IBM®

FORTRAN for the IBM 1130

Chapter 4

Programmed Instruction Course

IBM

FORTRAN for the IBM 1130

Chapter 4

Programmed Instruction Course

FOREWORD

The three previous chapters have introduced you to the three basic elements of scientific programming languages - arithmetic, control, and input/output - as they are programmed in the FORTRAN language. The features presented in this chapter give considerable added flexibility to the FORTRAN language through subprograms, which are defined as program segments that are executed under control of another program. These subprograms are usually tailored to perform some oft-repeated set of operations: a subprogram is written only once, but may be used again and again either in a single program set-up or in different programs. In either case, duplication of effort is avoided by eliminating the need for re-writing program segments to perform these common operations.

Also included in this chapter are the Specification statements. Specification statements differ from Arithmetic statements in that they are not executed but merely provide the computer with necessary information about the program.

Copies of this publication can be obtained through IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM DPD Education Development, Education Center, Endicott, New York.

1 You have become familiar with mathematical expressions as used in Arithmetic statements and IF statements. Through the use of the Arithmetic statement we can tell the computer to perform all kinds of basic mathematical operations.

2 Through the use of the basic arithmetic operations we can perform many more complicated functions such as trigonometric functions, logarithms, and roots. All of these types of functions can be computed approximately with simple arithmetic operations of addition, subtraction, multiplication, and division.

3 Since many of these complicated functions can be represented by a series of mathematical operations, it follows that these operations can be programmed for a computer. With the FORTRAN system of subprograms these functions can be made available to any program which might use them.

Q. (True or False) A function such as the trigonometric sine can be programmed for the computer.

●●●

A. True

4 If every computer user wrote a program to compute, for example, the trigonometric sine there would be an enormous amount of duplication of effort, therefore many FORTRAN systems provide a package of previously written programs to compute such common functions for your programs.

Q. The programs that compute these functions are under control of your program, and as such are called _______.

●●●

A. subprograms

5 The package of programs provided for FORTRAN users is called a "library" and the programs contained in the library are called "library functions". The use of these library programs is very simple and will be explained in the next few frames.

Q. FORTRAN provides a set of subprograms in the _______.

●●●

A. library

6 Every subprogram in the library has a particular name associated with it. If you wish to make use of a subprogram in the library all you have to do is use its name in a FORTRAN expression in the same fashion that you would use a variable name.

Q. A subprogram can be brought into use by mentioning its _______.

●●●

A. name

7 It is actually that simple: if you wished to compute the trigonometric sine for the value of X, for example, you might write the statement Y = SIN(X) which would place the sine value in Y. This statement in your program would automatically make use of the sine program in the library.

Q. (True or False) The sine program as used in the example above is a subprogram.

●●●

A. True

8 All library functions are programs which compute some functional relation of a single quantity called an "argument". In the example using the sine function, the argument is the angle whose sine is to be computed; that is, in the statement Y = SIN(X) the variable X is the argument.

Q. (True or False) The statement shown above is an Arithmetic statement.

●●●

A. True

9 All library functions have one or more arguments. The argument is the quantity that the function uses to compute the desired result. In a sense, the argument is an input quantity to the subprogram.

Q. In the statement Z = SIN(THETA) the argument is the variable _______.

●●●

A. THETA

10 When a function subprogram is called upon by mention of its name in an expression, the computer actually stops executing your program while it computes the function value. When the subprogram is finished executing, the computer comes back to your program with the computed result.

Q. In the statement Y = SIN(A) the value whose sine will be computed is the variable _______.

●●●

A. A

11 The function name may be used in an expression exactly like a variable name. When computing the expression, the function's value for the current value of the argument replaces the function name in the expression; this is similar to the way a variable works, with the value replacing the name.

Q. The statement Y = A*SIN(X)+B*SIN(Z) calls upon the sine function (how many?) _______ times.

●●●

A. two

12 The example Y = A*SIN(X)+B*SIN(Z), demonstrates the flexibility of the function notation. More than one function may appear in a single expression and the same function may be used more than once in a single statement. In the above case, the value of the sine of X will be multiplied by A, the value of the sine of Z will be multiplied by B and these values will be added.

Q. The final computed result of the expression shown above will become the value of the variable _______.

●●●

A. Y

13 As mentioned before, every function has a unique name. When the function is used in a statement, its name appears followed by a pair of parentheses containing the argument. The examples used in the preceding frames have shown one of these functions, with its argument, as in SIN(X).

Q. The argument in the expression SQRT(ARG) is the variable ______.

●●●

A. ARG

14 The argument may be a variable, constant, or any legal expression. Thus, SIN(X), SIN(3.141592), or SIN(A**2+B**2) are legal examples of arguments. In any case, when a function name appears in a statement the value of the argument is determined first, then the function is executed to obtain the result.

Q. The argument in the expression SIN(X**2) is ______.

●●●

A. X**2

15 As a matter of fact, the argument of a function can even be another function itself: SIN(SQRT(X)) is an example of nested functions. In cases like this, the innermost function (the one that is also an argument) is evaluated first to obtain a value for use of the outer function as an argument.

Q. In the example SIN(SQRT(X)) the function named ______ is an argument of another function.

●●●

A. SQRT

16 **The standard library distributed with the 1130 system contains many functions for mathematical computation. Some of these appear on Panel 4.1 with explanations. Turn to Panel 4.1.**

17 As you can see, all the library functions have pre-assigned names. Because these names refer to specific library routines, you must avoid using these same names when you create your own variable names. In fact, a good rule of thumb for beginners is to avoid using all FORTRAN defined words like DO, FORMAT, READ, etc., when creating variable names. If you create completely new names there is no chance that either the computer or someone looking at your program will misinterpret what you mean.

18 Two main points have been covered so far: 1. a set of commonly-used subprograms in function form is available to your programs in a library: 2. these function subprograms are brought into use by the mention of the appropriate name in an ordinary FORTRAN expression, using the name like a variable.

Q. (True or False) A function name must be accompanied by the correct number of arguments.

●●●

A. True

19 The library functions illustrated on panel 4.1 are written to use real arguments and yield real values for their result. This means that they are intended to be used in real expressions and must have real quantities for their argument.

Q. (True or False) The expression <u>SIN(I)</u> is legal.

●●●

A. False (the argument of the SIN function program must be real)

20 Since the function's name is a part of an ordinary FORTRAN expression and its value is used directly in the computation indicated in the expression, it follows that the mode should normally agree with the entire expression.

Q. (True or False) The statement <u>Y = I*SIN(X-Y)</u> does not mix modes.

●●●

A. False (the variable <u>I</u> is an integer quantity but SIN is a real function)

21 In general, then, library function names can be used in any way which is legal for ordinary variables. In fact, their meaning is very similar to that of a variable name: at the time the expression in which a function name is contained is executed, the current value for that function is computed and supplied to the expression to be used as indicated, just as is the value of a variable.

Q. Write a statement to compute the product of three quantities: 2.0 times the sine of X times the cosine of X, placing the result in the variable ANSWR. _______.

●●●

A. ANSWR = 2.0*SIN(X)*COS(X)

22 If your answer agrees with the one shown above, skip to frame 27; if your answer does not agree, go on to the next frame.

23 The problem requires an Arithmetic statement with two function names in the expression. The expression was to have computed the product of three terms: 2.0, sin X, and cos X. The use of the functions' names with their given arguments in an ordinary expression is sufficient to solve this problem.

Q. (True or False) When an expression containing a function is computed, the computed value of the function replaces the function name in the expression.

●●●

A. True

24 The expression given in the preceding problem required the computation of sin X and cos X. The functions in the library to compute these quantities are SIN and COS; in this problem they both had the argument X. Therefore, the complete expression to solve for the product has to be 2.0*SIN(X)*COS(X).

Q. Given that the sin(0.) = 0. and that cos(0.) = 1.0, the value of the expression 2.0*SIN(X)*COS(X) would be _______ when X has a value of 0.

●●●

A. 0. (2.0 times 0. times 1.)

25 Q. Try a similar problem requiring function notation: given that the tan X = $\frac{\text{sin X}}{\text{cos X}}$ write a statement to place the <u>tan THETA</u> in the variable TRIG. ________

●●●

A. TRIG = SIN(THETA)/COS(THETA)

26 If your answer agrees with the one shown above, continue to the next frame. If your answer does not agree, go back to frame 6 and review.

27 Another example of function notation using library function names might be: <u>Y = SIN(SQRT (ALOG(X(I)**2)))</u>

This is a three-deep nesting of functions, where the log function value is the argument for the square root function whose value is, in turn, the argument for the sine function.

Q. The first computation executed by the computer in the above example would be _______.

●●●

A. X(I)**2

28 That example, <u>Y = SIN(SQRT(ALOG(X(I)**2)))</u>, further illustrates the fact that an argument can be any legal FORTRAN expression. The computer treats "nested" functions by starting with the innermost point and computing its way out. Thus, <u>X(I)**2</u> is computed first; its value is given to the log function whose result is the argument to the square root function; the final result becomes the value of Y.

Q. The <u>entire</u> argument of the sine function shown above is (written out) _______.

●●●

A. **(SQRT(ALOG(X(I)**2))), which is still a single quantity**

29 Function notation is not limited to expressions appearing in Arithmetic statements either. For example, IF(SIN(OMEGA*T))10,20,30 will tell the computer to compute the value of OMEGA*T, give that result to the sine function, and, finally, test the final result for negative, zero, or positive status.

30 Perform Exercise 4.1 in your problem book.

Library functions represent only one type of function which may be used in FORTRAN programs. The four kinds of function subprograms are listed below. Their individual merits will soon become clear.

1. Library or built-in functions
2. Arithmetic statement functions **(also called statement functions)**
3. FORTRAN-written FUNCTION subprograms
4. SUBROUTINE subprograms

31 The library functions as described in the preceding frames are a set of previously written programs available to your program at execution time, being called into action wherever their particular name appears in your program.

Q. Write a statement, using the appropriate function, to compute the square root of the sum of A-squared and B-squared; placing the in HYPTN. _______.

●●●

A. HYPTN = SQRT(A**2+B**2)

32 The question of the preceding frame demonstrates a typical use of a library function. Note that the argument is itself an expression, which is perfectly permissible. Incidentally, the use of SQRT for computing square roots is more efficient than using the notation **0.5

Q. (True or False) In the answer shown above, the value of the expression A**2+B**2 is computed before the function is called into action to compute the square root.

●●●

A. True

33 If you use the same built-in function more than once in a program, the actual subprogram segment appears only once. This type of subprogram is called a "closed subprogram".

Q. The _______ functions are examples of closed subprograms.

●●●

A. built-in or library

34 Actually, the other three types of functions are all of the closed subprogram variety. This means that the subprogram segment appears only <u>once</u> regardless of how many times it is used in a given program. The advantage of this approach is that core storage is conserved.

●●●

A. open

35 All FORTRAN subprograms are named according to the same convention: the function name may have from 1 to 5 alphanumeric characters, the first of which must be alphabetic. (No special characters may be used.)

