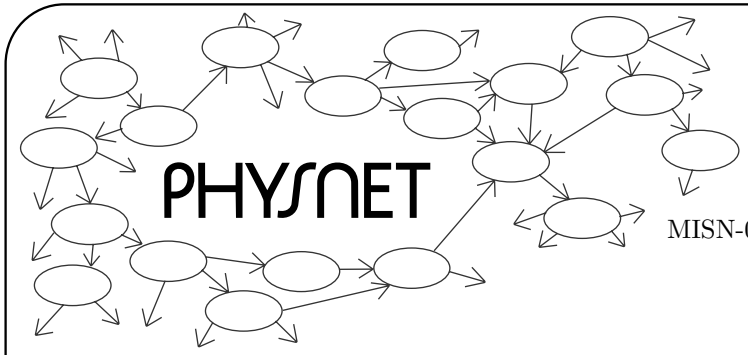

MISN-0-346

REVIEW OF ELEMENTARY FORTRAN

PROGRAM

BEGIN

WRITE

END

CONTINUE

20 CONTINUE GOTO
READ(10,*) STOP
EXIT FUNCTION
C DO 20 I =
SUBROUTINE IF CALL RETURN
101 FORMAT

Project PHYSNET Physics Bldg. Michigan State University East Lansing, MI

REVIEW OF ELEMENTARY FORTRAN

by

Robert Ehrlich
George Mason University

Title: **Review of Elementary Fortran**

Author: R. Ehrlich, George Mason Univ., Physics Dept., Fairfax, VA

Version: 2/1/2000 Evaluation: Stage 0

Length: 1 hr; 28 pages

Input Skills:

1. Vocabulary: program, algorithm, operation, flowchart, source program, machine language, floating point, compilation (MISN-0-370).
2. Read and write simple flow diagrams (MISN-0-370).
3. Successfully submit a batch job or run an interactive program on your computer (see your computer's Operating Guide).

Output Skills (Knowledge):

K1. Vocabulary: constant, variable, expression, function, statement numbers.

Output Skills (Rule Application):

R1. Recognize and write FORTRAN arithmetic expressions involving real and integer constants and variables.

R2. Recognize and write FORTRAN input/output statements using READ, WRITE, and simple FORMAT statements.

R3. Recognize and write FORTRAN branching statements using GOTO and IF statements.

R4. Recognize and use the FORTRAN statements CALL EXIT, COMMENT, CONTINUE and END.

Output Skills (Project):

P1. Run a canned program to compute the roots of a quadratic equation, with and without checking the sign of the discriminant.

P2. Write, enter and run a program to compute the height to which you can jump vertically in a gravitational field, using two sets of input data for initial velocity and the gravitational acceleration.

External Resources (Required):

1. A computer with FORTRAN.

THIS IS A DEVELOPMENTAL-STAGE PUBLICATION
OF PROJECT PHYSNET

The goal of our project is to assist a network of educators and scientists in transferring physics from one person to another. We support manuscript processing and distribution, along with communication and information systems. We also work with employers to identify basic scientific skills as well as physics topics that are needed in science and technology. A number of our publications are aimed at assisting users in acquiring such skills.

Our publications are designed: (i) to be updated quickly in response to field tests and new scientific developments; (ii) to be used in both classroom and professional settings; (iii) to show the prerequisite dependencies existing among the various chunks of physics knowledge and skill, as a guide both to mental organization and to use of the materials; and (iv) to be adapted quickly to specific user needs ranging from single-skill instruction to complete custom textbooks.

New authors, reviewers and field testers are welcome.

PROJECT STAFF

Andrew Schnepp	Webmaster
Eugene Kales	Graphics
Peter Signell	Project Director

ADVISORY COMMITTEE

D. Alan Bromley	Yale University
E. Leonard Jossem	The Ohio State University
A. A. Strassenburg	S. U. N. Y., Stony Brook

Views expressed in a module are those of the module author(s) and are not necessarily those of other project participants.

