DGCALC: A CALCULATION SPREADSHEET

FOR THE DATA GENERAL COMPUTER

Hong Zhou, Pei Di, and Donald O. Whittemore

Kansas Geological Survey
Lawrence, Kansas

Kansas Geological Survey Open-File Report No. 87-30

December, 1987

INTRODUCTION

The program DGCALC was written to be a calculation spreadsheet at a time when there was no spreadsheet available on the Data General computer. It was developed primarily by the student research assistants Zhou Hong and Di Pei under the direction of Donald O. Whittemore. The program was first used by Don Whittemore for research purposes in early 1986. The early versions of the program were written to have similar functions as the commercial program CALCSTAR. Development continued after first use and consisted of additional commands and improvements that allowed more flexible use on the Data General. Towards the end of 1987, the Kansas Geological Survey decided to purchase the commercial spreadsheet program MathPlan (PlanPerfect) which is compatible with WordPerfect, the word-processing program obtained a few months earlier for the Data General. MathPlan has most of the same type of functions and operations of DGCALC as well as additional capabilities that had not yet been written for DGCALC.

The spreadsheet operation and size of DGCALC were originally written primarily for handling water-quality data. Data for a water sample typically includes many variables, including information on the name, date, and location of the sample and values for the chemical properties and the concentrations of various chemical constituents. Several calculations are generally involved in processing of the chemical data such as conversions to other concentration units than reported in an analytical determination, computation of total-dissolved solids, charge balance error, hardness, sodium adsorption ratio, and ratios, and preparation of files for use in other computer programs. Thus, the column dimensions of the sheet must be large. The maximum size of the DGCALC sheet is 100 columns by 255 rows. Although the MathPlan sheet can handle up to 999 rows, it can only incorporate a maximum of 64 columns.

After the development of DGCALC had reached a point where we felt it was ready for use by others at the Survey, a copy was given to the Computer Services Section for evaluation. Other staff in the personnel office and the Geohydrology Section then began to use DGCALC. When the MathPlan spreadsheet became available, a procedure was developed for converting a DGCALC file into a MathPlan file and has now been incorporated into the present version of DGCALC. Although Don Whittemore is still using DGCALC for processing data already in DGCALC format, he is using MathPlan for new files. Documentation of the DGCALC program and the different components and subroutines involved in its operation are included in the following pages because the program itself or selected parts may be valuable for the development of other computer programs at the Survey. We note that the source code is available for the DGCALC spreadsheet but not for the MathPlan spreadsheet.

Kansas Geological Survey
Open-file Report

Disclaimer

The Kansas Geological Survey does not guarantee this document to be free from errors or inaccuracies and disclaims any responsibility or liability for interpretations based on data used in the production of this document or decisions based thereon. This report is intended to make results of research available at the earliest possible date, but is not intended to constitute final or formal publication.

DGCALC: A CALCULATION SPREADSHEET

FOR THE DATA GENERAL COMPUTER

Hong Zhou, Pei Di, and Donald O. Whittemore

Kansas Geological Survey
Lawrence, Kansas

Kansas Geological Survey Open-File Report No. 87-30

December, 1987

PRINTED IN U.S.A.

INTRODUCTION

The program DGCALC was written to be a calculation spreadsheet at a time when there was no spreadsheet available on the Data General computer. It was developed primarily by the student research assistants Zhou Hong and Di Pei under the direction of Donald O. Whittemore. The program was first used by Don Whittemore for research purposes in early 1986. The early versions of the program were written to have similar functions as the commercial program CALCSTAR. Development continued after first use and consisted of additional commands and improvements that allowed more flexible use on the Data General. Towards the end of 1987, the Kansas Geological Survey decided to purchase the commercial spreadsheet program MathPlan (PlanPerfect) which is compatible with WordPerfect, the word-processing program obtained a few months earlier for the Data General. MathPlan has most of the same type of functions and operations of DGCALC as well as additional capabilities that had not yet been written for DGCALC.

The spreadsheet operation and size of DGCALC were originally written primarily for handling water-quality data. Data for a water sample typically includes many variables, including information on the name, date, and location of the sample and values for the chemical properties and the concentrations of various chemical constituents. Several calculations are generally involved in processing of the chemical data such as conversions to other concentration units than reported in an analytical determination, computation of total-dissolved solids, charge balance error, hardness, sodium adsorption ratio, and ratios, and preparation of files for use in other computer programs. Thus, the column dimensions of the sheet must be large. The maximum size of the DGCALC sheet is 100 columns by 255 rows. Although the MathPlan sheet can handle up to 999 rows, it can only incorporate a maximum of 64 columns.

After the development of DGCALC had reached a point where we felt it was ready for use by others at the Survey, a copy was given to the Computer Services Section for evaluation. Other staff in the personnel office and the Geohydrology Section then began to use DGCALC. When the MathPlan spreadsheet became available, a procedure was developed for converting a DGCALC file into a MathPlan file and has now been incorporated into the present version of DGCALC. Although Don Whittemore is still using DGCALC for processing data already in DGCALC format, he is using MathPlan for new files. Documentation of the DGCALC program and the different components and subroutines involved in its operation are included in the following pages because the program itself or selected parts may be valuable for the development of other computer programs at the Survey. We note that the source code is available for the DGCALC spreadsheet but not for the MathPlan spreadsheet.

PRINTED IN U.S.A.

DGCALC DESCRIPTION AND COMMANDS

The following description and explanation of the capabilities and commands of the calculation spreadsheet DGCALC written for the Data General computer were taken from the subroutine DGCALC_HELP.F77 of the DGCALC program.

GENERAL DESCRIPTION

DGCALC is a calculation spreadsheet for the DATA GENERAL computer. It has 100 columns and 255 rows. The columns are labeled by A,B,C,...Y,Z,AA,AB,...CV. The rows are labeled by 1,2,...255. The location is specified by its row and column labels. Text, numeric data, or formulas can be entered or modified in a cell by moving the cursor to that cell and using the appropriate cell type command. The contents of a cell or a group of cells (window) can be copied, deleted, erased, saved, loaded, recalculated, moved, or printed. Columns or rows can be inserted or deleted. Column width and numeric precision can be changed.

COMMANDS TO MOVE THE CURSOR ON A DATA GENERAL TERMINAL

The cursor can be moved to any place of the spreadsheet by a cell location command, as well as by the following commands :

The ARROW keys move the cursor one position at a time
The SHIFT ARROW keys move the cursor one screen at a time
HOME : Move to the beginning of sheet (upper left-hand corner)
C1 : Move up to the top of sheet (in the same column)
C2 : Move down to the bottom of sheet (in the same column)
C3 : Move to the left end of sheet (in the same row)
C4 : Move to the right end of sheet (in the same row)*)

SPREADSHEET OPERATION COMMANDS

The following commands for the operation of the spreadsheet appear at the bottom of the screen when the ESC key is pressed:

Type (ESC) to resume the spreadsheet.
C)opy D)elete E)rase F)orm G)oto H)elp I)nsert J)oin L)oad
M)ove Q)uit R)ecalc S)ave T)ype $)CLI Command

A command is invoked by pressing the first letter of the command (the capitol letter before the parenthesis) or the $ symbol for temporarily returning to the CLI level. To obtain more information about a command, type H for the HELP command. Then select the letter of the command for which an explanation is desired.

COPY COMMAND

The Copy command copies the contents of one cell or a window, to another window. After the command is typed, the following appears:

C)ell or W)indow

Type C to copy a cell to a window and type W to copy a window to another window. If the copied cell contents are formula type, the following appears:

R)elative or N)o adjustment

Type R to copy a math formula from the source cell to the same column. This command also changes the variable names (cell location of values used in the formula) to the proper row labels thereby adjusting the equation for each cell location so that it uses the correct input data in each row.

Type N to copy the same contents to the destination window. The window is specified by its top left and bottom right corners. To copy from a window to another window, be sure there is no overlap between source and destination windows. The syntax of the window is:

Top left corner: [Cell location of top left corner :]
Bottom right corner: [Cell location of bottom right corner]

For example, to copy the contents of cell D5 into a window with the location G10 for the top left corner and K20 for the bottom right corner, type:

Copy from: [D5]
Top left corner: [G10]
Bottom right corner: [K20]

DELETE COMMAND

The Delete command deletes (removes from the spreadsheet) rows or columns of the working sheet. After the command is typed, the following appears:

R)ow C)olumn

Type C to delete a column or columns. Then answer the question:

How many columns (0-9) ?

Type R to delete a row or rows, then answer the question:

How many rows (0-9) ?

Type 0 to delete nothing, type 1 to delete the current column or row, type 9 to delete 9 columns or rows starting from the current one. Rows are counted from the current row to the bottom. Columns are counted from the current column to the right.

ERASE COMMAND

The Erase command can erase the whole sheet, a window, a row, a column, or an entry, while retaining the window, row, column, or entry as a blank cell or cells. After the command is typed, the following appears :

A)ll W)indow R)ow C)olumn E)ntry

Type A to erase the whole sheet, W to erase a window, R to erase the current row, C to erase the current column, and E to erase the entry. When erasing the whole sheet, the following question appears:

Are you sure ?

Type Y to confirm this command or type N to abort.

Erasing rows or columns leaves the erased locations blank, whereas deleting removes the contents and moves the following rows or columns to the deleted locations. When erasing a window, the top left corner and bottom right corner must be specified. When erasing an entry, the current cell content is erased. For more information on each of the Erase commands type the first letter of the following items:

A)ll W)indow R)ow C)olumn E)ntry

The All command will erase the whole working sheet. Type Y to confirm this command or N to abort.

The Window command will erase a portion of the sheet. The window to be erased is specified by its top left corner and bottom right corner. The size of this window can be as large as the whole sheet or can be as small as one cell. An error message will be displayed and nothing erased if invalid window locations are typed.

The Row command will erase the current row in the sheet. The current row is the row where the cursor is located. The difference between the Delete Row and Erase Row commands is that when deleting, no blank space is left in the deleted area, while when erasing, the erased area is left blank.

The Column command will erase the current column in the sheet. The current column is the column where the cursor is located. The difference between the Delete Column and Erase Column commands is that when when deleting, no blank space is left in the deleted area, while when erasing, the erased area is left blank.

The Entry command will only erase the entry of the sheet. The entry is the cell where the cursor is currently located.

FORM COMMAND

The Form command changes the form of the entry, i.e., the precision of real numbers or the width of a column or columns. After entering this command, the following appears :

P)recision W)idth

Type P to change the precision of the whole sheet, a row, a column, or only one cell. The precision, or the number of figures following the decimal point in a real number, can be changed at this level. The number of digits after the decimal point is specified by typing a number from 0 to 9. After the number is chosen, type A)ll to change the precision for the whole sheet, R)ow for the current row, C)olumn for the current column, or E)ntry for the current cell.

Type W to change the width of all columns or of the current column. The width of any column can be selected as from 3 to 75 characters by typing an integer number. If the number selected is less than 3 or greater than 75, an error message will be printed and no changes will be made. After the number is typed, enter A)ll to change the width of all columns or C)olumn to change the width of the current column.

GOTO COMMAND

The Goto command moves the cursor to any cell location in the spreadsheet. The location is determined by the column and row labels. The column label is from A,B,C,...AA,AB... to CV. The row label is from 1,2,3,... to 255. The syntax of the response needed for the Goto Command is:

Cell Name: [Destination Cell Label]

For example, to move the cursor to the position with the column label AB and the row label 50, type:

Cell name: [AB50] or [ab50]

HELP COMMAND

The Help command can be used to obtain help messages for all commands in DGCALC. When in Help mode, read the menu on the monitor and type one desired command. The monitor will print out the corresponding messages about that command. When the monitor prints:

Type one command for more information

followed by the list of commands, type the first letter of the desired command. Type (ESC) to resume the spreadsheet in any selection in help mode.

INSERT COMMAND

The Insert command inserts blank columns or rows into the sheet. After entering this command, the following appears:

R)ow or C)olumn

Type R to insert rows above the current row. Type C to insert columns to the left of the current column. The number of rows or columns can be chosen by answering the following questions with the desired number of rows or columns:

How many rows (0-9) ? or
How many columns (0-9) ?

Typing 0 inserts no rows or columns and is the same as aborting this selection.

JOIN COMMAND

The Join command will load another DGCALC file while retaining the working file. The second file can be placed at any location in the working sheet. The location where the joined file will be placed is specified by where the top left corner of the second file should appear on the spreadsheet. The location is selected by answering the question for the top left corner. The DGCALC file to be joined must be a file previously saved by using DGCALC. The differences between the Join and Load commands are:

1. The Load command erases the whole sheet, loads a new DGCALC file, and places it at the start location A1.
2. The Join command does not erase the sheet, but loads a DGCALC file and places it at any start location. The DGCALC file can be another working sheet which was previously saved by DGCALC.

With the Save Window command, DGCALC can split a working sheet into several sheets. With the Join command, DGCALC can combine several sheets to form a new sheet.

LOAD COMMAND

The Load command reads an input file into the spreadsheet. The Load command erases the working sheet, reads a new DGCALC file, and places it into the start location (A1) of the spreadsheet. The user needs to type an input file name, then type return. The file name cannot be longer than 20 characters. The syntax is:

File to read from: [PATHNAME]:[FILENAME]

A pathname may be required before the file name if the file is in a different directory. If there is no such file name, the following message appears:

The file does not exist !

If an invalid file name is typed, the following message appears:

File is not a DGCALC file !

In both of the above situations, the user is returned to the spreadsheet. For more information, the P)athname or F)ilename descriptions can be displayed under the L)oad explanation in the Help messages.

MOVE COMMAND

The Move command moves one window to another window location. The source window is specified by its top left corner and bottom right corner and the target window by its top left corner. Be sure there is no overlap between the source and the target window. The window boundary cannot exceed the limits of DGCALC (100 rows and 255 columns).

For example, to move a window with the top left corner B3 and the bottom right corner of C10 to the target window with the top left corner D20, the syntax is:

Top left corner: [B3]
Bottom right corner: [C10]
Top left target corner: [D20]

QUIT COMMAND

Type Q to quit the program when in command mode. This command returns the user to the CLI level without saving the contents of the spreadsheet. Be sure to use the Save or Print commands to save part or all of the spreadsheet to the desired format.

RECALCULATE COMMAND

The Recalculate command recalculates the results of formulas in the spreadsheet. The spreadsheet only displays the results calculated the first time a formula is entered. Because a cell with a formula is a function of the other cells addressed by the formula and DGCALC does not recalculate the sheet automatically, the user needs to Recalculate the spreadsheet when new numbers are entered into cells used in formulas, otherwise the results may not be correct. Thus, this command should be used after updating the values of cells used in formulas. The cells in which formulas will be recalculated can be selected by either the E)ntry, W)indow, or S)heet commands.

SAVE COMMAND

The Save command saves the information of the working sheet to a disk file. After the command is typed, the following appears:

D)GCALC P)rintable (data), or T)ernary file

Type D to save to a DGCALC formatted file, which can be retrieved with the Load

or Join commands, P to save to a file which can be printed with the Data General QPRINT command or to convert a file to a MathPlan (PlanPerfect) file, and T to save numerical data to a special file for the program TERNARY. A file name, no more than 20 characters, is needed after typing all options. For more information, select the D)GCALC, P)rintable (data), or T)ernary commands under the S)ave command in the Help messages.

The DGCALC Save option will save to a file which has the DGCALC format. This file can be retrieved with the Load or Join commands in DGCALC. After this option is chosen, the following appears:

S)heet or W)indow

Type S to save the whole sheet and W to save a window to a DGCALC file. A file name is needed after this command is chosen. The extension .DG is added to the file name automatically. If saving a window, the window location must be specified by its top left and bottom right corners. No blank areas in the sheet can be saved. If an invalid window location is chosen, an error message will appear and nothing will be saved. The following are examples of invalid window locations:

The column label is out of the range A to CV (1 to 100).
The row label is out of the range 1 to 255.
The left column label is greater than the right column label.
The top row label is greater then the bottom row label.

The Printable Save command saves the contents of the spreadsheet that are shown on the screen to a data file that can be used as any other text (ASCII) file on the Data General computer. The user can print this file with the QPRINT command at the CLI level. After the command is typed, the following appears:

S)heet W)indow or D)ump

Type S to save the whole sheet and type W to save a window to a printable file. If the Window option is selected, the user needs to determine the window location by its top left and bottom right corners. Note that the width of the spreadsheet must be small enough to print on the computer paper. The maximum number of characters that can be printed on a wide computer sheet is 132. A file name, up to 20 letters, must be typed after S or W is entered.

After the file name has been entered, the following will appear:

R)egular or M)athPlan

Select R to print a regular ASCII file or M to convert the contents of the sheet or window to be saved to a MathPlan (PlanPerfect) spreadsheet file on the Data General computer. The character "\" must be placed between each cell of the sheet or window of information to be saved before the conversion can be made. After either the R or M is typed, the following will appear:

Title (1)

Type a title that will appear on the first line at the beginning of the printable file or the MathPlan spreadsheet, then press the return key. If no

title is desired, type return. After the first line of the title is entered, a second and a third line of the title can be typed. Each line of the title can be no more than 100 characters.

The Dump file command saves the whole spreadsheet contents and information describing the location and data type of all cells containing some contents. This command dumps the information for the whole sheet into a printable file. The user needs to type a dump file name of no longer than 20 characters or type RETURN for the default name [PrintOut]. An extension [.DUMP] is added after the file name automatically. The dump file contains all information about every cell in the spreadsheet, including cell location, cell type, and cell contents. The file is printed in the order from A1 to A255, B1 to B255 and so on. Blank cells are not saved. This file can be printed using the QPRINT command at the CLI level.

The Ternary Save command saves numerical data for the program TERNARY on the Data General computer which generates a ternary diagram using the saved data file as input. The window of data to be saved must be specified by its top left corner and bottom right corner. Only numerical data can be used as valid input for the TERNARY program. The data window must be three columns wide and no more than 100 rows. The TERNARY program can plot up to 100 points that are individually identified on a ternary diagram. The program TERNARY is also available on an IBM PC or compatible microcomputer. To save data for transfer to a microcomputer, use the Window option under the Print Save command. Use the same type of data window size (three columns by no more than 100 rows).

TYPE COMMAND

The Type command is used to change a numerical value in a cell to numerical characters (interpreted as text) or vice versa. After the command is typed, the following appears:

Cell Type: N)umeric T)ext

Type N to change numerical characters to a real number that is right justified in the cell on the monitor. Type T to change a numerical value, integer or real, to equivalent numerical characters (interpreted as text) that are left justified on the monitor. Typing N if the cell is of numerical type or typing T if the cell is of text type will not change anything. Only valid numerical numbers can be changed with this command.

Valid numerical data must be integers or real numbers. Real numbers must be floating point. Exponents are invalid as real numbers. A minus sign must be the first character of a negative number. A plus sign is optional for a positive number. At least one digit, either 0, a munus sign, or other digit from 1 to 9, must be on the left of decimal point.

VALID NUMBERS		INVALID NUMBERS
1234	(Integer)	.1234
1234.	(Real)	1.234E5
-0.123	(Real)	-.123
0.1234	(Real)	
12.34	(Real)	

A mathematical formula is another type of data used in DGCALC. It must be in a special format, otherwise it does not work correctly. Mathematical formulas can be typed into the spreadsheet using the format described below:

1. Any formula or expression must start with a [#] sign.
   Any function must start with a [@] sign and all arguments must be in parentheses.

2. The valid operators are:

+	: Addition	-	: Subtruction
*	: Multiplication	/	: Division
%	: Percentage	**	: Exponent
( )	: Parentheses	<76>	: Range of variable names

3. The valid built in functions in DGCALC are:

@SIN : Sine	@COS : Cosine
@TAN : Tangent	@LOG : Logarithm

   @MIN : Return the smallest value of a range of cells.
   @MAX : Return the largest value of a range of cells.
   @SUM : The summation of a range of cells.

The following are syntax examples of math formulas:

1. Math expression without variable names:
   #1+123*23-3**2/2

2. Math expression with variable names:
   #A1+C3/4+G8*5 A1, C3, and G8 must be already defined.

3. Math expression with functions:
   #@SUM(A1<76>A50)/@MIN(A1<76>A50)

4. Other math expressions:
   #@MAX(A1<76>AK1)*((12+2)**2+3) #123+456+789

Note that the contents of the cells addressed by the variable names must bo of numeric type. The variable names can be any cells in which numeric cata has previously been entered. If the relative copy command is to be used, all of the cell labels used as variable names are incremented one row at a time.

The mathematical operators that can be used are +, -, *, /, **, ( ), and %. The have the same meaning and priority as standard in computer mathematics. Standard functions have the highest priority and addition and subtraction have the lowest. The operators which have the same priority are used from left to right. The < symbol is used to indicate that a continucus range of cells is to be used in a row or a column for the calculation. Thus, make sure that either

PRINTED IN U.S.A.

the row or the column label is the same in the variable names on both sides of the < symbol.

There are several built in functions in the spreadsheet:

@SIN : Sine	@COS : Cosine
@TAN : Tangent	@LOG : Logarithm with base 10

@MIN : Return the smallest value of a range of cells.
@MAX : Return the largest value of a range of cells.
@SUM : The summation of a range of cells.

The first character of any function must be a @ sign. The trigonometric functions have the same meaning as standard in computer mathematics. The unit for trigonometric functions must be in RADIUS.

$ (CLI) COMMAND

When the "$" symbol is typed, the operation is temporary changed from DGCALC to the CLI level of the Data General. The user can then type and execute any CLI commands as long as (ESC) is not typed. Typing (ESC) returns the operation to the DGCALC spreadsheet. Note that typing BYE will not sign the user off the computer when in this temporary escape mode; the user must first return to and quit DGCALC.

For more information, please type M)ore about CLI commands under the $ (CLI) option under the Help messages or read the CLI manual. When M is typed, DGCALC will call a subroutine which is equivalent to the Help command at the Data General CLI level.

```
+3333333333
```

This documentation describes the organization of DGCALC.

The following are processes to create and execute DGCALC:
1. Compile each DGCALC F77 file to create DGCALC OB files with the
   following command:
)F77 FileName
2. Link DGCALC files to create DGCALC.PR with the following command:
)F77LINK dgcalc dgcalc_formula dgcalc_commands dgcalc_help
         dgcalc_utility edit lib; del dgcalc.st
3. Execute DGCALC with the following command:
)X DGCALC

The following summarizes the names of procedures and functions in
the order as they are in the source program.

The whole DGCALC has following files :

Documentation

1.	DGINTRO.WP	Introduction to DGCALC
2.	DGDOC.WP	Description of DGCALC commands
3.	DGCALC.ORGANIZE	This file
4.	DGCALC.STRUCTURE	Statement about DGCALC structure

Programs

5.	DGCALC.F77	DGCALC main program
6.	DGCALC_COMMANDS.f77	commands subroutines
7.	DGCALC_FORMULA.f77	mathematical subroutines
8.	DGCALC_HELP.F77	help messages
9.	DGCALC_UTILITY.F77	utility subroutines
10.	LIB.F77	other subroutines
11.	EDIT.F77	edit line subroutines

Variables and parameters

12. DGCALC.GLOBAL
13. DGCALC.GLOBAL1
14. DGCALC.PARAMS
15. DGCALC.CELLTYPE

16.	DGCONV.F77	Separate program to convert E_D_Prog data to DGCALC format data

1. DGINTRO.WP
   Introductory description of DGCALC

2. DGDOC.WP
   Documentation and description of DGCALC operation, commands, and
   functions.

3. DGCALC.STRUCTURE
   Documentation about the data structure of DGCALC,
   including HEADER, FORMULA ARRAY, CELL TYPE, POINTERS,
   and other information about the structure of DGCALC.

4. DGCALC.ORGANIZE (This file)
   List of file names in the sequence that they are in the DGCALC
   source file.

5. DGCALC.F77
   Main program of DGCALC.

   MAIN PROGRAM
   CloseFiles
   MOVE (X, Y)
   MoveRightEnd

```
        MoveDownEnd
        InitialSheet
        InterpretCell ( X, Y, BackGround, Buffered )
        GetUserInput ( Command, CellInfo, CharsIn )
        PrintFormula ( Pointer )  : Print the formula if formula cell
        integer function CommandType ( Ch )
        GetCommand ( Command )
        GotoLoc
        InputCellName ( Name, Message )
        integer function GetWide ( )
        RedrawCursor ( NewX, NewY )
        DisplaySheet
        ChangeFormat
        ChangePrecision
        SetNewWidth
        UpdateCell ( CellContents, Length )
        DataToArray ( CellContents, Length, type, SigificalnFig )
        CopyCell ( Marker, Index )
        CompressData
        CellType ( Cell, Length, Type, SignificalnFig )
        ChangeCellType

    6.  DGCALC_COMMANDS.F77

        Insert                  : Insert rows or columns
        InputColumn             : Input number of columns to insert
        InputRow                : Input number of rows to insert
        InsertRow(Index,Number)     : Insert number of rows
        InsertColumn(Index,Number) : Ins ert number of columns
        Delete                  : Delete current row or column
        CheckEmpty              : Check if the sheet is empty
        Erase                   : Erase sheet, widow, rows, columns, or entry
        ClearCell ( Row, Column )
        DeleteRow
        DeleteColumn
        Recalculate
        CopyToCells
        CopyTheArea (X1,Y1,X2,Y2,Valid)
        CopySame ( Index )
        CopyRelative ( Index )
        MarkFree ( X, Y )
        BlankCell ( Row, Column )
        ClearSheet          : Clear the sheet
        WndwLocation(X1,Y1,X2,Y2,Valid)
                            : Get window location
        SaveCommands
        PSaveSheet(FileName, Valid)  : Printable sheet save
        PSaveWindow(FileName,Valid)  : Printable window save
        DMPFileSave(FileName,Valid)  : Dump file save
        DGSaveWindow(FileName,VAlid) : DGCALC window save
        DGSaveSheet (FileName,Valid) : DGCALC sheet save
        TernarySave (FileName,Valid) : Ternary data save
        LoadSheet           : Load sheet from input file
        JoinFile            : Join other file into DGCALC
        DoCliCommand        : Execute CLI commands
        IntToASCII ( Number, ASCII ) : Convert a number into ASCII code
        MoveArea            : Move a widow to another place

    7.  DGCALC_FORMULA.F77
        Handle mathemetical data.

        Push ( ValType )
        real function PopReal ( ErrorFlag )
```

```
      integer function PopInt ( )
      logical function PopBool ( ErrorFlag )
      charcter*240 function PopString ( ErrorFlag )
      InitResWords
      InitScalars
      Error ( Fault )
      GetCurrChar
      GetNumber
      GetIndent
      SkipComment
      GetConst
      logical function Compare ( Key, ID, Len )
      ProcessIndent
      ReturnTK ( TK, Size )
      GetToken
      GetCellValue
      Evaluation ( TermType, TokenList, RealList, ErrorFlag )
      logical function LegalExpr ( List )
      Transformation ( ErrorFlag )
      GetCellNumber ( CellName, Error )
      logical function SysFunction ( Term )
      GetSumMaxMin ( OPE, Error )

8.    DGCALC_HELP.F77
      Print help messages on the monitor.

      Help
      DGHelp ( Command )
      CHelp     ( Copy Help )
      DHelp     ( Delete Help )
      EHelp     ( Erase Help )
        AEHelp  ( All Erase )
        WEHelp  ( Window Erase )
        REHelp  ( Row Erase )
        CEHelp  ( Column Erase )
        EEHelp  ( Entry Erase )
      FHelp     ( Form Help)
        PForm   ( Precision Form )
        WForm   ( Width Form )
      GHelp     ( Goto Help )
      HHelp     ( Help Help )
      IHelp     ( Insert Help )
      JHelp     ( Join Help )
      LHelp     ( Load Help )
      MHelp     ( Move Help )
      PHelp     ( Print Help )
        SPHelp  ( Sheet Print )
        WPHelp  ( Window Print )
        DPHelp  ( Dump Print )
      QHelp     ( Quit Help )
      RHelp     (Recalculate Help )
      SHelp     ( Save Help )
        DGSaveHelp ( DGCALC file Save Help )
          SSHelp   ( Sheet Save Help )
          WSHelp   ( Window Save Help )
        PrintHelp  ( Printable Save Help )
        TSHelp     ( Ternary Save Help )
      THelp     ( Type of data Help )
        FType      ( Formula Type )
          SynFormula ( Syntax Formula Help )
          OpFormula  ( Operator Formula Help )
          FunFormula ( Function Formular Help )
        NType      ( Numerical Type )
```

FORM 1411 5-83
PRINTED IN U.S.A.

```
      TType       ( Text Type )
    DollarHelp  ( Dollar Sign Help )
    RetMess     ( Print return message )
    TypeAny     ( Type any key to continue )

9.  DGCALC_UTILITY.F77
    Utility sobroutines used by DGCALC

    PrintTopScale
    ClrEol : Clear End of Line
    UnderLine
    Normal
    Inverse
    character*2 function AConvert (Index)
                :Convert column index to characters
    PrintLeft ( Number )
    PrintMessArea ( Message )
    logical function LegalName ( CellName )
    logical function LegalNumber ( CellNumber )
    ValidCell ( CellName, X, Y, Valid )
    SkipSet ( Cell, Length, NoSkip, Chset, Index )
    PrintMess ( Message )  : Message is a string of any length
    ErrorMess ( Message )  : Same above except ring the bell
    WriteIt ( Data, Max, Width, Just, Buff )
    WriteData ( X, Y, Buffered )
    FlushBuffer
    WriteBuf ( Data, Max )
    PosBuf ( Row, Column )
    InverseBuf
    NormalBuf
    RealToASCii ( ASCii, Number, ErrorFlag )
    character*6 function Position ( X, Y )

10. LIB.F77
    Library subroutines.

    AssignS ( OutString, Instring )
    Left ( Source, Destination, Length )
    integer function Find (Source, Patern, Start )
    Concat ( First, Second )  : Concatinate the second to the first
    Right ( Source, Destination, Length )
    WString ( Source, FileID )
    NewLine ( FileID )
    Mid (Source,Destrination,Position,Length)
    integer function SLen ( Source )  : String Length
    RString ( Destination, InFileID ) : Read String
    OpenF ( FChannel, FName, FAccess, IError )
    Pos ( X, Y, FileID )           : Position curssor on the screen
    CLS ( FileID )                 : Clear Screen
    CLI ( CommandString, Result )  : Do CLI command
    KLI ( CommandString, Result )
    character*1 function UpperCase ( LowerCase )

11. EDIT.F77
    Edit subroutines.

    Edit (FOut,FIn,Line,CharsIn,Length,Int)
       :Return the codes typed from the key board.
        If it is a text character string followed by a
        catriage return, Line is the content of string,
        Length is the length of the string, and Int = .false.
        If a funtion key is typed, Int = .true. and the code
        of that function key is assigned to the first charater
```

of the string.
This routing is different for the KGS version and
Whittemore version since different control codes
are used for the cursor movement.

```
CtrlChar (Ch,Line,Place,Y,X,Length)
DelChar ( Line, Place, Y, X )
InsertChar ( Line, Place, Y, X, Ch, CharLength )
character*1 function GetChar ( FileID )
GetPos ( FileOut, FileIn, X, Y )
IntToChars ( Number, Q, H, L )
```

12. DGCALC.GLOBAL
    Global variables for DGCALC.

13. DGCALC.GLOBAL1
    Global variables for DGCALC.

14. DGCALC.PARAMS
    Global parameters for DGCALC.

15. DGCALC.CELLTYPE
    Global CellType delaration of DGCALC.

16. DGCONV.F77
    A program to convert data, created by D_E_Prog, to
    a file which can be read by the DGCALC Load or
    Jion command.

PRINTED IN U.S.A.

+7777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777+

$$$$ $$$$ $ $ $ $ $$$$$ $$$$$

$ $ $ $ $ $ $ $ $ $

$ $ $ $ $ $ $ $ $ $

$ $ $ $ $ $ $ $$$$$ $ $

$ $$ $ $$ $ $ $ $ $ $ $

$ $ $ $ $$ $$ $ $ $ $

 $$$$ $$$$ $ $ $ $ $$$$$ $

DEST=GGWHIT USER=GGWHIT QUEUE=LPT DEVICE=@LPB

SEQ=21373 QPRI=137 LPP=66 CPL=132 COPIES=1 LIMIT=6

CREATED: 4-JAN-88 10:05:56

ENQUEUED: 4-JAN-88 10:14:18

PRINTING: 4-JAN-88 10:14:47

PATH=:SI20:GGWHIT:$DGCALC:DGCALC.STRUCTURE

$$$$ $$$$ $$$ $ $ $$$ $$$ $$$$$ $$$$ $ $ $$$ $$$$$ $ $ $$$$ $$$$$

$ $

$ $

$ $ $ $ $ $ $ $ $ $ $$$ $ $$$$ $ $ $ $ $ $ $$$$ $$$$

$ $ $ $$ $ $$$$$ $ $ $ $ $ $ $ $ $ $ $ $ $ $ $ $ $

$ $

$$$$ $$$$ $$$ $ $ $$$$$ $$$ $$ $$$ $ $ $ $$$ $$$ $ $$$ $ $ $$$$$

+7777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777+

Kansas Geological Survey AOS/VS

AOS/VS REV 07.57
AOS/VS XLPT REV 07.56

FORM 1411 % Ge

PRINTED IN U.S.A.

