A PARTIAL IMPLIMENTATION OF THE CONTOUR MODEL OF BLOCK STRUCTURED PROCESSES

by

LYN D. LAVIANA

B.A., Kansas State University, 1972

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1974

Approved by:

William J. Hankley
Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

I. Introduction

The beginning (and frequently the more advanced) student of block structured languages often can be helped in understanding the implications of this structure by studying John B. Johnston's Contour Model [1]. Stepping through a program by hand using the Contour Model is effective, but tedious. A more useful aid is to show the contours and step by step execution of a program on a display screen. This would allow the student to step through the program examining each change as it happens without becomming involved in the many hardcopy pages necessary when this is done by hand.

The purpose of this paper is to describe a program that uses the Contour Model to display a predefined sequence of execution of a program. In other words, the user must know what each snapshot will be before writing the program necessary to display them. A logical extension of this program is to add an interpreter in front of the display modules so the input could be the program the user wishes to display using the Contour Model. An overall description of this system along with a suggested plan of attack for the interpreter is provided in the next section.

The display routines described in the third and fourth sections are written in FORTRAN for the Computek 300 CRT [2]. Due to the small size of the display screen (256x256), large complicated snapshots cannot be displayed in their entirety. Therefore, a useful addition to the display routines would be to automatically change the scale of the picture or to allow the user to determine the scale. The user has the option of stepping through the display or allowing the display to proceed naturally to its completion.

II. Overview of the System

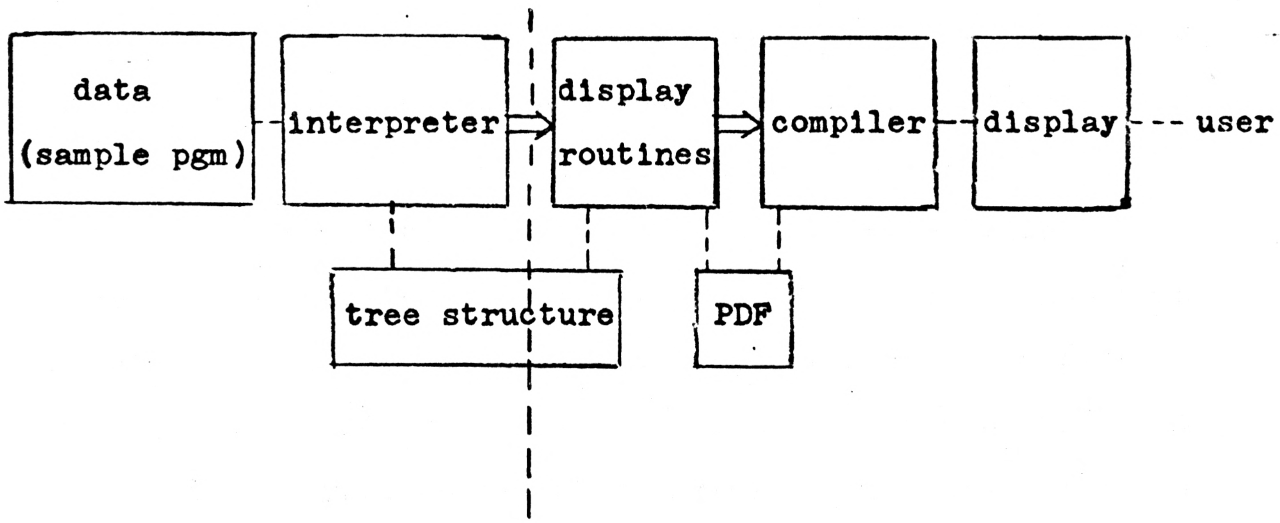


Figure 1.

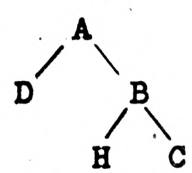
Figure 1 shows a possible system for implementing the Contour Model. As can be seen, it consists of two main parts, an interpreter and a display package. A program to be displayed would be used as input to the interpreter. As the interpreter stepped through the program, it would keep a record of execution to be translated to a series of calls to the display routines. At the same time, it would build a tree structure which contained information about the individual contours that would be created and destroyed. The routines to build such a tree structure currently exist for the purpose of defining the screen coordinates of the contour for the existing display routines.

The final output from the interpreter would be calls to the tree building and display routines. The routines to produce the tree could be executed immediately, but the display calls must be saved until the tree routines are finished or until the interpreter is through. This could be done by setting up an n x 4 array using each row to represent one call to a display routine and the first column to contain the number of the routine to be executed. Each of the remaining three columns would contain the arguments. Whenever a contour was created or deleted, a call to the proper display routine would be inserted in this array and a call to a corresponding tree building routine executed immediately. Since there is currently no interpreter, the user must produce a series of calls to the tree building and display routines as described in Section IV.

If the display routines always created contours as large or as small as possible, the continued erasing and redrawing of the picture might be highly distracting to the viewer, particularly if a low speed display is being used. Therefore, the display is done in two parts. Whenever a contour is created, it is immediately put on a tree, when it is deleted, it is removed. Then, before the display is started, a minimum size is calculated for each contour starting with the leaf nodes on the tree. These calculations take into account the numbers of variables in the contour and the placement of the processor in it. From these sizes, the lower left and upper right graph points of each contour are found along with the position of the processor in the positive quadrant. It is these graph points that are referred to by the display routines to obtain the final screen coordinates. The tree structure is linked through father, son, brother, and synonym pointers. If a contour, 'B', is created and then deleted as a son of contour 'A', and a new contour, 'C', is created as the son of 'A', this new contour is a synonym of the deleted contour, 'B', since it can occupy the same space in the display.

When all calls to the tree routines are completed, routines are called to find the minimum size each contour must be and the displacement of the processor from the lower left hand corner of the contour if the processor were to appear in it. The tree and size information are saved in the two arrays, NAME(20) and T(10,20). The name of a newly created contour is entered in NAME in the first unused space. The corresponding column in T is used to store the other information as Figure 2 illustrates. The first column is a dummy node used to make processing easier. The synonym pointer and processor height are reused later for other information.

In Figure 2, if the synonym pointer is ignored, the tree produced is:



NAME(20)

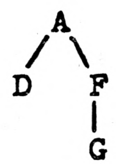
		A	D	B	H	C	F	G	. . .

T(10,20)

# of variables	0	2	3	2	2	1	2	3	
level #	0	1	2	2	3	3	2	3	
father	0	1	2	2	4	4	2	7	
son	0	3	0	5	0	0	8	0	
brother	0	0	4	0	6	0	0	0	. . .
synonym	0	0	0	7	0	0	0	0	
processor height	0	143	6	6	6	6	6	6	
processor width	0	92	14	14	14	14	14	14	
height of contour	0	161	48	62	38	28	62	48	
width of contour	0	241	78	237	78	78	237	78	

Figure 2

If B's synonym pointer is followed instead of B, the tree is:



This implies that contours B, H, and C were deleted before F and G were created. This information is used in calculating each contour's position since F and G can be displayed in the same positions occupied by B and H because they are never simultaneously displayed. All the information kept in T is used to find the screen coordinates for each contour. The lower left hand and upper right hand points of each contour are saved in the array S(4,20). The start position of the processor in the contour is saved in P(2,20). When all this information is found, the display routines can be executed. Information is passed to these routines through

NAME, T, S, P, and the arguments in the call statements. The display routines then call graphic primitives defined in an existing system [2,3] to put information into a pseudo display file (PDF). This file is processed by an existing "compiler" which actually handles communication with the Computek. The display routines allow a certain amount of interaction with the user as described later.

III. The Display

When used properly, the routines written for the display result in the creation of snapshots as described by Johnston with some minor notational differences. When a block is entered, a contour is created. Each contour is labeled outside the upper right hand corner. Every variable defined in the block is represented by a cell in the upper left hand corner. The cell is divided into three parts, the first for the variable name, the second for the environment and the third for the value. The environment is represented by the name of a contour rather than an arrow. In the case of a local integer variable, the environment is blank rather than dividing the cell into only two parts as Johnston does. A processor is displayed as Johnston defines it, however the environment pointer is never expressed as an arrow but always implicitly defined by its position inside the environment contour.

Positioning the contours on the screen must be done by an explicit algorithm. Therefore, in order to use space efficiently and still maintain a simple algorithm, alternate levels of contours are stacked in alternate directions in the display. For example, if A1 is the father of B1 and B2, and B1 is the father of C1, D1, and E1, contours B1 and B2 are displayed on top of each other inside A1, but contours C1, D1, and E1, since they are on the next lower level in the tree, are displayed side by side inside B1 as shown in Figure 3. The Π's in the diagram indicate the position of the processor if it were to be displayed in each contour.

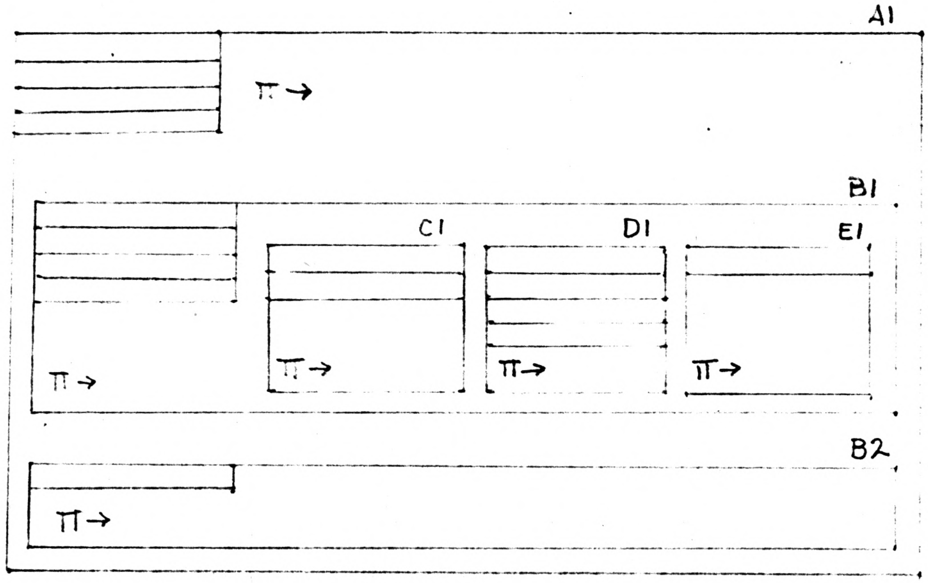


Figure 3

From the user's point of view, it would be helpful to be able to stop the display to examine it whenever he wished. Therefore, when the display begins, the message "# OF INSTRUCTION STEPS" appears at the bottom of the screen. The user responds with an integer number, n, followed by a carriage return. The display will then continue until the instruction pointer has changed n times. At this point, the display will stop and the message is repeated. In this way, the user may step through the program one statement at a time or in steps as large as he likes.

Currently, any series of contours displayed must fit entirely on the screen, This is a severe limitation as far as the user is concerned. Therefore, a useful addition to the system would be some scaling facilities. A description of a suggested scaling mechanism can be found in Appendix V. The following paragraphs describe a possible method of interfacing this scaling with the user.

If the response to "# OF INSTRUCTION STEPS" is -1 or a number that is larger than the number of steps to complete the program, the display would go to completion and return to the system. If the response were 0, scaling information would be requested via the message "AUTO (0) OR MANUAL (1) SCALING".

If manual scaling were to be requested, the "OUTERMOST CONTOUR" would be requested. The response should be a contour name consisting of no more than four lowercase letters. The effect would be to erase the screen and rescale existing contours to display this contour as the outermost one on the screen. This may result in inner contours becomming too small to display. All subsequent action would take place with this same constant scale. The scale could be changed at any time by changing the response to "OUTERMOST CONTOUR".

By default, scaling would be done automatically. A return to the automatic mode could be effected by responding with 0 when the message "AUTO (0) OR MANUAL (1) SCALING" was displayed. In this mode, if all contours happened to fit on the screen, no scaling would be done. If all contours did not fit on the screen at the same time, when the program started, the outermost contour would be displayed as the maximum size that would fit on the screen. Futher contours would be displayed until one was too small to contain all the pertinent data. The screen would be erased and the displayed contours rescaled so the father of the new contour would be the maximum screen size or as close to it as possible. The new contour would then be displayed according to this new scale.

A similar scaling algorithm could be used when deleting a contour. Alternately, if the contour to be deleted is not currently displayed on the screen, a message could be displayed in the lower right hand corner of the screen indicating that the contour was deleted.

IV. Display Routines

The display routines will be illustrated using the example in Figures 4-6. Unless indicated otherwise, all arguments in the routines are character strings of length one to four.

Figute 4 is a sample program that calculates factorials with the use of a recursive subroutine. The snapshots in Figure 6 show the record of execution of this program if

```
1   B1  begin integer N; label R, S;
2   F       procedure FACT(F); value F; integer F,X;
            begin
3               if F>1 then
4   B2              begin
5                   X=F-1;
6                   FACT(X);
7                   N=N*X;
8                   end;
9               else
10                  if F=0 then
11                      F=1;
12          end;
13      R: read N;
14         if N<0 then
15             goto S;
16         FACT(N);
17         print N;
18         goto R;
19      S: end;
```

Figure 4

the input is 2 and -1. This implies the tree structure of contours indicated by Figure 5. In this diagram, the last digit in each name indicates the activation number of a

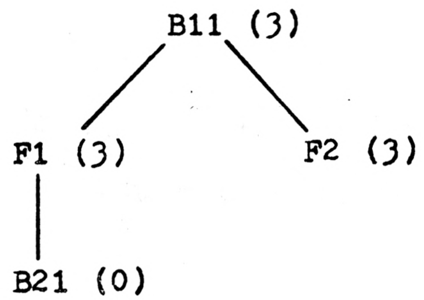


Figure 5

contour. The numbers in parentheses indicate the number of variables defined in each block.

The first step is to find the size of each contour and thus find their position on the screen. A list of the routines necessary to do this follows.

TINIT: This routine is to initialize the tree builder routines.

TREE(C1,C2,N): Place contour C1 on the tree as the son of C2. Save N as the number of variables defined in the contour including the return address if C1 is a subroutine. N is an integer.

DTREE(C1): Logically remove the contour C1 from the tree, but keep related information about it in T.

TREE and DTREE are called in the order of block execution in order to keep track of which contours must be simultaneously displayed.

SIZE: Find and save the graph position of the lower left and upper right corner of each contour once the entire tree is built. This routine must be called after all elements are on the tree and prior to calling the display routines.

Since Figure 5 shows the tree structure desired for the program in Figure 4 and execution of the program indicates that the first block entered is B1, then F, B2, and F again, the necessary statements to build the tree are:

```
CALL TINIT
CALL TREE('B11',' ',3)
CALL TREE('F1','B11',3)
CALL TREE('B21','F1',0)
CALL TREE('F2',B11',3)
```

The blocks are exited in exactly the reverse order so they are removed from the tree as follows:

```
CALL DTREE('F2')
CALL DTREE('B21')
CALL DTREE('F1')
CALL DTREE('B11')
CALL SIZE
```

In this case, all the elements were put on the tree before any were removed. This is not usually the case as different programs produce a much varied system of entering and leaving blocks. SIZE is called to complete the tree operations.

Once the dimensions of each contour are set, the actual display may begin. The specific routines to do this are listed below. Except for INCON, all routines change the display in some way. All arguments are character strings of no more than four characters unless otherwise specified.

INCON: This routine must be called first to initialize the display routines.

CP(N1,N2): Create a processor ($\pi \rightarrow$) contained in contour N2 pointing to instruction N1, N1 must be between five and eight characters long if it is given as a literal string. Otherwise, the variable must be declared CHARACTER*8. Contour N2 does not have to be currently displayed. The question '# OF INSTRUCTION STEPS' is displayed at the bottom of the screen. The user's response is described in Section III.

CC(C1): Create contour C1, i.e. display it on the screen.

CNAME(N1,N2,N3): Fill in a variable cell in the last contour defined. Normally a call to CC is immediately followed by as many CNAME references as are necessary to define every name in that contour. N1 is the variable name, N2 its environment and N3 its value. N2 is blank in the case of a local integer constant. N3 is blank if the variable is undefined upon entry to the block.

IP(N1): This routine replaces the old instruction pointer with the new one, N1. N1 must be between five and eight characters as described in the create processor routine. The message '# OF INSTRUCTION STEPS' may be displayed as described in Section III.

EP(N1): Change the environment pointer of the processor by moving it to the new contour N1. The instruction pointer is not saved, therefore, a call to EP must be followed immediately by a call to IP to redefine the instruction pointer.

AVAL(N1,N2,N3): Change the value of the variable N1 contained in contour N2 to the value of N3.

DC(N1): Delete contour N1 from the screen.

AREAD(N1,C1,VAL): Simulate a read instruction by changing the value of the variable N1 contained in contour C1 to VAL. The message 'READ:' N1 'IN' C1 'IS' VAL is displayed at the bottom of the screen to indicate a read has taken place.

APRINT(N1,C1): Print the value of the variable N1 contained in contour C1 at the bottom of the screen with the message 'PRINT:' N1 'IN' C1 'IS' VAL.

DP: Delete the processor and instruction pointer.

Figure 6 contains a series of snapshots that may be displayed using the above subroutines. Snapshot (a) shows the creation of a new processor after a call to the initializing routine. This is displayed with the statements

```
CALL INCON
CALL CP('1     ','B11')
```

In snapshot (b), contour B11 is created with

```
CALL CC('B11')
```

N,R, and S are declared in this contour, so they must be displayed here. Since N is a local integer constant that is undefined upon entry to the block, both its environment and value are blank. R and S are both label variables defined in B11 at statement numbers 13 and 19 respectively.

```
CALL CNAME('N',' ',' ')
CALL CNAME('R','B11','13')
CALL CNAME('S','B11','19')
```

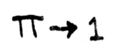
The last thing necessary in snapshot (b) is to redefine the instruction pointer with

```
CALL IP('13    ')
```

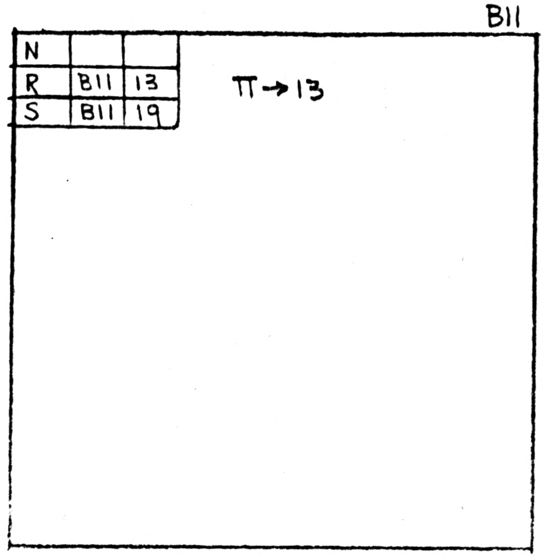
In snapshot (c), the value 2 is read in for the variable N in contour B11. The instruction pointer is changed to statement 14.

```
CALL AREAD('N','B11','2')
CALL IP('14    ')
```

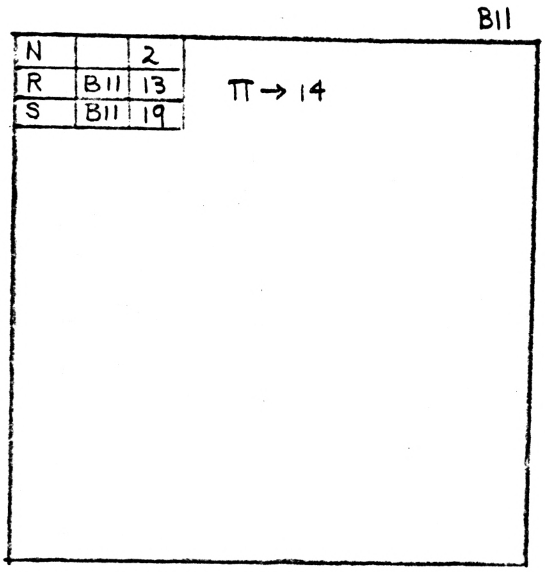
Snapshots (d-l) are similarly displayed, but snapshot (m) shows a contour deleted. While this is done in the order presented in (m) and (n), there is no way to stop the display



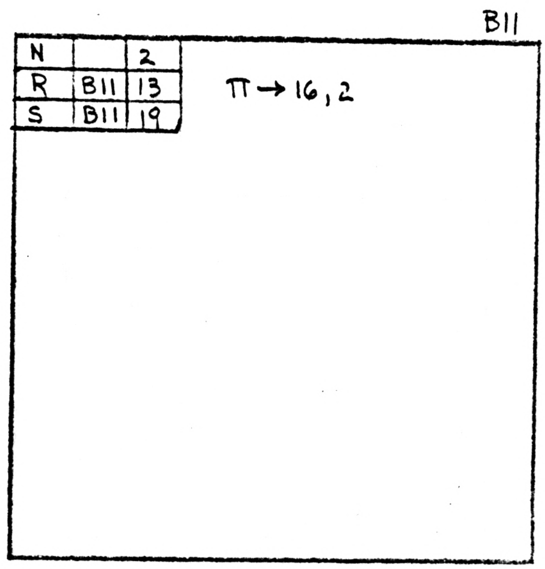
(a)



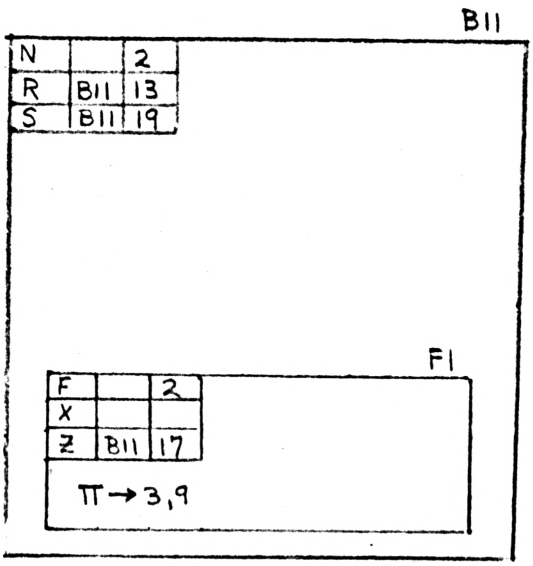
(b)



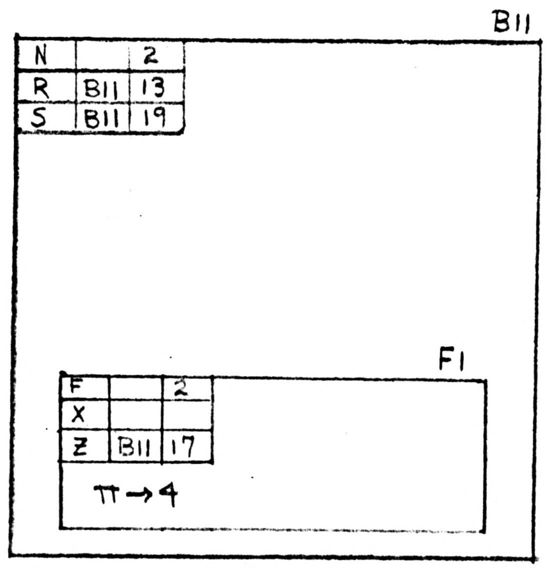
(c) READ: N=2



(d)



(e)



(f)

Figure 6

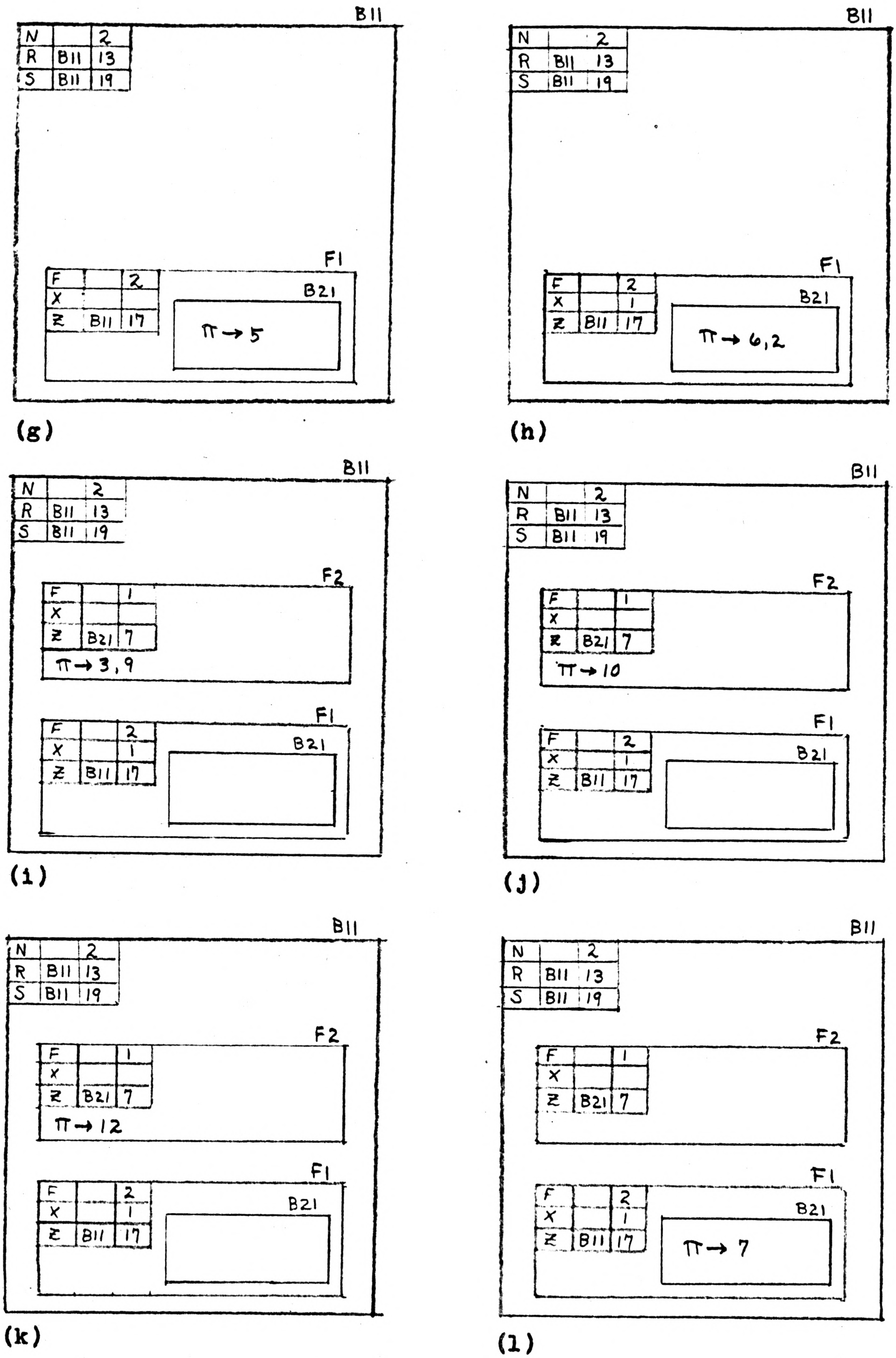


Figure 6 (cont.)

