PRL

TECHNICAL NOTE

TN-86-52

COMPUTER AIDED LEARNING OF FLOWCHARTING

By
D.R. KULKARNI

JANUARY 1986

PHYSICAL RESEARCH LABORATORY
AHMEDABAD-9

DOCUMENT CONTROL AND DATA SHEET

1. Report No.	: PRL-TN-86-52	
2. Title and Subtitle : 'Computer aided learning of flowcharting'	3. Report Date	: January 1986
	4. Type of Report	: Technical Report
	5. Pages	: 30
	6. Price	: Unpriced
	7. No.of References:	-
8. Author(s)	: D.R. Kulkarni	
9. A. Purpose	: To describe the package for self-teaching of flowcharting	
B. Useful for	: One who wants to learn programming	
10. Originating Unit/ Division & Address	: Computer Centre Physical Research Laboratory Navrangpura,Ahmedabad 380 009	
11. Sponsoring Agency	: -	
12. Abstract	: A drill-and-practice package has been developed for teaching flowcharting. This package accepts the flowchart given by the user as input. It checks its syntex and also lets the user know if it is logically correct by producing the result for the data. The package allows on-line editing of the flow-chart if the user wants to modify it. The package has been operational at PRL Computer Centre and has been found to be quite useful for beginners in learning programming quickly.	
13. Keywords	: Computer programming, Flowcharting, Computed aided learning.	
14. Distribution Statement	: Distribution unlimited	
15. Security Classification	: Unclassified	

Computer-aided Learning Of Flowcharting

D.R.Kulkarni

ABSTRACT

A drill-and-practice, computed-aided learning package CALFLOW has been developed to study a flowcharting which is a pictorial way of developing the logic of a computer program. For a beginner who is absolutely new to a computer way of thinking, the flowcharting offers a basic understanding of how a machine is made to execute the steps involving the primitive control structures such as sequencing and selection, leading to a solution of a given problem. However, before he/she acquires this skill, one has to undergo a process of trial-and-error in which one draws and redraws number of flowcharts. Since the self-diognosis of the error is very crucial in the fast learning of flowcharting, the package can be gainfully used after a few hours of class-room teaching. The package accepts the flowchart drawn by the student as an input and gives him/her the final solution, alongwith the tracing of each step in the flowchart. The package has four parts. The first part allows the student to draw the flowchart on a graphics terminal. The second part redraws the same flowchart, if required, using the data file generated in the first part.

The third part allows the student to update interactively any block in the flowchart independently. The fourth one generates the FORTRAN program which can be executed to get the tracing and the final solution of the problem. Before the FORTRAN program is generated, the package, however, verifies if the flowchart is syntactically correct or not. If it is not correct, it points out explicitly all the syntax errors in the flowchart. The package has been developed on DEC-1091 system using PLOT-10 interactive graphic package.

I. INTRODUCTION :

A drill-and-practice, computed-aided learning (CAL) package CALFLOW has been developed to study a flowcharting which is a pictorial way of developing the logic of a computer program. For a beginner who is completely new to a computer way of thinking, the flowcharting offers a basic understanding of how a machine is made to execute the steps involving primitive control structures such as sequencing and selection, leading to a solution of a given problem. In fact to envisage the solution of a problem in terms of these control structures is essentially the art of computer programming. Another important constraint which a beginner has to keep in mind while developing the program logic is that the flowcharting steps should consist of only those actions and operations which a computer is capable of performing. These actions broadly include the simple input/output instructions, defining the variable through assignment and the evaluation of expressions involving only arithmetic and relational operators.

It has been observed that before a student develops enough skill for logic building, he has to put in great amount of efforts to orient himself to this novel way of thinking. Often he has to draw and redraw number of flowcharts and convince himself that they really depict

the computer way of thinking to obtain the correct solutions of the problems. It has been experienced that in this process of trial-and-error, the self-diognosis of the errors plays a very crucial role as it accelerates the learning process much faster than a direct suggestion from a teacher or a colleague. The package CALFLOW has been basically designed to help the students at this stage. It is expected that a student has already undergone a few hours (usually 3 to 4 hours) of class-room teaching to learn the rudiments of flowcharting before he can acrue the maximum advantages from this package.

The package CALFLOW accepts the flowchart drawn by the student as an input. With the help of this package the student can immediately find if the flowchart is both syntactically and logically correct or not. Further the package encourages the student to detect the logical errors without any external help.

To be precise the package CALFLOW helps the students in the following four ways.

i.) To a great extent it assists the student to draw interactively a syntactically correct flowchart. It means that the package will not even allow various blocks in the flowchart to be connected wrongly. It also indicates to him/her in the end if the flowchart is fully syntactically correct or not. The syntax errors are pointed out explicitly for correcting the flowchart easily.

ii) It provides the student a versatile, interactive update facility by which he can update (viz. insert, delete and modify) any block in the flowchart independently. It offers him the facility to redraw the same or the modified flowchart quickly without repeating the process of drawing anew.

iii) If the flowchart is syntactically correct, it quickly offers him the solution as he would have obtained manually using the flowchart for any desired data. This helps the student to judge for himself if the flowchart is logically correct or not.

iv.) If the flowchart is not logically correct, it supplies him the step-by-step tracing of the flowchart. It is a common knowledge that to obtain the complete tracing is usually a very cumbersome and tiresome task. The complete ready-made tracing enables him to detect the errors in the flowchart without any external help.

