MILITARY INTERACTIVE SYMBOL DESIGN PACKAGE

by

JUSTIN GUY BALLOU, III

B.S., Embry-Riddle Aeronautical University, 1975

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

Approved by:

Major Professor

TABLE OF CONTENTS

Page

LIST OF FIGURES ii

I. INTRODUCTION 1

A. BACKGROUND 1
B. OBJECTIVE 1
C. APPROACH 1
D. ORGANIZATION 2

II. SYSTEM REQUIREMENTS AND SPECIFICATIONS 3
A. REQUIREMENTS 3

B. SPECIFICATIONS 9
1. Military Symbols 9
2. Map System 12

III. SYSTEM DESIGN AND IMPLEMENTATION 14

A. SYSTEM DESIGN 14
1. Chromatics Terminal 14
2. Soltec Plotter 15
3. Peripheral Drivers 15
4. Human Interface 16

B. IMPLEMENTATION 16
1. Chromatics 17
2. Soltec Plotter 20
3. Interactive Follow Through of Drawing a Rectangle 22
4. Human Interface 24
a. Man-Machine Communication 24
b. Display 25

IV. SUMMARY 26

A. CONCLUSIONS 26
B. RECOMMENDATIONS 26

BIBLIOGRAPHY 28

APPENDICES
APPENDIX A (User's Guide) 31
APPENDIX B (Source Code) 37
APPENDIX C (Procedure Dictionary) 79

LIST OF FIGURES

Page

Figure 1. Display Screen 6

Figure 2. Screen Coordinates Transformation 7

Figure 3. 1:50,000 Map Grid 13

Figure 4. Chromatics Screen 25

Figure 5. Chromatics Display Screen 33

I. INTRODUCTION

A. BACKGROUND

On the battlefield of the future, a varied and complex combination of people, cultures, and languages in a vast geographical area could place the United States Army in a difficult command and control situation. Because of the lack of a common language on the battlefield, the commander of a multi-national army will find it extremely difficult to communicate time sensitive requirements to his subordinate units. One potential solution is the use of high resolution graphics using common military graphic symbology. The commanders would be able to use a computer graphic terminal to display the desired military situation and transfer the information to their subordinate units in the form of an overlay by means of a plotter.

B. OBJECTIVE

The objective of this project is to design and implement an interactive graphic package which will build military symbols and transfer the military symbols to a desired location on an overlay using the U.S. Military Grid Reference System.

C. APPROACH

The overall approach for this project was divided into two phases: 1. system requirements and specifications, 2. system design and implementation. The system design and

implementation was divided into two phases. First, the Chromatics driver was integrated with the 8/32 Interdata. This allowed the user to interactively draw military symbols on the Chromatics graphic terminal. Secondly, the Soltec driver was integrated into the program to be able to transfer the designed military symbols to an overlay on the plotter.

D. ORGANIZATION

The first chapter of this document contains discussions of the rationale for the project, the stated objective and the approach used to accomplish the objective. The second chapter provides the project requirements and specifications. The third chapter contains the system design and implementation on the Chromatics terminal, the Soltec plotter, and the human interface between user and computer. The last chapter presents the conclusions that were derived from this effort and the recommendations for further work in this area. Appendices include the user's manual, source code and procedure dictionary.

II. SYSTEM REQUIREMENTS AND SPECIFICATIONS

A. REQUIREMENTS

Having decided to design and implement an interactive graphic package which will build military symbols and transfer the symbols to an overlay, the first step was to do a literature search of the available command and control (C&C) systems. The Defense Technical Information Center (DTIC) was used to identify the C&C systems now being used. Review of literature was centered on the Command and Control Technical Center (CCTC) of the Defense Communications Agency, CCTC of Supreme Headquarters Allied Powers Europe (SHAPE) and the World Wide Military Command and Control Systems (WWMCCS). [Lov80, Sch80, Dav77]

After extensive review of manuals and reports of C&C system, it was realized that the graphic package developed for this project would have to be limited to the basic essential functional elements of a C&C system; however, the requirements of an optimum system will be reviewed. This will make it easier to see how this project fits in an overall command and control system.

A model C&C system requires three main components: display screen, database and plotter. The three components will be discussed in reverse order.

The plotter should be of sufficient size to be able to draw overlays for a 1:50000 map sheet (18" x 20"). The plotter should have a resolution to match the display device

(100 pixel per inch) and it should have a set of primitives which would draw the necessary military symbols and graphics by passing the primitives commands straight from the display device to the plotter. The inherent storage capacity of the plotter should match the capacity of the display device so that upon transmission of a buffer, the plotter would have the same capability.

The database for the C&C system should have three schemes: the map scheme, the military symbols and graphics scheme, and the report scheme.

The map scheme should be large enough to include the commander's area of operation (AO), both flanks' AO's, and the area to the front and rear of the commander's AO and flanks. The map scheme will always differ somewhat depending on the military operations and personality of the commanders. The map scheme should be broken into window sizes which will be of the same size as the viewport of the display terminal.

The scheme for the military symbols and graphics should include all the military symbols in FM 21-30 [MIL70] and military graphics in FM 101-5-1 [OPE80]. The subscheme for military symbols could be broken down into three groups: basic symbols, functional symbols, and size symbols. The military graphics could be broken down into operational groups such as: offense, defense, delay, retrograde, and special operations.

The report scheme should consist of all the reports required by the command's Standard Operating Procedure (SOP) and, to facilitate the quick completion of the reports, a subscheme of common phrases should be included by operation or alphabetical order.

Being that one of the primary reasons for the display in a C&C system is to draw overlays, the display screen should be at least 1 meter by 1 meter; however, another feasible display device could be the use of a video projector. The video projector would show the whole AO with a window projected on the video screen which would correspond with a high resolution display terminal screen for greater detail. The display device should have a resolution of at least 100 pixels per inch. The commander must be able to see as much of his AO as possible at one time; therefore, the map window should be as large as the display screen will allow. The only exception would be a viewport which would display the X and Y coordinates along the bottom and left side of the display (Fig. 1).

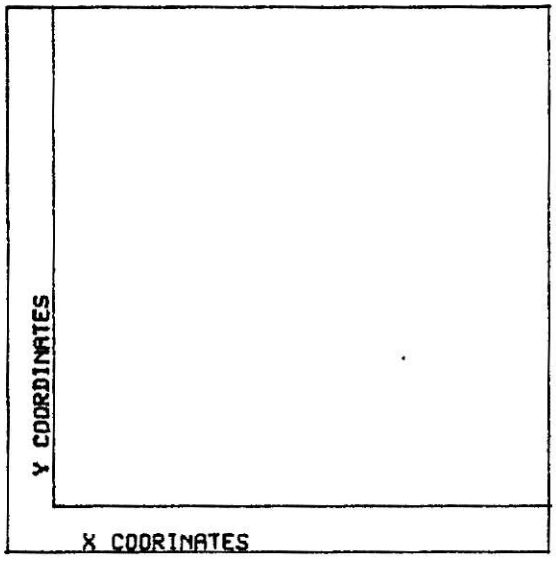


Figure 1. Display Screen

Any standard map segment can be scrolled in the viewport by use of a 16 position joystick, 9 position cursor/control or by the MOVE command on the keyboard. The key to changing the map segments in the viewport is by keeping track of one point in the window as it is moved in the viewport. This will require the transformation of the map coordinates (X_m, Y_m) into screen coordinates (X_S, Y_S). The most commonly kept track of point (0, 0) is the lower left corner of the viewport. To keep track of this point (0, 0), it requires a two step operation. First, the map coordinates must be transferred to screen coordinates and secondly, the translation of the (0, 0) point must be accomplished.

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

The transformation of map to screen coordinates requires the use of the two following operations [NEW79]. Figure 2 illustrates the transformation from map to screen coordinates.

$$X_s = \frac{V_{xr} - V_{xl}}{M_{xr} - M_{xl}} (X_m - M_{xL}) + V_{xL} \qquad (1)$$

$$Y_s = \frac{V_{yt} - V_{yb}}{M_{yt} - M_{yb}} (Y_m - M_{yb}) + V_{yb} \qquad (2)$$

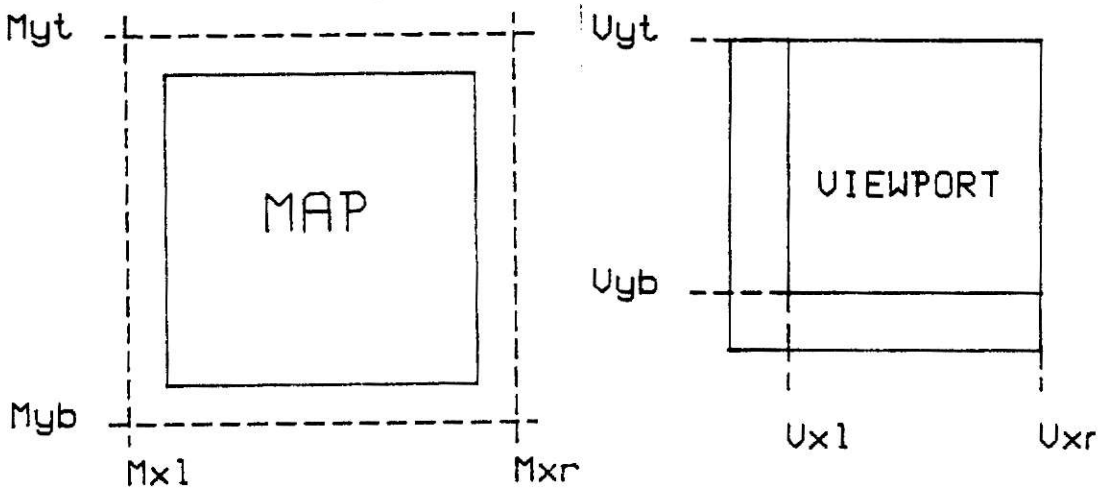


Figure 2. Screen Coordinates Transformation

Once the transformation to screen coordinates has been made the translation from 0, 0 to any point in the viewport is accomplished through the use of matrix transformation. The translation of the 0, 0 point to a new point is accomplished with the following equation [NEW79].

$$[X^1Y^1] = [XY1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix} \qquad (3)$$

As an example, consider the scrolling of the screen coordinates 25, 25 or the map coordinates 20, 30 being

translated 300 units to the right and 350 units up (T_x =300, T_y = 350). The new location in screen coordinates would be (X^1 = 325, Y^1 = 375). Once any desired point is translated then by any predetermined primitive can be drawn. If the scale of the viewport is to be changed the following matrix transformation is used:

$$[X^1Y11] = [XY1] \begin{bmatrix} S_x & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (4)$$

Therefore, the desired scale can be enlarged by changing S_x' S_y from 1 to the desired size. The translation and scaling equation can be concatenated into two simpler equations to maximize efficiency.

$$X^1 = S_xX + T_x^X \qquad (5)$$

$$Y^1 = S_yX + T_yX \qquad (6)$$

Because of such a large map database, a window of the whole map database with a pointer to the location showing on the lower left corner of the display would be needed. The overall picture viewport would only be displayed on command of the user.

The menu selection of the display should be able to be accessed through the keyboard or through use of variable menu buttons located around the peripheral of the display screen. The available menus could be selected by the push of a peripheral button or keyboard key and the selection of items from individual menus could also be selected by the use of keyboard or peripheral buttons. The positioning of the items on the map from the menu would be with the cursor

position. The cursor should be able to be moved by the joystick, cursor control, or the move command.

In either the overlay or the report mode, the system must have the capability of storing the overlays and reports in memory and recalling them at some later time for modification or update. The overlays and reports must be able to be transmitted to other displays or plotters.

The complexity of a C&C system is beyond the capability of a master's project therefore, only one portion of the system was undertaken. It was desirable to be able to build military symbols on a display and transmit the symbol to an overlay; therefore the following requirements were imposed on this project.

1. Develop an interactive graphical method for building military symbols.
2. Develop a method to transfer military symbols from a graphic display to an exact location an on overlay.
3. Select a high level language and program structure that will facilitate ease of modification and use on the Interdata 8/32 computer.

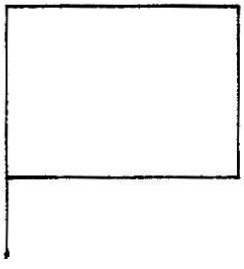
B. SPECIFICATIONS

1. Military Symbols

This project will incorporate the standard military symbols approved for use by the member nations of the North Atlantic Treaty Organization (NATO), the

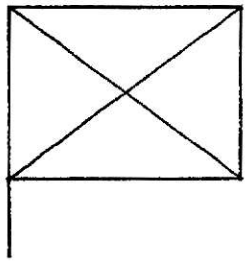
Central Treaty Organization (CENTO), and the Southeast Asia Treaty Organization (SEATO) [Mil70, Ope80]. Each military symbol will consist of the basic functional and size symbols. The basic symbol is a geometric figure used to represent units, installations and activities. Examples follow:

A Unit

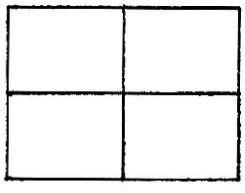


A Headquarters

The functional symbol is used to represent the branch. The functional symbol is placed within the basic symbol to show the type of unit, installation or activity as shown below:

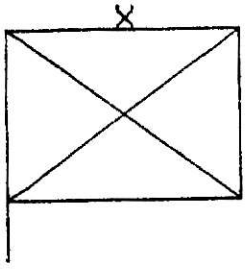


An Infantry Headquarters

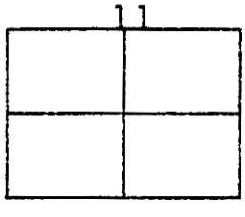


A Medical Unit

The size symbol shows the organizational strength of a specific unit. The size symbol is placed on top of the basic symbol as shown below:



An Infantry Brigade Headquarters



A Battalion Medical Unit

2. Map System

The requirement to transfer military symbols to a precise location and to satisfy the military requirements in identifying locations will require the use of the U.S. Military Grid Reference System. [Map64, Ope80] This system designates a precise location by the use of the military principles, Read RIGHT then UP. The precision desired determines the number of digits to be read beyond the principal digit. Therefore, the grid coordinates 0203 locate the 1000 meter grid square in which a precise point is located. The large A in Figure 3 represents section 0203.