REVIEW OF ELEMENTARY FORTRAN

by

Robert Ehrlich
George Mason University

1. Introduction

1a. Overview. This module is a review of some of the fundamentals of FORTRAN IV. FORTRAN is one of the languages used in communication with the computer. It is a language that is well-suited to solving many scientific and engineering problems. FORTRAN, which stands for FORmula TRANslation, has gone through many modifications since its introduction in the early 1950's. We shall be exclusively concerned with the "FORTRAN IV" (also known as FORTRAN 66) subset of the FORTRAN version in present use, FORTRAN V (also known as FORTRAN 77). The basic elements of FORTRAN statements are: constants, variables, operations, expressions, and functions. We shall first discuss these basic elements, and then see how they can be combined into FORTRAN statements, and finally see how the statements can be combined into programs.

1b. FORTRAN Programs. A FORTRAN program consists of a series of statements, each of which is an instruction to the computer. There are four general types of FORTRAN statements. One specifies the execution of various arithmetic or logical operations: these are usually the heart of a program. Another type of statement calls for input of data or output of results. Statements of these first two types are executed in the order in which they appear in the program: this is an important point. A third statement type can alter the sequence in which other statements are executed. The fourth type of statement provides documentary information about the procedure without specifying that any computation be done.

1c. FORTRAN Compilers. The collection of FORTRAN statements comprises a "source program." These statements need to be translated into a language that the computer can "understand"—machine language. The process of the translation of source code into machine language is known as "compilation." Compilation is necessary because the computer is only capable of carrying out very simple steps and each FORTRAN statement may include a very large number of such steps.

2. Elements of a Fortran Statement

2a. Constants: Integer or Real. A constant in a FORTRAN statement appears as a value, a specific number: this is in contrast to a variable, which has a name and may take on a range of values. There are two kinds of constants that are commonly used in FORTRAN: "integer" constants and "real" constants.

An integer constant may be any whole number (positive, negative or zero), and it may be restricted to some maximum number of decimal digits that depends on the computer. On some computers the largest allowed integer is $2^{16} - 1 = 32,767$.

Real numbers are represented inside the computer in floating point form which consists of a mantissa and an exponent. This is similar to scientific ("power of ten") notation. Due to computer storage limitations there are limits on both the number of digits of precision in the mantissa (often eight), as well as the maximum range of exponents (often between 10^{-40} and 10^{+40}). Greater ranges in the number of digits are possible on most computers, using double precision, but we will postpone consideration of this.[1] An integer constant is distinguished from a real constant by the absence of a decimal point. The following are acceptable integer constants since they do not contain decimal points:

$$0 \quad -5678 \quad 1000 \quad +3$$

Real constants may be written with or without a power of ten. The power of ten is designated by the letter E followed by a positive or negative exponent. These are acceptable real constants:

$$15.123 \quad 2.0 \quad -3.14$$

$$-.000352 \quad 3.000 \quad 0.1$$

$$-0.1\text{E}7 \quad 5.0\text{E}{-12} \quad -0.7\text{E}{-22}$$

The following real constants would be unacceptable in most versions of FORTRAN:

12,562.3 (commas not allowed)

[1] See "Advanced Features of FORTRAN" (MISN-0-347).

+592 (decimal point missing)

1.2E+83 (exponent too large)

5.2E7.3 (only integral exponents allowed)

E−7 (exponent alone not allowed)

1E−3 (decimal point missing)

2b. Variables: Integer or Real. A variable refers to any quantity used in FORTRAN that is referred to by name rather than as a specific number. Variables may take on many values during the execution of a program, while constants are restricted to one value. Like constants, variables may be either integer or real. The distinction between real and integer variable is an important one, given the difference in the way arithmetic operations are carried out for the two types of variables or constants.

An integer variable may only take on integral values. The name of an integer variable has one to six letters or digits, the first of which must be either I, J, K, L, M, or N. Acceptable integer variable names include: J, KLM, L234, MATRIX.

Real variables are represented in the same manner as real constants, namely in scientific notation. The name of a real variable has one to six letters or digits, the first of which is a letter other than I, J, K, L, M, or N. Acceptable real variable names include: X, VAR, F001, VECTOR. The choice of variable names is completely up to the programmer. You should usually choose names that remind you of the variables' meanings.