Q. (True or False) The name <u>ARF</u> is a legal function name according to the above definition.

●●●

A. True

36 The naming rules in the above frame should sound familiar to you. Actually, these are the same rules used for constructing a variable name. Of course, the library functions are already named, but, even so, they are subject to the same set of rules.

37 The following frame contains a list of function names. Identify the legal ones with a plus sign and the illegal ones with a minus sign. Remember, they follow the same rules that govern the naming of a variable.

38 Q. Sample function names: (identify legal(+) and illegal (-) ones)

SIN	______	FIRST	______
OFF/	______	R2345	______
FOFXF	______	A*MQ?	______
FLOAT	______	FUNCTION	______

●●●

A.

SIN	+	
OFF/	-	(contains special characters)
FOFXF	+	
FLOAT	+	
FIRSTF	+	
R23456	+	
A*MQ?	-	(contains special characters)
FUNCTION	-	**(greater than five characters)**

39 Let's take a moment to review the last few frames. You have been told that there are four kinds of FORTRAN subprograms, but so far only the library or built-in functions have been discussed. You have also learned the rules for naming subprograms, which very conveniently are the same rules that apply to naming variables.

Q. (True or False) FORTRAN subprograms are all governed by the same naming rules.

●●●

A. True

40 The question of mode comes up with functions as well as with variables and expressions. In the case of function subprograms, mode conventions have to be observed on two counts: the function value itself and the arguments. The mode of the function value is denoted by the function name, as defined in the next frame.

41 The computed value of the function has a particular mode associated with it, since it is used in an expression with other quantities. Just as with variable names, the first letter of a function name indicates its mode. Thus, function names that begin with the letters I through N indicate integer mode, and all other letters indicate real mode. This mode-defining convention is known as the implicit way of specifying mode.

Q. The function name ALOG (library function) indicates a _______ mode value for the function.

●●●

A. real

42 The argument or arguments of a function may be of either mode; that is, the name of the function itself does not imply that an argument is of a particular mode. When a function itself is programmed, the mode of any arguments is defined, and it is up to the user of a function to see that the argument supplied is the correct one.

Q. (True or False) It is possible to have a real function which uses an integer argument.

●●●

A. True

43 It should be pointed out that normally the mode of a function used in an expression matches the mode of the expression. The mode required for the arguments, however, does not have any bearing on the mode of the expression in which the function appears.

Q. According to the above definitions, the statement Y = A+B+FLOAT(I) is (legal or illegal) _______.

●●●

A. legal

44 The following is a list of points that you should be thoroughly familiar with before continuing:

1. A function name appearing in a FORTRAN expression results in the execution of a program.
2. This program has been previously written.
3. Each of these subprogram types is made available through a library.
4. A function is recognized by its name.
5. A function name has 1 to 5 characters, the first of which must be alphabetic.
6. The mode of the function is determined implicitly by the first letter of its name.
7. The modes of function arguments need not be the same as the function itself.

45 So far, you have learned about the implicit method of specifying modes for variables and functions. Mode in this method depends upon the first letter of the name, and is a predetermined convention of FORTRAN.

Q. The name QUICK implies the _______ mode.

●●●

A. real

46 Often, this implicit convention prevents us from using names which start with the wrong mode letters when writing expressions. For instance, if we wanted the name MASS to be real, we would have to change it to XMASS or AMASS or some other acceptable first-letter variation.

47 To solve this problem, **1130 FORTRAN** provides us with a Type statement which allows the programmer to specify the modes of specific names regardless of their first letter. The Type statement is a specification statement like the FORMAT statement, and, as such provides information to the program but is not executed.

Q. If the variable name ALPHA were to be defined as an integer, a _______ statement would be needed.

●●●

A. Type

48 There are two Type statements which affect mode: INTEGER and REAL. The form of the Type statement is the word INTEGER or REAL followed by a list of function or variable names separated by commas. All names in the list will assume the mode of the Type statement because the implicit mode of their first letters will be overridden. For example, the statement INTEGER PEARS, APPLE, X, Y123 would assign the integer mode to PEARS, APPLE, X, and Y123.

Q. Write a Type statement to assign the real mode to the variables MINOR and MAJOR.

●●●

A. REAL MINOR, MAJOR

49 Q. Write the Type statements to assign the integer mode to the variables EENY, MEENY, ZAP, and the real mode to NUTS, PECAN, GRAPE.

●●●

A. INTEGER EENY, MEENY, ZAP; REAL NUTS, PECAN, GRAPE; (Note that it is permissible to reassign the same mode to a variable which is already defined implicitly, such as GRAPE)

50 Because we are explicitly assigning mode through the Type statement, the form of mode assignment is known, quite naturally, as <u>explicit</u> mode specification. Type statements must precede all executable statements in the source program and must also precede <u>all other</u> specification statements.

51 To put the physical ordering of the source program in perspective, here's the way it might look at this point:

```
Type statements
FORMAT statements
Executable program steps
END
```

52 Perform Exercise 4.2 in your problem book.

The built-in library functions are automatically made available to any program which makes use of the appropriate name. The following frames will describe some more library functions available with **1130 FORTRAN.** These functions are in addition to those illustrated on Panel 4.1. Incidentally, these do not need to be committed to memory since they are written up in reference manuals, but a familiarity and working knowledge of these built-in functions will be helpful in their use.

53 One of the simplest of the built-in functions provides <u>absolute value</u> of a quantity. This means that the value of the function will be the positive magnitude of the argument, regardless of the original sign of the argument quantity. The function is named ABS (or IABS for integer use).

Q. By the above definition, the value of the expression <u>ABS(X)</u> will always be (sign) _______.

●●●

A. positive (plus)

54 Important note: this function, like every other function, does nothing to the argument itself. For example, the statement <u>Y = ABS(A)</u> uses the <u>value</u> of A to determine the value of the function, but the value of A remains intact in its original form.

Q. If A in the above example has a value of <u>-1.234</u>, the value of Y after execution would be _______.

●●●

A. +1.234

55 Thus, you can see that the only operation performed by the ABS or IABS functions is to set the sign of the argument value (but not the argument itself) to positive. This can be a valuable function, however simple it seems, as in the example <u>SQRT(ABS(X))</u> which assures that the argument to the square root function is a positive quantity (which it indeed must be to avoid an error message).

Q. If you needed the absolute value of the variable **NUMBR** you would write the expression: _______.

●●●

A. **IABS(NUMBR) (the IABS function must be used, since it is designed to handle integer arguments)**

56 This last question brings up an important point: these built-in functions are designed to handle arguments of a particular mode. The mode of the argument does not necessarily agree with the mode of the function, but in the case of IABS it does happen to agree, and, in fact, must be written that way.

Q. (True or False) The statement Y = ABS(NUMBR) is legal.

●●●

A. False (ABS is the name used with real arguments only)

57 Perform Exercise 4.3 in your problem book.

A built-in function is available to change the mode of a single argument; the value of this function is then the argument's value in the opposite mode. The form FLOAT (argument) is used to convert from integer to real; the form IFIX (argument) is used for the opposite conversion.

Q. In which mode do you think the argument of IFIX should be? _______.

●●●

A. real (this function converts from real to integer)

58 As the last question implies, the FLOAT and IFIX forms must have arguments of opposite mode to the name of the function: thus, FLOAT(N) and IFIX(A) would be legal uses of these functions, but the form FLOAT(A) or IFIX(K) would yield meaningless numbers.

Q. (Yes or No) Is there any difference in the result of the statements Y = K and Y = FLOAT(K)? _______

●●●

A. No (either form will produce the real value of K for the variable Y)

59 The IFIX form can hold some surprises in store for you if you forget that the integer mode contains only integers and the conversion from real involves dropping (not rounding) the fraction part. Thus, IFIX (1.999999) has a value of 1 (integer, truncated).

Q. If X has a value of 3.0, the value of Y after executing Y = FLOAT(IFIX(X/2.0)) would be _______.

●●●

A. 1.0 (X/2.0 equals 1.5; when this is truncated the value becomes 1).

60 A built-in function is available to transfer the algebraic sign from one quantity to another. The function SIGN (argument 1, argument 2) will have as its value the absolute magnitude of the first argument and the algebraic sign of the second argument. (Both arguments are real.)

61 This function has its counterpart in the integer form as ISIGN (arg1,arg2) which performs the same function for integer quantities. Thus, for example, SIGN(-1.5,2.3) would equal +1.5 after execution, while SIGN(-1.,-5.) would have the value -1.

Q. If L and M both have the value -10, the expression ISIGN(L,M) will equal _______.

●●●

A. -10

62 **Panel 4.2 presents summaries of the built-in function types with information concerning number and mode of arguments, mode of function value, etc. Turn to Panel 4.2. This panel, together with Panel 4.1, lists all library functions available with the 1130 version of FORTRAN.**

63 The library subprograms discussed so far in this chapter are of the previously-written variety; that is, they already exist as you read this text, with their own special names and function programs.

64 To repeat, the built-in functions are previously-written programs of general interest. The next type of function, the Arithmetic statement function, is used where no library function is available. This type of function is defined by the programmer right in the program in which it is to be used.

65 Any functional relationship which can be expressed in a <u>single expression</u> can be applied to an Arithmetic statement function. To show a trivial example, suppose you desired a function which would sum up its three arguments:

<u>SUM(A,B,C) = A+B+C</u>

Q. The three arguments in the example above are _______, _______, and _______.

●●●

A. A,B,C

66 Every Arithmetic statement function is defined by a single statement which consists of the selected name with its arguments followed by an equals sign and followed in turn by the FORTRAN expression which defines the desired functional relationship. The example <u>SUM(A,B,C) = A+B+C</u> is such a statement.

Q. The name of the function defined by <u>SUM(A,B,C) = A+B+C</u> is _______.

●●●

A. SUM

67 The single statement which defines the Arithmetic statement function's relationships is called a "function defining statement". When such a statement appears in a program, the indicated operations will be performed wherever the function's <u>name</u> appears in an expression elsewhere in the program.

Q. (True or False) The statement <u>SUM(A,B,C) = A+B+C</u> is a function defining statement.

●●●

A. True

68 In other words, if you write the statement SUM(A,B,C) = A+B+C, a function defining statement, in your program, you can use this function simply by writing its name in an expression elsewhere in the program, as, for example, Y = SUM(X,Y,Z)/SUM(P,Q,R). X, Y, and Z (and P, Q, and R) will be operated on in the manner prescribed by the arguments in the function defining statement.

Q. If X, Y, and Z have the values 1., 3., and 2. respectively, the value of SUM(X,Y,Z) is _______.

●●●

A. 6.0 (or 6.)

69 The function defining statement contains argument variables both with the function name and also in the expression on the right of the equals sign. When that function name then appears in an expression elsewhere in the program, whatever quantities are given there as arguments will be used in the manner prescribed by the function defining statement.