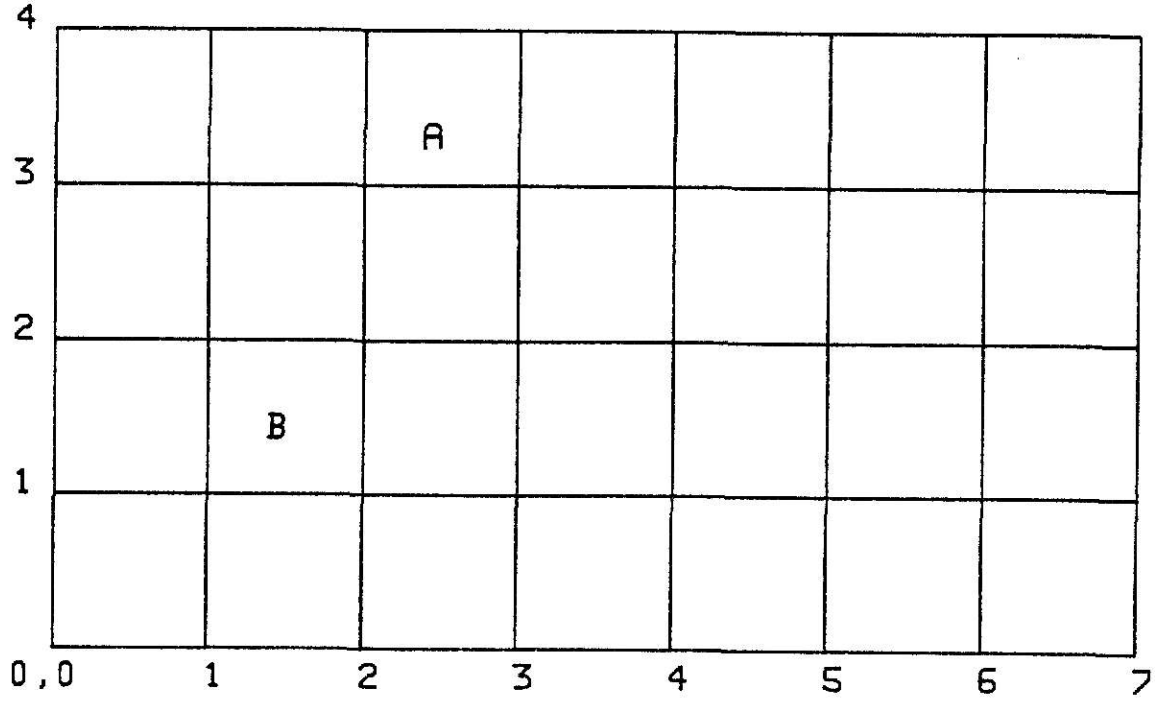


Figure 3. 1:50,000 map grid

The precision of a point location in this project will be to the nearest 100 meters. Therefore, both the X and Y axis require a three digit number. Grid coordinates are written as one continuous number without spaces. The grid coordinates 015015 would be located at the B on Figure 3.

The grid coordinates scale used for this project is 1:50,000. The scale of a map permits the determination of ground distance from the map. This numerical scale of a map expresses the ratio of one unit of measure on the map is equal to 50,000 of the same unit on the ground. The grid lines in Figure 3 are representative of a scale of 1:50,000.

III. SYSTEM DESIGN AND IMPLEMENTATION

The Military Interactive Symbology Design Package (MISDP) is written entirely in SPASCAL for the Interdata 8/32 computer. It is implemented using the Chromatics color graphic terminal for building the desired military symbol and it transfers the military symbol to an overlay using the Soltec plotter.

A. SYSTEM DESIGN

1. Chromatics Terminal

The Chromatic 1999 color graphic display terminal is the peripheral device used to build military symbols. The Chromatics 1999 is a self-contained, high resolution color graphic system with an integral Z-80 micro-processor and I/O memory. It is designed for use as a display element for a computer system or a stand alone system for data processing and display applications [Chr78]. The Kansas State University's Chromatics terminal will not support a high level structured language in the stand alone mode; therefore it was decided to use the Chromatic terminal as a display element in conjunction with the Interdata 8/32. The procedures needed to put the Chromatics into operating mode are covered in the User's Guide (Appendix A).

2. Soltec Plotter

The Soltec plotter is the output device used to produce the overlays. The Soltec 281 plotter is an intelligent digital plotter with an integral Z-80 microprocessor [Sol79]. The necessary commands to make the Soltec plotter functional are covered in the User's Guide (Appendix A).

3. Peripheral Drivers to Interdata 8/32

A significant portion of MISDP uses two drivers written by Theodore J. Socolofsky; therefore, a description of his work is in order. Socolofsky developed two drivers written in SPASCAL. Each driver interfaced separately with the Interdata 8/32 and each driver is capable of performing most of the manufacturer's functions as described in the User's Manual. However, due to the lack of documentation, the interactive procedures were extremely difficult to use and to understand. Both drivers make calls to the Interdata operating system through use of Supervisor Calls (SVC) to be able to communicate to the peripheral devices. Because of the lack of buffering, the

Chromatics communicates with the Interdata by individual bits as the data is typed on the Chromatics terminal. The difficulties encountered in using the drivers will be described in the implementation section of Chapter III.

4. Human Interface

This section details the steps that were taken in designing the input for the package being developed. The factors that affect the design of a human interface are many, complex and overlapping in scope. The methods used to analyze these factors were the author's experience and a literature search [Mcc80, Ram79, Bel80]. A successful human interface for a computer program must be friendly, easy to learn and natural. It should allow for easy detection and correction of errors [Fol74]. It should also allow easy changes in tasks so that the user can start one task, change to another and then return to the first with a minimum of effort.

The textual man-machine communications format should require little training to master and be relearnable almost immediately after a long period of non-use [Spe77].

B. IMPLEMENTATION

Both of the drivers for the Chromatics (Chrofix) and Soltec plotter (Plotfix) had to be reviewed and modified. A

significant portion of time had to be spent in testing and debugging each non-documented driver. Individual programs were constructed and implemented in the non-interactive mode and then interactive procedures were added. Once the two drivers were working independently, they were combined into one driver. The combination of both drivers required the modification of the Plotfix command primitive procedure names with an underscore character to distinguish the differences between Chrofix and Plotfix procedures with the same name.

1. Chromatics

The procedures to generate the primitive commands on the Chromatics use the identical names of the commands in the Chromatics Preliminary Operator's Manual [Chr78]. For example, the Chromatics command for rectangle requires the ASCII key stroke (:43:) or '+' to be struck and followed by two sets of coordinates representative of the bottom left and upper right corners, respectively. Procedure RECTANGLE in Chrofix accomplishes the same task with the following Procedure:

```
PROCEDURE RECTANGLE (X1,Y1,X2,Y2:INTEGER);

     Begin;
          OUTPUT ('+')
          OUTPUT_COORD (X1,Y1);
          OUTPUT_COORD (X2,Y2);
          SHIP_OUT;
     END
```

Procedure RECTANGLE makes calls to Procedures OUTPUT_COORD, OUTPUT and SHIP_OUT. Procedure OUTPUT places the ASCII character, (:43:) or '+', in the next available location in an array called OUTLINE. Next, Procedure OUTPUT_COORD accepts two integers, X1 and Y1, it makes two calls on OUTPUT_DEC (X1 AND Y1). The OUTPUT_DEC procedure converts the integer into a string of ASCII characters which are the character equivalent of the integer. Example: Integer 123 is converted to the character string '123'. Each character of the string is moved to the next empty space in the array OUTLINE by a call on the Procedure OUTPUT. The same above steps are repeated for the second set of coordinates, X2 and Y2. Once the required string has been created Procedure SHIP_OUT is used to transfer the string of characters to the Chromatics, where the Chromatics uses these characters as commands to draw a picture. A copy of Procedure SHIP_OUT follows:

```
PROCEDURE SHIP_OUT;
     BEGIN;
          SVC11_OUT.SVC1_BEFFEND : = SVC1_OUT;
          SVC1_BUFFSTART + OUTPUT_COUNTER-1;
          SVC1 (SVC1_OUT);
          OUTPUT_COUNTER: = 0;
     END
```

Procedure SHIP_OUT calculates the length (ending address) of the buffer that is being sent to the Chromatics (OUTLINE) by adding the starting address plus the length of the buffer (OUTPUT_COUNTER) minus 1 to adjust the address to its proper location. For example, assume starting address of

SVC1_BUFFSTART = 1000, and OUTPUT_COUNTER = 10. In order to transfer the address 1000 to 1009, one must be substracted from the starting address, SVC1_BUFFEND = 1000 + 10 - 1 = 1009.

The statement SVC1 (SVC1_OUT) makes a call on an external procedure which exists in a module called PASDRIVR (see explanation below in Procedures SVC1). It passes to the SVC a parameter block called SVC1_OUT. The counter OUTPUT_COUNTER is then zeroed out for future calls on building a new output SVC1.

Procedure SVC1 is an interface with the operating system. A module called PASDRIVR receives the SVC1_BLOCK passed as SV_OUT and generates an assembly language supervisor call number 1 (input or output) using the information in the SVC1_BLOCK passed to it. After the call has been generated, it is transferred to the operating system on behalf of the program by PASDRIVR. The program is halted until the supervisor call is finished and the ending status information is returned by the operating system to PASDRIVR which in return passed the status back to the program for further use if needed. When used as an input call along with the status the input characters are returned to the appropriate variable location in the program by the operating system and PASDRIVR.

2. Soltec Plotter

The procedures to generate the primitive commands to the plotter are given names which relay the function the procedure is performing. For example, the Procedure Rectangle_ shown below uses calls on Procedure PEN_UP, PEN_DOWN and MOVE_CURSOR_. The names explain what the procedures do.

```
PROCEDURE RECTANGLE_(X,Y1:INTEGER)
     BEGIN;
          PEN_UP;
          MOVE_CURSOR_(X1,Y1)
          PEN_DOWN
          MOVE_CURSOR_(X1,Y1+220)
          MOVE_CURSOR_(X1+300,Y1+220)
          MOVE_CURSOR_(X1+300,Y1)
          MOVE_CURSOR_(X1,Y1)
          PEN_UP
     END
```

Unlike the Chromatics, the plotter does not have any built in functions such as RECTANGLE. All functions must be built up using lower level primitives.

```
PROCEDURE PEN_UP;
     BEGIN PEN_IS_DOWN:=FALSE; OUTPUT_CHAR(P_U)END
```

The PEN_UP procedure first sets a global boolean caled PEN_IS_DOWN to false (for use if necessary to discover what state the pen is in, up or down). The procedure makes a call on OUTPUT_CHAR and passes the character 'H' which is shipped to the plotter. It is decoded in the command mode to move the pen up (page 11 in the Soltec User Manual, Part

II). PEN_DOWN is similar to PEN_UP, except the global boolean PEN_IS_DOWN is set to true and the character 'I' is passed to OUTPUT_CHAR which again ships it to the plotter to be decoded in the command mode as a pen down instruction (page 10, Soltec User Manual, Part II).

The procedure OUTPUT_CHAR increments a global integer variable called OUTPUT_COUNT by one. The counter is used as an index into a global array of characters called TEXT to signify the next empty position in the array for the character being passed into this procedure. If the counter is equal to 80 then a call is made to the prefix procedure WRITELINE passing the TEXT which contains 80 characters. The counter is reset to zero so that is ready for another 80 characters to be stored in TEXT. Prefix Procedure WRITELINE is a hard-coded assembly language SVC1 call in PASDRIVR which takes a line of text and writes it out to logical unit 2 (in this case PA42:, the Soltec Plotter).

Procedure MOVE_CURSOR_(X1,Y1:integer) receives as parameters an X coordinate and a Y coordinate with which it makes calls to OUTPUT_INT and OUTPUT_CHAR. This procedure calls OUTPUT_INT with the X coordinate. Then it calls OUTPUT_CHAR with the character '/' which the plotter recognizes as the coordinates separator. The Y coordinates is then output with OUTPUT_INT and followed by a call to OUTPUT_CHAR with the character 'K' which the plotter recognizes as a move absolute command.

Procedure OUTPUT_INT takes the integer passed to it and conveys it to its equivalent ASCII character and passes each character to the output array called TEXT by making a call on OUTPUT_CHAR for each digint converted. OUTPUT_INT always generates a four character representation of the integer passed to it. Examples follow:

123 is converted to '0123'.

1000 is converted to '1000'.

3. Interactive Follow Through of Drawing a Rectangle

The program is loaded and run with the following assignments:

```
LO MS
AS 0,CON:  The Chromatics must be run under the
           control of the multi-terminal monitor
           (MTM).
AS 2,PA42:
ST
```

The whole program has only one statement which is a call to the Procedure INITIALIZE. Procedure INITIALIZE performs the following statements: INIT_SVC (this sets up the output SVC1_BLOCK for use as output to the Chromatics); SET_SVC1_INPUT (this sets up the input SVC1 BLOCK for use as input to the Chromatics); OUTPUT_COUNTER:=0 (this zeros out the text counter for output to the plotter); HEIGHT_CHAR:=30,WIDTH_CHAR:=30 (this sets the height and width of the characters printed on the plotter to be 30 mm); OUTPUT_CHAR (P_ON) (this sends the character (:1:) to the plotter which causes the plotter to be logically turned on); USER_PROG (this is a procedure call to run the users portion

on the program, see explanation below); PEN_UP (see explanation under plotter); OUTPUT_CHAR(P_OFF) (this outputs a character (:3:) to the plotter which logically turns the plotter off); FLUSH_TEXT (if less than 80 characters have been put into TEXT for the plotter, the rest of TEXT is filled with nulls (:00:) and shipped to the plotter).

Procedure USER_PROG sets up the screen, sets a boolean FIRST_TIME_THRU to true and calls BUILD_SYMBOL_TEXT and passes FIRST_TIME_THRU to the procedure.

Procedure BUILD_SYMBOL_TEXT is the procedure which asks the interactive questions to build a military symbol. When the question is asked to select a unit symbol the program performs the statement INPUT(A_B) which accepts one and only one character from the console and puts it into a local variable A_B. The case where A_B='A' will be followed through the rest of the program. After A B is inputted, it is plotted on the working area of the screen (Fig. 3) and the next two questions are asked and answered to fill the three local variables, A_B, C_D,X. These three variables are passed to TRANSFER_SYMBOL_TEXT. When the XY coordinates have been inputted into local variables X_1 and Y_1 and the request to transfer the military smbol build in the working area has been received, the five local variables are passed to Procedure PLOT. PLOT consists of three case statements. Case A_B of 'A' calls Procedure A_PLOT with X_1 and Y_1

coordinates. Procedure A_PLOT calls Procedure Rectangle_ and draws a predetermined size rectangle and on the plotter at the desired XY coordinates.

4. Human Interface

a. Man-Machine Communication

The best summarization of the design constraints is in terms of complexity, vocabulary and prompting. The complexity of interactive textual interface was restricted to a narrow range. Three steps were used to facilitate building of the complete military symbol. Grid coordinates were broken down into two three-digit numbers and then combined and verified on the screen before transferring the military symbols to the overlay. The vocabulary was matched to the normal operators of command and control terminals. Finally, the prompting was adapted to ensure quick but easy operation.