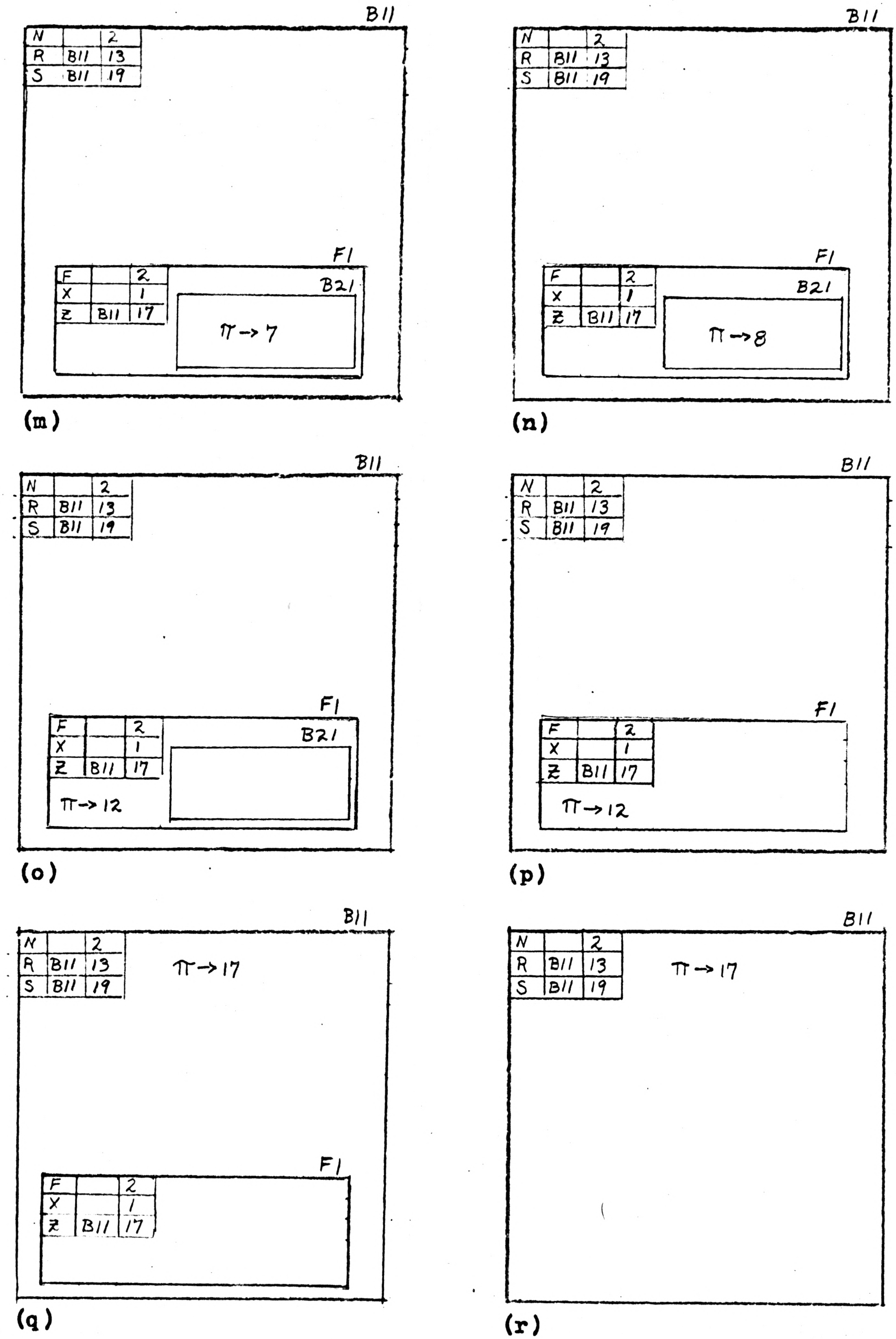


Figure 6 (cont.)

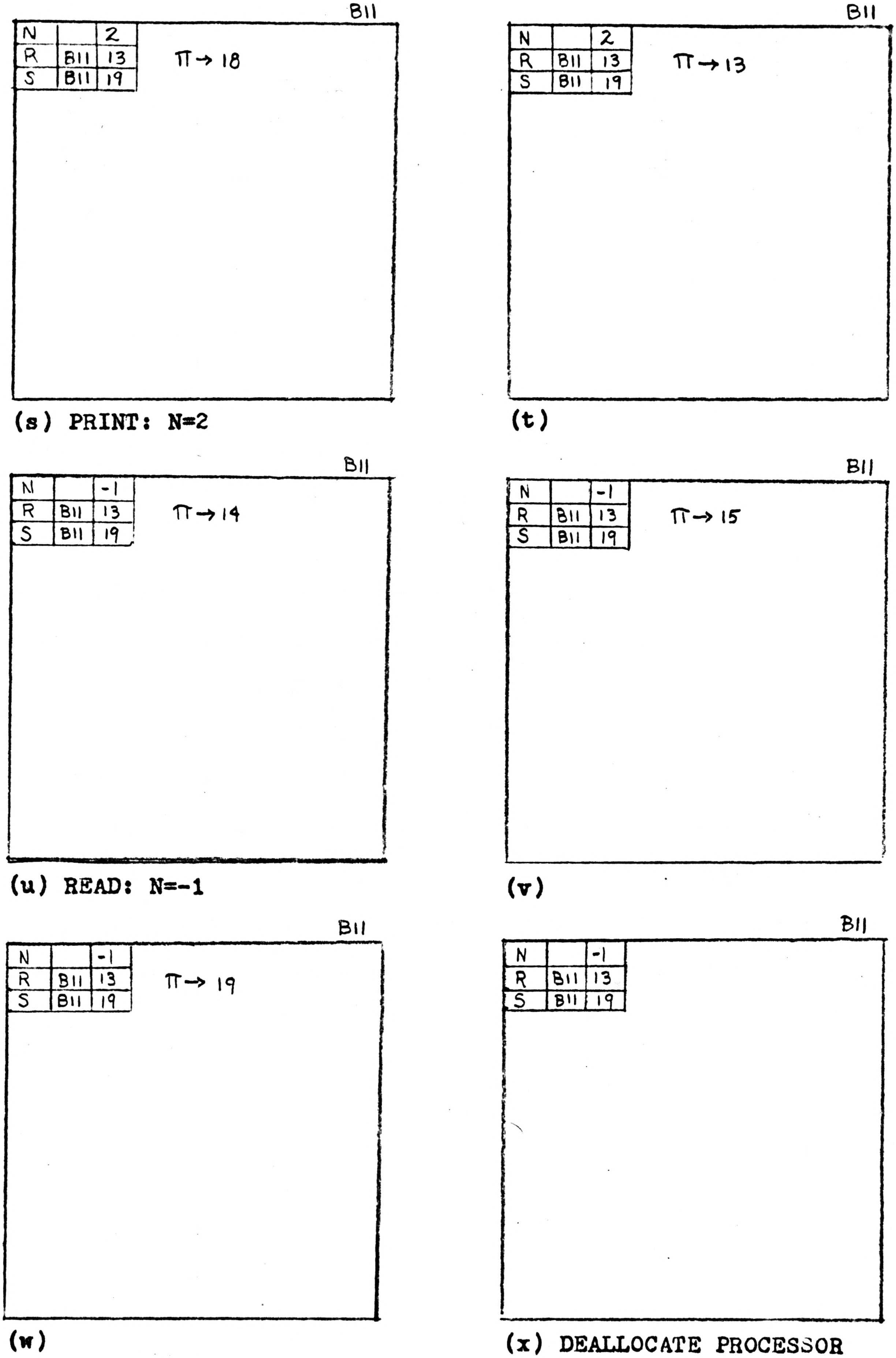


Figure 6 (cont.)

(y) Deallocate last contour

Figure 6 (cont.)

at (m) as it can be stopped only after the instruction pointer is changed. Therefore, the statements

```
CALL DC('F2')
CALL AVAL('N','B11','4')
CALL IP('8    ')
```

result in snapshot (n).

Figure 7 gives a complete listing of the driving program necessary to display the snapshots in Figure 6. The letters to the left of the call statements indicate which statements are used to generate each snapshot.

```
      CALL TINIT
      CALL TREE('B11',' ',3)
      CALL TREE('F1','B11',3)
      CALL TREE('B21','F1',0)
      CALL TREE('F2','B11',3)
      CALL DTREE('F2')
      CALL DTREE('B21')
      CALL DTREE('F1')
      CALL DTREE('B11')
      CALL SIZE
      CALL INCON
a   { CALL CP('1     ','B11')
    { CALL CC('B11')
    { CALL CNAME('N',' ',' ')
b   { CALL CNAME('R','B11','13')
    { CALL CNAME('S','B11','19')
    { CALL IP('13    ')
c   { CALL AREAD('N','B11','2')
    { CALL IP('14    ')
d   { CALL IP('16,2 ')
    { CALL CC('F1')
    { CALL CNAME('F',' ','2')
e   { CALL CNAME('X',' ', ' ')
    { CALL CNAME('Z','B11','17')
    { CALL EP('F1')
    { CALL IP('3,9  ')
```

Figure 7

```
f   { CALL IP('4    ')

    { CALL CC('B21')
g   { CALL EP('B21')
    { CALL IP('5    ')

h   { CALL AVAL('X','F1','1')
    { CALL IP('6,2  ')

    { CALL CC('F2')
    { CALL CNAME('F',' ','1')
i   { CALL CNAME('X',' ',' ')
    { CALL CNAME('Z','B21','7')
    { CALL EP('F2')
    { CALL IP('3,9  ')

j   { CALL IP('10   ')

k   { CALL IP('12   ')

l   { CALL EP('B21')
    { CALL IP('7    ')

n   { CALL DC('F2')
    { CALL IP('8    ')

o   { CALL EP('F1')
    { CALL IP('12   ')

    { CALL DC('B21')
q   { CALL EP('B11')
    { CALL IP('17   ')
```

Figure 7 (cont.)

s	CALL DC('F1') CALL APRINT('N','B11') CALL IP('18 ')
t	CALL IP('13 ')
u	CALL AREAD('N','B11','-1') CALL IP('14 ')
v	CALL IP('15 ')
w	CALL IP('19 ')
x	CALL DP
y	CALL DC('B11')

Figure 7 (cont.)

BIBLIOGRAPHY

1. Johnston, John B., "The Contour Model of Block Structured Processes", Symposium on Data Structures in Programming Languages, SIGPLAN Notices, Feb. 1971.

2. "300 Series User's Manual", Computek Inc., Cambridge, Mass., Sept 1972.

3. "Class Notes 286-830", Department of Computer Science, Kansas State University, Spring 1974.

APPENDIX I.

Common Variable Description

A. Common Blocks

```
blank  NXT,MAXLEV,NAME(20),T(10,20)    or
       MAXENT,MAXLEV,NAME(20),T(10,20)

/INDEX/VAR,LEVL,FAT,SON,BRO,SYN,DEL,PH,PW,H,W,
       X1,Y1,X2,Y2

/POINT/S(4,20),P(2,20)

/PROSCR/PX,PY,IPVAL

/SCLE/SF,DX,DY,NC,NV,P2(2,20),S1(4,20),S2(4,20)

/STEP/ISTEP,ISTAT,SCALCN

/VNAME/V(200),IVPTR
```

B. Common Variables

NOTE:

1. In all arrays, one column cooresponds to one node in the tree structure. With the current dimensions, the limit on the number of contours is 19. To change this, change the dimensions of all arrays with 20 columns so the number of columns is equal to the maximum number of contours desired plus one.

2. All variables are of default type unless specified otherwise.

3. Variables marked as not used are for future expansion of the program to provide scaling.

BRO-integer constant 4-used as an index to the brother pointer in T

DEL-integer constant 7-used as an index to the node deleted indicator in T

DX-not used

DY-not used

FAT-integer constant 3-used as an index to the father pointer in T

H-integer constant 9-used as an index to the height of the contour in T

IPVAL-character*8-the last value of the instruction pointer

ISTAT-not used(0=auto scaling mode;1=manual scaling mode)

ISTEP-number of instruction steps to simulate before halting

IVPTR-used as an index in V

LEVL-integer constant 2-used as an index in T pointing to the level number of a contour

MAXENT-after SIZE is called, it replaces NXT in the blank common-its value is the index of the maximum entry in T

MAXLEV-maximum level number used is the tree

NAME-character*4 1x20 array-table of contour names

NC-not used

NV-not used

NXT-used in the tree building routines as a pointer to the next available node in NAME and T

P-real 2x20 array-contains the X and Y positions of the processor in a given node

PH-integer constant 7-used as an index in T pointing to the Y displacement of the processor in a given contour -in the display routines,used as an index in T pointing to an index in V

PW-integer constant 8-used as an index in T pointing to the X displacement of the processor in a given contour

PX-last X position of the processor

PY-last Y position of the processor

P2-real 2x20 array-contains screen coordinates of the processor

S-real 4x20 array-contains the coordinates of the lower left and upper right corners of each contour

SCALCN-not used (name of outermost contour if in manual scaling mode)

SF-not used (scale factor)

SON-integer constant 4-used as an index in T pointing to the son pointer of a given node

SYN-integer constant 6-used as an index in T pointing to the synonym pointer of a given node

S1-real 4x20 array-contains the screen coordinates of the lower left and upper right corners of each contour

S2-real 4x20 array-not used (save the available space of a displayed contour)

T-integer 10x20 array-used to store the tree structure each column represents one node

V-character*4 1x200 array-array to save all contour variable names and values

VAR-integer constant 1-used as an index in T pointing to the number of variables in a given contour

W-integer constant 10-used as an index in T pointing to the width of a given contour

X1-integer constant 1-used as an index in various arrays to point to the first X value

X2-integer constant 3-used as an index in various arrays to point to the second X value

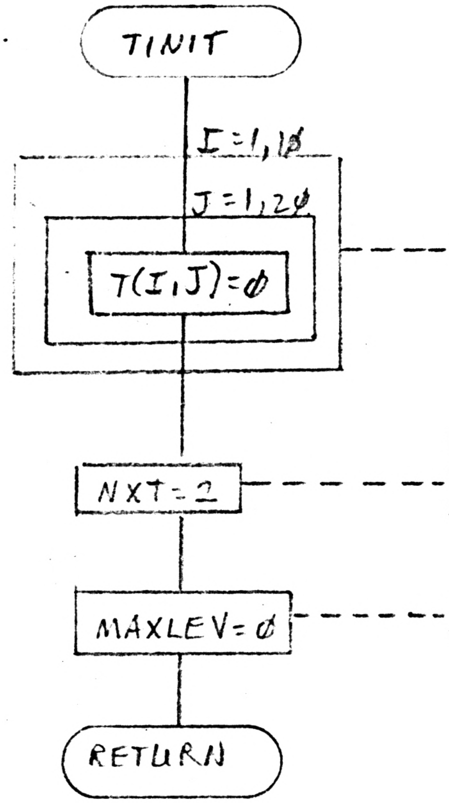
Y1-integer constant 2-used as an index in various arrays to point to the first Y value

Y2-integer constant 4-used as an index in various arrays to point to the second Y value

APPENDIX II.

Flowcharts for Selected Tree and Display Routines

TINIT-Initialize the tree builder routines.

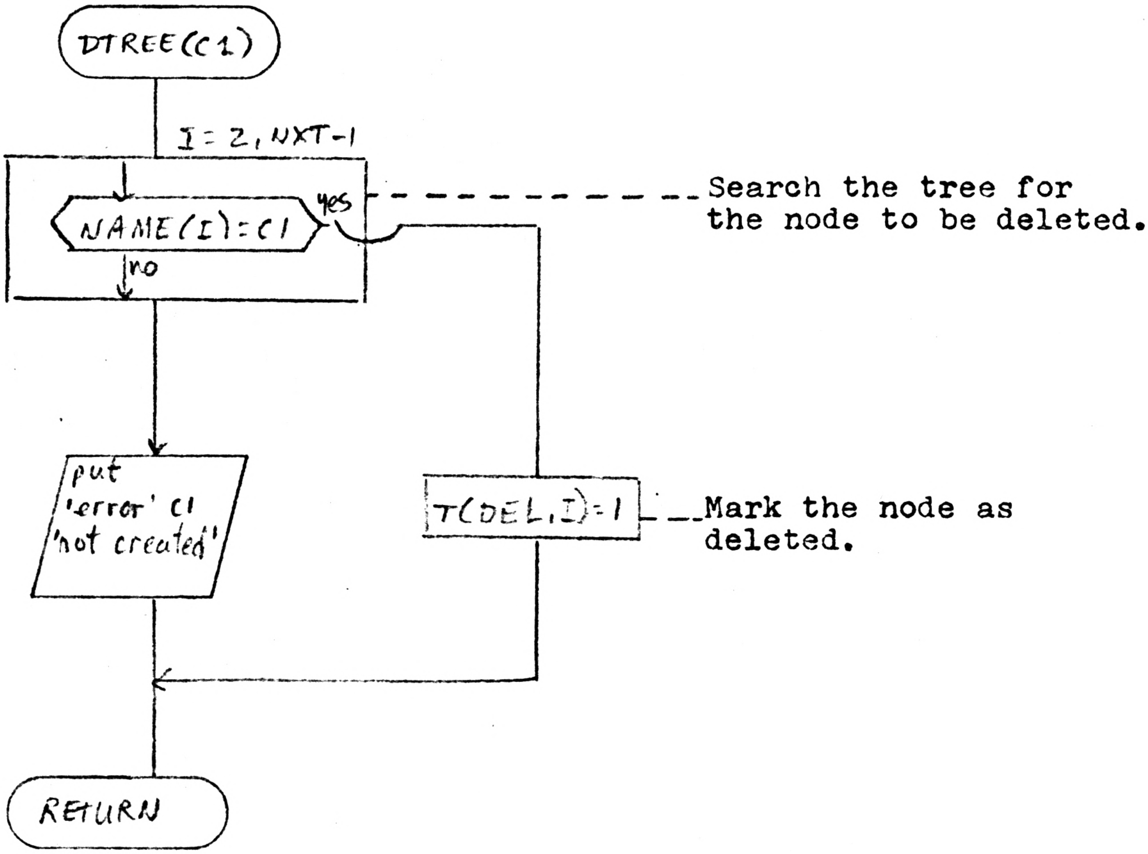


Initialize the tree structure to 0.

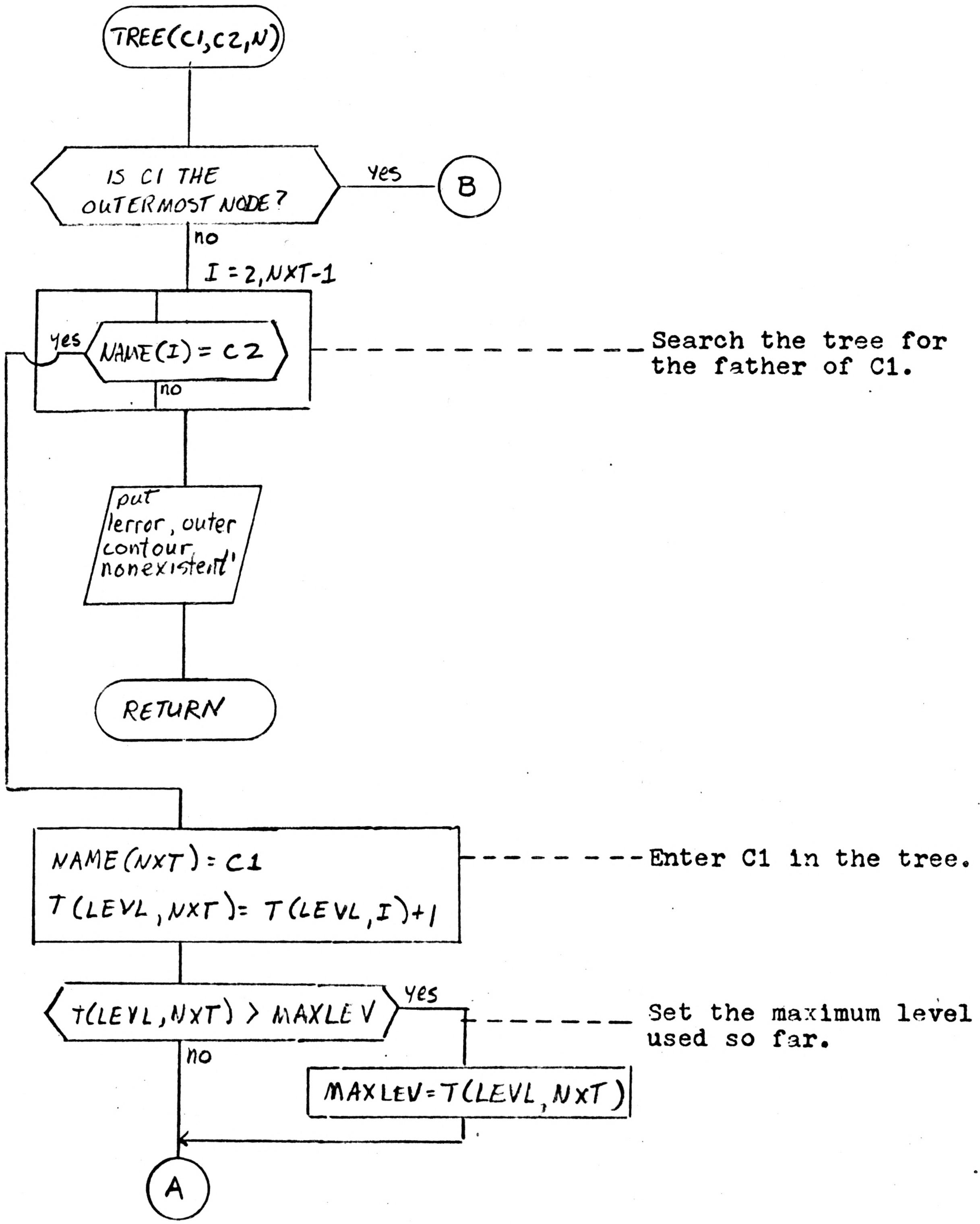
The first node in the tree is a dummy node, therefore the next available node in the tree is the second.