Q. The expression FIRST(A,B) will provide the values of _______ and _______ to the subprogram called FIRST.

●●●

A. A, B

70 The variables listed as arguments in the function defining statement, then, are dummy arguments, meaning that they are not variables in the usual sense but rather are used to show what is to be done with the values of the actual arguments as indicated in the function notation elsewhere in the program.

Q. Given the statement SUM(A,B,C) = A+B+C, the expression SUM(2.,2.,2.) has a value of _______.

●●●

A. 6.

71 Any functional relationship which can be expressed in a single expression can be expressed in the Arithmetic statement function form. The expression on the right of the function defining statement may contain variables (unsubscripted), constants, and even other function names.

Q. (True or False) The statement <u>CUTOF(X) = A(I)**2+X(I)**2</u> is a legal function defining statement.

●●●

A. False (subscripted variables are not permitted in the expression of a function defining statement)

72 The function defining statement's expression may make use of ordinary variables in addition to the dummy argument variables listed on the left. When this is the case, these ordinary variables will contribute their current values to the computation of the function, while, of course, the argument values will be supplied by the statement which called on the function.

Q. In the statement <u>SOMEF(X) = A*X**2</u> the expression contains an argument (dummy) variable _______ and an ordinary variable _______.

●●●

A. X, A

73 The function-defining statement <u>SOMEF(X) = A*X**2</u> is an expression containing an argument variable and an ordinary variable. A statement elsewhere in the program such as <u>Y = SOMEF(Z)</u> will cause the <u>SOMEF</u> function to be computed using the current value of A; and the value of Z is used where X appears in the definition.

Q. Every statement containing the name SOMEF will call upon the function defined above, specifying an argument which will be treated like the variable _______ in the definition.

●●●

A. X

74 All function defining statements must come before all executable statements in your program. You see, this statement is not executed in the same sense that an ordinary Arithmetic statement is; it appears only once in the program but its indicated operations are carried out each time its name appears in an expression, and control remains with the statement which called upon the function.

75 The function defining statements are placed physically before any executable statements in your source program. Here's how the physical ordering of the source program might look at this point:

Type statements
FORMAT statements **(may be placed elsewhere)**
Function defining statements
Executable program steps
END

76 Since the Arithmetic statement function is written by a programmer for his special application, it follows that he must also assign a name to that function. The name must conform to the usual function-naming convention: 1 to 5 alphanumeric characters, first alphabetic, with the mode determined implicitly by the first letter of the name, unless a Type statement overrides the implicit convention.

Q. (True or False) The names NYOLD, SCOFF, XACT, and ONOFF are all legal real function names.

●●●

A. True

77 Let's look at a practical example. Suppose you need to compute the expression $\sqrt{x^2 + y^2}$ several times with different values for the parameters. Rather than repeat this expression in each statement that needs it, you can define a function called, say, POLAR(X,Y) which can be used by name in each expression that requires it. A suitable function defining statement would be POLAR(X,Y) = SQRT(X**2+Y**2).

Q. Using the above definition, the value of POLAR(3.,4.), would be ________.

●●●

A. 5.

78 Q. Write a function defining statement using the name ROOT(A,B,C) that will solve the expression

$\frac{-b+\sqrt{b^2-4ac}}{2a}$: _______

●●●

A. ROOT(A,B,C) = (-B+SQRT(B**2-4.*A*C))/(2.*A)

79 If your answer agrees with the one shown, skip to frame 101. If you did not get the correct answer, continue to the next frame.

80 Given the relation $\frac{-b+\sqrt{b^2 - 4ac}}{2a}$ and the potential name of the function, ROOT, the writing of the function defining statement should be a simple matter of filling in the blanks: name, arguments, equals sign, and expression (the library SQRT is permitted in the expression of a function defining statement).

Q. Write a function defining statement for DISCR(A,B,C) to compute the square root in the above expression: _______

●●●

A. DISCR(A,B,C) = SQRT(B**2-4.*A*C)

81 The basic idea of these Arithmetic statement functions is to program a commonly used set of operations in a form that can be used with function notation throughout the program. For example, a statement such as IF(DISCR(X,Y,Z))10,20,30 used in a program containing the above definition will carry out those indicated operations without re-writing them, as many times as you wish to use it.

Q. (True or False) Once an arithmetic statement function is defined by a function defining statement, it may be used repeatedly throughout a program by referencing its name.

●●●

A. True

82 Q. Try another case. Write a function defining statement for a function called TRIG which will compute the square root of the sine of the first argument and multiply this by the second argument; use THETA and PHI for the argument variables.

●●●

A. TRIG(THETA,PHI) = SQRT(SIN(THETA))*PHI

83 If your answer agrees with the one shown, continue to the next frame. If your answer is wrong, better go back to frame 65 and read carefully the material on Arithmetic statement functions.

84 Perform Exercise 4.4 in your problem book.

You have now become familiar with two of the four types of functions used in FORTRAN programs: the library or built-in, and Arithmetic statement functions. The first type is available automatically and consists of a specific set of pre-written programs. The second type is written for a particular program's use by the programmer himself.

Q. Since the name SOMEF does not belong to the set of library or built-in functions, it must be an _______ _______ function.

●●●

A. Arithmetic statement

85 Most of the function usage can be satisfied by these two function types. Occasionally, however, a function is needed that is not in the library or built-in packages and also cannot be expressed in a single expression, as required for the Arithmetic statement functions. For these cases, FORTRAN-written subprograms are used.

Q. (True or False) Arithmetic statement functions can only be used where the function's value can be defined in a single function defining statement.

●●●

A. True

86 The FORTRAN-written subprogram fulfills the same purpose that the Arithmetic statement function does, except that this type of function permits as many statements as necessary to fully define the functional relationship for which the function is being written.

Q. (True or False) The third type of function, the FORTRAN-written subprogram, permits only one statement in the function definition.

●●●

A. False (as many statements as are needed are permitted)

87 The FORTRAN-written subprograms are of two types: the FUNCTION subprogram and the SUBROUTINE subprogram. To provide a mental picture of all four subprograms in the overall subprogram set-up, a diagram is shown below:

FORTRAN SUBPROGRAMS (4 TYPES)

$$\underbrace{\begin{array}{ll}\text{Built-In} & \text{Arithmetic}\\ \text{or Library} & \text{Statement}\end{array}}_{\begin{array}{l}\text{Only these two have}\\ \text{been discussed so far.}\end{array}} \quad \begin{array}{ll}\text{FUNCTION} & \text{SUBROUTINE}\\ \text{Subprograms} & \text{Subprograms}\end{array}$$

Q. The two types of subprograms not yet covered in this text are _______ and _______.

●●●

A. FUNCTION subprograms, SUBROUTINE subprograms.

88 The FUNCTION subprogram (the capital letters are used intentionally to distinguish this function type) is itself a complete program; that is, it is written separately from any program which uses it and even has its own END statement, which is used for all programs to signify the last statement in the program.

Q. When a FUNCTION subprogram is written with another program, at least _______ END statements are needed.

●●●

A. two (one for each FUNCTION subprogram and one for the other program)

89 The FUNCTION subprogram is a completely separate segment of programming from the program which makes use of the function. This type of subprogramming enables you to write a complete set of statements which will be executed only if another program makes reference to the chosen name of the FUNCTION.

Q. (True or False) The FUNCTION is essentially the same in use as the first two types of functions discussed.

●●●

A. True

90 The FUNCTION subprogram, then, is used like a library function; that is, it is a previously-written program which is executed wherever its name appears in another program. In other words, if you find you need a function and it is not available in the library, you can write it yourself with FORTRAN statements in FUNCTION form.

Q. (True or False) The FUNCTION subprogram is written completely in FORTRAN language.

●●●

A. True

91 The FUNCTION subprogram is distinguished from an ordinary program by the very first statement in the program. A special statement is used, beginning with the word FUNCTION followed by its name, an open parenthesis, a list of "dummy" arguments, and a close parenthesis: e.g. FUNCTION NAME(ARG1,ARG2,ARG3).

Q. The FUNCTION statement shown above must be the ______ statement in the FUNCTION subprogram.

●●●

A. first

92 Following the FUNCTION statement you may write any combination of statements to properly define the functional relationship between the argument quantities and the single result which becomes the value of the function.

Q. A program beginning with the statement <u>FUNCTION THING (A,B,C)</u> is a _______ subprogram.

●●●

A. FUNCTION

93 In addition to identifying a program as a FUNCTION, the FUNCTION statement itself fulfills two purposes: it defines the name of the FUNCTION and also indicates, through the list of dummy argument variables, the quantities whose values are to be supplied by the program which uses the FUNCTION.

Q. The FUNCTION statement defines the _______ of the FUNCTION.

●●●

A. name

94 Functions of this third type have the same naming rules as the other two function types: The name can have from 1 to 6 alphanumeric characters, the first of which must be alphabetic, and no special characters may be used. In other words, the third function type also has the same naming rules as ordinary variables.

Q. (True or False) The name <u>A</u> would be a legal FUNCTION name.

●●●

A. True

95 You will remember that the mode of the library functions is pre-defined while the mode of Arithmetic statement functions unless overridden by a Type statement is determined implicitly by the name. The mode of FUNCTION subprograms is determined just like the Arithmetic statement functions.

Q. The FUNCTION subprogram name JUNK belongs to the _______ mode.

●●●

A. integer

96 In addition to the two ways of declaring the type of a FUNCTION subprogram name (i.e. implicitly or explicitly), there is also the option of explicitly assigning mode by preceding the word FUNCTION with one of the following words: INTEGER or REAL. For example, REAL FUNCTION NUMBR (X,Y,Z) would define the function named NUMBR as a real FUNCTION.

Q. Identify the mode of the following FUNCTIONS:

Name	Mode
QUICK(X,Y)	________
IRATE(YES,SO)	________
INTEGER FUNCTION ALPHA (A,B)	________
LIMIT(Z)	________
REAL FUNCTION IBEX(Q)	________

●●●

A. real, integer, integer, integer, real

97 So far, then, you have seen that you can write your own FUNCTION subprograms to be used by other programs of yours in exactly the same manner that you use the library and other functions. Unlike the Arithmetic statement functions, however, the FUNCTION is a separate program segment headed by the FUNCTION statement, followed by statements defining the function, and terminated by its own END statement.

98 The FUNCTION statement which heads the FUNCTION subprogram contains the name you have chosen for the FUNCTION and also contains the list of dummy variables whose values are to be supplied to the subprogram by the program which uses the FUNCTION. The name is chosen by the same rules applied to variables.

Q. (True or False) The statement FUNCTION ABCDEFG(X,Y,Z) is valid.

●●●

A. False (the FUNCTION name has too many characters)

99 All FUNCTION subprograms use a special statement to tell the computer to return to the program which called on the FUNCTION and pick up where it left off. When the FUNCTION subprogram has finished its task of computing its particular value, the RETURN statement signals the computer to go back to the other program (consists of the word RETURN).