2c. Arithmetic Operations and Expressions. There are five basic arithmetic operations in FORTRAN, each of which is designated by the symbol shown below:

addition	+
subtraction	−
multiplication	*
division	/
exponentiation	**

A "FORTRAN expression" is a rule for computing a numerical value. It may consist of one or more variables and constants, perhaps combined

with suitable arithmetic operations. Examples of valid expressions include:

Expression	Meaning
K	The value of the integer K
X∗Y	The value of the product of X and Y
(A+B)/(C∗∗2)	The value of the sum of A and B divided by the square of C
PI/2.0	The value of the variable PI divided by 2.0

As indicated in the third example, parentheses may sometimes be used to convey the order of operations. In the absence of parentheses there is an assumed hierarchy of operations: all exponentiations are performed first, then all multiplications and divisions and then all additions and subtractions. Within a given class (e.g. multiplication and division) the order is from left to right in the expression. Thus, the meaning of (A+B)/C∗∗2 would not differ from the third example, but the meaning of A+B/C∗∗2 would differ from it. In the latter case, the value of A is to be added to the value resulting from B divided by the square of C. There are many rules which valid expressions must obey. These rules are illustrated in Table 1 which shows examples of correct and incorrect FORTRAN expressions. Here are some more rules:

1. Do not use two operation symbols next to each other, such as: A∗−B. (In this context, note that the ∗∗ representing exponentiation is considered one symbol.)

2. Do not use ambiguous expressions such as A∗∗B∗∗C, which could mean either A∗∗(B∗∗C) or (A∗∗B)∗∗C.

3. Do not mix real and integer quantities in the same expression, except when raising a real quantity to an integral power.

4. Do not use parentheses alone to convey multiplication. For example (A+B)∗(C+D) is valid but (A+B)(C+D) is not.

Table 1: Some Correct and Incorrect FORTRAN Expressions.

Mathematical Notation	Correct Expression	Incorrect Expression
$a \cdot b$	A∗B	AB
$a + 2$	A + 2.0	A+2
$\frac{a \cdot b}{c \cdot d}$	A∗B/(C∗D)	A∗B/C∗D
$((a + b)/c)^{2.5}$	((A+B)/C)∗∗2.5	A+B/C∗∗2.5
$a/[1 + b/(1.5 + c)]$	A/(1.0+B/(1.5+C))	A/(1.0+B/1.5+C)
$a \cdot (-b)$	A∗(−B)	A∗−B

2d. Mathematical Functions. The FORTRAN language provides for the use of a number of common mathematical functions: some of these are listed in Table 2. To use a function in an expression you need only write the pre-assigned name given in Table 2 and enclose the argument in parentheses. For example, if we wish to compute the sine of an angle X we need only write SIN(X). The argument of a function need not be a single variable or constant: if we wish to find the square root of $b^2 - 4ac$, we can write SQRT(B∗∗2−4.0∗A∗C).

Table 2: Some Of The Mathematical Functions In FORTRAN.

Function	FORTRAN name
exponential	EXP
natural logarithm (base e)	ALOG
common logarithm (base 10)	ALOG10
sine of an angle in radians	SIN
cosine of an angle in radians	COS
hyperbolic tangent	TANH
square root	SQRT
arctangent	ATAN
absolute value	ABS

3. Arithmetic Assignment Statements

3a. The Equal Sign in FORTRAN. Any FORTRAN statement containing an equal sign is known as an assignment statement. The general form of the assignment statement is:

variable = expression,

where "expression" stands for any mathematical expression involving variables, constants and mathematical functions. Thus the following are valid assignment statements:

X = 1.0

C = A + B

Z = (A + 1.0)∗∗2 + C∗X

The left hand side must be a variable so these are invalid FORTRAN statements:

1.0 = X

A + B = C

(A + 1.0)∗∗2 + C∗X = Z

The reason that a single variable must be all that appears to the left of the equal sign becomes clear when one realizes that the assignment statement is an instruction to the computer to:

1. find a numerical value for the entire expression to the right of the equal sign, and then to
2. store that value in the memory location reserved for the variable that appears to the left of the equal sign.