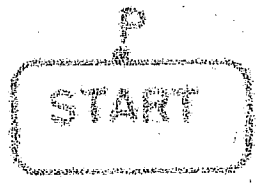
Thus the package CALFLOW assists the student to draw the correct flowchart, indicates to him if the flowchart is syntactically and logically correct and encourages him to detect the logical bugs by himself. It, thus, may prove to be a very useful tool in teaching flowcharting.

II. SYNTAX OF FLOWCHARTING :

In this section we describe the syntax considerations for drawing and connecting various blocks in the flowchart

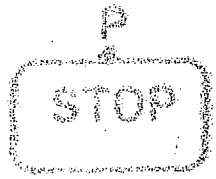
Fig. 1

1)



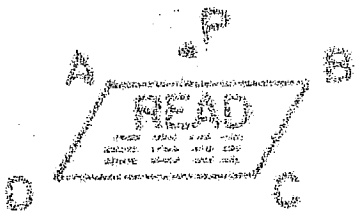
B block

2)



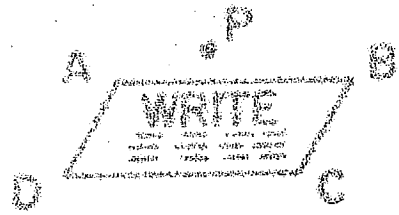
E block

3)



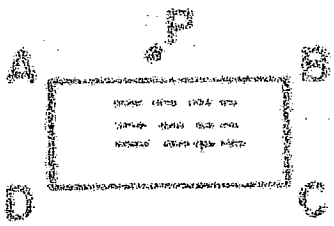
I block

4)



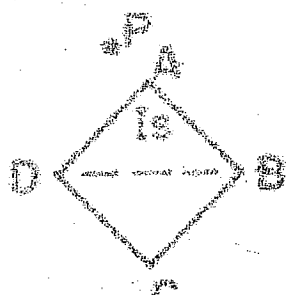
O block

5)



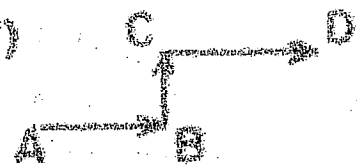
P block

6)



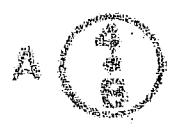
D block

7)



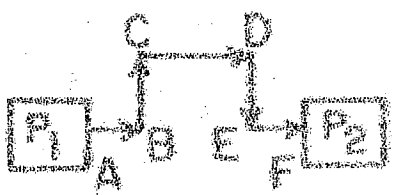
A block

8)



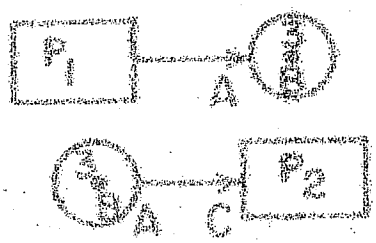
C block

9)



$P_1 - P_2$ joined with points.

10)



$P_1 - P_2$ joined with C blocks.

Fig. 1. Various symbols allowed in flowchart.

and their contents. The blocks to be included in the flowchart are shown in figure 1. It may be mentioned that while most of the syntax considerations are general some of them are enforced by our package.

1. Begin Block (B block) :

This should be the first block in every flowchart. There should not be more than one B block in the flowchart. It has a rectangular shape with curved edges and a word 'START' inscribed in it. The point P is the position with respect to which this block is drawn. The user has to provide this point before he/she would give command for drawing this block. This block indicates the beginning of the flowchart.

2. End Block (E block) :

There should be at least one E block in every flowchart. It indicates the logical end of your flowchart. There can be more than one E block in the flowchart implying those many points for logical termination of the flowchart. The E block has a shape similar to B block and has a word 'STOP' inscribed in it. The point P is the position with respect to which this block is drawn as in the case of B block. There can be maximum 5 E blocks in a given flowchart.

3. Input Block (I block) :

The input block is meant for accepting the data from the external source. Unless there is an external file present

containing the data, this block will not be executed. A flowchart may not have I block at all or it may contain number of I blocks depending on your problem. Its shape is a parallelogram with a slant on the right-hand side. A word 'READ' is inscribed in it. The variables to be read appear in the lines following this word. It contains maximum five lines, each of 30 characters. The variables in these lines are separated by commas. No continuation of the text is expected or implied from one line to the next line. In this sense they are independent. The points A, B, C, and D (see fig.1) should be provided by the user in this very order to cover the text so that they can form a parallelogram. The points are also provided to position the lines in the block. P is the point with respect to which the whole block is positioned. There can be maximum 5 input blocks in a given flowchart.