All interactive textual interfaces will only accept input choices as displayed on the screen. If the user inputs anything but a three digit number for grid coordinate an error message will appear and will request that the user retype the grid coordinates. To facilitate speed in building the three step military symbol the user doesn't have to push RETURN; however, grid coordinate input requires the user to push RETURN to show the user the grid coordinates are correct.

b. Display

The Chromatics display was divided five physical windows (see Figure 4); however, logically only four windows could be used. The four windows were needed so that each window could be erased separately. The detailed explanation of the Chromatics display is in the User's Guide (Appendix A).

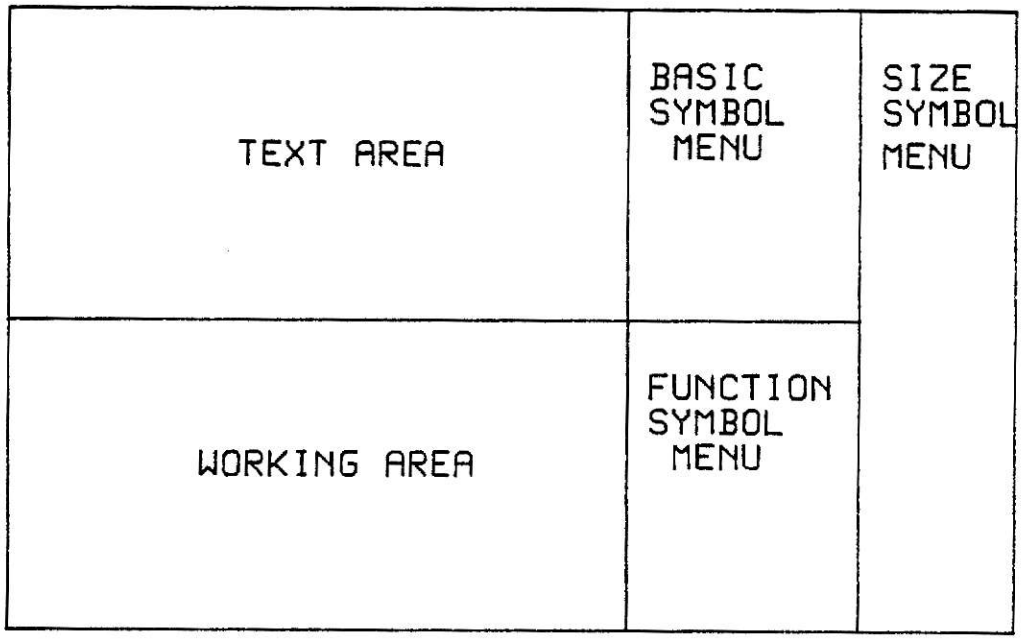


Figure 4. Chromatics Screen

IV. SUMMARY

A. CONCLUSIONS

MISDP as designed represents an effective computer aided tool in a military command and control system. It allows the user to quickly build a military symbol and transfer the location of the unit through the use of an overlay.

The program was tested by several students and every possible command was executed. The successful testing of this program clearly demonstrated that the objectives of this project have been met.

B. RECOMMENDATIONS

Although MISDP accomplished the objective of the project, there are still many areas where further improvement can be made. These areas are summarized below:

1. The MISDP program currently is designed only to build a limited number of military symbols. It would be desirable to increase the number of military symbols by increasing the number of primitives in each menu.
2. The map coordinate system used in MISDP only accepts six digit coordinates; therefore, the grid zone identification must be known before the overlay will correspond with the user's map.

3. Before the MISDP program would be a totally effective tool in a command and control operations the operational terms and graphics from FM 121-5-1 would have to be implemented into the package [Ope80].

BIBLIOGRAPHY

Agi78 A Guide in Using MTM at Kansas State University, Revision 2, Computer Science Department, Kansas State University, 1978.

Bel80 Beling, G. J. and Drake P. An Interactive Raster Graphics Terminal Application for Military Information Display, SHAPE Technical Report, pp. 174, July 1980.

Chr78 Chromatics Preliminary Operator's Manual (Revised), Chromatics, Inc., Atlanta, Ga., November 1978.

Dav77 Davis, Paul R. Interactive Computer Graphics, Command and Control Technical Center, Defense Communication Agency Technical Report, TR 116-77, September 1977.

Fol74 Foley, James D. "The Art of Natural Graphic Man-Machine Conversation," Proceedings of the IEEE, April 1974.

Lov80 Lovelock, R. L. SHEWS Display Interaction: An Example of High Speed Graphics in Command and Control, SHAPE Technical Report, STC pp. 175, May 1980.

Map64 Map Reading, FM 21-26, Department of the Army, January 1969.

Mcc80 McCann, Carol. Maps for Command and Control: The Human Factors Issues, Department of National Defense - Canada, DCIEM Report 80-R-06, January 1980.

Mil70 Military Symbols, FM 21-30, Department of the Army, May 1970.

New79 Newman, W. and Sproull, R. Principles of Interactive Computer Graphics, Second Edition, McGraw-Hill Book Co., 1979.

Ope80 Operational Terms and Graphics, FM 101-5-1, Department of the Army, March 1980.

Pro78 Programmer Reference Manual, OS/32, Perkin-Elmer, Oceanport, N.J., November 1978.

Ram79 Ramsey, H. and Atwood, Michael E. Human Factors in Computer Systems: A Review of the Literature, Science Applications, Inc., September 1979.

Sch80 Schmidt, W. H. P. Digital Television Graphics as a Means of Displaying Military Situations, SHAPE Technical Report, STC pp. 177, November 1980.

Spe77 Spence, Robert, and Apperley, Mark. "The Interactive-Graphic Man-Computer Dialogue in Computer-Aided Circuit Design," IEEE Transactions on Circuits and Systems, Fall 1977.

Sol79 Soltec Plotter 281 User's Manual, October 1979.

APPENDICES

APPENDIX A

USER'S GUIDE

A. GENERAL DESCRIPTION

MISDP is designed to allow the user to rapidly show military unit disposition at another location by use of an overlay. This is accomplished by using a Chromatics color graphic display terminal to draw the designed military symbol on the screen of the CRT. Once the military symbol has been depicted on the screen it can be transferred to an overlay by using a six digit grid coordinate. The transfer to an overlay is made on a Soltec plotter. The U.S. Army Military Grid Reference System is used for plotting the desired location using 1:50,000 scale.

B. OPERATING INSTRUCTIONS

Before the user can operate MISDP both the Chromatics and the Soltec must be placed in the proper operating and communications mode. Therefore, the first step is to contact the operations system operator to ensure that both the Chromatics terminal and the Soltec plotter are connected to the Interdata 8/32.

Once the Chromatics terminal has been logged on to the Interdata, the following sequence of buttons and keys must be activated.

1. Power — Push white button located above keyboard on right hand side of Chromatics. When the button has been activated, it will be illiminated.

2. RESET — Generates a hand-wired master clear signal.

3. BOOT — Causes all I/O devices to be initialized to default conditions.

4. CRTOS — Activates the Chromatics' operating system.

At the completion of pushing CRTOS two white parallel blank lines should be visible in the upper left hand corner of screen. This indicates that the Chromatics is operational.

5. ESC H — Press ESC and release and then press H. This causes the terminal to operate in half duplex. If full duplex is desired, press ESC F.

6. ESC R09 — This command causes the serial line to operate at 2400 baud. Ensure that 0, not zero is pressed.

At this time a white asterisk (*) should be visible in the upper left hand corner. This indicates that the Interdata 8/32 is ready for signon. It is assumed that the user knows the signon procedures and that the MISDP is compiled on the desired account number.

The Soltec plotter control panel must be set in the following sequence:

1. POWER - ON
2. CHART - HOLD
3. PEN - REMOTE
4. PLOT - REMOTE

If additional information is desired, refer to the Soltec plotter 281 user's manual.

If the above commands have been completed MISDP is now ready to be activated. The following 5 commands must be typed on the Chromatics terminal keyboard to run MISDP.

1. PROMPT — Turns off prompts to 8/32.
2. LO MISDP — Used to load task into the Chromatics user segment.
3. AS 0,CON: — Assigns the Chromatics terminal as the input device to 8/32.
4. AS 2,PA42: — Assigns the Soltec plotter as the output device.
5. ST — Start is used to initiate task execution

C. MISDP OPERATION

The command ST will activate MISDP and the Chromatics' display will be divided in 5 physical windows as shown in Figure 5.

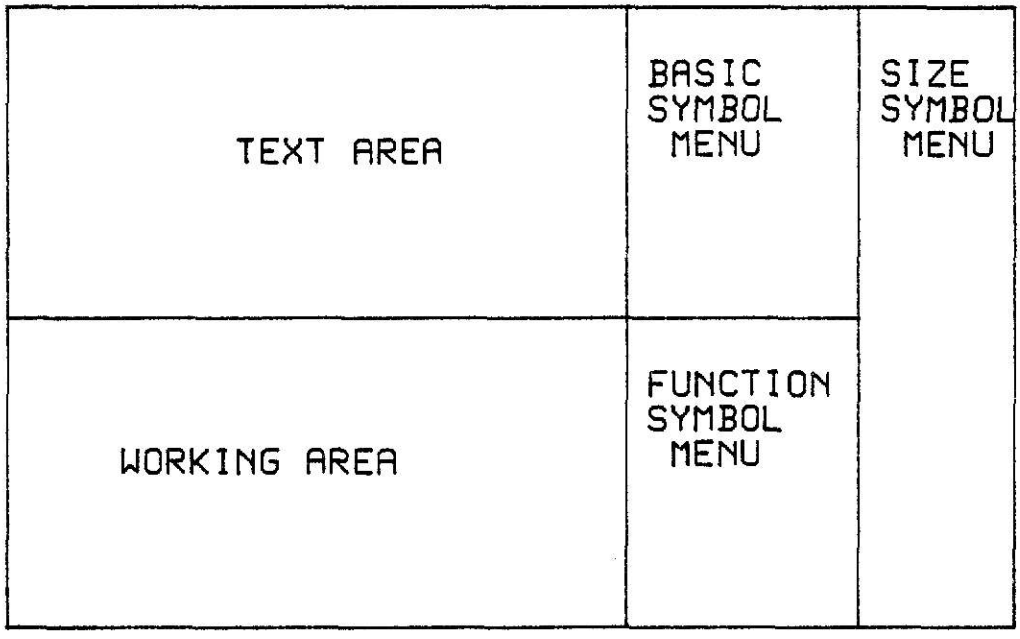


Figure 5. Chromatics Display Screen

The text area is used for interactive textual interface for all inputs into MISDP. The three menus are used to choose the desired features of the military symbol as it is developed in the working area. The first interactive question displayed is:

DO YOU WISH TO BUILD A MILITARY SYMBOL? TYPE YES (Y) OR NO (N)

If the user types N the program is exited and BYE appears on the display screen. When the user types Y, the text area is cleared and the next statement is displayed.

SELECT UNIT SYMBOL BY TYPING A OR B

The user must select from the basic unit menu either the standard unit symbol (A) or the headquarters symbol (B). Once the user selects A or B the symbol appears centered in the working area and the next statement is displayed.

SELECT FUNCTIONAL SYMBOL BY TYPING CORRESPONDING LETTER

The user must select from the functional menu the functional symbol for the infantry (C), medical corps (D) or the engineers (E). The selection of one of the above 3 functional symbols will cause the functional symbol to appear centered inside of the unit symbol and the next statement will appear in the text area.

SELECT UNIT SIZE BY TYPING CORRESPONDING NUMBER

The user must select from the size menu one of eight size designators from Army Group to Company. The selection of the size designation will cause the size designation to

appear centered and above the military symbol in the work area.

Once the symbol has been completely built in the work area, the text area will display the following question:

DO YOU WISH TO TRANSFER MILITARY SYMBOL TO OVERLAY
TYPE Y OR N

If the user selects N the text area will display the first question asked.

DO YOU WISH TO BUILD A MILITARY SYMBOL
TYPE YES (Y) OR NO (N)

The user may terminate the program by typing N; however, the inputing of Y will start the cycle to build a military symbol again. If the user wished to transfer the military symbol to an overlay, the next statement appears on the display:

TYPE 3 DIGIT X COORDINATE AND PUSH RETURN

The user must type in a three number sequence and push RETURN. This will cause the next statement to appear on the display.

TYPE 3 DIGIT Y COORDINATE AND PUSH RETURN

Again the user must type in 3 numbers representing the Y coordinate and push RETURN. The next statement is used to confirm the six digit coordinates to which the military symbol will be transferred:

ARE COORDINATES _____ CORRECT? TYPE (Y) YES OR (N) NO

If the user types N the sequence to transfer coordinates is repeated; however, if the user typed Y the military symbol is transferred to the overlay to the desired coordinates. The text area and work area are cleared and the next statement appears in the text area:

DO YOU WISH TO BUILD ANOTHER MILITARY SYMBOL? TYPE (Y)
YES OR (N) NO

If the user types Y the sequence to build another military symbol is repeated. If the user wishes to terminate the program he must type N.

