SECTION X
Segmentation

Because the maximum size of a BASIC/3000 program is necessarily limited by memory resources, BASIC/3000 provides language facilities for segmenting programs into units that can call each other. Each unit or subprogram must be saved in the user's library; from there it may be called by the currently executing program into the user's work area.

Two statements are used for interprogram transfer: INVOKE and CHAIN; and one statement, COM, allows variables to be used in common by several programs.

CHAIN Statement

The CHAIN statement terminates the current program and begins execution of another program, optionally starting at a specified statement number.

Form

The forms of CHAIN are:

CHAIN string expression

CHAIN string expression,integer expression

The *string expression*, when evaluated, is the name of a BASIC/3000 program that is in the user's library. This may be a fully qualified file name (see Section VIII, Files). When evaluated, the optional *integer expression* is a label in the called program. If present, execution begins at the first executable statement at or after the label; the exact label need not be present in the called program. If omitted, execution begins at the first executable statement in the called program.

Explanation

CHAIN calls the program identified by the string expression, and it replaces the current program. When the program called by CHAIN finishes execution, it terminates and does not automatically return to the calling program. The called program may call another program, including the original calling program, with another CHAIN statement or an INVOKE statement.

Only variables declared in a COM statement are saved during a CHAIN operation. All variables and arrays of the current program that were not declared in COM are lost when the new program begins execution, and all files opened in the current program are closed.

Examples

```
MAIN
  10 REM..PROGRAM MAIN
  20 LET X=200,A=X**3
  30 PRINT "A=";A
  40 PRINT "LEAVE MAIN AND ENTER SUBA AT LINE 30"
  50 CHAIN "SUBA",30
  60 REM..THIS STATEMENT IS NEVER EXECUTED


SUBA
  10 REM..PROGRAM SUBA
  20 PRINT "THIS STATEMENT IS NOT EXECUTED"
  30 PRINT "ENTER SUBA - LINE 30"
  40 LET B=125,C=B**2
  50 PRINT "C=";C
  60 PRINT "END OF SUBA - TERMINATE HERE"
  70 END
>RUN MAIN
MAIN
A= 8.00000E+06
LEAVE MAIN AND ENTER SUBA AT LINE 30
ENTER SUBA - LINE 30
C= 15625
END OF SUBA - TERMINATE HERE
```

The main program, MAIN, calls program SUBA with a CHAIN command in line 50. Execution of SUBA begins in line 30, and execution terminates with the last line of SUBA. None of the variable values from MAIN are saved following execution of CHAIN.

INVOKE Statement

The INVOKE statement is similar to the CHAIN statement, except that the calling program is suspended rather than terminated and resumes execution when the program called by INVOKE terminates. The called program can be explicitly terminated with an END statement, otherwise it is implicitly terminated by the end of the program. In both cases, control returns to the suspended program that performed the INVOKE. If the called program is terminated by a STOP statement or a terminal error, the entire run is terminated including any suspended programs.

Form

The forms of INVOKE are:

INVOKE string expression

INVOKE string expression,integer expression

The *string expression* evaluates to the name of a BASIC/3000 program in the user's library. It may be a fully qualified file name. The *integer expression* evaluates to the label of a starting statement.

Explanation

The statements and variables of the current program are saved in a temporary file created by BASIC/3000 so that execution of the program can be continued at a later time. Variables declared in COM statements are passed to the program called by INVOKE. Files opened in the current program are not closed by INVOKE.

Execution of the new program begins at the first executable statement in the program or, if specified, at or after the label. The exact label need not be present in the called program.

When an implicit or explicit END statement in the called program is executed, control returns to the suspended program at the point immediately following the INVOKE statement. A STOP statement in the called program will, however, terminate the entire run including any suspended programs.

INVOKE operations can be nested; that is, a program that has been called by INVOKE can itself invoke another program including a recursive INVOKE of itself.