4. Output Block (O block) :

The output block is meant for outputting the result on an external device which should be available at the time of its execution. Generally one can not conceive a flowchart without any O block. Therefore one expects that at least one O block is present in the flowchart. Syntactically, however, one can draw a flowchart without any O block. Otherwise the description of O block is

exactly same as that of I block except that a word 'WRITE' is inscribed in it.

5. Process Block (P block) :

The process block in the flowchart is meant to store the result of the expression in a given memory location. In other words it is similar to an assignment statement in any programming language. Generally no meaningful flowchart can be drawn without using a single P block. However, syntactically such a flowchart is acceptable. We allow maximum 10 process blocks in the flowchart. Its shape is either rectangular or square. Its text consists of maximum 5 lines, each containing 50 characters. Each line accommodates one assignment statement using = (equal to) sign as an separator between the left-hand and the right-hand sides. The right-hand side can be any REAL or CHARACTER expression and the left-hand side can be any scalar/subscripted variable. The user has to position these lines appropriately along with the points A, B, C and D (strictly in this order--see fig.1) to cover the text to form a rectangle or square. P is the position with respect to which the process block is drawn.

6. Decision Block (D block) :

The decision block in the flowchart is the most important block as it decides the way program logic flows. It resembles the selection control structure of the type

IF-THEN-ELSE, having both true and false branches coming out explicitly. As in the case of P block, one can hardly draw a meaningful flowchart without having a single D block. We allow maximum 7 blocks in the flowchart. A decision block contains a single relational expression using the relational operators viz. .GT., .GE., .LT., .EQ., .NE. The result of any relational expression is either true or false. The expression consists of maximum 30 characters. The shape of the decision block is a vertical rhombus in which the points A and C appear on the same vertical line. A word 'IS' is inscribed in it. The user has to define the position of the expression as well as the points A, B, C and D (strictly in the same order--see fig.1) to cover the text of the block to form a vertical rhombus. The complete block is positioned with respect to a point P to be specified in the beginning.

7. Arrow (A block) :

The blocks described above are connected by arrows in the flowchart. These arrows indicate the sequence in which the blocks will be executed to lead us to final solution. Except the terminal blocks (ie. B and E blocks) all blocks in the flowchart should have one or more number of in-going arrows and only one out-coming arrow.However, the decision block has two out-coming arrows. The begin block will have no in-going and the end block will have no out-coming arrows. The arrow can be defined between two blocks in two ways.

i.) It can be defined by straight vertical and horizontal lines connecting maximum six points between two blocks as shown in figure 1.

ii.) It can be defined by using a connector block and the two-point arrow as shown in figure 1.

The connector block is simply a circle with a user-defined number inscribed in it. If we want to have arrow using connector blocks, we require to draw two parts.

i.) First part consists of arrow and the connector block to be defined using A and B (its centre) with a given number inscribed in it.

ii) Second part consists of the C block (with the same number inscribed in it) defined by points A and B and the arrow defined by the points A and C.

Thus block P1 is connected to a block P2. The package allows maximum 20 arrows in the flowchart.

8. Variables :

The package allows scalar variables as well as array variables having one or two dimensions. The variable can hold a numeric value of REAL type or it may hold a value of CHARACTER type. The sympolic name (identifier) for any variable is defined as a string of maximum six alpha-numeri characters with first character to be alphabetic. The identifier thus defined always corresponds to a REAL

variable. The identifier followed by a string '-CH' corresponds to a CHARACTER variable.

The one-dimensional array is read or written syntactically as

XARRAY(1..N)

where N can have maximum value 25. Note the two consecutive dots between the lower limit of the subscript which is always 1 and the upper limit of the subscript N which is either a pre-defined variable or a constant. Similarly the two-dimensional array is read/written as

MAT(1..ROW,1..COL)

where the variables ROW and COL can not have values more than 5. Note the presence of comma between the first and second subscript ranges. In the flowchart these variables are referred as the subscripted variables, the number of subscripts being one or two depending on the dimension of the array.

The package CALFLOW ensures that the flowchart is in conformity with the syntax described in the section. If not, it points out the blocks and text lines in which the errors are committed. It may be worth mentioning that some syntactic considerations such as maximum number of blocks allowed, maximum values permitted for each subscripts, maximum number of lines of text allowed etc. have been mainly enforced keeping in view the limited complexity

of the class-room problems and the small screen size of the graphic terminal. They can, however, be easily changed if required. In the next section we describe the broad features of the package CALFLOW.

III. GENERAL FEATURES OF CALFLOW :

The figures 2A and 2B describe completely how the package functions. The package consists of five important modules

a. Module DRWFLW

b. Module RDRWFL

c. Module MODFLW

d. Module GENPRG

e. Sub-module TSTPRG

In the following sections we describe each of these modules separately in details. In this section we will describe only the general features of the package.

Conventionally a flowchart need not declare explicitly the type of information it is processing. In our package, however, since the flowchart gets converted into a program to be executed, it becomes necessary that the flowchart should explicitly indicate the type of the information it wants to process. This practice in flowcharting imposed by the package directly leads to the good programming habit of declaring the types of variables explicitly.

