SECTION VI
User-Defined Functions

A user-defined function is one that is defined within the user program and is called within that program in the same way that a built-in function is called. The name of a function that returns a numeric value consists of three letters, the first two of which are "FN" (e.g., FNA, FNB, etc.). String-valued functions have the same set of names available with the addition of a dollar sign ($) at the end (e.g., FNA$, FNB$,...). Thus, up to 26 numeric-valued functions and 26 string-valued functions are allowed in any one program.

A function is called within an expression by referring to its name and a list of parameter values enclosed in parentheses. The value returned by the function takes its place in the expression.

There are two levels of complexity in the definition of a BASIC/3000 function. At the simple level, a one-line function simply relates a function name and list of parameters to any expression which may use the parameters to calculate the result value. The multiline function is a more complex entity; it can consist of many statements and local variable declarations. It returns its result value with a RETURN statement.

For a discussion of BASIC/3000 built-in functions, see Functions in Section II. A complete list of the built-in functions available to the BASIC/3000 user is contained in Appendix E.

One-line Function

A one-line function is defined completely in one line, using the function DEF statement; its result is calculated by an expression.

Form

The forms of one-line function definitions are:

DEF function-name(formal parameter list)=numeric expression

DEF type function-name(formal parameter list)=numeric expression

DEF string-function-name(formal parameter list)=string expression

where *function-name* has the form *FNletter*, and *string-function-name* has the form *FNletter$*. The function is referenced by this name.

If included, *type* specifies the data type of the result: INTEGER, REAL, LONG, or COMPLEX. If omitted, the data type is assumed to be real. The type specification is not allowed in the third form that returns a string result.

The *formal parameter list* includes

- Real parameters (i.e., untyped variables)
- Typed parameters (i.e., a type indicator followed by a variable name)
- String parameters (i.e., string variables)

Array parameters are indicated by a (*) or (*,*) following the variable name to indicate the number of dimensions. String parameters are indicated by an optional (*) following the name of a simple string variable, or by a required (*,*) following an array string variable name. Parameters in the parameter list are separated by commas.

The expression can be any legal numeric or string expression, and can make use of both parameters and other program variables.

Explanation

The parameters in a function definition are formal parameters; when the function is called, they are replaced by the actual parameters which are passed to the function. All variables used as formal parameters are local to the function; that is, they are unrelated to any program variables having the same name. The formal and actual parameters are matched according to their position in the list.

The DEF statement is not executable; the function it defines can be entered only by referring to the function name within an expression.

All parameters are assumed to be real unless the name contains a $ or is preceded by a type indicator or by another parameter that is typed. Untyped parameters that follow a string parameter are treated as real.

Examples

```
10 DEF FNA(A,B)=(A*B)+(A/B)
```

The function FNA is type real. The formal parameters A and B are also type real. When called, the actual parameters will give values to A and B, the expression ((A*B)+(A/B)) will be evaluated, and the result will replace the function name where it appears in an expression.

```
20 DEF FNA$(A$,B$)=A$+B$+"STOP"
```

The function FNA$ is a string function. The formal parameters A$ and B$ are string variables that will be assigned values according to the matching actual parameters in the function call. When called, the literal string resulting from the concatenation of the values of A$,B$, and "STOP" will replace the function name in the expression where it appears.

```
30 DEF FNB(INTEGER A,X2)=A*X2+(A/X2)
```

The function FNB is a numeric function that results in a real value when called. The computations will be performed in integer arithmetic because both A and X2 are integers.

Multiline Function

A multiline function is written as several contiguous statements beginning with a DEF statement and ending with an FNEND statement. Execution of the function ends when a function RETURN statement is encountered; this sends the result value back to the place of call.

Form

A multiline function definition has three parts; the function head, the function body, and the function end.

The function head appears as

DEF function-name(formal parameter list)

DEF type function-name(formal parameter list)

DEF string-function-name(formal parameter list)

The first two forms return numeric values, the third returns a string value.

All parts of these function definitions are the same as described for one-line functions.

The function body consists of a sequence of statements, including at least one function RETURN statement:

RETURN numeric expression

RETURN string expression

The *expression* is numeric or string depending on whether the function is numeric or string. For numeric functions, the RETURN expression is converted to the type of the function.

The function end consists of a one-word statement:

FNEND

This statement must always be the last statement in the function definition.

Explanation

The body of a function can contain any BASIC/3000 statements with the following restrictions:

- Local variables except formal parameters must be declared (even if they are type real) using the type or DIM statements; they can duplicate the name of other variables because they are known only within this function and are created dynamically each time the function is called.
- A function definition cannot appear within a function body, but function calls are allowed, including calls to the same function.
- The function body must be self-contained; FOR loops and DO-blocks must be completed within the body and branches must not occur into or out of the body.

The formal parameters in a multiline function head are specified in the same way as those in the one-line function definition.