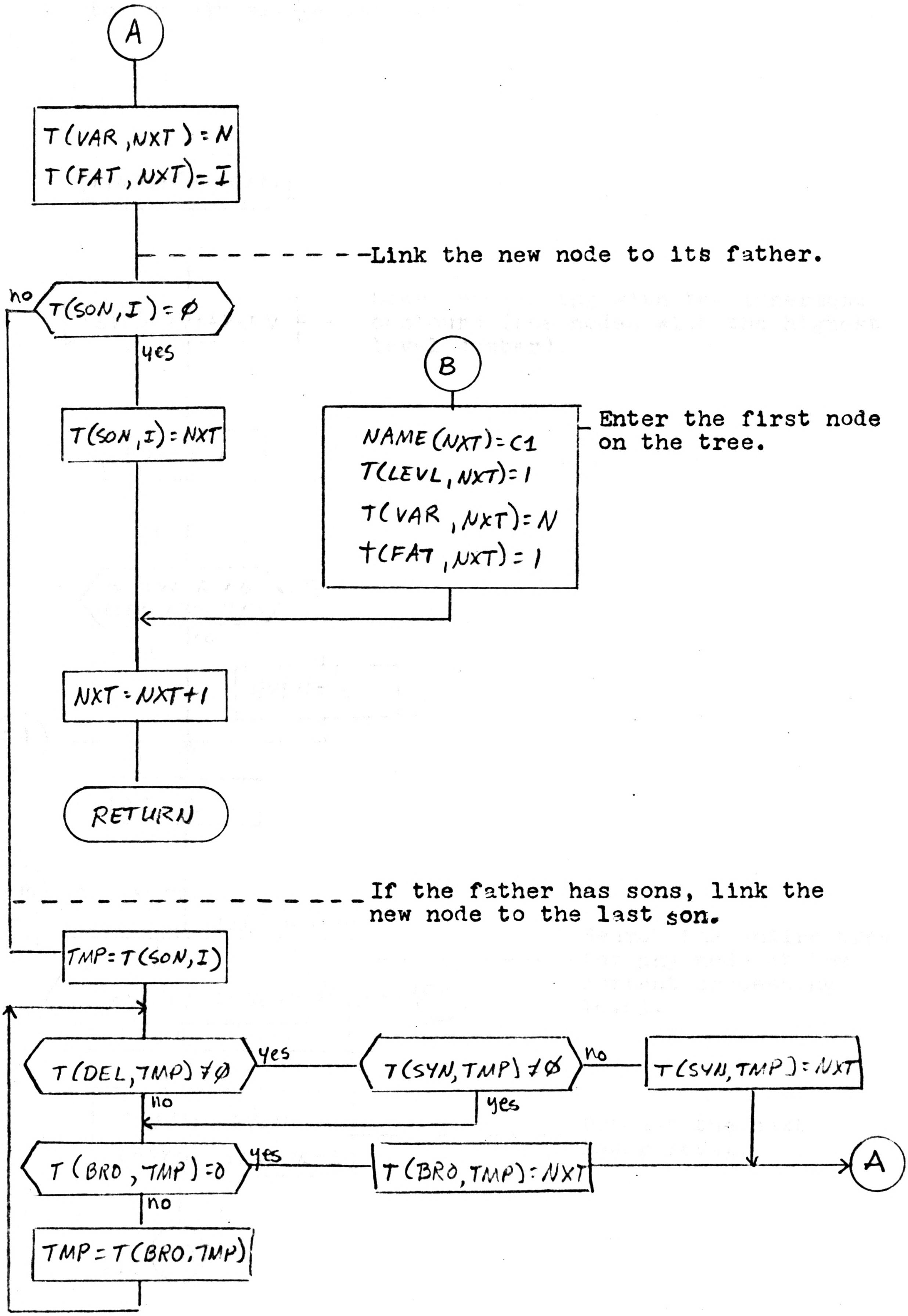
Initially, the maximum level in the tree is 0.

DTREE-Remove element C1 from the tree.

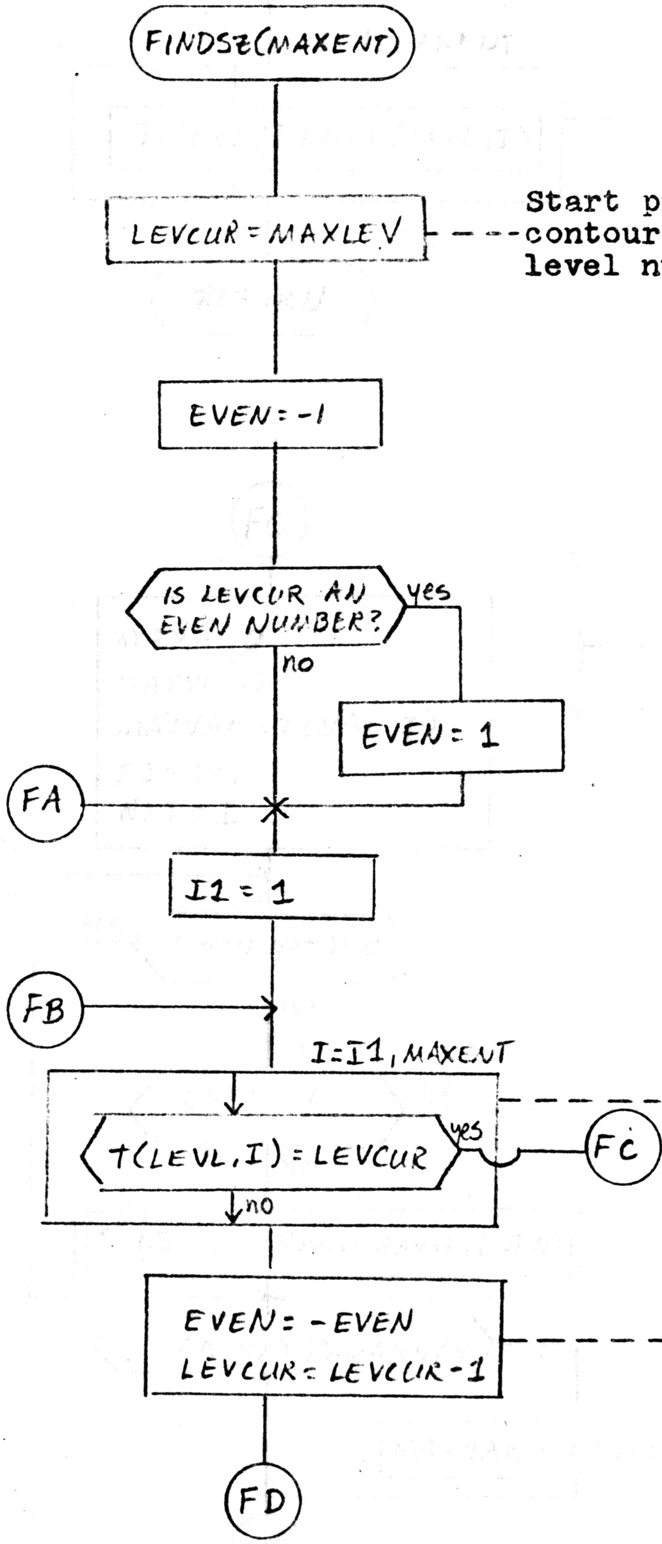


TREE-Place contour C1 on the tree as the son of C2. N is the number of symbols defined in C1 (including any return pointer necessary).




A
T(VAR,NXT)=N
T(FAT,NXT)=I
Link the new node to its father.
no
T(SON,I)=ϕ
yes
T(SON,I)=NXT
B
NAME(NXT)=C1
T(LEVL,NXT)=1
T(VAR,NXT)=N
T(FAT,NXT)=1
Enter the first node on the tree.
NXT=NXT+1
RETURN
If the father has sons, link the new node to the last son.
TMP=T(SON,I)
T(DEL,TMP)≠ϕ
yes
T(SYN,TMP)≠ϕ
no
T(SYN,TMP)=NXT
no
yes
T(BRO,TMP)=0
yes
T(BRO,TMP)=NXT
A
no
TMP=T(BRO,TMP)

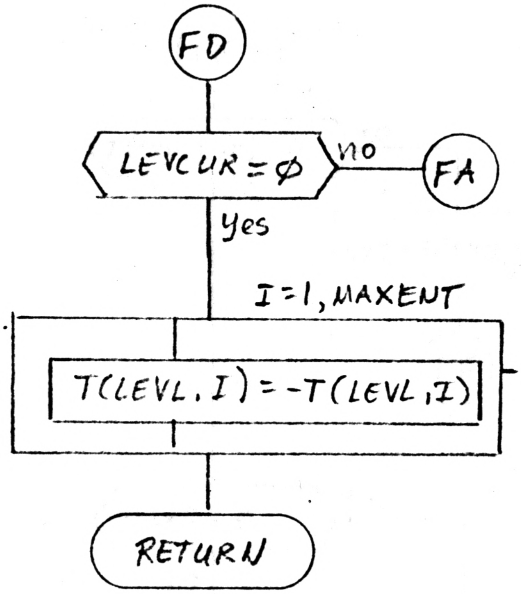
FINDSZ-Find the minimum size each contour must be in order to contain all of its sons.



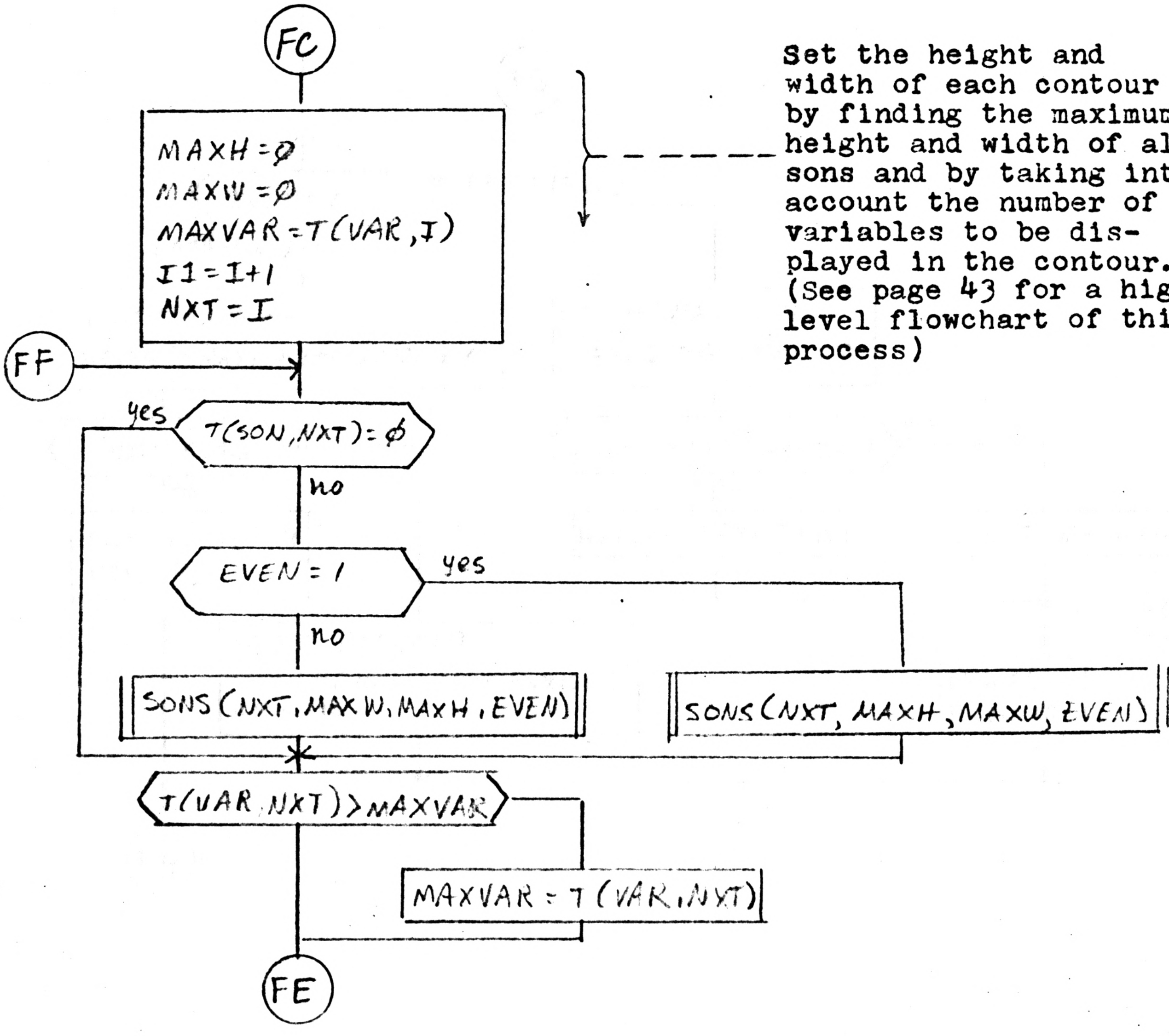
Start processing with the innermost contours (the hodes with the highest level number).

Search the entire tree for any node at the current processing level.

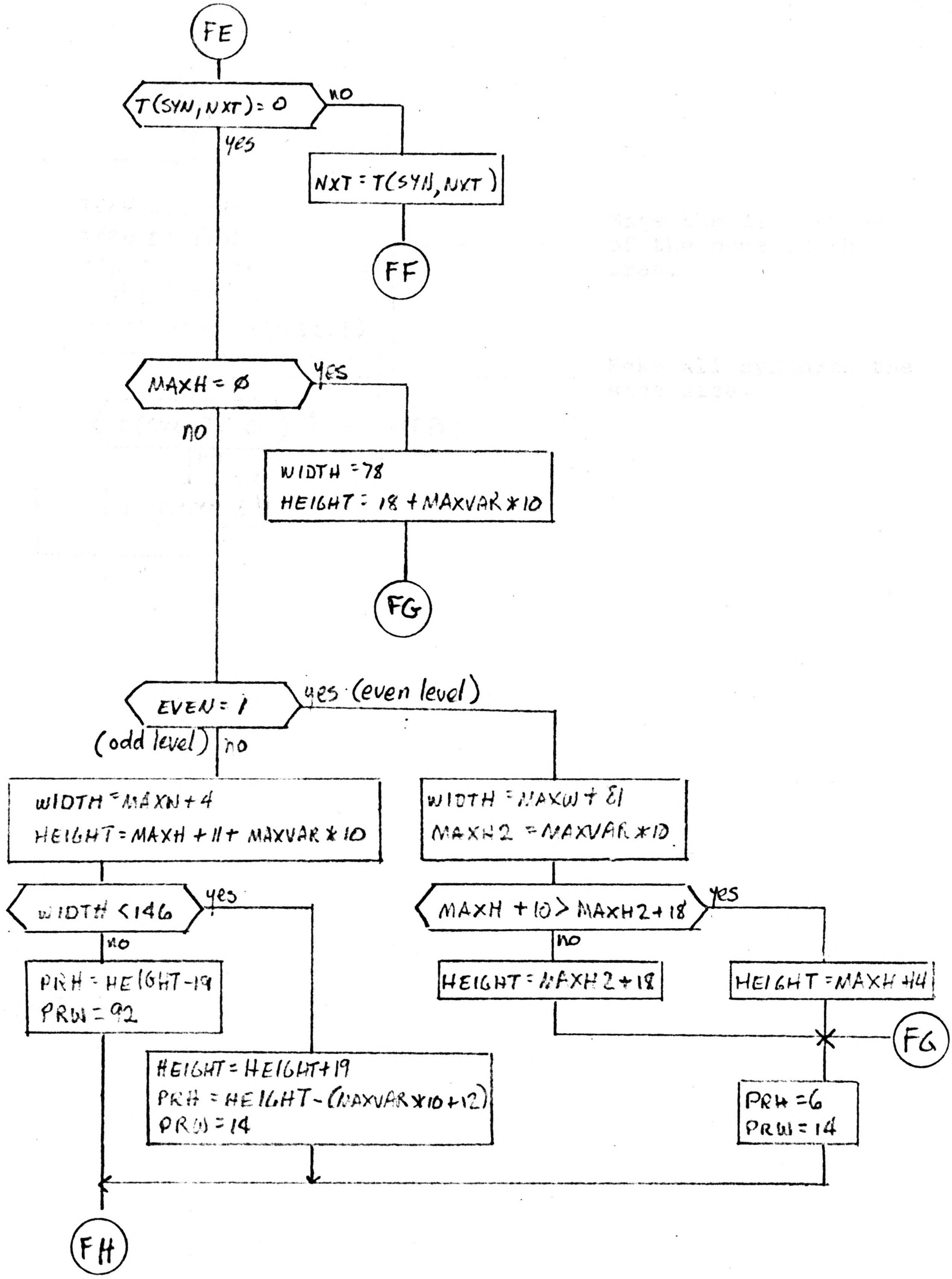
Process the next lower level.

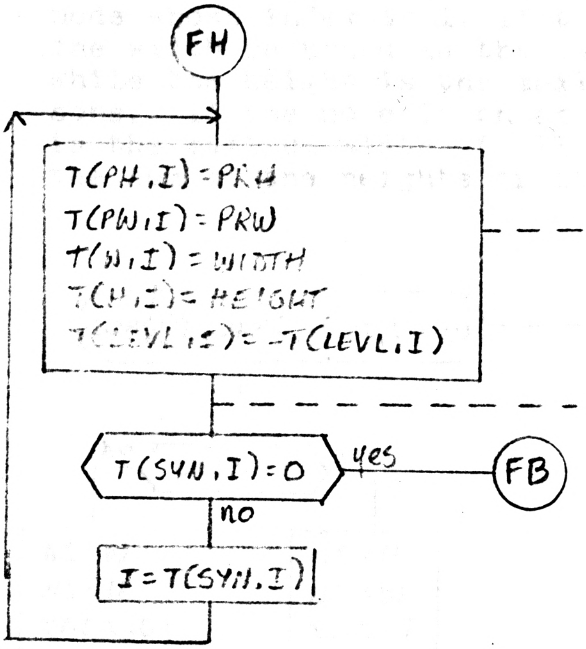


Reset the pointers that indicate a node has been processed.



Set the height and width of each contour by finding the maximum height and width of all sons and by taking into account the number of variables to be displayed in the contour. (See page 43 for a high level flowchart of this process)

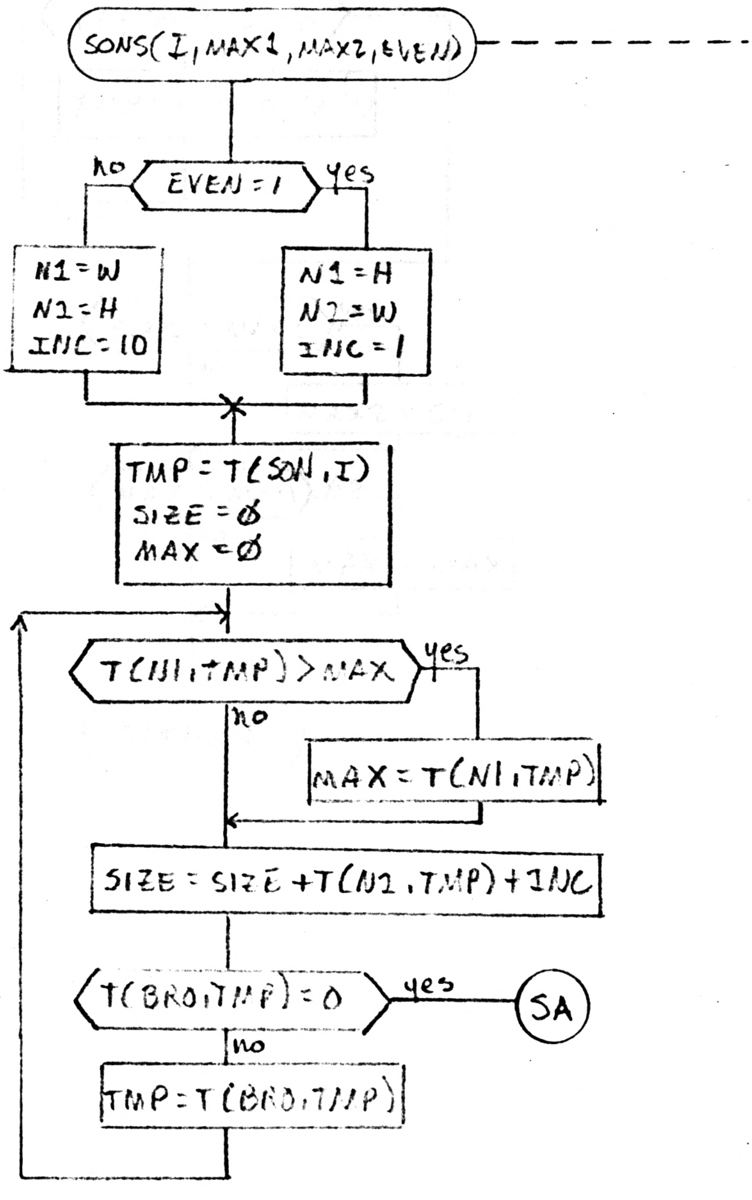

FE
T(SYN, NXT) = 0
no
yes
NXT = T(SYN, NXT)
FF
MAXH = Ø
yes
no
WIDTH = 78
HEIGHT = 18 + MAXVAR * 10
FG
EVEN = 1
yes (even level)
(odd level) no
WIDTH = MAXN + 4
HEIGHT = MAXH + 11 + MAXVAR * 10
WIDTH = MAXW + 81
MAXH2 = MAXVAR * 10
WIDTH < 146
yes
no
PRH = HEIGHT - 19
PRW = 92
HEIGHT = HEIGHT + 19
PRH = HEIGHT - (MAXVAR * 10 + 12)
PRW = 14
MAXH + 10 > MAXH2 + 18
yes
no
HEIGHT = MAXH2 + 18
HEIGHT = MAXH + 4
FG
PRH = 6
PRW = 14
FH



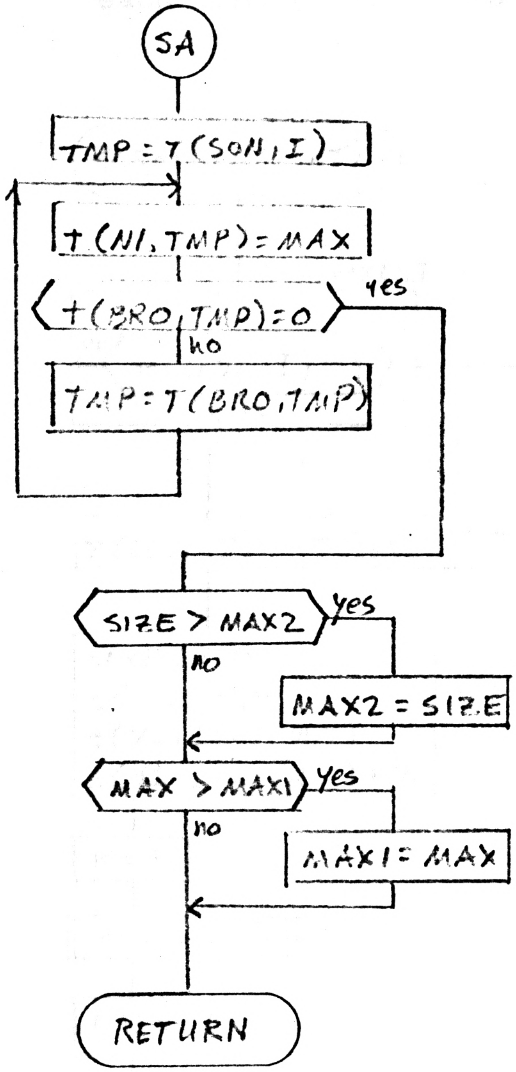
Save the dimensions of the node in the tree.

Make all synonyms the same size.