Each variable appearing in a FORTRAN program is associated with a different location in the computer memory. For example, the statement

C = A + B

is an instruction to add together the numerical values stored in the memory locations reserved for variables A and B, and then store the result in the memory location reserved for the variable C. For the statement to be meaningful, it must come after other statements that instruct the computer to store actual numerical values in the memory locations reserved for variables A and B. This might occur in several previous assignment statements. For example:

A = 2.0

B = A + 1.0

C = A + B

After these three statements are executed, the memory locations corresponding to the variables A, B, and C contain the numerical values 2.0, 3.0, 5.0, respectively. Check these numbers!

If a variable appears to the left of an equal sign in more than one assignment statement, its value is modified as each relevant statement is executed. For example, after the computer executes the two statements

X = 1.0

X = X + 1.0

X has the value 2.0. Note that the statement X = X + 1.0 is a perfectly valid assignment statement: it means: "Add 1.0 to the present value of X and call that the new value of X." The use of the equal sign in FORTRAN is quite different from its use in mathematics since, as an equation, $x = x + 1$ is meaningless. The real meaning of the assignment statement might have been better conveyed if the symbol "←" were used in a place of the equal sign:

X ← X + 1.0

In any case, even if it is not always the most appropriate symbol, the equal sign is an established part of the FORTRAN language.[2]

The order in which a series of FORTRAN statements appears can be important, since the statements are normally executed in the order in which they appear in the program. As an illustration, suppose the following three statements appear in a program:

X = 1.0

X = X + 2.0

X = 2.0∗X

After the computer executes these statements, X has the value 6.0. Now suppose the second and third statements are interchanged:

X = 1.0

X = 2.0∗X

X = X + 2.0

[2]This peculiar use of the equal sign for both equality and assignment is also part of the BASIC language. However, the corresponding assignments in PASCAL and MUSIMP would be, respectively, X: = X + 1 and X : X + 1.

After the computer executes these statements, X has the value 4.0.

3b. Integer and Real Arithmetic. The distinction between integer and real quantities is important because arithmetic computer operations are carried out differently in the two cases.

In the case of integers, each arithmetic operation yields a result which is truncated to the nearest lower integer. This is known as integer arithmetic or fixed point arithmetic. For example, the result of the division 7/2 would be 3 according to the rules of integer arithmetic. The loss of precision that occurs when integer arithmetic is used can sometimes be used advantageously.[3]

In the case of reals, arithmetic operations are performed in the usual way according to real "floating point" arithmetic. Although the result of the division 7/2 would be 3 according to the rules of integer arithmetic, the result of 7.0/2.0 would be 3.5 according to the rules of real arithmetic.

All the variables and constants which appear in an assignment statement to the right of the equal sign must be of the same type, either all real or all integer. The four possibilities consistent with this rule are:

case #	variable to left of "equals" sign	expression to right of "equals" sign
1.	real	real
2.	integer	integer
3.	real	integer
4.	integer	real

Here is an example of each case:

1. real = real: C = 5.0∗A/B
   If this statement appears in a program after A and B have been assigned values of 2.0 and 3.0, respectively, then the execution of this statement causes C to be assigned a value 3.3333333. (The exact number of significant digits depends on the word size of the particular computer).

2. integer = integer: K = 2∗(L/2)/3
   If L has been previously assigned a value of 5, then the execution of this statement causes K to be assigned a value of 1 (obtained by the

[3]By contrast with FORTRAN, integer arithmetic is more accurate in MUSIMP, where up to 600 digits may be kept in an integer.

following sequence of integer arithmetic operations: $5/2 = 2$, $2 \times 2 = 4$, $4/3 = 1$).

3. real = integer: X = 5∗K/2 + 1
   If K has been previously assigned a value of 1, then the execution of this statement causes X to be assigned a value of 3.0. Note that the expression is first evaluated according to integer arithmetic and then it is converted to a floating point (real) number.