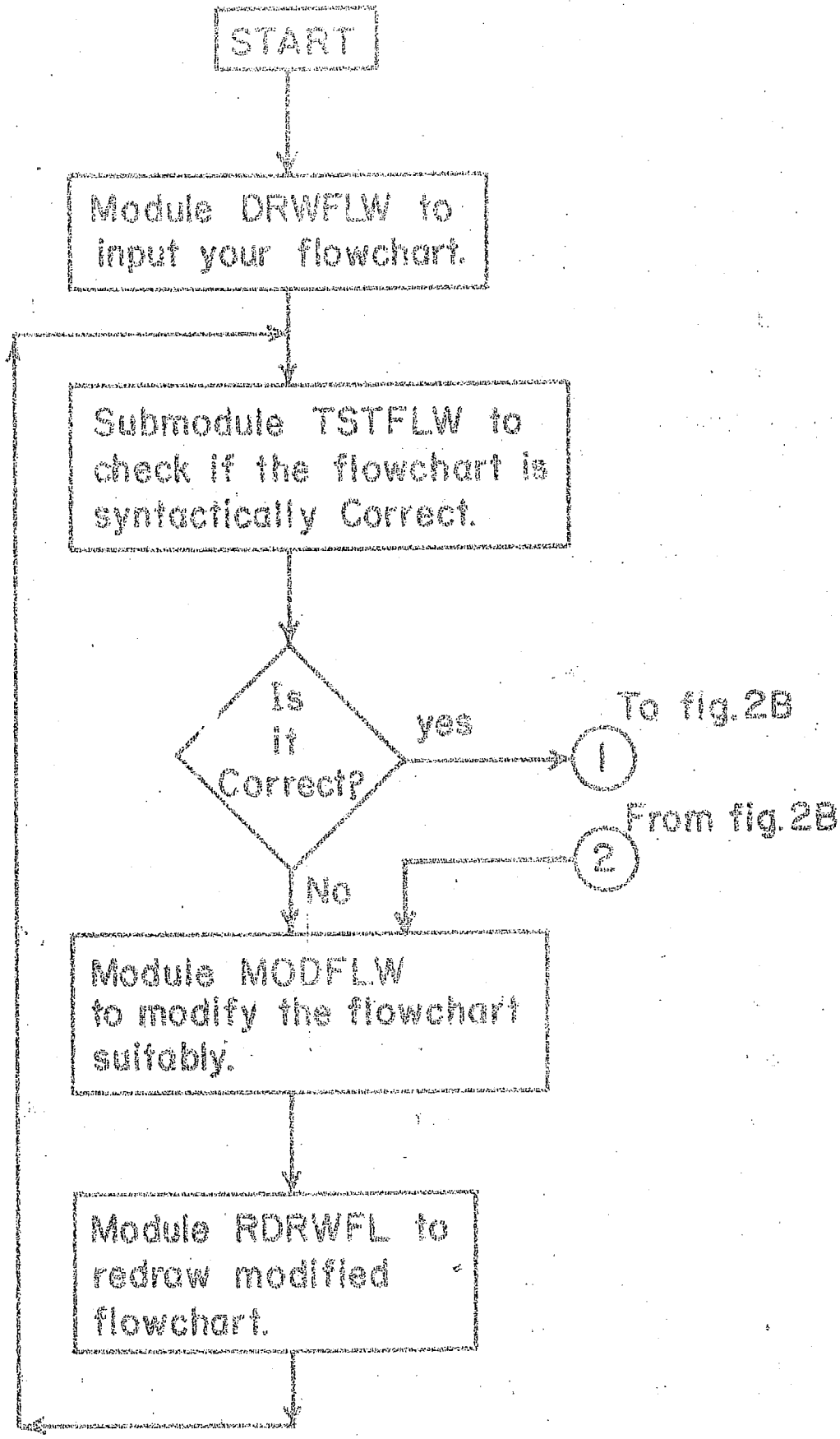


Fig.2A. This part of the package checks if the flowchart is syntactically Correct.