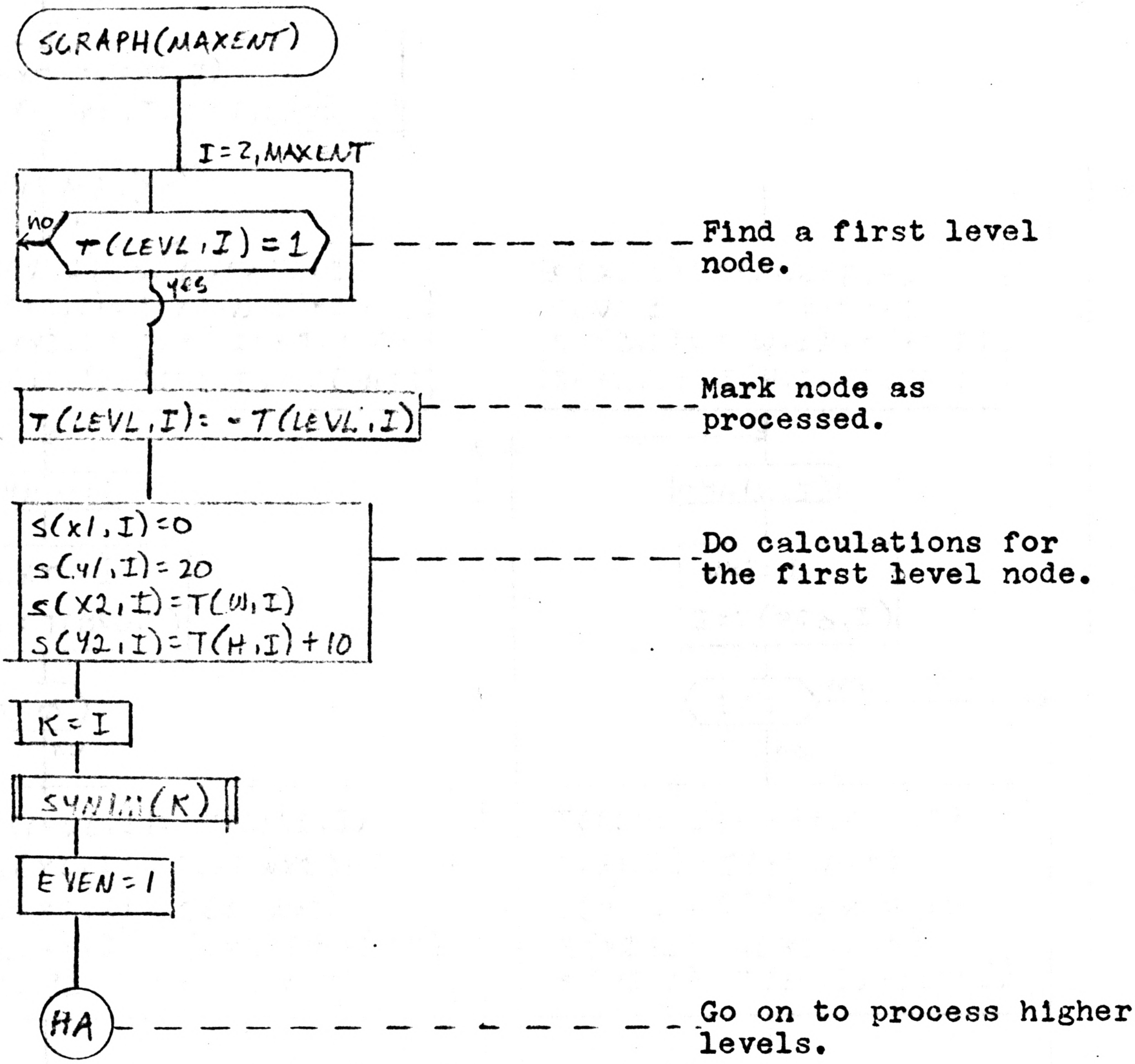
SONS-Find the maximum height and width of all sons of the node whose index is I. If the node is on an even level, the width returned is the sum of the widths of its sons while the height is the maximum height of all of its sons. If the node is on an odd level, the width returned is the maximum width of all its sons, and the height is the sum of the heights of its sons.

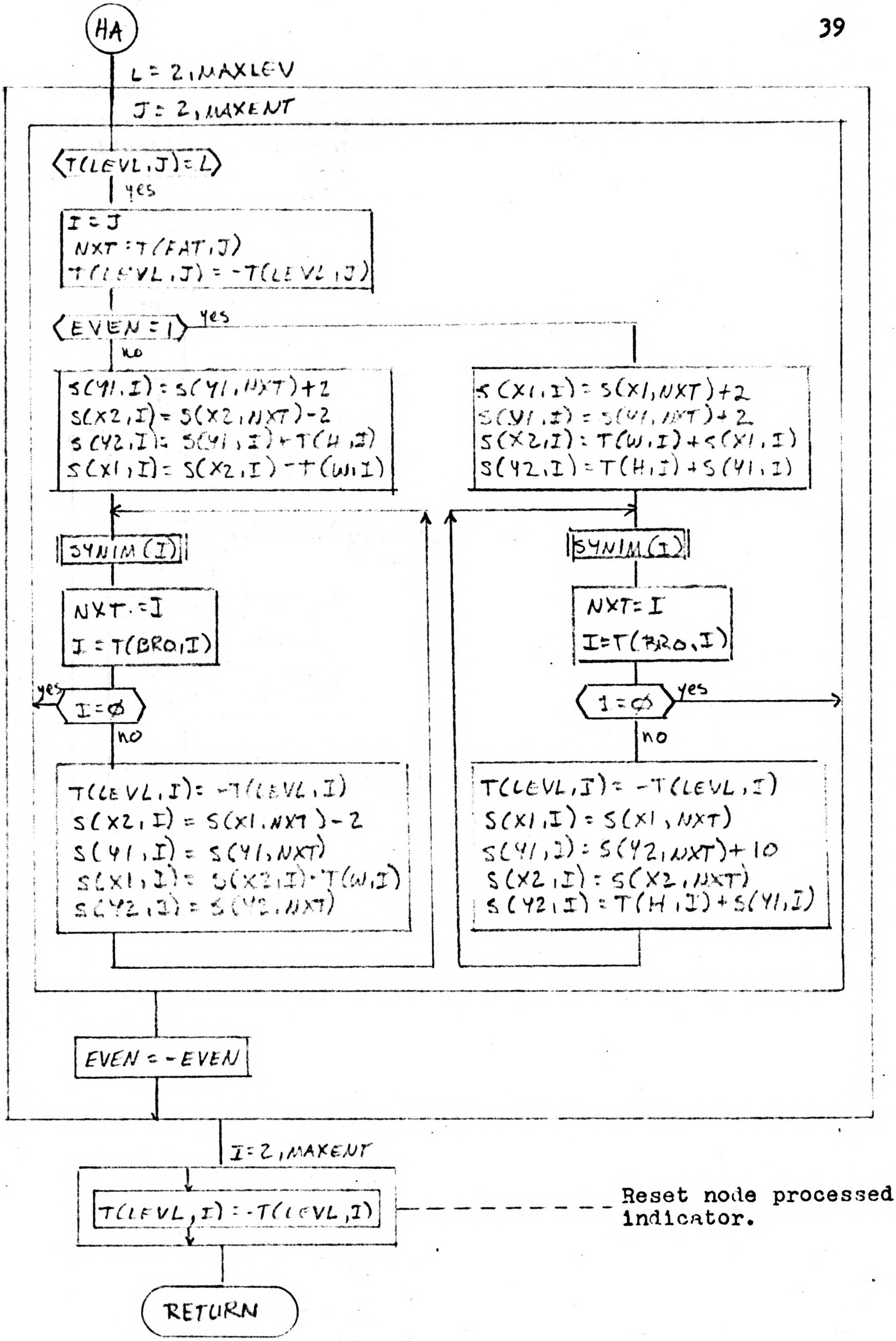


MAX1 and MAX2 are previously calculated minimum height and width for a synonym of the node I. Their values may be changed on exit from this routine.

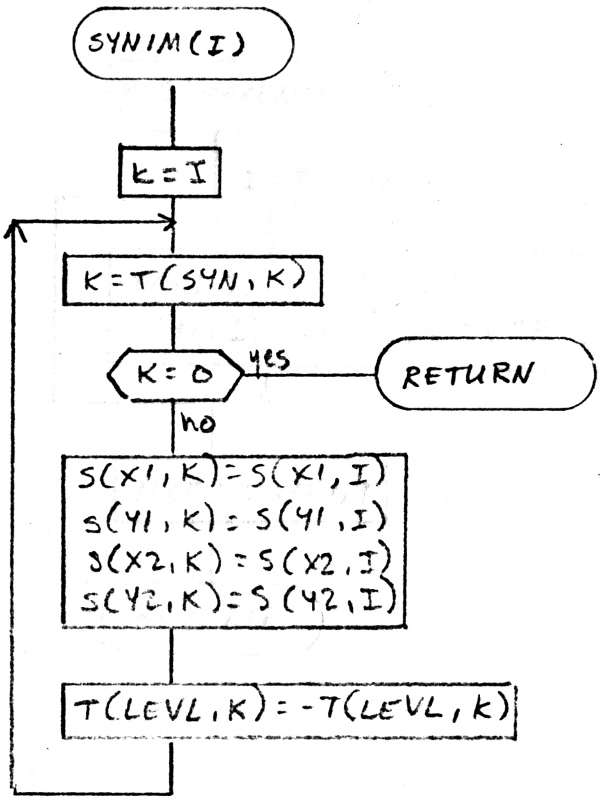

SA
TMP = T(SON, I)
T(NI, TMP) = MAX
T(BRO, TMP) = 0
yes
no
TMP = T(BRO, TMP)
SIZE > MAX2
yes
no
MAX2 = SIZE
MAX > MAX1
yes
no
MAX1 = MAX
RETURN

SGRAPH-Find the lower left and upper right coordinates of each contour in the positive quadrant.

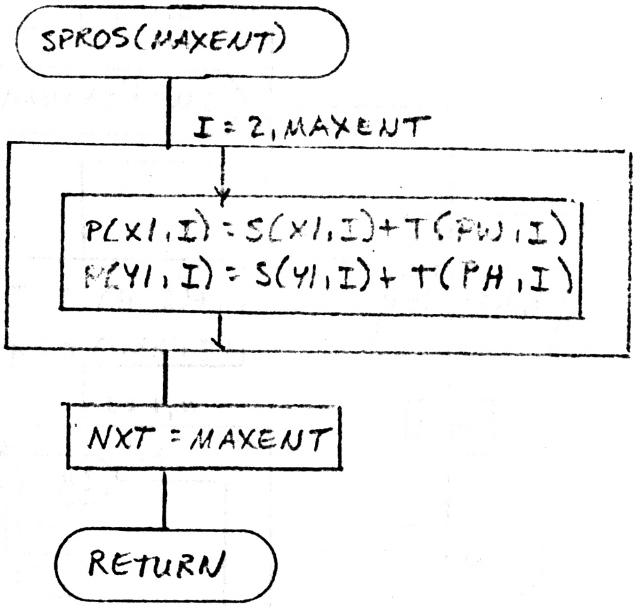



HA
L = 2, MAXLEV
J = 2, MAXENT
T(LEVL, J) = L
yes
I = J
NXT = T(FAT, J)
T(LEVL, J) = -T(LEVL, J)
EVEN = 1
yes
no
S(Y1, I) = S(Y1, NXT) + 2
S(X2, I) = S(X2, NXT) - 2
S(Y2, I) = S(Y1, I) + T(H, I)
S(X1, I) = S(X2, I) - T(W, I)
S(X1, I) = S(X1, NXT) + 2
S(Y1, I) = S(Y1, NXT) + 2
S(X2, I) = T(W, I) + S(X1, I)
S(Y2, I) = T(H, I) + S(Y1, I)
SYNIM (I)
SYNIM (I)
NXT = I
I = T(BRO, I)
NXT = I
I = T(BRO, I)
yes
I = Ø
no
I = Ø
yes
no
T(LEVL, I) = -T(LEVL, I)
S(X2, I) = S(X1, NXT) - 2
S(Y1, I) = S(Y1, NXT)
S(X1, I) = S(X2, I) - T(W, I)
S(Y2, I) = S(Y2, NXT)
T(LEVL, I) = -T(LEVL, I)
S(X1, I) = S(X1, NXT)
S(Y1, I) = S(Y2, NXT) + 10
S(X2, I) = S(X2, NXT)
S(Y2, I) = T(H, I) + S(Y1, I)
EVEN = -EVEN
I = 2, MAXENT
T(LEVL, I) = -T(LEVL, I)
Reset node processed indicator.
RETURN

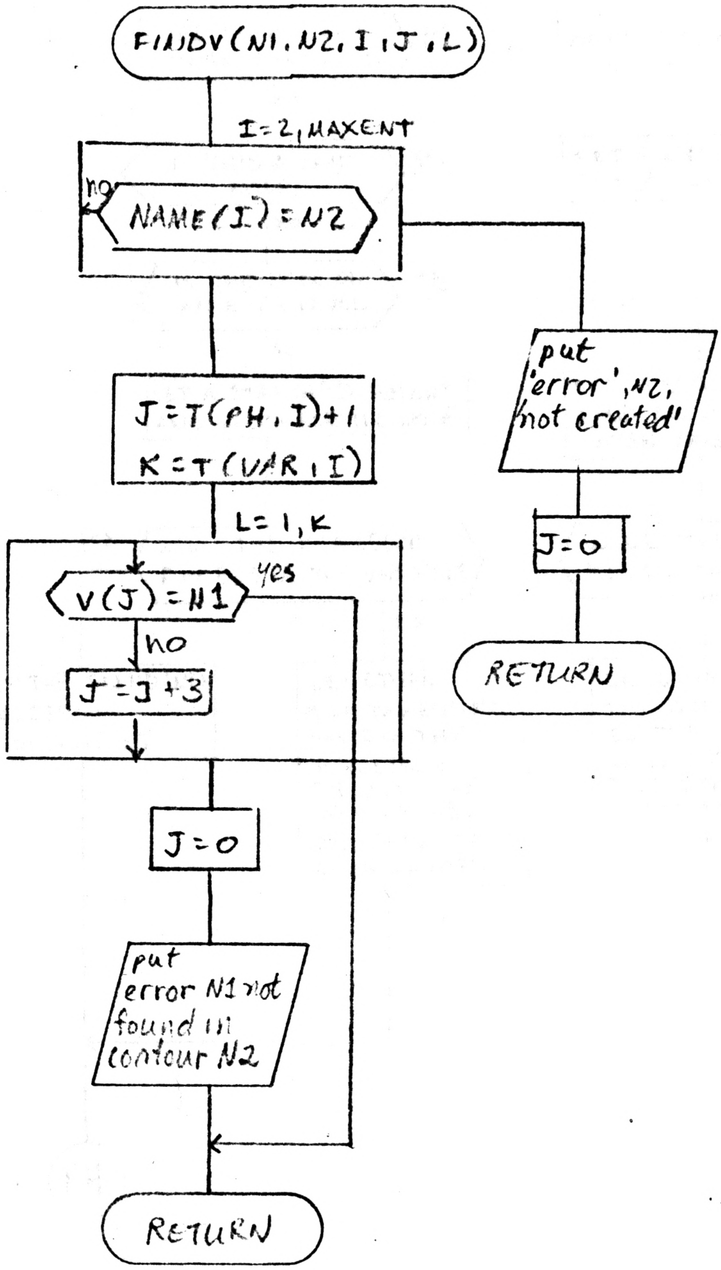
SYNIM-Set all synonyms of the node whose index is I to identical coordinates.



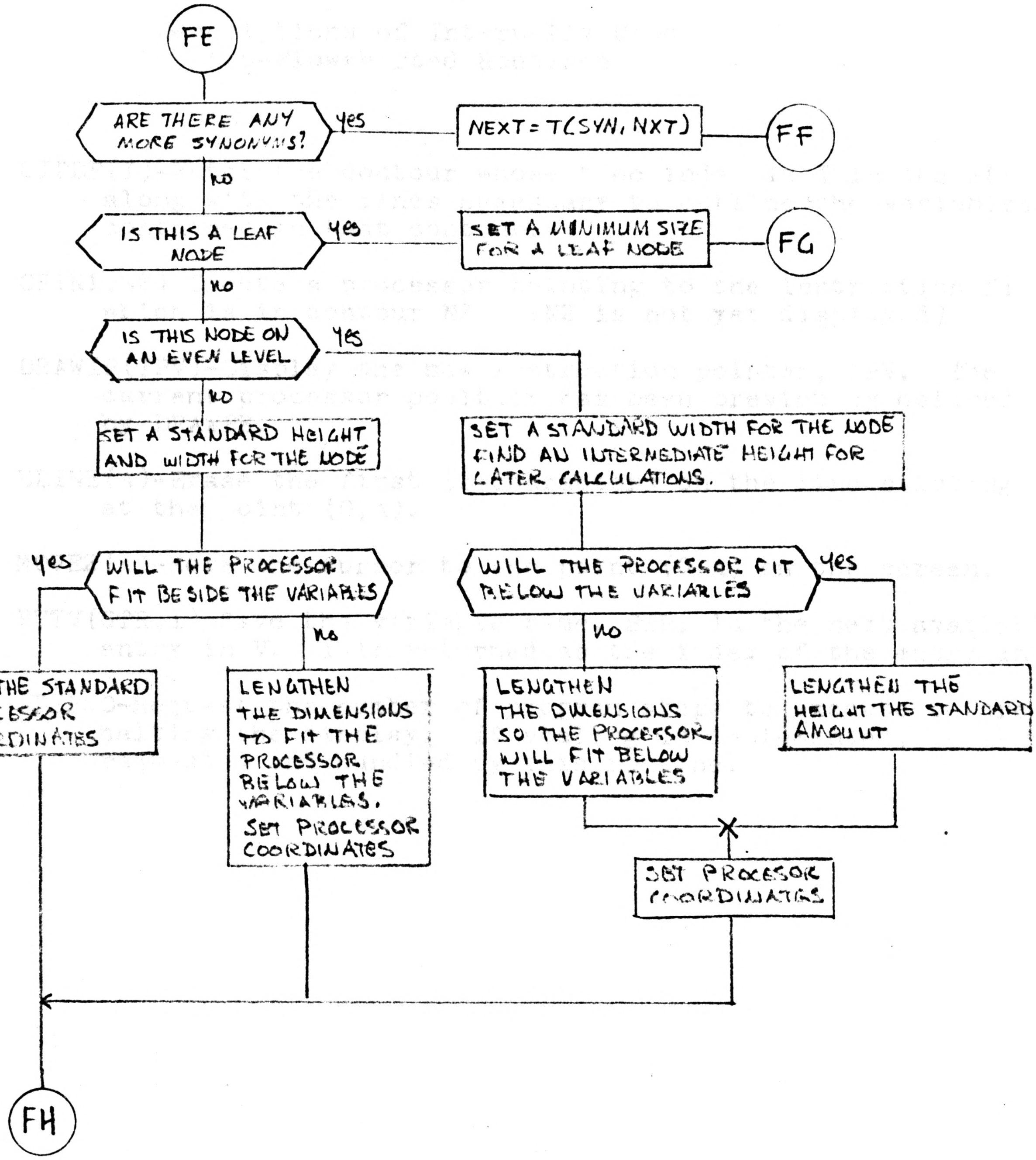
SPROS-Set the processor coordinates for all contours.



FINDV-Find the variable N1 (defined in contour N2) in the array that contains the user defined variable names,v. On return, I=is the index in NAME of N2; j is the index of N1 in V; L is the number of the variable defined in contour N2.



High Level Flowchart Description of the Calculations in FINDSZ



APPENDIX III.

Descriptions of Internally Used Non-Flowcharted Routines

CIPDF(I)-Enter the contour whose tree index is I in the PDF along with the lines necessary to outline the variables contained in that contour.

CP(N1,N2)-Create a processor pointing to the instruction N1 which is in contour N2. (N2 is not yet displayed)

DRAWIP(IPV)-Display the new instruction pointer, IPV, The current processor position has been previously defined by DRAWPR.

ELINE(A)-Erase the first 32 characters on the line starting at the point (0,A).

MOVE2(A)-Move the cursor to the point (0,A) on the screen.

PUTV(STR,I)-Save the variable name, STR, in the next available entry in V. I is returned as the index of the entry in V.

STEPNO-Request the number of program steps to execute before halting the display. If scaling is implemented, it is requested and handled by this routine.

APPENDIX IV.