4. integer = real: K = 0.5∗A∗∗2
   If A has been previously assigned a value of 3.0, then the execution of this statement causes K to be assigned a value of 4. Note that the expression is first evaluated according to floating point arithmetic and then the result is truncated to the next lower integer. This example illustrates the one exception to the rule that all variables and constants appearing to the right of an equal sign must be of same type: the exception is that integral exponents are written without a decimal point.

4. Input/Output Statements

4a. READ Statements. The computer is instructed to input information using a READ statement. Information may be accepted from a variety of input devices connected to the computer, including a keyboard, magnetic tape, or disk. A **READ** statement needs to specify:

1. the quantities to be read into the computer,
2. the input device to be used to read the data: magnetic tape unit, keyboard, etc.,
3. the detailed format of the data as it appears on the input medium.

As an example of a complete READ statement, we have:

READ(2,100)Q1,Q2

The 2 appearing in parentheses indicates that the values to be read for the variables Q1 and Q2 are to be read from device number 2. (The assignment of particular numbers to each device is computer-dependent). The second number in the parentheses after the comma (100) is the statement number of another statement in the program, a so-called FORMAT statement, which specifies the detailed format of the data which is to be

read. A FORMAT statement in the same program as the previous READ statement might be:

100 **FORMAT**(2F10.5)

The 2F10.5 appearing in parentheses provides this information:

2:	There are two quantities to be read.
F:	They are both reals, numbers with decimal points.
10:	Each number takes up ten spaces, including blanks, decimal point, and sign.
.5:	Each number has five digits after the decimal point.

This code is discussed in more detail in "Advanced Features of FORTRAN" (MISN-0-347). Unlike most other kinds of statements, it does not matter where in the program a FORMAT statement is placed. This is because the FORMAT statement is not actually an instruction to the computer to do anything but rather a statement that is used as a reference by some READ or WRITE statement(s) in the program. A good practice is to put all FORMAT statements in one part of the program for handy reference e.g., just before the end.

4b. WRITE Statements. To instruct the computer to output certain quantities we use a WRITE statement. The correct form of the WRITE statement is very similar to the READ statement. For example,

WRITE(3,101)Q1,Q2

is an instruction to write out values for Q1 and Q2 on device number 3 according to the format specified by **FORMAT** statement number 101. Note that the same **FORMAT** statement could be referred to by more than one **READ** or **WRITE** statement if the format of the quantities being read or written is the same.

5. Branching Statements

5a. Statement Numbers. We have already seen the use of statement numbers in connection with **FORMAT** statements. Whether a statement has an identifying number is usually optional in FORTRAN. The most common reason for giving a statement a number is to be able to refer to that statement elsewhere in the program. This occurs particularly in connection with format statements and statements that alter the normal sequential flow of a program.

5b. IF Statements. The **IF** statement is the FORTRAN version of a two- or three-way branch. The general form of the **IF** statement is

IF (expression) N1,N2,N3

where "expression" stands for any FORTRAN expression, and N1, N2, and N3 are the statement numbers of three other statements in the program. The **IF** statement is an instruction to the computer to

1. find a numerical value for the expression in parentheses, and then
2. go to the statement numbered either N1, N2, or N3 for the next instruction, depending on whether the numerical value of the expression is negative, zero, or positive, respectively.

The **IF** statement thus permits a three-way branch. If only a two-way branch is desired, two of the three statement numbers N1, N2, and N3 must be set equal. For example, a two-way branch is provided by the statement

IF (Q1)30,20,30

which instructs the computer to branch to statement 30 for its next instruction if the value of Q1 is either negative or positive, and to branch to statement 20 it it is zero.

The type of ! statement that we have discussed is known as the arithmetic IF statement. Another useful type of statement is the "logical IF" statement, which we shall not discuss because it cannot be used in all versions of the FORTRAN IV language.