Q. The RETURN statement sends the control back to the program which used the _______ of the FUNCTION.

●●●

A. name

100 This brings the total of new statements in this chapter to two: the FUNCTION statement and the RETURN statement. The RETURN statement is used in a subprogram where you might use a STOP in a regular program; that is, as the last statement to be executed (remember, the END statement is actually last).

Q. The three statement types that must be included in every FUNCTION subprogram are FUNCTION, _______, and END.

●●●

A. RETURN

101 One other statement is required of every FUNCTION subprogram: an Arithmetic statement which sets the name of the FUNCTION equal to the final computed result, treating the name as an ordinary variable (using only the name, without any parentheses or arguments). This may be placed anywhere in the program, although it often comes just before the RETURN statement, and it links the result with the other program. Examples of this will be shown in future frames.

102 Now that the general framework has been defined, we can study an example of a FUNCTION program and how it is used by another program. The program on Panel 4.3 will provide as its value the sum of the array specified as its first argument, considering the array to be of length defined by the second argument. Turn to Panel 4.3 now.

103 Because this subprogram involves an array, it is now necessary to discuss the DIMENSION statement which is the second statement of the subprogram. The DIMENSION statement is a specification statement like the FORMAT and Type statements. As such, it is not executed, but merely provides the computer with information.

104 The DIMENSION statement is used for specifying the size of arrays used in the program. The statement consists of the word DIMENSION followed by any number of array-size specifications separated by commas. Note: learn well the spelling of the word DIMENSION, as the FORTRAN language accepts only the correct form!

Q. A DIMENSION statement must be used in all programs which contain _______.

●●●

A. arrays

105 The array-size specifications consist of the name of the array followed by a pair of parentheses containing a <u>constant</u> which defines the size of the array: for example, an array of 100 numbers called A would have an array-size specification of <u>A(100)</u>.

Q. The DIMENSION statement to specify the array described above would be _______.

●●●

A. DIMENSION A(100)

106 If your program contains more than one array, you may specify all the arrays in a single DIMENSION statement or use more than one DIMENSION statement if you so choose. For example, a program containing three arrays called A, B, and C, each of which is 1000 numbers long, must have the statement <u>DIMENSION A(1000), B(1000), C(1000)</u> included in the program.

Q. (True or False) The sequence of statements <u>DIMENSION A(1000)</u>; <u>DIMENSION B(1000)</u> ; <u>DIMENSION C(1000)</u> would be equivalent to the statement shown in the example above.

●●●

A. True

107 Every variable which is to be used with a subscript must appear in a DIMENSION statement. If this rule is not followed, the subscripted variable will look exactly like a FUNCTION name with argument. In fact, it will be treated as exactly that if its name is not listed in a DIMENSION.

Q. (True or False) The statement DIMENSION A(10)B(20)C(30)D(40) is legitimate.

●●●

A. False (commas must be used to separate the individual array-size specifications)

108 Sometimes the size needed for an array will vary from run to run. In cases like these, an array size should be chosen that is large enough for the maximum size of array you will need. In no case should a subscript be allowed to exceed the size specified for the array in the DIMENSION statement.

Q. The statement DIMENSION ARRAY(100), BLOCK(500) specifies array sizes of _______ and _______.

●●●

A. 100,500

109 When a dummy argument in a FUNCTION or SUBROUTINE is to be treated as an array (that is, it will be used with subscripts), the subprogram must also have a DIMENSION statement (in addition to the DIMENSION statement that must appear in the program that calls upon the subprogram).

Q. (True or False) Any program that contains a subscripted variable must list that variable in a DIMENSION statement.

●●●

A. True

110 Repeat: a FUNCTION that has a dummy argument which is an array must have a DIMENSION statement containing that name (even if the specified array size is different from that of the calling program) to permit subscripted use of the variable. Without this specification, the dummy array will be thought of as a FUNCTION by the subprogram, which would obviously be incorrect.

Q. If a subprogram FUNCTION SUM(A,N) refers to a 10-number array A, the subprogram must contain the statement _______.

●●●

A. DIMENSION A(10)

111 Basically, then, the DIMENSION statement is used to give the computer some bookkeeping information. All variables that are used with subscripts must appear in a DIMENSION statement in the same program or subprogram; otherwise, the computer looks for a non-existent FUNCTION subprogram and comes to a halt.

Q. The statement DIMENSION X(15),Y(100000),Z(50) specifies array sizes for the variables _______, _______, and _______.

●●●

A. X,Y,Z

112 An important note: the DIMENSION statement must appear before any statements using the array names specified therein. In the case of FUNCTION subprograms, the DIMENSION statement(s) follows the subprogram heading statement which contains the word FUNCTION. In main programs, the DIMENSION statement(s) is placed before any executable statements, and immediately follows the Type statements. For example,

```
Subprogram:                  Main Program:

FUNCTION GAMMA (A,B,C)       Type statements
DIMENSION                    DIMENSION
_______                      FORMAT
_______                      Function defining statements (arith)
_______                      _______
_______                      _______
RETURN                       _______
END                          END
```

113 Now that you understand something about DIMENSION statements, look again at Panel 4.3. The first statement in the subprogram is, of course, the FUNCTION statement, defining the chosen name and the dummy arguments. The second statement is the DIMENSION statement, which defines the A array as being 100 numbers in size. The actual statement is defined in the next three statements: initialize the SUM and loop to keep adding successive numbers in the array into SUM (notice that the FUNCTION name, SUM, is treated like a variable). When the loop is completed, the RETURN will take the computer back to whatever program has used the FUNCTION.

Q. (True or False) The sample program would not be complete without the END statement.

●●●

A. True (all programs, even subprograms, require an END statement)

114 This example demonstrates another feature of FUNCTION subprograms that is not available with the other types; the argument quantities may be <u>arrays</u> (the argument, for example, of the SIN library routine must be a <u>single</u> quantity). You must be careful to treat the argument quantities accordingly, however!

Q. In this example, the program headed by <u>FUNCTION SUM(A,N)</u> treated the argument named _______ as an array.

●●●

A. A

115 It is important to understand that the arguments as specified in the FUNCTION statement are <u>dummy</u> variables. This means that their names are used in the subprogram without definition of value, and the value used for any particular execution will be supplied by the statement in the other program. For instance, in the example the variable A was one of the dummy variables, and anywhere the name A appeared in the FUNCTION the value of the <u>first</u> argument in the other program would be used, even if its <u>name</u> were not A.

116 Turn to Panel 4.4. The panel shows two programs: the one on the left is the same FUNCTION SUM(A,N) described before. The one on the right is part of another program which uses SUM. Notice that the program on the right uses the arguments BLOCK and KOUNT. This means that BLOCK and KOUNT will be used by the FUNCTION wherever the names A and N appear; A and N are just dummies (notice that the respective modes agree!).

Q. (True or False) Both programs on the panel must be terminated by END statements as shown.

●●●

A. True

117 Suppose the program on the right has another statement such as Z = SUM(VECTR,K). This simply means that the FUNCTION called SUM would be executed again, this time using the values of VECTR and K where the dummy names A and N appear in the FUNCTION.

Q. The statement in the program which uses the FUNCTION name shown above must contain exactly _______ arguments.

●●●

A. two

118 When you write a FUNCTION subprogram and use it in various other programs with statements containing its name, make sure that you specify the same number of arguments in both places (the FUNCTION statement and the expression in the other program). Also, make certain that the mode agrees with what is expected.

Q. (True or False) The statement Y = SUM(X,Y) would be legal for the FUNCTION just described.

●●●

A. False (the second argument must be integer)

119 Since subprograms of the FUNCTION type are written independently of the program that makes use of them, there is no correlation of variable names and statement numbers between the programs. The only way a FUNCTION can make use of a quantity in another program is through the dummy arguments.

Q. The statement Y = SUM(X,N) provides the subprogram SUM with the values of _______ and _______.

●●●

A. X, N

120 Thus, a program and an associated subprogram can both use the same variable names and statement numbers without any conflict. The only time a variable in a subprogram has a value related to another program is when that subprogram variable is a dummy argument.

Q. A program headed by the statement FUNCTION SQUAR(X,Y) utilizes a total of _______ quantities from the program which uses the FUNCTION.

●●●

A. two

121 Here are some check points in the writing and using of FORTRAN subprograms:

1. Choose or define a name that is of the mode you desire. Use the same naming rules as for variables.

2. Make certain that all the information needed in the FUNCTION is included in the list of arguments. The arguments listed in the other program must agree in number and mode with the FUNCTION.

3. Be sure that the FUNCTION has the proper FUNCTION, RETURN, and END statements.

4. Include a statement in the FUNCTION that sets its name equal to the desired result.

5. If the FUNCTION deals with arrays, include an appropriate DIMENSION statement.

122 Q. With the FUNCTION SUM(A,N) as described in Panel 4.4, the statement Y = SUM(X,3)/SUM(Z,4) would result in a value of _______ for Y if the X array contained the numbers 4., 3., and 5., and the Z array contained the values 1., 2., 0., and 1.

●●●

A. 3.

123 If your answer agrees with the one given, skip to frame 145. If you did not arrive at the correct answer, go to the next frame.

124 The FUNCTION SUM(A,N) performed the task of adding the N quantities in the A array, giving the sum as the value of the FUNCTION. Therefore, the statement Y = SUM(X,3)/SUM(Z,4) is simply a statement which uses this FUNCTION twice and divides the two results.

Q. When the expression SUM(X,3) is used, the value 3 is used in the FUNCTION where the variable _______ appeared.

●●●

A. N

125 Given that the X array contained the three values of 4., 3., and 5., and that the Z array contained the four values 1., 2., 0., and 1., you could tell by adding them that the expression SUM(X,3)/SUM(Z,4) is nothing more than 12. divided by 4. giving the result of 3. for the variable Y.

Q. If the third number in the X array had been 9., the value of the expression above would have been _______.

●●●

A. 4.

126 Q. Try a similar question: using the same FUNCTION SUM(A,N) and given that the B array contains the integers 1. through 5. and the C array contains the integers 2. through 5., and K equals 5: what is the value of the expression SUM(B,K)-SUM(C,K-1)?

●●●

A. 1.

127 If you answered this question correctly, you are probably getting the hang of the FUNCTION writing and usage. If not, go back to frame 86 and review the material on FUNCTION subprograms. If you feel confident of this material, go to the next frame.

128 Perform Exercise 4.5 in your problem book.

The fourth type of subprogram, the SUBROUTINE, which was mentioned earlier in this chapter, is similar to the FUNCTION in many respects: the naming rules are the same, they both require a RETURN statement and an END statement, and they both contain the same sort of dummy argument variables. Here the similarity ends, as will be explained in the next several frames.

129 Like the FUNCTION, the SUBROUTINE is also a set of commonly-used operations grouped in subprogram form to be used again and again without re-writing, but it does not restrict itself to a single value for the result, as does the FUNCTION. In fact, a SUBROUTINE can be used for almost any operation with as many results as desired.