Examples

```
MAINX
  10 REM..PROGRAM MAINX
  20 LET A=25
  30 INVOKE "SUBY"
  40 PRINT "CONTROL RETURNS TO MAINX"
  50 PRINT "C=A*10=";A*10
SUBY
  10 REM..PROGRAM SUBY
  20 PRINT "EXECUTION OF SUBY BEGINS"
  30 PRINT "B=";8**-3
  40 PRINT "END OF SUBY - RETURN TO MAINX"
  50 END

>RUN MAINX
MAINX
EXECUTION OF SUBY BEGINS
B= 1.95313E-03
END OF SUBY - RETURN TO MAINX
CONTROL RETURNS TO MAINX
C=A*10= 250
```

MAINX uses an INVOKE command in line 30 to call for execution of SUBY. Execution begins with the first executable statement in SUBY. When SUBY terminates, control returns to line 40 of MAINX. The value of the variable A has been saved during execution of SUBY.

EXAMPLE USING CHAIN AND INVOKE

The example below has four programs: A, B, C, and D. All four programs have been saved in the user's library. The command RUN A brings A into the user's work area as the current program. With each successive CHAIN or INVOKE, a new program replaces the previous program in the work area; in this case A is the final as well as the first program in the work area.

When D terminates it returns to C, and when C terminates it returns to A because C was chained to by B which was invoked by A.

```
A
  10 REM..PROGRAM A
  20 PRINT "ENTER PROGRAM A"
  30 INVOKE "B"
  40 PRINT "BACK IN PROGRAM A - TERMINATE"



B
  10 REM..PROGRAM B
  20 PRINT "ENTER PROGRAM B"
  30 CHAIN "C",30
  40 REM..CONTROL SHOULD NEVER RETURN TO THIS POINT
  41 REM..DUE TO CHAIN IN LINE 30
  50 PRINT "NOT TO BE EXECUTED - B"



C
  10 REM..PROGRAM C - EXECUTION STARTS IN LINE 30
  20 PRINT "NOT TO BE EXECUTED - C"
  30 PRINT "ENTER PROGRAM C - LINE 30"
  40 INVOKE "D",25
  50 PRINT "BACK IN C - RETURN TO A"
  60 REM..END IN LINE 70 WILL CAUSE RETURN TO PROGRAM A
  61 REM..BECAUSE PROGRAM B WAS INVOKED BY PROGRAM A
  62 REM..NOTE THAT THE "END" IS UNNECESSARY
  70 END



D
   5 REM..PROGRAM D - EXECUTION STARTS IN LINE 25
  15 PRINT "NOT TO BE EXECUTED - D"
  25 PRINT "ENTER PROGRAM D - LINE 25"
  35 PRINT "RETURN TO PROGRAM C"
  45 REM..IMPLICIT END WILL CAUSE RETURN TO PROGRAM C
  47 REM  NOTE THAT EXPLICIT "END" WOULD DO THE SAME
>RUN A
A
ENTER PROGRAM A
ENTER PROGRAM B
ENTER PROGRAM C - LINE 30
ENTER PROGRAM D - LINE 25
RETURN TO PROGRAM C
BACK IN C - RETURN TO A
BACK IN PROGRAM A - TERMINATE
```

Files and Segmentation

Within a program, a file is referenced by its file number as determined by its position in the FILES statement (see Section VIII). When programs call one another with CHAIN or INVOKE, it is possible to reference files that were declared with FILES statements in other programs.

BASIC/3000 maintains an internal file numbering scheme that assigns an internal file number to every file declared in a FILES statement. For main programs, these numbers are the same as the file numbers to which the user refers. Files declared in a program called by INVOKE are assigned internal file numbers beginning with a value one greater than the last internal file number. Files declared in a program called by CHAIN are assigned internal file numbers beginning with the same number as the files in the calling program. Whether the program is a main program, is called by INVOKE, or is called by CHAIN, the local file numbers used to refer to files within each program will begin with 1.

When a program calls another program with INVOKE, all the files in the calling program remain open; when a program calls with CHAIN, the files in the calling program are closed.