APPENDIX B

Source Code

```
"ROBERT YOUNG"
"DEPARTMENT OF COMPUTER SCIENCE"
  CONST COPYRIGHT = 'COPYRIGHT ROBERT YOUNG 1978'

"//////////
#  PREFIX  #
##########"

; CONST NUL = '(:00:)'
  ; MODE = '(:01:)'
  ; X_BAR = '!'
  ; BS = '(:08:)'
  ; NAK = '(:15:)'
  ; Y_BAR = '"'
  ; NL = '(:10:)'
  ; A = 'A'
  ; INCR_X_BAR = '#'
  ; FF = '(:12:)'
  ; C = 'C'
  ; INCR_Y_BAR = '$'
  ; CR = '(:13:)'
  ; G = 'G'
  ; DOT_ = '%'
  ; CAN = '(:24:)'
  ; H = 'H'
  ; INCR_DOT = '&'
  ; EM = '(:25:)'
  ; M = 'M'
  ; VECT = '(:39:)'
  ; SUB = '(:26:)'
  ; N = 'N'
  ; CONCAT_VECTOR = '('
  ; ESC = '(:27:)'
  ; Q = 'Q'
  ; ARC = ')'
  ; FS = '(:28:)'
  ; U = 'U'
  ; CIRCLE_ = '*'
  ; GS = '(:29:)'
  ; V = 'V'
  ; RECT = '+'
  ; RS = '(:30:)'
  ; W = 'W'
  ; ZERO = '0'
  ; US = '(:31:)'
  ; X = 'X'
  ; ONE = '1'
  ; SP = '(:32:)'
  ; Y = 'Y'
  ; TWO = '2'
```

```
  ; DEL = '(:127:)'
  ; P_S = 'F'
  ; THREE = '3'
  ; P_U = 'H'
  ; P_D = 'I'
  ; FOUR = '4'
  ; M_A = 'K'
  ; M_R = 'J'
  ; FIVE = '5'
  ; D_L = 'L'
  ; D_C = 'Z'
  ; SIX = '6'
  ; D_S = '%'
  ; D_F = '#'
  ; SEVEN = '7'
  ; C_E = 'B'
  ; C_D = CR
  ; EIGHT = '8'
  ; P_C = '0'
  ; R_C = '/'
  ; NINE = '9'
  ; P_ON = '(:1:)'
  ; P_OFF = '(:3:)'
  ; SIGNS = [ '-' , '+' ]

; CONST PAGELENGTH = 512

; TYPE PAGE = ARRAY (. 1 .. PAGELENGTH .) OF CHAR

; CONST LINELENGTH = 132

; TYPE LINE = ARRAY (. 1 .. LINELENGTH .) OF CHAR

; CONST IDLENGTH = 12

; TYPE IDENTIFIER = ARRAY (. 1 .. IDLENGTH .) OF CHAR

; TYPE FILE = 1 .. 2

; TYPE FILEKIND
 = ( EMPTY , SCRATCH , ASCII , SEQCODE , CONCODE )

; TYPE FILEATTR
    = RECORD
        KIND : FILEKIND
      ; ADDR : INTEGER
      ; PROTECTED : BOOLEAN
      ; NOTUSED : ARRAY (. 1 .. 5 .) OF INTEGER
      END

; TYPE IODEVICE
    = ( TYPEDEVICE , DISKDEVICE , TAPEDEVICE , PRINTDEVICE
      , CARDDEVICE )
```

```
; TYPE IOOPERATION = ( INPUT , OUTPUT , MOVE , CONTROL )

; TYPE IOARG = ( WRITEEOF , REWIND , UPSPACE , BACKSPACE )

; TYPE IORESULT
    = ( COMPLETE , INTERVENTION , TRANSMISSION , FAILURE
       ;ENDFILE, ENDMEDIUM , STARTMEDIUM )

; TYPE IOPARAM
    = RECORD
        OPERATION : IOOPERATION
      ; STATUS : IORESULT
      ; ARG : IOARG
      END

; TYPE TASKKIND = ( INPUTTASK , JOBTASK , OUTPUTTASK )

; TYPE ARGTAG = ( NILTYPE , BOOLTYPE , INTTYPE
, IDTYPE , PTRTYPE )

; TYPE POINTER = @ BOOLEAN

; TYPE PASSPTR = @ PASSLINK

; TYPE PASSLINK
    = RECORD
        OPTIONS : SET OF CHAR
      ; FILLER1 : ARRAY (. 1 .. 7 .) OF INTEGER
      ; FILLER2 : BOOLEAN
      ; RESET_POINT : INTEGER
      ; FILLER3 : ARRAY (. 1 .. 11 .) OF POINTER
      END

; TYPE ARGTYPE
    = RECORD
        CASE TAG : ARGTAG
        OF NILTYPE , BOOLTYPE : ( BOOL : BOOLEAN )
        ; INTTYPE : ( INT : INTEGER )
        ; IDTYPE : ( ID : IDENTIFIER )
        ; PTRTYPE : ( PTR : PASSPTR )
      END

; CONST MAXARG = 10

; TYPE ARGLIST = ARRAY (. 1 .. MAXARG .) OF ARGTYPE

; TYPE ARGSEQ = ( INP , OUT )

; TYPE PROGRESULT
    = ( TERMINATED , OVERFLOW , POINTERERROR , RANGEERROR
      , VARIANTERROR , HEAPLIMIT , STACKLIMIT , CODELIMIT
      ,TIMELIMIT , CALLERROR )

; PROCEDURE READ ( VAR CH : CHAR )
```

```
; PROCEDURE WRITE ( CH : CHAR )

; PROCEDURE OPEN
 ( F : FILE ; ID : IDENTIFIER ; VAR FOUND : BOOLEAN )

; PROCEDURE CLOSE ( F : FILE )

; PROCEDURE GET
 ( F : FILE ; P : INTEGER ; VAR BLOCK : UNIV PAGE )

; PROCEDURE PUT
 ( F : FILE ; P : INTEGER ; VAR BLOCK : UNIV PAGE )

; FUNCTION LENGTH ( F : FILE ) : INTEGER

; PROCEDURE MARK ( VAR TOP : INTEGER )

; PROCEDURE RELEASE ( TOP : INTEGER )

; PROCEDURE IDENTIFY ( HEADER : LINE )

; PROCEDURE ACCEPT ( VAR CH : CHAR )

; PROCEDURE DISPLAY ( CH : CHAR )

; PROCEDURE READPAGE
 ( VAR BLOCK : UNIV PAGE ; VAR EOF : BOOLEAN )

; PROCEDURE WRITEPAGE ( BLOCK : UNIV PAGE ; EOF : BOOLEAN )

; PROCEDURE READLINE ( VAR TEXT : UNIV LINE )

; PROCEDURE WRITELINE ( TEXT : UNIV LINE )

; PROCEDURE RUN
    ( ID : IDENTIFIER
    ; VAR PARAM : ARGLIST
    ; VAR LINE : INTEGER
    ; VAR RESULT : PROGRESULT
    )

; PROGRAM P ( PARAM : LINE )

"*************************************************************
          OS/32MT3 SVC INTERFACE ROUTINE TYPES
*************************************************************"
"MISCELLANEOUS DATA TYPES"

  ; TYPE CHAR1 = PACKED ARRAY [ 1 .. 1 ] OF CHAR

  ; TYPE CHAR3 = PACKED ARRAY [ 1 .. 3 ] OF CHAR

  ; TYPE CHAR8 = PACKED ARRAY [ 1 .. 8 ] OF CHAR
```

```
  ; TYPE CHAR4 = PACKED ARRAY [ 1 .. 4 ] OF CHAR

  ; TYPE CHAR16 = ARRAY [ 1 .. 16 ] OF CHAR

  ; TYPE CHAR28 = ARRAY [ 1 .. 28 ] OF CHAR

"SVC1 PARAMETER BLOCK"

  ; TYPE SVC1_BLOCK
      = RECORD
          SVC1_FUNC : BYTE "FUNCTION CODE"
        ; SVC1_LU : BYTE "LOGICAL UNIT NUMBER"
        ; SVC1_STAT : BYTE "DEV-INDEP STATUS"
        ; SVC1_DEV_STAT : BYTE "DEV-DEPENDENT STATUS"
        ; SVC1_BUFSTART : INTEGER "ADDRESS(BUFFER)"
        ; SVC1_BUFEND : INTEGER
         "ADDRESS(BUFFER)+SIZE(BUFFER)-1"
        ; SVC1_RANDOM_ADDR : INTEGER
         "RANDOM ADDRESS FOR DASD"
        ; SVC1_XFER_LEN : INTEGER "TRANSFER LENGTH"
        ; SVC1_RESERVED : INTEGER "RESERVED FOR ITAM USE"
        END

"SVC 1 FUNCTION CODES"

  ; CONST SVC1_DATA_XFER = #00
    ; SVC1_COMMAND = #80
    ; SVC1_READ = #40
    ; SVC1_WRITE = #20
    ; SVC1_TESTSET = #60
    ; SVC1_TESTIO = #00
    ; SVC1_ASCII = #00
    ; SVC1_BINARY = #10
    ; SVC1_PROCEED = #00
    ; SVC1_WAIT = #08
    ; SVC1_SEQL = #00
    ; SVC1_RANDOM = #04
    ; SVC1_CWAIT = #00
    ; SVC1_UNC_PROC = #02
    ; SVC1_FORMAT = #00
    ; SVC1_IMAGE = #01
    ; SVC1_REW = #40
    ; SVC1_BSR = #20
    ; SVC1_FSR = #10
    ; SVC1_WFM = #08
    ; SVC1_FSF = #04
    ; SVC1_BSF = #02
    ; SVC1_RESV_FN = #01

"SVC 1 DEVICE-INDEPENDENT STATUS CODES"

  ; CONST SVC1_OK = #00
    ; SVC1_ERROR = #80
```

```
    ; SVC1_ILGFN = #40
    ; SVC1_DU = #20
    ; SVC1_EOM = #10
    ; SVC1_EOF = #08
    ; SVC1_UNRV = #04
    ; SVC1_RECV = #02
    ; SVC1_ILGLU = #01
    ; SVC1_DEVBUSY = #7F

"FILE DESCRIPTOR FOR SVC7 REQUESTS"

  ; TYPE FD_TYPE
      = PACKED RECORD
                  VOLN : CHAR4 "VOLUME NAME"
                ; FN : CHAR8 "FILE NAME"
                ; EXTN : CHAR3 "EXTENSION"
                ; ACCT : CHAR "ACCOUNT NUMBER CODE"
                ;
                END

"SVC 7 PARAMETER BLOCK"

  ; TYPE SVC7_BLOCK
      = RECORD
          SVC7_CMD : BYTE "COMMAND"
        ; SVC7_MOD : BYTE "MODIFIER/DEVICE TYPE"
        ; SVC7_STAT : BYTE "STATUS"
        ; SVC7_LU : BYTE "LOGICAL UNIT NUMBER"
        ; SVC7_KEYS : SHORTINTEGER "READ/WRITE KEYS"
        ; SVC7_RECLEN : SHORTINTEGER "LOGICAL RECORD LENGTH"
        ; SVC7_FD : FD_TYPE "FILE DESCRIPTOR"
        ; SVC7_SIZE : INTEGER "FILE(/INDEX) SIZE"
        ;
        END

"SVC 7 COMMAND CODES"

  ; CONST SVC7_ALLOC = #80
    ; SVC7_ASSIGN = #40
    ; SVC7_CHAP = #20
    ; SVC7_RENAME = #10
    ; SVC7_REPROT = #08
    ; SVC7_CLOSE = #04
    ; SVC7_DELETE = #02
    ; SVC7_CHECKPT = #01
    ; SVC7_FETCH_ATTR = #00

"SVC 7 MODIFIER CODES - ACCESS PRIVILEGES"

  ; CONST SVC7_AP_SRO = #00
    ; SVC7_AP_ERO = #20
    ; SVC7_AP_SWO = #40
    ; SVC7_AP_EWO = #60
    ; SVC7_AP_SRW = #80
```

```
    ; SVC7_AP_SREW = #A0
    ; SVC7_AP_ERSW = #C0
    ; SVC7_AP_ERW = #E0

"SVC 7 MODIFIER CODES - BUFFERING/FILE TYPE"

  ; CONST SVC7_BUF_DEFAULT = #00
    ; SVC7_BUF_PHYS = #08
    ; SVC7_BUF_LOG = #10
    ; SVC7_BUF_SVC15 = #18
    ; SVC7_FTYPE_CONTIG = #00
    ; SVC7_FTYPE_CHAIN = #01
    ; SVC7_FTYPE_INDEX = #02
    ; SVC7_FTYPE_ITAM = #07


  ; TYPE COMMAND_TYPE
      = ( S_MOVE_CURSOR , S_X_BAR , S_Y_BAR , S_INCR_X_BAR
        , S_INCR_Y_BAR , S_DOT , S_INCR_DOT , S_VECTOR
        , S_CONCAT_VECTOR , S_ARC , S_CIRCLE , S_RECTANGLE ,
        CHARACTER)

  ; TYPE CHAR_STRING = ARRAY [ 1 .. 80 ] OF CHAR

  ; CONST BLACK = 0
    ; BLUE = 1
    ; GREEN = 2
    ; CYAN = 3
    ; RED = 4
    ; MAGENTA = 5
    ; YELLOW = 6
    ; WHITE = 7
    ; ORANGE = 8
    ; PURPLE = 9

  ; VAR ALINE : LINE
    ; SVC1_IN : SVC1_BLOCK
    ; J , TEMPX , TEMPY , COUNT : INTEGER
    ; OUTLINE : ARRAY [ 1 .. 132 ] OF CHAR
    ; OUTPUT_COUNTER : INTEGER
    ; C : CHAR
    ; X_1 , Y_1 , X , Y , X1 , Y1 , X2 , Y2,RADIUS :INTEGER
    ; COLOR_NUM : INTEGER
    ; SVC1_OUT : SVC1_BLOCK
    ; OUTPUT_COUNT , X3 , X_C , Y_C : INTEGER
    ; TEXT , COORD : LINE
    ; COMMAND : COMMAND_TYPE
    ; WIDTH_CHAR , HEIGHT_CHAR , V_OR_H_CHAR : INTEGER
    ; BAD_INTEGER , PEN_IS_DOWN , PLOT_ON_ , SET_BACK_ON
      , FIRST_I_X_BAR
      : BOOLEAN
    ; FIRST_I_Y_BAR , FIRST_I_DOT , FIRST_I_VECTOR : BOOLEAN
    ; CH , CHA , X_OR_Y : CHAR
```

```
"****************** USEFULL PROCEDURES **********************"

  ; PROCEDURE SVC1 ( SV_OUT : SVC1_BLOCK )
    ; EXTERN

  ; PROCEDURE SET_SVC1_INPUT

    ; BEGIN
        WITH SVC1_IN
        DO BEGIN
          SVC1_FUNC := SVC1_READ + SVC1_WAIT + SVC1_IMAGE
        ; SVC1_LU := 0
        ; SVC1_STAT := 0
        ; SVC1_DEV_STAT := 0
        ; SVC1_BUFSTART := ADDRESS ( C )
        ; SVC1_BUFEND := ADDRESS ( C ) + 1 - 1
        END
      END

  ; PROCEDURE INPUT ( VAR CURRENT_CHAR : CHAR )

    ; BEGIN
        SVC1 ( SVC1_IN )