Q. A SUBROUTINE, being a complete program segment, requires an _______ statement as the last statement.

●●●

A. END

130 The FUNCTION has a single value for its result which becomes the value of the FUNCTION name in the expression in which the name is used. Since the SUBROUTINE does not have just a single result, the manner in which the SUBROUTINE is called into action is different from the way the FUNCTION is called into action.

Q. In the statement K = WOW(L,M,N)*I+J the value of the FUNCTION will replace the name _______ in computing the expression.

●●●

A. WOW

131 As the last question illustrates, the FUNCTION is called into action by mentioning its name in an expression. The SUBROUTINE is called by a special statement: the CALL statement, which consists of the word CALL followed by the name of the subprogram and its parenthesized list of arguments.

Q. (True or False) By the above definition, the statement CALL MTMPY(A,B,N) is legal.

●●●

A. True

132 Remember, the name of a SUBROUTINE is chosen by the same rules as those of the FUNCTION subprogram. They may have 1 to 5 alphanumeric characters, the first of which must be alphabetic. Unlike the FUNCTION, however, there is no mode associated with the SUBROUTINE name.

Q. In the statement CALL MTMPY(A,B,N) the SUBROUTINE name is _______.

●●●

A. MTMPY

133 A programmer usually selects the SUBROUTINE name in a meaningful fashion. For example, the name MTMPY might be aptly applied to a SUBROUTINE which computed the product of two matrices. Since there is no numeric value associated with the name (as with a FUNCTION) there is no concern about mode.

Q. If you want to make use of a SUBROUTINE for your program, you must use a _______ statement.

●●●

A. CALL

134 The SUBROUTINE itself is constructed with FORTRAN statements. You may use any sort of statement combinations you wish to perform the desired operations. Since the SUBROUTINE is a separate subprogram the variables and statement numbers do not relate to any other program (except the dummy argument variables).

Q. A statement such as GO TO 15 in a subprogram requires that the subprogram have a statement numbered _______.

●●●

A. 15 (such a statement would not send the computer to statement number 15 in another program)

135 The SUBROUTINE subprogram must begin with a special statement, like the FUNCTION statement. This statement consists of the word SUBROUTINE followed by the subprogram name and its parenthesized list of dummy argument variables, as, for example SUBROUTINE ORDER(X,Y,N).

Q. The name of the subprogram defined in the statement above is _______.

●●●

A. ORDER

136 The SUBROUTINE subprogram must be headed by an appropriate SUBROUTINE statement as described in the previous frame; it must also end its execution with a RETURN statement, which, like the FUNCTION program, directs the computer back to the statement which called the subprogram in another program.

Q. A SUBROUTINE, like a FUNCTION, must end its execution with a _______ statement.

●●●

A. RETURN

137 Naturally, the last statement in a SUBROUTINE subprogram must be an END statement. The END statement is the one which sets off program segments as complete main programs or subprograms. Panel 4.5 demonstrates a SUBROUTINE subprogram and a program which calls upon it. The object of the subprogram is to simply copy one array directly into another. Turn to Panel 4.5.

138 The program on the right is an ordinary main program. Eventually, it executes the statement CALL COPY(X,Y,K) which will bring the SUBROUTINE program on the left into action. As a result the COPY subprogram will copy the X array into the Y array, K numbers long. The statement SUBROUTINE COPY(A,B,N) in the program on the left defines the operations performed on the dummy arguments A, B, and N such that a DO-loop is executed N times to copy the first given array into the second.

139 The preceding example demonstrates that the SUBROUTINE not only receives information from the arguments but also defines the value of some of the arguments. This is the way in which the desired results are given back to the program which called on the SUBROUTINE. Thus the argument list becomes a two-way street.

Q. In this example, the dummy argument (array) called _______ had its values developed in the subprogram.

●●●

A. B(B is the dummy argument in the SUBROUTINE; Y was the actual argument in the calling program)

140 A SUBROUTINE subprogram (or a FUNCTION, too, for that matter) can call upon another subprogram of any kind: library or built-in functions, Arithmetic statement functions, or other SUBROUTINE or FUNCTION subprograms written in this subprogram form.

Q. (True or False) The SQRT library function can be used in any SUBROUTINE subprogram.

●●●

A. True

141 Subprograms can call on other subprograms to any desired number of levels. The RETURN statement or its equivalent will always send the computer back to the program which called on the subprogram, so you are assured that control will eventually get back to your main program.

Q. If a main program calls upon a SUBROUTINE which in turn calls upon a FUNCTION, the RETURN statement in the FUNCTION will send the computer back to the (main or SUBROUTINE) _______ program.

●●●

A. SUBROUTINE (the RETURN statement always takes the computer back to the next program in sequence)

142 A SUBROUTINE can be used for a large variety of jobs: sorting arrays, matrix manipulation, input or output, polynomial operations, etc. You should use a SUBROUTINE wherever you encounter this general type of operation and need to perform it more than once in a given problem. Panel 4.6 will show a main program that reads an array from cards and calls on a SUBROUTINE to sort the array in algebraic order. Turn to Panel 4.6.

143 The main program on the left side of the panel reads the N numbers into the A array and calls in the ORDER routine to do the sorting. It then prints out the sorted array. The program on the right is the SUBROUTINE which does the sorting, using the method of comparing adjacent numbers and swapping their position if they're out of the desired order. Admittedly, the sort operation is performed only once in this example, but this is only a demonstration; in practice a SUBROUTINE would be used only where its operation is required a number of times.

144 Notice that the preceding example showed both the actual arguments and the dummy arguments with the same name. This was done to show that there is no harm in doing so, but the use of the same variable name is merely coincidental. The arguments in the SUBROUTINE are still dummies and are related to the variables of the same name in the main program only by virtue of their relative position in this argument list.

Q. (True or False) The variable I as used in both programs in this example refers to a different quantity in each program.

●●●

A. True

145 Here are some check points in the writing and using of SUBROUTINE subprograms:

1. Choose a name that contains 1 to 5 alphanumeric characters, the first of which is alphabetic.
2. Make certain that all the information needed in the SUBROUTINE is included in the arguments.
3. Make certain that all the information to be given to the calling program is also included.
4. Be sure that the SUBROUTINE has the required SUBROUTINE, RETURN, and END statements.
5. The program that calls upon the SUBROUTINE does so with a CALL statement.
6. The arguments in the calling program must agree in number and mode with the SUBROUTINE.

146 So far, then, this chapter has introduced you to four types of subprograms: library functions or built-in functions, Arithmetic statement functions, FUNCTION subprograms, and SUBROUTINE subprograms.

147 Turn to Panel 4.7. Here you see a summary of subprogram characteristics. You may use this chart for review purposes as you continue through this chapter. It is not necessary to memorize the information on this panel because when you are actually programming you will find similar information in greater detail in the 1130 FORTRAN reference manual.

Q. Here are some questions regarding subprograms:

1. Which one is referenced by a CALL statement?
2. Which subprograms do not use RETURN and END statements?
3. If you wanted to return more than one value to the main program, which subprogram would you use?

●●●

A.
1. SUBROUTINE subprogram
2. Library and Arithmetic statement functions
3. SUBROUTINE subprogram

148 Perform Exercise 4.6 in your problem book.

In addition to the subprograms discussed so far, four special FORTRAN-supplied SUBROUTINE subprograms are available to the programmer. They are provided to perform special indicator tests which are needed by many programs. Because they are SUBROUTINE subprograms, they are all referenced by using a CALL statement. The next few frames will discuss their use.

149 SLITE and SLITET are two SUBROUTINES that manipulate and test the status of sense lights. Normally, sense lights are lights on a computer console that can be turned on and off and tested by program statements. Ironically, the name "sense light" is actually a misnomer for the 1130, because there are physically no such lights! Sense lights do exist on many computers, however, and the term "sense light" is simply a holdover. Even thought there are no physical sense lights on the 1130, the SLITE and SLITET routines simulate their functions, thus the effect on the program is the same as if we actually had them.

150 Because sense lights can be turned on and off by the program, and because their status (i.e. on or off) can be tested by the program, sense lights are used where it is possible for the program itself to dictate a future branching condition. To put it another way, sense lights can be used by the programmer to set up logical branches within his program.

Q. (True or False) The program can test to see whether a "sense light" is on or off.

●●●

A. True

151 There are four "sense lights" on the 1130. To turn them on, the statement <u>CALL SLITE (i)</u> is used, where i is an integer expression and may have the value 0, 1, 2, 3, or 4. If i = 1, sense light 1 would be turned on. Similarly, if i = 2, 3, or 4, the corresponding sense light would be turned on. If i = 0, <u>all</u> sense lights would be turned <u>off</u>.

Q. What sense light would turn on as a result of the <u>CALL SLITE (4)</u> statement? _______.

●●●

A. sense light 4

152 Q. Suppose sense lights 1 and 3 were on and the statement <u>CALL SLITE(0)</u> were executed. What sense lights would be on? _______.

●●●

A. None. CALL SLITE (0) turns them all off.

153 In addition to turning sense lights on and off, the programmer also needs to be able to test their status (i.e. on or off). The SUBROUTINE to do this is SLITET. Its form is the word SLITET, followed by an integer expression, a comma, and an integer variable both enclosed in parentheses. For example, CALL SLITET (3,IAB)

Q. The name of the SUBROUTINE that tests sense lights is _______, and in the above example, the integer variable is _______.

ooo

A. SLITET, IAB

154 The integer expression (i = 1,2,3, or 4) indicates which light is to be tested, and if the light is on, the integer variable will be set to 1. If it is off, the integer variable will be set to 2. For example, CALL SLITET (2,JAR) will test sense light 2. If light 2 is on, JAR will equal 1. If light 2 is off, JAR will equal 2.

Q. If sense light 4 is off, what value will MOP have after the statement CALL SLITET (4,MOP) is executed? _______.

ooo

A. 2

155 In addition to testing a sense light, the SLITET statement will always turn off the tested light if it was on. For example, if sense light 3 were on, and we executed CALL SLITET (3,NUMBR), sense light 3 would be turned off. NUMBR, of course, would have the value of 1.

Q. Write a statement to test sense light 1 and place its tested value in JINX. _______.

ooo

A. CALL SLITET (1,JINX)

156 To make use of the SLITET test, the programmer will often test the value of the integer variable by an arithmetic IF statement. To see how this might work, turn to panel 4.8. Here's the explanation - sense light 2 is either turned on or left off depending on previous calculations in the program. If it is on, we want to compute the square root of Y later in the program. If it is off, we want to compute the cube of Y. The IF statement tests LINK to see if it equals one which would mean that sense light 2 was on. If so, statement 20 is executed, otherwise statement 10 would be executed.

Q. Would sense light 2 be on or off when statement 30 is reached? _______.

●●●

A. Off. The action of the SLITET statement restores the tested light to an off condition.

157 The 1130 has some indicators which will warn of possible error conditions. These conditions are overflow, underflow, and division by zero. If you understand the meaning of overflow and underflow you may skip to frame 160.