Examples

In the following multiline function, there are no local variables. It returns a long value.

```
100 DEF LONG FNX(A,B,INTEGER X,Y,LONG M,N,P)
120   M=A**-Y
130   N=B**Y
140   P=M*N
150   RETURN P
160 FNEND
```

The following multiline function definition returns an integer value:

```
210 DEF INTEGER FNM(A[*],INTEGER N)
215   REM-A(*) IS REAL ARRAY-FNM RETURNS INDEX OF LARGEST ELEMENT
220   REAL I,J
230   J=1
240   FOR I=2 TO N
250     IF A[I]>A[J] THEN J=I
260   NEXT I
270   RETURN J
280 FNEND
```

The use of the variables I and J are local to the function. It is good practice to define local variables within the function definition.

The following multiline function returns a string value; its formal parameter is a string variable:

```
10 DEF FNR$(A$)
20   REM..FNR$ RETURNS THE REVERSE OF A$
30   IF LEN(A$)<=1 THEN RETURN A$
40   RETURN FNR$(A$[2])+A$[1,1]
50 FNEND
```

The functions defined in these three examples and the one-line function definitions in the previous set of examples will be called in the examples under Calling A User-Defined Function.

Calling a User-Defined Function

A user-defined function is called by referring within an expression to the function name followed by a list of actual parameters in parentheses. The function call is replaced by the value returned by the function.

Form

A function call has the form:

function-name(actual parameter list)

string-function-name(actual parameter list)

The first form is a numeric function; a real type result is returned unless the name in the function definition is preceded by a type specification: INTEGER, LONG, or COMPLEX. The second form is a string function and a string result is returned.

The *actual parameter list* contains one or more *actual parameters* separated by commas. An *actual parameter* may be:

- numeric expression
- numeric array name followed by (*) or (*,*)
- string expression
- string variable name optionally followed by (*)
- string array name followed by (*,*)

Explanation

Actual parameters may be used to pass single values or entire arrays to a function, usually to be used within the function although this is not required. Even if parameter values are not needed in the execution of the function, at least one "dummy" parameter must be included in the formal parameter list of the function definition and in the actual parameter list of the function call.

The number of actual parameters in the function call must be the same as the number of formal parameters in the function definition. The names of corresponding parameters need not be the same. Actual and formal parameters correspond according to their positions in the two lists. For instance, the third actual parameter in a function call corresponds to the third formal parameter in the DEF statement.

If the formal parameter is a simple numeric variable (V) then the actual parameter can be a numeric expression resulting in a single value, or a simple or subscripted numeric variable (2*V,V,5*7, V(5)). If the variables are different types or the actual parameter is an expression, any necessary conversion is performed as described in Section IV, Numeric Assignment.

If the formal parameter is a simple string variable (V$ or V$(*)) the corresponding actual parameter must be either a string expression or a simple string variable ("ABC", "X", V$, V$(*), "A"+"B").

If the formal parameter is a numeric array (A(*) or A(*,*)) then the actual parameter must also be a numeric array of the same type and number of dimensions. No conversion is performed in this case.

If the formal parameter is a string array (A$(*,*)) then the corresponding actual parameter also must be a string array.

Examples

To call the one-line function:

```
10 DEF FNA(A,B)=(A*B)+(A/B)
```

the actual parameters are numeric variables of the same type:

```
500 LET X=2.57,Y=7.98
510 PRINT FNA(X,Y)
>RUN
 20.8307
```

The actual parameters might also be numeric expressions:

```
520 PRINT FNA(2.57,7.98)
>RUN
 20.8307
```

To call the string function:

```
20 DEF FNA$(A$,B$)=A$+B$+"STOP"
```

The actual parameters can be string variables:

```
530 X$="O",Y$=""
540 PRINT FNA$(X$,Y$)
>RUN
OSTOP
```

or string expressions:

```
550 PRINT FNA$("A","1")
>RUN
A1STOP
```

To call the function FNB returning a real value:

```
30 DEF FNB(INTEGER A,X2)=A*X2+(A/X2)
```

the actual parameters can be variables:

```
 500 LET X=150,Y=2*35
 510 PRINT FNB(X,Y)
>RUN
 10502
```

or integer expressions:

```
 520 PRINT FNB(500,2.7)
>RUN
 1666
```

Each of the above examples is a one-line function for which a single value is returned. The formal parameters are not affected by execution of the function. In a multiline function, the formal parameters may be altered in the body of the function. Depending on the type of actual parameter passed to the function, the value of the actual parameter may also be affected by the change to the formal parameter.