This is the documentation to describe the data structure of DGCALC.
(The last modificatrion is on Jan. 4, 1988.)

1. DATA STRUCTURE OF FORMULA ARRAY:
FormulaArray is an array of characters to store information about
every cell. The maximum number of characters can be stored
is 300000. It stores information about all cells in the spreadsheet,
including texts, formulas, or numbers.

2. DATA STURCTURE OF CELL POINTERS:
CellPointer is a two dimesional array of integer with 100 columns and
255 rows. The column and row indexes correspond to the location of a
cell in the spreadsheet. The value of the CellPointer is the location
of the cell in the FormulaArray.

3. CELL HEADER:
There are four characters in the CELL HEADER.

Index	Contents
1	Type ID, Text:char(241), Real:char(242), Integer:char(243), Wrong:char(240), Formula:char(244).
2	Significant Figure of a real number.
3	Column index in ASC II code.
4	Row index in ASC II code.

The contents are followed the HEADER, every cell is terminated with
a special character }, char (253).

4.DATA STRUCTURE OF STORED OR LOADED DATA FILE:
The following is the data structure of the file stored or read by the
DGCALC. The file is stored as an ACS II code string. The file has
four fields:
   1. HEADING BLOCK
   2. CONTENTS OF FORMULA ARRAY BLOCK
   3. CELL POINTER BLOCK
   4. ENDING BLOCK

The following is the description of above each block.
1. HEADING BLOCK has 6 characters. Each character has its special
   meaning.

INDEX	CH	FUNCTIONS
1	}	A flag to specify this is a DGCALC file.
2		CH 2 to 4 are codes to determine the size or length of
3		FormulaArray with data stored. They form a three digit
4		number with the base of 256.
5		Row number of the sheet in ACS II code
6		Column number of the sheet in ACS II code

2. CONTENTS OF FORMULA ARRAY BLOCK is the place to store the contents of
   the spreadsheet. Its size is equal to (NEXTDATA - 1).

3. CELL POINTER BLOCK is the place to save cell pointers of the spread-
   sheet. It is an integer array (rmax,cmax). The vulue of a pointer is
   converted into a 3 digit ASC II code, like it is done in the HEADING
   BLOCK, to determine the size of FormulaArray. The total bytes of this
   block is rmax * cmax * 3 = 76800 bytes. Cell pointers are saved from
   column 1, row 1 to row max, then column 2 form row 1 to row max,
   finally, column cmax, from row 1 to row mow max.

4. ENDING BLOCK stores information about the width of every column.
   The width is converted into an ASC II code. So the size of this block
   is 100 bytes since there are only 100 columns.

+5555555555

```
      Program DGCalc
c****** This module includes the main program and Move, Goto, Width,
c****** Save, Load, ChangeType commands.

      implicit none
c     ******* This is a file called DGCALC.PARAMS *******
c      This file only contains the definitions for commands.
      integer QuitSys
      parameter(QuitSys=1000)
      integer NoCommandSys
      parameter(NoCommandSys=0)
      integer MoveRightSys
      parameter(MoveRightSys=1)
      integer MoveLeftSys
      parameter(MoveLeftSys=2)
      integer MoveUpSys
      parameter(MoveUpSys=3)
      integer MoveDownSys
      parameter(MoveDownSys=4)
      integer MoveDownScrSys
      parameter(MoveDownScrSys=5)
      integer MoveUpScrSys
      parameter(MoveUpScrSys=6)
      integer MoveLeftScrSys
      parameter(MoveLeftScrSys=7)
      integer MoveRightScrSys
      parameter(MoveRightScrSys=8)
       integer MoveLeftEndSys
       parameter (MoveLeftEndSys=9)
      integer MoveRightEndSys
      parameter(MoveRightEndSys=10)
      integer MoveHomeSys
      parameter(MoveHomeSys=11)
      integer MoveDownEndSys
      parameter(MoveDownEndSys=12)
       integer MoveUpEndSys
       parameter (MoveUpEndSys=13)
c
c
      integer DoCliCommandSys
      parameter(DoCliCommandSys=20)
      integer GotoLocationSys
      parameter(GotoLocationSys=21)
      integer HelpSys
      parameter(HelpSys=22)
      integer ChangeFormatSys
      parameter(ChangeFormatSys=23)
      integer LoadSys
      parameter(LoadSys=24)
      integer SaveSys
      parameter(SaveSys=25)
      integer CopySys
      parameter(CopySys=26)
      integer TypeSys
      parameter(TypeSys=27)
      integer RecalculateSys
      parameter(RecalculateSys=28)
      integer InsertSys
      Parameter(InsertSys=29)
      integer DeleteSys
      Parameter(DeleteSys=30)
      integer PrintSys
      Parameter(PrintSys=31)
```

FORM 1411 ½ Gb
PRINTED IN U.S.A.

```
      integer MoveAreaSys
      parameter ( MoveAreaSys = 32 )
      integer EraseSys
      parameter ( EraseSys = 33 )
      integer JoinSys
      parameter ( JoinSys = 34 )
c     ********* End of DGCACL.PARAMS *********

c     ******** This is a file called DGCALC.GLOBAL *********
      integer Black, White
      parameter( Black=0, White=1 )
       character*1 defsig
      integer ArraySize*4
      parameter(ArraySize=300000)
      integer HSheetSize
      parameter(HSheetSize=100)
      integer VSheetSize
      parameter(VSheetSize=255)
       character*1 FormulaArray(ArraySize)
       character*(1920) ScreenBuffer
       character*(80) Spaces, Stars, Dashes, Wrong
       character*(20) WorkFileName
       logical Changed, Empty
       integer CellPointer(HSheetSize,VSheetSize)
       integer CurrX,CurrY,NextData
       integer CellWidth(HSheetSize)
       integer Top, Left, ScreenPtr
       integer OutFile, InFile, SaveFile, LoadFile
       common /FNames/ WorkFileName
       common /Chara/ defsig
       common /CBuffer/ ScreenBuffer
       common /IBuffer/ ScreenPtr
       common /Lines/ Spaces, Stars, Dashes, Wrong
       common /Files/ OutFile, InFile, SaveFile, LoadFile
       common /CSheet/ FormulaArray
       common /ISheet/ CellPointer,CurrX,CurrY,CellWidth,Top,Left
       common /IPointer/ NextData
c     ********* End of DGCALC.GLOBAL *********
c     ******** This is a file called DGCALC.GLOBAL1 *********
       integer Currentpos,Intvalue,Eol,StringLen
       integer IntstackPtr,RealStackPtr,BoolStackPtr,StringStackPtr
       character*(240) StringStack(10)
       integer IntStack(50)
       real RealStack(50)
       logical BoolStack(50)
       character*11 resword(23)
       real realvalue
       logical boolvalue
       integer identlen
       integer StrLenStack(10)
       character*240 stringvalue
       character*1 nextchar,currentchar
       character*11 token
       character*11 exprfollow,celltype
       character*11 restoken(23)
       character*11 ident
       common /INumber/Eol,identlen,currentpos
       common/IPtr/ IntStackPtr
       common/IStack/ IntStack
       common/Ivalue/ IntValue
       common/RPtr/ RealStackPtr
       common/RStack/RealStack,RealValue
       common/BPtr/ BoolStackPtr
```

```
      common/BStack/ BoolStack,BoolValue
      common/SPtr/ StringStackPtr,StrLenStack,StringLen
      common/SStack/ StringStack,StringValue
      common/Setofchar/ nextchar,currentchar
      common/Tokentype/ token,exprfollow,restoken,celltype
      common/Keytype/ resword
      common/CIdent/Ident
c     ********* This is the end of DGCALC.GLOBAL1 *********
      integer Command, CharsIn,i,GetWide, InfoLength
      parameter ( InfoLength = 80 )  ! Max width = 75
      character*(InfoLength) CellInfo
      logical NotDone /.True./
      character*1 Ch, GetChar, UpperCase
c     ********* This is a file called DGCALC.CELLTYPE *********
      character*1 Textcell,Intcell
      character*1 Realcell, Formulacell
      character*1 Plotcell
      character*1 Truecell, Falsecell,Wrongcell
      character*1 Eos, Unused
      integer Cellheader
      parameter (Cellheader=4)
      character*1 sp
      character*1 nul
      nul=char(0)
      sp=char(32)
      wrongcell=char(240)
      textcell=char(241)
      realcell=char(242)
      intcell=char(243)
      formulacell=char(244)
      plotcell=char(245)
      truecell=char(247)
      falsecell=char(248)
      eos=char(253)
      unused=char(254)
c     ********* End of DGCALC.CELLTYPE *********
      call InitialSheet
      call InitResWords
      call PrintMessArea ('   Edit')
      do while (NotDone)
         do i=1,68
            CellInfo(i:i)=' '
         end do
         call GetUserInput(Command,CellInfo,CharsIn)
         if((Command.ge.MoveRightSys).and.(Command.le.MoveUpEndSys))then
            if (Command.eq.MoveRightSys) then
               call Move(CurrX+1,CurrY)
            else if (Command.eq.MoveLeftSys) then
               call Move(CurrX-1,CurrY)
            else if (Command.eq.MoveUpSys) then
               call Move(CurrX,CurrY-1)
            else if (Command.eq.MoveDownSys) then
               call Move(CurrX,CurrY+1)
            else if (Command.eq.MoveUpScrSys) then
               call Move(CurrX,CurrY-19)
            else if (Command.eq.MoveDownScrSys) then
               call Move(CurrX,CurrY+19)
            else if (Command.eq.MoveRightScrSys) then
               i=CurrX+GetWide()
               call Move(i,CurrY)
            else if (Command.eq.MoveLeftScrSys) then
               i=CurrX-GetWide()
               call Move(i,CurrY)
```

```
            else if (Command.eq.MoveRightEndSys) then
               call MoveRightEnd
            else if (Command.eq.MoveLeftEndSys) then
               call Move ( 1, CurrY )
            else if (Command.eq.MoveUpEndSys) then
               call Move ( CurrX, 1 )
            else if (Command.eq.MoveDownEndSys) then
               call MoveDownEnd
            else if (Command.eq.MoveHomeSys) then
               call Move (1,1)
            end if
         else if (Command.eq.NoCommandSys) then
               call PrintMessArea('   Edit')
      else
         if (Command.eq.GotoLocationSys) then
               call GotoLoc
            else if (Command.eq.ChangeFormatSys) then
               call ChangeFormat
            else if (Command.eq.CopySys)  then
               call CopytoCells
            else if (Command.eq.LoadSys) then
               call LoadSheet
            else if (Command.eq.SaveSys) then
               call SaveCommands
            else if (Command.eq.TypeSys) then
               call ChangeCellType
            else if (Command.eq.QuitSys) then
               NotDone = .false.
               call Normal
            else if (Command.eq.HelpSys) then
               call Help
            else if (Command.eq.DoCliCommandSys) then
               call DoCliCommand
            else if  (command.eq.RecalculateSys) then
               call Recalculate
            else if (command.eq.InsertSys) then
               call Insert
            else if (command.eq.DeleteSys) then
               call Delete
            else if (command.eq.PrintSys) then
               call ErrorMess ( ' ' )
            else if ( Command .eq. MoveAreaSys ) then
               call MoveArea
            else if ( Command .eq. EraseSys ) then
               call Erase
            else if ( Command .eq. JoinSys  ) then
               call JoinFile
            end if
            if ( Command .ne. QuitSys ) then
               call PrintMessArea('   Edit')
            end if
         end if
      if (NotDone) then
         call PrintFormula ( CellPointer ( CurrX, CurrY ) )
      else
         if ((.not. Empty) .and. (Changed)) then
               call Pos (21,25,OutFile)
               write (OutFile) 'Working file is ', WorkFileName
               call PrintMess
     *   ('The file is not saved ! Do you still want to quit ?[Y/N]')
               Ch = ' '
               do while ((Ch.ne.'Y').and.(Ch.ne.'N'))
                 Ch = GetChar(InFile)
```

```
                  Ch = UpperCase(Ch)
                end do
                if (Ch .eq. 'N') NotDone = .true.
             end if
          end if
          end do
          call Pos ( 23,0, OutFile )
          call ClrEol
          call PrintMess ( 'Thank you for using DGCALC !' )
          call Pos ( 22, 0, OutFile )
          call CloseFiles
          end      ! End of Main program

          subroutine CloseFiles
c         Close all files and ready to leave DGCALC.
          implicit None
%include 'DGCalc.global'
          close ( OutFile )
          close ( InFile )
          close ( SaveFile )
          close ( LoadFile )
          return
          end    ! End of close files


          Subroutine Move(X,Y)
          implicit none
%include 'DGCalc.global'
          integer X,Y
          logical flag
          if ((X.ne.CurrX).or.(Y.ne.CurrY)) then
             flag=.false.
          else
             flag=.true.
          end if
          if (flag) then
              call Pos(22,0,OutFile)
              call ClrEOL
          end if
          if ((X.lt.1).or.(X.gt.HSheetSize).or.(Y.lt.1).or.
     &       (Y.gt.VSheetSize)) then
             call ErrorMess ( 'Wrong range !' )
          end if
          if (x.lt.1) x=1
          if (x.gt.Hsheetsize) x=Hsheetsize
          if (y.lt.1) y=1
          if (y.gt.Vsheetsize) y=Vsheetsize
          call RedrawCursor(X,Y)
          return
          end

          subroutine MoveRightEnd
c          Move the cursor to the right most position.
          implicit none
%include 'DGCalc.global'
          integer i,j,rmax
          rmax=1
          do i=1,Hsheetsize
             do j=1,Vsheetsize
                if (cellpointer(i,j).ne.0) then
                   if (i.gt.rmax) rmax=i
                end if
             end do
```

```
        end do
        call Move (rmax,Curry)
        return
        end

        subroutine MoveDownEnd
c        Move the cursor down to the lowest position.
        implicit none
%include 'DGCalc.global'
        integer i,j,cmax
        cmax=1
        do i=1,Hsheetsize
           do j=1,Vsheetsize
              if (cellpointer(i,j).ne.0) then
                 if (j.gt.cmax) cmax=j
              end if
           end do
        end do
        call Move(CurrX,cmax)
        return
        end

        Subroutine InitialSheet
        implicit none
%include 'DGCalc.global'
        integer  i, j
        ScreenPtr=1
        NextData=1
        SaveFile=20
        LoadFile=21
        defsig=char(3)
        close(SaveFile)
        close(LoadFile)
        InFile  = 8
        close(InFile)
        open(InFile,File='@input',form='unformatted',mode='binary',
     &           recfm='dynamic',screenedit='no',iostat=i)
        if (i.ne.0) then
          print *, 'Error on Input File open: ',i
          stop
        end if
        OutFile = 7
        close(OutFile)
        open(OutFile,File='@output',form='unformatted',mode='binary',
     &      carriagecontrol='none',recfm='dynamic')
         Changed = .false.
         Empty = .true.
        CurrX = 1
        CurrY = 1
        Top = 1
        Left = 1
         do I =1,20
           WorkFileName(I:I)=' '
         end do
         Empty = .true.
         Changed = .false.
        do i=1,HSheetSize
           CellWidth(i)=8
        end do
        do i=1,HSheetSize
           do j=1,VSheetSize
           CellPointer(i,j)=0
           end do
```

```
            end do
            Spaces = ' '
            do i=1,80
               Dashes(i:i)='-'
               Stars(i:i)='*'
               Wrong(i:i)='?'
            end do
            call DisplaySheet
            call Interpretcell(1,1,white,.false.)
            return
            end     ! End of InitialSheet

            Subroutine InterpretCell(X,Y,Background,buffered)
      c  This routine puts the cursor to the position X, Y and prints the
      c  contents ( Spaces if empty cell ) of cell (X, Y) on the monitor.
            implicit none
      %include 'DGCalc.global'
            integer X,Y,Background
            integer  startx, starty,i
            logical NotDone
            logical buffered
            startx = 5
            do i=left,X-1
               startx=startx+CellWidth(i)
            end do
            starty = 1+(Y-Top)
            if (buffered) then
               call PosBuf(starty,startx)
            else
               call Pos(starty,startx,OutFile)
            end if
            if (Background.eq.White) then
               if (buffered) then
                     call InverseBuf
               else
                     call Inverse
               end if
            else
               if (buffered) then
                     call NormalBuf
               else
                     call Normal
               end if
            end if
            if (CellPointer(X,Y).eq.0) then
               call writeit(Spaces,CellWidth(x),CellWidth(x),'Ljust',buffered)
            else
               call writeData (X,Y,buffered)
            end if
            if (buffered) then
               call NormalBuf
            else
               call Normal
            end if
            return
            end      ! End of InterpretCell

            Subroutine GetUserInput(Command,CellInfo,CharsIn)
            implicit none
      %include 'DGCalc.Params'
      %include 'DGCalc.global'
            character*(80) CellInfo
            logical esc
```

```
            integer Command, CommandType,CharsIn
            call Pos(21,10,OutFile)
            call ClrEOL
            esc = .false.
            CellInfo=' '
            CharsIn=0
            call Edit(OutFile,InFile,CellInfo,CharsIn,80,esc)
            if (esc) then
               Command = CommandType(CellInfo(1:1))
               if (.not.((Command.ge.MoveRightSys).and.(Command.le.MoveUpEndSys))) then
                  call GetCommand(Command)
               end if
            else
               if (CharsIn.ne.0) then
                  call UpdateCell(CellInfo,CharsIn)
                  call Move(currX,CurrY)
                  call Pos(21,10,OutFile)
                  call ClrEOL
               end if
               Command = NoCommandSys
            end if
            call Normal
            return
            end

             subroutine PrintFormula ( Index )
      c      Display the expression on the monitor if it is formula type.
      c      Display the current cell type otherwise.
      c      Display the label of current cell.
             implicit None
      %include 'DGcalc.global'
             integer Start, Last, Length, Index, I
             character*100 CellInfo
             character*2 Label, AConvert
      %include 'DGcalc.CellType'
             Start = CurrX
             Label = AConvert ( Start )
             call Pos ( 22, 0, OutFile )
             print *,       Label, CurrY,'  W=',CellWidth(CurrX),'  '
      c  call Pos ( 23, 0, OutFile )
               call ClrEol
             if ((FormulaArray(Index).eq.FormulaCell).or.
            &      (FormulaArray(Index).eq.WrongCell)) then
               Start = Index + CellHeader
               Last = Start
               do while((FormulaArray(Last).ne.';').and.(FormulaArray(Last)
            &            .ne.Eos) .and. (Last.lt.ArraySize))
                     Last = Last + 1
               end do
               Length = Last - Start
                     do I = 1, 100
                       CellInfo ( I:I ) = ' '
                     end do
                     do I = 0, Length - 1
                       CellInfo (I+1:I+1) = FormulaArray (Start + I)
                     end do
                     if (FormulaArray(Index).eq.FormulaCell) then
                        write(OutFile) 'FORMULA is : '
                     else
                        write(OutFile) 'BAD cell is : '
                     end if
                     write(OutFile) CellInfo (1:Length)
             else if (FormulaArray( Index ) .eq. TextCell ) then
```

```
                write(OutFile)'Text cell.'
            else if (FormulaArray(Index) .eq. RealCell ) then
                write(OutFile) 'Real number cell.'
            else if (FormulaArray(Index) .eq. IntCell ) then
                write(OutFile) 'Integer cell.'
            else if (FormulaArray(Index) .eq. TrueCell ) then
                write(OutFile)'Boolean cell, TRUE.'
            else if (FormulaArray(Index) .eq. FalseCell ) then
                write(OutFile) 'Boolean cell, FALSE.'
            end if
            return
            end   ! End of Print Formula

          integer function CommandType(ChComm)
          implicit none
    %include 'DGCalc.params'
          character*1 ChComm,uppercase
          integer ascii
          ChComm=uppercase(ChComm)
          ascii = ichar(ChComm)
          if (ascii.eq.72) then     ! H
             CommandType = HelpSys
          else if (ascii.eq.81) then ! Q
             CommandType= QuitSys
          else if (AscII.eq.23) then ! Up arrow
             CommandType = MoveUpSys
          else if (ascii.eq.24) then   ! Right arrow
             CommandType = MoveRightSys
          else if (ascii.eq.25) then   ! Left arrow
             CommandType = MoveLeftSys
          else if (AscII.eq.26) then   ! Down arrow
             CommandType = MoveDownSys
          else if (ascii.eq.49) then   ! Shift up arrow
             CommandType = MoveUpScrSys
          else if (ascii.eq.50) then   ! Shift right arrow
             CommandType = MoveRightScrSys
          else if (ascii.eq.51) then   ! Shift left arrow
             CommandType = MoveLeftScrSys
          else if (ascii.eq.52) then   ! Shift down arrow
             CommandType = MoveDownScrSys
           else if (ascii.eq.92) then   ! C1
             CommandType = MoveUpEndSys
          else if (ascii.eq.93) then    ! C2
             CommandType = MoveDownEndSys
           else if (ascii.eq.94) then   ! C3
             CommandType = MoveLeftEndSys
          else if (ascii.eq.95) then   !  C4
             CommandType = MoveRightEndSys
          else if (ascii.eq.8) then
             CommandType = MoveHomeSys
          else if (ascii.eq.70) then
             CommandType = ChangeFormatSys
          else if (ascii.eq.71) then
             CommandType = GotoLocationSys
          else if (ascii.eq.76) then
             CommandType = LoadSys
          else if (ascii.eq.83) then
             CommandType = SaveSys
          else if (ascii.eq.84) then
             CommandType = TypeSys
          else if (ascii.eq.67) then
             CommandType = CopySys
          else if (ascii.eq.82) then
```

FORM 1411 % 02
PRINTED IN U.S.A.

```
            CommandType = RecalculateSys
          else if (ascii.eq.80) then
            commandType = PrintSys  ! This command is no longer in use
          else if (ascii.eq.73) then
            CommandType = InsertSys
          else if (ascii.eq.68) then
            CommandType = DeleteSys
          else if (ascii.eq.36) then
            CommandType = DoCliCommandSys
           else if ( ascii .eq. 77 ) then   ! Ascii(77) = 'M'
            CommandType = MoveAreaSys
           else if ( AscII .eq. 69 ) then   ! AscII(69) = 'E'
            CommandType = EraseSys
           else if ( AscII .eq. 74 )then    ! AscII(74) = 'J'
            CommandType = JoinSys
          else
            CommandType = NoCommandSys
          end if
          return
          end   ! End of CommandType

          Subroutine GetCommand(Command)
    c      Print guide message for commands and return command.
          implicit none
    %include 'DGCALC.GLOBAL'
          integer Command, CommandType
          Character*1 Char, GetChar
          call PrintMessArea ('Command')
          call Pos(22,0,OutFile)
          write (OutFile) 'C)opy D)elete E)rase F)orm G)oto H)elp '
          write (OutFile) 'I)nsert J)oin L)oad M)ove Q)uit '
          call ClrEOL
          call Pos (23,0,outfile)
           call ClrEol
          write (outfile) ' R)ecalc S)ave T)ype $)CLI commands '
           write(outfile) ' Type (ESC) to resume the sheet.'
           call Pos(21,20,OutFile)
           call ClrEol
           if (WorkFileName(1:1).eq.' ') then
             write (outFile) 'No working file now      '
           else
             write (outFile) 'Working file is : ',WorkFileName
           end if
          Char=GetChar(InFile)
          command = CommandType(Char)
          call Pos(23, 0,outfile)
          call ClrEOL
          call Pos(22,0,Outfile)
          call ClrEOL
          return
          end    ! End of Get Command

          Subroutine GotoLoc
          implicit none
          Integer OutFile, InFile
          common /Files/ OutFile, InFile
          character*6 Name
          Logical Valid /.false./
          Integer X,Y
          Call Inputcellname (Name,' Cell name: ')
          Call Validcell(Name, X, Y ,Valid )
          If (Valid) then
              call Move (X, Y)
```

```
          else
              call ErrorMess ( 'Invalid cell name ! ' )
          end if
          return
          end   ! End of Goto Location

          subroutine Inputcellname(Name,message)
          implicit none
    c      %include 'DGCalc.params'
          character*6 Name
          character*(*) Message
          Logical esc
          Integer Outfile, Infile,Index
          Common /Files/Outfile,Infile
          esc=.false.
          Name = ' '
          Index=0
          call PrintMess ( Message )
          call Edit(Outfile,Infile,Name,Index,6,esc)
          call Pos(22,15,Outfile)
          call ClrEOL
          return
          end  ! End of Input Cell Name

           integer function GetWide()
           implicit none
    %include 'DGCalc.global'
          integer i,temp
          temp=CellWidth(Left)
          i=Left
          do while ((temp.le.75).and.(i.le.HSheetSize))
             i=i+1
             temp=temp+CellWidth(i)
           end do
           GetWide=i-Left
           return
           end

          Subroutine RedrawCursor(NewX,NewY)
          implicit none
    %include 'DGCalc.global'
          integer NoWide, NewX, NewY, DeltaX, DeltaY,GetWide
          logical NeedRedraw
          call InterpretCell (CurrX,CurrY,Black,.false.)
          CurrX = NewX
          CurrY = NewY
          Nowide = GetWide()
          DeltaX = NewX-Left
          DeltaY = NewY-Top
          NeedRedraw = .false.
          if ((DeltaX.lt.0).or.(DeltaX.ge.Nowide)) then
              if (DeltaX.lt.0) then
                 Left=NewX-Int(Nowide/2)
                 if (Left.lt.1) Left=1
              else
                 Left=NewX - Int(Nowide/2)
              end if
              NeedRedraw = .true.
          end if
          if ((DeltaY.lt.0).or.(DeltaY.ge.19)) then
              if (DeltaY.lt.0) then
                 Top = NewY- Int(19/2)
                 if (Top.lt.1) Top=1
```

```
            else
               Top = NewY - Int(19/2)
            end if
            NeedRedraw = .true.
         end if
         if (NeedRedraw)  call DisplaySheet
         call InterpretCell(NewX,NewY,White,.false.)
         return
         end      ! End of RedawCursor

         Subroutine DisplaySheet
c  This routine calls other subroutinines to display the sheet on the
c  monitor.
c  It calls : PrintLeft to print left index,
c             PrintTopScale to print top index,
c             InterpretCell to print the contents of each cell.
         implicit none
%include 'DGCalc.global'
         integer i, j, Wide,Index1,Index2,GetWide
         Wide =GetWide()
         call CLS(OutFile)
         do i=0,18
            if ((Top+i).le.VSheetSize) then
               call PrintLeft(Top+i)
            end if
         end do
         call PrintTopScale
c        do i=0,Wide-1
c           do j=0,18
          do J = 0, 18
            do I = 0, Wide - 1
               Index1=i+Left
               Index2=j+Top
               if ((index1.le.HSheetSize).and.(index2.le.VSheetSize)) then
                  if (CellPointer(index1,index2).ne.0) then
                  call InterpretCell(Index1,Index2,Black,.true.)
                  end if
               end if
            end do
         end do
         call FlushBuffer
         call Pos(20,0,OutFile)
         call Normal
         write (OutFile) Dashes(1:78)
         call Printmessarea('   Edit')
         return
         end  ! End of Display Sheet

          Subroutine ChangeFormat
          implicit none
%include 'DGCALC.global'
          character*1 ch,getchar,uppercase
          call PrintMess ( 'P)recision  W)idth' )
          ch=' '
          do while((ch.ne.'P').and.(ch.ne.'W').and.(ch.ne.char(27)))
             ch=getchar(infile)
             ch=uppercase(ch)
          end do
          if (ch.eq.'P') then
             call ChangePrecision
          else if (ch.eq.'W') then
             call SetNewWidth
          end if
```

```
            return
            end

            subroutine ChangePrecision
            implicit none
    %include 'DGCalc.global'
            character*1 ch,getchar,uppercase
            integer i,j,precision,index,wide,GetWide
    %include 'DGCalc.celltype'
            call PrintMess ( 'Precision (0..9): ' )
            ch=' '
            do while((ch.lt.'0').or.(ch.gt.'9'))
               ch=getchar(infile)
            end do
            precision=ichar(ch)-ichar('0')
            call PrintMess ( 'A)ll  R)ow  C)olumn  E)ntry' )
            ch=' '
            do while ((ch.ne.'A').and.(ch.ne.'R').and.(ch.ne.'C')
           &            .and.(ch.ne.'E').and.(ch.ne.char(27)))
               ch=getchar(infile)
               ch=uppercase(ch)
            end do
            call Pos (22,0,Outfile)
            call ClrEOL
            if (ch.eq.'A') then
               do i=1,HSheetSize
                  do j=1,VSheetSize
                     index=CellPointer(i,j)
                     if (index.ne.0) then
                        if (FormulaArray(index).eq.formulacell) then
                           FormulaArray(index+1)=char(precision)
                        end if
                     end if
                  end do
               end do
               defsig=char(precision)
               call DisplaySheet
            else if (ch.eq.'R') then
               do i=1,HSheetSize
                  index=CellPointer(i,CurrY)
                  if (index.ne.0) then
                     if (FormulaArray(index).eq.formulacell) then
                        FormulaArray(index+1)=char(precision)
                     end if
                  end if
               end do
               wide=GetWide()
               do i=0,Wide-1
                  index=i+Left
                  if (index.le.HSheetSize) then
                     call InterpretCell(index,CurrY,Black,.false.)
                  end if
               end do
            else if (ch.eq.'C') then
               do j=1,VSheetSize
                  index=CellPointer(CurrX,j)
                  if (index.ne.0) then
                     if (FormulaArray(index).eq.formulacell) then
                        FormulaArray(index+1)=char(precision)
                     end if
                  end if
               end do
               do j=0,18
```

```
                index=j+top
                if (index.le.VSheetSize) then
                  call InterpretCell(CurrX,Index,Black,.false.)
                end if
             end do
          else if (ch.eq.'E') then
             index=CellPointer(CurrX,CurrY)
             if (FormulaArray(index).eq.formulacell) then
                FormulaArray(index+1)=char(precision)
             end if
          end if
          call Move(CurrX,CurrY)
          return
          end   ! End of Change Precision

        Subroutine SetNewWidth
        implicit none
%include 'DGCalc.global'
        character*1 ch,getchar,uppercase
        integer  NewWidth,CurrWide,GetWide,i,temp
        call PrintMess ('Please enter new column width:(3-75) ')
        read (*,10) NewWidth
   10   format(i2)
        if ((NewWidth.ge.3).and.(NewWidth.le.75)) then
           call PrintMess ( 'A)ll  or  C)olumn : ' )
           ch=' '
           do while((ch.ne.'A').and.(ch.ne.'C').and.(ch.ne.char(27)))
              ch=getchar(infile)
              ch=uppercase(ch)
           end do
           if (ch.ne.char(27)) then
              if (ch.eq.'A') then
                 do i=1,HSheetSize
                    CellWidth(i)=NewWidth
                 end do
              else if (ch.eq.'C') then
                 CellWidth(CurrX)=NewWidth
              end if
              CurrWide=GetWide()
              temp=0
              do i=Left,CurrX
                 temp=temp+CellWidth(i)
              end do
              if (temp.gt.75) then
                 Left=CurrX-CurrWide/2
                 if (Left.lt.1) Left=1
              end if
              call DisplaySheet
              call InterpretCell(CurrX,CurrY,White,.false.)
           end if
        else
           call ErrorMess ( 'Out of the range 3 - 75 !' )
        end if
        return
        end  ! End of Set New Width

         subroutine UpdateCell(CellContents,Length)
c  Put new contents into current cell in the sheet.
c  It compresses formula array first, then puts new data into the
c  sheet if the avialable memory is big enough to store the data.
         implicit none
         character*(*) CellContents
         integer  Length
```

```
            character*1 ty,sig  !ty:data type;  sig:significant figure
            character*(50) Message
      %include 'DGCalc.global'
      %include 'DGCalc.celltype'
            if (CellPointer(CurrX,CurrY).ne.0) then
               call MarkFree(CurrX,CurrY)
            end if
            call CellType(CellContents,Length,ty,sig)
            if ((NextData+Length+cellheader+1).gt.ArraySize) then
               call CompressData
            end if
            if ((NextData+Length+cellheader+20).le.ArraySize) then
               CellPointer(CurrX,CurrY)=NextData
               call DataToArray(CellContents,Length+cellheader,ty,sig)
            else
               Message='Sheet filled to capacity with data -- No more Room!'
               call ErrorMess(Message)
            endif
            return
            end

            subroutine DataToArray(CellContents,Length,ty,sig)
      c  It stores data type(ty), significant figure after decimal(sig),
      c  column (CurrX) and row (CurrY) locations, input data, and end of
      c  data flag ( } ) into FormulaArray.
            implicit none
      %include 'DGCalc.global'
      %include 'DGCalc.global1'
            integer Length,i,j
            logical errorflag
            character*1 ty,sig
            character*11 ascii
            character*(*) CellContents
      %include 'DGCalc.celltype'
            errorflag=.false.
            FormulaArray(NextData)=ty
            FormulaArray(NextData+1)=sig
            FormulaArray(NextData+2)=char(CurrX)
            FormulaArray(NextData+3)=char(CurrY)
            if ((ty.eq.intcell).or.(ty.eq.realcell)) then
              i=NextData
              do while ((i.lt.(NextData+Length-CellHeader)).and.(CellContents
           &     (i-NextData+1:i-NextData+1).ne.sp))
                 FormulaArray(i+CellHeader)=CellContents(i-
           &     NextData+1:i-NextData+1)
                 i=i+1
              end do
              FormulaArray(i+CellHeader)=eos
              NextData=i+CellHeader+1
            else if ((ty.eq.truecell).or.(ty.eq.falsecell)) then
                 FormulaArray(NextData+CellHeader)=eos
                 NextData=NextData+CellHeader+1
            else if ((ty.eq.textcell).or.(ty.eq.formulacell)) then
                 do i=NextData, NextData+Length-CellHeader-1
                    FormulaArray(i+CellHeader)=CellContents(i-
           &        NextData+1:i-NextData+1)
                 end do
                 FormulaArray(NextData+Length)=eos
                 NextData=NextData+Length+1
                 if (ty.eq.formulacell) then
                    call GetCellValue (CurrX,CurrY,celltype)
                    if (celltype.eq.'Formula    ') then
                       call realtoascii (ascii,realvalue,errorflag)
```

```
                        if (.not.errorflag) then
                           i=NextData-1
                           FormulaArray(i)="="
                           do j=1, 11
                                FormulaArray(i+j)=ascii(j:j)
                           end do
                           FormulaArray(i+j)=eos
                           NextData=i+j+1
                        end if
                     end if
                  end if
               end if
               return
               end      ! End of Data To Array

               subroutine CopyCell(marker,index)
               implicit none
      %include 'DGCalc.global'
               integer index,marker
      %include 'DGCalc.celltype'
               FormulaArray(marker)=FormulaArray(index)
               FormulaArray(marker+1)=FormulaArray(index+1)
               CellPointer(ichar(FormulaArray(index+2)),ichar
           &              (FormulaArray(index+3)))=marker
               marker=marker+2
               index=index+2
               do while ((index.le.NextData).and.(FormulaArray(marker-1)
           &           .ne.eos))
                  FormulaArray(marker)=FormulaArray(index)
                  marker=marker+1
                  index=index+1
               end do
               return
               end

               subroutine CompressData
      c        Return the unused memory to FormulaArray
               implicit none
      %include 'DGCalc.global'
               integer marker,index
      % include 'DGCalc.celltype'
               marker=1
               index=1
               do while (index.le.ArraySize)
                  do while (FormulaArray(index).eq.unused)
                        index=index+1
                  end do
                  if (index.lt.NextData) then
                     call CopyCell(marker,index)
                  else
                     index=ArraySize+1
                  end if
               end do
               NextData=marker
               return
               end   ! End of Compress Data

               subroutine CellType (Cell,Length,ty,sig)
               implicit none
               character*(*) Cell
               character*1 ty,sig
               character*7 Chset
               integer Length,index,no,noskip
```

```
%include 'DGCalc.global'
%include 'DGCalc.celltype'
        ty=textcell
        sig=defsig
        index=2
        if ((Cell(1:1).eq.'+').or.(Cell(1:1).eq.'-').or.((Cell(1:1)
     &     .ge.'0').and.(Cell(1:1).le.'9'))) then
           Chset='Numaric'
           call Skipset (Cell,Length,noskip,Chset,index)
           Chset='Space  '
           call Skipset (cell,length,noskip,Chset,index)
           if (index.gt.Length) then
              ty=intcell
              sig=nul
           else if (Cell(index:index).eq.'.') then
                index=index+1
                Chset='Numaric'
                call Skipset(Cell,Length,noskip,Chset,index)
                no=noskip
                Chset='Space  '
                call Skipset(Cell,Length,noskip,Chset,index)
                if (index.gt.Length) then
                   ty=realcell
                   sig=char(no)
                end if
           end if
        else if ((Length.ge.2).and.(Cell(1:1).eq.'.')) then
                if (Cell(2:2).eq.'T') ty=truecell
                if (Cell(2:2).eq.'F') ty=falsecell
        else if (Cell(1:1).eq.'#') then
                ty=formulacell
                Length = Length + 1
                Cell(Length:Length) = ';'
        end if
        return
        end   ! End of CellType

        subroutine ChangeCellType
c       Change the cell type from numeric to text, or vise versa.
c       It could change formula cell to text cell to see the formula
c       in the current cell before Sept. 27, 1986. After that day, since
c       formula can be seen on the monitor, it does not need to keep
c       this function any more. It is saved only for reference.
        implicit none
%include 'DGCalc.global'
%include 'DGCalc.global1'
        character*1 ty,ch,GetChar,Uppercase,type
        character*11 celloftype
        Logical Modified
        integer index,start,last,length,I,J,X1,X2, Y1,Y2,X,Y
        integer XT,YT
%include 'DGCalc.celltype'
           call Pos(22,15,OutFile)
           write(OutFile) 'Cell Type: N)umeric     T)ext'
           call ClrEOL
        Ch = ' '
        do while((ch.ne.'N').and.(ch.ne.'T').and.(Ch.ne.char(27)))
           ch=GetChar(InFile)
           ch=Uppercase(ch)
        end do
        if ( Ch .ne. char (27)) then
          if (ch.eq.'N') then
             ty=realcell
```

```
            else
               ty=textcell
            end if
            call PrintMess ('C)olumn     R)ow    or    E)ntry')
            Ch = ' '
            do while ((Ch.ne.'C').and.(Ch.ne.'R').and.(Ch.ne.'E'))
              Ch = GetChar(InFile)
              Ch = UpperCase(Ch)
            end do
            Modified = .false.
            XT = CurrX
            YT = CurrY
            if (Ch.eq.'C') then
              Y1 = 1
              Y2 = VSheetSize
              X1 = CurrX
              X2 = CurrX
            else if (Ch.eq.'R') then
              Y1 = CurrY
              Y2 = CurrY
              X1 = 1
              X2 = HSheetSize
            else
              Y1 = CurrY
              Y2 = CurrY
              X1 = CurrX
              X2 = CurrX
            end if
            do CurrY = Y1, Y2
            do CurrX = X1, X2
            X = CurrX
            Y = CurrY
            index=CellPointer(X,Y)
            type=FormulaArray(index)
            if (index.ne.0) then
    c     Modified = .true.
              Ch = FormulaArray(Index)
              if ((Ch.eq.TextCell).or.(Ch.eq.IntCell).or.(Ch.eq.RealCell)) then
                FormulaArray(index)=ty
                if (ty.eq.realcell) then
                  call GetCellValue (X,Y,celloftype)
                  if (celloftype.ne.'Realconst ') then
                     FormulaArray(index)=type
                     call errormess('Cell can not be numeric type')
                  else
                     if (FormulaArray(currentpos).ne.eos) then
                          FormulaArray(index)=type
                          call errormess('Cell can not be numeric type')
                     end if
                  end if
                end if
              end if
            end do  ! I loop
            end do  ! J loop
            CurrX = XT
            CurrY = YT
            call DisplaySheet
            call InterpretCell (CurrX,CurrY,White,.false.)
            end if
            return
            end  ! End of Change Cell Type
```