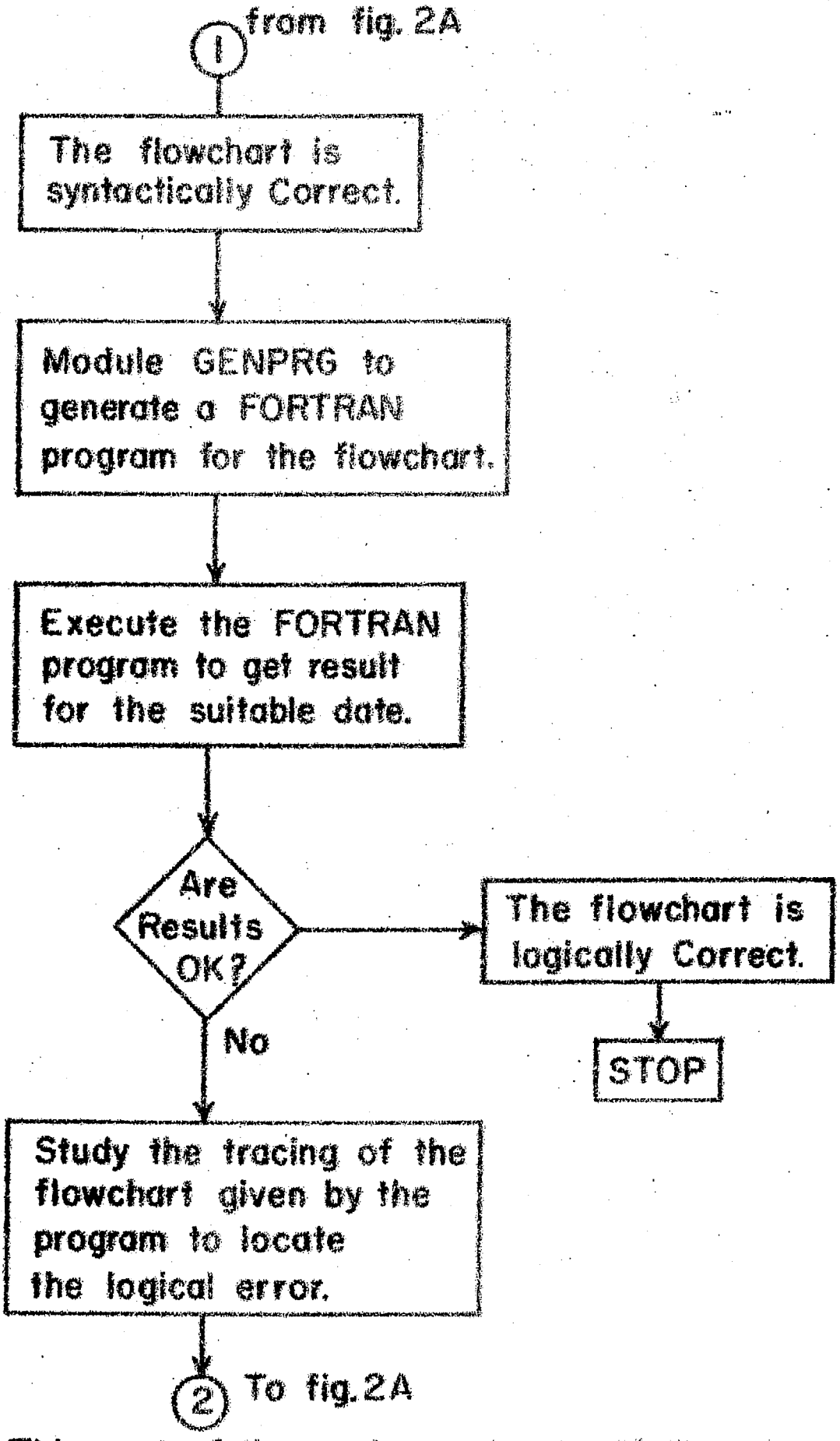


Fig. 2B. This part of the package checks if the flowchart is logically Correct.

The package permits the information of both REAL and CHARACTER types having maximum two-dimensional array structure.

Since this package converts the given flowchart into a FORTRAN program it is possible to use some pre-defined functions available in FORTRAN language as a part of the text in the flowchart. In fact all the functions which have arguments of type REAL and CHARACTER can be used in the text appearing in the flow-chart. This enables the user to generate simple mathematical tables using flowcharting even in the early stage of learning. This, in a way, enhances the enthusiasm of the beginner and instils in him the good habit of using the efficient pre-defined functions offered by the compiler.

The package has been implemented using two terminals viz. the graphic terminal (Tektronix 4012) and the video terminal (VT-100). The video terminal is used for inputting and outputting the non-graphic information and the graphic terminal is meant exclusively for graphic information only. The use of two terminals increases the operational ease considerably. This is particularly so as the tektronix terminal 4012 uses the storage tube and does not permit the selective erasure on the screen. It, however, allows the interactive graphic-input mode by displaying a cross-hair cursor on the screen.

The package CALFLOW functions in two phases. In the first phase the package ensures that the flowchart is fully correct syntactically. If the flowchart is wrong it will explicitly indicate the block and the text that may be in error. In the second phase it generates the program from the flowchart, which can be executed to get the results. These results will indicate to the user if the flowchart is logically correct or not. The program also generates the full step-by-step tracing of the flowchart to enable the user to detect the logical errors in case the results are wrong.

The package has been developed on the system DEC-10 using interactive graphics package PLOT-10 offered by Tektronix. It has been coded in FORTRAN-10, version-7 which is similar to FORTRAN-77. It consists of four main programs and about 82 user-defined subroutines. The total code consists of about 3600 lines.

IV. MODULE DRWFLW : This module does three jobs.

i. It helps the user to draw the flowchart on graphic terminal.

ii. It checks if the flowchart is syntactically correct or not using the subroutine TSTFLW.

iii. It creates the file X.FLW where X is any user-defined file name. It contains all the data pertaining to the

flowchart drawn on the screen.

When you execute this module it will ask you to specify the block type you want to draw. The blocks are specified by using the following codes.

B (begin block), E (end blocks), P (process block), C (connector block), A (arrow), I(input), O(output).