      ; CURRENT_CHAR := CHR ( ORD ( C ) MOD 128 )
      END

  ; PROCEDURE INIT_SVC

    ; BEGIN
        WITH SVC1_OUT
        DO BEGIN
        ; SVC1_FUNC := SVC1_WRITE + SVC1_IMAGE
        ; SVC1_LU := 0
        ; SVC1_STAT := 0
        ; SVC1_DEV_STAT := 0
        ; SVC1_BUFSTART := ADDRESS ( OUTLINE )
        END
      ; OUTPUT_COUNTER := 0
      END

  ; PROCEDURE SHIP_OUT

    ; BEGIN
        SVC1_OUT . SVC1_BUFEND
        := SVC1_OUT . SVC1_BUFSTART + OUTPUT_COUNTER - 1
      ; SVC1 ( SVC1_OUT )
      ; OUTPUT_COUNTER := 0
      END

  ; PROCEDURE OUTPUT ( C : CHAR )

    ; BEGIN
        OUTPUT_COUNTER := SUCC ( OUTPUT_COUNTER )
      ; OUTLINE [ OUTPUT_COUNTER ] := C
```

```
      END

  ; PROCEDURE OUTPUT_LINE ( ALINE : LINE )
    ; FORWARD

  ; PROCEDURE READ_INT ( VAR INT : INTEGER )

    ; VAR C : CHAR
      ; STRING_INT : ARRAY [ 1 .. 12 ] OF CHAR
      ; START_INT , END_INT , POWER_OF_10 , I,SIGN: INATEGER
      ; INPUT_SYMBOLS : SET OF CHAR

    ; BEGIN
        INPUT_SYMBOLS
        := [ '0' , '1' , '2' , '3' , '4' , '5' , '6' , '7'
         , '8' , '9'
           , '-' , '+' , ' ' ]
      ; INPUT ( C )
      ; WHILE ( C = ' ' ) OR ( C = NL ) DO INPUT ( C )
"*****************************************************************
* READ INA VALID CHARACTER REPRESENTATION OF THE NUMBER    *
*****************************************************************"
      ; I := 0
      ; REPEAT IF C IN INPUT_SYMBOLS
          THEN BEGIN
            I := I + 1
          ; STRING_INT [ I ] := C
          ; INPUT ( C )
          END
          ELSE BEGIN
            OUTPUT_LINE
            ( 'INVALID SYMBOL IN INPUT(:10:) (:13:)$' )
          ; WHILE ( C <> ' ' ) & ( C <> NL ) DO INPUT ( C )
          ; WHILE ( C = ' ' ) OR ( C = NL ) DO INPUT ( C )
          END "IF"
        UNTIL ( C = ' ' ) OR ( C = NL ) OR ( C = CR )
"*****************************************************************
* DETERMINE THE SIGN OF THE INTEGER AND MARK THE START AND*
* END OF THE INTEGER.                                      *
*****************************************************************"
      ; END_INT := I
      ; IF STRING_INT [ 1 ] = '-'
        THEN BEGIN SIGN := - 1 ; START_INT := 2 ; END
        ELSE BEGIN
          SIGN := + 1
        ; IF STRING_INT [ 1 ] = '+'
          THEN START_INT := 2
          ELSE START_INT := 1
        END
"*****************************************************************
* CONVERT THE CHARACTER REPRESENTATION OF THE NUMBER TO IT
* NUMERIC REPRESENTATION
*****************************************************************
"
```

```
    ; POWER_OF_10 := 1
    ; FOR I := START_INT TO END_INT - 1
      DO POWER_OF_10 := POWER_OF_10 * 10
    ; INT := 0
    ; FOR I := START_INT TO END_INT
      DO BEGIN
        INT := INT
      ; POWER_OF_10
        := POWER_OF_10 DIV 10
           + ( ORD ( STRING_INT [ I ] ) - 48 )
           * POWER_OF_10
      END "FOR"
    ; INT := INT * SIGN
    END "READ_INT"

; PROCEDURE DISPLAY_ ( MSG : CHAR_STRING )

  ; VAR I : INTEGER

  ; BEGIN
      I := 0
    ; REPEAT I := I + 1
      ; DISPLAY ( MSG [ I ] )
      UNTIL MSG [ I ] = '$'
    ; DISPLAY ( NL )
    END

; PROCEDURE DISPLAY_LINE ( ALINE : LINE )

  ; VAR I : INTEGER

  ; BEGIN
      I := 1
    ; WHILE ( ALINE [ 1 ] <> NL ) OR ( I > 80 )
      DO BEGIN
        DISPLAY ( ALINE [ I ] )
      ; I := SUCC ( I )
      END
    ; DISPLAY ( NL )
    END

; PROCEDURE ACCEPT_LINE
( VAR ALINE : LINE ; VAR I : INTEGER )

  ; VAR C : CHAR

  ; BEGIN
      I := 0
    ; REPEAT BEGIN
          I := SUCC ( I )
        ; ACCEPT ( C )
        ; ALINE [ I ] := C
        END
      UNTIL ( ALINE [ I ] = NL ) OR ( I = 80 )
```

```
      ; IF ALINE [ I ] <> NL
        THEN BEGIN
          IF I = 80
          THEN ALINE [ 80 ] := NL
          ELSE ALINE [ SUCC ( I ) ] := NL
        END
      END

" OUTPUT THE 3 ASCII CHAR THAT ARE THE DECIMAL VALUE OF THE
  INTEGER"

  ; PROCEDURE DISPLAY_DEC ( I : INTEGER )

    ; VAR I100 : INTEGER

    ; BEGIN
      ; I100 := I DIV 100
        "CALCULATE THE HUNDREDS PLACE."
      ; DISPLAY ( CHR ( I100 + 48 ) )
        "DISPLAY THE HUNDREDS PLACE."
      ; DISPLAY ( CHR ( ( ( I - I100 * 100 ) DIV 10 ) + 48))
"DISPLAY THE TENS PLACE."
      ; DISPLAY
          ( CHR ( ( I MOD 10 ) + 48 ) )
          "DISPLAY THE ONES PLACE."
      END

  ; PROCEDURE OUTPUT_DEC ( I : INTEGER )

    ; VAR I100 : INTEGER

    ; BEGIN
      ; I100 := I DIV 100 "CALCULATE THE HUNDREDS PLACE."
      ; OUTPUT ( CHR ( I100 + 48 ) )
               "OUTPUT THE HUNDREDS PLACE."
      ; OUTPUT ( CHR ( ( ( I - I100 * 100 ) DIV 10 ) + 48 ))
"DISPLAY THE TENS PLACE."
      ; OUTPUT ( CHR ( ( I MOD 10 ) + 48 ) )
         "DISPLAY THE ONES PLACE."
      END

  ; PROCEDURE OUTPUT_HEX ( I : INTEGER )

" OUTPUT THE VALUE OF I AS TWO ASCII HEX DIGITS."

    ; VAR J , K : INTEGER

    ; BEGIN
      ; J := ( I DIV 16 ) + 48
      ; IF J > 57
        THEN J := J + 8
      ; OUTPUT ( CHR ( J ) )
      ; K := I - J * 16 + 48
      ; IF K > 57
```

```
      THEN K := K + 8
    ; OUTPUT ( CHR ( K ) )
    END

; PROCEDURE DISPLAY_HEX ( I : INTEGER )

" OUTPUT THE VALUE OF I AS TWO ASCII HEX DIGITS."

  ; VAR J , K : INTEGER

  ; BEGIN
    ; J := ( I DIV 16 ) + 48
    ; IF J > 57
      THEN J := J + 8
    ; DISPLAY ( CHR ( J ) )
    ; K := I - J * 16 + 48
    ; IF K > 57
      THEN K := K + 8
    ; DISPLAY ( CHR ( K ) )
    END

; FUNCTION DCOS ( RADIANS : REAL ) : REAL
  "CALLED FROM THE"
  ; FORTRAN "LIB"

; FUNCTION DSIN ( RADIANS : REAL ) : REAL
  "CALLED FROM THE"
  ; FORTRAN "LIB"

; PROCEDURE GET_INTEGER ( TIMES : INTEGER )

  ; BEGIN END

; PROCEDURE CONVERT ( VAR J , COUNT : INTEGER )

  ; VAR I , N : INTEGER
    ; CH : CHAR

  ; BEGIN
      I := 0
    ; N := 1
    ; CH := COORD [ N ]
    ; WHILE N <= J
      DO BEGIN
        I := I * 10 + ( ORD ( CH ) - ORD ( '0' ) )
      ; N := SUCC ( N )
      ; CH := COORD [ N ]
      END
    ; IF COUNT = 1
      THEN X1 := I * 66 DIV 10
    ; IF COUNT = 2
      THEN Y1 := I * 46 DIV 10
    ; IF COUNT = 3
      THEN X2 := I * 66 DIV 10
```

```
      ; IF COUNT = 4
        THEN Y2 := I * 46 DIV 10
      ; IF COUNT = 5
        THEN BEGIN Y2 := Y2 * 10 DIV 46 ; X3 := I END
      END

  ; PROCEDURE OUTPUT_CHAR ( CH : CHAR ) "IMAGE MODE OUTPUT"

"GLOBAL VAR OUTPUT_COUNT:INTEGER ; TEXT:LINE "

    ; BEGIN
      ; OUTPUT_COUNT := SUCC ( OUTPUT_COUNT )
      ; TEXT [ OUTPUT_COUNT ] := CH
      ; IF OUTPUT_COUNT = 80
        THEN BEGIN WRITELINE ( TEXT ) ; OUTPUT_COUNT:=0 END
      END

  ; PROCEDURE FLUSH_TEXT

    ; BEGIN
        FOR OUTPUT_COUNT := OUTPUT_COUNT + 1 TO 80
        DO TEXT [ OUTPUT_COUNT ] := '(:00:)'
      ; WRITELINE ( TEXT )
      ; OUTPUT_COUNT := 0
      END

  ; PROCEDURE OUTPUT_INT ( I : INTEGER )
    ; FORWARD

  ; PROCEDURE OUTPUT_LINE_ ( STRING : LINE )

    ; VAR INDEX : INTEGER

    ; BEGIN "TERMINATE ON NL,'$' OR 80 BUT DO NOT SHIP NL"
        INDEX := 1
      ; OUTPUT_CHAR ( D_C )
      ; OUTPUT_INT ( HEIGHT_CHAR )
      ; OUTPUT_CHAR ( SP )
      ; OUTPUT_INT ( V_OR_H_CHAR )
      ; OUTPUT_CHAR ( SP )
      ; OUTPUT_INT ( WIDTH_CHAR )
      ; OUTPUT_CHAR ( SP )
      ; OUTPUT_CHAR ( C_E )
      ; WHILE ( STRING [ INDEX ] <> NL )
              AND ( INDEX < 81 )
              AND ( STRING [ INDEX ] <> '$' )
        DO BEGIN
          OUTPUT_CHAR ( STRING [ INDEX ] )
        ; INDEX := SUCC ( INDEX )
        END
      ; OUTPUT_CHAR ( C_D )
      ; FLUSH_TEXT
      END
```

```
; PROCEDURE OUTPUT_INT

  ; VAR I1000 , I100 , I10 : INTEGER

  ; BEGIN
      I1000 := I DIV 1000
    ; OUTPUT_CHAR ( CHR ( I1000 + 48 ) )
    ; I100 := ( I - I1000 * 1000 ) DIV 100
    ; OUTPUT_CHAR ( CHR ( I100 + 48 ) )
    ; I10 := ( I - I1000 * 1000 - I100 * 100 ) DIV 10
    ; OUTPUT_CHAR ( CHR ( I10 + 48 ) )
    ; OUTPUT_CHAR ( CHR ( ( I MOD 10 ) + 48 ) )
    END

; PROCEDURE PEN_DOWN

  ; BEGIN PEN_IS_DOWN := TRUE ; OUTPUT_CHAR ( P_D ) END

; PROCEDURE PEN_UP

  ; BEGIN PEN_IS_DOWN := FALSE ; OUTPUT_CHAR ( P_U ) END

; PROCEDURE SET_COLOR

  ; BEGIN
      READ ( CHA )
    ; OUTPUT_CHAR ( P_S )
    ; OUTPUT_CHAR ( CHR ( ORD ( CHA ) + 1 ) )
    ; OUTPUT_CHAR ( SP )
    ; FLUSH_TEXT
    END

; PROCEDURE SET_H_D_W_CHAR

  ; BEGIN
      OUTPUT_CHAR ( D_S )
    ; OUTPUT_INT ( HEIGHT_CHAR )
    ; OUTPUT_CHAR ( SP )
    ; OUTPUT_INT ( V_OR_H_CHAR )
    ; OUTPUT_CHAR ( SP )
    ; OUTPUT_INT ( WIDTH_CHAR )
    ; OUTPUT_CHAR ( SP )
    END

; PROCEDURE MOVE_CURSOR_ ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      OUTPUT_INT ( X_1 )
    ; OUTPUT_CHAR ( R_C )
    ; OUTPUT_INT ( Y_1 )
    ; OUTPUT_CHAR ( M_A )
    END

; PROCEDURE CAL_LENGTH_STRING
```

```
( STRING : LINE ; VAR I : INTEGER )

   ; BEGIN
       I := 1
     ; WHILE ( STRING [ I ] <> NL )
             AND ( I < 80 )
             AND ( STRING [ I ] <> '$' )
       DO I := SUCC ( I )
     ; I := PRED ( I )
     END

 ; PROCEDURE START_STRING_AT
     ( STRING : LINE ; X_START , Y_START : INTEGER )

   ; BEGIN
       MOVE_CURSOR_ ( X_START , Y_START )
     ; OUTPUT_LINE_ ( STRING )
     END

 ; PROCEDURE END_STRING_AT