PRINTED IN U.S.A.

+9999999999

```
c******* This module includes Insert, Delete, Recalculate, Printsheet,
c******* Copycell, Save, Load ,DoCLI, and MoveArea commands.

      subroutine Insert
      implicit none
%include 'DGCalc.global'
      logical valid
      character*1 ch, getchar,uppercase
      call PrintMess ( ' R)ow  or  C)olumn' )
      ch=' '
      do while ((ch.ne.'R').and.(ch.ne.'C').and.(ichar(ch).ne.27))
         ch=Getchar(InFile)
         ch=uppercase(ch)
      end do
      if (ch.eq.'R') then
         call InputRow
      else if (ch.eq.'C') then
         call InputColumn
      end if
      call Pos (22,15,OutFile)
      call ClrEOL
      return
      end ! Insert

      subroutine InputColumn
      implicit none
%include 'DGCalc.global'
      integer i
      character*1 ch,getchar
      ch='a'
      call PrintMess ( 'How many columns (0-9) ?' )
      do while((ch.lt.'0').or.(ch.gt.'9'))
         ch=Getchar(InFile)
      end do
      i=ichar(ch)-ichar('0')
      call InsertColumn (currX,i)
      return
      end  ! Input column

      subroutine InputRow
      implicit none
%include 'DGCalc.global'
      integer j
      character*1 ch, getchar
      ch=' '
      call PrintMess ( 'How many rows (0-9) ?' )
      do while ((ch.lt.'0').or.(ch.gt.'9'))
         ch=getchar(InFile)
      end do
      j=ichar(ch)-ichar('0')
      call InsertRow(currY,j)
      return
      end

      subroutine InsertRow (index,number)
      implicit none
      integer index,number,i,j,temp
%include 'DGCalc.global'
      if (number.ne.0) then
         Changed = .true.
         do i=1,Hsheetsize
            do j=Vsheetsize,index+number,-1
              cellpointer(i,j)=cellpointer(i,j-number)
```

```
                  if (cellpointer(i,j).ne.0) then
                     temp=cellpointer(i,j)
                     FormulaArray(temp+2)=char(i)
                     FormulaArray(temp+3)=char(j)
                  end if
               end do
            end do
            do i=1,Hsheetsize
               do j=index,index+number-1
                  cellpointer(i,j)=0
               end do
            end do
            call Displaysheet
            call move(currx,curry)
         end if
         return
         end ! Insert Row

         subroutine InsertColumn(index,number)
         implicit none
%include 'DGCalc.global'
         integer index, number,i,j,temp
         if (number.ne.0) then
            Changed = .true.
            do i=Hsheetsize,index+number,-1
               CellWidth(i)=CellWidth(i-number)
               do j=1,Vsheetsize
                  cellpointer(i,j)=cellpointer(i-number,j)
                  if (cellpointer(i,j).ne.0) then
                     temp=cellpointer(i,j)
                     FormulaArray(temp+2)=char(i)
                     FormulaArray(temp+3)=char(j)
                  end if
               end do
            end do
            do i=index,index+number-1
               CellWidth(i)=9
               do j=1,Vsheetsize
                  cellpointer(i,j)=0
               end do
            end do
            call Displaysheet
            call move (currx,curry)
         end if
         return
         end ! Insert Column

         subroutine Delete
c        Delete number of rows or columns counted from current
c        row or column. Number is counted from current X or Y.
         implicit none
%include 'DGCalc.global'
         integer TempX, TempY, I, J
         character*1 ch,getchar,uppercase
         call PrintMess ( '  R)ow  or  C)olumn' )
         ch=' '
         do while ((ch.ne.'R').and.(ch.ne.'C').and.(ch.ne.char(27)) )
            ch=getchar(InFile)
            ch=uppercase(ch)
         end do
         call Pos (22,15,OutFile)
         call ClrEOL
         if (ch .ne. char(27)) then
```

```
            if (ch.eq.'R') then
               call DeleteRow
            else if (ch.eq.'C') then
               call DeleteColumn
            end if
            call CheckEmpty
          end if
          return
          end    ! End of Delete

          subroutine CheckEmpty
%include 'DGCALC.global'
          integer I,J
            Empty = .true.
            do I = 1,HSheetSize
              do J = 1, VSheetSize
                   if (CellPointer(I,J).ne.0) then
                      Empty = .false.
                      go to 999
                   end if
              end do
            end do
999       continue
          return
          end ! Check Empty

          subroutine Erase
c         Erase the sheet,a window, current row or column, or entry.
c         Current row or column is where the cursor is.
          implicit none
%include 'DGCalc.global'
          logical valid
          integer LeftX, RightX, TopY, BottomY
          integer TempX, TempY, I, J
          character*6 Name
          character*1 ch,getchar,uppercase
          call PrintMess ( 'A)ll  W)indow  R)ow  C)olumn  E)ntry ')
          ch=' '
          Valid = .true.
          do while ((ch.ne.'R').and.(ch.ne.'C').and.(ch.ne.char(27)).and.
     &              (ch.ne.'A').and.(Ch.ne.'W').and.(ch.ne.'E'))
             ch=getchar(InFile)
             ch=uppercase(ch)
          end do
          if (Ch .eq. Char(27)) Return
          if ( Ch .eq. 'A' ) then
                   call ClearSheet
                   Changed = .false.
                   Empty = .true.
                   return
          end if
          if (ch.eq.'E') then
             call BlankCell (CurrX,CurrY)
             call PrintMess ( ' Current cell is erased. ' )
             Changed = .true.
             call CheckEmpty
             return
          end if
          if (ch.eq.'R') then
                   LeftX = 1
                   RightX = HSheetSize
                   TopY = CurrY
                   BottomY = Curry
```

```
                do j=currY,currY+k-1
                   if (cellpointer(i,j).ne.0) then
                      x=i
                      y=j
                      call Markfree(x,y)
                   end if
                end do
             end do
             do i=1,Hsheetsize
              do j=curry,Vsheetsize-k
                 cellpointer(i,j)=cellpointer(i,j+k)
                 if (cellpointer(i,j).ne.0) then
                    temp=cellpointer(i,j)
                    FormulaArray(temp+2)=char(i)
                    FormulaArray(temp+3)=char(j)
                 end if
              end do
             end do
             do i=1,Hsheetsize
              do j=Vsheetsize,Vsheetsize-k+1
                 cellpointer(i,j)=0
              end do
             end do
             call Displaysheet
             call move (currx,curry)
          else
             call PrintMess ( 'No rows are deleted.' )
          end if
          return
          end    ! End of Delete Rows

          subroutine DeleteColumn
c         Delete number of columns counted from current column.
          implicit none
%include 'DGCalc.global'
          character*1 ch,getchar
          integer i,j,temp,k,x,y
          call PrintMess ( 'How many columns (0-9) ? ' )
          ch=' '
          do while ((ch.lt.'0').or.(ch.gt.'9'))
             ch=getchar(InFile)
          end do
          k=ichar(ch)-ichar('0')
          if ( K .ne. 0 ) then
            do i=currX,currX+k-1
             do j=1,Vsheetsize
                if (cellpointer(i,j).ne.0) then
                   x=i
                   y=j
                   call Markfree(x,y)
                end if
             end do
            end do
            do i=currx,Hsheetsize-k
             CellWidth(i)=CellWidth(i+k)
             do j=1,Vsheetsize
                cellpointer(i,j)=cellpointer(i+k,j)
                     if (cellpointer(i,j).ne.0) then
                        temp=cellpointer(i,j)
                        FormulaArray(temp+2)=char(i)
                        FormulaArray(temp+3)=char(j)
                     end if
             end do
```

FORM L411 ½ GB
PRINTED IN U.S.A.

```
              end do
              do i=Hsheetsize-k+1,Hsheetsize
               CellWidth(i)=9
               do j=1,Vsheetsize
                    cellpointer(i,j)=0
               end do
              end do
              call Displaysheet
              call move (currx,curry)
            else
              call PrintMess ( 'No columns are deleted.' )
            end if
            return
            end  ! End of Delete Columns


            subroutine ReCalculate
    c       Recalculate formulas and update the sheet with new values.
            implicit none
    %include 'DGCalc.global'
    %include 'DGCalc.global1'
            logical Error,Valid
            character*11 characters,celloftype
            integer i,j,index,l, X,Y, XL,XR, YU, YD ! Left Right Up Down
            character*1 Ch, GetChar, UpperCase
    %include 'DGCalc.celltype'
            call PrintMess ('E)ntry      W)indow      S)heet')
            Ch = ' '
            do while ((Ch.ne.'E').and.(Ch.ne.'W').and.(Ch.ne.'S').and.(Ch.ne.Char(27)))
              Ch = GetChar (InFile)
              Ch = UpperCase (Ch)
            end do
            if (Ch .eq. Char(27)) return
            X = CurrX
            Y = CurrY
            if (Ch .eq. 'E') then
               XL = CurrX
               XR = CurrX
               YD = CurrY
               YU = CurrY
            else if (Ch .eq. 'S') then
               XL = 1
               XR = HSheetSize
               YU = 1
               YD = VSheetSize
            else if (Ch.eq.'W') then
               call WndwLocation (XL,YU,XR,YD,Valid)
               if (.not.Valid) return
            end if
            do i=XL,XR      !1,Hsheetsize
               do j=YU,YD   !1,Vsheetsize
                    index=cellpointer(i,j)
                    if (index.ne.0) then
                       if (FormulaArray(index).eq.formulacell) then
                            currentpos=index+cellheader
                            currX=i
                            currY=j
                            call getcellvalue(currX,currY,celloftype)
                            if (celloftype.eq.'Formula    ') then
                              call realtoascii(characters,realvalue,error)
                              if (.not.error) then
                                do l=1,11
                                  FormulaArray(currentpos+l)=characters(l:l)
```

```
                                end do
                              end if
                            end if
                          end if
                        end if
                      end do
                    end do
                    CurrX = X
                    CurrY = Y
                    call DisplaySheet
                    call Move (CurrX,CurrY)
                    return
                    end  ! End of Re Calculate

                    subroutine CopytoCells
                    implicit none
%include 'DGCalc.global'
                    logical valid
                    character*6 name
                    character*1 ch,getchar,uppercase
                    integer X,Y,X1, Y1, index
%include 'DGCalc.celltype'
                    valid=.true.
                    call PrintMess ( 'C)ell    or    W)indow' )
                    Ch = ' '
                    do while((Ch.ne.'C').and.(Ch.ne.'W').and.(Ch.ne.Char(27)))
                           Ch = GetCHar ( InFile )
                           Ch = UpperCase ( Ch )
                    end do
                    if ( Ch .eq. 'C' ) then
                      call InputcellName(name,'Copy from: ')
                      call Validcell (name,x,y,valid)
                      if (valid.and.( cellpointer(x,y).ne.0)) then
                         index=cellpointer(x,y)
                         if (FormulaArray(index).ne.formulacell) then
                            call CopySame(index)
                         else
                            call PrintMess ( 'R)elative     N)o adjustment ' )
                            ch=' '
                          do while((ch.ne.'R').and.(ch.ne.'N').and.(ch.ne.char(27)))
                            ch=getchar(InFile)
                            ch=uppercase(ch)
                          end do
                            if (ch.eq.'N') then
                               call CopySame(index)
                            else if (ch.eq.'R') then
                               call CopyRelative(index)
                            end if
                         end if
                      end if   ! Valid
                    else if ( Ch .eq. 'W' ) then
                      call WndwLocation ( X, Y, X1, Y1, Valid )
                      if ( Valid ) then
                           call CopyTheArea(X,Y, X1,Y1, Valid )
                      end if
                    end if
                    return
                    end       ! Copy to Cell

                    subroutine CopyTheArea(LeftX,TopY,RightX,BottomY,Valid)
c                   This routine copies the source area to the target area.
c                   It receives the location of source and copies it to the
c                   target if both source and target locations are valid,
```

```
c       otherwise it return Valid = .false.
        implicit none
%include 'DGcalc.global'
%include 'DGCalc.global1'
        integer LeftX, TopY, RightX, BottomY
        logical Valid
        integer TargetX, TargetY, TargetX1, TargetY1, SourceX, SourceY
        integer GapX, GapY, WidthX, WidthY
        integer TempX, TempY, I, J, K, Start, Last, Length
        character*6 CellName
        character*100 TempChar
%include 'DGCalc.CellType'
        TempX = CurrX
        TempY = CurrY
        WidthX = RightX - LeftX
        WidthY = BottomY - TopY
        call InputCellName ( CellName, 'Copy to left top corner: ' )
        call ValidCell ( CellName, TargetX, TargetY, Valid )
        if ( Valid ) then
          GapX = TargetX - LeftX
          GapY = TargetY - TopY
          TargetX1 = TargetX + WidthX
          TargetY1 = TargetY + WidthY
          if ((TargetX .ge. LeftX) .and. (TargetX .le. RightX) .and.
     &        (TargetY .ge. TopY ) .and. (TargetY .le.BottomY) .or.
     &        (TargetX1.ge. LeftX) .and. (TargetX1.le. RightX) .and.
     &        (TargetY1.ge. TopY ) .and. (TargetY1.le.BottomY) .or.
     &        (TargetX1.gt.HSheetSize).or.(TargetY1.gt.VSheetSize)) then
            call ErrorMess ( 'Invalid location !' )
            Valid = .false.
          end if
        end if
        if ( Valid ) then
          do I = TargetX, TargetX1
            do J = TargetY, TargetY1
              do K = 1, 100
                TempChar ( K:K ) = ' '
              end do
              SourceX = I - GapX
              SourceY = J - GapY
              K = CellPointer ( SourceX, SourceY )
              if ( K .gt. 0 ) then ! Not empty cell
                Start = K + CellHeader
                Last = Start
                do while((FormulaArray(Last).ne.eos).and.
     &                   (Last .lt.ArraySize))
                  Last = Last + 1
                end do
                Length = Last - Start
                do K = 1, Length
                  TempChar( K:K ) = FormulaArray ( Start + K - 1 )
                end do
                CurrX = I
                CurrY = J
                call UpdateCell ( TempChar, Length )
              end if   ! Not Empty cell
            end do  ! End of J loop
            if (CellWidth(I) .lt. CellWidth(SourceX) ) then
              CellWidth(I) = CellWidth(SourceX)
            end if
          end do  ! End of I loop
          call DisplaySheet
          CurrX = TempX
```

```
               CurrY = TempY
               call Move ( CurrX, CurrY )
            end if
            return
            end   ! End of Copy the Area

            subroutine CopySame(index)
            implicit none
    %include 'DGCalc.global'
            integer index,Leftx,Rightx,TopY,BottomY,i,j,tempX,tempY
            integer row,column,start,last,length
            character*6 name
            character*100 tempchar
            logical valid
    %include 'DGCalc.celltype'
            do i=1,100
               tempchar(i:i)=' '
            end do
            call WndwLocation ( LeftX,TopY,RightX,BottomY,Valid )
            if (valid) then
               tempX=currX
               tempY=currY
               start=index+cellheader
               last=start
               do while((FormulaArray(last).ne.eos).and.(last.lt.ArraySize))
                  last=last+1
               end do
               length=last-start
               do i=1,length
                  tempchar(i:i)=FormulaArray(start+i-1)
               end do
               do i=leftX,rightX
                  do j=TopY,BottomY
                     currX=i
                     currY=j
                     call UpdateCell(tempchar,length)
                  end do
               end do
               call DisplaySheet
               currX=tempX
               currY=tempY
               call Move(currX,currY)
            end if
            return
            end

            subroutine CopyRelative (index)
            implicit none
    %include 'DGCalc.global'
    %include 'DGCalc.global1'
            integer index,LeftX,RightX,TopY,BottomY,i,j,l,start,length
            integer tempX,tempY,row,column,k,temp
            character*6 name,position,temppposition
            character*100 tempchar
            logical valid,ok
    %include 'DGCalc.celltype'
            ok=.true.
            do i=1,100
               tempchar(i:i)=' '
            end do
            call WndwLocation (LeftX,TopY,RightX,BottomY,Valid)
            if (valid) then
               tempX=currX
```

PRINTED IN U.S.A.

```
                        l=l+1
                     end do
                  end if
                  start=currentpos
               end do
               k=k-1
               currX=i
               currY=j
               call UpdateCell(tempchar,k-1)
            end do
         else
            call errormess ('Illegal copy range')
            ok=.false.
         end if
      else
         call errormess ('Illegal copy range or cell name')
         ok=.false.
      end if
      if (ok) then
         currX=tempX
         currY=tempY
         call DisplaySheet
         call Move(currX,currY)
      end if
      return
      end

      Subroutine MarkFree (X, Y)
      Implicit none
%include 'DGCalc.global'
      Integer X, Y, Index
%include 'DGCalc.celltype'
      index = Cellpointer(X,Y)
      Do while ((FormulaArray(index).ne.eos).and.(index.lt.ArraySize))
         FormulaArray(index) = unused
         index = index +1
      end do
      FormulaArray (index) = unused
      return
      end

      Subroutine BlankCell (Row, Column)
      implicit none
%include 'DGCalc.global'
      integer Row, Column
      if (Cellpointer(Row,Column).ne.0) then
         call MarkFree (Row, Column)
         Cellpointer(Row, Column) = 0
         call InterpretCell (Row, Column, White,.false.)
      end if
      return
      end

      Subroutine ClearSheet
      implicit none
%include 'DGCalc.global'
      integer i,j
      character*1 ch, GetChar, UpperCase
      call PrintMess ( 'Are you sure to clear the sheet ? (Y/N) ' )
      Ch = ' '
      do while((Ch.ne.'Y').and.(Ch.ne.'N').and.
     &      (Ch.ne.Char(13)).and. (Ch .ne. Char(27) ) )
            Ch = GetChar(InFile)
```

```
                Ch = UpperCase ( Ch )
            end do
            if ( Ch.eq.'Y' ) then
                do i=1, HSheetsize
                    do j=1,VSheetsize
                        Cellpointer(i,j) = 0
                    end do
                    CellWidth (I) = 8
                end do
                call InterpretCell(CurrX,CurrY,Black,.false.)
                do I=1,20
                    WorkFileName(I:I)=' '
                end do
                Empty = .true.
                Changed = .false.
                Top = 1
                Left = 1
                CurrX = 1
                CurrY = 1
                NextData = 1
                call DisplaySheet
                call InterpretCell (CurrX,CurrY, white,.false.)
                call ErrorMess ( 'Whole sheet is erased.' )
            end if
            return
            end    ! End of Clear Sheet

            subroutine WndwLocation(LeftX,TopY,RightX,BottomY,Valid)
      c     Return the window location to the calling program if the
      c     valid locations are type, otherwise Valid = .false.
            implicit none
      %include 'DGCalc.global'
            logical valid
            integer LeftX, RightX, TopY, BottomY
            character*6 Name
            call InputCellName ( Name, 'Top left corner: ' )
            call ValidCell ( Name, LeftX, TopY, Valid )
            if ( Valid ) then
                 call InputCellName ( Name, 'Bottom right corner: ')
                 call ValidCell ( Name, RightX, BottomY, Valid )
                 if ( Valid ) then
                   if ((LeftX.gt.RightX) .or. (TopY.gt.BottomY) .or.
           &          (LeftX.lt.1) .or. (RightX.gt.HSheetSize) .or.
           &          (TopY .lt.1) .or. (BottomY.gt.VSheetSize)) then
                      Valid = .false.
                    end if
                 end if
            end if
            if ( .not. Valid ) then
                    call ErrorMess ( 'Invalid location ! ' )
            end if
            return
            end   ! End of Window Location

            subroutine SaveCommands
      c     Save the data to a file. The data can be the whole sheet or
      c     a window for DGCALC file, or can be the whole sheet or window
      c     for printable fiule, or a TERNARY file.
            implicit None
      %include 'DGCALC.global'
            character*1 GetChar, UpperCase, Ch1, Ch2
            character*(20) FileName
            logical Valid, Escape
```