5c. GO TO Statements. The **GO TO** statement is the FORTRAN version of an unconditional branch:

GO TO N

where N is the statement number of another statement in the program. The GO TO statement is an instruction, or command, to unconditionally branch to the statement numbered N. Another form of the **GO TO** statement is the computed **GO TO** which has the general form:

GO TO (N1,N2,...,Nm),i

In this statement N1 , ..., Nm are numbers of other statements in the program, and i is the name of a variable. This **GO TO** statement instructs the computer to go to statement number N1 if the variable i has

the value 1, to statement number N2 if i has the value 2, and so forth. Here is an example:

GO TO (10,20,30),IJK

The variable IJK would have been defined in a prior statement. If its value were 1, 2, or 3, the statement would instruct the computer to go to statement 10, 20, or 30, respectively, for its next instruction. A value of IJK other than 1, 2, or 3 would cause the computer to simply execute the next statement in sequence.[4]

6. Other Fortran Statements

6a. CONTINUE Statement. The CONTINUE statement, as the name implies, means "proceed to the next instruction," and is a dummy statement. Its only purpose is to provide a numbered statement to branch to, from some other part of the program.

6b. COMMENT Statements. A COMMENT statement, unlike all other types of statements, is not an instruction to the computer. Its only purpose in a program is to explain something to anyone who reads the FORTRAN program. Even if no one else will be looking at your program, it is still necessary to use many COMMENT statements so that when you yourself look at the program later you can more easily remember the program's purpose and content. A COMMENT statement is designated by the letter "C" in the first position on a typed line. The remainder of the line may contain any comment as indicated in this example:

C col. 1

C ADD A AND B, AND STORE THE RESULT IN X.

C X = A + B

6c. CALL EXIT and END Statements. The statement CALL EXIT is an instruction to the computer to stop the program since there is nothing else that needs to be done. On some computers the statement used for this purpose is **STOP**. A **STOP** or CALL EXIT could be located anywhere in the program. The **END** statement has a different purpose; it tells the computer that there are no more statements in the program. The **END** statement must, therefore, always be physically the last statement in the program.

[4] In "structured programming" the use of **GO TO** statements is strongly discouraged ("deprecated"). See the Appendix and MISN-0-347.

7. Try These Programs

7a. Program 1: Solution of Quadratic Equation. Here is an example of a program that makes use of all the kinds of statements we have described so far:

```
C  THIS PROGRAM FINDS THE TWO ROOTS OF A QUADRATIC
C  EQUATION
C
      READ(5,100)A,B,C
      DISCR=B**2-4.0*A*C
      IF(DISCR)10,20,20
   10 CALL EXIT
   20 CONTINUE
      ROOT1=(-B+SQRT(DISCR))/2.*A
      ROOT2=(-B-SQRT(DISCR))/2.*A
      WRITE(6,101)ROOT1,ROOT2
      GO TO 10
  100 FORMAT(3F10.5)
  101 FORMAT(2F10.5)
      END
```

This program computes the two roots of a quadratic equation. This program tells the computer to:

1. Input values for the three variables A, B, and C from device number 5 according to a format specified in statement number 100.
2. Compute ROOT1 and ROOT2 which are the two roots of a quadratic equation having A, B, and C as its coefficients.
3. Output numerical values for the two roots onto output device number 6 according to a format specified in statement number 101.
4. Stop since there is nothing else to be done.

The discriminant, $(b^2 - 4ac)$, is computed prior to finding the roots. If the IF statement finds that the value of the discriminant, DISCR, is negative, the program instructs the computer to branch to statement 10 and quit. For non-negative values of the discriminant the program branches to statement 20 and then continues. The use of the CONTINUE statement was somewhat arbitrary in this example. It would have also been possible to omit the CONTINUE and make the statement immediately

following it number 20. Note that the program will only produce output if the discriminant is non-negative. Without the "**IF** statement test" for a non-negative discriminant, the computer might (depending on the input values for A, B, and C, have been asked to take the square root of a negative number. Under ordinary circumstances this would not yield an imaginary result, but would instead be interpreted as an error, and would possibly cause the computer to terminate the program instantly, even if there is more computation to be done.