158 Overflow is the condition which occurs when a number becomes too large for the computer to handle due to some mathematical operation. You have probably seen a desk calculator with the adder register full of 9's and when 1 is added to this quantity - zip! All digits go to zero with an imaginary 1 overflowing to the left. The imaginary 1 has no place to go and is therefore lost. This is exactly equivalent to overflow in the computer. Underflow is the condition which occurs when a number becomes too small for the computer to handle, as in the case of extremely small fractional numbers.

Q. Overflow is the condition which occurs when a number becomes too _______, while underflow occurs when a number becomes too _______.

●●●

A. large, small

159 Consider a fictitious computer with room enough for a five-digit number: the largest number this computer can handle is 99999. Suppose that value (99999) has the quantity 10 added to it as the result of a programmed operation - the result would be 1000009, but our computer can only hold the lower five digits: 00009. This computer would tell you that your answer was 00009, which is not correct. This situation may occur on any computer which has a limit to the size of number it can handle (limit on the 1130 is approximately $10^{\pm 38}$ for real numbers).

160 **On the 1130 if either an overflow or an underflow occurs because of some arithmetic operation in the real mode, an overflow/underflow indicator is set. This indicator can be subsequently tested by a FORTRAN-supplied SUBROUTINE subprogram named OVERFL.**

Q. (True or False) The OVERFL routine will test for either an overflow or an underflow condition.

●●●

A. True

161 An overflow/underflow condition is tested by the statement CALL OVERFL(j) where j is an integer variable name. If an overflow condition exists, j is set to 1. If an underflow condition exists, j is set to 3. If neither overflow or underflow exists, j is set to 2.

Q. Given an underflow condition, what value will LAMB have after the statement <u>CALL OVERFL(LAMB)</u> is executed? ______.

●●●

A. 3

162 After execution the CALL OVERFL(j) statement will always reset the computer to a no overflow condition. For example, if an overflow/underflow condition exists and the OVERFL routine is used, the computer will show a no overflow/underflow condition after the test has been made.

Q. Given an overflow condition, what value will JIM have after the two following statements are executed sequentially?

```
CALL OVERFL(JIM)
CALL OVERFL(JIM)
```

●●●

A. 2 (after the first test JIM has a value of 1, but the computer is restored to a no overflow status. The second test will set JIM to 2, thus reflecting no overflow).

163 In practice, the integer variable which holds the overflow status would typically be tested by **a computed GO TO statement** to cause branching to an error routine. Perhaps the error routine would cause a message to print, or it might halt the computer. In any event, the OVERFL test is provided to alert the programmer to error conditions, and to allow him to take appropriate action.

164 A serious error condition can exist if you accidentally try to tell the computer to divide by the quantity zero. When this is attempted, the division does not take place and an indicator called "divide check" is turned on. The program, however, proceeds as if the division had been done correctly.

Q. The divide check indicator is turned on when an attempt is made to divide by _______.

●●●

A. zero

165 The statement CALL DVCHK(j) will test the status of the divide check indicator. If the indicator is on, the integer variable j will be set to 1. If the indicator is off, j will be set to 2. For example, if the divide check indicator is on, and the statement CALL DVCHK(NOON) is executed, NOON will have a value of 1.

Q. In the above example, what value would NOON have if divide check was not on? ________.

●●●

A. 2

166 Just as with the OVERFL test, the DVCHK test will always restore its indicator to the off condition after the test.

Q. (True or False) If a divide check occurs and is tested, another divide check would have to occur in order to turn the indicator on again.

●●●

A. True

167 Remember the conditions of overflow/underflow and divide check are not the same: the overflow/underflow condition occurs when the result of any calculation is a real number larger or smaller than the computer can handle (**$10^{\pm 38}$ on the 1130, for** example); the divide check condition occurs only when you attempt to divide by <u>zero</u>.

Q. (True or False) When a divide check condition occurs, usually by accident, the numerical results are incorrect because the divide operation does not take place.

●●●

A. True

168 Q. To see if you understand DVCHK, try this problem. What will be the next statement executed after the following sequence of statements?

```
A = 0.
ANS = Y**2/A
CALL DVCHK(MILL)
IF(MILL-2) 34,14,14
```

●●●

A. Statement 34. MILL has a value of 1 because a divide check has occurred.

169 If you did not answer the preceding question correctly you should return to frame **164** and review DVCHK.

170 This concludes our discussion of special FORTRAN-supplied SUBROUTINE subprograms. You have learned about SLITE, SLITET, OVERFL, and DVHCK. These routines allow the program to test indicators and to alter its course based on the results of the test.

171 The next topic to be discussed is the use of double subscripts when dealing with matrices. You will remember that the purpose of a subscript is simply to identify individual numbers within a group or list of numbers. Double subscripts serve the same purpose except that now we are identifying a number in a particular group when there is more than one list within that group.

172 FORTRAN permits a double-subscripting notation that is handy for referring to matrices and other groups having similar cross-reference notations. In this system, the variable name is followed by the open parenthesis, as usual, and followed in turn by two subscripts, separated by a comma, and a close parenthesis.

Q. (True or False) According to the above definition, A(I,J) is a legitimate double-subscript notation.

●●●

A. True

173 The double-subscripted array is treated as a series of sub-arrays: the first subscript denotes the position referred to within the sub-array and the second subscript refers to the particular sub-array. Thus, a 10 by 5 array called A is thought of as 5 blocks of 10 numbers each, and, for example, A(7,4) refers to the seventh number in the fourth block.

Q. A 5 by 8 array would be thought of as being _______ blocks of _______ numbers in length.

●●●

A. eight, five

174 In order to use double-subscripting notation with a variable, it must be specified in a special way: the DIMENSION statement for such a variable will have two constants in the array-size specification. The first constant denotes the sub-array length, and the second constant specifies the number of sub-arrays.

Q. (True or False) According to the above definition, the specification of a 10 by 5 array called A must be DIMENSION A(10,5).

●●●

A. True

175 Thus, an array specified by DIMENSION A(10,5) is actually a 50 number array divided into 5 blocks of 10 numbers each. Double-subscript form can be used only with variables that are double-dimensioned as above, and a variable can only be specified one way or the other (for double or single subscripting).

Q. A reference to the variable A as specified above such as A(8,3) refers to the _______ number in the _______ block of the array.

●●●

A. Eighth, third

176 Any combination of double subscripts will refer to a unique position in the overall array; that is, there is no ambiguity such that two different combinations might refer to the same quantity. The only limitation on the subscript values is that they cannot exceed their corresponding DIMENSION constant.

Q. (True or False) With a specification of <u>DIMENSION X(10,10)</u> a reference to X(12,5) is legal.

●●●

A. False (the first subscript, 12, exceeds the first DIMENSION constant, 10)

177 The array-size specifications in the DIMENSION statement must be constants that fix the array size when the program is written. The subscripts, however, may be variables or limited expressions, as in single subscript form, as long as their values are not permitted to exceed their corresponding DIMENSION specification.

Q. With a specification of <u>DIMENSION F(5,5)</u> the reference of F(K,5) is legal as long as K does not exceed _______.

●●●

A. 5

178 The term "limited expression" in the preceding frame refers to the fact that either or both of the double subscripts may be in any one of the following forms: constant, variable, variable plus (or minus) a constant, constant times a variable, or constant times a variable plus (or minus) a constant.

Q. (True or False) The above-defined forms are the only ones permitted for single or double subscripts.

●●●

A. True

179 The following examples represent <u>legitimate</u> uses of double-subscripting form:

```
A(1,1)              BLOCK(I,J)
X12(2*K,6)          ARRAY(N-1,M-1)
VALUE(K,5)          VECT(2*K-6,4*K+1)
NUMBR(1,INDEX)      LIST(5,4)
```

180 Whether or not a double subscript combinations involves constants, variables, or "limited expressions", the resultant values of the subscripts will make reference to a particular quantity in the array. For example, with a specification of DIMENSION A(5,5) a reference to A(I,J) with values of 3 and 4 for I and J respectively, will refer to the third number in the fourth block of the A array.

Q. The A array as specified above contains a total of ______ numbers.

●●●

A. 25

181 Double subscript notation is very useful for such arrays as matrices. A matrix is nothing more than a group of numbers in rows and columns of a rectangular array. A 3 by 3 matrix is shown on Panel **4.9** in symbols.

182 The matrix on Panel **4.9** uses double subscripts to denote row and column position in the matrix. For example, a_{23} is in the second row and third column. You can see that a matrix could be represented in FORTRAN notation with double subscripts in the same fashion.

Q. An array A is a matrix; the reference to A(1,3) refers to the ______ row and the ______ column.

●●●

A. first, third

183 A rectangular array, then, can be expressed in a single array divided into groups of sub-arrays, where each sub-array is a column (the first subscript is the row count). FORTRAN then becomes a useful tool for matrix handling, which in turn is a valuable tool for many mathematical problems.

Q. A 6 by 8 matrix called AMATR would be specified by the statement ______.

●●●

A. DIMENSION AMATR(6,8)

184 A doubly-subscripted array is actually laid out in single-array form in the following order (using a 3 by 4 array for illustration):

$$\underbrace{A(1,1)\ A(2,1)\ A(3,1)}_{\text{first column}} \qquad \underbrace{A(1,2)\ A(2,2)\ A(3,2)}_{\text{second column}}$$

$$\underbrace{A(1,3)\ A(2,3)\ A(3,3)}_{\text{third column}} \qquad \underbrace{A(1,4)\ A(2,4)\ A(3,4)}_{\text{fourth column}}$$

Q. The first number in each column, collected together, makes up a _______ of the matrix.

●●●

A. row

185 Since the computer must treat these two-dimensional arrays as single arrays partitioned into sub-arrays, it is important to understand how such an array is ordered. The list in the preceding frame shows this order, with the first subscript in each column advancing more rapidly than the second subscript as the numbers in the list are counted off.

Q. A double-subscripted reference to the fourth number in the sixth sub-array (column) of a matrix called ARRAY would be _______.

●●●

A. ARRAY(4,6)

186 The 1130 version of FORTRAN also permits a three-dimensional array notation. This is specified in a DIMENSION statement with three constants. The size of such an array is determined as in two-dimensional arrays, by the product of the DIMENSION constants: a 10 by 10 by 10 array contains a total of 1000 numbers.

Q. The statement DIMENSION A(6,5,4) specifies a _______ dimensional array.

●●●

A. three (containing 120 numbers)

187 Arrays which are specified in a DIMENSION statement as three-dimensional arrays are referred to in the program with triple-subscript form. The rules for such subscripts are the same as for two-dimensional arrays: the same form can be used for the subscripts themselves, and they are separated by commas.

Q. (True or False) A reference such as ARRAY(I,J,K) is legal in three-dimensional arrays.

●●●

A. True

188 The three-dimensional array is treated by the computer as a single array divided into sub-arrays, each of which is a two-dimensional array sub-divided as usual. For example, a 2 by 3 by 4 array is a 24-number array consisting of four groups, each of which is a 2 by 3 array. Each 2 by 3 array is, of course, three groups of two numbers each.