PRINTED IN U.S.A.

```
      integer Index, IErr, I
      call PrintMess('D)GCALC   P)rintable (data),  or  T)ernary file')
      Ch1 = ' '
      Ch2 = ' '
      do while ((Ch1.ne.'D').and.(Ch1.ne.'P').and.(Ch1.ne.'T')
     &          .and.(Ch1.ne.char(27)))
          Ch1 = GetChar ( InFile )
          Ch1 = UpperCase ( Ch1 )
      end do
      if ( Ch1 .ne. char(27) ) then
        if ( Ch1 .eq. 'D') then
          call PrintMess ( 'S)heet     or    W)indow' )
          do while((Ch2.ne.'S').and.(Ch2.ne.'W').and.(Ch2.ne.char(27)))
              Ch2 = GetChar(Infile)
              Ch2 = UpperCase ( Ch2 )
          end do
        else if ( Ch1 .eq. 'P' ) then
          call PrintMess ( 'S)heet     W)indow    or     D)ump' )
          do while((Ch2.ne.'S').and.(Ch2.ne.'W').and.(Ch2.ne.'D')
     &          .and.(Ch2.ne.char(27)))
              Ch2 = GetChar(Infile)
              Ch2 = UpperCase ( Ch2 )
          end do
         end if
          if ( Ch2 .eq. char(27) ) return
          call PrintMess('Save to file : ')
          call Edit(OutFile,InFile,FileName,Index,20,Escape)
              if (Escape) return
          if ( Ch1 .eq. 'D' ) then
              I = 1
              do while ((FileName(I:I).ne.' ').and.(I.le.20))
                  I = I + 1
              end do
              FileName(I:I+2) = '.DG'
          end if
          call Openf (SaveFile, FileName, 'W', IErr )
          if ( IErr .ne. 0 ) then
              Valid = .false.
              call ErrorMess ( 'Something is wrong ! ')
              write(outFile) 'Error type ', IErr, '.'
              return
          end if
          if ((Ch1.eq.'D').and.(Ch2.eq.'S')) then
                  call DGSaveSheet (FileName,Valid) !Sheet for DGCALC
              if ( Valid ) then
              call PrintMess ( 'The sheet is saved to: ' )
              end if
          else if ((Ch1.eq.'D').and.(Ch2.eq.'W')) then
                  call DGSaveWindow (FileName,Valid)!Window for DGCALC
              if ( Valid ) then
              call PrintMess ( 'The window is saved to: ')
              end if
          else if ((Ch1.eq.'P').and.(Ch2.eq.'S')) then
                  call PSaveSheet(FileName,Valid)! Sheet for Printing
              if ( Valid ) then
              call PrintMess ('Printable sheet is saved to: ' )
              end if
          else if ((Ch1.eq.'P').and.(Ch2.eq.'W')) then
                  call PSaveWindow(FileName,Valid)!Window for Printing
              if ( Valid ) then
              call PrintMess('Printable window is saved to: ' )
              end if
          else if ((Ch1.eq.'P').and.(Ch2.eq.'D')) then
```

```
                    call DMPFileSave(FileName,Valid)!Dump for Printing
                  if ( Valid ) then
                  call PrintMess('The dump file is saved to: ' )
                  end if
              else
                  call TernarySave(FileName,Valid)
              end if
                  if ( Valid ) then
                          write (OutFile) FileName
                  end if
          end if
          return
          end       ! End of Save Command


          subroutine PSaveSheet ( FileName, Valid )
c         Save the whole sheet as a printable file.
          implicit None
%include 'DGCALC.global'
          character*6 Corner
          logical Valid,esc
          character*100 Title(3)
          character*(*)  FileName
          character*17 OutFileName
          character*1 Command, GetChar, UpperCase, Ch
          integer i,j,indexx,LeftX,RightX,TopY,BottomY,IErr,x,y, TempFile
          integer TX, TY
                  X = 0
                  Y = 0
                  do I = 1, HSheetSize
                     do J = 1, VSheetSize
                        TX = I
                        TY = J
                        if (CellPointer(TX,TY) .ne. 0 ) then
                          if ( TX .gt. X ) X = TX
                          if ( TY .gt. Y ) Y = TY
                        end if
                     end do
                  end do
          call PrintMess ('R)egular    or    M)athPlan')
          Ch = ' '
          do while ((Ch.ne.'R').and.(Ch.ne.'M'))
            read (InFile) Ch
            Ch = UpperCase (Ch)
          end do
                  LeftX = 1
                  RightX = X
                  TopY = 1
                  BottomY = Y
                  if ((RightX .lt. 1 ) .or. (BottomY .lt. 1) ) then
                    call ErrorMess ( 'The sheet is empty.' )
                    return
                  end if
                  do i=1,3
                          do j=1,100
                                  title(i)(j:j)=' '
                          end do
                  end do
             I = 1
             Index = 100
             do while((i.le.3).and.(index.ne.0))
                call PrintMess ( 'Title (' )
                write (OutFile) char (I+48),')'
```

```
                call edit (OutFile,InFile,Title(i),index,100,esc)
                i=i+1
             end do
          call PrintMess ( 'Please wait a moment...' )
          TempFile = OutFile
          OutFile=SaveFile
          do j=1,i-1
                  write (OutFile) Title(j),char(10)
                  if (Ch.eq."M") write (OutFile) "\"
          end do
          do j=TopY,BottomY
                  do i=LeftX,RightX
                       x=i
                       y=j
                    if (Cellpointer(x,y).ne.0) then
                          call WriteData (x,y,.false.)
                    else
                          write (OutFile) Spaces(1:cellwidth(x))
                    end if
                  if (Ch.eq."M") write (OutFile) "\"
                  end do
                     write (OutFile) char(10)
          end do
          OutFile = TempFile
          close (SaveFile)
          return
          end  ! End of Save Print Sheet

          subroutine PSaveWindow ( FileName, Valid )
c         Save a window to a printable text file.
          implicit None
%include 'DGCALC.global'
          character*6 Corner
          logical Valid,esc
          character*100 Title(3)
          character*(*)  FileName
          character*17 OutFileName
          character*1 Command, GetChar, UpperCase, Ch
          integer i,j,index,LeftX,RightX,TopY,BottomY,IErr,x,y, TempFile
          integer TX, TY
          call WndwLocation (LeftX,TopY,RightX,BottomY,Valid)
          if ( .not. Valid ) return
                  if ((RightX .lt. 1 ) .or. (BottomY .lt. 1) ) then
                    call ErrorMess ( 'The sheet is empty.' )
                    return
                  end if
                  do i=1,3
                          do j=1,100
                                  title(i)(j:j)=' '
                          end do
                  end do
             I = 1
             Index = 100
             do while((i.le.3).and.(index.ne.0))
                call PrintMess ( 'Title (' )
                write (OutFile) char (I+48),')'
                call edit (OutFile,InFile,Title(i),index,100,esc)
                i=i+1
             end do
          call PrintMess ('R)egular   or   M)athPlan')
          ch = ' '
          do while ((Ch.ne."R").and.(Ch.ne."M"))
            read(InFile) Ch
```

```
            Ch = uppercase (Ch)
          end do
          call PrintMess ( 'Please wait a moment...' )
          TempFile = OutFile
          OutFile=SaveFile
          do j=1,i-1
                write (OutFile) Title(j),char(10)
            if (Ch.eq.'M') write (OutFile) '\'
          end do
          do j=TopY,BottomY
                do i=LeftX,RightX
                     x=i
                     y=j
                     if (Cellpointer(x,y).ne.0) then
                        call WriteData (x,y,.false.)
                     else
                        write (OutFile) Spaces(1:cellwidth(x))
                     end if
                if (Ch.eq.'M') write (outFile)'\'
                end do
                   write (OutFile) char(10)
          end do
          OutFile = TempFile
          close (SaveFile)
          return
          end  ! End of Save Print Window

          subroutine DMPfileSave (FileName, Valid)
c         Dump the whole sheet to a printable text file.
          implicit none
%include 'DGCalc.global'
          integer i,j,IErr,index,x,y,k, TFile
          character*(*) FileName
          logical Valid
          character*17 DMPfile
          character*6 position
%include 'DGCAlc.celltype'
             call PrintMess ( 'Create a DMP file...' )
             TFile = OutFile
             OutFile=SaveFile
             do i=1,Hsheetsize
                write(TFile) '.'
                do j=1,Vsheetsize
                   if (cellpointer(i,j).ne.0) then
                      x=i
                      y=j
                      index=cellpointer(x,y)
                      write(OutFile) 'Position: '
                      write(OutFile) position(x,y)
                      write(OutFile) 'Type: '
                      if (FormulaArray(index).eq.intcell) then
                         write(OutFile) 'Integer '
                      else if (FormulaArray(index).eq.realcell) then
                         write(OutFile) 'Real    '
                      else if (FormulaArray(index).eq.formulacell) then
                         write(OutFile) 'Formula '
                      else if (FormulaArray(index).eq.truecell) then
                         write(OutFile) 'True    '
                      else if (FormulaArray(index).eq.falsecell)then
                         write(OutFile) 'False   '
                      else if (FormulaArray(index).eq.textcell) then
                         write(OutFile) 'Text    '
                      else
```

```
                        write(OutFile) 'Wrong    '
                     end if
                     k=index+cellheader
                     do while (FormulaArray(k).ne.eos)
                        write(OutFile) FormulaArray(k)
                        k=k+1
                     end do
                     write(OutFile) char(10)
                  end if
               end do
            end do
         OutFile= TFile
         close (SaveFile)
         return
         end   ! End of Create DUMP file


         subroutine DGSaveWindow (FileName, Valid )
c        Save a portion of the working sheet into an output DGCALC file.
         implicit none
%include 'DGCalc.global'
         character*(*) FileName
         logical Valid,Esc
         integer LeftX, RightX, TopY, BottomY, Xmax, Ymax
         integer I, J, TI,TJ, TempX, TempY,TLength,Walk
         character*1 Q,H,L
         character*1 TArray ( ArraySize )
         integer TPointer ( HSheetSize, VSheetSize )
         integer TWidth (HSheetSize)
%include 'DGCALC.CellType'
         call WndwLocation(LeftX,TopY,RightX,BottomY,Valid)
         if ( Valid ) then
c        Initialize Pointers
         do J = 1, VSheetSize
           do I = 1, HSheetSize
              TPointer (I,J) = 0
           end do
         end do
c        Transfer the contents of the window
            call PrintMess ( 'Wait a moment !' )
            TLength = 1
            do J = TopY, BottomY
              do I = LeftX, RightX
                 Walk = CellPointer ( I,J )
                 if ( Walk .ne. 0 ) then
                   TI = I - LeftX + 1
                   TJ = J - TopY + 1
                   TPointer(TI,TJ) = TLength
                   do while (FormulaArray(Walk).ne.eos)
                     TArray(TLength)=FormulaArray(Walk)
                     TLength = TLength + 1
                     Walk = Walk + 1
                   end do
                   TArray (TLength) = FormulaArray(Walk)
                   TLength = TLength + 1
                   TArray(TPointer(TI,TJ)+2)= char (TI)
                   TArray(TPointer(TI,TJ)+3)= char (TJ)
                 end if
                 end do  ! End of I loop
               end do    ! End of J loop
c        Set the cell width for output
                 call PrintMess ( 'Please wait a moment !' )
                 do I = LeftX, RightX
```

```
                  TWidth ( I-LeftX+1 ) = CellWidth (I)
                end do
                do I = RightX-LeftX + 2, HSheetSize
                  TWidth ( I ) = 8
                end do
c       Chech how many rows and columns
        Xmax = 0
        Ymax = 0
        do J = 1, BottomY-TopY+1
          do I = 1, RightX-LeftX+1
                if (TPointer(I,J) .ne. 0) then
                  if (I.gt.Xmax) Xmax=I
                  if (J.gt.Ymax) Ymax=J
                end if
          end do
        end do
c       Save the Header of the window
                call PrintMess ( 'Wait a moment !' )
                call IntToChars ( TLength-1, Q,H,L )
                write ( SaveFile ) eos, Q,H,L
                write ( SaveFile ) char (Xmax)
                write ( SaveFile ) char (Ymax)
c       Save the body of the window
                do I = 1, TLength-1
                        write (SaveFile) TArray(I)
                end do
c       Save pointer of the window
                call PrintMess ( 'Please wait a moment !')
                do J = 1, Ymax
                  do I = 1, Xmax
                    call IntToChars ( TPointer (I,J), Q,H,L )
                    write (SaveFile) Q,H,L
                  end do
                end do
c       Save the cell width of the window
                call PrintMess ( 'Wait a moment !' )
                do I = 1, HSheetSize
                  write(SaveFile) char ( TWidth(I) )
                end do
          close ( SaveFile )
        end if
        return
        end     ! End of Save Window

        subroutine DGSaveSheet (FileName, Valid)
c       Save the whole sheet to a DGCALC file.
c       The sheet has DGCALC format so it can be retrieved by DGCALC.
        implicit none
%include 'DGCalc.global'
        character*1 q,h,l,ch,GetChar,UpperCase
        character*(*) FileName
        character*300  buffer
        integer i,j,k,row,column,rmax,cmax,DataLength
        integer X,Y,Xmax,Ymax
        integer  Index,Number,Counter
        logical esc, Valid
%include 'DGCalc.celltype'
        esc=.false.
        Xmax=0
        Ymax=0
                call CompressData
                DataLength=NextData-1
                do X = 1  ,Hsheetsize
```

```
                  do Y = 1, Vsheetsize
                     if (CellPointer( X, Y ).ne.0) then
                       if ( X .gt. Xmax)  Xmax = X
                       if ( Y .gt. Ymax)  Ymax = Y
                     end if
                  end do
               end do
               call IntToChars(DataLength,q,h,l)
               write(SaveFile) eos,q,h,l
               write(SaveFile) char(Xmax),char(Ymax)
               write(outfile) '.'
               i=1
               counter=1
               do while ((i+299).le.datalength)
                  do k=1,300
                     buffer(k:k)=FormulaArray(i)
                     i=i+1
                  end do
                  write (Savefile) buffer(1:300)
                  counter=counter+1
                  if (mod(counter,8).eq.0) write (outfile) '.'
               end do
               do k=i,DataLength
                  write (Savefile) FormulaArray(k)
               end do
               write (outfile) '.'
               do j=1,Ymax
                  k=1
                  do i=1,Xmax
                     number=CellPointer(i,j)
                     call IntToChars(number,q,h,l)
                     buffer(k:k)=q
                     buffer(k+1:k+1)=h
                     buffer(k+2:k+2)=l
                     k=k+3
                  end do
                  write (Savefile) buffer(1:3*Xmax)
               end do
               write (outfile) '.'
               do i=1,HSheetSize
                  buffer(i:i)=char(CellWidth(i))
               end do
               write (savefile) buffer(1:HSheetSize)
            close(SaveFile)
         return
         end  ! End of Save Sheet

         subroutine TernarySave ( FileName, Valid )
c        This subroutine creates a data file. The data
c    file is unformatted. The first field is '(', the second field is
c    an integer which is the number of columns, the third field is a
c    integer which is the number of rows and followed by real numbers.
c    The rest is a two dimesional array of real numbers.
         implicit none
%include 'DGCalc.global'
%include 'DGCalc.global1'
         character*(*)  FileName
         integer index,i,j,k,LeftX,RightX,TopY,BottomY,IErr,tempI,tempJ
         character*6 Corner,ASCii,Position,Cellname
         character*1 ch,GetChar,UpperCase,sig
         logical Valid,Error
%include 'DGCalc.celltype'
         call WndwLocation (LeftX,TopY,RightX,BottomY,Valid )
```

```
            if (valid) then
               write (OutFile) 'Create data file...'
               write (Savefile) '('
               i=1
               k=RightX-LeftX+1
               write (Savefile) k
               k=BottomY-TopY+1
               write (Savefile) k
               do j=TopY,BottomY
                  do i=LeftX,RightX
                     k=CellPointer(i,j)
                     tempI=i
                     tempJ=j
                     cellname=position(tempI,tempJ)
                     if (k.eq.0) then
                        realvalue=0.0
                        write (Savefile) realvalue
                     else
                        if ((FormulaArray(k).eq.intcell).or.
     &                     (FormulaArray(k).eq.realcell).or.
     &                     (FormulaArray(k).eq.formulacell)) then
                           call GetCellNumber (cellname,error)
                           write (savefile) realvalue
                        else
                           realvalue=0.0
                           write (Savefile) realvalue
                        end if
                     end if
                  end do
               end do
            end if
            close (Savefile)
            return
            end         ! End of Ternary Save File

            subroutine LoadSheet
      c     Load a work sheet, which is previously saved by Save Sheet or
      c     Window command is save options.
            implicit none
      %include 'DGCalc.global'
            character*1 ch,q,high,low,row,column
            character*(20) FileName
            character*300 buffer
            integer i,j,size,IErr,k,counter
            integer index
            logical esc
      %include 'DGCalc.celltype'
            esc=.false.
            FileName=' '
            index=0
            call PrintMess ( 'File to read from : ' )
            call Edit (OutFile,InFile,FileName,index,20,esc)
            if (.not.esc) then
               call Openf(LoadFile,FileName,'R',IErr)
               if (IErr.ne.0) then
                  if (IErr.eq.21) then
                     call ErrorMess('The file does not exist !')
                  else
                     call ErrorMess('Something is wrong !')
                  end if
               else
      c     Initialize all conditions
                   ScreenPtr = 1
```

```
 NextData = 1
CurrX = 1
CurrY = 1
Top = 1
 Left = 1
 do i = 1, HSheetSize
    CellWidth(I) = 8
 end do
 do I = 1, HSheetSize
   do J = 1, VSheetSize
         CellPointer(I,J) = 0
   end do
 end do
read (LoadFile) ch
if (ch.ne.eos) then
    call ErrorMess('File is not a DGCALC file !')
else
    WorkFileName = FileName
    Changed = .false.
    Empty = .false.
    call Pos(22,15,OutFile)
    call ClrEOL
    write (OutFile) 'Loading from the memory...'
    read (LoadFile) q,high,low,row,column
    size=ichar(q)*256*256+ichar(high)*256+ichar(low)
    i=1
    counter=1
    do while ((i+299).le.size)
      read (Loadfile) buffer(1:300)
      do k=1,300
        FormulaArray(i)=buffer(k:k)
        i=i+1
      end do
      counter=counter+1
      if (mod(counter,8).eq.0) write(outfile) '.'
    end do
    do k=i,size
      read (Loadfile) FormulaArray(k)
    end do
    NextData=size+1
    write (outfile) '.'
    do j=1 ,ichar(column)
       k=1
       read (Loadfile) buffer (1:3*ichar(row))
       do i=1, ichar(row)
           ch=buffer(k:k)
           CellPointer(i,j)=ichar(ch)*256*256
           ch=buffer(k+1:k+1)
           CellPointer(i,j)=CellPointer(i,j)+ichar(ch)*256
           ch=buffer(k+2:k+2)
           CellPointer(i,j)=CellPointer(i,j)+ichar(ch)
           k=k+3
       end do
    end do
    write (outfile) '.'
    read (Loadfile) buffer(1:HSheetSize)
    do i=1,HSheetSize
       ch=buffer(i:i)
       CellWidth(i)=ichar(ch)
    end do
    write (outfile) '.'
    do i=ichar(row)+1, Hsheetsize
       do j=1,Vsheetsize
```

```
                        CellPointer(i,j)=0
                    end do
                end do
                write(outfile) *,*
                do i=1, ichar(row)
                    do j=ichar(column)+1, Vsheetsize
                        CellPointer(i,j)=0
                    end do
                end do
                close(LoadFile)
                call DisplaySheet
                call InterpretCell(CurrX,CurrY,White,.false.)
            end if
        end if
    end if
    return
    end   ! End of Load Sheet

    subroutine JoinFile
c   This routine loads another DGCALC data file into DGCALC and
c   puts it to the location specified by its Left Top Corner.
    implicit none
%include 'DGCalc.global'
    character*1 Ch
    character*(40) FileName
    character*(6) Corner
    integer X, Y, X1, Y1, I1, J1
    logical Valid
    integer TempSize, TempNext
    character*1 TempArray (ArraySize)
    integer TempPointer ( HSheetSize, VSheetSize )
    integer TempCellWidth(HSheetSize)
    character*1 q,high,low,row,column
    character*300 buffer
    integer i,j,k, tsize,IErr,counter
    integer index
    logical esc
%include 'DGCalc.celltype'
    esc=.false.
    call PrintMess ( 'File to read from : ' )
    FileName=' '
    index=0
    call Edit (OutFile,InFile,FileName,index,40,esc)
    if (.not.esc) then
       call Openf(LoadFile,FileName,'R',IErr)
       if (IErr.ne.0) then
            Valid = .false.
           if (IErr.eq.21) then
              call ErrorMess('The file does not exist !')
           else
              call ErrorMess('Something is wrong !')
           end if
           return
       else
           read (LoadFile) ch
           if (ch.ne.eos) then
               call ErrorMess('File is not a DGCALC file !')
               Valid = .false.
               close ( LoadFile )
               return
           else
               call InputCellName ( Corner, 'The start location: ' )
               call ValidCell (Corner, X, Y, Valid)
```

```
                if ( .not. Valid ) then
                     call ErrorMess( 'Invalid location !' )
                     return
                end if
                do I = 1, HSheetSize
                   do J = 1, VSheetSize
                     TempPointer(I,J) = 0
                   end do
                end do
                call CompressData
                call PrintMess ( 'Joining from the memory...' )
                read (LoadFile) q,high,low,column, row
                tsize=ichar(q)*256*256+ichar(high)*256+ichar(low)
                if ((tsize + NextData).ge.(ArraySIze - 100)) then
                  call ErrorMess('This file is too large to join !')
                  return
                end if
               X1 = IChar(Column)
               Y1 = IChar(Row)
               if(((X1+X).gt.HSheetSize).or.((Y1+Y).gt.VSheetSize)) then
                 call ErrorMess('Too many rows or column !')
                 return
               end if
                i=1
                counter=1
                do while ((i+299).le.tsize)
                  read (Loadfile) buffer(1:300)
                  do k=1,300
                     TempArray(i)=buffer(k:k)
                     i=i+1
                  end do
                    counter=counter+1
                    if ( mod(counter,8).eq.0) write(outfile) '.'
                end do
                do k=i,tsize
                  read (Loadfile) TempArray(k)
                end do
                write (outfile) '.'
                do j=1  , Y1
                    k=1
                    read (Loadfile) buffer (1:3 * X1)
                    do i=1, X1
                        ch=buffer(k:k)
                        TempPointer(i,j)=ichar(ch)*256*256
                        ch=buffer(k+1:k+1)
                        TempPointer(i,j)=TempPointer(i,j)+ichar(ch)*256
                        ch=buffer(k+2:k+2)
                        TempPointer(i,j)=TempPointer(i,j)+ichar(ch)
                        k=k+3
                    end do
                end do
                write (outfile) '.'
                read (Loadfile) buffer(1:HSheetSize)
                do i=1,HSheetSize
                   ch=buffer(i:i)
                   TempCellWidth(i)=ichar(ch)
                end do
                write (outfile) '.'
                close(LoadFile)
            end if
        end if
      do J = 1,tsize
        FormulaArray(NextData+J-1) = TempArray(J)
```

```
            end do
            do J = 1, Y1
              do I = 1, X1
                    I1 = X + I - 1
                    J1 = Y + J - 1
               if(CellPointer(I1,J1) .ne. 0) then
                    call MarkFree(I1, J1)
                    CellPointer(I1,J1) = 0
               end if
               if ( TempPointer ( I, J ) .ne. 0 ) then
                 CellPointer(I1,J1) = TempPointer(I,J) + NextData - 1
                 FormulaArray(CellPointer(I1,J1)+2) = char (I1)
                 FormulaArray(CellPointer(I1,J1)+3) = char (J1)
               end if
              end do    ! I loop
            end do      ! J loop
            NextData = NextDAta + tsize
            do I = 1, X1
              if (TempCellWidth(I).gt.CellWidth(X + I - 1)) then
                    CellWidth(X + I - 1) = TempCellWidth(I)
              end if
            end do
            Empty = .false.
            Changed = .true.
            call CompressData
            call DisplaySheet
            call InterpretCell(CurrX,CurrY,WHite, .false.)
            end if
            return
            end  ! End of JoinFile

            subroutine DoCliCommand
      c     Escape from DGCALC temporarily and execute any CLI commands
      c     until an (ESC) is typyed.
            implicit none
      %include 'DGCalc.global'
            character*100 command
            integer result,index,i
            logical esc
            call CLS(outfile)
            esc=.false.
            do while (.not.esc)
               write (outfile) 'Please type one CLI command or'
               write (OutFile) ' type (ESC) return to DGCALC :<NL>'
               do i=1, 100
                  command(i:i)=' '
               end do
               index=0
               call edit (outfile,infile,command,index,100,esc)
               if (.not.esc) then
                    write (OutFile) '<NL>'
                    call CLI(command,result)
               end if
               print*,char(10)
            end do
            call DisplaySheet
            call Move (CurrX,CurrY)
            return
            end     ! End of Do Cli Command

            subroutine InttoAscii(number,Ascii)
      c     Convert an integer into its ASC II codes.
            implicit none
```

```
            integer divisor,number,i,index
            character*6 Ascii
            ascii='      '
            divisor=100000
            do while ((number.lt.divisor).and.(divisor.gt.1))
               divisor=divisor/10
            end do
            i=1
            do while (divisor.ge.1)
               index=Int(number/divisor)
               number=number-index*divisor
               ascii(i:i)=char(index+48)
               divisor=divisor/10
               i=i+1
            end do
            return
            end     ! End of Integer convert to ACS ii code

            subroutine MoveArea
c  It moves an area, specified by the Left Top and Right Bottom corner,
c  to another area which is specified by its Left Top corner.
            implicit none
%include 'DGCalc.global'
%include 'DGCalc.global1'
            integer LeftX, RightX, TopY, BottomY, TargetX, TargetY, I, J
            integer TargetX1, TargetY1, TempX, TempY
            character*6 Corner
            character*1 Ch, GetChar, UpperCase, Sig
            logical Valid
%include 'DGCalc.CellType'
            call WndwLocation ( LeftX,TopY,RightX,BottomY,Valid )
            if ( Valid ) then
               call InputCellName ( Corner, 'Top left target corner: ')
               call ValidCell ( Corner, TargetX, TargetY, Valid )
               TargetX1 = TargetX + RightX - LeftX
               TargetY1 = TargetY + BottomY - TopY
               if  ((TargetX .ge. LeftX) .and. (TargetX .le. RightX) .and.
     &             (TargetY .ge. TopY ) .and. (TargetY .le. BottomY) .or.
     &             (TargetX1.ge. LeftX) .and. (TargetX1.le. RightX) .and.
     &             (TargetY1.ge. TopY ) .and. (TargetY1.le. BottomY) .or.
     &             (TargetX1.ge.HSheetSize).or.(TargetY1.ge.VSheetSize))
     &          then
                    call ErrorMess ( 'Invalid location !' )
                    Valid = .false.
               end if
            end if
            if ( Valid ) then
              do I = TargetX, TargetX1
                 do J = TargetY, TargetY1
                    if (CellPointer (I,J) .ne. 0 ) then
                      TempX = I
                      TempY = J
                      call MarkFree (TempX,TempY)
                    end if
                    TempX = LeftX + I - TargetX
                    TempY = TopY + J - TargetY
                    CellPointer(I,J)=CellPointer(TempX,TempY)
                    FormulaArray ( CellPointer(I,J) + 2 ) = char (I)
                    FormulaArray ( CellPointer(I,J) + 3 ) = char (J)
                    CellPointer( TempX, TempY ) = 0
                    if ( CellWidth(I) .lt. CellWidth(TempX) ) then
                         CellWidth(I) = CellWidth(TempX)
                    end if
```

```
                end do  ! One column is moved.
              end do  ! The whole area is moved.
              call DisplaySheet
              call InterpretCell ( CurrX, CurrY, White, .false. )
            end if
            return
            end    ! End of Move an Area

c       End of OGCALC_COMMANDS.F77
```

+4444444444

```
      subroutine Push (Valtype)
      implicit none
%include 'DGCalc.global1'
      character*11 Valtype
      if (Valtype.eq.'Inttype    ') then
         IntStackPtr=IntStackPtr+1
         if (IntStackPtr.gt.50) then
              call ErrorMess ('Stack is overflow')
         else
              IntStack(IntStackPtr)=Intvalue
         end if
      else if (Valtype.eq.'Realtype   ') then
         RealStackPtr=RealStackPtr+1
         if (RealStackPtr.gt.50) then
              call ErrorMess ('Stack is overflow')
         else
              RealStack(RealStackPtr)=realvalue
         end if
      else if (Valtype.eq.'Booltype   ') then
         BoolStackPtr=BoolStackPtr+1
         if (BoolStackPtr.gt.50) then
              call ErrorMess ('Stack is overflow')
         else
              BoolStack(BoolStackPtr)=boolvalue
         end if
      else if (Valtype.eq.'Stringtype ') then
         StringStackPtr=StringStackPtr+1
         if (StringStackPtr.gt.10) then
              call ErrorMess ('Stack is overflow')
         else
              StrLenStack(StringStackPtr)=stringlen
              StringStack(StringStackPtr)=stringvalue
         end if
      end if
      return
      end

      real function PopReal (errorflag)
      implicit none
      logical errorflag
      integer RealStackPtr
      real RealStack(50)
      common /RPtr/ RealStackPtr
      common /RStack/ RealStack
      if (RealStackPtr.gt.0) then
         PopReal=RealStack(RealStackPtr)
         RealStackPtr=RealStackPtr-1
      else
         call ErrorMess ('Real stack underflow')
         errorflag=.true.
      end if
      return
      end

      integer function PopInt ( )
      implicit none
      integer IntStackPtr
      integer IntStack(50)
      common /IStack/ IntStack
      common /IPtr/ IntStackPtr
      logical errorflag
      if (IntStackPtr.gt.0) then
         PopInt=IntStack(IntStackPtr)
```

```
            IntStackPtr=IntStackPtr-1
         else
            call ErrorMess ('Integer stack underflow')
            errorflag=.true.
         end if
         return
         end

         logical function PopBool (errorflag)
         implicit none
         integer BoolStackPtr
         logical BoolStack(50)
         common /BPtr/ BoolStackPtr
         common /BStack/ BoolStack
         logical errorflag
         if (BoolStackPtr.gt.0) then
            PopBool=BoolStack(BoolStackPtr)
            BoolStackPtr=BoolStackPtr-1
         else
            call ErrorMess ('Boolern stack underflow')
            errorflag=.true.
         end if
         return
         end

         character*240 function PopString (errorflag)
         implicit none
         integer StringStackPtr,StringLen
         integer StrLenStack(10)
         character*240 StringStack(10)
         common /SPtr/ StringStackPtr,StrLenStack,StringLen
         common /SStack/ StringStack
         logical errorflag
         if (StringStackPtr.gt.0) then
            PopString=StringStack(StringStackPtr)
            StringLen=StrLenStack(StringStackPtr)
            StringStackPtr=StringStackPtr-1
         else
            call ErrorMess ('String stack underflow')
            errorflag=.true.
         end if
         return
         end

         subroutine InitResWords
         implicit none
%include 'DGCalc.global1'
         resword( 1)='OR          '
         resword( 2)='DIV         '
         resword( 3)='AND         '
         resword( 4)='NOT         '
         resword( 5)='TRUE        '
         resword( 6)='@SIN        '
         resword( 7)='@COS        '
         resword( 8)='@TAN        '
         resword( 9)='@MIN        '
         resword(10)='@MAX        '
         resword(11)='FALSE       '
         resword(12)='@LOG        '
         resword(13)='@SUM        '
         restoken( 1)='Orsy        '
         restoken( 2)='Divsy       '
         restoken( 3)='Andsy       '
```

```
            restoken( 4)='Notsy      '
            restoken( 5)='Truesy     '
            restoken( 6)='Sinsy      '
            restoken( 7)='Cossy      '
            restoken( 8)='Tansy      '
            restoken( 9)='Minsy      '
            restoken(10)='Maxsy      '
            restoken(11)='Falsesy    '
            restoken(12)='Logarithmsy'
            restoken(13)='Sumsy      '
            return
            end

            subroutine InitScalars
            implicit none
    %include 'DGCalc.global1'
            currentpos=1
            currentchar=' '
            nextchar=' '
            eol=1
            IntStackPtr=1
            RealStackPtr=1
            BoolStackPtr=1
            StringStackPtr=1
            return
            end

            subroutine Error (Fault)
            implicit none
            integer OutFile
            common /Files/ OutFile
            character*(*) Fault
            call Pos (23,15,OutFile)
            call ClrEOL
            write (OutFile) '****'
            if (Fault.eq.'Toolong') then
               write (OutFile) 'Identifier is too long.'
            else if (Fault.eq.'Overflow') then
               write (OutFile) 'Integer constant','<76>','maxint.'
            else
               write (OutFile) 'Need one more " to form string constant'
            end if
            write (OutFile) '****'
            return
            end

            subroutine GetCurrChar
            implicit none
    %include 'DGCalc.params'
    %include 'DGCalc.global'
    %include 'DGCalc.global1'
            if (currentpos.lt.NextData) then
               currentchar=FormulaArray(currentpos)
               nextchar=FormulaArray(currentpos+1)
               currentpos=currentpos+1
            end if
            return
            end

            subroutine GetNumber
            implicit none
    %include 'DGCalc.global1'
            real divisor
```

```
            divisor=10.0
            realvalue=0.0
            do while ((currentchar.ge.'0').and.(currentchar.le.'9'))
               realvalue=(10*realvalue)+(ichar(currentchar)-ichar('0'))
               call GetCurrChar
               if (currentchar.eq.'.') then
                    call GetCurrChar
                    do while ((currentchar.ge.'0').and.(currentchar.le.'9'))
                       realvalue=realvalue+(ichar(currentchar)-ichar('0'))/
     &                    divisor
                       divisor=divisor*10.0
                       call GetCurrChar
                    end do
               end if
            end do
            token='Realconst  '
            currentpos=currentpos-1
            return
            end

            subroutine GetIdent
            implicit none
    %include 'DGCalc.global1'
            integer index
            index=1
            if (currentchar.eq.'@') then
               do while(((currentchar.ge.'@').and.(currentchar.le.'Z')).or.
     &                 ((currentchar.ge.'a').and.(currentchar.le.'z')).and.
     &                 (index.le.11))
                  ident(index:index)=currentchar
                  call GetCurrChar
                  index=index+1
               end do
            else
              do while ((((currentchar.ge.'0').and.(currentchar.le.'9')).or.
     &         ((currentchar.ge.'@').and.(currentchar.le.'Z')).or.
     &         ((currentchar.ge.'a').and.(currentchar.le.'z'))).and.
     &         (index.le.11))
                  ident(index:index)=currentchar
                  call GetCurrChar
                  index=index+1
              end do
            end if
            identlen=index-1
            do while (index.le.11)
               ident(index:index)=' '
               index=index+1
            end do
            currentpos=currentpos-1
            return
            end

            subroutine SkipComment
            implicit none
    %include 'DGCalc.global1'
    %include 'DGCalc.celltype'
            call GetCurrChar
            do while ((currentchar.ne.eos).and.((currentchar.ne.'*').or.
     &         (nextchar.ne.')')))
               call GetCurrChar
            end do
            if (currentchar.ne.eos) then
               call GetCurrChar
```