Enter the program and run it using three sets of quadratic coefficients, one of which would yield a negative discriminant. Check the computer through doing the calculation by hand.

7b. Program 2: Solution of Quadratic Equation. This program is almost the same as the previous one except that there is no IF statement which checks that the discriminant is non-negative.

```
      READ(5,100)A,B,C
      DISCR=B**2-4.0*A*C
      ROOT1=(-B+SQRT(B**2-4.0*A*C))/(2.*A)
      ROOT2=(-B-SQRT(B**2-4.0*A*C))/(2.*A)
      WRITE(6,101)ROOT1,ROOT2
      CALL EXIT
  100 FORMAT(3F10.5)
  101 FORMAT(2F10.5)
      END
```

Run the program using A, B, C coefficients which would result in a negative discriminant and report what the computer does when asked to take the square root of a negative number.

7c. Program to Find the Height You Can Jump. The formula that produces the height you can jump is $y = v^2/2g$, where v is your vertical take-off velocity and g is the acceleration due to gravity. Write a program that reads in values for v and g, computes y and prints the result. Run the program using $v = 10\,\text{ft/sec}$ and use two different values of g: $32\,\text{ft}/s^2$, the acceleration on earth, and $5\,\text{ft}/s^2$, the acceleration on the moon.

Acknowledgments

Preparation of this module was supported in part by the U. S. Coast Guard Academy for a Directed Studies program. It was also supported in

part by the National Science Foundation, Division of Science Education Development and Research, through Grant #SED 74-20088 to Michigan State University.

Appendix

- Typing FORTRAN Statements -

The individual statements in a FORTRAN program are typed on separate lines. There are certain rules that must be obeyed concerning the format of the statements:

1. For numbered FORTRAN statements, the statement number must be typed in positions 1-5 on a line.
2. The FORTRAN statement itself must be between columns 7 and 72 (inclusive). If a statement does not fit on a single line, it may be continued on additional lines which have numbers in column 6 to indicate that they are continuations.
3. All statements must be typed exactly as they appear in the program, including every decimal point and comma. One exception is that extra blank spaces can (almost) always be added to a statement to improve its appearance if this is desired. For example, it is permitted to write X = 1.0 in place of X=1.0 since blanks are ignored in FORTRAN, except when embedded in phrases contained in quotes.
4. The numbers on data lines must have a format that is consistent with that specified in the **FORMAT** statement which will be used to read in the data. As an illustration, consider the data lines to be read by the program in Sect. 4h. According to the **READ** statement, each data line should contain numerical values for the variables A, B, and C. The format of the data should be consistent with the statement

   100 **FORMAT**(3F10.5)

   which specifies:

   a. There are three numbers with decimal points on a line.
   b. Each number is contained within a ten column "field" on the line. (The first number must be within columns 1-10, and the second within columns 11-20).

   c. There are five digits after each decimal point.

The last rule, concerning the placement of the decimal point, is actually necessary only if the decimal point is not explicitly stated on the line. If the decimal point is typed explicitly then it may appear anywhere within the field without regard to what is specified in the **FORMAT** statement. It is always a good idea to include an explicit decimal point in numbers that are read in "F" format, as this eliminates the possibility of the decimal point being inserted improperly by the computer.

LOCAL GUIDE

See this module's *Model Exam*, which shows that you must bring the original of your annotated computer output, not a copy, to the Exam Room when you come to take your exam.

PROBLEM SUPPLEMENT

1. Identify which of the following numbers are unacceptable as real FORTRAN constants:

   $$356 \qquad 3.56 \qquad -53,000 \qquad 10^{11} \qquad .0000562 \qquad -10^{-13}$$

2. Identify which of the following numbers are unacceptable as integer constants:

   $$+324 \qquad -562. \qquad 53,000 \qquad 2|\mathrm{E}| \qquad +20000000000$$

3. Identify which of the following names are unacceptable for integer variables, which are unacceptable for real variables, and which are unacceptable for any variable:

   H, ALPHA, ALPHA 123, I, KLM, KM−12, ATO6SH, 12AH6, CDC162, KAPPA, EPSILON, B1.1, A∗B, XSQUARED, XCUBED!