To illustrate:

```
M1
  10 REM  MAIN PROGRAM M1
  20 FILES A,B,C
  30 INVOKE "M2"

M2
  10 REM  SUBPROGRAM M2
  20 FILES D,E,F
  30 CHAIN "M3"

M3
  10 REM  SUBPROGRAM M3
  20 FILES G,H
```

The internal and local file numbers for this group of programs is:

Internal File Numbers	Local File Numbers
A = 1	M1
B = 2	A = 1
C = 3	B = 2
	C = 3
D = 4	M2
E = 5	D = 1
F = 6	E = 2
	F = 3
G = 4	M3
H = 5	G = 1
(6 is unassigned)	H = 2

By using the *#integer* file designator in a FILES statement, a program may reference files declared in another program that invoked it. The value of the integer is the internal file number of a previously declared file; the position of the designator in the FILES statement is used to assign the local file number.

Examples

```
M1
  10 FILES A,B,C
  20 PRINT #1,1;"FILE A"
  30 PRINT #2,1;"FILE B"
  40 PRINT #3,1;"FILE C"
  50 INVOKE "M2"

M2
  10 FILES D,E,F,#3
  20 DIM A$[6]
  30 PRINT #1,1;"FILE D"
  40 PRINT #2,1;"FILE E"
  50 PRINT #3,1;"FILE F"
  60 READ #4,1;A$
  70 PRINT A$
  80 CHAIN "M3"

M3
  10 FILES G,H,#1,#2
  20 DIM A$[6],B$[6]
  30 PRINT #1,1;"FILE G"
  40 PRINT #2,1;"FILE H"
  50 READ #3,1;A$
  60 READ #4,1;B$
  70 PRINT A$,B$

>RUN M1
M1
FILE C
FILE A           FILE B
```

Within M2, reference to local file #4 is the same as a reference to internal file number 3 (file C). However, the statement FILES D,E,F,#3 is not the same as the statement FILES D,E,F,C. This latter statement, if specified, would have treated file C as a logically different file from file C in M1; the file would have been reopened with new buffers and access restrictions. By using #3, there is only one logical file C and any accessing affects that file, and the file is not reopened. In the same way, the statement FILES G,H,#1,#2 in M3 differs from a possible statement FILES G,H,A,B.

COM Statement

The COM statement is used to pass data values between segmented programs. Variables specified in a COM statement are placed in a common area so that values assigned to these variables in one program will be retained when transferring to another program with CHAIN or INVOKE. This area is known as a COM block. There may be more than one COM block, and it may or may not have an identifying label.

COM statements must precede all DIM, Type, or DEF statements in a program. All typing and dimensioning of variables is done within the COM statement, and any variables that appear in a COM statement must not also appear in a type or DIM statement in the same program.

BASIC/3000 permits ten COM blocks for each run, one unlabeled and nine labeled uniquely with the digits 1 through 9.

Form

The forms of the COM statement are:

COM com item list

COM(nonzero digit)com item list

The *com item list* consists of a list of variable declarations. Simple variables are indicated by the variable name; arrays are indicated by the array name and a bounds indicator. The bounds indicator is equivalent to the dimension specification used in a DIM statement if the block is being created; it indicates only the number of dimensions if the block has been created and is currently active. The number of dimensions are specified with (*) or (*,*).

The type of items in the *com item list* is assumed to be real unless the variable name contains a $ to indicate a string variable, or the variable is preceded by a type specifier (INTEGER, LONG, COMPLEX, or REAL). The type specifier assigns that type to all succeeding variables until the end of the list or the next type specifier or string variable.

The optional block indicator is specified as a *nonzero digit* between 1 and 9. This assigns a label to a block being created, or specifies an existing labeled block.

Explanation

Programs execute in BASIC/3000 on a dynamic level basis. The original program run with the RUN command starts at level 1. When a CHAIN is executed, the new program executes also at level 1 since the old program terminates. However, when an INVOKE is executed, the new program executes at level n+1 where n is the level of the invoking program. When control returns to the invoking program, the level is reduced to n.

COM blocks become active whenever a program declares a COM block that is not currently existing or active. COM blocks created at level n are active until the dynamic level drops to n-1 or the run terminates.

If a COM statement references an inactive COM block, then numeric bounds for the arrays and strings are specified as in a DIM statement. If, however, a COM statement references an active block, then it need only indicate the number of dimensions. For a one-dimensional array or a string variable, the variable name is followed by (*); for a two-dimensional array or a string array, the variable name is followed by (*,*). The (*) can be omitted for simple string variables and will be assumed. Numeric bounds may be specified for an active COM block, but in this case they must be identical to the original COM statement bounds.