PRINTED IN U.S.A.

```
            call GetCurrChar
         end if
         return
         end

         subroutine GetConst
         implicit none
%include 'Dgcalc.global1'
         character*1 eos
         eos=char(253)
         stringlen=1
         call GetCurrChar
         do while ((currentchar.ne.eos).and.(currentchar.ne.'"'))
            stringvalue(stringlen:stringlen)=currentchar
            stringlen=stringlen+1
            call GetCurrChar
         end do
         if (Stringlen.gt.1) then
            stringlen=stringlen-1
         end if
         if (currentchar.ne.'"') then
            call Error ('Cherror ')
         end if
         return
         end

         logical function Compare (Key,Id,Len)
         implicit none
         integer index,Len
         logical match
         character*(*) Key,Id
         character*1 uppercase
         index=1
         match=.true.
         do while(match.and.(index.le.len))
            if (uppercase(Key(index:index)).ne.uppercase(id(index:index))) then
                  match=.false.
            else
                  index=index+1
            end if
         end do
         compare=match
         return
         end

         subroutine ProcessIdent
         implicit none
%include 'Dgcalc.global1'
         logical found,valid,compare
         integer index,X,Y
         token='Invalidsy  '
         found=.false.
         index=1
         do while ((index.lt.24).and.(.not.(found)))
            if (compare(resword(index),ident,identlen)) then
                  found=.true.
            else
                  index=index+1
            end if
         end do
         if (found) then
            token=restoken(index)
         else
```

```
            call ValidCell(ident,X,Y,valid)
            if (.not.valid) then
                 token='Invalidsy  '
            else
                 token='Cellsy     '
            end if
         end if
         return
         end

         subroutine returntk (tk,size)
         implicit none
%include 'Dgcalc.global1'
         integer counter,size
         character*11 tk
         do counter=1,size
            call GetCurrChar
         end do
         token=tk
         return
         end

         subroutine GetToken
         implicit none
%include 'Dgcalc.global1'
         integer NextData
         common / IPointer / NextData
         character*1 eos
         eos=char(253)
         eol=NextData
         if (currentpos.gt.eol) then
            call returntk('Eossy      ',0)
      else
            call GetCurrChar
            do while ((currentchar.eq.' ').and.(nextchar.ne.eos))
                 call GetCurrChar
            end do
            if ((currentchar.eq.'(').and.(nextchar.eq.'*')) then
                 call skipcomment
            end if
            if (((currentchar.ge.'@').and.(currentchar.le.'Z')).or.
     &           ((currentchar.ge.'a').and.(currentchar.le.'z'))) then
                 call GetIdent
                 call ProcessIdent
            else
                 if ((currentchar.ge.'0').and.(currentchar.le.'9')) then
                    call GetNumber
                 else
                    if (currentchar.eq.'(') then
                         call returntk('Lparensy   ',0)
                    else if (currentchar.eq.')') then
                         call returntk('Rparensy   ',0)
                    else if (currentchar.eq.'*') then
                         if (nextchar.eq.'*') then
                            call returntk ('Exponentop ',1)
                         else
                            call returntk('Multop     ',0)
                         end if
                    else if (currentchar.eq.'/') then
                         call returntk('Divsy      ',0)
                    else if (currentchar.eq.'%') then
                         call returntk('Percentage ',0)
                    else if (currentchar.eq.'+') then
```

```
                    call returntk('Addop      ',0)
                  else if (currentchar.eq.'-') then
                    call returntk('Minusop    ',0)
                  else if (currentchar.eq.'"') then
                    call GetConst
                    call returntk('Stconst    ',0)
                  else if (currentchar.eq.'.') then
                    if (nextchar.eq.'.') then
                      call returntk('Thrusy     ',1)
                    else if ((nextchar.eq.'T').or.(nextchar.eq.'t'))
     &                then
                      call returntk('Truesy     ',5)
                    else if ((nextchar.eq.'F').or.(nextchar.eq.'f'))
     &                then
                      call returntk('Falsesy    ',6)
                    end if
                  else if (currentchar.eq.':') then
                    if (nextchar.eq.'=') then
                      call returntk('Assop      ',1)
                    else
                      call returntk('Colonsy    ',0)
                    end if
                  else if (currentchar.eq.';') then
                    call returntk('Semicolonsy',0)
                  else if (currentchar.eq.'<074>') then
                    if (nextchar.eq.'=') then
                      call returntk('Leop       ',1)
                    else if (nextchar.eq.'<076>') then
                      call returntk('Neop       ',1)
                    else
                      call returntk('Ltop       ',0)
                    end if
                  else if (currentchar.eq.'=') then
                    call returntk('Eqop       ',0)
                  else if (currentchar.eq.'<076>') then
                    if (nextchar.eq.'=') then
                      call returntk('Geop       ',1)
                    else
                      call returntk('Gtop       ',0)
                    end if
                  else if (currentchar.eq.'[') then
                    call returntk('Lbracksy   ',0)
                  else if (currentchar.eq.']') then
                    call returntk('Rbracksy   ',0)
                  else
                    call returntk('Illegalsy  ',0)
                  end if
                end if
              end if
            end if
            return
            end

            subroutine GetCellValue(row,column,celloftype)
            implicit none
    %include 'DGCalc.params'
    %include 'DGCalc.global'
            logical notok
            integer row,column,i,sigh,original
            character*11 celloftype
            character*1 cellty
    %include 'DGCalc.global1'
    %include 'DGCalc.celltype'
```

```
      celloftype='Formula     '
      sigh=1
      if (CellPointer(row,column).ne.0) then
         currentpos=CellPointer(row,column)+CellHeader
         original=cellpointer(row,column)
         celltY=FormulaArray(currentpos-cellheader)
         if (celltY.eq.textcell) then
              do i=1,240
                 stringvalue(i:i)=' '
              end do
              i=currentpos
              do while (FormulaArray(i).ne.eos)
                  stringvalue(i+1-currentpos:i+1-currentpos)=FormulaArray(i)
                  i=i+1
              end do
              token='Stconst    '
              CellofType=token
         else if (celltY.eq.formulacell) then
              if (FormulaArray(currentpos).ne.'#') then
                 call errormess('Missing a "#" in front of fomular')
                 celloftype='Illegalsy  '
              else
                 currentpos=currentpos+1
                 call Transformation (notok)
                 if (notok) then
                    celloftype='Illegalsy  '
                    FormulaArray(original)=wrongcell
                 end if
              end if
         else if ((celltY.eq.realcell).or.(celltY.eq.intcell)) then
              call GetToken
              if (token.eq.'Addop      ') then
                 call GetToken
              else if (token.eq.'Minusop    ') then
                 sigh=-1
                 call GetToken
              end if
              if (token.eq.'Realconst  ') then
                 realvalue=realvalue*sigh
              else
              celloftype='Illegalsy  '
              end if
              celloftype=token
          else if (celltY.eq.truecell) then
              token='Truesy     '
              celloftype=token
              boolvalue=.true.
          else if (celltY.eq.falsecell) then
              token='Falsesy    '
              celloftype=token
              boolvalue=.false.
          else
              celloftype='Illegalsy  '
          end if
       else
          celloftype='Illegalsy  '
       end if
       return
       end

       subroutine Evaluation (termtype,tokenlist,reallist,errorflag)
       implicit none
       real real1,real2, reallist(50),PopReal
```

```
      integer index
      character*11 tokenlist(50),termtype,operation
      logical errorflag,LegalExpr
      real realvalue,realstack(50)
      common /RStack/ RealStack,realvalue
      errorflag=.false.
      index=1
      if (LegalExpr(Tokenlist)) then
         if (termtype.eq.'Realtype   ') then
              operation=tokenlist(index)
              do while ((operation.ne.'Semicolonsy').and.(.not.errorflag))
                 if (operation.eq.'Realconst  ') then
                      realvalue=reallist(index)
                      call Push('Realtype   ')
                 else if (operation.eq.'Addop      ') then
                      real1=Popreal(errorflag)
                      real2=PopReal(errorflag)
                      realvalue=real1+real2
                      call Push('Realtype   ')
                 else if (operation.eq.'Minusop    ') then
                      real1=PopReal(errorflag)
                      real2=PopReal(errorflag)
                      realvalue=real2-real1
                      call Push('Realtype   ')
                 else if (operation.eq.'Multop     ') then
                      real1=PopReal(errorflag)
                      real2=PopReal(errorflag)
                      realvalue=real1*real2
                      call Push('Realtype   ')
                 else if (operation.eq.'Divsy      ') then
                      real1=PopReal(errorflag)
                      real2=PopReal(errorflag)
                      if (real1.lt.1.0E-20) then
                         errorflag=.true.
                         call ErrorMess('Real number divided by zero')
                      else
                         realvalue=real2/real1
                         call Push('Realtype   ')
                      end if
                 else if (operation.eq.'Exponentop ') then
                      real1=Popreal(errorflag)
                      real2=PopReal(errorflag)
                      realvalue=real2**real1
                      call Push('Realtype   ')
                 else if (operation.eq.'Logarithmsy') then
                      real1=PopReal(errorflag)
                      if (real1.gt.0) then
                         realvalue=alog10(real1)
                         call Push ('Realtype   ')
                      else
                         errorflag=.true.
                         call errormess ('Argument of LOG must GT 0')
                      end if
                 else if (operation.eq.'Sinsy      ') then
                      real1=PopReal(errorflag)
                      realvalue=sin(real1)
                      call Push('Realtype   ')
                 else if (operation.eq.'Cossy      ') then
                      real1=PopReal(errorflag)
                      realvalue=cos(real1)
                      call Push('Realtype   ')
                 else if (operation.eq.'Tansy      ') then
                      real1=PopReal(errorflag)
```

```
                realvalue=tan(real1)
                call Push('Realtype    ')
            else if (operation.eq.'Percentage ') then
                real1=PopReal(errorflag)
                realvalue=100*real1
                call Push('Realtype    ')
            else
                errorflag=.true.
                call ErrorMess('Illegal operation in expression')
            end if
            index=index+1
            if (index.eq.50) then
                errorflag=.true.
                call ErrorMess('Formala is too long')
            else
                operation=tokenlist(index)
            end if
         end do
         if (.not.errorflag) then
            realvalue=PopReal(errorflag)
         else
            realvalue=0.0
         end if
      else
            realvalue=0.0
            errorflag=.true.
            call ErrorMess ('Illegal type of expression')
      end if
   else
      realvalue=0.0
      call ErrorMess ('Formula is wrong ')
         errorflag=.true.
      end if
      return
      end

      logical function LegalExpr(list)
      implicit none
      logical SysFunction
      character*11 list(50)
      integer index,operation, operator
      legalexpr=.true.
      index=1
      operation=0
      operator=0
      do while ((list(index).ne.'Semicolonsy').and.(index.le.50))
         if (list(index).eq.'Realconst  ') then
            operator=operator+1
         else
            if (.not.(SysFunction(list(index)))) then
               operation=operation+1
            end if
         end if
         index=index+1
      end do
      if (operation.ne.(operator-1)) legalexpr=.false.
      if (index.gt.50) legalexpr=.false.
      return
      end

      subroutine Transformation(errorflag)
      implicit none
%include 'DGCalc.global1'
```

```
      character*11 typeofexpr,tokenoflist(50),tokenstack(50)
      real realoflist(50)
      integer index1,index2
      logical errorflag,SysFunction
      typeofexpr='Realtype   '
      errorflag=.false.
      index1=1
      index2=1
      tokenstack(index1)='Lparensy   '
      call GetToken
      do while ((token.ne.'Illegalsy  ').and.(token.ne.'Semicolonsy')
     &          .and.(typeofexpr.eq.'Realtype   ').and.(.not.errorflag))
         if (token.eq.'Realconst  ') then
              realoflist(index2)=realvalue
              tokenoflist(index2)=token
              index2=index2+1
         else if (token.eq.'Cellsy     ') then
              call GetCellNumber(ident,errorflag)
              realoflist(index2)=realvalue
              tokenoflist(index2)='Realconst  '
              index2=index2+1
         else if ((token.eq.'Maxsy      ').or.(token.eq.'Minsy      ')
     &             .or.(token.eq.'Sumsy      ')) then
              call GetSumMaxMin (token,errorflag)
              realoflist(index2)=realvalue
              tokenoflist(index2)='Realconst  '
              index2=index2+1
         else if (token.eq.'Lparensy   ') then
              index1=index1+1
              tokenstack(index1)=token
         else if (token.eq.'Rparensy   ') then
              do while(tokenstack(index1).ne.'Lparensy   ')
                 tokenoflist(index2)=tokenstack(index1)
                 index1=index1-1
                 index2=index2+1
              end do
              index1=index1-1
         else if ((token.eq.'Addop      ').or.(token.eq.'Minusop    ')) then
              do while (tokenstack(index1).ne.'Lparensy   ')
                 tokenoflist(index2)=tokenstack(index1)
                 index1=index1-1
                 if (index1.eq.0) then
                      errorflag=.true.
                      index1=1
                 end if
                 index2=index2+1
              end do
              index1=index1+1
              tokenstack(index1)=token
         else if ((token.eq.'Multop     ').or.(token.eq.'Divsy      ')) then
              do while ((tokenstack(index1).eq.'Multop     ').or.
     &                  (tokenstack(index1).eq.'Divsy      '))
                 tokenoflist(index2)=tokenstack(index1)
                 index2=index2+1
                 index1=index1-1
              end do
              do while (tokenstack(index1).eq.'Exponentop ')
                 tokenoflist(index2)=tokenstack(index1)
                 index1=index1-1
                 index2=index2+1
              end do
              do while (SysFunction(tokenstack(index1)))
                 tokenoflist(index2)=tokenstack(index1)
```

```
                  index1=index1-1
                  index2=index2+1
               end do
               index1=index1+1
               tokenstack(index1)=token
            else if (token.eq.'Exponentop ') then
               do while (tokenstack(index1).eq.'Exponentop ')
                  tokenoflist(index2)=tokenstack(index1)
                  index1=index1-1
                  index2=index2+1
               end do
               do while (SysFunction(tokenstack(index1)))
                  tokenoflist(index2)=tokenstack(index1)
                  index1=index1-1
                  index2=index2+1
               end do
               index1=index1+1
               tokenstack(index1)=token
            else if (SysFunction(token)) then
               index1=index1+1
               tokenstack(index1)=token
            else
               typeofexpr='Stringtype '
            end if
            call GetToken
         end do
         if ((token.eq.'Semicolonsy').and.(.not.errorflag)) then
            do while ((tokenstack(index1).ne.'Lparensy   ').and.
     &           (.not.errorflag))
               tokenoflist(index2)=tokenstack(index1)
               index1=index1-1
               if (index1.eq.0) then
                  errorflag=.true.
                  index1=1
               end if
               index2=index2+1
            end do
            tokenoflist(index2)=token
            if (index1.eq.1) then
               call Evaluation (typeofexpr,tokenoflist,realoflist,errorflag)
            else
               realvalue=0.0
               call ErrorMess('Formular is wrong')
               errorflag=.true.
            end if
         else
            realvalue=0.0
            if (.not.errorflag) then
               call ErrorMess('Missing a ";" at the end or wrong symbol')
            end if
            errorflag=.true.
         end if
         return
         end

         subroutine GetCellNumber (Cellname,error)
         implicit none
%include 'DGCalc.params'
%include 'DGCalc.global'
%include 'DGCalc.global1'
         logical error,valid
         character*(*) CellName
         character*11 celloftype
```

```
      integer x,y,i,j,sign,temp
%include 'DGCalc.celltype'
      sign=1
      error=.false.
      call ValidCell(CellName,x,y,valid)
      i=CellPointer(x,y)
      temp=currentpos
      if ((i.eq.0).or.((x.eq.currX).and.(y.eq.currY))) then
         realvalue=0
      else
         if ((FormulaArray(i).eq.intcell).or.(FormulaArray(i).eq.
     &          realcell)) then
              currentpos=i+cellheader
              call GetToken
              if (token.eq.'Minusop    ') then
                 sign=-1
                 call GetToken
              else if (token.eq.'Addop      ') then
                 call Gettoken
              end if
              realvalue=realvalue*sign
         else if (FormulaArray(i).eq.formulacell) then
              j=i+cellheader
              do while (FormulaArray(j).ne.eos)
                 j=j+1
              end do
              j=j-11
              currentpos=j
              call GetToken
              if (token.eq.'Minusop    ') then
                 sign=-1
                 call GetToken
              else if (token.eq.'Addop      ') then
                 call GetToken
              end if
              realvalue=realvalue*sign
         else
              error=.true.
              call errormess('There is a cell which can not be quoted in formula')
         end if
         currentpos=temp
      end if
      return
      end

      logical function SysFunction(term)
      implicit none
      character*11 term
      if ((term.eq.'Sinsy      ').or.(term.eq.'Cossy      ').or.
     &    (term.eq.'Tansy      ').or.(term.eq.'Logarithmsy').or.
     &    (term.eq.'Percentage ')) then
         SysFunction=.true.
      else
         SysFunction=.false.
      end if
      return
      end

      subroutine GetSumMaxMin (ope,error)
      implicit none
%include 'DGCalc.global1'
      character*11 operation,ope
      logical error,valid
```

```
      integer x1,x2,y1,y2,i,j,tempX,tempY
      real sum,min,max
      character*6 position, tempcell
      operation=ope
      valid=.false.
      error=.false.
      sum=0
      max=-1e11
      min=1e11
      call Gettoken
      if (token.eq.'Lparensy   ') then
         call gettoken
         if (token.eq.'Cellsy     ') then
            call validcell(ident,x1,y1,valid)
            valid=.false.
            call gettoken
            if (token.eq.'Gtop       ') then
               call gettoken
               if (token.eq.'Cellsy     ') then
                  call validcell(ident,x2,y2,valid)
                  valid=.false.
                  call gettoken
                  if (token.eq.'Rparensy   ') then
                     if ((x1.le.x2).and.(y1.le.y2)) valid=.true.
                  end if
               end if
            end if
         end if
      end if
      if (valid) then
         do i=x1,x2
            do j=y1,y2
               tempX=i
               tempY=j
               tempCell=Position(tempX,tempY)
               call getcellnumber(tempCell,error)
               if (error) then
                  call errormess('There is a wrong cell')
                  error=.true.
                  goto 10
               else
                  if (operation.eq.'Sumsy      ') then
                     sum=sum+realvalue
                  else if (operation.eq.'Maxsy      ') then
                     if (max.lt.realvalue) then
                        max=realvalue
                     end if
                  else if (operation.eq.'Minsy      ') then
                     if (min.gt.realvalue) then
                        min=realvalue
                     end if
                  end if
               end if
            end do
         end do
      else
         call errormess ('Wrong cell range ')
         error=.true.
      end if
10    if (valid.and..not.error) then
         if (operation.eq.'Sumsy      ') then
            realvalue=sum
         else if (operation.eq.'Maxsy      ') then
```

```
            realvalue=max
         else
            realvalue=min
         end if
      else
         error=.true.
         realvalue=0
      end if
      return
      end
```

```
 $$$$   $$$$  $    $  $    $  $$$$$  $$$$$
$       $     $    $  $    $  $      $
$       $     $    $  $    $  $      $
$       $     $ $  $  $$$$$  $      $
$  $$   $  $$  $ $  $  $    $  $      $
$    $  $    $ $$ $$  $    $  $      $
 $$$$   $$$$  $    $  $    $  $$$$$  $

   DEST=GGWHIT  USER=GGWHIT  QUEUE=LPT  DEVICE=@LPB
 SEQ=21183  QPRI=137  LPP=66  CPL=132  COPIES=1  LIMIT=49

    CREATED:  25-AUG-87  17:32:16
  ENQUEUED:  31-DEC-87  11:19:48
  PRINTING:  31-DEC-87  11:23:33

 PATH=:SI20:GGWHIT:$DGCALC:DGCALC_HELP.F77

$$$$    $$$$   $$$     $    $       $$$          $    $  $$$$$  $       $$$$          $$$$$  $$$$$  $$$$$
$    $  $      $   $   $ $  $      $   $         $    $  $      $       $   $         $          $      $
$    $  $      $       $   $  $      $            $    $  $      $       $   $         $          $      $
$    $  $      $       $   $  $      $            $$$$$  $$$$   $       $$$$          $$$$       $      $
$    $  $  $$  $       $$$$$  $      $            $    $  $      $       $             $          $      $
$    $  $   $  $    $  $    $  $      $   $        $    $  $      $       $             $          $      $
$$$$    $$$$   $$$   $   $  $$$$$   $$$   $$$$$  $    $  $$$$$  $$$$$  $        $$   $          $      $

Kansas Geological Survey AOS/VS

AOS/VS REV 07.57
AOS/VS XLPT REV 07.56
```

```
c       This is DGCALC_HELP.F77 for DGCALC1 version.  May 28, 1987
c       There are subroutines to print help messages.
c       Help is called by the main program, DGCALC.F77.
c       Other subroutines are called by Help or sub help subroutines.

        subroutine Help
c       This subrouting prints Help messages, telling the
c       user more information about the DGCALC and how to
c       use the DGCLAC.
        implicit none
%include 'DGCalc.global'
        character*1 Command, GetChar
        call DGHelp ( Command )  ! Information of DGCALC
        if ( Command .eq. 'C') then
             call CHelp     ! Copy Help
        else if (Command .eq. 'D') then
                call DHelp  ! Delete Help
        else if ( Command .eq. 'E' ) then
                call EHelp  ! Erase Help
        else if (Command .eq. 'F') then
                call FHelp  ! Form Help
        else if (Command .eq. 'G') then
                call GHelp  ! Goto Help
        else if (Command .eq. 'H') then
                call HHelp  ! Help Help
        else if (Command .eq. 'I') then
                call IHelp  ! Insert Help
        else if ( Command .eq. 'J' ) then
                call JHelp  ! Join Help
        else if (Command .eq. 'L') then
                call LHelp  ! Load Help
        else if (Command .eq. 'M') then
                call MHelp  ! Move Help
        else if (Command .eq. 'P') then
                print*,'<007>'!  call PHelp  ! Print Help
        else if (Command .eq. 'Q') then
                call QHelp  ! Quit Help
        else if (Command .eq. 'R') then
                call RHelp  ! Recalculate Help
        else if (Command .eq. 'S') then
                call SHelp  ! Save Help
        else if (Command .eq. 'T') then
                call THelp  ! Type Help
        else if (Command .eq. '$') then
                call DollarHelp  ! CLI Command Help
        else
             print *, '<007><007>'    ! Bell alarm
        end if
        call DisplaySheet
        call Move (CurrX, CurrY)
        return
        end

        subroutine DGHelp ( Command )
c       This procedure prints messages about DGCALC
%include 'DGCalc.global'
        character*1 Command, GetChar, UpperCase
        call CLS ( outfile )  ! Clear the screen
        call Pos ( 0, 0, outfile )
        print 50
50      format ( '   DGCALC is a calculation spreadsheet for the DATA GENERAL computer.'/'
     &  It has 100 columns and 255 rows. The columns are labeled by A,B,C,...Y,Z,AA,AB,'/'
     &  ...CV. The rows are labeled by 1,2,...255. The location is specified by its row'/'
```

```
     &  and column labels. Text, numeric data, or formulas can be entered or modified"/"
     &  in a cell by moving the cursor to that cell and using the appropriate cell type"/"
     &  command.  The contents of a cell or a group of cells (window) can be copied,"/"
     &  deleted, erased, saved, loaded, recalculated, moved, or printed. Columns or"/"
     &  rows can be inserted or deleted. Column width and numeric precision can be"/"
     &  changed. The cursor can be moved to any place of the spreadsheet by a cell"/"
     &  location command, as well as by the following commands :"//"
     &                  Commands to move the cursor"/"
     &          The ARROW keys move the cursor one position at a time"/"
     &          The SHIFT ARROW keys move the cursor one screen at a time"/"
     &     HOME : Move to the beginning of sheet (upper left-hand corner)"/"
     &     C1  : Move up to the top of sheet ( in the same column)"/"
     &     C2  : Move down to the bottom of sheet (in the same column)"/"
     &     C3  : Move to the left end of sheet ( in the same row)"/"
     &     C4  : Move to the right end of sheet (in the same row)")
          call pos ( 20, 0, outfile )
          write (outfile) "Type one Command for more information."
          write (outfile) "Type (ESC) to resume the spreadsheet.<012>"
          write (outfile)" C)opy D)elete E)rase F)orm G)oto H)elp I)nsert
     &  J)oin  L)oad<012>"
          write (outfile)" M)ove Q)uit  R)ecalc S)ave T)ype $)CLI Command"
          Command = "k"
          do while ((Command .ne. "C") .and. (Command .ne. "D")
     &      .and. (Command .ne. "E") .and. (Command .ne. "F")
     &      .and. (Command .ne. "G") .and. (Command .ne. "H")
     &      .and. (Command .ne. "I") .and. (Command .ne. "J")
     &      .and. (Command .ne. "L") .and. (Command .ne. "M")
     &                               .and. (Command .ne. "Q")
     &      .and. (Command .ne. "R") .and. (Command .ne. "S")
     &      .and. (Command .ne. "T") .and. (Command .ne. "$")
     &      .and. (Command .ne. Char (27)))
               Command = GetChar (Infile)
               Command = UpperCase (Command)
               end do
          return
          end         ! End of DGHelp

          subroutine CHelp
c         This subroutine prints information about Copy command
%include "DGCalc.global"
          character*1 Command, GetChar
          call CLS (OutFile)    ! Clear screen
          call pos (0, 1, OutFile)
          print 50          ! Print message about copy command
50        format ("  The Copy command copies the contents of one cell or a window, to another"/"
     &    window."/"
     &    After the command is typed, the following appears:"//"
     &                C)ell  or  W)indow"//"
     &    Type C to copy a cell to a window and type W to copy a window to another"/"
     &    window."/"
     &    If the copied cell contents are formula type, the following appears:"//"
     &                R)elative   or   N)o adjustment"//"
     &    Type R to copy a math formula from the source cell to the same column and"/"
     &    This command also changes the variable names (cell location of values used in"/"
     &    the formula) to the proper row labels thereby adjusting the equation for each"/"
     &    cell location so that it uses the correct input data in each row."//"
     &    Type N to copy the same contents to the destination window. The window is"/"
     &    specified by its top left and bottom right corners." )
          call PrintMess ("Type any key to continue or type (ESC) to resume DGCALC." )
          Command = GetChar(InFile)
          if ( Command .ne. char ( 27 ) ) then
                  call CLS (OutFile)
                  print 60
```

```
60      format ( '  To copy from a window to another window, be sure there is no overlap between'/'
     &  source and destination windows.'//'
     &  The syntax of the window is :'//'
     &        Top left corner : [ Cell location of top left corner : ]'/'
     &        Bottom right corner : [ Cell location of bottom right corner ]'//'
     &  For example, if you want to copy the contents of cell D5 into a window with the'/'
     &  location G10 for the top left corner and K20 for the bottom right corner.'/'
     &  you need to type :'//'
     &        Copy from : [D5]'/'
     &        Top left corner : [G10]'/'
     &        Bottom right corner : [K20]')
         call pos (20,0,OutFile)
         call PrintMess ('Type any key to continue.')
         Command = GetChar (Infile)
         end if
         return
         end        ! End of Copy Help

         subroutine DHelp
c        This subroutine prints information about delete command
%include 'DGCalc.global'
         character*1 Command, GetChar
         call CLS (OutFile)    ! Clear screen
         call pos (1, 1, OutFile)
         print 50
50       format ('  The Delete command deletes (removes from the spreadsheet) Rows or Columns'/'
     &  of the working sheet.'//'
     &  After the command is typed, the following appears:'//'
     &              R)ow         C)olumn'//'
     &     type C to delete a column or columns. Then answer the question:'/'
     &        How many columns ( 0-9 ) ?'//'
     &     Type R to delete a row or rows, then answer the question:'/'
     &        How many rows ( 0-9 ) ?'//'
     &     Type 0 to delete nothing, type 1 to delete the current column or row,'/'
     &     type 9 to delete 9 columns or rows starting from the current one.'//'
     &     Rows are counted from the current row to the bottom.'/'
     &     Columns are counted from the current column to the right.')
         call PrintMess ( 'Type any key to continue.' )
           Command = GetChar(InFile)
         return
         end        ! End of Delete Help

         subroutine EHelp
c        Erase help message.
%include 'DGCalc.global'
         character*1 Command, GetChar, UpperCase
         call CLS (OutFile)    ! Clear screen
         call pos (0, 1,OutFile)
         print 50
50       format ('  The Erase command can erase the whole sheet, a window, a row, a column, or'/'
     &  an entry, while retaining the window, row, column, or entry as a blank cell'/'
     &  or cells.'/'
     &  After the command is typed, the following appears :'//'
     &           A)ll  W)indow  R)ow  C)olumn  E)ntry'//'
     &  Type A to erase the whole sheet, W to erase a window, R to erase the current'/'
     &  row, C to erase the current column, and E to erase the entry.'//'
     &  When erasing the whole sheet, the question appears : Are you sure ?'/'
     &  Type Y to confirm this command or type N to abort.'/'
     &  Erasing rows or columns leaves the erased locations blank, whereas deleting'/'
     &  removes the contents and moves the following rows or columns to the deleted'/'
     &  locations. When erasing a window, the top left corner and bottom right corner'/'
     &  must be specified. When erasing an entry, the current cell content is erased.'//'
     &  For more information, please type :'//'
```

```
     &            A)ll  W)indow  R)ow  C)olumn  E)ntry* )
        call RetMess
        Command = ' '
        do while ((Command .ne. 'A') .and. (Command .ne. 'W')
     &      .and. (Command .ne. 'R') .and. (Command .ne. 'C')
     &      .and. (Command .ne. 'E') .and.
     &            (Command .ne. Char(27)) )
          Command = GetChar ( Infile )
          Command = UpperCase (Command)
          end do
        if ( Command .ne. Char (27) ) then
                if ( Command .eq. 'A' ) then
                        call AEHelp ( OutFile )
                else if ( Command .eq. 'W' ) then
                        call WEHelp ( OutFile )
                else if ( Command .eq. 'R' ) then
                        call REHelp ( OutFile )
                else if ( Command .eq. 'C' ) then
                        call CEHelp ( OutFile )
                else if ( Command .eq. 'E' ) then
                        call EEHelp ( OutFile )
                end if
           call PrintMess ( 'Type any key to continue.' )
           Command = GetChar( InFile )
        end if
        return
        end   ! End of Erase Help

        subroutine AEHelp ( Screen )
c       Explain Erase All command. KBD is Key BoarD.
        integer Screen
        call CLS (Screen )    ! Clear screen
        call pos (1,1,Screen)
        print 50
50      format (' This command will erase the whole working sheet '//'
     &  You need to type Y to confirm this command or N to abort.'/ )
        return
        end    ! End of All Erase Help

        subroutine WEHelp ( Screen )
c       Explain Erase Window command
        integer Screen
        call CLS (Screen)    ! Clear screen
        call pos (1,1,Screen)
        print 50
50      format ('   This command will erase a portion of the sheet.'//'
     &  The window to be erased is specified by its Top Left Corner and Bottom Right'/'
     &  Corner. The size of this window can be as large as the whole sheet or can be'/'
     &  as small as one cell. It will print an error message and erase nothing if'/'
     &  invalid window locations are typed.'/ )
        return
        end    ! End of Window Erase Help

        subroutine REHelp ( Screen )
c       Explain Erase Row command
        integer Screen
        call CLS (Screen)    ! Clear screen
        call pos (1,1,Screen)
        print 50
50      format (' This command will erase the current row in the sheet.'//'
     & The current row is the row where the cursor is located.'//'
     & The difference between Delete Row and Erase Row is : '//'
     & When Deleting, no blank space is left in the deleted area.'/'
```

```
      & When Erasing, the erased area is left blank.*// )
         return
         end    ! End of Row Erase Help

         subroutine CEHelp ( Screen )
c        Explain Erase Column command
         integer Screen
         call CLS (Screen)    ! Clear screen
         call pos (1,1,Screen)
         print 50
50       format (* This command will erase the current column in the sheet.*/*
      & The current column is the column where the cursor is located.*//*
      & The difference between Delete Column and Erase Column is : *//*
      & When Deleting, no blank space is left in the deleted area.*/*
      & When Erasing, the erased area is left blank.* )
         return
         end    ! End of Column Erase Help

         subroutine EEHelp ( Screen )
c        Explain Erase Entry command
         integer Screen
         call CLS (Screen)    ! Clear screen
         call pos (1,1,Screen)
         print 50
50       format (* This command will only erase the entry of the sheet.*//*
      & The entry is the cell where the cursor is currently located.*/)
         return
         end    ! End of Entry Erase Help

         subroutine FHelp
c        It explains Form command
      %include 'DGCalc.global'
         character *1 Command, GetChar, UpperCase
         call CLS (OutFile)    ! Clear screen
         call pos (1,1,OutFile)
         print 50
50       format( *  The Form command changes the form of the entry, i.e., the precision of*/*
     &   real numbers or the width of a column or columns.*/*
     &   After entering this command, the following appears :*//*
     &           P)recision            W)idth*//*
     &   type P to change the precision of the whole sheet, a row, a column, or only*/*
     &   one cell by selecting either  A)ll, R)ow, C)olumn, or E)ntry, respectively.*//*
     &   Type W to change the width of all columns or of the current column.*//*
     &   For more information please type :*//*
     &              P)recision              W)idth* )
         call RetMess
         Command = * *
         do while ((Command .ne. *P*) .and. (Command .ne. *W*)
     &      .and. (Command .ne. char (27) ) )
                Command = GetChar ( InFile )
                Command = UpperCase ( Command )
         end do
         if (Command .ne. char (27) ) then
           if  ( Command .eq. 'P' ) then
                call PForm
           else
                call WForm
           end if
           call PrintMess ( 'Type any key to continue.' )
                Command = GetChar ( InFile )
         end if
         return
         end     ! End of Form Help
```

```
      subroutine PForm
C   It explains how to change the Precision.
%include 'DGCALC.GLOBAL'
      call CLS (OutFile)
      call Pos (1,1,OutFile)
      print  50
50    format ('  The precision, or the number of figures following the decimal point in a'/'
     &  real number, can be changed at this level.'//'
     &  The number of digits after the decimal point is specified by typing a number'/'
     &  from 0 to 9.'//'
     &  After the number is chosen, type A)ll to change the precision for the whole'/'
     &  sheet, R)ow for the current row, C)olumn for the current column, or E)ntry'/'
     &  for the current cell.' )
      return
      end       ! End of Precision Form

      subroutine WForm
c     It explains how to change the width of the spreadsheet.
%include 'DGCALC.GLOBAL'
      call CLS (OutFile)
      call Pos (1,1,OutFile)
      print  50
50    format ( '  The width of any column can be selected from 3 to 75 characters by typing'/'
     &  an integer number. After the number is typed, the following appears :'//'
     &             A)ll   or   C)olumn'//'
     &  Type A to change the width of all columns and C to change the width of'/'
     &  the current column.'//'
     &  If the number is less than 3 or greater than 75, an error message will be'/'
     &  printed and no changes will be made.' )
      return
      end       ! End of Width Form

      subroutine GHelp
c     This subroutine prints information about Goto command
%include 'DGCalc.global'
      character*1 Command, GetChar
      call CLS (OutFile)    ! Clear screen
      call pos (1,1,OutFile)
      print 50
50    format ('  The Goto command moves the cursor to any cell location in the spreadsheet.'/'
     &  The location is determined by the column and row labels. The column label is'/'
     &  from A,B,C,...AA,AB... to CV. The row label is from 1,2,3,... to 255.'/'
     &  The syntax of the response needed for the Goto Command is : '//'
     &             Cell Name : [ Destination Cell Label ] '//'
     &  For example, to move the cursor to the position with the column label AB and'/'
     &  the row label 50, type :'//'
     &        Cell name : [AB50]    or   [ab50] ')
      call PrintMess ( 'Type any key to continue.' )
        Command = GetChar ( Infile )
      return
      end       ! End of Goto Help

      subroutine HHelp
c     This subroutin prints information about Help command
%include 'DGCalc.global'
      character*1 Command, GetChar
      call CLS (outfile)    ! Clear screen
      call pos (1,1,OutFile)
      print 50
50    format ('  The following procedure can be used to obtain help messages for all'/'
     &  commands in DGCALC :'//'
```

```
     &  When in Help mode, read the menu on the monitor and type one desired command.'/'
     &  The monitor will print out the corresponding messages about that command.'/'
     &  When the monitor prints :'//'
     &               Type one command for more information'//'
     &                        [Command LIST]'//'
     &  Type the first letter of the desired command.'//'
     &  Type (ESC) to resume the spreadsheet in any selection in help mode.')
        call PrintMess ('Type any key to continue.')
          Command = GetChar ( Infile )
        return
        end         ! End of Help Help

        subroutine IHelp
c       This subroutine explains Insert command
%include 'DGCalc.global'
        character*1 Command, GetChar
        call CLS (OutFile)    ! Clear screen
        call pos (1,1,OutFile)
        print 50
50      format ('  The Insert command inserts blank columns or rows into the sheet.'//'
     &  After entering this command, the following appears :'//'
     &            R)ow    or    C)olumn'//'
     &  Type R to insert rows above the current row.'/'
     &  Type C to insert columns to the left of the current column.'//'
     &  The number of rows or columns can be chosen by answering the following question'/'
     &  with the desired number of rows or columns :'//'
     &          How many rows ( 0-9 ) ? or'/'
     &          How many columns ( 0-9 ) ?'//'
     &  Typing 0 inserts no rows or columns and is the same as aborting this selection.')
        call PrintMess ( 'Type any key to continue.' )
          Command = GetChar ( Infile )
        return
        end         ! End of Insert Help

        subroutine JHelp
c       This subroutine explains Join command
c       Join commanc loads the second file and puts it to any location
c       in the working sheet. The location is specified by the Top Left
c       corner of the second file.
%include 'DGCalc.global'
        character*1 Command, GetChar
        call CLS (outfile)    ! Clear screen
        call pos (1,1,OutFile)
        print 50
50      format ('  The Join command will load a DGCALC file and put it into any location in the'/'
     &  working sheet. The location is specified by its top left corner.'//'
     &  The DGCALC file is a file previously saved by using DGCALC.'//'
     &  The differences between the Join and Load commands are :'//'
     &  The Load command erases the whole sheet, loads a new DGCALC file, and places it'/'
     &  at the start location A1.'//'
     &  The Join command does not erase the sheet, but loads a DGCALC file and places'/'
     &  it at any start location. The DGCALC file can be another working sheet which'/'
     &  was saved by DGCALC previously.'//'
     &  With the Save Window command, DGCALC can split a working sheet into several'/'
     &  sheets.'//'
     &  With the Join command, DGCALC can combine several sheets to form a new sheet.')
        call PrintMess ( 'Type any key to continue.' )
          Command = GetChar ( Infile )
        return
        end         ! End of Join Help

       subroutine LHelp
c       This subroutine explains Load command.
```

```
c       Load reads an input file into the spreadsheet.
%include 'DGCalc.global'
        integer RESULT
        character*1 Command, GetChar, UpperCase
        call CLS (OutFile)    ! Clear screen
        call pos (1,1,OutFile)
        print 50
50      format (' The Load command erases the working sheet, reads a new DGCALC file, and places'/'
     &  it into the start location of the spreadsheet.'//'
     &  The user needs to type an input file name, then type return.'/'
     &  The syntax is :'//'
     &                  File to read from : [PATHNAME]:[FILENAME]'//'
     &  A pathname may be required before the file name if the file is in a'/'
     &  different directory.'//'
     &  If there is no such file name, the message appears : The file does not exist !'/'
     &  If an invalid file name is typed, it prints : File is not a DGCALC file !'/'
     &  In both of the above situations, the user is returned to the spreadsheet.'//'
     &          For more information, please type :'//'
     &                   P)athname            F)ilename'//'
     &  Currently, the file name cannot be longer than 20 characters.')
        call RetMess
        Command = 'k'
        do while ((Command .ne. 'P') .and. (Command .ne. 'F') .and.
     &           (Command .ne. Char (27)))
                  Command = GetChar (InFile)
                  Command = UpperCase (Command)
                 end do

        if (Command .ne. Char(27)) then
            call CLS ( Outfile )
            if (Command .eq. 'P') then
                call CLI ( 'Help *Pathnames', RESULT )
            else if (Command .eq. 'F') then
                write (OutFile) '<nl> <nl> <nl>'
                call CLI ( 'Help *Filenames', RESULT )
            end if
                call PrintMess ( 'Type any key to continue.' )
                   Command = GetChar ( Infile )
        end if
        return
        end    ! End of Load Help

        subroutine MHelp
c       It explains Move Command
%include 'DGCalc.global'
        character*1, Command, GetChar
        call CLS ( OutFile )
        call Pos ( 1, 1, OutFile )
        print 50
50      format ('  The Move command moves one window to another window location.'//'
     &  The source window is specified by its Top Left Corner and Bottom Right Corner'/'
     &  and the target window by its Top Left Corner.'//'
     &  Be sure there is no overlap between the source and the target window. The'/'
     &  window boundary cannot exceed the limits of DGCALC (100 rows and 255 columns).'//'
     &  To move a window with a Top left corner B3 and a Bottom right corner of C10 to'/'
     &  the target window with a Top left corner D20, the syntax is :'//'
     &          Top left corner : [B3]'/'
     &          Bottom right corner : [C10]'//'
     &          Top left target corner : [D20]')
        call PrintMess ( 'Type any key to continue.' )
             Command = GetChar( Infile )
        return
        end   ! End of Move Help
```

```
      subroutine PHelp
c     This subroutine explains Print command.
%include 'DGCalc.global'
      character*1 Command, GetChar, UpperCase
      call CLS (OutFile)    ! Clear screen
      call pos (1,1,OutFile)
      print 50
50    format (' The Print (data) Save option stores spreadsheet information in a data filee that'/'
     &  can be used as any other text(ASCII) file on the Data General computer.'/'
     &  The user can print this file with the GPRINT command at the CLI level.'//'
     &  After the command is typed, the following appears :'//'
     &            S)heet    W)indow    or    D)ump file'//'
     &  The Print(data) option saves the contents of the spreadsheet that can be viewed'/'
     &  on the screen. If the Print file option is selected, the user can indicate'/'
     &  whether to save the whole sheet or a window by typing either S or W.'//'
     &  If Window is selected, the user needs to determine the window location by'/'
     &  its tope left and bottom right corners.'//'
     &  Note that the width of the spreadsheet must be small enough to print on the'/'
     &  computer paper. The maximum number of characters that can be printed on a'/'
     &  wide computer sheet is 132.'//'
     &  The Dump file command saves the whole spreadsheet contents and information'/'
     &  describing the location and data type of all cells containing some contents.'//'
     &  Type any key to continue or type (ESC) to resume DGCALC.')
      Command = GetChar (InFile)
      if (Command .eq. char(27) ) then
            return
      end if
      call CLS (OutFile)
      print  60
60    format (' A file name, no more than 20 characters, is needed for the output file.'//'
     &  The saved print(data) file may have up to 3 titles to describe the saved'/'
     &  information. Each title can have up to 100 characters.'//'
     &  For more information, please type :'//'
     &            S)heet      W)indow     or    D)ump')
      call RetMess
      Command = ' '
      do while ((Command.ne.'S').and.(Command.ne.'D')
     &     .and.(Command.ne.'W').and.(Command.ne.char(27)))
         Command = GetChar(InFile)
         Command = UpperCase(Command)
      end do
      if (Command.ne.char(27)) then
         call CLS (OutFile)
         call Pos (0,1,OutFile)
         if (Command .eq. 'S') then
              call SPHelp       ! Sheet Print Help
         else if (Command .eq. 'W') then
              call WPHelp       ! Window Print Help
         else
              call DPHelp       ! Dump Print Help
         end if
         call PrintMess ('Type any key to continue.')
         Command = GetChar(InFile)
      end if
      return
      end         ! Print Help

      subroutine SPHelp
c     Sheet Print Help Message
      print 50
50    format (' S option saves the screen contents of the whole spreadsheet to a printable'/'
     &  file which can be printed with GPRINT command.'//'
```

```
     &  The user can type up to 3 titles or type RETURN for less or no titles.*//*
     &  Each title can have up to 100 characters.*///*
     &  Note that the width of the spreadsheet printable file must be small to print*//*
     &  on the computer printer. The maximum number of characters that can be printed*//*
     &  on a wide computer printer is 132.*)
        return
        end ! Sheet Print Help

        subroutine WPHelp
c       Window Print Help Message
        print 50
50      format(* W option saves the screen contents of a window to a printable file.*///*
     &  The window is specified by its Top left and Bottom right corners.*//*
     &  The user can type up to 3 titles or type RETURN for less or no titles.*//*
     &  Each title can have up to 100 characters.*///*
     &  The output is the same as it is on the monitor. The user needs to use the*//*
     &  QPRINT command to get the output from the printer.* )
        return
        end   ! End of Window Print Help

        subroutine DPHelp
c       Dump file Print Help
        print 50
50      format ( * This command Dumps the information for the whole sheet into a printable file.*///*
     &  The user needs to type a dump file name or type RETURN for the default name*//*
     &  [PrintOut]. An extension [.DUMP] is added after the the file name automatically.*///*
     &  The dump file contains all information about every cell in the spreadsheet,*//*
     &  including cell location, cell type, and cell contents.*///*
     &  The file is printed in the order from A1 to A255, B1 to B255 and so on.*///*
     &  No blank cells can be saved. The user needs to use the QPRINT command to print*//*
     &  this file on the printer.*)
        return
        end   ! End of Dump file print help

        subroutine QHelp
c       It explains Quit command
%include '$DGCalc.global'
        character*1  Command, GetChar
        call CLS (outfile)    ! Clear screen
        call pos (1,1,OutFile)
        print 50
50      format (* Type Q to quit the program when in command mode.*///*
     &  This command returns the user to the CLI level without saving the contents of*//*
     &  the spreadsheet. Be sure to use the Save or Print commands to save part or*//*
     &  all of the spreadsheet to the desired format.* )
        call PrintMess ( *Type any key to continue.* )
          Command = GetChar ( Infile )
        return
        end      ! End of Quit command

        subroutine RHelp
c       It explains the Recalculate command
%include '$DGCalc.global'
        character*1 Command, GetChar
        call CLS (outfile)    ! Clear screen
        call pos (1,1,OutFile)
        print 50
50      format( *  The Recalculate command recalculates the results of formulas in the*//*
     &  spreadsheet.*///*
     &  The spreadsheet only displays the results calculated the first time a formula*//*
     &  is entered.*///*
     &  Because a cell with a formula is a function of the other cells addressed by the*//*
     &  formula and DGCALC does not recalculate the sheet automatically, the user needs*//*
```

```
     &  to Recalculate the spreadsheet when new numbers are entered into cells used in'/'
     &  formulas, otherwise the results may not not be correct.'//'
     &  Thus, remember to use this command after updating the values of cells used in'/'
     &  formulas." )
            call PrintMess ( 'Type any key to continue.')
            Command = GetChar ( Infile )
          return
          end     ! End of Recalculate command

          subroutine SHelp
c         It explains Save command and its options.
          implicit None
%include 'DGCalc.global'
          character*1 Command, GetChar, UpperCase
          call CLS (OutFile)    ! Clear screen
          call pos (0,1,OutFile)
          print 50
50        format( ' The Save command saves the information of the working sheet to a disk file.'/'
     &  After the command is typed, the following appears:'//'
     &         D)GCALC   P)rint (data),   or   T)ernary file'//'
     &  Type D to save to a DGCALC formatted file, which can be retrieved with Load'/'
     &  or Join commands, P to save to a file which can be printed with the Data'/'
     &  General QPRINT command, and T to save numerical data to a special file for'/'
     &  the program TERNARY.'//'
     &  A file name, no more than 20 characters, is needed after typing the all'/'
     &  options.'//'
     &  For more information, please type :'//'
     &         D)GCALC     P)rint (data)    or    T)ernary' )
          call RetMess
          Command = 'k'
          do while ((Command .ne. 'D') .and. (Command .ne. 'P') .and.
     &    (Command.ne.'T').and.(Command.ne.Char(27)))
                   Command = GetChar ( InFile )
                   Command = UpperCase ( Command )
                  end do
          if (Command .ne. Char(27) ) then
              call CLS ( Outfile )
              if (Command .eq. 'D') then
                  call DGSaveHelp
              else if (Command .eq. 'P') then
                  call PHelp
              else
                  call TSHelp  ! Ternary Save Help
              end if
              if (Command.eq.'T') then
                  call PrintMess ( 'Type any key to continue.' )
                  Command = GetChar(InFile)
              end if
          end if
          return
          end   ! Save Help Message

          subroutine DGSaveHelp
          implicit None
%include 'DGCALC.Global'
          character*1 Ch, GetChar, UpperCase
          call Pos ( 1, 0, OutFile )
          print 50
50        format(' The DGCALC Save option will save to a file which has the DGCALC format.'/'
     &  This file can be retrieved with the Load or Join commands in DGCALC.'//'
     &  After this option is chosen, the following appears:'//'
     &               S)heet      or     W)indow'//'
     &  Type S to save the whole sheet and W to save a window to a DGCALC file.'/'
```

```
     &  A file name is needed after this command is chosen. The extension DG is added*//*
     &  to the file name automatically. If saving a window, the window location must*//*
     &  de specified by its top left and bottom right corners.*///*
     &  For more information, please type :*///*
     &                S)heet     or     W)indow* )
        call RetMess
        Ch = ' '
        do while ((Ch.ne.'S').and.(Ch.ne.'W').and.(Ch.ne.char(27)))
              Ch = GetChar(InFile)
              Ch = UpperCase(Ch)
        end do
        if (Ch.ne.char(27)) then
           call CLS (OutFile)
           if (Ch.eq.'S') then
                call SSHelp    ! Sheet Save Help
           else if (Ch.eq.'W') then
                call WSHelp    ! Window Save Help
           end if
           call PrintMess ('Type any key to continue.')
           Ch = GetChar(InFile)
        end if
        return
        end

        subroutine SSHelp
c       Sheet Save Help message
        print 50
50      format ( ' The whole sheet can be saved with the Sheet DGCALC Save command.*///*
     &  The saved sheet can be retrieved with the Load or Join commands.*///*
     &  No blank areas in the sheet can be saved.' )
        return
        end   ! End of Sheet Save Help

        subroutine WSHelp
c       Window Save Help message
        print 50
50      format ( ' The window can be saved as another spreadsheet with the Window DGCALC Save command.*///*
     &  The window is specifid by its top left corner and bottom right corner.*///*
     &  If an invalid window location is chosen, an error message will appear and*//*
     &  nothing will be saved.*///*
     &  The following are examples of invalid window location:*/*
     &      The column label is out of the range A to CW (1 to 100).*/*
     &      The row label is out of the range 1 to 255.*/*
     &      The left column label is greater than the right column label.*/*
     &      The top row label is greater then the bottom row label.')
        return
        end   ! End of Window Save Help

        subroutine PrintHelp
c       Printable file save help message
        print 50
50      format ( ' The Printable command can save the information of the working sheet to*//*
     &  a printable file. The information can include the whole sheet or a portion of*//*
     &  the sheet, depends upon typing S or W in the following options.*///*
     &  After the command is typed, the following appears :*///*
     &            S)heet     or     W)indow*///*
     &  Type S to save the whole sheet and type W to save a window to a printable*//*
     &  file. A file name, up to 20 letters, needs be typed after S or W is typed.*///*
     &  Note : The printer can only print 132 characters in horizontal direction.')
        return
        end    ! End of Print Save Help

        subroutine TSHelp
```

```
c       Ternary Save Help message
        print 50
50      format ( ' The Ternary Save command saves numerical data for the program TERNARY on the'/'
     &  Data General computer which generates a ternary diagram using the saved data'//'
     &  file as input.'//'
     &  The window of data to be saved must be specified by its top left corner and'//'
     &  bottom right corner.'//'
     &  Only numerical data can be used as valid input for the TERNARY program.'/'
     &  The data window must be three columns wide and no more than 100 rows.'/'
     &  The TERNARY program can plot up to 100 points that are individually identified.'/'
     &  on a ternary diagram.'//'
     &  The program TERNARY is also available on an IBM PC or compatible microcomputer.'/'
     &  To save data for transfer to a microcomputer, use the Window option under the'/'
     &  Print Save command. Use the same type of data window size (three columns by'/'
     &  no more than 100 rows)." )
        return
        end   ! End of Ternary Save Message


        subroutine THelp
c       It explains Type command.
%include 'DGCalc.global'
        character*1 Command, GetChar, UpperCase
        call CLS (outfile)    ! Clear screen
        call pos (1, 1, outfile )
        print 60
60      format ('  The Type command is used to change a numerical value in a cell to numerical'/'
     &  characters (interpreted as text) or vice versa.'//'
     &  After the command is typed, the following appears :'/'
     &                  Cell Type: N)umeric     T)ext'/'
     &  Type N to change numerical characters to a real number that is right'/'
     &  justified in the cell on the monitor. '//'
     &  Type T to change a numerical value, integer or real, to numerical characters'/'
     &  that are left justified on the monitor.'//'
     &  Typing N if the cell is of numerical type or typing T if the cell is of text'/'
     &  type will not change anything.'//'
     &  A mathematical formula is another type of data used in DGCALC. It must be in'/'
     &  a special format, otherwise it does not work correctly.'//'
     &  For more information, type : '//'
     &               N)umeric       T)ext      F)ormula')
        call RetMess
        Command = 'k'
        do while ((Command .ne. 'N') .and. (Command .ne. 'T').and.
     &      (Command.ne.'F').and. (Command .ne. Char(27)))
            Command = GetChar ( InFile )
            Command = UpperCase ( Command )
        end do
        if ( Command .ne. Char(27) ) then
           call Cls ( OutFile )
           call Pos ( 1, 1, OutFile )
           if (Command .eq. 'N') then
                call NType    ! Explain Numeric Type
           else if (Command .eq. 'T') then
                call TType    ! Explain Text Type
           else if (Command .eq. 'F') then
                call FType    ! Explai Formula Type
           end if
           if ( Command .ne. 'F' ) then
                call PrintMess ( 'Type any key to continue.' )
                Command = GetChar ( InFile )
           end if
        end if
        return
```

```
          end      ! End of Type Help


          subroutine FType
    c     This subroutine explains FORMULA TYPE data used in the sheet.
    %include 'DGCalc.global'
          character*1 Command, GetChar, UpperCase
          print 50
    50    format ('  Mathematical formulas can be typed into the spreadsheet.'/'
         &  The formulas must be in a specific format as described below :'/'
         &    1. Any formula or expression must start with a [#] sign.'/'
         &       Any function must start with a [@] sign and all arguments must be'/'
         &       in parentheses.'/'
         &    2. The valid operators are :  '/'
         &                 +  : Addition                -  : Subtruction '/'
         &                 *  : Multiplication          /  : Division '/'
         &                 %  : Percentage              ** : Exponent '/'
         &               (  ) : Parentheses              <76> : Range of variable names'/'
         &    3. The valid built in functions in DGCALC are : '/'
         &             @SIN : Sine                   @COS : Cosine '/'
         &             @TAN : Tangent                @LOG : Logarithm   '/'
         &             @MIN : Return the smallest value of a range of cells.'/'
         &             @MAX : Return the largest value of a range of cells.'/'
         &             @SUM : The summation of a range of cells.'//'
         &      For more information, please type :'/'
         &             S)yntax examples     O)perator        F)unctions ')
          call RetMess
          Command = 'k'
          do while ((Command .ne. 'S') .and. (Command .ne. 'O') .and.
         &          (Command .ne. 'F') .and. (Command .ne.Char(27)))
                     Command = GetChar ( InFile )
                     Command = UpperCase ( Command )
                    end do
          call CLS ( Outfile )
          if (Command .ne. Char(27) ) then
            if (Command .eq. 'S') then
                  call SynFormula       ! Syntax Examples
            else if (Command .eq. 'O') then
                  call OpFormula        ! Operator message
            else if (Command .eq. 'F') then
                  call FunFormula       ! Function message
            end if
                  call PrintMess ( 'Type any key to continue.' )
                  Command = GetChar ( Infile )
          end if
          return
          end      ! End of Formula Type Explain


          subroutine SynFormula
    c     Examples for the syntax of math formulas.
          print  50
    50    format (' The following are syntax examples of math formulas :'//'
         &    1. Math expression without variable names : '/'
         &          #1+123*23-3**2/2'/'
         &    2. Math expression with variable names : '/'
         &          #A1+C3/4+G8*5    A1, C3, and G8 must be already defined.'/'
         &    3. Math expression with functions :'/'
         &          #@SUM(A1<76>A50)/@MIN(A1<76>A50)'/'
         &    4. Other math expressions :'/'
         &          #@MAX(A1<76>AK1)*((12+2)**2+3)    #123+456+789'//'
         &  Note that the contents of the cells addressed by the variable names must be'/'
         &  of numeric type. The variable names can be any cells in which numeric data'/'
```

```
     &  has previously been entered.'/'
     &  If the relative copy command is to be used, all of the cell labels used as'/'
     &  variable names are incremented one row at a time.' )
        return
        end      ! End of Syntax samples


        subroutine OpFormula
c       Explain mathematical operators.
        print  50
50      format ('  Only +, -, *, /, **, (  ), % are valid operators. The have the same meaning'/'
     &  and priority as standard in computer mathematics. Standard functions have the'/'
     &  highest priority and addition and subtraction have the lowest. The operators'/'
     &  which have the same priority are used from left to right. The <076> symbol is'/'
     &  used to indicates that a continuous range of cells is to be used in a row cr'/'
     &  a column for the calculation.'//'
     &  Thus, make sure that either the row or the column label is the same in the'/'
     &  variable names on both sides of the <076> symbol.' )
        return
        end      ! End of Operator Expanation


        subroutine FunFormula
c       Explain functions used in the spreadsheet.
        print 50
50      format (' There are several built in functions in the spreadsheet. They are :'//'
     &      @SIN : Sine                 @COS : Cosine '/'
     &      @TAN : Tangent              @LOG : Logarithm with base 10'/'
     &      @MIN : Return the smallest value of a range of cells.'/'
     &      @MAX : Return the largest value of a range of cells.'/'
     &      @SUM : The summation of a range of cells.'//'
     &  The first character of any function must be a @ sign !!!'/'
     &  The trigonometric functions have the same meaning as standard in computer'/'
     &  mathematics.'/'
     &  The unit for trigonometric functions must be in RADIUS.' )
        return
        end   ! End of Function Explanation.


        subroutine NType
c       This subroutine explains NUMERIC TYPE data used in the sheet.
        print 50
50      format ('  Valid numerical characters can be changed into real numbers that are right'/'
     &  justified in the cell on the monitor by using the Numeric selection.'//'
     &  Valid numerical data must be integers or real numbers.'/'
     &  Real numbers must be floating point. Exponents are invalid as real numbers.'/'
     &  A minus sign must be the first character of a negative number.'/'
     &  A plus sign is optional for a positive number.'/'
     &  At least one digit, either 0, a munus sign, or other digit from 1 to 9,'/'
     &  must be on the left of decimal point.'//'
     &             VALID  NUMBERS              INVALID NUMBERS'/'
     &          1234    ( Integer )              .1234'/'
     &          1234.   ( Real )               1.234E5'/'
     &         -0.123   ( Real )               -.123'/'
     &          0.1234  ( Real )               '/'
     &         12.34    ( Real )                  ')
        return
        end    ! End of Numeric Type Explaination

        subroutine TType
c       This subroutine explains TEXT TYPE data used in the sheet.
        print 60
60      format ('  The Text function is used to change numerical values (either integers or real'/'
```

```
     &  numbers), to their equivalant numerical characters (interpreted as text) that*/*
     &  left justified on the monitor. Only valid numerical numbers can be changed*/*
     &  with this command.' )
        return
        end    ! End of Text Type Explaination

        subroutine DollarHelp
c       It explains how to execute CLI commands while in the DGCALC.
%include 'DGCalc.global'
        integer RESULT
        character*1, Command, GetChar
        call CLS (outfile)    ! Clear screen
        call pos (1, 1, outfile )
        print 50
50      format (*  When the ($) symbol is typed, the operation is temporary changed from DGCALC to*/*
     &  the CLI level of the Data General.*///*
     &  The user can then type and execute any CLI commands as long as (ESC) is not*/*
     &  typed.*/*
     &  Typing (ESC) returns the operation to the DGCALC spreadsheet.*///*
     &  Note that you cannot sign off the computer by using BYE when in this temporary*/*
     &  escape mode, but must return to and quit DGCALC first.*///*
     &  For more information, please type M)ore or read the manual about CLI.*///*
     &          M)ore about CLI commands* )
        call RetMess
        do while ((Command .ne. 'M') .and. (Command .ne. 'm')
     &       .and.(Command .ne. Char ( 27 ) ) )
                Command = GetChar ( Infile )
                end do
        if ((Command .eq. 'M') .or. (Command .eq. 'm')) then
             call CLS ( outfile )
             call CLI ( 'Help', RESULT )
             call PrintMess ( 'Type any key to continue.* )
                Command = GetChar ( Infile )
          end if
          return
          end      ! End of Dollar Sign Help

          subroutine RetMess
c         Print carriage return message on the monitor.
%include 'DGCalc.global'
          call pos ( 22, 10, outfile )
          write (outfile) ('Type (ESC) to resume the spreadsheet.')
          return
          end      ! End of Print Return Message

          subroutine TypeAny
c         The program stops until any key is pressed.
          integer KBD    ! Key Board
          common Screen, KBD
          character*1, Ch, GetChar
          Ch = GetChar ( KBD )
          return
          end    ! End of Type Any Key
```