Program Listings For Tree and Display Routines

```
      SUBROUTINE TINIT
C     INITIALIZE TREE BUILDER
      IMPLICIT INTEGER (A-Z)
      ASCII NAME(20)
      COMMON NXT,MAXLEV,NAME,T(10,20)
      DO 10 I=1,10
      DO 10 J=1,20
   10 T(I,J)=0
      NXT=2
       MAXLEV=0
       RETURN
       END
       SUBROUTINE DTREE(C1)
C      REMOVE ELEMENT C1 FROM THE TREE
C      BUT NOT FROM THE TABLE
C
      IMPLICIT INTEGER (A-Z)
       ASCII NAME(20),C1
       COMMON NXT,MAXLEV,NAME,T(10,20)
       COMMON/INDEX/VAR,LEVL,FAT,SON,BRO,SYN,DEL,PH,PW,H,W,X1,
&      Y1,X2,Y2
       NXT1=NXT-1
       DO 10 I=2,NXT1
       IF(NAME(I).EQ.C1) GO TO 20
   10  CONTINUE
       PRINT:"***ERROR*** ",C1,"NOT CREATED"
       RETURN
   20  T(DEL,I)=1
       RETURN
       END
C
C
C
       SUBROUTINE TREE(C1,C2,N)
C
C       PLACE NODE C1 ON THE TREE.
C       C2 IS ITS FATHER.
C       N IS THE NO OF USER SYMBOLS
C       DEFINED IN CONTOUR C1.
C
       IMPLICIT INTEGER (A-Z)
       ASCII NAME(20),C1,C2
       COMMON NXT,MAXLEV,NAME,T(10,20)
       COMMON/INDEX/VAR,LEVL,FAT,SON,BRO,SYN,DEL,PH,PW,
&      H,W,X1,Y1,X2,Y2
       IF(C2.NE."    ") GO TO 10
       NAME(NXT)=C1
       T(LEVL,NXT)=1
       T(VAR,NXT)=N
       T(FAT,NXT)=1
       GO TO 120
   10  NXT1=NXT-1
       DO 20 I=2,NXT1
```

```
      IF(NAME(I).EQ.C2) GO TO 30
 20   CONTINUE
      PRINT:"ERROR-OUTER CONTOUR NON-EXISTANT"
      RETURN
 30   NAME(NXT)=C1
      T(LEVL,NXT)=T(LEVL,I)+1
      IF(T(LEVL,NXT).GT.MAXLEV) MAXLEV=T(LEVL,NXT)
      T(VAR,NXT)=N
      T(FAT,NXT)=I
      IF(T(SON,I).EQ.0) GO TO 40
C     NODE ALREADY HAS A SON
      TMP=T(SON,I)
 50   IF(T(DEL,TMP).NE.0) GO TO 60
 70   IF(T(BRO,TMP).EQ.0) GO TO 80
      TMP=T(BRO,TMP)
      GO TO 50
 60   IF(T(SYN,TMP).NE.0) GO TO 70
      T(SYN,TMP)=NXT
      GO TO 120
 80   T(BRO,TMP)=NXT
      GO TO 120
 40   T(SON,I)=NXT
 120  NXT=NXT+1
      RETURN
      END


      SUBROUTINE FINDSZ(MAXENT)
C     FIND INTERMEDIATE SIZE OF EACH CONTOUR
      IMPLICIT INTEGER (A-Z)
      ASCII NAME(20)
      COMMON NXT,MAXLEV,NAME,T(10,20)
      COMMON/INDEX/VAR,LEVL,FAT,SON,BRO,
&      SYN,DEL,PH,PW,H,W,X1,Y1,X2,Y2
      LEVCUR=MAXLEV
      EVEN=-1
      IF(LEVCUR/2*2.EQ.LEVCUR)EVEN=1
 10   I1=1
 20   DO 30 I=I1,MAXENT
      IF(T(LEVL,I).EQ.LEVCUR) GO TO 40
 30   CONTINUE
      EVEN=-EVEN
      LEVCUR=LEVCUR-1
      IF(LEVCUR.NE.0) GO TO 10
      DO 50 I=1,MAXENT
      T(LEVL,I)=-T(LEVL,I)
 50   CONTINUE
      RETURN
 40   MAXH=0
      MAXW=0
      MAXVAR=T(VAR,I)
      I1=I+1
      NXT=I
```

```
   60 IF(T(SON,NXT).EQ.0) GO TO 80
      IF(EVEN.EQ.1) GO TO 70
C     WORKING ON ODD LEVEL
      CALL SONS(NXT,MAXW,MAXH,EVEN)
      GO TO 80
C     WORKING ON EVEN LEVEL
   70 CALL SONS(NXT,MAXH,MAXW,EVEN)
   80 IF(T(VAR,NXT).LE.MAXVAR) GO TO 90
      MAXVAR=T(VAR,NXT)
   90 IF(T(SYN,NXT).EQ.0) GO TO 100
      NXT=T(SYN,NXT)
      GO TO 60
C     CALCULATE SIZE OF NODE AND ALL SYNONYMS
  100 IF(MAXH.EQ.0) GO TO 110
      IF(EVEN.EQ.1) GO TO 120
C     ODD LEVEL NO
      WIDTH=MAXW+4
      HEIGHT=MAXH+11+MAXVAR*10
      IF(WIDTH.LT.146) GO TO 130
      PRH=HEIGHT-19
      PRW=92
      GO TO 150
  130 HEIGHT=HEIGHT+19
      PRH=HEIGHT-(MAXVAR*10+12)
      PRW=14
      GO TO 150
C     EVEN LEVEL NO
  120 WIDTH=MAXW+81
      MAXH2=MAXVAR*10
C     DECIDE WHERE PROCESSOR SHOULD GO
      IF((MAXH+10).GT.(MAXH2+18)) GO TO 140
      HEIGHT=MAXH2+18
      GO TO 115
  140 HEIGHT=MAXH+14
      GO TO 115
C     LEAF NODE FOUND
  110 WIDTH=78
      HEIGHT=18+MAXVAR*10
  115 PRH=6
      PRW=14
  150 T(PH,I)=PRH
      T(PW,I)=PRW
      T(W,I)=WIDTH
      T(H,I)=HEIGHT
      T(LEVL,I)=-T(LEVL,I)
      IF(T(SYN,I).EQ.0)GO TO 20
      I=T(SYN,I)
      GO TO 150
      END
```

```
      SUBROUTINE SONS(I,MAX1,MAX2,EVEN)
      IMPLICIT INTEGER (A-Z)
      ASCII NAME(20)
      COMMON NXT,MAXLEV,NAME,T(10,20)
     COMMON/INDEX/VAR,LEVL,FAT,
&     SON,BRO,SYN,DEL,PH,PW,H,W,
&     X1,Y1,X2,Y2
      IF(EVEN.EQ.1) GO TO 10
      N1=W
      N2=H
     INC=10
      GO TO 20
10    N1=H
      N2=W
      INC=1
20    TMP=T(SON,I)
      SIZE=0
      MAX=0
30    IF(T(N1,TMP).GT.MAX)MAX=T(N1,TMP)
      SIZE=SIZE+T(N2,TMP)+INC
      IF(T(BRO,TMP).EQ.0) GO TO 40
      TMP=T(BRO,TMP)
      GO TO 30
C     NOW SET HEIGHTS(ODD LEVEL) OR WIDTHS(EVEN LEVEL)
C     TO MAX HEIGHT OR WIDTH
40    TMP=T(SON,I)
50   T(N1,TMP)=MAX
      IF(T(BRO,TMP).EQ.0) GO TO 60
      TMP=T(BRO,TMP)
      GO TO 50
60    IF(SIZE.GT.MAX2)MAX2=SIZE
      IF(MAX.GT.MAX1) MAX1=MAX
      RETURN
      END
      BLOCK DATA
      IMPLICIT INTEGER (A-Z)
      COMMON/INDEX/VAR,LEVL,FAT,SON,BRO,SYN,DEL,
&     PH,PW,H,W,X1,Y1,X2,Y2
      DATA VAR/1/,LEVL/2/,FAT/3/,SON/4/,BRO/5/,SYN/6/,DEL/7/,
&     PH/7/,PW/8/,H/9/,W/10/,X1/1/,Y1/2/,X2/3/,Y2/4/
      END
```

```
      SUBROUTINE SGRAPH(MAXENT)
C
C FIND COORDINATES OF THE LOWER LEFT
C (X1,Y1) AND UPPER RIGHT (X2,Y2)
C POINT OF EACH CONTOUR IN THE
C POSITIVE QUADRANT. THE 1ST 20
C Y POSITIONS ARE SAVED FOR MESSAGES
C ON THE SCREEN
C
      IMPLICIT INTEGER (A-Z)
      REAL S(4,20),P(2,20)
      COMMON NEW,MAXLEV,NAME(20),
     & T(10,20)
      COMMON /POINT/S,P
      COMMON/INDEX/VAR,LEVL,FAT,SON,
     & BRO,SYN,DEL,PH,PW,H,W,X1,
     & Y1,X2,Y2
C     SET OUTERMOST CONTOUR
      DO 10 I=2,MAXENT
      IF(T(LEVL,I).NE.1) GO TO 10
      T(LEVL,I)=-T(LEVL,I)
      S(X1,I)=0.
      S(Y1,I)=20.
      S(X2,I)=T(W,I)
      S(Y2,I)=T(H,I)+10
      MAXX=S(X2,I)
      MAXY=S(Y2,I)
      K=I
      CALL SYNIN(K)
      GO TO 20
   10 CONTINUE
C     START ON EVEN LEVEL
   20 EVEN=1
      DO 30 L=2,MAXLEV
      DO 40 J=2,MAXENT
      IF(T(LEVL,J).NE.L) GO TO 40
      I=J
      NXT=T(FAT,J)
C     MARK NODE AS PROCESSED
      T(LEVL,J)=-T(LEVL,J)
      IF(EVEN.EQ.1) GO TO 50
C     ODD LEVEL
      S(Y1,I)=S(Y1,NXT)+2
      S(X2,I)=S(X2,NXT)-2
      S(Y2,I)=S(Y1,I)+T(H,I)
      S(X1,I)=S(X2,I)-T(W,I)
   60 CALL SYNIN(I)
      NXT=I
      I=T(BRO,I)
      IF(I.EQ.0) GO TO 40
      T(LEVL,I)=-T(LEVL,I)
      S(X2,I)=S(X1,NXT)-2
      S(Y1,I)=S(Y1,NXT)
```

```
      S(X1,I)=S(X2,I)-T(W,I)
      S(Y2,I)=S(Y2,NXT)
      GO TO 6Ø
C     EVEN LEVEL
   5Ø S(X1,I)=S(X1,NXT)+2
      S(Y1,I)=S(Y1,NXT)+2
      S(X2,I)=T(W,I)+S(X1,I)
      S(Y2,I)=T(H,I)+S(Y1,I)
   7Ø CALL SYNIM(I)
      NXT=I
      I=T(BRO,I)
      IF(I.EQ.Ø) GO TO 4Ø
      T(LEVL,I)=-T(LEVL,I)
      S(X1,I)=S(X1,NXT)
      S(Y1,I)=S(Y2,NXT)+1Ø
      S(X2,I)=S(X2,NXT)
      S(Y2,I)=T(H,I)+S(Y1,I)
      GO TO 7Ø
   4Ø CONTINUE
C     SWITCH LEVEL NO
      EVEN=-EVEN
   3Ø CONTINUE
C     SET ALL LEVELS BACK TO POSITIVE NUMBERS
      DO 8Ø I=2,MAXENT
   8Ø T(LEVL,I)=-T(LEVL,I)
      RETURN
      END
C
C
C
      SUBROUTINE SYNIM(I)
C
C     SET ALL SYNONYMS OF I TO
C     IDENTICAL COORDINATES
C
      IMPLICIT INTEGER (A-Z)
      REAL S(4,2Ø),P(2,2Ø)
      COMMON NXT,MAXLEV,NAME(2Ø),T(1Ø,2Ø)
      COMMON/POINT/S,P
      COMMON/INDEX/VAR,LEVL,FAT,SON,
&      BRO,SYN,DEL,PH,PW,H,W,
&      X1,Y1,X2,Y2
      K=I
   1Ø K=T(SYN,K)
      IF(K.EQ.Ø) RETURN
      S(X1,K)=S(X1,I)
       S(Y1,K)=S(Y1,I)
       S(X2,K)=S(X2,I)
       S(Y2,K)=S(Y2,I)
C      MARKØNODE AS PROCESSED
       T(LEVL,K)=-T(LEVL,K)
       GO TO 1Ø
       END
```

```
C
C
C
      SUBROUTINE SPROS(MAXENT)
C
C     FIND PROCESSOR COORDINATES
C     CORRESPONDING TO THE
C     CONTOUR COORDIAATES FOUND BY
C     SGRAPH
C
      IMPLICIT INTEGER (A-Z)
      REAL S(4,20),P(2,20)
      COMMON NXT,MAXLEV,NAME(20),T(10,20)
      COMMON /POINT/S,P
      COMMON/INDEX/VAR,LEVL,FAT,SON,
&      BRO,SYN,DEL,PH,PW,H,W,X1,Y1,
&      X2,Y2
      DO 10 I=2,MAXENT
      P(X1,I)=S(X1,I)+T(PW,I)
   10 P(Y1,I)=S(Y1,I)+T(PH,I)
      NXT=MAXENT
      RETURN
      END
```

```
      SUBROUTINE FINDV(N1,N2,I,J,L)
C
C     FIND VARIABLE NAME IN V
C     N1=VAR NAME
C     N2=CONTOUR NAME
C     I=INDEX OF NAME IN NAME
C     J=RETURN INDEX IN V
C     L=NO. OF THE VARIABLE N1
C
      INTEGER T,PH,VAR
      ASCII N1,N2,NAME,V
      COMMON MAXENT,MAXLEV,NAME(20),T(10,20)
      COMMON/INDEX/VAR,DUM(6),PH
      COMMON/VNAME/V(200),IVPTR
C     FIND CONTOUR IN TABLE
      DO 10 I=2,MAXENT
      IF(NAME(I).EQ.N2) GO TO 20
   10 CONTINUE
      PRINT:"***ERROR***",N2," NOT CREATED"
      J=0
      RETURN
C     FIND VARIABLE IN V
C     J=START INDEX
C     K=NO OF VAR
   20 J=T(PH,I)+1
      K=T(VAR,I)
```

```
      DO 30 L=1,M
      IF(V(J).EQ.N1) GO TO 40
   30 J=J+3
      J=0
      PRINT:"***ERROR*** ",N1," NOT FOUND IN CONTOUR ",N2
   40 RETURN
C     FOUND VARIABLE NAME SO RETURN
      END
C
C
C
      SUBROUTINE AENV(N1,N2,N3)
C     CHANGE ENVIRONMENT OF VARIABLE
C     N1 TO N3. N2 IS THE STATIC
C     ENVIRONMENT.
      ASCII N1,N2,N3,V
      INTEGER T,X1,Y1,X2,Y2
      COMMON MAXENT,MAXLEV,NAME(20),T(10,20)
      COMMON/SCLE/D(5),P2(2,20),S(4,20),S2(4,20)
      COMMON/INDEX/NUM(11),X1,Y1,X2,Y2
      COMMON/VNAME/V(200),IVPTR
      ADD=26
      ID=1
      GO TO 10
C
C
C
      ENTRY AVAL(N1,N2,N3)
C     CHANGE VALUE OF THE VARIABLE N1
C     TO N3.
      ADD=52
      ID=2
C     FIND N1 IN NAME ARRAY
   10 CALL FINDV(N1,N2,I,J,M)
C     IF N1 IS NOT FOUND,RETURN
      IF(J.EQ.0) RETURN
      INDX=J+ID
C     FIND POINT OF DISPLAY ON SCREEN
      CALL START
      SX=S(X1,I)+2+ADD
      SY=S(Y2,I)-M*10+2
C     ERASE PREVIOUS ENVIRONMENT
      CALL MOVE(SX-1.,SY,0.)
      CALL EMODE
      CALL HTEXT(4,"    ")
      CALL WMODE
C     DSPLY NEW ENVIRONMENT
      CALL MOVE(SX,SY,0)
      CALL HTEXT(4,N3)
C     SAVE NEW ENVIRONMENT IN V
      V(INDX)=N3
```

```
C     DISPLAY AND RETURN
      CALL MOVE(0,0,0)
      CALL SENDPDF
      CALL COMPIL(1,1,LEN)
      RETURN
      END
C
C.
C
      SUBROUTINE EP(N1)
C
C     CHANGE VALUE OF EP TO N1
C
      INTEGER X1,Y1
      CHARACTER*8 IPVAL
      ASCII N1,NAME
      COMMON MAXENT,MAXLEV,NAME(20),T(10,20)
      COMMON/PROCSR/PX,PY,IPVAL
      COMMON/SCLE/D(5),P(2,20),S1(4,20),S2(4,20)
      COMMON/INDEX/DUM(11),X1,Y1
C     FIND CONTOUR
      DO 10 I=2,MAXENT
       IF(NAME(I).EQ.N1) GO TO 20
 10    CONTINUE
       PRINT:"***ERROR*** ",N1," IS INVALID EP"
       RETURN
C      ERASE OLD PROCESSOR
 20    CALL START
       CALL EMODE
       CALL MOVE(PX,PY,1.)
       CALL HTEXT(10,'          ')
       CALL VMODE
C      PUT NEW PROCESSOR IN CORRECT
C      ENVIRONMENT
       CALL DRAWPR(P(X1,I),P(Y1,I))
C
C      IF CONTOUR TOO SMALL TO
C      DISPLAY THE PROCESSOR, PRINT EP
C
       IF(P(Y1,I).GT.10.) GO TO 30
       CALL MOVE(196.,0.,0.)
       CALL HTEXT(3,'EP=')
       CALL MOVE(220.,0.,0.)
       CALL HTEXT(4,N1)
 30    CALL SENDPDF
       CALL COMPIL(1,1,LEN)
       RETURN
       END
```

```
C
C
C
      SUBROUTINE IP(N1)
C
C     CHANGE THE INSTRUCTION POINTER
C
      INTEGER TXTIND
      CHARACTER*8 IPVAL,N1
      COMMON/GRAF/INDEX,PDF(4,400),TXTIND(2,50)
      COMMON/STEP/ISTEP,ISTAT,SCALCN
      COMMON/PROCSR/PX,PY,IPVAL
C     ERASE OLD VALUE
      CALL START
      CALL EMODE
      CALL DRAWIP("        ")
C     PRINT NEW VALUE
      CALL WMODE
      CALL DRAWIP(N1)
      CALL SENDPDF
      CALL COMPIL(1,1,LEN)
C
C     RESET TXTIND POINTERS
C
      TXTIND(1,1)=2
      TXTIND(2,1)=1
      IF(ISTEP.EQ.0) CALL STEPNO
      ISTEP=ISTEP-1
      RETURN
      END
C
C
C
      SUBROUTINE DRAWIP(IPV)
C
C     DISPLAY IP FOR CURRENT PROCESSOR
C
      CHARACTER*8 IPV,IPVAL
      COMMON/PROCSR/PX,PY,IPVAL
      IPVAL=IPV
      CALL MOVE(PX+12.5,PY,1)
      CALL HTEXT(8,IPVAL)
      RETURN
      END
```

```
C
C
C
      SUBROUTINE DRAWPR(XP,YP)
C
C     DRAW PROCESSOR AT POINT(XP,Y6P6YP)
C
      CHARACTER*8 IPVAL
      COMMON/PROCSP/PX,PY,IPVAL
      PX=XP
      PY=YP
      YP6=YP+6
      CALL MOVE(XP,YP6,1)
      CALL VEC(XP+4,YP6,1)
      CALL MOVE(XP+1,YP6,1)
      CALL VEC(XP+1,YP+1,1)
      CALL MOVE(XP+3,YP+1,1)
      CALL VEC(XP+3,YP6,1)
C     DRAW ARROW
      CALL MOVE(XP+6.5,YP+3.5,1)
      CALL VEC(XP+10.5,YP+3.5,1)
      CALL MOVE(XP+8.5,YP+5.5,1)
      CALL VEC(XP+10.5,YP+3.5,1)
      CALL VEC(XP+8.5,YP+1.5,1)
      RETURN
      END
C
C
C
      SUBROUTINE CP(N1,N2)
C
C     CREATE GROUNDED PROCESSOR AND
C     DISPLAY POINTING TO STMT N1.
C     PROCESSOR WILL BE ENCLOSED
C     IN CONTOUR N2 WHEN IT IS CREATED
C
      CHARACTER*8 N1
      INTEGER X1,Y1
      ASCII N2,NAME
      COMMON MAXENT,MAXLEV,NAME(20),T(10,20)
      COMMON/INDEX/DUM(11),X1,Y1
      DO 10 I=2,MAXENT
      IF(NAME(I).EQ.N2) GO TO 20
   10 CONTINUE
      PRINT:"***ERROR*** ",N1," NOT DEFINED"
      RETURN
C     DISPLAY PROCESSOR
   20 CALL STEPNO
      CALL START
      CALL DRAWPR(196.,10.)
      CALL DRAWIP(N1)
      CALL SENDPDI
      CALL COMPIL(1,1,LFN)
      RETURN
      END
```

```
C
C
      SUBROUTINE STEPNO
C
C     REQUEST THE NO OF PROG STEPS
C     TO EXECUTE WITHOUT HALTING
C     IF INPUT IS NEGATIVE,EXECUTE
C     REST OF PROGRAM.
C     IF INPUT=0 THEN REQUEST NEW
C     SCALE MODE
C
      ASCII SCALCN
      COMMON/STEP/ISTEP,ISTAT,SCALCN
    5 CALL ELINE(10.)
      CALL ELINE(0.)
      CALL MOVE2(20.)
      PRINT:"# OF INSTRUCTION STEPS "
      READ:ISTEP
      IF(ISTEP.NE.0) RETURN
C     RESET MODE
      CALL ELINE(10.)
      CALL ELINE(0.)
      CALL MOVE2(20.)
      PRINT:"AUTO (0) OR MANUAL (1) SCALING"
      READ:ISTAT
      IF(ISTAT.EQ.0) GO TO 10
C     MANUA SCALING
      CALL ELINE(10.)
      CALL ELINE(0.)
      CALL MOVE2(20.)
      PRINT:"OUTERMOST DISPLAYED CONTOUR IS "
      READ:SCALCN
   10 GO TO 5
      END
C
C
C
      SUBROUTINE MOVE2(A)
C
C     MOVE TO POINT(0,A) ON SCREEN
C
      CALL START
      CALL MOVE(0.,A,0.)
      CALL SENDPDF
      CALL CONFIL(1,1,LEN)
      RETURN
      END
```

```
C
C
      SUBROUTINE ELINE(A)
C     ERASE LINE A ON SCREEN
      CALL START
      CALL MOVE(0.,A,0.)
      CALL EMODE
      CALL ATEXT(32,'                                ')
      CALL VMODE
      CALL SENDPDF
      CALL COMPIL(1,1,LIN)
      RETURN
      END
```

```
      SUBROUTINE CC(C1)
C
C     DRAW OUTER CONTOUR AND VARIABLES
C     SPACES ON SCREEN. MARK CONTOUR
C     AS DISPLAYED.
C     SCALE IF NECESSARY
C
      REAL SCR1(4),SCR2(4)
      ASCII NAME(20),C1,V(200),SCALCN
 INTEGER T,VAR,LEVL ,FAT,SON,BRO,SYN,DEL,PH,PW,H,W,X1,X2,Y1,Y2
      COMMON NAMENT,MAXLEV,NAME,T(16,20)
      COMMON/STEP/ISTEP,ISTAT,SCALCN
      COMMON /POINT/S(4,20),P(2,20)
      COMMON/INDEX/VAR,LEVL,FAT,SON,BRO,
&     SYN,DEL,PH,PW,H,W,X1,Y1,X2,Y2
      COMMON/SCLE/SF,DX,DY,NC,NV,P2(2,20),S1(4,20),S2(4,20)
      COMMON/VNAME/V,IVPTR
      COMMON/GRAF/INDEX,PDF(4,400),TXTIND(2,50),TEXT(200),OUT(600),PA
C      INITIALIZE COUNTERS AND POINTERS
      CALL START
      NV=0
C     FIND NAME IN TREE
      DO 10 I=2,MAXENT
      IF(NAME(I).EQ.C1) GO TO 15
   10 CONTINUE
      PRINT:"***ERROR*** ",C1,"NOT INITIALIZED"
      RETURN
C     SET CURRENT CONTOUR
   15 NC=I
      CALL PUTV(C1,II)
      T(PH,I)=II
```

```
C
C     SET ORIGINAL DISPLAY POINTS
C
   5Ø DO 45 J=1,4
   45 S1(J,I)=S(J,I)
      P2(X1,I)=P(X1,I)
      P2(Y1,I)=P(Y1,I)
      T(PW,I)=1
C
C     PUT NEW CONTOUR IN PDS
C
      CALL CIPDF(I)
      CALL MOVE(S(X2,I)-26.,S(Y2,I)+2.,Ø)
      CALL HTEXT(4,C1)
   55 T(SYN,I)=-1
   6Ø CALL MOVE(Ø,Ø,Ø)
      CALL SENDPDF
      CALL COMPILE(1,1,LEN)
      RETURN
      END
C
C
C
      SUBROUTINE CIPDF(I)
C
C     ENTER CONTOUR AND VARIABLE BOXES IN PDF
C
      INTEGER T,VAR,X1,Y1,X2,Y2
      COMMON NXT,MAXLEV,NAME(2Ø),T(1Ø,2Ø)
       COMMON/SCLE/D(5),P2(2,2Ø),S(4,2Ø),S2(4,2Ø)
       COMMON/INDEX/VAR,DUM(7),PW,H,W,X1,Y1,X2,Y2
       SX1=S(X1,I)
       SY1=S(Y1,I)
       SX2=S(X2,I)
       SY2=S(Y2,I)
C
C      ENTER CONTOUR IN PDF
C
       CALL MOVE(SX2,SY2,Ø.)
       CALL VEC(SX1,SY2,Ø.)
       CALL VEC(SX1,SY1,Ø.)
       CALL VEC(SX2,SY1,Ø.)
       CALL VEC(SX2,SY2,Ø.)
C      IF VARIABLE BOXES DONT FIT, RETURN
       IF(T(PW,I).EQ.-1) RETURN
C      ENTER VARIABLE BOXES IN PDF
       JJ=T(VAR,I)
       IF(JJ.EQ.Ø) RETURN
       SY1=SY2
       SX2=SX1+75
       DO 3Ø J=1,JJ
       SY2=SY2-1Ø
       CALL MOVE(SX1,SY2,Ø)
   3Ø  CALL VEC(SX2,SY2,Ø)
```

```
      DO 40 J=1,3
      SX1=SX1+20
      CALL MOVE(SX1,SY1,0)
   40 CALL VEC(SX1,SY2,0)
      RETURN
      END
C
C
C
      SUBROUTINE INCON
C
C     INITIALIZE CONTOUR ROUTINES
C
      COMMON/TXTPTS/ITEXT,ITPTR
      COMMON/SCLE/SF,DX,DY,NC,NV,P2(2,20),S1(4,20),S2(4,20)
      COMMON/VNAME/V(200),IVPTR
      CALL START
      CALL CLEAR
      CALL SENDPDF
      CALL COMPIL(1,1,LEN)
C     SET CSALE FACTOR TO 1
      SF=1
      DX=0.
      DY=0.
C     SET TEXT POINTERS
      IVPTR=1
      ITPTR=1
      ITEXT=1
C     INITIALIZE SCALE ROUTINE
      RETURN
      END
C
C
C
      SUBROUTINE PUTV(STR,I)
C     SAVE STR IN VARIABLE NAME ARRAY
      ASCII STR,V(200)
      COMMON/VNAME/V,IVPTR
      I=IVPTR
      V(IVPTR)=STR
      IVPTR=IVPTR+1
      IF(IVPTR.GT.200) PRINT:"VARIABLE OVERFLOW"
      RETURN
      END
```

```
      SUBROUTINE CNAME(N1,N2,N3)
C     ENTER NAME IN TEXT
C     DRAW VARIABLE NAME AND VALUE ON
C     SCREEN. (GIVES SCALED PTS)
      INTEGER T,VAR,X1,Y1,X2,Y2
      ASCII N(3),N1,N2,N3
      COMMON MAXENT,MAXLEV,NAME(20),T(10,20)
      COMMON/INDEX/VAR,DUM(10),X1,Y1,X2,Y2
      COMMON/SCLE/SF,DX,DY,NC,NV,P2(2,20),S(4,20),S2(4,20)
      N(1)=N1
      N(2)=N2
      N(3)=N3
      CALL START
      NV=NV+1
C     FIND SCALED COORDINATES
C     FOR NAME
   10 SX=(S(X1,NC)+2)
      SY=S(Y2,NC)-NV*10+2
      ADD=26
      DO 20 I=1,3
      CALL MOVE(SX,SY,0)
      CALL HTEXT(4,N(I))
      CALL PUTV(N(I),J)
   20 SX=SX+ADD
      CALL MOVE(0,0,0)
      CALL SENDPDF
      CALL COMPIL(1,1,LEN)
      RETURN
      END
```

```
      SUBROUTINE DC(N1)
C     ERASE CONTOUR N1 FROM SCREEN
C     ALONG WITH ALL CHARACTER STRINGS
C     ASSOCIATED WITH IT
      ASCII N1,NAME(20),V
      INTEGER T,VAR,SYN,X1,Y1,X2,Y2
      COMMON MAXENT,MAXLEV,NAME,T(10,20)
      COMMON/VNAME/V(200),IVPTR
      COMMON/INDEX/VAR,LEVL,FAT,SON,BRO,SYN,DEL,PH,PW,H,W,X1,Y1,X2,
      COMMON/SCLE/SF,DX,DY,NC,NV,P2(2,20),S(4,20),S2(4,20)
      CALL START
C     FIND NAME IN TREE
      DO 10 I=2,MAXENT
      IF(NAME(I).EQ.N1) GO TO 20
   10 CONTINUE
      PRINT:"***ERROR***",N1,"NOT INITIALIZED FOR DELETE"
      RETURN
C     CHECK FOR FILE DISPLAYED
   20 IF(T(SYN,I).EQ.-1) GO TO 30
      PRINT:"***ERROR***",N1,"NOT DISPLAYED"
```

```
      RETURN
C     PUT SCREEN IN ERASE MODE
   30 CALL EMODE
C
C     IF CONTOUR NAME NOT
C     DISPLAYED, CONTOUR GONE
C
      IF((S(X2,I)-S(X1,I)).LE.26) GO TO 45
      CALL MOVE(S(X2,I)-26,S(Y2,I)+2,Ø)
      CALL HTEXT(4,"    ")
   35 T(SYM,I)=1
C
C     IF VARIABLES NOT DISPLAYED, DONT
C     BLANK THEM OUT
C
      IF(T(PW,I).EQ.-1) GO TO 45
C     FIND VARIABLES TO BLANK OUT
      KK=T(VAR,I)
      IF(KK.EQ.Ø) GO TO 45
      SY=S(Y2,I)-9
      DO 4Ø K=1,KK
      SX=S(X1,I)+2
      CALL MOVE(SX,SY,Ø.)
      CALL HTEXT(12,'            ')
   4Ø SY=SY-1Ø
   45 CALL CIPDF(I)
      CALL VMODE
C     ERASE CONTOUR
   5Ø CALL MOVE(Ø,Ø,Ø)
      CALL SENDPDF
      CALL COMPIL(1,1,LEN)
      RETURN
      END
C
C
C
      SUBROUTINE AREAD(N1,C1,VAL)
C
C     READ THE VALUE OF VARIABLE
C     N1 IN CONTOUR C1 (WHICH
C     CONTAINS NO MORE THAN 4
C     CHAR) THE ACTUAL INPUT
C     IS DONE PRIOR TO THE START
C     OF THE DISPLY
C
      ASCII N1,C1,V,VAL
      COMMON/VNAME/V(2ØØ),IVPTR
C
C     ERASE BOTTOM LINES ON SCREEN
C
      CALL ELINE(Ø.)
      CALL ELINE(1Ø.)
      CALL MOVE2(1Ø.)
      PRINT:'READ:',N1,' IN ',C1,' IS ',VAL
```

```
C
C     CHANGE N1 IN C1 ON THE SCREEN
C
      CALL AVAL(N1,C1,VAL)
      RETURN
      END
C
C
C
      SUBROUTINE DP
C
C     ERASE PROCESSOR
C
      CHARACTER*8 IPVAL
      COMMON/PROCSR/PX,PY,IPVAL
      CALL START
      CALL HMODE
      CALL MOVE(PX,PY,0.)
      CALL HTEXT(10,'          ')
      CALL VMODE
      CALL SENDPDF
      CALL COMPIL(1,1,LEN)
      RETURN
      END
C
C
C
      SUBROUTINE APRINT(N1,C1)
C
C     PRINT THE VALUE OF VAR
C     N1 IN CONTOUR C1 AT THE
C     BOTTOM OF THE SCREEN
C
      ASCII N1,C1,VAL,V
      COMMON/VNAME/V(200),IVPTR
C
C     ERASE BOTTOM LINES ON SCREEN
C
      CALL ELINE(0.)
      CALL ELINE(10.)
      CALL MOVE2(10.)
C
C     FIND THE VALUE OF N1
C
      CALL FINDV(N1,C1,I,J,L)
      IF(J.EQ.0) GO TO 10
      VAL=V(J+2)
      PRINT:'PRINT:',N1,' IN ',C1,' IS ',VAL
      RETURN
   10 PRINT:"***ERROR*** ",N1," IN PRINT STMT DOES NOT EXIST"
      RETURN
      END
```

```
C
C
C
      SUBROUTINE SIZE
C
C     CALCULATE SIZES AND PTS FOR ALL
C     CONTOURS
C
      COMMON NNT,MAXLEN,NAME(20),T(10,20)
      MAXENT=NNT-1
      CALL FINDSZ(MAXENT)
      CALL SGRAPH(MAXENT)
      CALL SPPOS(MAXENT)
      RETURN
      END
```