     ( STRING : LINE ; END_X , END_Y : INTEGER )

   ; VAR I , X_START : INTEGER

   ; BEGIN
       CAL_LENGTH_STRING ( STRING , I )
     ; X_START := END_X - I * WIDTH_CHAR
     ; START_STRING_AT ( STRING , X_START , END_Y )
     END

 ; PROCEDURE CENTER_STRING_IN
     ( STRING : LINE ; XA , YA , XB , YB : INTEGER )

   ; VAR I , X_START , Y_START : INTEGER

   ; BEGIN
       CAL_LENGTH_STRING ( STRING , I )
     ; IF ( XB = XA )
       THEN WIDTH_CHAR := 30
       ELSE WIDTH_CHAR :=
        ROUND ( CONV ( XB - XA ) / CONV ( I + 4 ) )
     ; HEIGHT_CHAR := WIDTH_CHAR
     ; X_START
       := ROUND ( CONV ( XB - XA - ( I - 3 )
       * WIDTH_CHAR ) / 2.0 )
          + XA
     ; Y_START :=
       ROUND ( CONV ( YB - YA - HEIGHT_CHAR ) / 2.0 ) + YA
     ; START_STRING_AT ( STRING , X_START , Y_START )
     END

 ; PROCEDURE CENTER_STRING_AT
( STRING : LINE ; X_1 , Y_1 : INTEGER )
```

```
    ; BEGIN
     CENTER_STRING_IN ( STRING , X_1 , Y_1 , X_1 , Y_1 ) END

  ; PROCEDURE DOT_ ( X_1 , Y_1 : INTEGER )

    ; BEGIN MOVE_CURSOR_ ( X_1 , Y_1 )
    ; PEN_DOWN ; PEN_UP END

  ; PROCEDURE VECTOR_ ( X1 , Y1 , X2 , Y2 : INTEGER )

    ; BEGIN
        PEN_UP
      ; MOVE_CURSOR_ ( X1 , Y1 )
      ; PEN_DOWN
      ; MOVE_CURSOR_ ( X2 , Y2 )
      ; PEN_UP
      ; FLUSH_TEXT
      END

  ; PROCEDURE RECTANGLE_ ( X1 , Y1 : INTEGER )

    ; BEGIN
        PEN_UP
      ; MOVE_CURSOR_ ( X1 , Y1 )
      ; PEN_DOWN
      ; MOVE_CURSOR_ ( X1 , Y1 + 220 )
      ; MOVE_CURSOR_ ( X1 + 300 , Y1 + 220 )
      ; MOVE_CURSOR_ ( X1 + 300 , Y1 )
      ; MOVE_CURSOR_ ( X1 , Y1 )
      ; PEN_UP
      END

  ; PROCEDURE CIRCLE_ ( X_1 , Y_1 , R , B_D , E_D : INTEGER)

    ; VAR X_B , Y_B : INTEGER

    ; BEGIN
        X_B := ROUND ( DCOS ( CONV ( B_D ) ) * CONV ( R ) )
      ; Y_B := ROUND ( DSIN ( CONV ( B_D ) ) * CONV ( R ) )
      ; PEN_UP
      ; MOVE_CURSOR_ ( X_1 + X_B , Y_1 + Y_B )
      ; PEN_DOWN
      ; OUTPUT_CHAR ( P_C )
      ; OUTPUT_INT ( R )
      ; OUTPUT_CHAR ( SP )
      ; OUTPUT_INT ( B_D )
      ; OUTPUT_CHAR ( SP )
      ; OUTPUT_INT ( E_D )
      ; OUTPUT_CHAR ( SP )
      ; PEN_UP
      END

" DISPLAY THE COORDINATE"
```

```
  ; PROCEDURE DISPLAY_COORD ( X , Y : INTEGER )
    ; BEGIN ; DISPLAY_DEC ( X ) ; DISPLAY_DEC ( Y ) END
  ; PROCEDURE OUTPUT_COORD ( X , Y : INTEGER )
    ; BEGIN OUTPUT_DEC ( X ) ; OUTPUT_DEC ( Y ) END
  ; PROCEDURE MODE "SEND THE MODE CHARACTOR. "
    ; BEGIN OUTPUT ( CHR ( 1 ) ) END
  ; PROCEDURE ESC "SEND THE ESCAPE CHARACTOR. "
    ; BEGIN OUTPUT ( CHR ( 27 ) ) END
  ; PROCEDURE BELL "SEND A BELL"
    ; BEGIN OUTPUT ( CHR ( 7 ) ) ; SHIP_OUT END
"*****************************************************************
 *
 * THE FOLLOWING PROCEDURES ARE DIRECT COPIES FROM THE
 * OPERATORS MANUAL.
 *
 *****************************************************************"
  ; PROCEDURE BOOT "2.4.1"
    ; BEGIN ESC ; OUTPUT ( 'G' ) ; SHIP_OUT END
  ; PROCEDURE CRTOS "2.4.1"
    ; BEGIN ESC ; OUTPUT ( 'T' ) ; SHIP_OUT END
  ; PROCEDURE CPUOS "2.4.2"
    ; BEGIN ESC ; OUTPUT ( 'Z' ) ; SHIP_OUT END
  ; PROCEDURE BASIC "2.4.3"
    ; BEGIN ESC ; OUTPUT ( 'B' ) ; SHIP_OUT END
  ; PROCEDURE RE_ENTER "2.4.3"
    ; BEGIN ESC ; OUTPUT ( 'E' ) ; SHIP_OUT END
  ; PROCEDURE DISK_OS "2.4.4"
    ; BEGIN ESC ; OUTPUT ( 'D' ) ; SHIP_OUT END
  ; PROCEDURE TEXT_EDIT "2.4.5"
    ; BEGIN ESC ; OUTPUT ( 'X' ) ; SHIP_OUT END
```

```
  ; PROCEDURE ASMB "2.4.6"

    ; BEGIN ESC ; OUTPUT ( 'A' ) ; SHIP_OUT END

  ; PROCEDURE FUNCTION_KEY ( I : INTEGER ) "2.4.8"

    ; BEGIN
        ESC
      ; OUTPUT ( 'J' )
      ; OUTPUT ( CHR ( I + 48 ) )
      ; SHIP_OUT
      END

  ; PROCEDURE TRANSMIT_CURSOR_POS ( VAR X , Y : INTEGER )
"3.3.4"

    ; VAR C : CHAR

    ; BEGIN
        ESC
      ; OUTPUT ( 'Y' )
      ; SHIP_OUT
      ; ACCEPT ( C )
      ; X := ( ORD ( C ) - 48 ) * 100
      ; ACCEPT ( C )
      ; X := X + ( ORD ( C ) - 48 ) * 10
      ; ACCEPT ( C )
      ; X := X + ORD ( C ) - 48
      ; ACCEPT ( C )
      ; Y := ( ORD ( C ) - 48 ) * 100
      ; ACCEPT ( C )
      ; Y := Y + ( ORD ( C ) - 48 ) * 10
      ; ACCEPT ( C )
      ; Y := Y + ORD ( C ) - 48
      END

  ; PROCEDURE LOG_DEVICE_ASSIGN
      ( IO , LOGICAL : CHAR ; PHYSICAL : INTEGER )


    ; BEGIN
        ESC
      ; OUTPUT ( IO )
      ; OUTPUT ( LOGICAL )
      ; OUTPUT ( CHR ( PHYSICAL + 48 ) )
      ; SHIP_OUT
      END

  ; PROCEDURE DEC_COORD_MODE "3.5.1"

    ; BEGIN MODE ; OUTPUT ( 'E' ) ; SHIP_OUT END

  ; PROCEDURE BI_COORD_MODE "3.5.2"
```

```
  ; BEGIN MODE ; OUTPUT ( 'B' ) ; SHIP_OUT END

; PROCEDURE VISIBLE_CURSOR "3.6.1.1"

  ; BEGIN MODE ; OUTPUT ( 'J' ) ; SHIP_OUT END

; PROCEDURE BLIND_CURSOR "3.6.1.2"

  ; BEGIN MODE ; OUTPUT ( 'K' ) ; SHIP_OUT END

; PROCEDURE CURSOR_COLOR ( COLOR_NUM : INTEGER ) "3.6.1.3"

  ; BEGIN
      MODE
    ; OUTPUT ( 'Q' )
    ; OUTPUT ( CHR ( COLOR_NUM + 48 ) )
    ; SHIP_OUT
    END

; PROCEDURE HOME "3.6.2.1"

  ; BEGIN OUTPUT ( CHR ( 28 ) ) ; SHIP_OUT END

; PROCEDURE TAB "3.6.2.3"

  ; BEGIN OUTPUT ( CHR ( 9 ) ) ; SHIP_OUT END

; PROCEDURE BACKSPACE "3.6.2.5"

  ; BEGIN OUTPUT ( CHR ( 8 ) ) ; SHIP_OUT END

; PROCEDURE VERTICAL_TAB "3.6.2.6"

  ; BEGIN OUTPUT ( CHR ( 11 ) ) ; SHIP_OUT END

; PROCEDURE CURSOR_RIGHT "3.6.2.7"

  ; BEGIN OUTPUT ( CHR ( 29 ) ) ; SHIP_OUT END

; PROCEDURE SET_INTERLINE ( I : INTEGER ) "3.6.2.8"

  ; BEGIN MODE ; OUTPUT ( 'A' ) ; OUTPUT_DEC ( I ) ;
    SHIP_OUT END

; PROCEDURE ONE_DOT_UP "3.6.3.1"

  ; BEGIN OUTPUT ( CHR ( 5 ) ) ; SHIP_OUT END

; PROCEDURE ONE_DOT_DOWN "3.6.3.2"

  ; BEGIN OUTPUT ( CHR ( 22 ) ) ; SHIP_OUT END

; PROCEDURE ONE_DOT_LEFT "3.6.3.3"
```

```
  ; BEGIN OUTPUT ( CHR ( 25 ) ) ; SHIP_OUT END

; PROCEDURE ONE_DOT_RIGHT "3.6.3.4"

  ; BEGIN OUTPUT ( CHR ( 31 ) ) ; SHIP_OUT END

; PROCEDURE MOVE_CURSOR ( X , Y : INTEGER ) "3.6.3.5"

  ; BEGIN
      MODE
    ; OUTPUT ( 'U' )
    ; OUTPUT_COORD ( X , Y )
    ; SHIP_OUT
    END

; PROCEDURE ALTERNATE_SET_ON "3.7.1.1"

  ; BEGIN OUTPUT ( CHR ( 14 ) ) ; SHIP_OUT END

; PROCEDURE ALTERNATE_SET_OFF "3.7.1.2"

  ; BEGIN OUTPUT ( CHR ( 15 ) ) ; SHIP_OUT END

; PROCEDURE SELECT_UPPER_SET ( I : INTEGER ) "3.7.1.3"

  ; BEGIN
      MODE
    ; OUTPUT ( 'S' )
    ; OUTPUT ( CHR ( I + 48 ) )
    ; SHIP_OUT
    END

; PROCEDURE SET_CHAR_Y ( Y : INTEGER ) "3.7.2.1"

  ; BEGIN MODE ; OUTPUT ( 'Y' ) ; OUTPUT_DEC ( Y )
   ; SHIP_OUT END

; PROCEDURE SET_CHAR_X ( X : INTEGER ) "3.7.2.2"

  ; BEGIN MODE ; OUTPUT ( 'X' ) ; OUTPUT_DEC ( X )
  ; SHIP_OUT END

; PROCEDURE HORIZONTAL_MODE "3.7.3.1"

  ; BEGIN MODE ; OUTPUT ( 'H' ) ; SHIP_OUT END

; PROCEDURE VERTICAL_MODE "3.7.3.2"

  ; BEGIN MODE ; OUTPUT ( 'V' ) ; SHIP_OUT END

; PROCEDURE SELECT_COLOR ( COLOR_NUM : INTEGER ) "3.7.4.1"

  ; BEGIN
```

```
      MODE
    ; OUTPUT ( 'C' )
    ; OUTPUT ( CHR ( COLOR_NUM + 48 ) )
    ; SHIP_OUT
    END

; PROCEDURE BACKGROUND_ON "3.7.4.3"

  ; BEGIN MODE ; OUTPUT ( 'M' ) ; SHIP_OUT END

; PROCEDURE BACKGROUND_OFF "3.7.4.4"

  ; BEGIN MODE ; OUTPUT ( 'N' ) ; SHIP_OUT END

; PROCEDURE BLINK_ON "3.7.4.4"

  ; BEGIN MODE ; OUTPUT ( '1' ) ; SHIP_OUT END

; PROCEDURE BLINK_OFF "3.7.4.5"

  ; BEGIN MODE ; OUTPUT ( '2' ) ; SHIP_OUT END

; PROCEDURE ERASE_PAGE "3.7.5.1"

  ; BEGIN OUTPUT ( CHR ( 12 ) ) ; SHIP_OUT END

; PROCEDURE ERASE_LINE "3.7.5.2"

  ; BEGIN MODE ; OUTPUT ( '@' ) ; SHIP_OUT END

; PROCEDURE TEST ( C : CHAR ) "3.7.5.3"

  ; BEGIN MODE ; OUTPUT ( 'T' ) ; OUTPUT ( C )
  ; SHIP_OUT END

; PROCEDURE ERASE_TO_END "3.7.5.4"

  ; BEGIN MODE ; OUTPUT ( '3' ) ; SHIP_OUT END

; PROCEDURE SET_WINDOW
  ( X1 , Y1 , X2 , Y2 : INTEGER ) "3.8.1"

  ; BEGIN
      MODE
    ; OUTPUT ( 'W' )
    ; OUTPUT_COORD ( X1 , Y1 )
    ; OUTPUT_COORD ( X2 , Y2 )
    ; SHIP_OUT
    END

; PROCEDURE KEYBOARD_SYNC "3.8.2.1"

  ; BEGIN MODE ; OUTPUT ( '0' ) ; SHIP_OUT END
```

```
; PROCEDURE PLOT_ON "3.9.1"

  ; BEGIN MODE ; OUTPUT ( 'G' ) ; SHIP_OUT END

; PROCEDURE PLOT_OFF "3.9.2"

  ; BEGIN OUTPUT ( CHR ( 21 ) ) ; SHIP_OUT END

; PROCEDURE X_BAR ( X1 , Y1 , X : INTEGER ) "3.9.5.1"

  ; BEGIN
      OUTPUT ( '!' )
    ; OUTPUT_COORD ( X1 , Y1 )
    ; OUTPUT ( CHR ( X + 48 ) )
    ; SHIP_OUT
    END

; PROCEDURE Y_BAR ( X1 , Y1 , Y : INTEGER ) "3.9.5.2"

  ; BEGIN
      OUTPUT ( '"' )
    ; OUTPUT_COORD ( X1 , Y1 )
    ; OUTPUT ( CHR ( Y + 48 ) )
    ; SHIP_OUT
    END

; PROCEDURE DOT ( X , Y : INTEGER ) "3.9.5.5"

  ; BEGIN OUTPUT ( '%' ) ; OUTPUT_COORD ( X , Y )
   ; SHIP_OUT END

; PROCEDURE VECTOR
 ( X1 , Y1 , X2 , Y2 : INTEGER ) "3.9.5.7"

  ; BEGIN
      OUTPUT ( CHR ( 39 ) )
    ; OUTPUT_COORD ( X1 , Y1 )
    ; OUTPUT_COORD ( X2 , Y2 )
    ; SHIP_OUT
    END

; PROCEDURE CONCAT_VECTOR "3.9.5.7"

  ; BEGIN OUTPUT ( '(' ) ; SHIP_OUT END

; PROCEDURE ROLL_ON "4.1.1"

  ; BEGIN MODE ; OUTPUT ( 'R' ) ; SHIP_OUT END

; PROCEDURE ROLL_OFF "4.1.2"

  ; BEGIN MODE ; OUTPUT ( 'P' ) ; SHIP_OUT END

; PROCEDURE OVERSTRIKE "4.1.3"
```

```
    ; BEGIN MODE ; OUTPUT ( 'O' ) ; SHIP_OUT END

  ; PROCEDURE LATCH_OVERSTRIKE "4.1.4"

    ; BEGIN MODE ; OUTPUT ( ']' ) ; SHIP_OUT END

  ; PROCEDURE UNLATCH_OVERSTRIKE "4.1.5"