+8888888888

```
      Subroutine PrintTopScale ! Print top index
      implicit none
%include 'DGCalc.global'
      integer Width, Current, LefthalfCell, RighthalfCell,GetWide
      character*2 XCell, AConvert
      integer i
      Width = GetWide()
      Current = Left
      call Pos(0,0,OutFile)
      write (OutFile) Spaces(1:4)
      call Underline
      do i=0,Width-1
         if (cellwidth(Current+i).gt.2) then
            lefthalfcell=(cellwidth(Current+i)-3)/2+1
            righthalfcell=(cellwidth(Current+i)-2)/2
         else
            lefthalfcell=0
            righthalfcell=0
         end if
         XCell = AConvert(Current+i)
         write(OutFile) Spaces(1:LefthalfCell)
         write(OutFile) XCell
         write(OutFile) Spaces(1:RighthalfCell)
      end do
      call ClrEOL
      call Normal
      return
      end     ! End of PrintTopScale

      Subroutine ClrEOL  ! Clear the rest of line on monitor.
      implicit none
      common /Files/ OutFile
      integer OutFile
      write (OutFile) '<013>'
      return
      end

      Subroutine Underline
      implicit none
      common /Files/ OutFile
      integer OutFile
      write (OutFile) '<024>'
      return
      end

      Subroutine Normal
      implicit none
      common /Files/ OutFile
      integer OutFile
      write (OutFile) '<035><025><036><105><017><022>'
      return
      end

      Subroutine Inverse
      implicit none
      common /Files/ OutFile
      integer OutFile
      write (OutFile) '<036><104>'
      return
      end

      character*2 function AConvert(index)
      implicit none
```

```
          integer index, temp
          Index = Index-1
          temp = index/26
          if (temp.lt.1) then
             AConvert(1:1)=' '
          else
             AConvert(1:1)=char(ichar('A')-1+temp)
          end if
          AConvert(2:2)=char(ichar('A')+mod(index,26))
          return
          end

          Subroutine PrintLeft(number)  ! Print left index on sheet
          implicit none
          integer number
          write (*,10) number
       10 format(i4,'| ')
          return
          end

          Subroutine PrintMessArea(Message)
      c  Print message on the monitor.
          implicit none
          integer OutFile
          common /Lines/ Spaces, Stars, Dashes
          common /Files/ OutFile
          character*(80) Dashes, Stars, Spaces
          character*(*) Message
          call Normal
          call Pos(21,0,OutFile)
          write (OutFile) Message,'==<076> '
           call pos (22, 60, outfile )
           write (outfile) '(ESC) for MENU.'
           call ClrEol       ! Clear the rest of line
      c    call Underline
      c    write (OutFile) Spaces(1:68)
      c    call Normal
          call ClrEOL
          return
          end     ! End of Print MessArea

          logical function LegalName(Cell_name)
      c  Check if input character is in ['A'..'Z'] or in ['a'..'z'].
      c  Return false if not.
          implicit none
          character*1 Cell_name
          integer ascii
          ascii = ichar(Cell_name)
          if ((ascii.ge.65).and.(ascii.le.90)) then
              legalname = .true.
          else if ((ascii.ge.97).and.(ascii.le.122)) then
              legalname = .true.
          else
              legalname = .false.
          end if
          return
          end   ! End of LegalName

          logical function LegalNumber(Cell_number)
      c  Check if the input character is in ['0'..'9'].
      c  Return false if not.
          implicit none
          character*1 Cell_number
```

```
          integer ascii
          ascii = ichar(Cell_number)
          if ((ascii.ge.48).and.(ascii.le.57)) then
              legalnumber = .true.
          else
              legalnumber = .false.
          end if
          return
          end    ! End of Legal Number

          Suoroutine ValidCell (Cellname, X, Y, Valid)
c         Return cell location X and Y, Valid = true if CellName is valid.
%include 'DGCalc.global'
          character*6 Cellname
          character*1 uppercase
          Integer X, Y, index
          logical Valid, legalname, legalnumber
          Index = 1
          X = 0
          Y = 0
          Do while  (legalname(Cellname(index:index)).and.(index.lt.7))
              X = X*26 + ichar(uppercase(Cellname(index:index))) - 64
              index = index +1
          end do
          Do while (legalnumber(Cellname(index:index)).and.(index.lt.7))
              Y = Y*10 + ichar(Cellname(index:index)) - ichar('0')
              index = index + 1
          end do
          if ((X.gt.0).and.(X.le.HSheetSize).and.(Y.gt.0).and.
     &        (Y.le.VSheetSize)) then
              Valid= .true.
           else if (CellName.eq.' ') then
              valid=.true.
              X=CurrX
              Y=CurrY
            else
              Valid= .false.
          end if
          return
          end   ! End of Valid Cell

           subroutine SkipSet (Cell,Length,Noskip,Chset,index)
           implicit none
           integer Length,index,noskip
           character*(*) Cell
           character*(7) chset
           noskip=index
           if (chset.eq.'Numaric') then
              do while ((index.le.Length).and.(Cell(index:index).ge.'0')
     &                  .and.(Cell(index:index).le.'9'))
                 index=index+1
              end do
           else if (chset.eq.'Signs  ' )then
              do while ((index.le.Length).and.((Cell(index:index).eq.'+')
     &                  .or.(Cell(index:index).eq.'-')))
                 index=index+1
              end do
           else
              do while ((index.le.Length).and.(Cell(index:index).eq.' '))
                 index=index+1
              end do
           end if
                 noskip=index-noskip
```

```
            return
            end

            subroutine PrintMess (Message)
      c     Print message on the screen
            implicit none
            character*(*) Message
            integer OutFile
            common /Files/ OutFile
            call Normal
            call Pos(22,0,OutFile)
            call ClrEOL
            call Pos(22,15,OutFile)
            write (OutFile) Message
            return
            end   ! Print mesage

            subroutine ErrorMess (Message)
      c     Print error message and beep
            implicit none
            character*(*) Message
            integer OutFile
            common /Files/ OutFile
            call Normal
            call Pos(22,0,OutFile)
            call ClrEOL
            call Pos(22,15,OutFile)
            write (OutFile) Message
            call Pos(21,14,OutFile)
            call ClrEOL
            print*,'<007>'
            return
            end   ! End of Error Message

            subroutine WriteIt (Data,Max,Width,Just,Buff)
            implicit none
      %include '/DGCalc.global'
            integer i, Max, w2, whitespace,Width
            character*(*) Data
            character*(5) Just
            logical Buff
            if (buff) then
               if ((ScreenPtr+Max).gt.1920) call FlushBuffer
               if (Max.ge.Width) then
                    do i=ScreenPtr, ScreenPtr+Width-1
                       ScreenBuffer(i:i)=Data(i-ScreenPtr+1:i-ScreenPtr+1)
                    end do
                    ScreenPtr=ScreenPtr+Width
               else if (Just.eq.'Ljust') then
                    do i=ScreenPtr, ScreenPtr+Max-1
                       ScreenBuffer(i:i)=Data(i-ScreenPtr+1:i-ScreenPtr+1)
                    end do
                    do i=ScreenPtr+Max, ScreenPtr+Width-1
                       ScreenBuffer(i:i)=' '
                    end do
                    ScreenPtr=ScreenPtr+Width
               else if (Just.eq.'Rjust') then
                    do i=ScreenPtr, ScreenPtr+Width-Max-1
                       ScreenBuffer(i:i)=' '
                    end do
                    do i=ScreenPtr+Width-Max, ScreenPtr+Width-1
                       ScreenBuffer(i:i)=Data(i-(ScreenPtr+Width-Max)+1:
     &                                 i-(ScreenPtr+Width-Max)+1)
```

```
                end do
                ScreenPtr=ScreenPtr+Width
            else if (Just.eq.'Cjust') then
                Whitespace=Width-Max
                w2=Whitespace/2
                do i=ScreenPtr, ScreenPtr+w2-1
                   ScreenBuffer(i:i)=' '
                end do
                ScreenPtr=ScreenPtr+w2
                do i=ScreenPtr, ScreenPtr+Max-1
                   ScreenBuffer(i:i)=Data(i-ScreenPtr+1:i-ScreenPtr+1)
                end do
                ScreenPtr=ScreenPtr+Max
                Whitespace=Whitespace-w2
                do i=ScreenPtr, ScreenPtr+Whitespace-1
                   ScreenBuffer(i:i)=Data(i-ScreenPtr+1:i-ScreenPtr+1)
                end do
                ScreenPtr=ScreenPtr+Whitespace
            end if
        else
            if (Max.ge.Width) then
                write (OutFile) Data(1:Width)
            else if (Just.eq.'Ljust') then
                write (OutFile) Data(1:MAX),Spaces(1:Width-Max)
            else if (Just.eq.'Rjust') then
                write (OutFile) Spaces(1:Width-Max),Data(1:Max)
            else if (Just.eq.'Cjust') then
                whitespace=Width-Max
                w2=whitespace/2
                write (OutFile) Spaces(1:w2),Data(1:Max),Spaces(1:
     &                          whitespace-w2)
            end if
        end if
        return
        end    ! End of WriteIt

        subroutine WriteData(X,Y,Buffered)
        implicit none
%include 'DGCalc.global'
        character*1 ty,sig
        character*80 tempchars
        integer last,start,index,Length,X,Y,i
        logical buffered
%include 'DGCalc.celltype'
        start=CellPointer(X,Y)
        ty=FormulaArray(start)
        sig=FormulaArray(start+1)
        start=start+CellHeader
        last=start
        do while ((FormulaArray(last).ne.eos).and.(last.lt.ArraySize))
           last=last+1
        end do
        Length=last-start
        do i=1,length
           tempchars(i:i)=FormulaArray(start+i-1)
        end do
        if (ty.eq.truecell) then
           if (CellWidth(x).gt.3) then
                call WriteIt ('TRUE',4,CellWidth(x),'Ljust',buffered)
           else
                call WriteIt ('T',1,CellWidth(x),'Ljust',buffered)
           end if
        else if (ty.eq.falsecell) then
```

```
            if (CellWidth(x).gt.4) then
                 call WriteIt ('FALSE',5,CellWidth(x),'Ljust',buffered)
            else
                 call WriteIt ('F',1,CellWidth(x),'Ljust',buffered)
            end if
         else if ((ty.eq.intcell).OR.(ty.eq.realcell)) then
            call WriteIt (tempchars(1:length),Length,
     &           CellWidth(x),'Rjust',buffered)
         else if (ty.eq.textcell) then
            call WriteIt (tempchars(1:length),Length,
     &           CellWidth(x),'Ljust', buffered)
         else if (ty.eq.formulacell) then
            start=last-11
            index=0
            do while ((FormulaArray(start+index).ne.'.').and.
     &           (FormulaArray(start+index).ne.eos))
                 index=index+1
            end do
            index=index+ichar(sig)+1
            if (index.gt.11) index=11
            do i=1,index
               tempchars(i:i)=FormulaArray(start+i-1)
            end do
            call writeit(tempchars(1:index),index,
     &           cellwidth(x),'Rjust',buffered)
         else
            call WriteIt (wrong,length,CellWidth(x),'Ljust',buffered)
         end if
         return
         end    ! End of WriteData

         subroutine FlushBuffer
         implicit none
%include 'DGCalc.global'
         if (ScreenPtr.gt.1) then
            write (OutFile) ScreenBuffer(1:ScreenPtr-1)
         end if
         ScreenPtr=1
         return
         end

         subroutine WriteBuf (Data,Max)
         implicit none
%include 'DGCalc.global'
         character*(*) Data
         integer Max, i
         if ((ScreenPtr+Max).gt.1920) call FlushBuffer
         do i=ScreenPtr, ScreenPtr+Max-1
            ScreenBuffer(i:i)=Data(i-ScreenPtr+1:i-ScreenPtr+1)
         end do
         ScreenPtr=ScreenPtr+Max
         return
         end


         subroutine PosBuf(Row,Column)
         implicit none
         integer Row, Column
         call writeBuf (char(020k),1)
         call writeBuf (char(Column),1)
         call writeBuf (char(Row),1)
         return
         end
```

```
      subroutine InverseBuf
      call writeBuf ('<036><104>',2)
      return
      end

      subroutine NormalBuf
      call writeBuf ('<035><025><036><105><017><022>',6)
      return
      end

      subroutine RealtoAscii (ascii,number,errorflag)
c  Convert a real number into a 11 character ASC II string.
      implicit none
      logical errorflag
      integer index,i,j
      real number
      integer*4 divider
      character*11 ascii
      errorflag=.false.
      index=1
      if (number.lt.0) then
         ascii(index:index)='-'
         index=index+1
      end if
      number=abs(number)
      if (number.ge.1E10) then
         errorflag=.true.
         call errormess('The number is too big')
         ascii(index:11)='10000000000'
      else
         divider=1000000000
         do while ((number.lt.divider).and.(divider.gt.1))
              divider=divider/10
         end do
         do while (divider.ge.1)
              i=int(number/divider)
              number=number-i*divider
              ascii(index:index)=char(i+48)
              divider=divider/10
              index=index+1
         end do
         ascii(index:index)='.'
         index=index+1
         number=number*10.0**(11-index+1)
         divider=1*10**(11-index)
         do while(index.le.11)
              i=int(number/divider)
              number=number-i*divider
              ascii(index:index)=char(i+48)
              divider=divider/10
              index=index+1
         end do
      end if
      return
      end     ! End of RealToAscii

      character*6 function position(x,y)
      implicit none
      integer x,y,k,index,divider
      character*2 aconvert,chs
      k=1
```

```
      position='     '
      chs=aconvert(x)
      if (chs(k:k).eq.' ') then
         position(k:k)=chs(k+1:k+1)
         k=k+1
      else
         position(1:k+1)=chs
         k=k+2
      end if
      divider=100
      do while(y.lt.divider)
         divider=divider/10
      end do
      do while (divider.ge.1)
         index=Int(y/divider)
         y=y-index*divider
         position(k:k)=char(index+48)
         divider=divider/10
         k=k+1
      end do
      return
      end      ! End of position
```