Q. The references A(1), B(1,1), and C(1,1,1) all refer to the (first or last) _______ quantity in their respective arrays.

●●●

A. first

189 The order of the equivalent single array of a three-dimensional array is determined by a similar rule to that of two-dimensional arrays: in counting off their positions, the first subscript is advanced most rapidly; each time it reaches its limit, the second subscript is advanced; each time the second subscript reaches its limit, the third subscript is advanced. An example showing this occurs in the next frame.

Q. In a 4 by 5 by 6 array, the reference A(4,5,6) refers to the (first or last) _______ number in the array.

●●●

A. last

190 The following list is the order (in single array form) of a three-dimensional array, 2 by 3 by 4, using symbols:

```
A(1,1,1) A(2,1,1) A(1,2,1) A(2,2,1) A(1,3,1) A(2,3,1) A(1,1,2) A(2,1,2)
A(1,2,2) A(2,2,2) A(1,3,2) A(2,3,2) A(1,1,3) A(2,1,3) A(1,2,3) A(2,2,3)
A(1,3,3) A(2,3,3) A(1,1,4) A(2,1,4) A(1,2,4) A(2,2,4) A(1,3,4) A(2,3,4)
```

191 The multiple-subscripting form extends to self-indexed lists for input and output statements also. The double-subscript notation used with self-indexed lists permits two index definitions acting like nested DO-loops. This is denoted with parentheses within parentheses, each pair containing an index definition.

192 An Input or Output statement may have two-dimensional arrays in its list. The self-indexing form then contains two index definitions. The format of this notation is: two open parentheses, the variable name, two dummy subscripts (variables) in parentheses, a comma, and an index definition for one of the dummy subscripts, followed by a close parenthesis balancing the innermost of the outside open-parentheses. Then comes a comma and a second index definition for the other dummy subscript, followed by the last parenthesis.

Q. (True or False) According to the above definition, the statement READ(2,1) ((A(I,J),I = 1,10), J = 1,20) is a legal example of two-dimensional self-indexing form.

●●●

A. True

193 The foregoing example: READ(2,1) ((A(I,J), I = 1,10), J = 1,20) demonstrates the list which will read in a two-dimensional array called A. As pointed out before, this double indexing behaves like nested DO-loops, such that I in this example is cycled from 1 to 10, then J is stepped by 1 before I cycles again.

Q. The dummy subscripts in the example shown above are _______ and _______.

●●●

A. I, J

194 The statement READ(2,1) ((A(I,J),I = 1,10), J = 1,20) will read 200 quantities into the A array in the following order: A(1,1), A(2,1), A(3,1),.,A(10,1) A(1,2), A(2,2), A(3,2),...,A(10,20). In other words, the innermost index definition corresponds to the innermost DO-loop, cycling most rapidly.

Q. The 31st quantity read by the statement shown above will be placed in A(________, _______).

●●●

A. 1, 4 (the first number in the fourth block of ten)

195 As the example READ(2,1) ((A(I,J), I = 1,10),J = 1,20) has demonstrated, the double-subscripted version of self-indexed lists is nothing more than an ordinary self-indexed sequence (inner parentheses) which cycles normally and repeats its complete cycle according to the control of the second index definition (outer parentheses). For every combination of index values a specific reference is made to the indicated array.

Q. (True or False) The outer parentheses enclosing the second index definition must also enclose the entire self-indexed sequence.

●●●

A. True

196 Failure to place commas where required causes problems for FORTRAN programmers. A statement such as READ(2,1)((A(I,J), I = 1,10),J = 1,20) has six commas, and every one of them is absolutely required! Note their position, and be sure that you include them, not only for this type of statement, but for all statements which require commas.

Q. Punctuate the statement: WRITE(3 1)((A(JK)J = 1 5)K = 1 8)

●●●

A. WRITE(3,1)((A(J,K),J = 1,5),K = 1,8) (six commas in all)

197 While it seems perfectly natural to read double-subscripted arrays by cycling the first subscript most rapidly in the self-indexed notation, it is perfectly permissible to reverse the index definitions, as follows: READ (2,1)((A(I,J),J = 1,10), I = 1,20) which will read A(1,1),A(1,2),A(1,3), etc. in that order.

Q. Reversing the index definitions as given in the previous example, the _______ (first or second) subscript varies most rapidly in the index cycling.

●●●

A. second

198 Two dimensional arrays that are matrices are often read by a statement such as READ(2,1)((A(I,J),J = 1,10), I = 1,20) so that the numbers can be punched on the cards in row order rather than column order. Remember this statement reads in the order A(1,1),A(1,2),A(1,3), etc., which corresponds to the row designation.

Q. The "normal" order of A(1,1),A(2,1),A(3,1), etc. corresponds to _______ (row or column) order for matrices.

●●●

A. column

199 Any two-dimensional array that is thought of in terms of rows and columns should certainly be printed in rows of numbers that represent the rows of the matrix. The reverse indexing scheme just illustrated for reading is usually applied to printing for this reason.

Q. Write a statement to print the matrix A using I and J for dummy subscripts, printing the 10 by 10 array row-wise. (Use 3 as the output reference number and 1 as the FORMAT statement number). _______.

●●●

A. WRITE(3,1)((A(I,J),J = 1,10),I = 1,10)

200. Like the single self-indexed list with one index, the double indexed form can be used with more than one variable: for example, READ(2,1),((X(I,J),Y(I,J),I = 1,10), J = 1,100) will read a pair of numbers for each combination in indices, cycling them in the usual fashion.

Q. The third number read with the above statement will be placed in _______ (_______, _______).

●●●

A. X(2,1) The first number in X(1,1), the second, in Y(1,1).

201 Any Input or Output statement can have more than one self-indexed list (single or double indexed) in a single statement along with single variables if desired. For example, a list can become as complicated as WRITE(3,1)M,N,(A(I),I = 1,M), ((B(I,J),I = 1,M),J=1,N) which contains two single variables and single and double indexed lists.

Q. The third value printed with the above statement is (name) _______.

●●●

A. A(1)

202 So far, then, you have seen examples of double-subscript notation, which can be used with any variable appearing in a DIMENSION statement in two-dimensional form, used in arithmetic expressions and in Input and Output statements. The latter application requires two index definitions for the two dummy subscripts in the list, with two sets of parentheses setting off the inner and outer definitions. These double index specifications then behave like nested DO-loops, cycling the inner index for each value of the outer.

203 Q. Write a statement to read from the paper tape reader according to FORMAT number 15, reading first a 20 x 20 array called BLOCK and then a 10 by 50 array called ARRAY, reading the former in row-order (second subscript cycled most rapidly) and the latter in column order. Use I and J for the subscript-index variables.

●●●

A. READ(4,15)((BLOCK(I,J),J = 1,20), I = 1,20),((ARRAY(I,J), I = 1,10),J = 1,50)

204 If your answer agrees exactly with the one shown (all twelve commas and fourteen parentheses required) you may skip to frame 226. If your answer was incorrect, go to the next frame.

205 The problem was to read paper tape with FORMAT number 15, so the first part of the statement poses no difficulty: READ(4,15)..; the real test is in constructing the two double indexed lists correctly. The first array, named BLOCK, is 20 by 20 in size and is to be read row-wise, meaning that the second dummy index is cycled by the inner index definition, as in ((BLOCK(I,J),J = 1,20),I = 1,20).

Q. The third number read with the list shown above will become the value of BLOCK (_______, _______).

●●●

A 1,3

206 The second array, named ARRAY, is 10 by 50 in size and is to be read column-wise, meaning that its first dummy subscript is to be cycled with the inner index definition (most rapidly). Thus a double indexed list for this array looks like ((ARRAY(I,J),I = 1,10),J = 1,50). Since the problem stated that you were to read the first array completely and then read the second array, the two separate indexed lists appear in sequence in the input statement.

Q. The third number read with the list shown above will become the value of ARRAY (_______, _______).

●●●

A. 3,1

207 Q. **Try a similar problem. Write a statement to write on the typewriter according to FORMAT number 1010, writing the AMATR array, of dimension 30 by 50, row-wise followed by the values of the variable X and Y. Use dummy subscripts I and J as before.**

ooo

A. `WRITE(1,1010)((AMATR(I,J),J=1,50),I=1,30),X,Y`

208 If you answered this question correctly, you have probably corrected your difficulties. If you still have problems with double indexed lists, go back to frame **191** and review this material.

209 Perform Exercise **4.7** in your problem book.

The last two specification statements to be covered in this chapter are the EQUIVALENCE and the COMMON statements. These statements are provided to give the programmer some control over the assignment of variable names and the allocation of storage locations to variables. Only some basic uses of these statements will be covered in the following frames, as the more complex forms are beyond the scope of this course. Additional details can be found in the reference manuals for **1130 FORTRAN.**

210 Because a computer has a limited number of storage locations, a programmer will often want to write his program so that it uses as few storage locations as possible. If this is not done, the storage locations needed by the program may exceed the storage capacity of the computer, thus forcing the program to be rewritten more efficiently or to be broken into smaller segments to be run at different times.

211 The total amount of storage needed for a given program is in part composed of all of the storage needed for data (i.e. variables and constants). Every time the programmer defines a new variable name, storage locations are reserved for that variable. As the program grows in size, more and more locations are set aside for the storage of data that the program uses when it is executed.

Q. (True or False) Each variable name reserves storage locations.

ooo

A. True

212 One way of saving storage is to use the same storage locations for storing more than one variable, thus in effect making the locations do multiple duty. The EQUIVALENCE statement allows the programmer to assign different variables to the <u>same</u> storage locations.

Q. If the variable X, QUACK, and GO were to be assigned the same storage location, the kind of specification statement needed is _______.

●●●

A. EQUIVALENCE

213 The form of the EQUIVALENCE statement consists of the word EQUIVALENCE followed by sets of parentheses that contain any number of variables to be assigned the same storage locations. The variables within the parentheses and the sets of parentheses are separated by commas. For example, the statement <u>EQUIVALENCE(A,B,CAD),(HAH,HUH,T)</u> would assign the same storage location to A, B, and CAD, and would assign another storage location to HAH, HUH, and T.

Q. Write an EQUIVALENCE statement to assign the same storage location to the variables ALPHA, BETA, GAMMA, DELTA. _______.

●●●

A. EQUIVALENCE (ALPHA,BETA,GAMMA,DELTA)

214 The variables listed in an EQUIVALENCE statement may be subscripted, as long as the subscript is an integer constant. The variables may also be of different modes.

Q. (True or False) The statement EQUIVALENCE(JACK,G,HQ), (A(1),B(K)) is legal.

●●●

A. False (B(K) is not an integer constant subscript)

215 When a subscripted variable is made equivalent to another subscripted variable, an equivalence is thus established between other elements of their arrays. Consider this example:
EQUIVALENCE (A(4),I(2),D(1)). The equivalences established would be

```
A(1)
A(2)
A(3)I(1) - Share the same location
A(4)I(2)D(1) - Share the same location
A(5)I(3)D(2) - Share the same location
 .   .   .   - Share the same location
 .   .   .   - Share the same location
```

Thus, A(3) and I(1) share the same location, A(4), I(2), D(1), share the same location, A(5), I(3), D(2) share the same location, and so on. Remember, all of this is the result of just one EQUIVALENCE statement.