The code S is specified to terminate the process of drawing. It may be noted that the C block can not be drawn independently. It is drawn when two blocks are joined by arrows with C blocks as shown in fig.1. The module further wants you to confirm the block by pressing the return key once more. This may be necessary if you happen to press wrong key accidently. As each block is being drawn a label is generated by the module and is displayed along with the block. The label contains the block code and the number of its occurrence in the flowchart. For example the second process block in the flowchart is labelled as P2. These labels are used for updating the blocks and for drawing the arrows connecting them. Before drawing the blocks B, E, P, D, I, and O the package asks the user to position it on the screen using the cross-hair cursor. The data regarding the block such as the co-ordinates of the vertices, line positions etc. are stored relative to this point so that it can be positioned anywhere while redrawing. The point located on the screen using the cross-hair cursor is marked by a dot(.).

The package offers the number of ways for the user to cancel the block when it is still being drawn. This facility, in practice, has been proved to be quite useful especially for the beginners. Even the current line of the text in the block can be cancelled if required. All these facilities save lot of trouble of redrawing the figure again and again in case of minor mistakes committed during the process of drawing. The package interactively gives the appropriate instructions to guide the user to take proper actions.

While drawing arrow the package will not allow you to draw more than one arrow emnating from the same block (with the exception of D block which has two arrows emnating from it). It will also not allow any arrow to start from E block or to terminate on B block.

The terminal blocks (B or E) contain the fixed text and hence they are drawn automatically by the package with fixed dimensions. The other blocks (I, O, P and D) contains varying user-defined text. They are therefore, drawn by the user to cover the text appropriately.

It has been mentioned earlier that the information regarding positioning the blocks and the text therein has been stored in relative co-ordinates to facilitate the redrawing of these blocks. However, it, in turn,

implies that the arrows connecting them can not be of fixed length. Therefore the package does not store any information that is required for drawing the arrows. Consequently they have to be drawn explicitly at the time of redrawing after the blocks are repositioned.

The package, however, stores the information regarding arrows such as the starting and ending labels of the blocks and the code indicating the mode of drawing. The code P shows that the arrow be drawn defining the points alone and the code C shows that it be drawn using both points and C blocks. Further in the case of code C the first part starting from the block to C block is indicated by the code F and the second part starting from C block to other block by S code. Sometimes it may happen that only one part (usually the first one) may be needed. This information is obtained from the codes F or S.

In order to run this module two terminals are made available near each other. The graphic terminal is in 'SLAVE' mode and has been assigned to video terminal.

You can transfer your graphic output to it from the video terminal which is used for logging in and also for the conversation with the package. Then just type

DO DRWFLW[211,102] n

where n is the graphic terminal number you intend to use. The module will then help you to proceed further as described above.

Appendix I gives the list of all subroutines referenced by the module DRWFLW. The list excludes the graphic routine used from PLOT-10 package. Appendix II gives the list of monitor commands in the file DRWFLW.MIC used in DO command given above.

V. MODULE RDRWFL :

This module is used for redrawing the same or the modified flowchart using the file X.FLW created by the module DRWFLW. Before it draws the flowchart it indicates to the user if the flowchart is syntactically correct or not. The blocks in the file are drawn in the same sequence as given in the module DRWFLW. Before it draws a block the module will ask you to position it on the screen. It is relative to this point that the block is drawn. The blocks drawn by this module are E, B, I, O, P and D. The data for drawing the arrows connecting these blocks is not stored in the file X.FLW as explained in module DRWFLW. However, the file does contain the information regarding various arrows that exist and their mode of drawing. Since there is freedom for positioning the blocks the lengths of the arrows are likely to be changed. Hence they are to be drawn explicitly after the blocks are suitably positioned. Using this module one can reproduce the complex blocks in the flowchart quickly.

In order to use this module user just types

DO RDRWFL 211,102 n,flnm

where n is the graphic terminal number and flnm is the name of the file generated by the module DRWFLW. The name does not include the extension .FLW.

Appendix I gives the list of all subroutines (other than graphic: routines) referenced by the module RDRWFL. Appendix II describes the file RDRWFL.MIC used in the DO command given above.

VI.MODULE MODFLW :

This module is used to update the flowchart stored in the file X.FLW and to create another file MX.FLW containing the updated flowchart. X and MX are the names of the files before and after updating the flowchart. The module allows three update operations viz. i.) inserting (I) the block ii.) modifying (M) the block iii.) deleting (D) the block. The letters in the brackets indicate the valid update codes. The update code 'S' indicates that the updates are over and the updated file be created.

The module allows only the blocks I, O, P, and D containing the varying text to be modified. It permits only the blocks I, D, P, D, E, and A to be inserted or added. It can delete any block from the flowchart. While modifying or deleting the block it is identified by its

label. When you insert the block you have to specify only its type and the label of the block after which it is to be inserted. When you want to modify a particular block the module will help you to draw a modified block alone on the screen to replace the old one. When you want to delete a particular block the module will just keep a suitable flag against it to indicate that it is deleted.

When you want to insert a block you must be little more careful. For example if you want to insert a block I2 between the blocks P1 and P2 you must do as follows.

a.) Delete the arrow connecting the blocks P1 and P2.

b.) Insert the block I2 after the block P1.

c.) Insert the arrows between the blocks P1 & I2 and I2 & P2.