PRINTED IN U.S.A.

+2222222222

```
C*****************************************************************
C*** Library Source Listing                                    ***
C*****************************************************************
C***            Name of Subroutine:                  Line no: ***
C***                                                         ***
C***   Subroutine AssignS(OutString,InString)          24    ***
C***   Subroutine Left(Source,Destination,Length)      56    ***
C***   Integer Function Find(Source, Pattern, Start)   88    ***
C***   Subroutine Concat(First,Second)                123    ***
C***   Subroutine Right(Source,Destination,Length)    155    ***
C***   Subroutine WString(Source,Fid)                 187    ***
C***   Subroutine NewLine(Fid)                        213    ***
C***   Subroutine Mid(Source,Destination,Position,Length) 223 ***
C***   Integer Function SLen(Source)                  256    ***
C***   Subroutine RString(Destination,iFid)           270    ***
C***   Subroutine Openf(Fchannel,Fname,Faccess,IErr)  312    ***
C***   Subroutine Pos(row,column,Fid)                 348    ***
C***   Subroutine GetChar(Input_char,Fid)             369    ***
C***   Subroutine CLS(Fid)                            383    ***
C***   Subroutine CLI(CommandString, Result)          396    ***
C***   Subroutine KLI(CommandString, Result)          438    ***
C***                                                         ***
C*****************************************************************
      Subroutine AssignS(OutString,InString)
C=================================================================
C=   Create a dynamic string from a fixed format string.  Usually =
C=   InString will be a constant.  Outstrings length is set by    =
C=   the calling program.  The last byte in Outstring is set to   =
C=   the length of the string.  Therefore AssignS will only be    =
C=   valid for strings of length less the 256.  AssignS is meant  =
C=   to be an initializing subroutine in order that the user can  =
C=   benefit from other subroutines in this library which mani-   =
C=   pulate dynamic strings.                                      =
C=================================================================
      Character*(*) InString, OutString
C
C     MDL is the Maximum String Length of the Destination string.
C
      Integer MDL
      MDL = len(Outstring)
C
C     icount is the length of the Source string.
C
      icount = Len(InString)
C
C     if the length of the source string is longer than the destination
C     string them only the first (MDL-1) characters are assigned to the
C     destination string.
C
      if (icount.gt.MDL-1) icount = MDL-1
      OutString(1:icount) = InString(1:icount)
      OutString(MDL:MDL) = char(icount)
      return
      end

      Subroutine Left(Source,Destination,Length)
C=================================================================
C=   Left return the (Length) of leftmost characters in the string =
C=   Source in the Destination string, both which are dynamic      =
C=   strings.                                                     =
C=================================================================
      Character*(*) Source, Destination
C
```

```
C   Length -- number of leftmost characters to transfer.
C   SourceLength -- dynamic length of source string.
C   MSL -- maximum length of source string.
C   MDL -- maximum length of destination string.
C
      Integer Length, SourceLength, MSL, MDL
      MDL = len(Destination)
C
C   if the destination cannot hold the requested number of
C   characters then no transfer takes place.
C
      if (MDL-1.lt.Length) return
      MSL = len(Source)
      SourceLength = ichar(Source(MSL:MSL))
C
C   if the source string is shorter than the requested number of
C   characters then no transfer takes place.
C
      if ((Length.lt.1).or.(Length.gt.SourceLength)) return
      Destination(1:Length) = Source(1:Length)
      Destination(MDL:MDL)=char(Length)
      return
      end

      Integer Function Find(Source, Pattern, Start)
C=====================================================================
C=  Find operates on dynamic strings.  Its function is to find a     =
C=  character Pattern in a Source string.  The Start variable       =
C=  is the index into the source at which the search should          =
C=  commence.  Pattern is NOT a dynamic string but usually a        =
C=  string constant.                                                =
C=====================================================================
      Character*(*) Source, Pattern
C
C   Length -- length of the source string.
C   Start  -- position in the source string to start the search.
C   MSL    -- maximum length of the source string.
C
      Integer Length, Start, MSL
      MSL = len(Source)
      Length = ichar(Source(MSL:MSL))
C
C   initially set the position found to zero
C
      Find = 0
C
C   if the Pattern would not fit within the length of the source
C   string then return a 0.
C
      if (Len(Pattern).gt.(Length-Start-1)) return
      do 10, icount = Start,(Length-Len(Pattern)+1)
         if (Source(icount:icount+Len(Pattern)-1).eq.Pattern) then
            Find = icount
            return
         endif
   10 continue
      return
      end

      Subroutine Concat(First,Second)
C=====================================================================
C=  Concatenates the First string and Second string together and     =
C=  place the result into the First string.  The strings, First     =
```

```
C=   and Second, must be dynamic strings.                          =
C====================================================================
      Character*(*) First, Second
C
C    Length1 -- dynamic length of first string.
C    Length2 -- dynamic length of second string.
C    M1L     -- maximum length of first string.
C    M2L     -- maximum length of second string.
C
      Integer Length1, Length2, M1L, M2L
      M1L = len(First)
      M2L = len(Second)
      Length1 = ichar(First(M1L:M1L))
      Length2 = ichar(Second(M2L:M2L))
C
C    if resultant string will not fit into the first string then
C    fill up the first string as far as it will go.
C
      if ((Length1+Length2).gt.M2L-1) Length2=M1L-1-Length1
C
C    if the Second string is empty then no need to process anything.
C
      if (Length2.lt.1) return
      First(Length1+1:Length1+Length2) = Second(1:Length2)
      First(M1L:M1L)=char(Length1+Length2)
      return
      end

      Subroutine Right(Source,Destination,Length)
C====================================================================
C=   Given two dynamic strings, Source and Destination, and a number =
C=   of characters to transfer; copy the rightmost (Length) chars    =
C=   from Source into Destination.                                   =
C====================================================================
      Character*(*) Source, Destination
C
C    Length -- number of rightmost chars to transfer from Source.
C    temp   -- dynamic size of Source.
C    MSL    -- maximum size of Source.
C    MDL    -- maximum size of Destination.
C
      Integer Length, temp, MSL, MDL
      MSL = len(Source)
      MDL = len(Destination)
      temp = ichar(Source(MSL:MSL))
C
C    if the requested number of chars won't fit into destination then
C    don't transfer any.
C
      if (MDL-1.lt.Length) return
C
C    if the requested number of chars isn't available in Source then
C    don't transfer any.
C
      if (temp.lt.Length) return
      Destination(1:Length) = Source(temp-Length+1:temp)
      Destination(MDL:MDL) = char(Length)
      return
      end

      Subroutine WString(Source,Fid)
C====================================================================
C=   Write out a dynamic string to an unformatted file opened for    =
```

```
C=   output.  Use the OPENF subroutine provided in this library to    =
C=   open the output file.                                           =
C====================================================================
C
C    Source -- dynamic string to print.
C
       Character*(*) Source
C
C    Fid  -- File identifier (channel number)
C    temp -- dynamic length of Source string.
C    MSL  -- maximum length of Source string.
C
       Integer Fid, temp, MSL
       MSL = len(Source)
C
C    if invalid File identifier than return
C
       if (Fid.lt.1) return
       temp = ichar(Source(MSL:MSL))
       write (Fid) Source(1:temp)
       return
       end

       Subroutine NewLine(Fid)
C====================================================================
C=   Send a carriage return-line feed to the device with channel #  =
C=   Fid.  Must be a DG-compatible terminal                         =
C====================================================================
       Integer Fid
       write (Fid) '<NL>'
       return
       end

       Subroutine Mid(Source,Destination,Position,Length)
C====================================================================
C=   Return in the Destination string, the contents of the Source   =
C=   string from position Position for Length number of characters. =
C====================================================================
       character*(*) Source, Destination
C
C    Position -- the char position within Source to begin the copy.
C    Length   -- the number of chars to transfer.
C    temp     -- char position of last char to transfer from Source.
C    MSL      -- maximum size of Source string.
C    MDL      -- maximum size of Destination string.
C
       integer Position, Length, temp, MSL, MDL
       MSL = len(Source)
       MDL = len(Destination)
C
C    if requested transfer length is longer than the Destination
C    length then don't transfer.
C
       if (MDL-1.lt.Length) return
C
C    if Position is not in the bounds of the Source string then
C    don't try transfer.
C
       If ((Position.lt.1).or.(Position.ge.MSL)) return
       temp = Position+Length-1
       If (temp.ge.ichar(Source(MSL:MSL))) return
       Destination(1:Length) = Source(Position:temp)
       Destination(MDL:MDL) = char(Length)
```

```
            return
            end

            Integer Function SLen(Source)
      C=====================================================================
      C=   Return the integer dynamic length of the Source string.        =
      C=====================================================================
            character*(*) Source
      C
      C     MSL -- maximum length of Source string.
      C
            integer MSL
            MSL = len(Source)
            SLen = ichar(Source(MSL:MSL))
            return
            end

            Subroutine RString(Destination,iFid)
      C=====================================================================
      C=   Read a string from an input file, echo it to an output file and  =
      C=   return it in a dynamic Destination string.                      =
      C=====================================================================
      C
      C     Destination -- input string.
      C
            Character*(*) Destination
      C
      C     InChar -- current input character read from file IFid.
      C
            Character*1 InChar
      C
      C     iFid   -- File identifier(channel) from which a char is to be read.
      C     temp   -- ASCII value of current input char.
      C     MDL    -- maximum length of Destination string.
      C     index  -- current position for char to placed in Destination string
      C
            Integer iFid, temp, MDL, index
            MDL = len(Destination)
            do 10 index = 1,MDL-1
            read (iFid) InChar(1:1)
            temp=ichar(InChar)
      C
      C     if InChar is an <cr> or <nl> then string has been read.
      C
            if ((temp.eq.10).or.(temp.eq.13)) goto 20
      C
      C     update Destination with the new input char.
      C
            Destination(index:index)=InChar(1:1)
         10 continue
         20 index = index-1
      C
      C     update Destination dynamic string length.
      C
            Destination(MDL:MDL)=char(index)
            return
            end


            Subroutine Openf(Fchannel,Fname,Faccess,IErr)
      C=====================================================================
      C=   Open a file for dynamic access.  Files opened by this sub-      =
      C=   routine are used by a variety of other subprograms in this     =
```

```
C=   library.                                                          =
C=======================================================================
C
C     Fname -- File name to be opened.
C     Faccess -- either an 'R' or 'W' for read and write respectively.
C
        Character*(*) Fname
        Character*1 Faccess
C
C     Fchannel -- identifier(channel) to open this file under.
C     IErr     -- resultant error on open(0 if normal).
C
        Integer Fchannel, IErr
C
C     restrict legal file channels to between 1 and 255.
C
        if ((Fchannel.gt.255).or.(Fchannel.lt.1)) then
           stop  'Channel # illegal on file open'
        endif
        if ((Faccess.eq.'r').or.(Faccess.eq.'R')) then
           open(Fchannel,recfm='dynamic',file=Fname,form='unformatted',
     +     mode='binary',Status='old',IOStat=IErr)
        else
           if ((Faccess.eq.'w').or.(Faccess.eq.'W')) then
           open (Fchannel,file=Fname,iointent='output',mode='binary',
     +       recfm='dynamic',form='unformatted',IOStat=IErr,
     +       status='Fresh' )
           endif
        endif
        return
        end


        Subroutine Pos(X,Y,Fid)
C=======================================================================
C=   Position Cursor to position X,Y on a DG-compatible terminal       =
C=   where X is the row and Y is the column.                           =
C=======================================================================
C
C     X   -- row position.
C     Y   -- column position.
C     Fid -- File identifier for output terminal.
C
        integer X,Y, Fid
C
C     if an attempt is made to locate the cursor off of the screen
C     then ignore the command and resume calling routine.
C
        if ((X.lt.0).or.(X.gt.23).or.(Y.lt.0).or.(Y.gt.79)) return
        Write (Fid) char(020K)//char(Y)//char(X)
        Return
        End


        Subroutine CLS(Fid)
C=======================================================================
C=   Clear the screen associated with the File identifier(FID).        =
C=   The FID must be a channel opened for output and unformatted.      =
C=   The best way to accomplish this is to use the OPENF subrou-       =
C=   tine found in this library.  The input device must be a DG        =
C=   compatible terminal.                                              =
C=======================================================================
        integer Fid
```

```
        write (Fid) char(014K)
        return
        end

        Subroutine CLI(CommandString, Result)
C=======================================================================
C=   The purpose of CLI is to execute a single CLI command followed   =
C=   by the return to the calling procedure.  CommandString is the    =
C=   CLI command to be executed and Result is the code returned by    =
C=   AOS/VS upon termination.                                         =
C=======================================================================
        Integer AddrOfProgramName
        Integer AddrOfString
        Integer AddrOfMessageHeader
        Integer AC0, AC1, AC2
        Integer ISYS
        Integer Result
        Integer*2 ProcPacket(0:31) /32*-1/
          Equivalence (AddrOfProgramName, ProcPacket(2))
          Equivalence (AddrOfMessageHeader, ProcPacket(4))
        Integer*2 ISendHeader(0:7) /8*0/
          Equivalence (AddrOfString, ISendHeader(6))
        Character*(*) CommandString
        Character*(256) TemporaryText
****    INTEGER*4 parameters for SYSID  ****

         INTEGER*4  ISYS_PROC
         PARAMETER (ISYS_PROC = 214)     ! ?.PROC = 326K

****    Parameters for PARU   ****

         INTEGER*2  ISYS_PFEX
         PARAMETER (ISYS_PFEX = 8192)    ! ?PFEX = 20000K

        TemporaryText = CommandString
        ISendHeader(5) = 128
        AddrOfString = WordAddr(TemporaryText)
        ProcPacket(0) = ISYS_PFEX
        AddrOfProgramName = ByteAddr(":CLI.PR<0>")
        AddrOfMessageHeader = WordAddr(ISendHeader)
        AC2 = WordAddr(ProcPacket)
        Result = ISYS(ISYS_PROC, AC0, AC1, AC2)
        Return
        End

        Subroutine KLI(CommandString, Result)
C=======================================================================
C=   The purpose of KLI is to execute :CLI.PR as a subprocess sending  =
C=   an initial message in the CommandString.  CommandString is the    =
C=   CLI command to be executed and Result is the code returned by     =
C=   AOS/VS upon termination.                                          =
C=======================================================================
        Integer AddrOfProgramName
        Integer AddrOfString
        Integer AddrOfMessageHeader
        Integer AC0, AC1, AC2
        Integer ISYS
        Integer Result
        Integer*2 ProcPacket(0:31) /32*-1/
          Equivalence (AddrOfProgramName, ProcPacket(2))
          Equivalence (AddrOfMessageHeader, ProcPacket(4))
        Integer*2 ISendHeader(0:7) /8*0/
```

```
          Equivalence (AddrOfString, ISendHeader(6))
        Character*(*) CommandString
        Character*(256) TemporaryText

****  F77 INCLUDE file for system parameters ****
****   INTEGER*4 parameters for SYSID  ****

         INTEGER*4  ISYS_PROC
         PARAMETER (ISYS_PROC = 214)     ! ?.PROC = 326K

****    Parameters for PARU   ****

         INTEGER*2  ISYS_PFEX
         PARAMETER (ISYS_PFEX = 8192)    ! ?PFEX = 20000K

****   TALOCK prevents process from terminating when placed
****   at offset ?IUFL (the user flag word) in an IPC Send
****   Header.

         INTEGER*2  ISYS_TALOCK
         PARAMETER (ISYS_TALOCK = 16384) ! ?TALOCK = 40000K

        TemporaryText = CommandString
        ISendHeader(1) = ISYS_TALOCK
        ISendHeader(5) = 128
        AddrOfString = WordAddr(TemporaryText)
        ProcPacket(0) = ISYS_PFEX
        AddrOfProgramName = ByteAddr(':CLI.PR<0>')
        AddrOfMessageHeader = WordAddr(ISendHeader)
        AC2 = WordAddr(ProcPacket)
        Result = ISYS(ISYS_PROC, AC0, AC1, AC2)
        Return
        End

        character*1 function UpperCase(LowerCase)
c  Convert low case letter into upper case one. No change otherwise.
        implicit none
        character*1 Lowercase
        if ((ichar(LowerCase).ge.97).and.(ichar(LowerCase).le.122)) then
            UpperCase = char(ichar(LowerCase) - 32)
        else
            UpperCase = lowercase
        end if
        return
        end   ! End of Upper Case
```

```
 $$$$   $$$$  $    $  $    $  $$$$$  $$$$$
$      $      $    $  $    $    $      $
$      $      $    $  $    $    $      $
$      $      $ $  $  $$$$$    $      $
$  $$  $  $$  $ $  $  $    $    $      $
$    $ $    $ $$ $$  $    $    $      $
 $$$$   $$$$  $    $  $    $  $$$$$    $

   DEST=GGWHIT  USER=GGWHIT  QUEUE=LPT  DEVICE=@LPB
SEQ=21180  QPRI=137  LPP=66  CPL=132  COPIES=1  LIMIT=11

     CREATED:  27-AUG-87  15:46:28
   ENQUEUED:  31-DEC-87  11:19:46
   PRINTING:  31-DEC-87  11:20:27

     PATH=:SI20:GGWHIT:$DGCALC:EDIT.F77

$$$$$  $$$$   $$$$$  $$$$$        $$$$$  $$$$$  $$$$$
$      $   $    $      $          $          $      $
$      $   $    $      $          $        $      $
$$$$   $   $    $      $          $$$$     $      $
$      $   $    $      $          $        $      $
$      $   $    $      $      $$  $        $      $
$$$$$  $$$$   $$$$$    $      $$  $        $      $
```

Kansas Geological Survey AOS/VS

AOS/VS REV 07.57
AOS/VS XLPT REV 07.56

```
      Subroutine Edit(fout,fin,line,CharsIn,length,int)
      implicit none
      common /ecomm/ esc, size,filein, fileout
      intrinsic len, ichar
      logical esc, int
      character*(*) line
      character*1 GetChar, ch
      integer length,x,y,place, size, filein, fileout, ich, fin, fout
      integer CharsIn
      CharsIn=0
      line = ' '
      filein = fin
      fileout = fout
      Call GetPos(fileout,filein,x,y)
      size = length
      int = .false.
      if (len(line).lt.size) size = len(line)
      place = 1
      ch = GetChar(filein)
      ich = ichar(ch)
      do while (ch.ne.char(012k))
       if ((ich.eq.8).or.(ich.eq.9).or.(ich.eq.22).or.(ich.eq.23).or.
     &     (ich.eq.24).or.(ich.eq.25).or.(ich.eq.26)) then
c       8 Home;  9 Tab;  22 ^V; 23 Up; 24 Down; 25 Left; 26 Right
         line(1:1) = ch
         int = .true.
         return
       else if (ich.eq.30) then
                ch = GetChar(FileIn)
                ich = ichar(ch)
                if ((ich.ge.23).and.(ich.le.26)) then ! Cursor Movement
                  line(1:1) = char (ich+26)
                  int = .true.
                  return
                else if ((ich.ge.92).and.(ich.le.95)) then
                  line(1:1) = ch
                  int = .true.
                  return
                else
                  write (FileOut) '<007>'
                end if
      end if
      Call GetPos(fileout,filein,x,y)
       x=x-place+1
         if (ich.eq.27) then
            int = .true.
            return
         else if (ich.lt.32) then
            Call CtrlChar(ch,line,place,y,x,charsIn)
            if ((ch.eq.char(10)).or.(ch.eq.char(27))) then
               int = esc
               return
            end if
         else if ((ich.lt.127).and.(place.le.size)) then
            write(fileout) ch
            line(place:place) = ch
            place = place+1
            CharsIn=CharsIn+1
         else if ((ich.eq.127).and.(place.gt.1))then
            call DelChar(line,place,y,x)
            if (CharsIn.ne.0) CharsIn=CharsIn-1
         else
            write(fileout) '<007>'
```

```
            endif
            ch = GetChar(filein)
            ich = ichar(ch)
         end do
         return
         end

         Subroutine CtrlChar(ch,line,place,y,x,length)
         implicit none
         common /ecomm/ esc, size, filein, fileout
         logical esc
         character*1 ch, GetChar
         character*(*) line
         integer place,filein,fileout, size, y, x,length
         if ((ch.eq.char(25)).and.(place.gt.1)) then
             write(fileout) ch
             place = place -1
         else if ((ch.eq.char(24)).and.(place.lt.size)) then
             write(fileout) ch
             place = place + 1
         else if (ch.eq.char(1)) then
             Call Pos(y,x+size-1,fileout)
             place = size
         else if (ch.eq.char(5)) then
             Call InsertChar(line,place,y,x,ch,length)
             if ((ch.eq.char(10)).or.(ch.eq.char(27))) return
         else if (ch.eq.char(8)) then
             Call Pos(y,x,fileout)
             place = 1
         else if (ch.eq.char(11)) then
             line(place:size)=' '
             write(fileout) line(place:size)
             Call Pos(y,x+place-1,fileout)
         else if (ch.eq.char(3)) then
             ch = GetChar(filein)
             if (ch.eq.char(2)) then
                stop
             else if (ch.eq.char(1)) then
                return
             end if
         else
            write(fileout) *<007>*
         end if
         return
         end

         Subroutine DelChar(line,place,y,x)
         implicit none
         common /ecomm/ esc, size, filein, fileout
         logical esc
         character*(*) line
         integer size, place, filein, fileout, y, x, index
         place = place-1
         line(place:size-1) = line(place+1:size)
         line(size:size) = ' '
         Call Pos(y,x+place-1,fileout)
         write (fileout) line(place:size)
         Call Pos(y,x+place-1,fileout)
         return
         end

         Subroutine InsertChar(line,place,y,x,ch,CharLength)
         implicit none
```

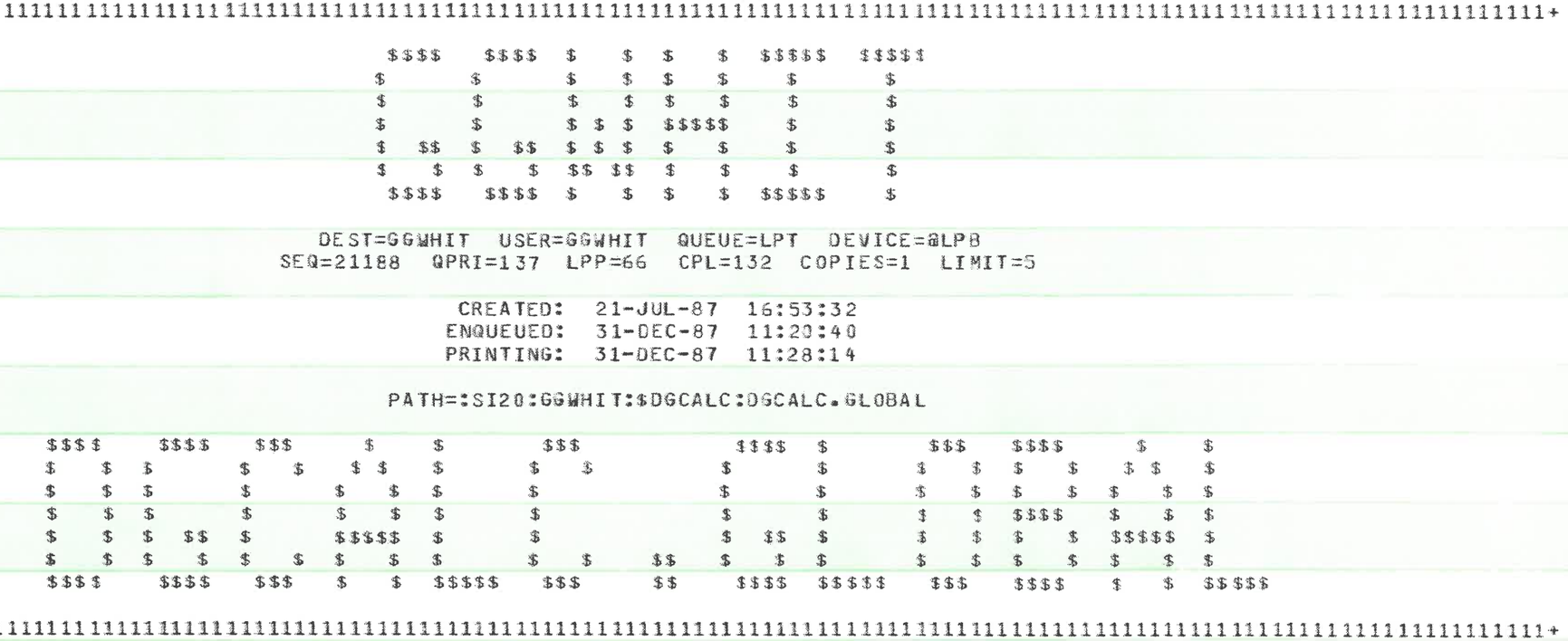
```
      common /ecomm/ esc, size, filein, fileout
      logical esc
      character*(*) line
      character*1 GetChar, ch
      integer place, filein, fileout, size, y, x, ich, index,charlength
      write (fileout) " ",line(place:size)
      Call Pos(y,x+place-1,fileout)
10    ch = GetChar(filein)
      ich = ichar(ch)
      if (ich.eq.1) then
         write(fileout) line(place:size-1)," ",line(size:size)
         place = size
         Call Pos(y,x+place-1,fileout)
      else if (ich.eq.5) then
         write(fileout) line(place:size)," "
         Call Pos(y,x+place-1,fileout)
         return
      else if (ich.eq.8) then
         place = 1
         Call Pos(y,x,fileout)
         write(fileout) " ",line(1:size)
         Call Pos(y,x,fileout)
      else if (ich.eq.10) then
           Call Pos(y,x,fileout)
           write(fileout) line(1:size)," "
           Call Pos(y,x+place-1,fileout)
           return
      else if (ich.eq.11) then
           line(place:)=" "
           write(fileout) line(place:size)
           Call Pos(y,x+place-1,fileout)
      else if ((ich.eq.24).and.(place.lt.size)) then
           write(fileout) line(place:place)," ",line(place+1:size)
           place = place + 1
           Call Pos(y,x+place-1,fileout)
      else if (ich.eq.25) then
           Call Pos(y,x+place-2,fileout)
           if (place.gt.1) place = place-1
           write(fileout) " ",line(place:size)
           Call Pos(y,x+place-1,fileout)
      else if (ich.eq.27) then
           esc = .true.
           Call Pos(y,x,fileout)
           write(fileout) line(1:size)," "
           return
      else if ((ich.ge.32).and.(ich.le.126)) then
         if (place.lt.size) write(fileout) ch
         do index = size-1,place,-1
            line(index+1:index+1)=line(index:index)
         end do
         line(place:place)=ch
         if (place.lt.size) then
            place = place + 1
            Charlength=charlength+1
         end if
         write(fileout) " ",line(place:size)
         Call Pos(y,x+place-1,fileout)
      else if ((ich.eq.127).and.(place.gt.1)) then
         place = place - 1
         if (charlength.ne.0) charlength=charlength-1
         line(place:size-1) = line(place+1:size)
         line(size:size) = " "
         Call Pos(y,x+place-1,fileout)
```

```
            write(fileout) * *,line(place:size)
            Call Pos(y,x+place-1,fileout)
         else
            write(fileout) *<007>*
         end if
         go to 10
         end

         Character*1 Function GetChar(fileno)
         implicit none
         integer fileno
         character*1 ch
         read(fileno) ch
         GetChar = ch
         return
         end

         Subroutine GetPos(fileout,filein,x,y)
         integer fileout,filein,x,y
         character*(3) loc
         write (fileout) *<005>*
         read (filein) loc
         y = ichar(loc(3:3))
         x = ichar(loc(2:2))
         return
         end

          subroutine IntToChars(number,q,h,l)
c  Convert an integer into a three digit number which has the base 256.
          implicit none
          integer number
          character*1 q,h,l
          q=char(number/(256*256))
          h=char(mod(number,256*256)/256)
          l=char(mod(number,256))
          return
          end
```



Kansas Geological Survey AOS/VS

AOS/VS REV 07.57
AOS/VS XLPT REV 07.56

```
c       ********* This is a file called DGCALC.GLOBAL *********
        integer Black, White
        parameter ( Black=0, White=1 )
        character*1 defsig
        integer ArraySize*4
        integer HSheetSize
        integer VSheetSize
        parameter (ArraySize=300000)
        parameter (HSheetSize=100)
        parameter (VSheetSize=255)
        character*1 FormulaArray(ArraySize)
        character*(1920) ScreenBuffer
        character*(80) Spaces, Stars, Dashes, Wrong
        character*(20) WorkFileName
        logical Changed, Empty
        integer CellPointer(HSheetSize,VSheetSize)
        integer CurrX,CurrY,NextData
        integer CellWidth(HSheetSize)
        integer Top, Left, ScreenPtr
        integer OutFile, InFile, SaveFile, LoadFile
        common /FNames/ WorkFileName
        common /Chara/ defsig
        common /CBuffer/ ScreenBuffer
        common /IBuffer/ ScreenPtr
        common /Lines/ Spaces, Stars, Dashes, Wrong
        common /Files/ OutFile, InFile, SaveFile, LoadFile
        common /CSheet/ FormulaArray
        common /ISheet/ CellPointer,CurrX,CurrY,CellWidth,Top,Left
        common /IPointer/ NextData
c       ********* End of DGCALC.GLOBAL *********
```

FORM 1411 X 08
PRINTED IN U.S.A.