    ; BEGIN MODE ; OUTPUT ( '[' ) ; SHIP_OUT END

  ; PROCEDURE SELECT_OVERLAY_PLANE ( I : INTEGER ) "4.1.6"

    ; BEGIN
        MODE
      ; OUTPUT ( ':' )
      ; OUTPUT ( CHR ( I + 48 ) )
      ; SHIP_OUT
      END

  ; PROCEDURE CHORMATIC_DELAY
  ( TENS_SECOND : INTEGER ) "4.1.7"

    ; BEGIN
        MODE
      ; OUTPUT ( '?' )
      ; OUTPUT_DEC ( TENS_SECOND )
      ; SHIP_OUT
      END

  ; PROCEDURE COMPLEX_FILL ( C : CHAR )

"4.1.8 AND OPTION 76 IN BACK OF MANUAL."

    ; BEGIN MODE ; OUTPUT ( '>' ) ; OUTPUT ( C )
      ; SHIP_OUT END

  ; PROCEDURE COMPLEX_REVERSE_FILL ( C : CHAR ) "4.1.9"

    ; BEGIN MODE ; OUTPUT ( '<' ) ; OUTPUT ( C ) ; SHIP_OUT
END

  ; PROCEDURE INSERT_LINE "4.1.10"

    ; BEGIN MODE ; OUTPUT ( 'I' ) ; SHIP_OUT END

  ; PROCEDURE DELETE_LINE "4.1.11"

    ; BEGIN MODE ; OUTPUT ( 'D' ) ; SHIP_OUT END

  ; PROCEDURE INSERT_CHAR "4.1.12"

    ; BEGIN OUTPUT ( CHR ( 23 ) ) ; SHIP_OUT END
```

```
; PROCEDURE DELETE_CHAR "4.1.13"

  ; BEGIN OUTPUT ( CHR ( 6 ) ) ; SHIP_OUT END

; PROCEDURE FILL_ON "4.2.1"

  ; BEGIN MODE ; OUTPUT ( 'F' ) ; SHIP_OUT END

; PROCEDURE FILL_OFF "4.2.2"

  ; BEGIN MODE ; OUTPUT ( 'L' ) ; SHIP_OUT END

; PROCEDURE ARC
    ( X , Y , RADIUS , START_DEG , DEG : INTEGER ) "4.2.3"

  ; BEGIN
      OUTPUT ( ')' )
    ; OUTPUT_COORD ( X , Y )
    ; OUTPUT_DEC ( RADIUS )
    ; OUTPUT_DEC ( START_DEG )
    ; OUTPUT_DEC ( DEG )
    ; SHIP_OUT
    END

; PROCEDURE PUSH_CIRCLE_BUTTON

  ; BEGIN OUTPUT ( '*' ) ; SHIP_OUT END

; PROCEDURE CIRCLE ( X , Y , RADIUS : INTEGER ) "4.2.4"

  ; BEGIN
      OUTPUT ( '*' )
    ; OUTPUT_COORD ( X , Y )
    ; OUTPUT_DEC ( RADIUS )
    ; SHIP_OUT
    END

; PROCEDURE PUSH_PERIOD_KEY

  ; BEGIN OUTPUT ( CHR ( 46 ) ) ; SHIP_OUT END

; PROCEDURE PUSH_RECTANGLE_BUTTON

  ; BEGIN OUTPUT ( '+' ) ; SHIP_OUT END

; PROCEDURE RECTANGLE
  ( X1 , Y1 , X2 , Y2 : INTEGER ) "4.2.5"

  ; BEGIN
      OUTPUT ( '+' )
    ; OUTPUT_COORD ( X1 , Y1 )
    ; OUTPUT_COORD ( X2 , Y2 )
    ; SHIP_OUT
    END
```

```
  ; PROCEDURE CREATE "4.3.1"

    ; BEGIN ESC ; OUTPUT ( 'C' ) ; SHIP_OUT END

  ; PROCEDURE VIEW_SUBBUFFER ( I : INTEGER ) "4.3.2"

    ; VAR J : INTEGER

    ; BEGIN ESC ; OUTPUT ( 'V' ) ; OUTPUT_HEX ( I )
     ; SHIP_OUT END

  ; PROCEDURE KILL_SUBBUFFER ( I : INTEGER ) "4.3.3"

    ; VAR J : INTEGER

    ; BEGIN ESC ; OUTPUT ( 'Q' ) ; OUTPUT_HEX ( I )
    ; SHIP_OUT END

  ; PROCEDURE APPEND_BUFFER

    ; BEGIN ESC ; OUTPUT ( 'Q' ) ; SHIP_OUT END

  ; PROCEDURE TRANSMIT_BUFFER

    ; BEGIN ESC ; OUTPUT ( 'U' ) ; SHIP_OUT END

  ; PROCEDURE REDRAW

    ; BEGIN ESC ; OUTPUT ( 'W' ) ; SHIP_OUT END

  ; PROCEDURE ZOOM ( X1 , Y1 , X2 , Y2 : INTEGER )

                                              "4.3.7"

    ; BEGIN
        MODE
      ; OUTPUT ( 'Z' )
      ; OUTPUT_COORD ( X1 , Y1 )
      ; OUTPUT_COORD ( X2 , Y2 )
      ; SHIP_OUT
      END

"*************************************************************
 *
 * THE FOLLOWING PROCEDURES ARE NOT PRIMITIVES OF
 *THE CHROMATICS.
 * THEY ARE ADDED FOR THE CONVENIANCE OF THE PROGRAMMER.
 *
 *************************************************************"

  ; PROCEDURE FIX ( K : INTEGER )

    ; BEGIN FOR J := K DOWNTO 0 DO "NOTHING" END
```

```
; PROCEDURE FILL_COORD_COLOR
    ( X , Y : INTEGER ; COLOR_NUM : INTEGER )

  ; BEGIN
      MOVE_CURSOR ( X , Y )
    ; BACKGROUND_ON
    ; CASE COLOR_NUM
      OF
        BLACK , BLUE , GREEN , CYAN
        , RED , MAGENTA , YELLOW , WHITE
        : BEGIN
          SELECT_COLOR ( COLOR_NUM )
        ; BACKGROUND_OFF
        ; COMPLEX_FILL ( CHR ( 32 ) )
        ; FIX ( 300000 )
        END
      ; ORANGE
        : BEGIN
          SELECT_COLOR ( RED )
        ; BACKGROUND_OFF
        ; SELECT_COLOR ( YELLOW )
        ; ALTERNATE_SET_ON
        ; SELECT_UPPER_SET ( 0 )
        ; COMPLEX_FILL ( CHR ( 97 ) )
        ; FIX ( 300000 )
        ; FIX ( 300000 )
        ; ALTERNATE_SET_OFF
        END
      ; PURPLE
        : BEGIN
          SELECT_COLOR ( BLUE )
        ; BACKGROUND_OFF
        ; SELECT_COLOR ( MAGENTA )
        ; ALTERNATE_SET_ON
        ; SELECT_UPPER_SET ( 0 )
        ; COMPLEX_FILL ( CHR ( 97 ) )
        ; FIX ( 300000 )
        ; FIX ( 300000 )
        ; ALTERNATE_SET_OFF
        END
      END
    END

; FUNCTION DCOS_ ( RADIANS : REAL ) : REAL
  ; FORTRAN

; FUNCTION DSIN_ ( RADIANS : REAL ) : REAL
  ; FORTRAN

; PROCEDURE PARA_ELLIPSE
    ( X , Y , SEMI_MINOR , SEMI_MAJOR , N : INTEGER
    ; INCLINATION : REAL
    )
```

```
  ; VAR X1 , Y1 , P , I1 , C1 , S1 , C2 , S2 ,
  C3 , S3 , T1 : REAL
    ; M , IX1 , IY1 , IX2 , IY2 : INTEGER

  ; BEGIN
      P := 2.0 * 3.14156 / CONV ( N - 1 )
    ; I1 := INCLINATION / 57.2957795
    ; C1 := DCOS ( I1 )
    ; S1 := DSIN ( I1 )
    ; C2 := DCOS ( P )
    ; S2 := DSIN ( P )
    ; C3 := 1.0
    ; S3 := 0.0
    ; CONCAT_VECTOR
    ; FOR M := 1 TO N
      DO BEGIN
        X1 := CONV ( SEMI_MAJOR ) * C3
      ; Y1 := CONV ( SEMI_MINOR ) * S3
      ; IX2 := X + ROUND ( X1 * C1 - Y1 * S1 )
      ; IY2 := Y + ROUND ( X1 * S1 + Y1 * C1 )
      ; IX1 := IX2
      ; IY1 := IY2
      ; OUTPUT_COORD ( IX1 , IY1 )
      ; SHIP_OUT
      ; T1 := C3 * C2 - S3 * S2
      ; S3 := S3 * C2 + C3 * S2
      ; C3 := T1
      END
    ; VECTOR ( IX1 , IY1 , IX2 , IY2 )
    ; DISPLAY ( NL )
    END

; PROCEDURE ELLIPSE ( X , Y , HEIGHT , WIDTH : INTEGER )

  ; VAR N , SMINOR , SMAJOR : INTEGER
    ; INCL : REAL

  ; BEGIN
      N := ( ( WIDTH + HEIGHT ) DIV 32 ) * 4 + 1
    ; INCL := 0.0
    ; SMINOR := ROUND ( CONV ( HEIGHT ) / 2.0 )
    ; SMAJOR := ROUND ( CONV ( WIDTH ) / 2.0 )
    ; PARA_ELLIPSE ( X , Y , SMINOR , SMAJOR , N , INCL )
    END

; PROCEDURE OUTPUT_LINE

  ; VAR I : INTEGER

  ; BEGIN
      I := 1
    ; REPEAT OUTPUT ( ALINE [ I ] )
      ; I := SUCC ( I )
```

```
      ;
      UNTIL ( I = 133 ) OR ( ALINE [ I ] = '$' )
    ; SHIP_OUT
    END

; PROCEDURE OUTPUT_INT_ ( I : INTEGER )

  ; VAR I_ST : ARRAY [ 1 .. 7 ] OF CHAR
    ; J , N , L : INTEGER

  ; BEGIN
      N := ABS ( I )
    ; FOR J := 7 DOWNTO 1
      DO BEGIN
        I_ST [ J ] := CHR ( N MOD 10 + 48 )
      ; N := N DIV 10
      END
    ; OUTPUT ( ' ' )
    ; IF I < 0
      THEN OUTPUT ( '-' )
    ; IF N <> 0
      THEN OUTPUT ( '*' )
    ; J := 1
    ; WHILE ( I_ST [ J ] = '0' ) AND ( J < 7 )
    DO J := J + 1
    ; FOR L := J TO 7 DO OUTPUT ( I_ST [ L ] )
    ; OUTPUT ( ' ' )
    ; SHIP_OUT
    END

";PROCEDURE ACCEPT_INT(VAR I:INTEGER)
;FORWARD"

; PROCEDURE ACCEPT_INT ( VAR I : INTEGER )

  ; VAR C : CHAR

  ; BEGIN
      BAD_INTEGER := FALSE
    ; ACCEPT ( C )
    ; IF C <> NL
      THEN BEGIN
      ; I := 0
      ; COUNT := 0
      ; REPEAT
        ; IF C IN [ '0' .. '9' ]
          THEN BEGIN
            I := I * 10 + ( ORD ( C ) - 48 )
          ; COUNT := COUNT + 1
          END
          ELSE BEGIN
            OUTPUT_LINE
              ( '(:10:)3 INTEGERS ONLY TRY AGAIN
              (:10:)(:13:)$' )
```

```
            ; BAD_INTEGER := TRUE
            END
          ; ACCEPT ( C )
          UNTIL ( C = NL ) OR ( COUNT = 3 )
        ; IF ( ( COUNT = 3 ) AND ( C <> NL ) )
          THEN REPEAT ACCEPT ( C ) UNTIL ( C = NL )
        END
      END

  ; PROCEDURE SET_UP_SCREEN

    ; BEGIN
        BACKGROUND_ON
      ; SELECT_COLOR ( BLACK )
      ; ERASE_PAGE
      ; FIX ( 300000 )
      ; PLOT_ON
      ; BACKGROUND_OFF
      ; SELECT_COLOR ( BLUE )
      ; RECTANGLE ( 000 , 000 , 511 , 511 )
      ; RECTANGLE ( 025 , 025 , 485 , 485 )
";FILL_COORD_COLOR(510,510,BLUE)"
      ; FIX ( 300000 )
      ; VECTOR ( 025 , 260 , 325 , 260 )
      ; VECTOR ( 325 , 025 , 325 , 485 )
      ; VECTOR ( 415 , 025 , 415 , 485 )
      ; PLOT_OFF
      ; LOG_DEVICE_ASSIGN ( 'O' , 'A' , 0 )
      ; SET_WINDOW ( 026 , 261 , 324 , 484 )
      ; BACKGROUND_ON
      ; SELECT_COLOR ( CYAN )
      ; BACKGROUND_OFF
      ; SELECT_COLOR ( BLUE )
      ; BLIND_CURSOR
      ; ERASE_PAGE
      ; FIX ( 300000 )
      ; LOG_DEVICE_ASSIGN ( 'O' , 'A' , 1 )
      ; SET_WINDOW ( 026 , 026 , 324 , 259 )
      ; BACKGROUND_ON
      ; SELECT_COLOR ( CYAN )
      ; BACKGROUND_OFF
      ; SELECT_COLOR ( BLUE )
      ; BLIND_CURSOR
      ; ERASE_PAGE
      ; FIX ( 300000 )
      ; LOG_DEVICE_ASSIGN ( 'O' , 'A' , 2 )
      ; SET_WINDOW ( 326 , 026 , 414 , 484 )
      ; BACKGROUND_ON
      ; SELECT_COLOR ( CYAN )
      ; BACKGROUND_OFF
      ; SELECT_COLOR ( BLUE )
      ; BLIND_CURSOR
      ; ERASE_PAGE
      ; FIX ( 300000 )
```

```
    ; LOG_DEVICE_ASSIGN ( 'O' , 'A' , 3 )
    ; SET_WINDOW ( 416 , 026 , 484 , 484 )
    ; BACKGROUND_ON
    ; SELECT_COLOR ( CYAN )
    ; BACKGROUND_OFF
    ; SELECT_COLOR ( BLUE )
    ; BLIND_CURSOR
    ; ERASE_PAGE
    ; FIX ( 300000 )
    END

; PROCEDURE BYE

  ; BEGIN
      LOG_DEVICE_ASSIGN ( 'O' , 'A' , 0 )
    ; SET_WINDOW ( 000 , 000 , 511 , 511 )
    ; PLOT_OFF
    ; BACKGROUND_ON
    ; SELECT_COLOR ( BLACK )
    ; BACKGROUND_OFF
    ; ERASE_PAGE
    ; FIX ( 300000 )
    ; SELECT_COLOR ( RED )
    ; MOVE_CURSOR ( 250 , 256 )
    ; OUTPUT_LINE ( 'BYE$' )
    ; CHORMATIC_DELAY ( 50 )
    ; BOOT
    ; CRTOS
    END

; PROCEDURE BASIC_SYMBOL