CORRESPONDENCE RULES. All variables in COM statements that reference an active COM block must match exactly in type, number of dimensions, and order within the COM statement. Also, if dimension size is specified instead of the *, these must match exactly. The names of corresponding variables need not be the same since equivalence is based on the order of appearance of the variables in the COM statements. A COM statement defining an active block must contain the same number of elements as the COM statement that created the block. More than one COM statement in a program can define the same block if the statements are contiguous.

The order of elements in a COM statement, or in more than one contiguous COM statement, implies the order of the variables in the COM block.

The rules governing correspondence between COM statements are checked when a CHAIN or INVOKE statement is executed. If any of these conditions is not met, a terminal error occurs.

Examples

Program A1 chains to program B1 which in turn chains to program C1:

```
A1
  10 COM B,B$[5],C,INTEGER D,E[5],LONG F[5,2]
  20 COM(5) Q,COMPLEX A,A$[2,2],P
  30 LET B=10,B$="ABCDE",C=20.5,D=1
  40 MAT READ E,F
  50 DATA 1,2,3,4,5,1.01L11,5.2L10,1.57L11,1.76L11,1.76L-9,1.53L11
  60 DATA 1.575L-4,1.57L6,1.57L-5,1.752L10
  70 LET Q=1954.75,A=(12.3,4),P=3.14
  80 MAT READ A$
  90 DATA "AB","CD"
 100 CHAIN "B1"
```

```
B1
  10 COM T,C$[* ],Q,INTEGER F,D[* ],LONG F1[*,* ]
  20 PRINT "START B1"
  30 PRINT T,C$,Q,F
  40 MAT PRINT D;F1
  50 CHAIN "C1"

C1
  10 COM(5) A,COMPLEX B,C$[*,* ],D
  20 PRINT "START C1"
  30 PRINT A,B,D
  40 MAT PRINT C$

>RUN A1
A1
START B1
 10              ABCDE              20.5               1
 1       2       3       4       5

 1.01000000000000L+11               5.20000000000000L+10

 1.57000000000000L+11               1.76000000000000L+11

 1.76000000000000L-09               1.53000000000000L+11

 1.57500000000000L-04               1.57000000000000L+06

 1.57000000000000L-05               1.75200000000000L+10

START C1
 1954.75         ( 1.23000E+01, 4.00000E+00)      3.14
AB

CD
```

In this example, program A1 creates the unlabeled COM block and also COM block 5. Program B1 references the unlabeled COM block and prints the data assigned to that block in A1. Program C1 references COM block 5 and prints the data assigned to that block in A1. In both B1 and C1, the number of dimensions of subscripted COM variables is indicated by (*) or (*,*).

Assume that program D1 chains to program E1:

```
D1
  10 COM(9) A[3,5],INTEGER D[6,6]
  20 MAT A=ZER
  30 MAT D=IDN
  40 CHAIN "E1"


E1
  10 COM(9) B[3,5],INTEGER P[6,6]
  20 MAT PRINT B;LIN(1),P;
>RUN D1
D1
 0      0      0      0      0

 0      0      0      0      0

 0      0      0      0      0


 1      0      0      0      0      0

 0      1      0      0      0      0

 0      0      1      0      0      0

 0      0      0      1      0      0

 0      0      0      0      1      0

 0      0      0      0      0      1
```

In this example, program D1 creates COM block 9, and program E1 references the data in block 9. Actual numeric bounds are specified in the COM statement in E1. This is legal only if the bounds are identical to the original bounds specification.