+2222222222

```
c       ********* This is a file called DGCALC.GLOBAL1 *********
         integer Currentpos,Intvalue,Eol,StringLen
         integer IntstackPtr,RealStackPtr,BoolStackPtr,StringStackPtr
         character*(240) StringStack(10)
         integer IntStack(50)
         real RealStack(50)
         logical BoolStack(50)
         character*11 resword(23)
         real realvalue
         logical boolvalue
         integer identlen
         integer StrLenStack(10)
         character*240 stringvalue
         character*1 nextchar,currentchar
         character*11 token
         character*11 exprfollow,celltype
         character*11 restoken(23)
         character*11 ident
         common /INumber/Eol,identlen,currentpos
         common/IPtr/ IntStackPtr
         common/IStack/ IntStack
         common/Ivalue/ IntValue
         common/RPtr/ RealStackPtr
         common/RStack/RealStack,RealValue
         common/BPtr/ BoolStackPtr
         common/BStack/ BoolStack,BoolValue
         common/SPtr/ StringStackPtr,StrLenStack,StringLen
         common/SStack/ StringStack,StringValue
         common/Setofchar/ nextchar,currentchar
         common/Tokentype/ token,exprfollow,restoken,celltype
         common/Keytype/ resword
         common/CIdent/Ident
c       ********* This is the end of DGCALC.GLOBAL1 *********
```

FORM 14/1 5 GB
PRINTED IN U.S.A.

+3333333333

```
c       ******* This is a file called DGCALC.PARAMS *******
c        This file only contains the definitions for commands.
        integer QuitSys
        parameter(QuitSys=1000)
        integer NoCommandSys
        parameter(NoCommandSys=0)
        integer MoveRightSys
        parameter(MoveRightSys=1)
        integer MoveLeftSys
        parameter(MoveLeftSys=2)
        integer MoveUpSys
        parameter(MoveUpSys=3)
        integer MoveDownSys
        parameter(MoveDownSys=4)
        integer MoveDownScrSys
        parameter(MoveDownScrSys=5)
        integer MoveUpScrSys
        parameter(MoveUpScrSys=6)
        integer MoveLeftScrSys
        parameter(MoveLeftScrSys=7)
        integer MoveRightScrSys
        parameter(MoveRightScrSys=8)
         integer MoveLeftEndSys
         parameter (MoveLeftEndSys=9)
        integer MoveRightEndSys
        parameter(MoveRightEndSys=10)
        integer MoveHomeSys
        parameter(MoveHomeSys=11)
        integer MoveDownEndSys
        parameter(MoveDownEndSys=12)
         integer MoveUpEndSys
         parameter (MoveUpEndSys=13)
    c
    c
          integer DoCliCommandSys
          parameter(DoCliCommandSys=20)
          integer GotoLocationSys
          parameter(GotoLocationSys=21)
          integer HelpSys
          parameter(HelpSys=22)
          integer ChangeFormatSys
          parameter(ChangeFormatSys=23)
          integer LoadSys
          parameter(LoadSys=24)
          integer SaveSys
          parameter(SaveSys=25)
          integer CopySys
          parameter(CopySys=26)
          integer TypeSys
          parameter(TypeSys=27)
          integer RecalculateSys
          parameter(RecalculateSys=28)
          integer InsertSys
          Parameter(InsertSys=29)
          integer DeleteSys
          Parameter(DeleteSys=30)
          integer PrintSys
          Parameter(PrintSys=31)
           integer MoveAreaSys
           parameter ( MoveAreaSys = 32 )
           integer EraseSys
           parameter ( EraseSys = 33 )
           integer JoinSys
```

```
      parameter ( JoinSys = 34 )
c     ********* End of DGCACL.PARAMS *********
```

```
 $$$$   $$$$  $   $  $   $  $$$$$  $$$$$
$       $     $   $  $   $  $      $
$       $     $   $  $   $  $      $
$       $     $ $ $  $$$$$  $      $
$   $$  $   $$  $ $ $  $   $  $      $
$    $  $    $  $$ $$  $   $  $      $
 $$$$   $$$$  $   $  $   $  $$$$$    $
```

DEST=GGWHIT USER=GGWHIT QUEUE=LPT DEVICE=@LP8
SEQ=21187 QPRI=137 LPP=66 CPL=132 COPIES=1 LIMIT=4

CREATED: 24-MAY-86 17:42:46
ENQUEUED: 31-DEC-87 11:20:40
PRINTING: 31-DEC-87 11:28:09

PATH=:SI20:GGWHIT:$DGCALC:DGCALC.CELLTYPE

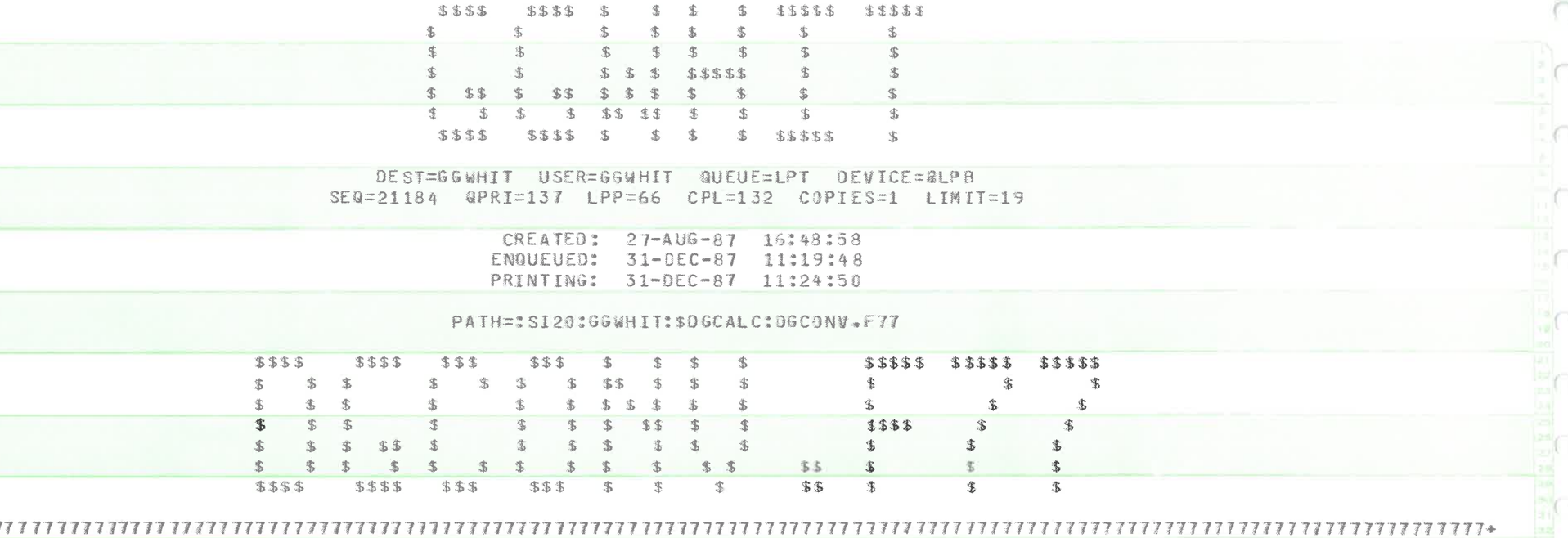
```
$$$$    $$$$   $$$      $    $       $$$          $$$   $$$$$  $       $       $$$$$  $   $  $$$$   $$$$$
$   $   $      $   $    $ $  $      $   $         $   $  $      $       $         $    $   $  $   $  $
$   $   $      $       $   $ $      $             $      $      $       $         $    $ $ $  $   $  $
$   $   $      $       $   $ $      $             $      $$$$   $       $         $    $ $ $  $$$$   $$$$
$   $   $   $$ $       $$$$$ $      $             $      $      $       $         $    $ $ $  $   $  $
$   $   $   $  $   $   $   $ $      $   $    $$   $      $      $       $         $    $ $ $  $   $  $
$$$$    $$$$   $$$   $   $  $$$$$   $$$     $$    $$$   $$$$$  $$$$$   $$$$$     $     $    $   $  $$$$$
```

Kansas Geological Survey AOS/VS

AOS/VS REV 07.57
AOS/VS XLPT REV 07.56

```
c       ********* This is a file called DGCALC.CELLTYPE *********
         character*1 Textcell,Intcell
         character*1 Realcell, Formulacell
         character*1 Plotcell
         character*1 Truecell, Falsecell,Wrongcell
         character*1 Eos, Unused
         integer Cellheader
         parameter (Cellheader=4)
         character*1 sp
         character*1 nul
         nul=char(0)
         sp=char(32)
         wrongcell=char(240)
         textcell=char(241)
         realcell=char(242)
         intcell=char(243)
         formulacell=char(244)
         plotcell=char(245)
         truecell=char(247)
         falsecell=char(248)
         eos=char(253)
         unused=char(254)
c       ********* End of DGCALC.CELLTYPE *********
```

FORM 1411 4 GB
PRINTED IN U.S.A.



Kansas Geological Survey AOS/VS

AOS/VS REV 07.57
AOS/VS XLPT REV 07.56

```
c    This is a program to convert a text data file, which is generated
c    by D_E_Prog with the form specified by WATCHEM.FORM or a text
c    editor into a file which has the form required by DGCALC.
c        Aug. 14, 1987 by Hong Zhou in Kansas Geological Survey.

c        The process to create executable program
c        )F77/LINEID DGCONV
C        )F77LINK DGCONV  EDIT  LIB
C        ignore the error message printed on the screen while linking
c
         integer  Count, INDEX, SUM, TOTAL, N1, N2, Xmax, Ymax
         integer LENGTH, DGLENGTH, I, J, K, ArraySize,CellLen,BP
         integer DGHEAD, DGTAIL, INHEAD, INTAIL
         character*1 Q, H, L, TYPE(100)
         parameter (ArraySize = 300000)
         integer Size(100), Sig(100)  !Sig : digits after decimal
         character*1 DGArray(ArraySize)
         character*271  BUFFER
         character InString*(40), OutString*(40),Temp*40
         integer KBD, OutFile, InFile, LoadFile,SaveFile,FormFile
         character LoadName*40, SaveName*40
         integer Pointer (100, 255)
         logical EMPTY, Yes, YesNo, AlignDecimal,FileForm,CR
         common /Files/ KBD,OutFile,InFile,LoadFile,SaveFile,FormFile
         logical Go, ESC
            InFile = 8
            OutFile = 7
            open(InFile,File='@input',Form='unformatted',mode='binary',
     &       recfm='Dynamic',ScreenEdit='No',IOStat=I)
            open(OutFile,File='@Output',Form='unformatted',mode='binary',
     &       recfm='Dynamic',CarriageControl='None' )
c        Open input and output files.
         KBD = 99     ! Key board input
         open (KBD, File='@input', Form='unformatted', Mode='Text',
     &         Recfm='Dynamic', ScreenEdit='no', IOStat= IoFlag )
         call Message    ! Message about the program
         print *, 'Do you want to continue ? Y/N '
         if ( YesNo (KBD) .eqv. .false. ) go to 9999
         FormFile = 70   ! Form file WATECHEM.FORM
         LoadFile  = 80    ! Input file
         SaveFile = 90    ! Output file
100      print *,'Type input file name to load :'
         call Edit(OutFile,InFile,LoadName,Index,40,ESC)
         if (ESC) then
           print *,'Do you want to exit ? [Y/N]'
           ESC = YesNo (KBD)
           if (ESC) then
              stop
           else
             go to 100
           end if
         end if
         open (LoadFile,File=LoadName,recfm='Dynamic',Form='Unformatted',
     &   mode='Binary', status='old',IOStat= IoFlag )
         if (IOFlag .ne. 0) then
           print*,'Error ',IOFlag, '  Something wrong in file ', LoadName
           print*,'Check if it is in the current directory ! Type ESC to quit.'
           stop
         end if
c
         call Initialize(Pointer,100,255,DGLengh,DGTail,Tail)
         CR = .false.
```

```
      Length = 0
      print 90
90    format (' You can choose the data form which can be defined by'/'
     &  either the form file WATCHEM.FORM or your own. If you type Y, the program'/'
     &  automatically reads data form from WATCHEM.FORM, which is in GGWHIT directory.'/'
     &  If you type N, you need follow the instructions on the screen and answer'/'
     &  every question correctly.'//'
     &  Read form from WATCHEM.FORM ? [Y/N]')
      FileForm = YesNo (KBD)
      if (.not. FileForm) then
95      print *
        print *,'How namy columns to read from input file ? (integer)'
        read *, Xmax
        if ((Xmax.lt.1).or.(Xmax.gt.100)) then
          print*,'The minimun column is 1 and the maximun is 100.'
          print *,'<007>'
          go to 95
        end if
        print *,'Does each record has a CARRIAGE RETURN at last ? [Y/N]'
        CR = YesNo (KBD)
        print *
        do I = 1, Xmax
          II = I
          call AssignCodes (Type(I),Size(I),Sig(I),II)
        end do
      else
        open (70, File=':udd:ggwhit:WATCHEM.FORM',RECFM='DS',
     &    MAXRECL=36, Form='formatted',IOStat=IOFlag)
        if (IOFlag .ne. 0) then
              print*,'Something wrong in file WATCHEM.FORM.'
              print*,'Check if it is in :UDD:GGWHIT directory.'
              stop
          end if
          Xmax = 41
          call ReadSize (FormFile, Size, Length)
          call AssignSig (Sig,AlignDecimal)
          call AssignTypeCode (Type,AlignDecimal)
        end if
        if (FileForm) then
          Length = 271
        else
          Length = 0
          do I = 1, Xmax
            Length = Length + Size(I)
          end do
        end if
c   Read characters from input file
15      print *, ('Start  at  the record number :' )
        read *, N1
        print *, ('Stop after the record number :' )
        read *, N2
        if (N1.lt.1) N1 = 1
        if ((N1.gt.N2).or.(N1.lt.1)) then
          print *, 'Wrong range ! Do again.'
          go to 15
        end if
c       Skip the first N1-1 records
        if ( N1 .gt. 1 ) then
          do I = 1, N1-1
                read (LoadFile,end=9999) Buffer (1:Length)
                if (CR) then
                  read(LoadFile) Q ! Q is Hard Return
                end if
```

```
            end do
          end if
c
          write(*,150)
150       format (/' The following statement refers to the question below.'//'
     &    If you want the decimal points of real numbers aligned at a certain position'/'
     &    for the whole column, please type Y for yes.  However, these numbers cannot'/'
     &    be used for mathematical calculations since they will have TEXT FORMAT.'//'
     &    If you want to use real numbers for mathematical calculation, please type N'/'
     &    so that the real numbers will have the real number format.'//)
          print *, 'Do you want to align decimal points ? [Y/N]'
          AlignDecimal = YesNo (KBD)
          print *
c
c         Read data from input file
          print *,'Type output file name:'
          call Edit(OutFile,InFile,SaveName,Index,40,ESC)
          open(SaveFile,File=SaveName,IOIntent='output',Form='Unformatted',
     &    STATUS='FRESH',mode='binary', recfm='dynamic',IOStat= IoFlag)
          if (IOFlag .ne. 0) then
            print*,'Something wrong in file ',SaveName
            stop
          end if
          InLength = 0
          Count = 0
c   Assign input array to DGArray
500       if ((Count.ge.(N2-N1+1)).or.(Count.ge.255)) go to 9990
            read (LoadFile,End=9990)Buffer(1:Length)
            Count = Count + 1
            if (CR) read (LoadFile) Q ! Skip Hard Return
            if (FileForm) then
               BP = 2
            else
               BP = 0    !Buffer Pointer
            end if
            do I = 1, Xmax
              CellLen = Size(I)
              Temp(1:CellLen) = Buffer(BP+1:BP+CellLen)
              BP = BP + CellLen
              Empty = .true.
              J = 0
              do while ((J .lt. CellLen) .and. (Empty .eqv. .true.))
                    J = J + 1
                    if (Temp(J:J) .ne. ' ') then
                          Empty = .false.
                    end if
              end do
              if (.not. Empty) then
                Pointer(I,Count) = DGTAIL + 1 !pointer of one cell
                if ((AlignDecimal) .and. (Type(I).eq.'R')) then
                   DGArray( DGTAIL+1 ) = char(241) ! character type
                else
                  if (Type(I).eq.'T') then
                     DGArray(DGTail+1) = char(241) !Character type
                  else if (Type(I).eq.'I') then
                     DGArray(DGTail+1) = char(243) !Integer type
                  else if (Type(I).eq.'R') then
                    DGArray(DGTail+1) = char(242) !Real type
                  end if
                end if
               if (TYPE(I) .eq. 'T' ) then
                  DGArray( DGTAIL+2 ) = char(3)
                  do J = 1, CellLen
```

```
                DGArray( DGTAIL + 4 + J ) = Temp (J:J)
              end do  ! Copy input array to DGArray
            else if (TYPE(I) .eq. 'I' ) then
              DGArray( DGTAIL+2 ) = char(0)
              call DeleteBlank (Temp,CellLen)
              do J = 1, CellLen
                 DGArray( DGTAIL + 4 + J ) = Temp (J:J)
              end do  ! Copy input array to DGArray
            else if (TYPE(I) .eq. 'R' ) then
              DGArray( DGTAIL+2 ) = char (Sig(I))
              if (.not. AlignDecimal) then
                 call DeleteBlank (Temp,CellLen)
                 do J = 1, CellLen
                    DGArray (DGTail + 4 + J) = Temp(J:J)
                 end do
              else
                 call DecimalAlign(Temp,OutString,CellLen,Sig(I))
                 do J = 1, CellLen
                    DGArray ( DGTail + 4 + J ) = OutString (J:J)
                 end do
              end if
            else
                 Print*, 'Something wrong !'
                 stop
            end if
              DGArray( DGTAIL + 3) = char (I) ! column code
              DGArray( DGTAIL + 4) = char (K) !  Row code
              DGArray (DGTAIL + CellLen + 5) = char (253) ! End Flag
              DGTAIL = DGTAIL + CellLen + 5
            end if
          TAIL = TAIL + Size(I)
            end do  ! Copied 41 column data to DGArray
          go to 500
c    Assign initial DGFILE CODES to DGArray
9990      write(SaveFile) char (253)
          call IntToChars ( DGTAIL, Q, H, L )
          write (SaveFile) Q,H,L
c    Assign row and column codes by finding the max rows and columns
          Xmax = 0
          Ymax = 0
          do J = 1,255
             do  I = 1,100
                if (Pointer(I,J).ne.0) then
                  if (J .gt. Ymax) Ymax = J
                  if (I .gt. Xmax) Xmax = I
                end if
             end do
          end do
          write (SaveFile) char(Xmax) ! save column code
          write (SaveFile) char(Ymax) ! save row code
c     Save DGArray into DGCALC input file.
          do I = 1, DGTAIL
            write(SaveFile) DGArray ( I )
          end do
c    Convert pointers to ASC II code and store them into DGArray
          do J = 1, Ymax
            do I = 1, Xmax
                 call IntToChars(POINTER(I,J), Q, H, L)
                 write (SaveFile) Q, H, L
            end do
          end do    ! Pointers are saved

c    Save column information
```

```
            do I = 1, Xmax
                    Q = char (SIZE(I) + 3)
                    write (SaveFile) Q
            end do      ! Save width of 41 columns
            do I = Xmax + 1, 100
                    Q = char (8)
                    write (SaveFile) Q
            end do

            print *, 'The output file is saved to ',SaveName
            print *
            print *,'Do you want to process data in the same input file ?[Y/N]'
            close (SaveFile)
            if (YesNo(KBD) .eqv. .true.) then
              rewind LoadFile
              call Initialize(Pointer,100,255,DGLengh,DGTail,Tail)
              go to 15
            end if
    9999    close (FormFile)
            close (LoadFile)
            close (SaveFile)
            print *
            print *,'Do you want to process another input file ? [Y/N]'
            if (YesNo(KBD) .eqv. .true.) then
                    go to 100
            end if
            close (KBD)
            print *,'Thank you for using DGCONVERTER.  Goodbye !'
            end     ! End of main program

            subroutine AssignCodes (Type,Size,Sig,I)
    c       Assign the Type, Size, and Significant Figure to column I.
            character*1 Type,UpperCase
            integer Size,Sig,I
            logical Yes,YesNo
            common /Files/KBD
            print *,'Please type the cell information for column ',I
    100     print *,'What is the cell type of column ',I,' ? [Real/Integer/Text]'
            read (KBD) Type
            Type = UpperCase (Type)
            if ((Type.ne.'R').and.(Type.ne.'I').and.(Type.ne.'T')) go to 100
    c       Assign Size
    110     print *,'What is the width of column ',I,' ? (integer)'
            read *, Size
            if ((Size.lt.1).or.(Size.gt.75)) then
              print *,'The range of size is from 1 to 75.'
              go to 110
            end if
    c       Assign Significant figure for a real number
            if ( Type .ne. 'R') then
              Sig = 0
            else
    120       print *,'What is the number of significant figures for column ',I,' ? (Integer)'
              read *, Sig
              if ((Sig.lt.0).or.(Sig.gt.20)) then
                 print *,'The range is 0 to 20.'
                 go to 120
              end if
            end if
            print *,'For column ',I
            if (Type.eq.'R') then
               print *,'Type            REAL NUMBER'
            else if (Type.eq.'I') then
```

```
            print *,'Type            INTEGER NUMBER'
          else if (Type.eq.'T') then
            print *,'Type            TEXT'
          end if
          print *,   'Width           ', Size, ' characters'
          if (Type.eq.'R') then
            print *, 'Sig. Figure     ', sig
          end if
          print *,' '
          print *,'Are they correct ? [Y/N]'
          Yes = YesNo (KBD)
          if ( .not. Yes) go to 100
          print *
          return
          end  ! Assign Codes

          subroutine AssignTypeCode (Type,AlignDecimal)
C     Assign type code for each column.
c       T for Text, I for integer, R for real.
          integer I
          character  TYPE ( 41 )
          logical AlignDecimal
          do I = 1, 3
                  TYPE(I) = 'T'
          end do
          TYPE(4) = 'I'
          TYPE(5) = 'T'
          TYPE(6) = 'I'
          TYPE(7) = 'T'
          TYPE(8) = 'R'
          do I = 9, 12
                  TYPE(I) = 'T'
          end do
          do I = 13, 41
                  TYPE(I) = 'R'
          end do
          return
          end   ! End of Assign Type Code

          subroutine AssignSig (Sig,AlignDecimal)
c       Assign digits after decimal points for real numbers.
          integer  Sig ( 41 )
          logical AlignDecimal
          integer I
          do I = 1, 3
             Sig ( I ) = -1      ! For Text
          end do
          Sig ( 4 ) = 0          ! for Integer
          Sig ( 5 ) = -1
          Sig ( 6 ) = 0
          Sig ( 7 ) = -1
          Sig ( 8 ) = 1          ! For a real
          do I = 9, 12
              Sig ( I ) = -1     ! For text
          end do
          Sig ( 13 ) = 0
          Sig ( 14 ) = 0
          Sig ( 15 ) = 2
          Sig ( 16 ) = 2
          do I = 17, 23
               Sig ( I ) = 1
          end do
          Sig ( 24 ) = 2
```

```
            Sig ( 25 ) = 1
            Sig ( 26 ) = 0
            do I= 27, 29
               Sig ( I )= 1
            end do
            do I = 30, 34
               Sig ( I ) = 3
            end do
            do I = 35, 36
               Sig ( I ) = 2
            end do
            do I = 37, 38
               Sig ( I ) = 4
            end do
            Sig ( 39 ) = 3
            Sig ( 40 ) = 4
            Sig ( 41 ) = 3
            return
            end   ! End of Assign Sig

            subroutine ReadSize ( FmFile, Size, Length)
      c   Read column size from Form File
            integer I, Other0, Other1, Other2
            integer Size ( 41 ), Length
            integer FmFile  ! Form File
            do I = 1, 90
                  read ( FmFile, 5 )
            end do   ! Skip first 90 lines
      5     format ( / )
            do I = 1, 41          ! Read form from form file
               read ( FmFile, 10 ) Other0, SIZE(I), OTHER1, OTHER2
      c        print 20, Other0, SIZE(I), Other1, Other2, I
               LENGTH = LENGTH + SIZE( I )
            end do
      10    format ( 6X, I2, 6X, I2, 6X, I2, 6X, I2 / )
      20    format ( ' The inputs : ', 4I6, '. The reading ', I2 )
            return
            end   ! End of reading SIZE

            subroutine DecimalAlign (InString, OutString, Size, Sig )
      c     Align the output string according to Sig, the digits on the
      c     roght of the decimal point.
            character InString *(20), OutString *(20)
            integer Size, Sig, P, S, I, FindPosition
            OutString ( Size-Sig:Size-Sig ) = '.'
            P = FindPosition ('.', InString) ! Decimal position in InString
            S = Size - Sig              ! Decimal position in OutString
            if ( S .eq. P ) then
                  OutString = InString
            else if ( P .gt. S ) then
               OutString(1:S-1) = InString (P-S+1:P-1)
               OutString(S+1:Size-(P-S)) = InString( (P+1):Size )
               do I = Size - (P-S), Size
                    OutString (I:I) = ' '
               end do
            else if ( P.lt. S ) then
               OutString(S+1:Size) = InString(P+1:Size - (S-P) )
               OutString (S-P+1:S-1) = InString (1:P-1)
               do I = 1, S-P
                    OutString ( I:I ) = ' '
               end do
               if ( OutString(S-1:S-1) .eq. ' ' ) then
                    OutString ( S-1:S-1 ) = '0'
```

```
            end if
          end if
          return
          end    ! End of Decimal point align

          integer function FindPosition (Obj, Target)
c     Return location of Obj in Target. Position = 0 if not found.
          character Target *(*)
          character*1 Obj
          integer I
          logical Found
             I = 1
          do while ( Target(I:I) .eq. ' ' )
                I = I + 1
          end do            ! Skip blanks
          Found = .false.
          do while ((Target(I:I) .ne. ' ').and. ( Found .eqv. .false. ))
             if ( Target ( I:I ) .eq. Obj ) then
                  Found = .true.
             else
                  I = I + 1
             end if
          end do
          FindPosition = I
          return
          end    ! End of find the position of decimal point

          subroutine Message
c     Print the message about the program
          write (*,*) '<014>'   !Clear screen
          print  50
   50     format ('        Wellcome to use DGCONVERTER.'//'
         &  It reads data from an input file and converts the input file into an'/'
         &  output file which has the format required by the DGCALC.'//'
         &  The input file can be generated by D_E_PROG or any other text editor.'/'
         &  If it is generated by D_E_PROG, the program will read  form file'/'
         &        WATCHEM.FORM    from GGWHIT directory.'//'
         &  If input file is generated by other text editor, make sure only use'/'
         &  non-document mode to save it.'//'
         &  If do not want to read form file WATCHEM.FORM, the user needs to define'/'
         &  his own form by answering many questions correctly.'//'
         &  The input and output files name can be no more than 40 characters.'/'
         &  The user needs to specify the starting and ending records by typing two'/'
         &  numbers. The maximum rows is 255 and maximun column is 100, which are the'/'
         &  limit of DGCALC.'//)
          return
          end   ! End of print message

          logical function YesNo (Channel)
c     It return YesNo = true if Y or y is typed, YesNo = false
c     if N or n is typed.
          integer Channel
          character Ch
          Ch = ' '
          do while ((Ch .ne. 'y') .and. (CH .ne. 'Y') .and.
         &          (Ch .ne. 'n') .and. (Ch .ne. 'N'))
              read (Channel) Ch
          end do
          if ((Ch .eq. 'Y') .or. (Ch .eq. 'y')) then
                YesNo = .true.
          else
                YesNo = .false.
          end if
```

```
      return
      end     ! End of YesNo

      subroutine Initialize(Pointer,Col,Row,DGLengh,DGTail,Tail)
      integer Col,Row,DGLength,DGTail,Tail,I,J
      integer Pointer(Col,Row)
c   Initialize pointers. I is the row index and J is the column index.
      do I = 1, Col
        do J = 1, Row
              Pointer (I, J) = 0
        end do
      end do       ! Set pointers to 0

      DGLENGTH = 0
      DGTAIL = 0
      TAIL = 0
      return
      end ! Initialize

      subroutine DeleteBlank (InString, Len)
c     Delete the blank from InString and return the new length
      integer Len, I,TLen
      character*(*) InString, TString*100
      TLen = 0
      do I = 1, Len
       if (InString(I:I).ne.' ') then
         TLen = TLen + 1
         TString (TLen:TLen) = InString(I:I)
       end if
      end do
      Len = TLen
      InString(1:Len) = TString(1:Len)
      return
      end  ! Delete Blank
```