APPENDIX V.

Flowcharts for Projected Scaling Routines

Variables Used in Scaling Routines
that Are Not Previously Described

RESCLE:

ADD1: }
ADD2: } The number of screen points that are to be left between brothers.

CFX: }
CFY: } Screen coordinates of the center of the brothers.

CSX: }
CSY: } Screen coordinates of the center of the free space.

DIF: Number of screen points available in free space for spacing contours.

DX: }
DY: } Number of screen points the center of the brothers must be translated to position it in the center of the free space.

HEIGHT: }
WIDTH: } Dimensions of all brothers combined.

ISAVE: Number of brothers to be displayed.

ISIB: Pointer in TREE.

J: Local variable.

K: Pointer in TREE.

TSTFIT:

ADD: A flag. Also used in setting values in AVAIL.

IFLG: A flag.

PUTNMS:

INME: An index in V.

KK: Number of variables in the contour.

SX: } Local variables containing x and y positions.
SY: }

DSPSC:

N1: Local variable.

TSTSCL:

C2: Index in TREE of the father of C1.

IC: Index in TREE of the outermost contour displayed.

IFIT: } Local integer variables.
K: }
NEXT: }
NEW: }

NEWSON: 1x20 integer array used to save unprocessed active contours.

SCR1: } 4x20 real arrays used for work spaces.
SCR2: }
SCR3: }
SCR4: }

SVSCR: 4x20 integer array used to save the available space in the unprocessed nodes.

RESCLE(I,SCR1,SCR2,EVEN,NBRO,*): Rescale the contour whose index is I so it and its brothers, if any, will fit within the screen coordinates defined in SCR1. Take the alternate return if a successful fit is found.

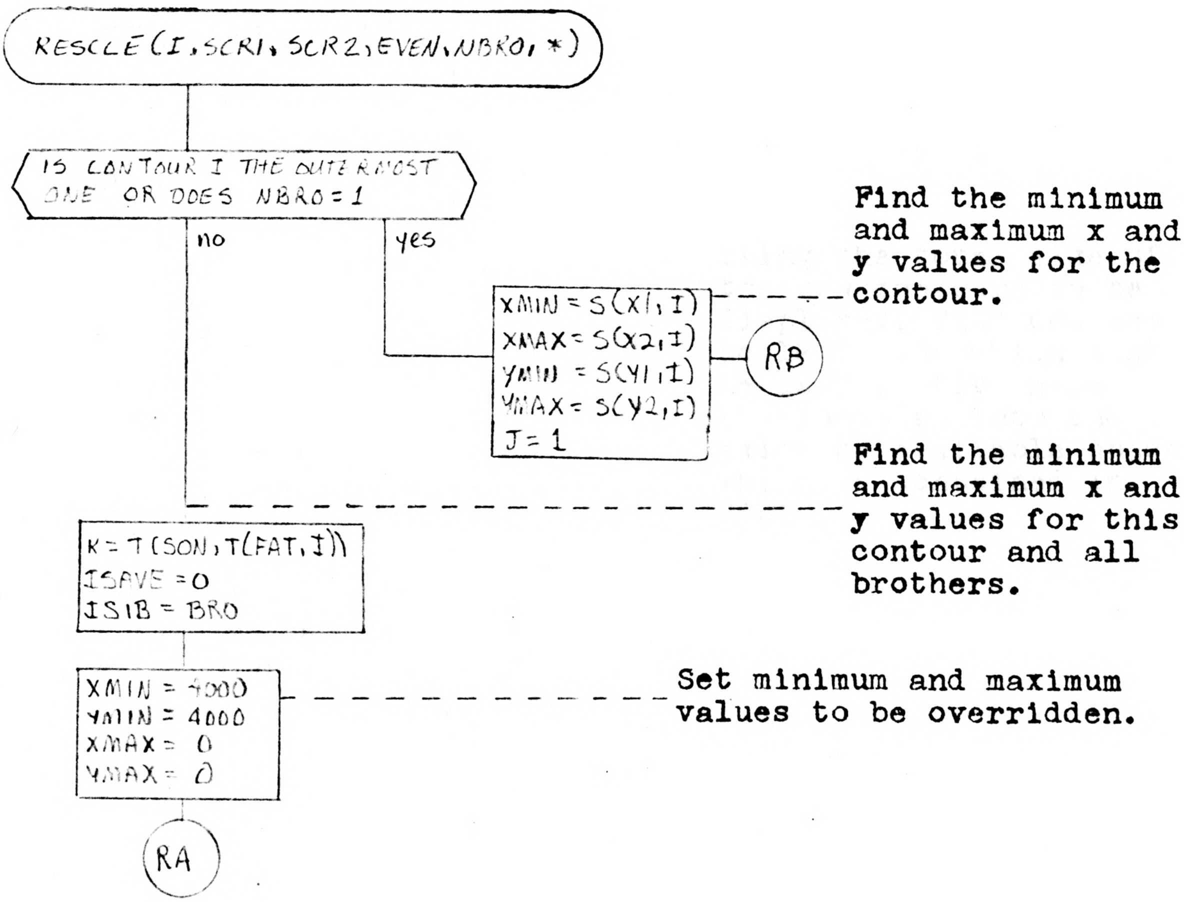
I:index of the contour name in NAME

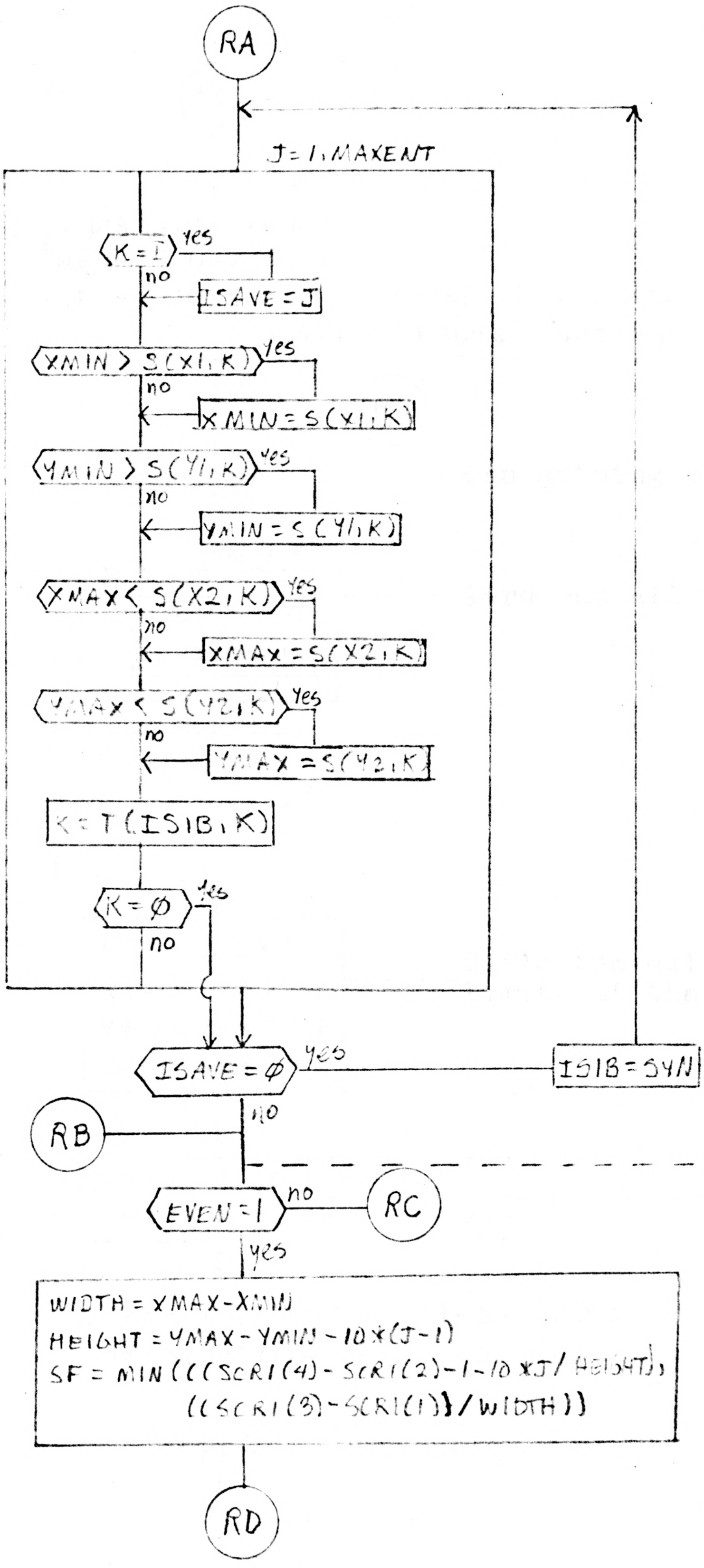
SCR1: (real array dimensioned 4) Contains the minimum and maximum screen coordinate values to be used. The return values are the lower left and upper right screen coordinates of the contour.

SCR2: (real array dimensioned 4) Contains the minimum and maximum screen coordinates left after the variable boxes have been placed in the contour.

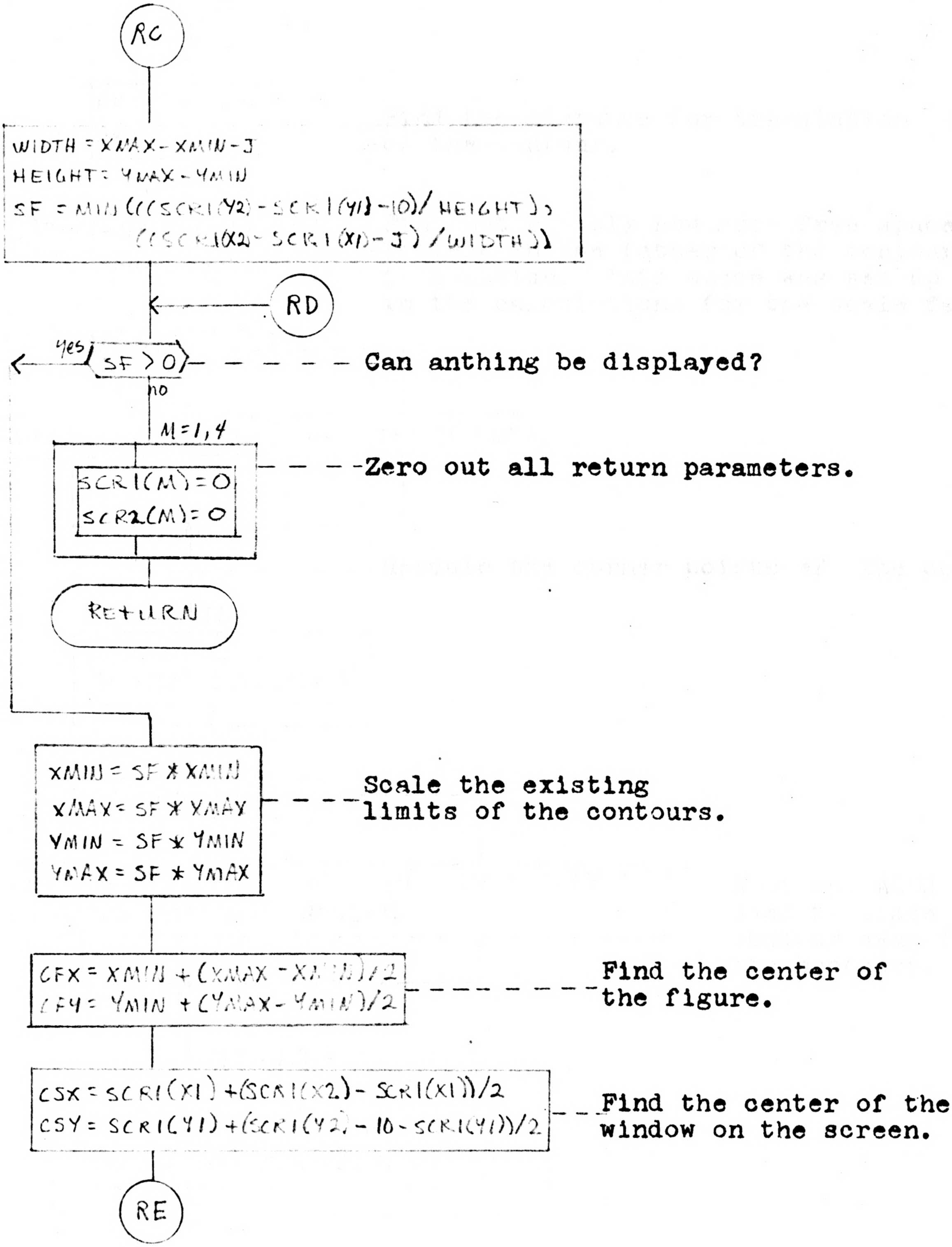
EVEN: (integer) On entry to the subroutine, EVEN equals 1 if the contour has an even level number, -1 if the contour has an odd level number.

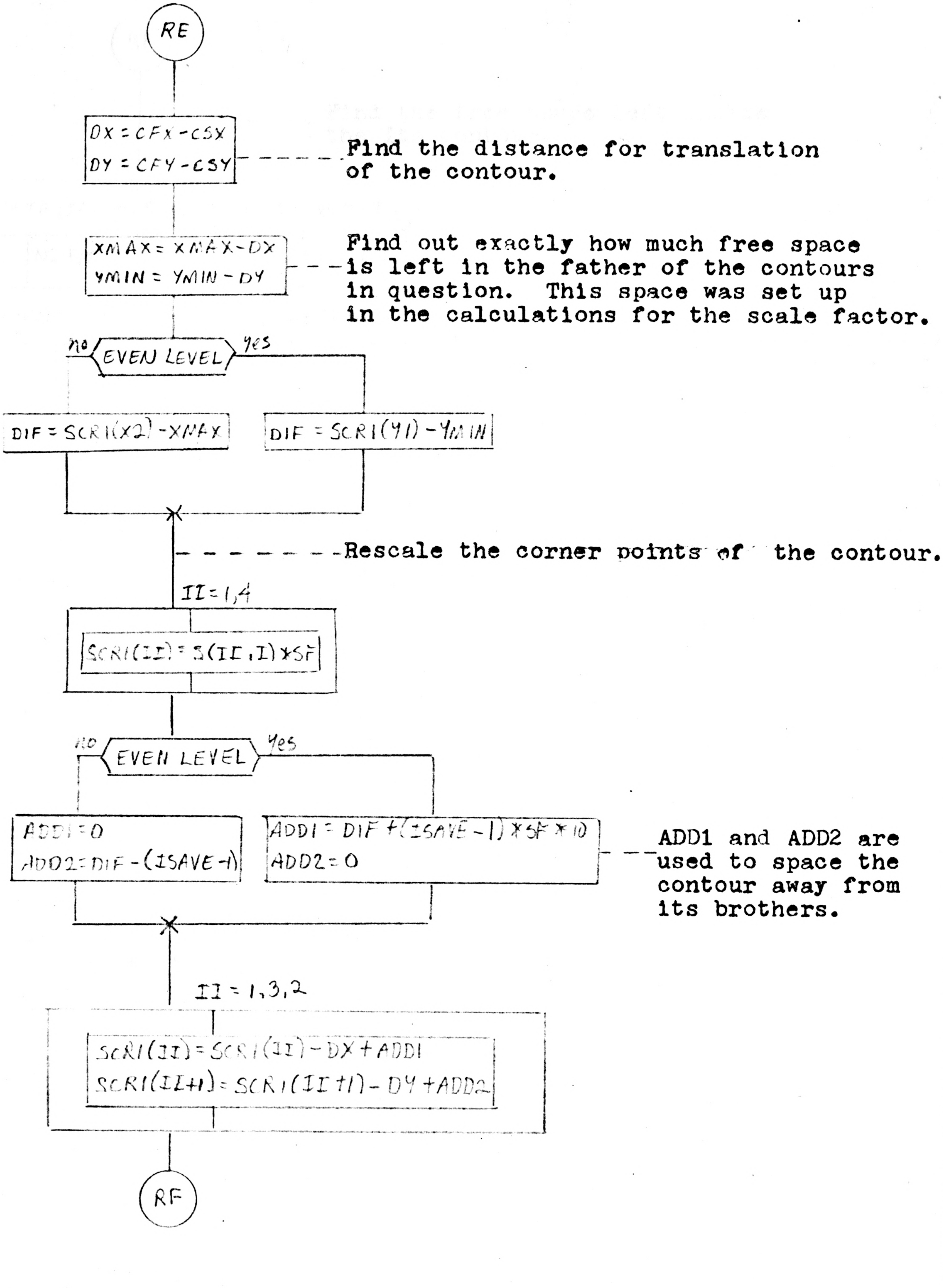
NBRO: (integer) On entry to the subroutine, NBRO is 1 if the contour is to be fitted to the given space with no brothers.