In the multiline function below, the values of the formal parameters are not changed by execution of the function:

```
210 DEF INTEGER FNM(A[*],INTEGER N)
215   REM-A(*) IS REAL ARRAY-FNM RETURNS INDEX OF LARGEST ELEMENT
220   REAL I,J
230   J=1
240   FOR I=2 TO N
250     IF A[I]>A[J] THEN J=I
260   NEXT I
270   RETURN J
280 FNEND
```

To call this function, the actual parameters may be an entire numeric array and a numeric variable:

```
300 DIM X[5]
305 LET Q=5
310 READ (FOR N=1 TO 5,X[N])
320 DATA 2732.1,765.32,7905.1,6543.89,195.72
330 PRINT FNM(X[*],Q)
RUN
3
```

or an entire numeric array and a numeric expression:

```
 300 DIM X[5]
 310 READ (FOR N=1 TO 5,X[N])
 320 DATA 2732.1,765.32,7905.1,6543.89,195.72
 330 PRINT FNM(X[*],5)
>RUN
 3
```

The multiline function below returns a string value that is the reverse of the string value input as the actual parameter:

```
10 DEF FNR$(A$)
20   REM..FNR$ RETURNS THE REVERSE OF A$
30   IF LEN(A$)<=1 THEN RETURN A$
40   RETURN FNR$(A$[2])+A$[1,1]
50 FNEND
```

To call this function, the actual parameter may be a string literal:

```
  70 PRINT FNR$("ABCDE")
>RUN
EDCBA
```

The actual parameter may also be a string variable:

```
  60 DIM X$[5]
  70 X$="12345"
  80 PRINT "FNR$ RETURNS:";FNR$(X$)
>RUN
FNR$ RETURNS:54321
```

PASSING PARAMETERS BY REFERENCE OR BY VALUE

When the value of a formal parameter may change during execution of the function, the value of the actual parameter is affected only if it is passed by *reference*. If the actual parameter is passed by *value*, then it will not be affected even if the value of the formal parameter is changed.

Parameters are passed by reference when the actual parameter is:

- an entire array, string or numeric
- a simple or subscripted numeric variable matching the formal parameter in type
- a simple or subscripted string variable without substring designators

In each of the above cases, a change to the formal parameter value during execution of the function also affects the value of the actual parameter.

Parameters are passed by value when the actual parameter is:

- any expression *except*: a simple or subscripted numeric variable of matching type or a simple or subscripted string variable with no substring designators
- a simple or subscripted numeric variable that differs in type from the corresponding formal parameter
- a simple or subscripted string variable with substring designators

Regardless of whether the formal parameter value is changed, these actual parameters will not be affected.

Examples

In the function definition FNX, three formal parameters are affected by execution of the function; they are the long variables M, N, and P:

```
100 DEF LONG FNX(A,B,INTEGER X,Y,LONG M,N,P)
120   M=A**-Y
130   N=B**Y
140   P=M*N
150   RETURN P
160 FNEND
```

In the function call, line 520 below, the actual parameters corresponding to M, N, and P are all passed by value because they are variables of a different type. These actual parameters are not affected when the formal parameters are modified. The actual parameters corresponding to X and Y are also passed by value because they are expressions. The actual variables A1 and B1 are passed by reference being variables matching the formal parameters in type. These variables are not changed because their corresponding formal parameters are not altered during execution.

```
500 LET M1=N1=P1=0
510 LET A1=8,B1=3
520 PRINT FNX(A1,B1,3,2,M1,N1,P1)
530 PRINT M1,N1,P1
>RUN
 1.406250000000000L-01
 0                0                0
```

In the following call, the variables M1, N1, and P1, used as actual parameters, are the same type as the values of the formal parameters M, N, and P. These actual parameters are, for this reason, passed by reference and their values will be set to the values of M, N, and P respectively following execution of the function call:

```
 500 LONG M1,N1,P1
 510 LET A1=8,B1=3
 520 X1=3,Y1=2
 530 PRINT "VALUE OF FNX=";FNX(A1,B1,X1,Y1,M1,N1,P1)
 540 PRINT "M1=";M1,"N1=";N1
 550 PRINT "P1=";P1
>RUN
VALUE OF FNX= 1.406250000000000L-01
M1= 1.562500000000000L-02      N1= 9.000000000000000L+00
P1= 1.406250000000000L-01
```

Note that M1, N1, and P1 do not have values prior to their use as actual parameters in the function call.

The table below summarizes the relations between actual and formal parameters:

Actual Parameter	Formal Parameter	Attribute
V, A(e), A(e,e)	V_s	reference
V, A(e), A(e,e)	V_d	value
e[1]	V	value
A(*)	A(*)	reference
A(*,*)	A(*,*)	reference
V$ or V$(*), A$(e)	V$ or V$(*)	reference
V$(e), V$(e,e), V$(e;e)	V$ or V$(*)	value
A$(e,e), A$(e,e,e), A$(e,e;e)	V$ or V$(*)	value
e$[1]	V$ or V$(*)	value
A$(*,*)	A$(*,*)	reference

V= simple numeric variable name

V_s=simple numeric variable name, same type

V_d=simple numeric variable name, different type

A= numeric array name

e= numeric expression

V$=simple string variable name

A$=string array variable name

e$=string expression

[1] not including an expression consisting of V, A(e), A(e,e), V$, V$(*), or A$(e) alone.