When you delete a block all the arrows starting and ending on it are deleted. These deletions further necessitate the changes in other blocks from which these arrows are starting or ending. Consequently the number of blocks which are properly connected to each other before deleting the block are now isolated and the flowchart becomes syntactically wrong. Hence the deletion of a single block may cause disturbance in many other blocks.

While executing the module it requires at the outset the input as the names of the file to be updated and the updated file. The module prevents any updates made on the

blocks which are not defined already. The label for the block type to be inserted is automatically given by the module. The module also ensures that the number of any type of blocks does not exceed its maximum value defined in the section II. It further checks if the total number of blocks in the flowchart exceeds 50.

In order to run this module just type

DO MODFLW 211, 102, n, flnm1, flnm2

where n is the graphic terminal number, flnm1 is the name of the file to be updated and flnm2 is the file name containing the updated flowchart. These names do not include the extension .FLW.

Appendix I gives the list of all non-graphic subroutine referenced by the module MODFLW. Appendix II describes the file MODFLW.MIC used in the DO command given above.

VII. MODULE GENPRG :

This module first checks if the flowchart is syntactically correct or not by using subroutine TSTFLW. If the flowchart is correct the module uses the flowchart file X.FLW to create a file X.FOR which contains the FORTRAN 77 program generated by the module. As stated before X is any user-defined valid filename. The program in the file X.FOR is executed for any desired data to get the results. From these results the user can judge for himself if the flowchart

is logically correct or not. In case it is not, the user can study the step-by-step tracing of the flowchart generated by the program to detect the logical bug. Thus the module enables the user to find if the flowchart is correct or not. It also encourages him to detect the errors by supplying him the complete tracing of all the flowcharting steps.

This module takes each block separately and simulates it by generating the suitable sequence of FORTRAN 77 statements for it. The executable block I, O, P and D start with a dummy statement CONTINUE and terminate with the GOTO statement to establish links with other blocks. The block E also starts with CONTINUE but ends with STOP statement. The CONTINUE statement in each type of block bears a pre-determined label for easy branching. The B block starts with PROGRAM statement followed by declaration of array and CHARACTER variables. These variables obtained by scanning the text in the blocks I, O and P are declared in the DIMENSION and CHARACTER statements immediately following the PROGRAM statement. Besides simulating each block the package generates some additional statements preceding every I and O block. These statements instruct the user regarding what input is expected and what output is generated by the program.

The program thus generated may give syntax error at the time of compilation if the expressions in the text are not written properly. There is also a minor restriction that the individual array elements are not read or written in the flowchart. It is the complete array that needs to be read or printed.

In order to run this module, type

DO GENPRG [211,102] flnm

where flnm is the name of the file containing the syntactically correct flowchart. The name does not include the extension .FLW. In order to get the hard copy of the results and the tracing of the flowchart, type

DO HDCOPY [211,102] flnm

where flnm is the name of the program file. This file has an extension .EXE which is not included in the name. This will create a file flnm.RES which may be printed to get a hard copy of the results and the tracing.

The appendix I gives the list of all the subroutines (excluding the graphic routines) referenced by the module GENPRG. The appendix II gives the monitor calls contained in the MIC files GENPRG.MIC and HDCOPY.MIC.

VIII. SUBROUTINE TSTFLW :

This subroutine checks if the flowchart fed by the user is syntactically correct and gives out the message

accordingly. This subroutine calls 15 other subroutines listed in the appendix I.

The subroutine checks if there is at least one E block denoting the logical termination of the flowchart. It also ensures that there is only one B block present in the flowchart. It verifies that the blocks P, I and O have at least one in-going arrow and only one out-coming arrow. The block D has only two out-coming arrows and at least one in-going arrow. There should be no in-going arrow for B block and no out-coming one from E block.

Besides verifying the general structure of the flowchart this subroutine also scans the text in the blocks I, O, P and D and validates it. In the blocks I and O it separates the variables from the text. For each variable it picks up the identifier and its dimensionality. The dimensionality is -1 for the literal constant, 0 for the scalar variable and 1 and 2 for one and two dimensional array variables. It also picks up the lower and upper values of each subscript range for all array variables in I and O blocks.

For P block it separates the right-hand side and the left-hand side of each line of text by searching for equal to (=) sign. It also identifies if there are any array variables on the left-hand side, which are not

noticed earlier. It checks that the parentheses are balanced on both the sides independently. However the expression on the right-hand side is not fully checked for its validity in this subroutine as an invalid expression invariably causes the error while compiling the generated program. The usual error that a beginner is apt to make is that he/she may not put the arithmetic operator explicitly in the expression. It is therefore recommended that a beginner should explicitly write all the operators in the expression which should contain enough number of parentheses to avoid any ambiguity.

For D block it separates the right-hand and left-hand operator. It verifies that the parentheses are balanced on both the sides independently.

Finally it checks if the shape of each block is according to syntax or not. If there is a considerable deviation from the prescribed shape it gives the message accordingly along with the statement describing the correct shape. The shape considerations are valid only for the blocks I, O, P and D, which a user draws on the screen to cover the varying text.