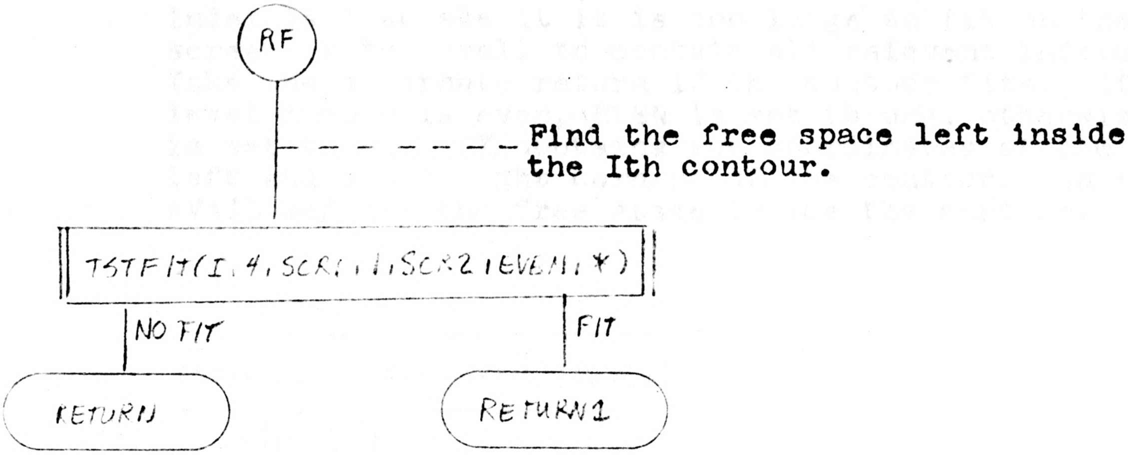




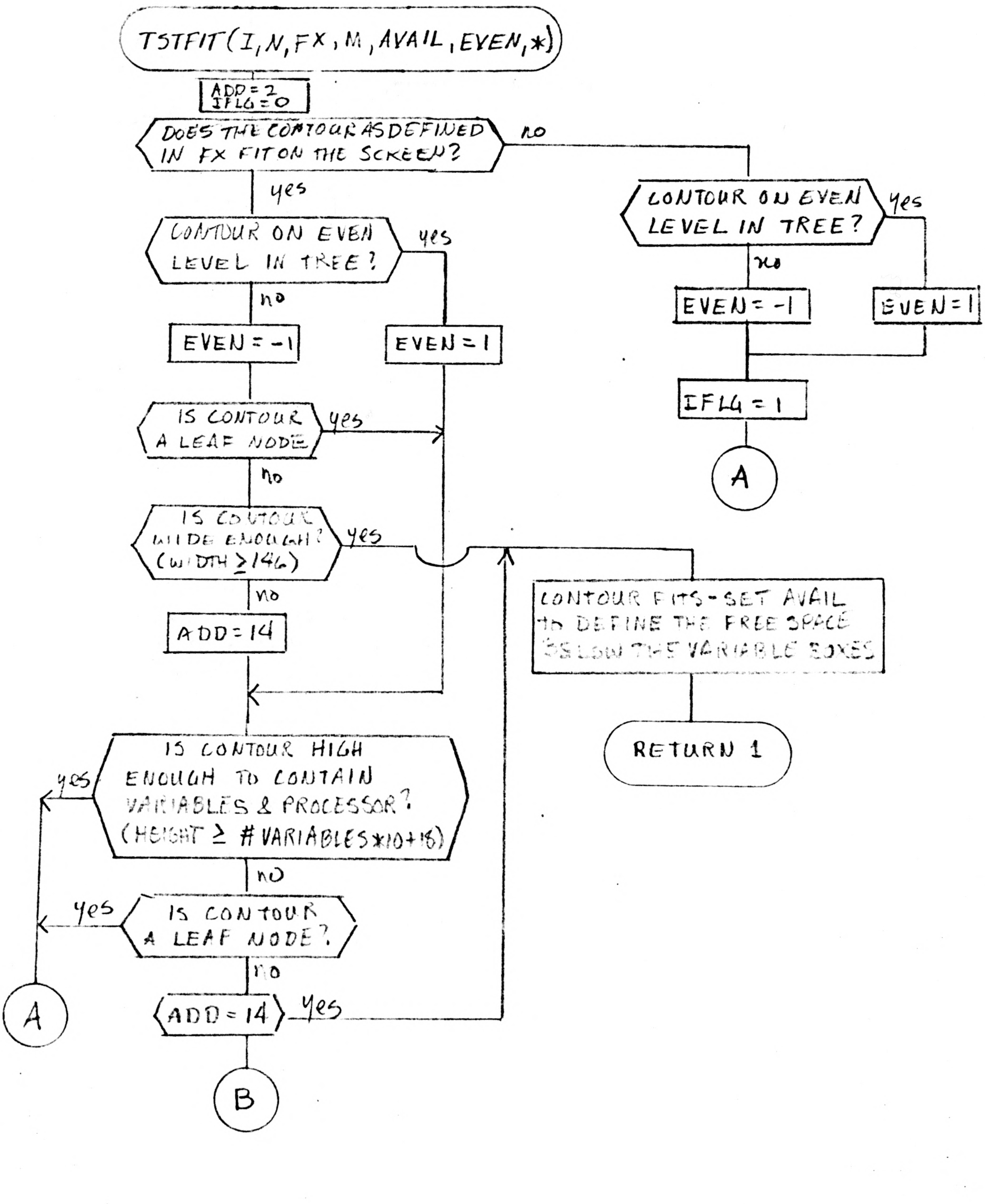
Using the total size of the new contours to be displayed, find out the scale factor which must be used to fit these new contours into the existing available space which is contained in SCR1.

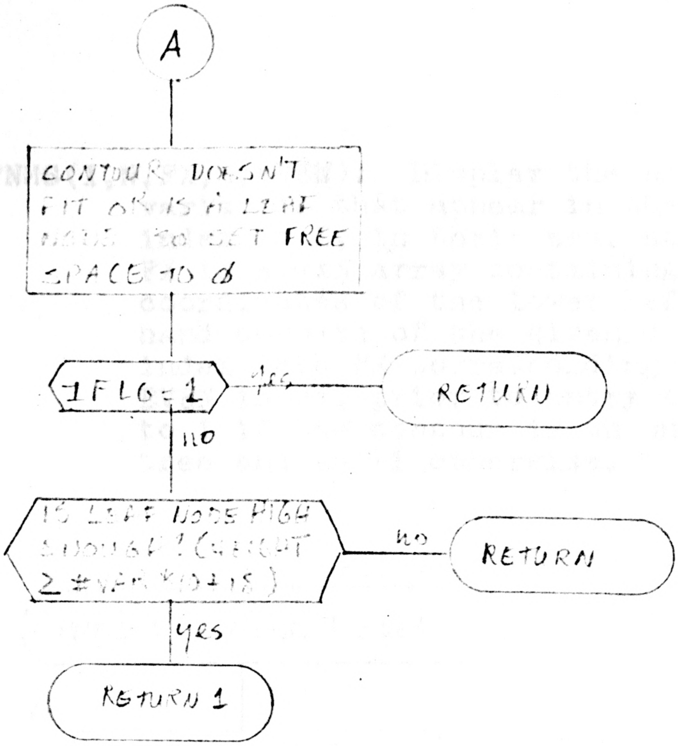

RC
WIDTH = XMAX - XMIN - J
HEIGHT = YMAX - YMIN
SF = MIN(((SCR1(Y2) - SCR1(Y1) - 10)/HEIGHT),
((SCR1(X2) - SCR1(X1) - J)/WIDTH))
RD
yes
SF > 0
no
Can anthing be displayed?
M=1,4
SCR1(M)=0
SCR2(M)=0
Zero out all return parameters.
RETURN
XMIN = SF * XMIN
XMAX = SF * YMAX
YMIN = SF * YMIN
YMAX = SF * YMAX
Scale the existing limits of the contours.
CFX = XMIN + (XMAX - XMIN)/2
CFY = YMIN + (YMAX - YMIN)/2
Find the center of the figure.
CSX = SCR1(X1) + (SCR1(X2) - SCR1(X1))/2
CSY = SCR1(Y1) + (SCR1(Y2) - 10 - SCR1(Y1))/2
Find the center of the window on the screen.
RE

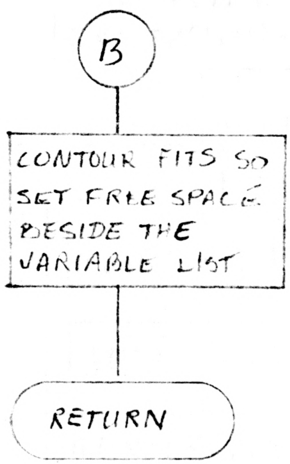

RE
DX = CFX − CSX
DY = CFY − CSY
Find the distance for translation of the contour.
XMAX = XMAX − DX
YMIN = YMIN − DY
Find out exactly how much free space is left in the father of the contours in question. This space was set up in the calculations for the scale factor.
no
EVEN LEVEL
yes
DIF = SCR1(X2) − XMAX
DIF = SCR1(Y1) − YMIN
Rescale the corner points of the contour.
II = 1,4
SCR1(II) = S(II,I) * SF
no
EVEN LEVEL
yes
ADD1 = 0
ADD2 = DIF − (ISAVE − 1)
ADD1 = DIF + (ISAVE − 1) * SF * 10
ADD2 = 0
ADD1 and ADD2 are used to space the contour away from its brothers.
II = 1,3,2
SCR1(II) = SCR1(II) − DX + ADD1
SCR1(II+1) = SCR1(II+1) − DY + ADD2
RF


RF
Find the free space left inside the Ith contour.
TSTFIT(I,4,SCR1,1,SCR2,LEVEL,*)
NO FIT
FIT
RETURN
RETURN1

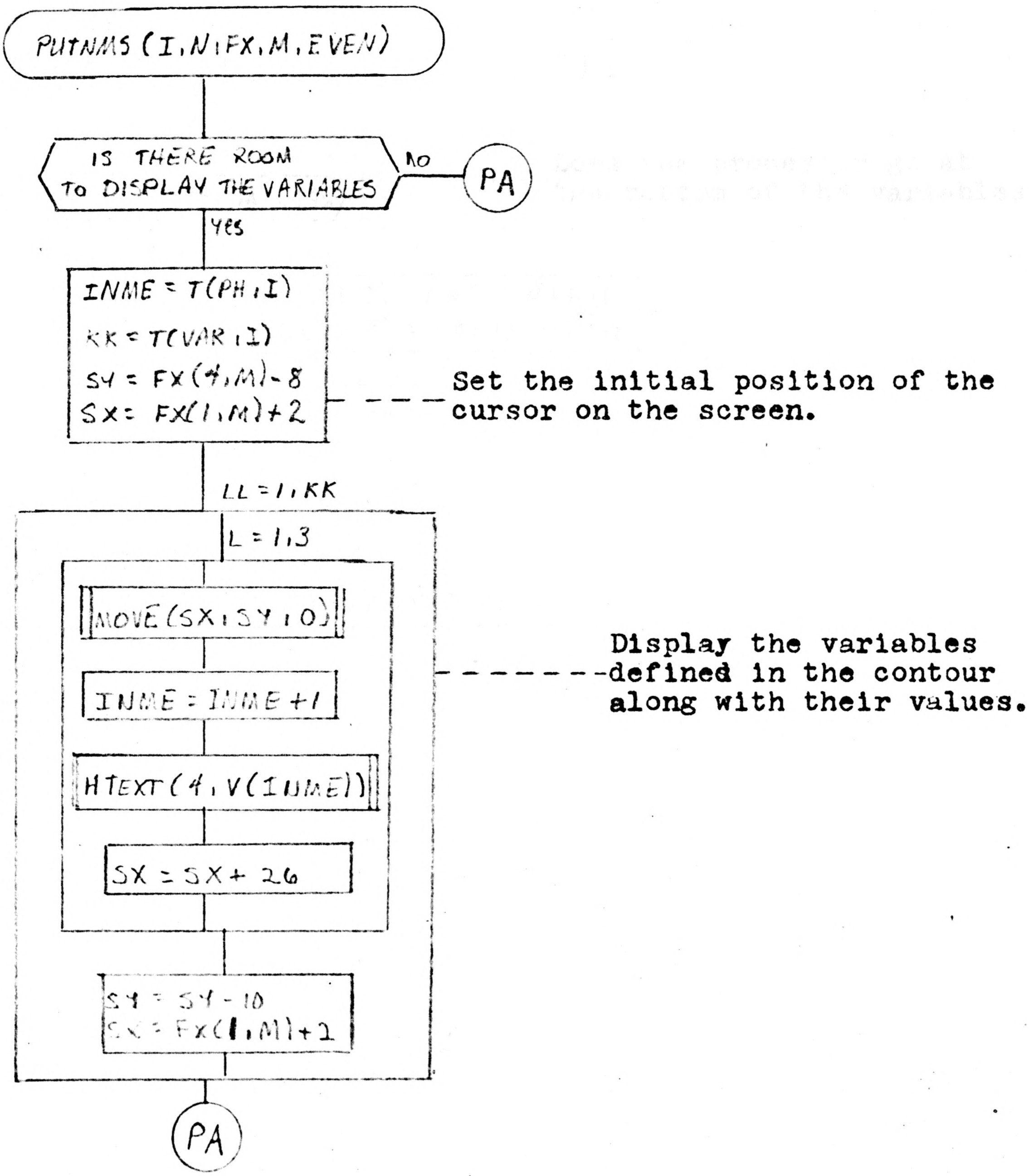
TSTFIT(I,N,FX,M,AVAIL,EVEN,*): Check the contour whose index is I to see it it is too large to fit on the screen or too small to contain all relevent information. Take the alternate return if the contour fits. If the level number is even, EVEN is set to one, otherwise it is set to -1. FX contains the coordinates of the lower left and upper right corners of the contour. On return, AVAIL defines the free space inside the contour.

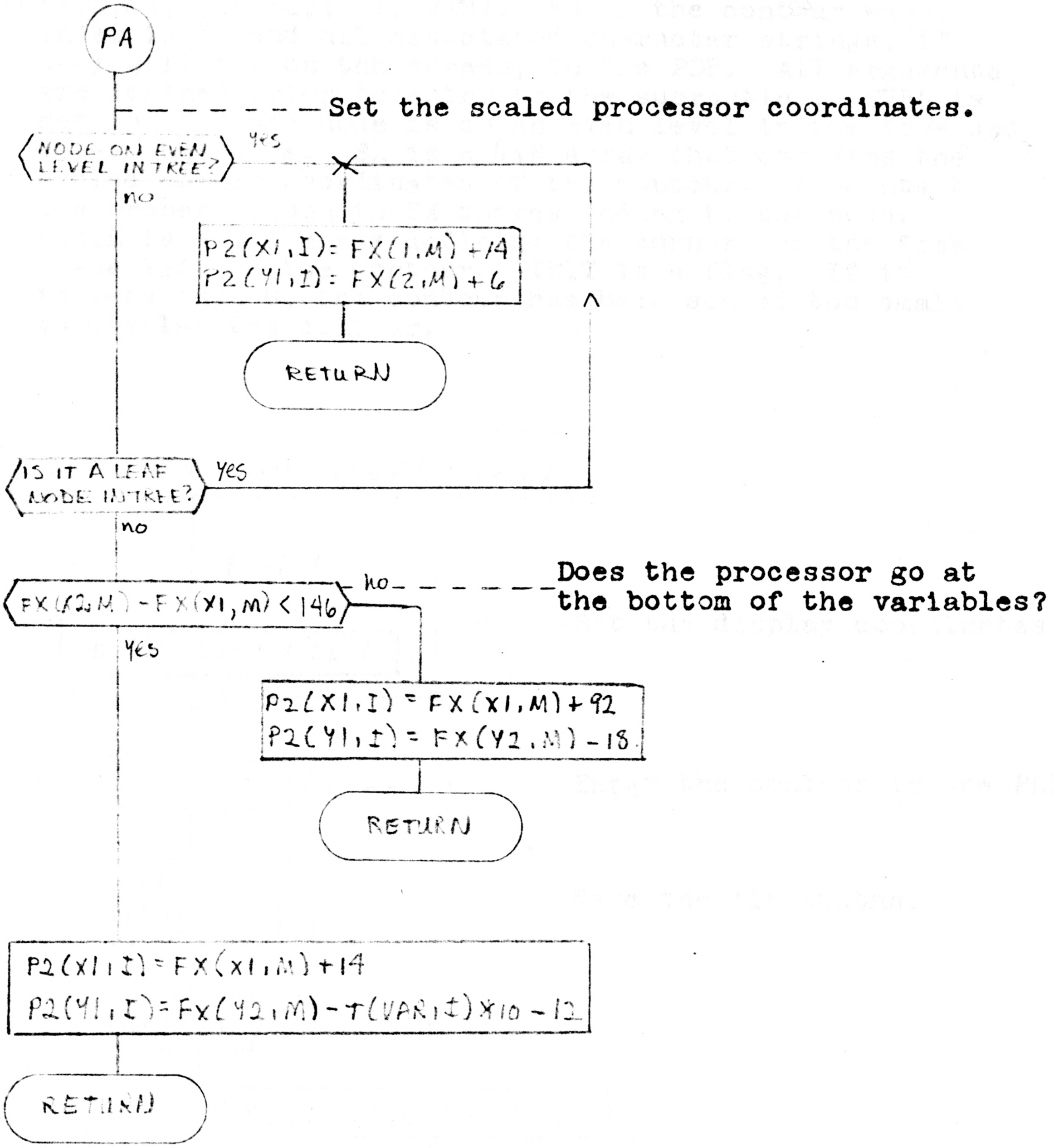



A
CONTOUR DOESN'T FIT OR IS A LEAF NODE SO SET FREE SPACE TO ∅
IFLG=1
yes
RETURN
no
IS LEAF NODE HIGH ENOUGH? (HEIGHT
no
RETURN
yes
RETURN 1

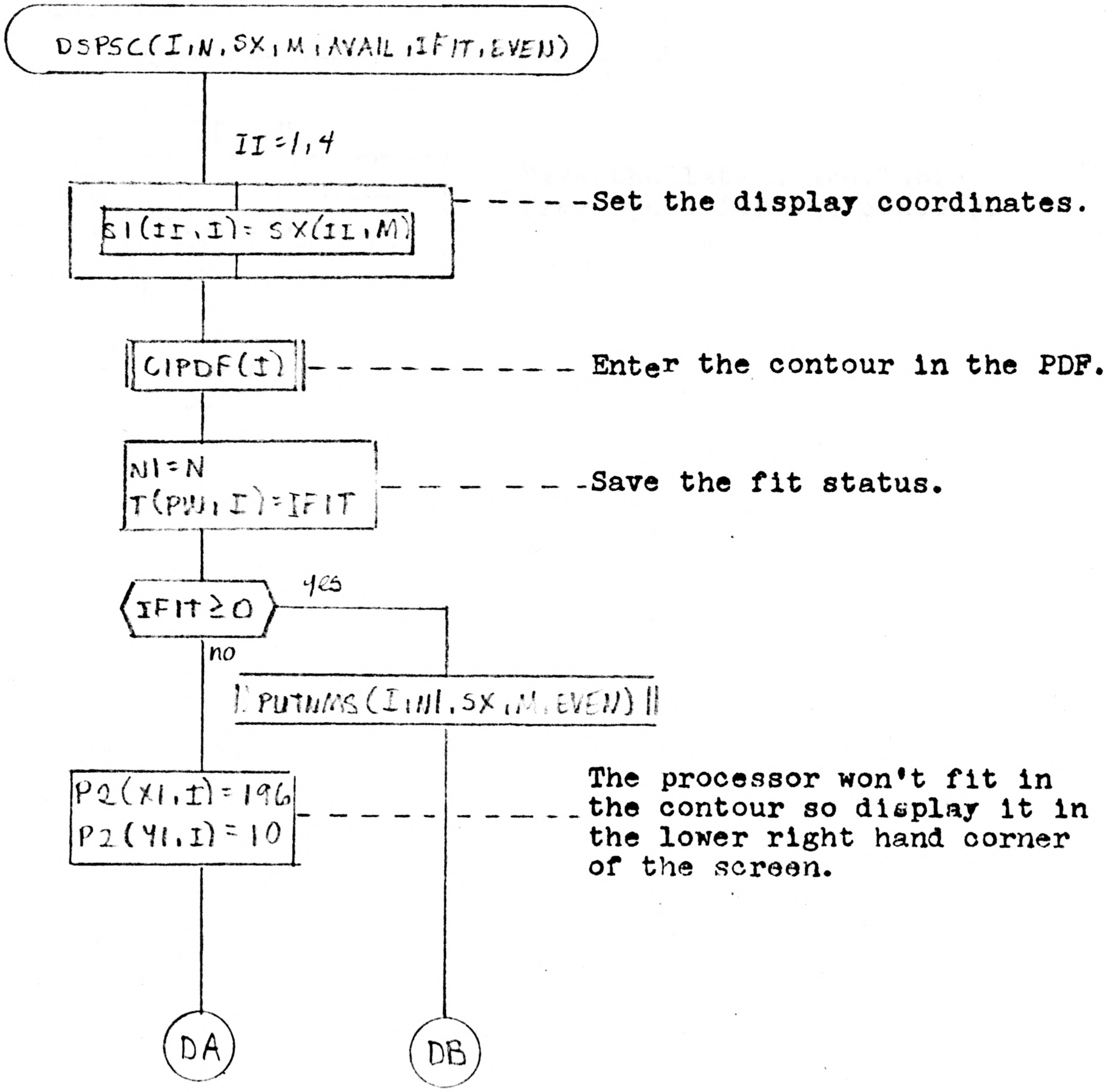

B
CONTOUR FITS SO SET FREE SPACE BESIDE THE VARIABLE LIST
RETURN

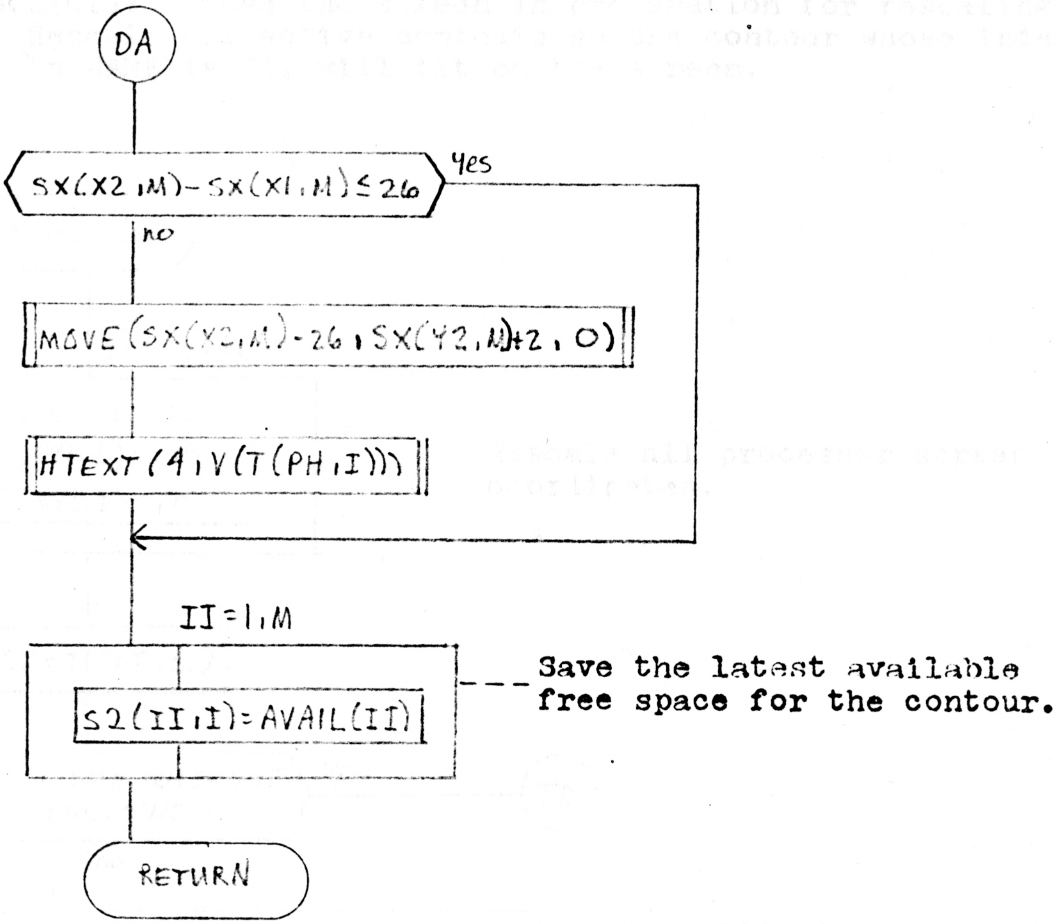
PUTNMS(I,N,FX,M,EVEN): Display the previously defined variables that appear in the contour whose index is I, in their new, scaled positions. FX is a 4xN array containing the screen coordinates of the lower left and upper right hand corners of the given contour. M is an index into FX corresponding to the given contour. EVEN is set prior to entry to the subroutine to 1 if the contour is on an even level in the tree and to -1 otherwise.