Q. Write an EQUIVALENCE statement to make the first 50 elements of the BOX array equivalent to the first 50 elements of the 100 element WOOD array. _______.

●●●

A. EQUIVALENCE(BOX(1),WOOD(1))

216 Q. Let's try a variation. Write an EQUIVALENCE statement to make the first 50 elements of the BOX array equivalent to the last 50 elements of the 100 element WOOD array. _______.

●●●

A. EQUIVALENCE(BOX(1),WOOD(51))

217 One important point must be kept in mind when using the EQUIVALENCE statement. Because the same array location can be referenced by different variable names, the programmer must make sure that the variables sharing that location are never needed at the same time. In practice, equivalences might be set up between variables used early in the program and never again referenced, and variables used later in the program.

Q. (True or False) Variable names that are made equivalent must be referenced at completely separate and distinct times within a program.

●●●

A. True

218 You have seen in the preceding frames that the EQUIVALENCE statement will allow different variables within the same program to share the same storage locations. Storage locations can also be shared between subprogram variables and main program variables through the use of the COMMON statement.

Q. Storage within the same program is shared through use of the _______ statement, but storage between subprograms and main programs is shared through use of the _______ statement.

●●●

A. EQUIVALENCE, COMMON

219 The COMMON statement consists of the word COMMON followed by a list of variable or array names. The names in the list are separated by commas. For example: COMMON APPLE, X, BETA, RHO

Q. (True or False) The statement COMMON, ALPHA, BETA is correct.

●●●

A. False (there should be no comma after COMMON)

220 A COMMON statement, like one scissors blade, is usually not used alone. When a COMMON statement is used in a program or subprogram, it will normally have at least one counterpart in another program. For example, if a main program contains the statement COMMON BREAD and a subprogram contains the statement COMMON BUTTR, the variable names BREAD and BUTTR will refer to the same storage location.

Q. (True or False) COMMON statements work together between main programs and subprograms.

●●●

A. True

221 As noted before, COMMON statements may contain more than one variable name. When this happens, the variables will share respective storage locations according to the order in which they appear in the COMMON statements. For example, if the main program contains the statement COMMON A,B,C and a subprogram contains the statement COMMON X,Y,Z; then A will share the same location as X; B will share the same location as Y; and C will share the same location as Z.

Q. Given the main program statement COMMON FUMAN,RAIN,GAMMA and the subprogram statement COMMON CHU,SPAIN,SWIFT show which pairs of variables will share the same locations.

1. ________, ________
2. ________, ________
3. ________, ________

●●●

A.
1. FUMAN, CHU
2. RAIN, SPAIN
3. GAMMA, SWIFT

222 Incidentally, there is no restriction as to the number of programs that can have COMMON statements. Thus, storage locations can be shared by many programs and subprograms.

Q. Given the following statements appearing in different programs and subprograms, show which variable names will share the same locations: COMMON BREAK,ABLE,SHORT; COMMON CATS, FAST, SQUAR; COMMON UMBRA, CORN,QUEUE.

1. ________, ________, ________
2. ________, ________, ________
3. ________, ________, ________

●●●

A.
1. BREAK, CATS, UMBRA
2. ABLE, FAST, CORN
3. SHORT, SQUAR, QUEUE

223 As the previous frame implies, the order in which variables appear in a COMMON statement governs how they will be matched with variables in other COMMON statements. Within a specific program or subprogram, variables and arrays are assigned storage locations in the sequence in which their names appear in a COMMON statement.

Q. A subprogram contains the statement COMMON HYPER,BEND and a main program has the variable ZETA which is to share the same location as BEND. What position in the list of the main program COMMON statement should ZETA occupy? _______.

● ● ●

A. ZETA should be the second position in the list, i.e., COMMON _______,ZETA.

224 In the previous frame, if there is no main program variable to be made COMMON to HYPER, a so-called "dummy" variable can be used to allow ZETA to maintain its correct position in the list. For example, assuming that X is the dummy variable, the main program COMMON statement would be COMMON X,ZETA. If ZETA had to be the third name in the list, two dummy variables could be used, e.g., COMMON X,Y,ZETA where both X and Y are dummy variables.

Q. (True or False) Dummy variables can be used to maintain the relative positions of other variables in a COMMON statement.

● ● ●

A. True

225 Variables used to pad out the COMMON statements are called dummys because they are not referenced anywhere else in the program. Their function is simply to allow the programmer to position variable names that otherwise would be in the wrong locations in a COMMON statement.

Q. A subprogram contains the statement COMMON AB,DROP,Q,ARRAY. A main program has a variable GROUP that is to share the same storage locations as ARRAY. Write a COMMON statement for the main program to accomplish this. (Make up your own dummy variable names). _______.

● ● ●

A. COMMON A1,B1,C1,GROUP. A1,B1, and C1 are the dummy variables in this statement. Other legitimate variable names could have been used.

226 So far, you have seen that the COMMON statements allows the sharing of storage between main programs and subprograms. The savings in storage can be quite significant, particularly when storage for arrays is being shared.

227 The COMMON statement does one more important thing, however. Because subprograms and main programs now have access to common storage locations, they have, in effect, a way of communicating with each other. Thus, a value computed by a subprogram can be placed in common storage and can be grabbed by the main program, or for that matter, by another subprogram. This facility is quite valuable in actual practice, and together with storage saving, makes the COMMON statement a useful programming tool.

Q. Give two important reasons for using the COMMON statement. _______.

●●●

A. 1. To save storage
2. To allow programs and subprograms to communicate

228 **Just like all the other specification statements you have learned in this course, the COMMON and EQUIVALENCE statements must precede all executable statements in the source program, and have a definite place among the specification statements. COMMON statements follow DIMENSION statements, and EQUIVALENCE follows COMMON. Here at last, is the final line-up of the source deck as presented in this course:**

Type statements
DIMENSION statements
COMMON statements
EQUIVALENCE statements
FORMAT statements (may be placed elsewhere)
Function defining statements
Executable program steps
END

229 That concludes the material about the FORTRAN language covered in this manual. The next several frames illustrate the ways of preparing a written program for actual execution on the computer. Naturally, not all points can be covered in detail, but the general concepts will be covered. Computing centers will be glad to furnish details and rules concerning their handling of FORTRAN programs.

230 All FORTRAN programs are written in basically the same language, which, as you are probably aware, is not the natural language of the computer. Consequently, each computer system must have its own means of translating a FORTRAN program into its own code system (language).

231 Fortunately, a program is supplied for the 1130 which will do this translation from the FORTRAN language into the computer's instruction code. This program is called the FORTRAN compiler and it reads a FORTRAN program as its input and produces a computer-coded program as output.

Q. The program which translates programs from FORTRAN language into computer code is called a _______ _______.

●●●

A. FORTRAN compiler

232 Once a program is written in FORTRAN language, the process of executing the program on the computer becomes a two-stage process: translating from FORTRAN to computer code and running the computer-coded version to get the answers.

Q. (True or False) Before the program can be executed on a computer, it must first be translated into the computer's own code.

●●●

A. True

233 Every computer equipped to handle FORTRAN programs, then, must have its own compiler which will read your FORTRAN program and produce for you a program ready to execute on that computer (or a similar computer).

Q. The input to the FORTRAN compiler is a _______ _______.

●●●

A. FORTRAN program

234 Obviously, there must be some way to transcribe your FORTRAN program into a form that the compiler can read. This is usually done by punching the statements on cards with a card-punch machine. FORTRAN statements are usually punched one statement per card (it is possible to have more than one card per statement) with a card layout as shown on Panel 4.10. Turn to it.

235 In this type of card layout a statement number, if any, is punched in columns 1 through 5 of the card, and the statement itself is punched anywhere in columns 7 through 72. As long as the statement appears in the proper field (columns 7 to 72) there is no restriction as to which columns are used for any part of the statement. You may use as many blanks as you like to separate names and symbols, but it is not permissible to have blanks within a name. Thus, GObbbbTOb15 is legal, but GbObTOb15 is not because of the blank between the G and the O.

236 Turn to Panel 4.11. It shows two different ways of punching the same FORTRAN statement. As far as the translating program is concerned, one way is as good as the other. But from the standpoint of writing the statement on the FORTRAN coding form and punching it in the FORTRAN statement card, the second way is obviously more practical than the other.

237 A FORTRAN program must be punched statement by statement into cards in the format shown on Panel 4.11. It often helps to have written the program on a coding sheet which is blocked off like the card layout. You have been looking at pictures of typical FORTRAN coding sheets in panels which illustrate program segments.

238 Blank or unpunched columns are also counted by the compiler as being significant when they are included within the quotation marks in a literal data format statement. You will remember that this is the way of writing alphanumeric information, and the blank is a legal character in this set. A statement such as 1 FORMAT('bTHISbISbTHEbOUTPUT') might be punched on a card as shown on Panel 4.12. Turn to it.

239 Statements which are too long to fit in a single card can be continued on up to five additional cards by placing a non-zero punch in column 6 of the card for each card which is a continuation of the preceding card (that is, the first card in such a series does not have such a punch). Turn to Panel 4.13.

240 The Panel shows a FORMAT statement with a literal data field so long that it requires three cards to contain the entire statement. Notice that the first card is a normal card (no continuation punch in column 6) but the next two have 1 and 2 respectively in this column, meaning that they are continuations of the statement beginning on the first card. The character chosen for the continuation mark is arbitrary; often the integers 1 to 5 are used to indicate the order. The effect of this punch is to create an equivalent card image that places columns 6 to 72 of the continuation card immediately after column 72 of the preceding one.

241 Turn to Panel 4.14. Any FORTRAN statement that has the letter C punched in column 1 (normally reserved for statement numbers) will not be included in the translated program. In fact, this card can contain any sequence of punches you desire. This permits you to intersperse comments that explain the program, and these statements (with C in column 1) will be printed in the copy of the FORTRAN statements that the computer produces with the translation.

242 Turn to Panel 4.15. The information found there will give you an idea of the step-by-step procedure followed in developing a total FORTRAN programming job.

243 This concludes this course in basic FORTRAN. You must understand that this course has covered only the elementary features of the language, along with a few common techniques of programming with FORTRAN. Many systems also provide advanced features such as BOOLEAN algebra statements, complex arithmetic, logical IF statements, special de-bugging aids, monitoring of the job execution, etc.. It is hoped that this course has given you sufficient background to understand the language (our main objective) and, with experience, to make use of it properly.

244 You may proceed to the examination for Chapter 4 on page 43 of your problem book, after which you may report to your advisor.

REFERENCE INDEX

R29-0104-0

Printed in U.S.A.

R29-0104-0

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York