IX. CONCLUSIONS :

The package has been already operational at Physical Research Laboratory, Ahmedabad. It has been tested on a few students who were exposed to the discipline of programming

for the first time. After four hours of class-room teaching they were encouraged to use the package to test their flowcharts. After overcoming the initial fear and diffidence to work on the terminals the students could soon start using package effectively to test their flow-charts. They could decide for themselves if the flowchart is correct or not. The ready-made, step-by-step tracing has helped them to get the insight into their flowcharts and thus encouraged them to locate the logical errors that have crept in. It can thus be said that this experience has been very encouraging so far as the purpose of the package is concerned.

APPENDIX I

The list of all non-graphics routines used in the package CALFLOW.

A. MODULE DRWFLW

1) Main module DRWFLW

2) Sub. TBLK	3) Sub. DRAWT	4) sub. ABLK
5) sub. ARROW	6) sub. DRAWA	7) sub. CHECKB
8) sub. DRAWC	9) sub. CHKIN	10) sub. DRAWAP
11) sub. DRAWAC	12) sub. IOBLK	13) sub. COLIND
14) sub. PNTOK	15) sub. GETPNT	16) BLOCKDATA
17) sub. CHECKI	18) sub. CHECKO	19) sub. DRAWI
20) sub. DRAWO	21) sub. PBLK	22) sub. DRAWP
23) sub. CHECKP	24) sub. DBLK	25) sub. DRAWD
26) sub. CHECKD	27) sub. BACK	28) sub. PACK
29) sub. TSTFLW	30) sub. LOWTUP	

B. MODULE RDRWFL

1) Main program RDRWFL

2) sub. RDRAWT	3) sub. RDRAWP	4) sub. RDRAWD
5) sub. RDRAWI	6) sub. RDRAWO	7) sub. MDRAWA
8) sub. GETPNT	9) sub. DRAWT	10) sub. ARROW
11) sub. PNTOK	12) sub. MESDIR	13) sub. YESNO
14) sub. MDRAWP	15) sub. DRAWC	16) sub. TSTFLW
17) sub. LOWTUP		

C. MODULE MODFLW

1) Main program MODFLW

2) sub. FINDSQ	3) sub. MODIFY	4) sub. MODI
5) sub. MDRAWI	6) sub. COLIND	7) sub. PNTOK
8) sub. GETPNT	9) sub. MODO	10) sub. MDRAWO
11) sub. OKBLK	12) sub. MODP	13) sub. MDRAWP
14) sub. MODD	15) sub. MDRAWD	16) sub. DELETE
17) sub. IOCOD	18) sub. ADDI	19) sub. ADDO
20) sub. ADDP	21) sub. ADDD	22) sub. ADDE
23) sub. DRAWT	24) sub. CHECKB	25) sub. CHECKI
26) sub. CHECKO	27) sub. CHECKP	28) sub. CHECKD
29) sub. MABLK	20) sub. CHKIN	31) sub. GENAWC
32) sub. LOWTYP		

D. MODULE GENPRG

1) Main program GENPRG

2) sub. SIMB	3) sub. SIMI	4) sub. SIME
5) BLOCKDATA	6) sub. PACK	7) fun. VINIPT
8) sub. SIMP	9) sub. SIMD	10) sub. FNDBR
11) sub. LOWTUP	12) sub. TSTFLW	13) sub. SIMO

E. MODULE TSTFLW

1) Subroutine TSTFLW

2) sub. VLDFLW	3) sub. GETVAR	4) sub. VLDTP
5) sub. VLDTD	6) sub. SHAPE	7) sub. MPACK
8) sub. SEPVAR	9) sub. FINDIM	10) sub. GETPAR
11) sub. TSTVAR	12) fun. DIGIT	13) fun. ALPHA
14) sub. PACK	15) sub. VINTNP	16) sub. GANGLE

APPENDIX II

The list of monitor commands in various MIC files used in the package CALFLOW.

1) File DRWFLW.MIC

```
.DEAS
.ASS TTY 50
.ASS TTY'a TTY
.RUN DRWFLW [211,102]
```

2) File RDRWFL.MIC

```
.DEAS
.ASS TTY 50
.ASS TTY'a TTY
.RUN RDRWFL [211,102]
*'b
```

3) File MODFLW.MIC

```
.DEAS
.ASS TTY 50
.ASS TTY'a TTY
.RUN MODFLW [211,102]
*'b
*'c
```

4) File HDCOPY.MIC

```
.DEAS
. ASS TTY 50
.RUN 'a.EXE
```

5) File GENPRG.MIC

```
. DEAS
.ASS TTY 50
.ERROR s
.RUN GENPRG [211, 102]
*'a
.IF (NOERROR) .GOTO START1
!Error while running the program GENPRG
```

```
^Z
START1 ::.error ?
.DEAS
.ASS TTY 80
.LOAD 'a
.IF (noerror) .GOTO START2
!Error while loading the generated program
^Z
START2:: .SAVE 'a
.RUN 'a
```