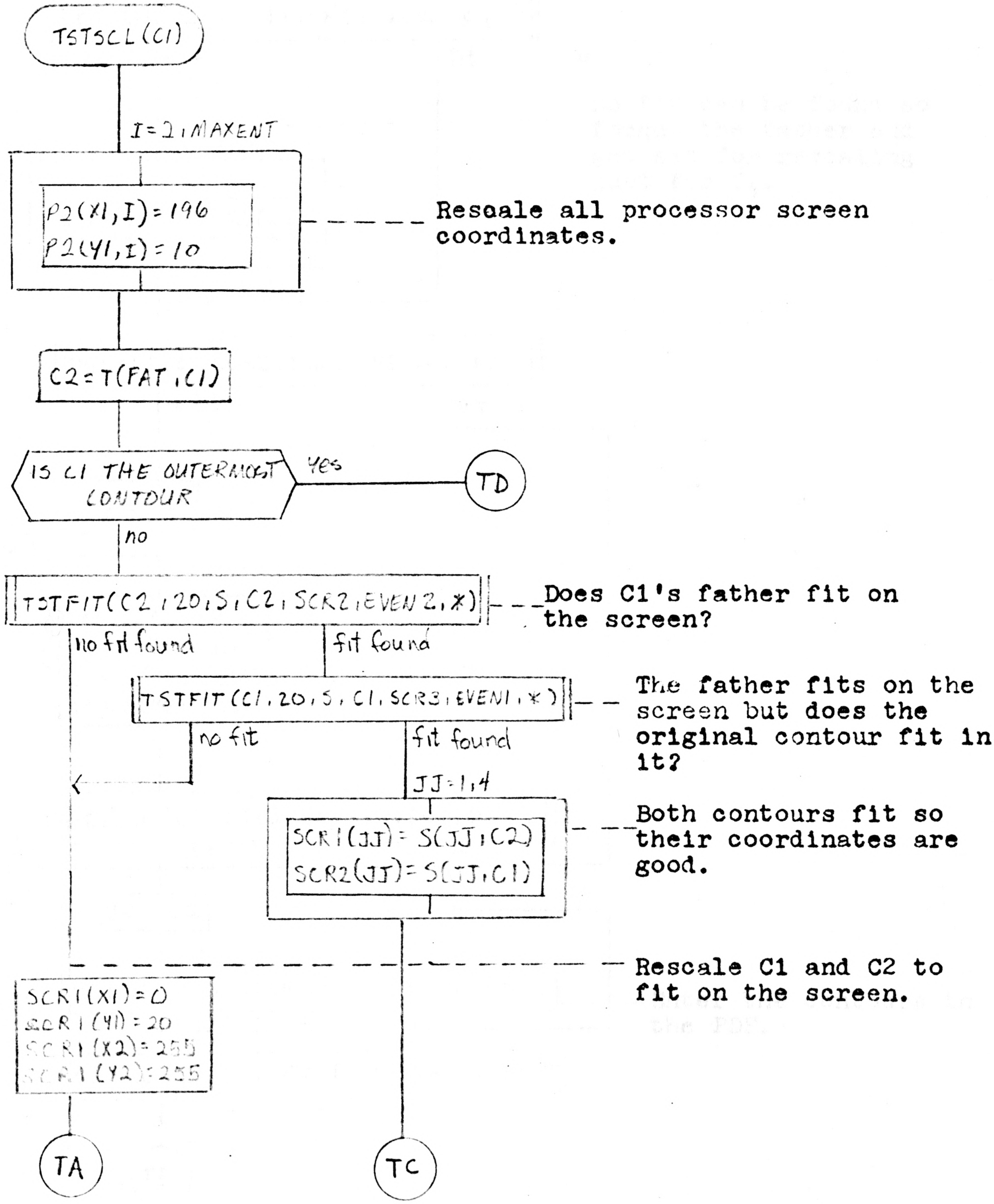

PA
Set the scaled processor coordinates.
NODE ON EVEN LEVEL IN TREE?
yes
no
P2(X1,I)= FX(1,M)+14
P2(Y1,I)= FX(2,M)+6
RETURN
IS IT A LEAF NODE IN TREE?
yes
no
FX(X2,M)-FX(X1,M)<146
no
Does the processor go at the bottom of the variables?
yes
P2(X1,I)=FX(X1,M)+92
P2(Y1,I)=FX(Y2,M)-18
RETURN
P2(X1,I)=FX(X1,M)+14
P2(Y1,I)=FX(Y2,M)-T(VAR,I)*10-12
RETURN

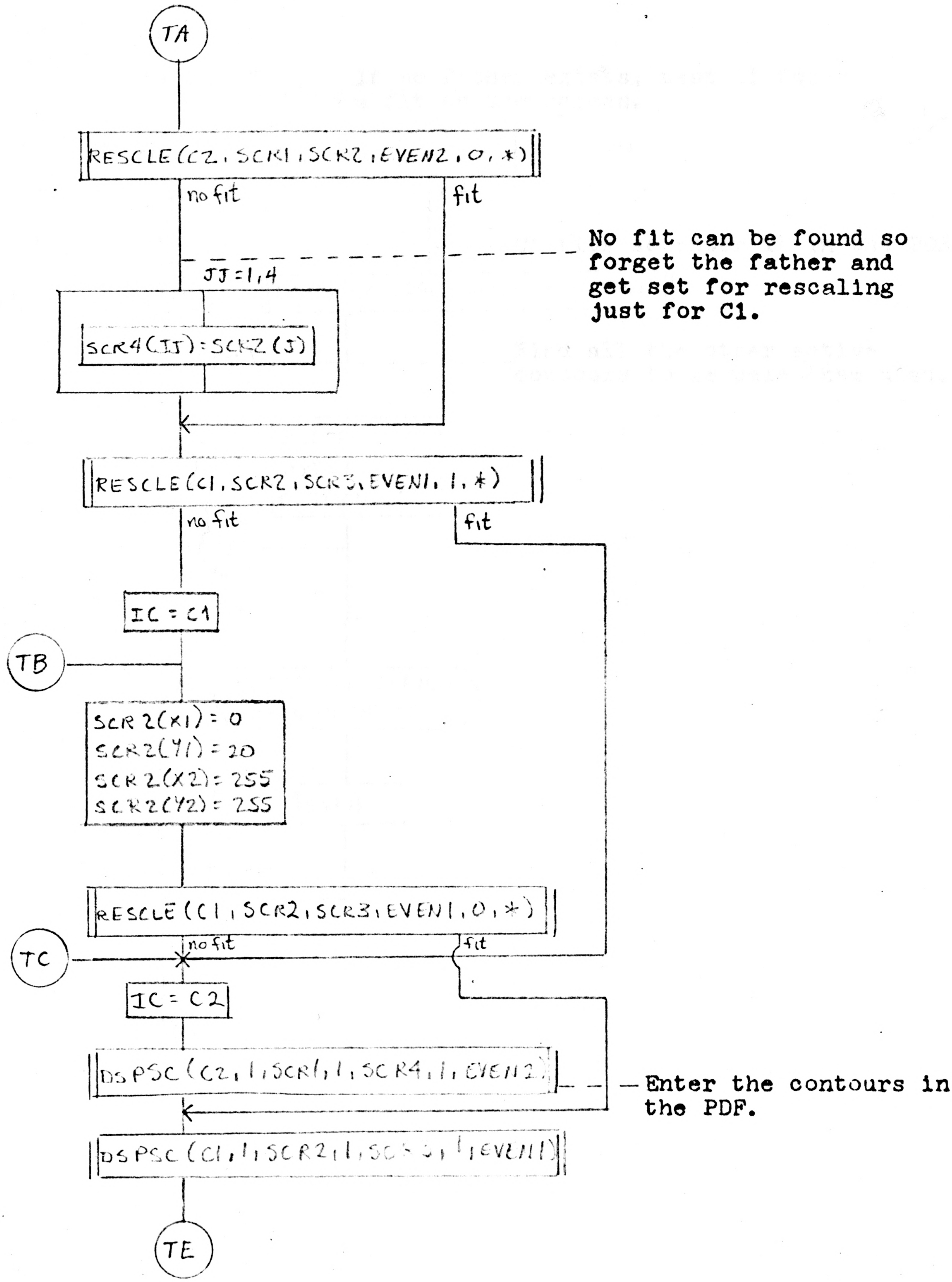
DSPSC(I,N,SX,M,AVAIL,IFIT,EVEN): Enter the contour whose index is I, and all associated character strings, if they will fit on thh screen, in the PDF. All arguments are defined prior to entry to the subroutine. EVEN is set to 1 if the node is on an even level in the tree and to -1 otherwise. SX is a 4xN array that contains the scaled corner coordinates of the contour. M points to the proper column in SX cooresponding to the node. AVAIL is a 1x4 array defining the corners of the free space left in the contour. IFIT is a flag. If it is less than 0, the contour has been scaled too small to display the contour.

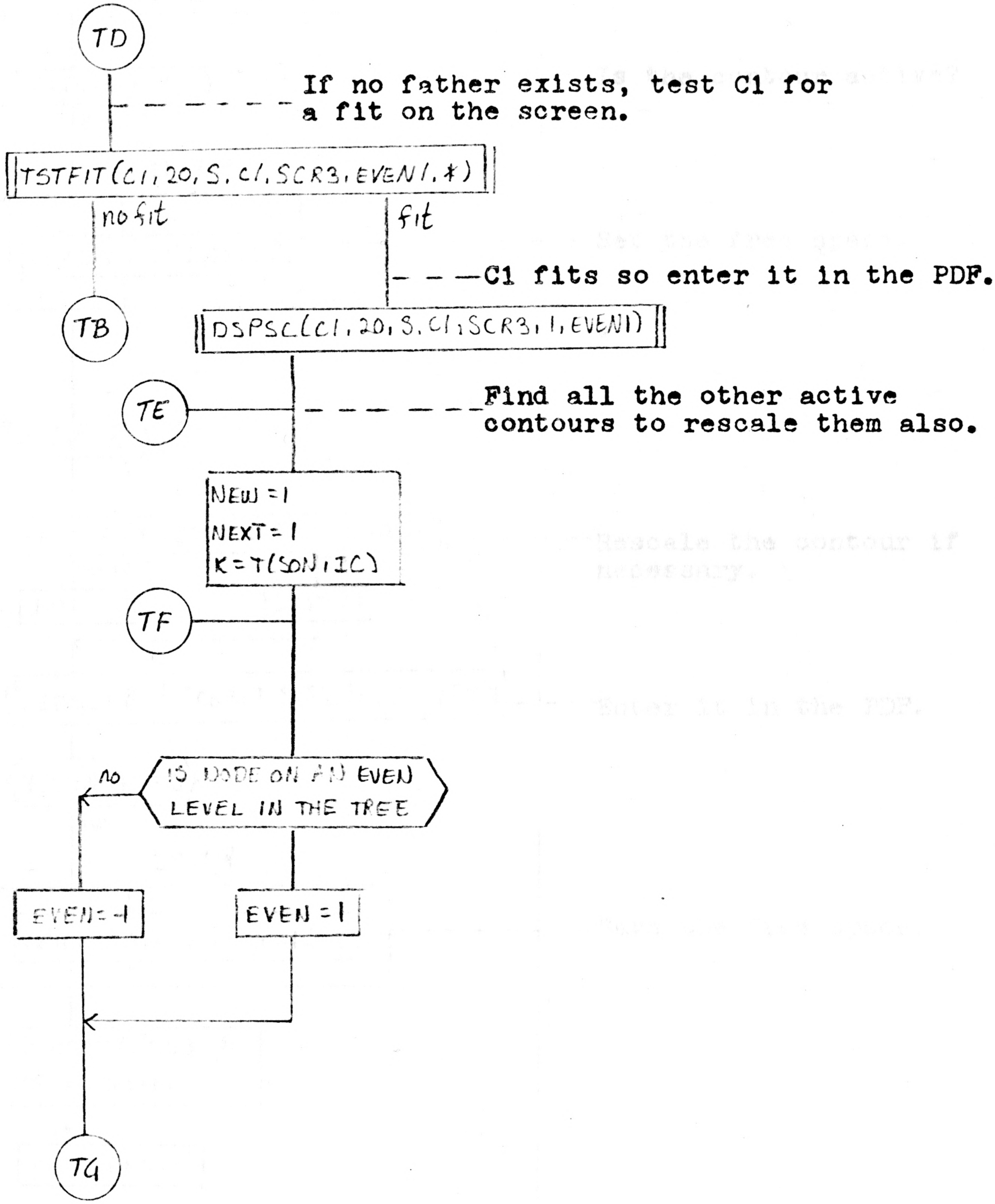


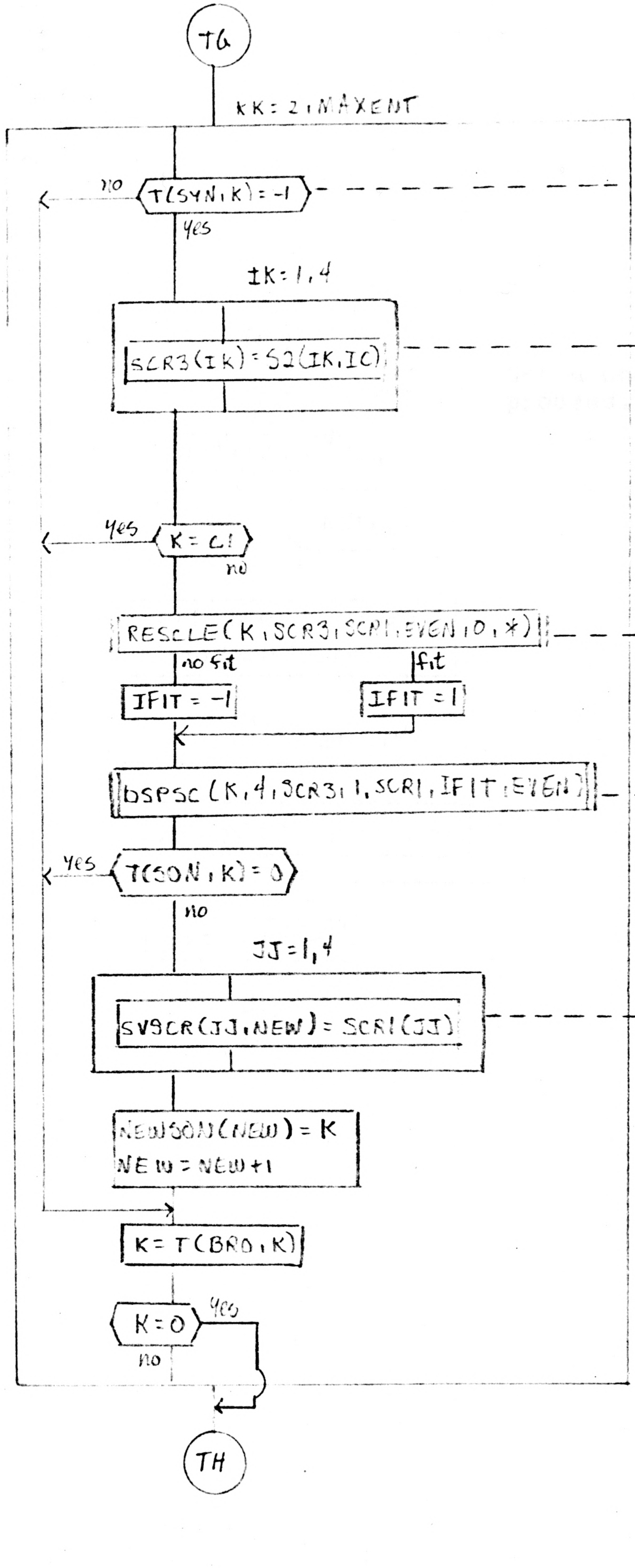

DA
SX(X2,M)-SX(X1,M)≤26
yes
no
MOVE(SX(X2,M)-26,SX(Y2,M)+2,0)
HTEXT(4,V(T(PH,I)))
II=1,M
S2(II,I)=AVAIL(II)
Save the latest available free space for the contour.
RETURN

TSTSCL(C1): Erase the screen in preparation for rescaling. Rescale all active contours so the contour whose index in NAME is C1, will fit on the screen.




TA
RESCLE(C2,SCR1,SCR2,EVEN2,0,*)
no fit
fit
JJ=1,4
SCR4(JJ)=SCR2(J)
RESCLE(C1,SCR2,SCR3,EVEN1,1,*)
no fit
fit
IC=C1
TB
SCR2(X1)=0
SCR2(Y1)=20
SCR2(X2)=255
SCR2(Y2)=255
RESCLE(C1,SCR2,SCR3,EVEN1,0,*)
no fit
fit
TC
IC=C2
DSPSC(C2,1,SCR1,1,SCR4,1,EVEN2)
DSPSC(C1,1,SCR2,1,SCR3,1,EVEN1)
TE
No fit can be found so forget the father and get set for rescaling just for C1.
Enter the contours in the PDF.


TD
If no father exists, test C1 for a fit on the screen.
TSTFIT(C1,20,S,C1,SCR3,EVEN1,*)
no fit
fit
C1 fits so enter it in the PDF.
TB
DSPSCL(C1,20,S,C1,SCR3,1,EVEN1)
TE
Find all the other active contours to rescale them also.
NEW=1
NEXT=1
K=T(SON,IC)
TF
IS NODE ON AN EVEN LEVEL IN THE TREE
NO
EVEN=-1
EVEN=1
TG



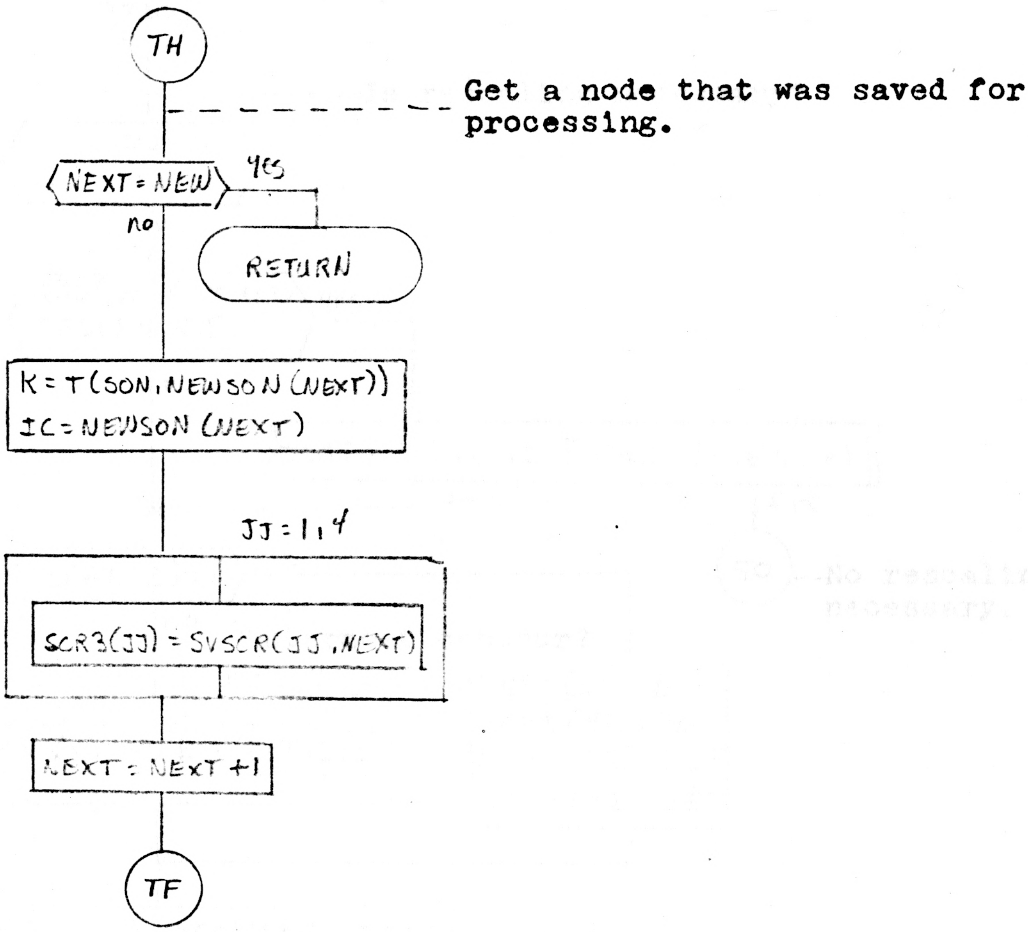
Is the contour active?

Set the free space.

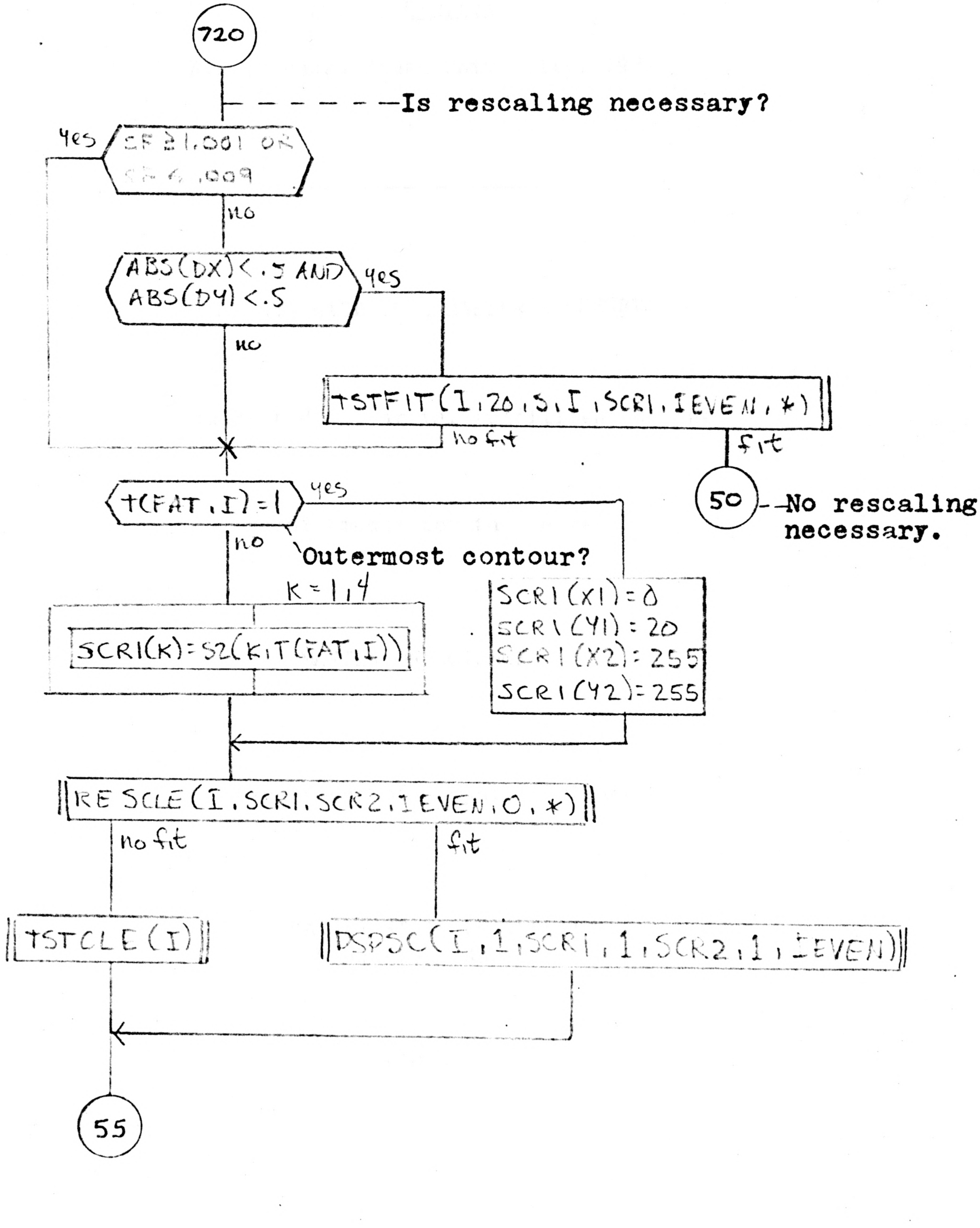
Rescale the contour if necessary.

Enter it in the PDF.

Save the free space.


TH
Get a node that was saved for processing.
NEXT = NEW
yes
no
RETURN
K = T(SON, NEWSON (NEXT))
IC = NEWSON (NEXT)
JJ = 1, 4
SCR3(JJ) = SVSCR(JJ, NEXT)
NEXT = NEXT + 1
TF

To link the scaling routines with the other routines, insert the following routine in the Create Contour routine. Other communication is handled through the previously described common blocks.



A PARTIAL IMPLIMENTATION OF THE CONTOUR MODEL OF BLOCK STRUCTURED PROCESSES

by

LYN D. LAVIANA

B.A., Kansas State University, 1972

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1974

The contour model developed by J. B. Johnston is a scheme for the graphic display of the dynamic execution of a program written in a block structured language. A natural extension of contour model idea is an interpreter to process block structured programs and to generate a dynamic contour model display on a vector graphic terminal. Such an interpreter would consist of two parts: a basic language interpreter and a set of basic contour construction, sizing, and display routines. This report documents an implementation of contour construction, sizing, and display routines. These are written in FORTRAN to drive a graphics package for the Computer 300 terminal. As implemented, interpretation of the program must be done manually; the contours are constructed using primitives such as "create contour", "create variable", "change variable", "create processor", etc. A sample display program is illustrated.