4. Write FORTRAN expressions corresponding to the following mathematical expressions:

   $$x + y^3 \qquad\qquad a + \frac{b}{c}$$

   $$\left(\frac{a+b}{c+d}\right)^2 + x^2 \qquad\qquad 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!}$$

5. State the value of A or I stored as a result of the following arithmetic assignment statements, and indicate whether the result is integer or real: A = 2∗6 + 1

   A = 2∗6 + 1

   A = 2∗6 + 1

   A = 2∗6 + 1

   A = 2∗6 + 1

6. Write arithmetic assignment statements to:

   a. Add the current value of a variable named ALPHA to the current value of a variable named DELTA and make the sum the new value of a variable named BETA.

   b. Subtract the value of a variable named A from the value of a variable named B, take the square root of the difference and assign it as the new value of W.

7. Write FORTRAN statements to compute values for these formulas:

$$g = \frac{1}{2}\log\left(\frac{1+\sin x}{1-\sin x}\right)$$

$$D = \log|\sec x + \tan x|$$

$$Y = (2\pi)^{1/2}x^{x+1} + e^{-x}$$

Brief Answers:

Have a FORTRAN-wise friend check your answers.

MODEL EXAM

1. See Output Skill K1 on this module's *ID Sheet*.

2. Identify which of the following numbers are unacceptable as real FORTRAN constants:

   ```
   56.004 .039E05 .00200041 -33,564 -1.E-23
   ```

3. Identify which of the following numbers are unacceptable as integer FORTRAN constants:

   ```
   -7729.  -7729 7729.  7729 .000007729
   ```

4. Identify which of the following names are unacceptable for integer variables, which are unacceptable for real variables, and which are unacceptable for any variable:

   ```
   XXX X2223A IJKLX K K22 SQUAREROOT
   SQUARE4 16SQ N1 N1.378 (A/B)*C
   ```

5. Write FORTRAN expressions corresponding to the following mathematical expressions:

$$a^5 + y^3 \qquad x + \frac{y}{z}$$

$$x^2 + \left(\frac{a*b}{c+d}\right)^4 \qquad \left[(1+x)^3 + (1+x)^4\right]^{5/3}$$

6. State the value of A or I stored as a result of the following arithmetic assignment statements, and indicate whether the result is integer or real:

   ```
   A = 2*6 + 1
   I = 2*(10/4)
   A = 1./3.  + 1./3.  + 1./3.
   I = 1./3.  + 1./3.  + 1./3.
   A = (4.0)**(3./2.)
   ```

7. Write arithmetic assignment statements to:

a. Multiply the current value of a variable named ALPHA by the current value of a variable named DELTA and make the product the new value of a variable named BETA.

b. Divide the value of a variable named A by the value of a variable named B, take the square root of the quotient and assign that as the new value of W.

8. Write FORTRAN statements to compute values for these formulas:

$$A = A_0 \sin(\omega t + \delta)$$

$$\phi = \tan^{-1} |(h^2 - b^2)^{1/2}/b|$$

$$Y = (4\pi)^{-1/2} e^{-\lambda x}$$

9. Attach, as part of these Exam Answer Sheets, your hand-annotated output (**not a copy**) for solution of the quadratic equation. Leave space on the Answer Sheet for the grader to mark this item, Item 9. Be sure that the output shows:

a. three cases, one of which yields a negative discriminant;

b. a check of the program against calculator values;

c. the resonse of the square root library program when it is asked to take the square root of a negative number.

10. Attach, as part of these Exam Answer Sheets, your hand-annotated output (**not a copy**) for producing the height you can jump. Leave space on the Answer Sheet for the grader to mark this item, Item 10. Be sure that the output shows:

a. the program that you created;

b. two runs of the program, one using $a = 32\,\text{ft/s}^2$ and one using $a = 5\,\text{ft/s}^2$, and with both using initial $v = 10\,\text{ft/s}$.

Brief Answers:

Have a FORTRAN-wise friend check your answers to questions 2-8.