In the following example, execution of the four programs starts in W:

W
```
  10 REM PROGRAM W
  20 COM(3) B[5]
  30 FOR I=1 TO 5
  40   B[I]=I
  50 NEXT I
  60 INVOKE "X"
  70 PRINT LIN(5);"BACK IN W - B="
  80 MAT PRINT B
  90 CHAIN "Z"
```

X
```
  10 REM.. PROGRAM X
  20 COM(3) A[*]
  30 COM(4) LONG C[3,2]
  40 PRINT LIN(5);"IN X -- A=";LIN(1)
  50 MAT PRINT A
  60 FOR I=1 TO 5
  70   A[I]=10*I
  80 NEXT I
  90 FOR I=1 TO 3
 100   FOR J=1 TO 2
 110     C[I,J]=10*I+J
 120   NEXT J
 130 NEXT I
 140 INVOKE "Y"
 150 PRINT LIN(5);"BACK IN X -- C=";LIN(1)
 160 MAT PRINT C
 170 PRINT LIN(1);"RETURN TO W"
```

Y
```
  10 REM.. PROGRAM Y
  20 COM(4) LONG F[*,*]
  30 PRINT LIN(5);"IN Y -- F=";LIN(1)
  40 MAT PRINT F
  50 FOR I=1 TO 3
  60   FOR J=1 TO 2
  70     F[I,J]=100*I+10*J
  80   NEXT J
  90 NEXT I
 100 PRINT LIN(1);"RETURN TO X"
```

Z
```
  10 REM.. PROGRAM Z
  20 COM(4) INTEGER I,J,K
  30 PRINT LIN(5);"IN Z-- COM(4)=";LIN(1)
  40 IF UND(I) THEN PRINT "  I UNDEFINED"
  50 ELSE PRINT "  I=";I
  60 IF UND(J) THEN PRINT "  J UNDEFINED"
  70 ELSE PRINT "  J=";J
  80 IF UND(K) THEN PRINT "  K UNDEFINED"
  90 ELSE PRINT "  K=";K
 100 PRINT "TERMINATE IN Z"
```

```
>RUN W
W


IN X -- A=

 1                  2                   3                   4                   5


IN Y -- F=

 1.10000000000000000L+01                1.20000000000000000L+01

 2.10000000000000000L+01                2.20000000000000000L+01

 3.10000000000000000L+01                3.20000000000000000L+01

RETURN TO X


BACK IN X -- C=

 1.10000000000000000L+02                1.20000000000000000L+02

 2.10000000000000000L+02                2.20000000000000000L+02

 3.10000000000000000L+02                3.20000000000000000L+02

RETURN TO W


BACK IN W - B=
 10                 20                  30                  40                  50




IN Z-- COM(4)=

  I UNDEFINED
  J UNDEFINED
  K UNDEFINED
TERMINATE IN Z
```

Note that Z can create a new COM block 4 since execution has dropped below the dynamic level 2 at which X created the first COM block 4.

The following three examples illustrate illegal COM usage:

Assume that program F1 invokes program G1:

```
F1
  10 COM(1) A[10,10],LONG B[10,10]
  20 MAT A=CON
  30 MAT B=ZER
  40 INVOKE "G1"


G1
  10 COM(1) LONG D[*,*],REAL E[*,*]
  20 MAT PRINT D,E
>RUN F1
F1
COM NOT SAME AS FIRST OCCURENCE IN G1
```

This is illegal because the corresponding variables in the two COM statements do not agree in type and order. If the type references were interchanged in program G1, this would be a legal example.

Assume that program HH chains to program JJ:

```
HH
  10 COM(7) A$[3,5],B[2,5]
  20 MAT B=IDN
  30 MAT READ A$
  40 DATA "ABCDE","FGHIJ","LMNOP"
  50 CHAIN "JJ"


JJ
  10 COM(7) Q$[*],D[*,*]
  20 PRINT Q$
  30 MAT PRINT D
>RUN HH
HH
COM NOT SAME AS FIRST OCCURENCE IN JJ
```

This is illegal because Q$ is a simple string variable, whereas A$ is a string array.

Assume that program K1 invokes program P1:

```
K1
  10 COM(8) X[*,*]
  20 INVOKE "P1"
  30 MAT PRINT X


P1
  10 COM(8) Y[10,3]
  20 MAT Y=ZER
>RUN K1
MISSING SUBSCRIPT SPECIFICATION IN FIRST OCCURENCE OF COM IN K1
```

This is illegal because the creator of a COM block must specify the actual physical size of arrays and strings. If, however, P1 invoked K1 this would be a legal example.