISONS

The operators can be used to compare numerical values. The numerical order is used when two numbers are compared.

For example

If X = 20.00 Y = 4.0 and M = 5.0

1. X + 5 .GT. M * Y

i.e. 25.0 .GT. 20.0 (The value is true)

2. X/Y .EQ. M

i.e. 20.0 / 4.0 = 5.0.

5.0 .EQ. 5.0 (The value is true)

3. 3 * (X + Y)/4 .LE. 2 * X + 5 * Y – 42.01

i.e. 18.0 . LE. 17.9 (The value is true)

CHARACTER COMPARISONS

We may use ASC11 (American Standard Code for Information Interchange) and EBCDIC (Extended Binary Coded Decimal Interchange Code) Codes to obtain the collating sequences for the characters for these codes. Thus, the relational expressions

'B' .GT. 'A'

'X' .LT. 'Y'

are all. True. Since 'B' must follow 'A' and 'X' must precide 'Y'

However,

'A' .GT. '1'

'*' .LT. 'C'

depends on the collating sequences used in a particular computer. The first is true and second false for ASCII but both are false for EBCDIC Code.

'COMPUTER' .GT. 'COMPUTER'

Is true since a blank character (b) precedes all letters in every collating sequences

'SHOLA' .EQ. 'SHOLA' is true

GOTO STATEMENT (Unconditional)

Unconditional Transfer: The unconditional transfer of control can be accomplished by writing the statement.

GOTO n

Where n is a statement number. This tells the computer to go, unconditionally, to the part of the program beginning with the statement labeled.

Obviously, the statement labeled n must be executable.

The Computed GOTO Statement

The computed GOTO statement is a conditional transfer statement that transfers to one of several executable statements, depending on the value of an integer numerical indicator. The general form for the computed GOTO statement is:

GOTO ($n_1, n_2, n_3,...\Pi_1$) intexp

Where $n_1, n_2, n_3, ...\Pi_1$ are the first, second, third etc., statement labels in a list of statements label and intexp is an integer –variable or integer –arithmetic expression (the numerical indicator) whose value determines to which of these labeled statements (first, second, third, etc) transfer parenthesis is optional and may be omitted. If the value of intexp is less than one (zero or negative integer) or greater than the number of statement labels in the list, the computed GOTO statement is ignored and execution continues with the next executable statement following the computed GOTO statement. Example.

GOTO (10, 25, 50, 35), K-J

Transfer is made to statement 10 if K – J = 1 ; to statement 25 if K- J = 2; to statement 50 if k – J = 3; and to statement label 35, the value K- J = 4. Since these are four statement labels, the value K – J must equal 1 through 4, inclusive, if the computed GOTO statement is to be executed, and the values for K and J must have been assigned earlier in the program.

IF Statement

There are three forms of IF statements Logical IF, block IF, and Arithmetic IF. The blocks IF is preferred except for simple selective structures, in which case the logical IF is used. The arithmetic IF is error prone and hence, its use is discouraged.

Logical IF Statement

The logical IF statement makes possible the conditional execution of a single statement depending upon whether a given logical expression is true. It has the form:

IF (Logical expression) statement where the statement is any executable FORTRAN statement, but it cannot be another IF statement and END statement, or a DO statement.

When a logical IF statements is executed, the value of the logical expression is determined and if it is true, the designated statement is executed (except when there is another transfer of control), the statement following the logical IF statement is executed next. But if the expression returns false, the designated statement is not executed, and the next statement to be executed is the one following the Logical IF statement.

Examples:

IF (TOKS .LE. 105) PRINT *, TOKS

Toks is printed only if it is ≤ 105.

IF (SALARY. LT. 3000.0 AND. STATUS .GT. 2.0) INC = INC + 1

Here, 1 is added to INC only if SALARY < 3000 and STATUS > 2.

IF (SCORE .GT. 90.0 .OR. GRADE .GE. 4.1) GOTO 99 Here, again, the program control is transferred to the statement labeled 99, if SCORE > 90 or GRADE ≥ 4.1

BLOCK IF STATEMENT (IF, THEN, ELSE, AND , END IF Statement)

The logical IF statement allows the programmer to control logic flow while minimizing the under transfer steps. The general form is:

IF (Logical expression) THEN

-)
-)Statement set 1
-)
-)

ELSE

-)
-) Statement set 2
-)

END IF

Basic Block IF structure.

In this structure, the logical expression in the IF THEN statement is identical to the form (s) used logical IF statement. Statement set 1 and statement set 2 normally consist of one or more statement to be executed. If the logical expression is true control is transferred to the first statement in statement set 1 and the statement between the IF THEN statement and the ELSE statements are executed. After the completion of the statements within statement, control is transferred to the statement immediately following the END IF statement. The statements in statement set 1 are ignored and control is transferred to the statement of statement set 2. When completion of the execution of the statements within statement set 2 is accomplished, control is transferred to the statement immediately following the END IF statement.

Example:

To calculate the square root of the difference between two numbers:

```
      Solution
      N= 0
10    READ (5, *) A, B
      N = N + 1
      IF (A – B.GE. 0.0) THEN
      ROOT = (A – B) * * 0.5
      WRITE (6,*) 'THE REAL ROOT OF A – B IS', ROOT
      ELSE
      ROOT = (B – A) * * 0.5
      WRITE (6,*) 'THE IMAGINARY SQUARE ROOT OF B – A IS', ROOT
      END IF
      IF (N. NE.5) GOTO 10
      STOP
      END
```

8.5 REPETITIVE STRUCTURES.

This is also known as interactive structure or program loop. In a repetitive structure, a set of program statements appears only once in the program. This results in a substantial reduction of the number of statement consists of an entry point including initialization of certain variables, a repetition, or loop, body, and an exit point.

The number of repetitions in a structure can be condition-controlled, or counter-controlled.

IF LOOP

The simplest of the repetitive structures is the If Loop in which the number of repetitions can be conditional controlled, or counter-controlled. It involves the use of the IF and the GOTO statements.

Example,

```
      N = O
1OO READ *, A, B,  SUM
```

```
SUM = A + B
PRINT *, A, B, SUM
N = N + 1
IF (N .LE. 10) GOTO 100
STOP
END
```

In the above program, N is used as a counter. In this case, it counts the number of interaction and terminates it when it reaches 10.

DO LOOP

The explicit DO statement is an executable statement that causes a portion of the program to be repeated a given number of times – a procedure called looping. The statements repeatedly executed during the looping procedure are referred as the DO Loop. The DO loop is initiated and controlled by an executable statement designated by statement label in the DO statement.

The general form is:

DO i j = k,m,n

Where i represents a statement label identifying the terminal statement (always an integer constant)

j represents an integer-variable name called the index

k and m represent, respectively, the initial and limiting integer values to be assigned to the index j

n represents the integer increment of the index (other than zero)

The index j is always an integer-variable name. The initial and limiting value k and m may be integer constants integer-variable names, or integer-arithmetic expressions evaluated positive, negative, or zero. The increment n may be an integer constant, an integer-variable name, or an integer-arithmetic expression, evaluated positive or negative, but not zero.

It is essential that a programmer know how many times a loop will be repeated. The number of increment is the difference between the limiting value m and the initial value k divided by the increment n, truncated to integer form. Therefore,

$$NR = \frac{m - k}{n} + 1$$

where NR = number of repetitions (or iterations) of the DO Statement

k = initial value specified in the DO statement

m = limiting value specified in the DO statement

n = increment specified in the DO statements.

Example,