  ; BEGIN
      LOG_DEVICE_ASSIGN ( 'O' , 'A' , 2 )
    ; OUTPUT_LINE ( '(:10:) UNIT SYMBOLS(:10:)(:13:)$' )
    ; PLOT_ON
    ; SELECT_COLOR ( BLUE )
    ; RECTANGLE ( 345 , 375 , 395 , 425 )
    ; SELECT_COLOR ( CYAN )
    ; VECTOR ( 326 , 483 , 326 , 360 )
    ; PLOT_OFF
    ; SELECT_COLOR ( BLUE )
    ; OUTPUT_LINE ( '       A   $' )
    ; PLOT_ON
    "; VECTOR ( 345 , 290 , 345 , 340 )
    ; VECTOR ( 345 , 340 , 395 , 340 )
    ; VECTOR ( 395 , 340 , 377 , 315 )
    ; VECTOR ( 377 , 315 , 395 , 290 )
    ; VECTOR ( 395 , 290 , 345 , 290 )"
    ; RECTANGLE ( 345 , 300 , 395 , 350 )
    ; VECTOR ( 345 , 300 , 345 , 280 )
    ; SELECT_COLOR ( CYAN )
    ; VECTOR ( 326 , 320 , 326 , 280 )
    ; PLOT_OFF
```

```
    ; SELECT_COLOR ( BLUE )
    ; OUTPUT_LINE ( '       B  $' )
    ; PLOT_ON
    ; VECTOR ( 325 , 260 , 415 , 260 )
    ; LOG_DEVICE_ASSIGN ( 'O' , 'A' , 0 )
    END

; PROCEDURE FUNCTION_SYMBOL

  ; BEGIN
      LOG_DEVICE_ASSIGN ( 'O' , 'A' , 2 )
    ; MOVE_CURSOR ( 326 , 270 )
    ; OUTPUT_LINE ( '(:10:) FUNCTION SYM $' )
    ; PLOT_ON
    ; VECTOR ( 355 , 190 , 385 , 220 )
    ; VECTOR ( 355 , 220 , 385 , 190 )
    ; VECTOR ( 355 , 150 , 385 , 150 )
    ; VECTOR ( 370 , 165 , 370 , 135 )
    ; VECTOR ( 355 , 075 , 355 , 105 )
    ; VECTOR ( 355 , 105 , 385 , 105 )
    ; VECTOR ( 385 , 105 , 385 , 075 )
    ; VECTOR ( 370 , 105 , 370 , 075 )
    ; SELECT_COLOR ( CYAN )
    ; VECTOR ( 326 , 180 , 326 , 185 )
    ; PLOT_OFF
    ; SELECT_COLOR ( BLUE )
    ; OUTPUT_LINE ( '       C  $' )
    ; PLOT_ON
    ; SELECT_COLOR ( CYAN )
    ; VECTOR ( 326 , 160 , 326 , 130 )
    ; PLOT_OFF
    ; SELECT_COLOR ( BLUE )
    ; OUTPUT_LINE ( '       D  $' )
    ; PLOT_ON
    ; SELECT_COLOR ( CYAN )
    ; VECTOR ( 326 , 076 , 326 , 055 )
    ; PLOT_OFF
    ; SELECT_COLOR ( BLUE )
    ; OUTPUT_LINE ( '       E  $' )
    ; LOG_DEVICE_ASSIGN ( 'O' , 'A' , 0 )
    END

; PROCEDURE UNIT_SIZE

  ; BEGIN
      LOG_DEVICE_ASSIGN ( 'O' , 'A' , 3 )
    ; FIX ( 300000 )
    ; OUTPUT_LINE ( ' (:10:)UNIT SIZE (:10:)(:10:)$' )
    ; OUTPUT_LINE ( ' ARMY GROUP (:13:)$' )
    ; OUTPUT_LINE ( ' XXXXX(1)(:10:)(:13:) $' )
    ; OUTPUT_LINE ( '(:10:)ARMY(:10:)(:13:)$' )
    ; OUTPUT_LINE ( ' XXXX(2)(:10:)(:13:)$' )
    ; OUTPUT_LINE ( '(:10:) CORPS(:10:)(:13:)$' )
    ; OUTPUT_LINE ( ' XXX(3)(:10:)(:13:)$' )
```

```
    ; OUTPUT_LINE ( '(:10:) DIVISION(:10:)(:13:)$' )
    ; OUTPUT_LINE ( ' XX(4)(:10:)(:13:)$' )
    ; OUTPUT_LINE ( '(:10:) BRIGADE(:10:)(:13:)$' )
    ; OUTPUT_LINE ( ' X(5)(:10:)(:13:)$' )
    ; OUTPUT_LINE ( '(:10:) REGIMENT(:10:)(:13:)$' )
    ; OUTPUT_LINE ( ' 111(6)(:10:)(:13:)$' )
    ; OUTPUT_LINE ( '(:10:) BATTALION(:10:)(:13:)$' )
    ; OUTPUT_LINE ( ' 11(7)(:10:)(:13:)$' )
    ; OUTPUT_LINE ( '(:10:) COMPANY(:10:)(:13:)$' )
    ; OUTPUT_LINE ( ' 1(8)$' )
    ; LOG_DEVICE_ASSIGN ( 'O' , 'A' , 0 )
    END

; PROCEDURE PLOT_UNIT ( A_B : CHAR )

  ; BEGIN
      IF A_B = 'A'
      THEN BEGIN
      ; PLOT_ON
      ; SELECT_COLOR ( BLUE )
      ; RECTANGLE ( 150 , 135 , 200 , 185 )
      END
      ELSE BEGIN
        PLOT_ON
      ; SELECT_COLOR ( BLUE )
      "; VECTOR ( 150 , 135 , 150 , 185 )
      ; VECTOR ( 150 , 185 , 200 , 185 )
      ; VECTOR ( 200 , 185 , 182 , 160 )
      ; VECTOR ( 182 , 160 , 200 , 135 )
      ; VECTOR ( 200 , 135 , 150 , 135 )"
      ; RECTANGLE ( 150 , 135 , 200 , 185 )
      ; VECTOR ( 150 , 135 , 150 , 120 )
      END
    ; PLOT_OFF
    ; FIX ( 30000 )
    END

; PROCEDURE PLOT_FUNCTION ( C_D : CHAR )

  ; BEGIN
    ; SELECT_COLOR ( BLUE )
    ; PLOT_ON
    ; CASE C_D
      OF
        'C'
        : BEGIN
          VECTOR ( 150 , 135 , 200 , 185 )
        ; VECTOR ( 150 , 185 , 200 , 135 )
        END
      ; 'D'
        : BEGIN
        ; VECTOR ( 150 , 160 , 200 , 160 )
        ; VECTOR ( 175 , 185 , 175 , 135 )
        END
```

```
      ; 'E'
        : BEGIN
          VECTOR ( 160 , 145 , 160 , 175 )
        ; VECTOR ( 160 , 175 , 190 , 175 )
        ; VECTOR ( 190 , 175 , 190 , 145 )
        ; VECTOR ( 175 , 175 , 175 , 145 )
        END
      END
    ; PLOT_OFF
    END

; PROCEDURE PLOT_STRENGTH ( X : CHAR )

  ; BEGIN
      LOG_DEVICE_ASSIGN ( 'O' , 'A' , 1 )
    ; SELECT_COLOR ( BLUE )
    ; CASE X
      OF
        '1'
        : BEGIN
          MOVE_CURSOR ( 160 , 195 )
        ; OUTPUT_LINE ( 'XXXXX$' )
        END
      ; '2'
        : BEGIN
          MOVE_CURSOR ( 163 , 195 )
        ; OUTPUT_LINE ( 'XXXX$' )
        END
      ; '3'
        : BEGIN
          MOVE_CURSOR ( 166 , 195 )
        ; OUTPUT_LINE ( 'XXX$' )
        END
      ; '4'
        : BEGIN
          MOVE_CURSOR ( 169 , 195 )
        ; OUTPUT_LINE ( 'XX$' )
        END
      ; '5'
        : BEGIN
          MOVE_CURSOR ( 171 , 195 )
        ; OUTPUT_LINE ( 'X$' )
        END
      ; '6'
        : BEGIN
          MOVE_CURSOR ( 166 , 195 )
        ; OUTPUT_LINE ( '111$' )
        END
      ; '7'
        : BEGIN
          MOVE_CURSOR ( 169 , 195 )
        ; OUTPUT_LINE ( '11$' )
        END
      ; '8'
```

```
        : BEGIN
          MOVE_CURSOR ( 171 , 195 )
        ; OUTPUT_LINE ( '1$' )
        END
      END
    ; LOG_DEVICE_ASSIGN ( 'O' , 'A' , 0 )
    END

; PROCEDURE A_PLOT ( X_1 , Y_1 : INTEGER )

  ; BEGIN ; RECTANGLE_ ( X_1 , Y_1 ) END

; PROCEDURE B_PLOT ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      VECTOR_ ( X_1 , Y_1 , X_1 , Y_1 + 75 )
    ; RECTANGLE_ ( X_1 , Y_1 + 75 )
    END

; PROCEDURE C_PLOT ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      VECTOR_ ( X_1 , Y_1 , X_1 + 300 , Y_1 + 220 )
    ; VECTOR_ ( X_1 , Y_1 + 220 , X_1 + 300 , Y_1 )
    END

; PROCEDURE D_PLOT ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      VECTOR_ ( X_1 + 150 , Y_1 + 220 , X_1 + 150 , Y_1 )
    ; VECTOR_ ( X_1 , Y_1 + 110 , X_1 + 300 , Y_1 + 110 )
    END

; PROCEDURE E_PLOT ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      VECTOR_ ( X_1 + 50 , Y_1 + 50 , X_1 + 50
        , Y_1 + 170 )
    ; VECTOR_ ( X_1 + 50 , Y_1 + 170 , X_1 + 250
      , Y_1 + 170 )
    ; VECTOR_ ( X_1 + 250 , Y_1 + 170 , X_1 + 250 ,
     Y_1 + 50 )
    ; VECTOR_ ( X_1 + 150 , Y_1 + 170 , X_1 + 150
     , Y_1 + 50 )
    END

; PROCEDURE X1_PLOT ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( 'XXXXX$' , X_1 + 082 , Y_1 + 220 )
    ; FLUSH_TEXT
    END

; PROCEDURE X2_PLOT ( X_1 , Y_1 : INTEGER )
```

```
  ; BEGIN
      START_STRING_AT ( 'XXXX$' , X_1 + 095 , Y_1 + 220 )
    ; FLUSH_TEXT
    END

; PROCEDURE X3_PLOT ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( 'XXX$' , X_1 + 108 , Y_1 + 220 )
    ; FLUSH_TEXT
    END

; PROCEDURE X4_PLOT ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( 'XX$' , X_1 + 136 , Y_1 + 220 )
    ; FLUSH_TEXT
    END

; PROCEDURE X5_PLOT ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( 'X$' , X_1 + 143 , Y_1 + 220 )
    ; FLUSH_TEXT
    END

; PROCEDURE X6_PLOT ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( '111$' , X_1 + 108 , Y_1 + 220 )
    ; FLUSH_TEXT
    END

; PROCEDURE X7_PLOT ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( '11$' , X_1 + 136 , Y_1 + 220 )
    ; FLUSH_TEXT
    END

; PROCEDURE X8_PLOT ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( '1$' , X_1 + 143 , Y_1 + 220 )
    ; FLUSH_TEXT
    END

; PROCEDURE PLOT
 ( A_B , C_D , X : CHAR ; X_1 , Y_1 : INTEGER )

  ; BEGIN
    ; CASE A_B
      OF
```

```
        'A' : A_PLOT ( X_1 , Y_1 )
      END
    ; CASE C_D
      OF
        'C' : C_PLOT ( X_1 , Y_1 )
      ; 'D' : D_PLOT ( X_1 , Y_1 )
      ; 'E' : E_PLOT ( X_1 , Y_1 )
      END
    ; CASE X
      OF
        '1' : X1_PLOT ( X_1 , Y_1 )
      ; '2' : X2_PLOT ( X_1 , Y_1 )
      ; '3' : X3_PLOT ( X_1 , Y_1 )
      ; '4' : X4_PLOT ( X_1 , Y_1 )
      ; '5' : X5_PLOT ( X_1 , Y_1 )
      ; '6' : X6_PLOT ( X_1 , Y_1 )
      ; '7' : X7_PLOT ( X_1 , Y_1 )
      ; '8' : X8_PLOT ( X_1 , Y_1 )
      END
    END

; PROCEDURE C_PLOT_HQ ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      VECTOR_ ( X_1 , Y_1 + 75 , X_1 + 300 , Y_1 + 295 )
    ; VECTOR_ ( X_1 , Y_1 + 295 , X_1 + 300 , Y_1 + 75 )
    END

; PROCEDURE D_PLOT_HQ ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      VECTOR_ ( X_1 + 150 , Y_1 + 295
       , X_1 + 150 , Y_1 + 75 )
    ; VECTOR_ ( X_1 , Y_1 + 185 , X_1 + 300 , Y_1 + 185 )
    END

; PROCEDURE E_PLOT_HQ ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      VECTOR_ ( X_1 + 50 , Y_1 + 125
                , X_1 + 50 , Y_1 + 245 )
    ; VECTOR_ ( X_1 + 50 , Y_1 + 245
          , X_1 + 250 , Y_1 + 245 )
    ; VECTOR_ ( X_1 + 250 , Y_1 + 245 , X_1 + 250 ,
     Y_1 + 125 )
    ; VECTOR_ ( X_1 + 150 , Y_1 + 245 , X_1 + 150 ,
     Y_1 + 125 )
    END

; PROCEDURE X1_PLOT_HQ ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( 'XXXXX$' , X_1 + 080 , Y_1 + 295 )
    ; FLUSH_TEXT
```

```
    END

; PROCEDURE X2_PLOT_HQ ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( 'XXXX$' , X_1 + 095 , Y_1 + 295 )
    ; FLUSH_TEXT
    END

; PROCEDURE X3_PLOT_HQ ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( 'XXX$' , X_1 + 110 , Y_1 + 295 )
    ; FLUSH_TEXT
    END

; PROCEDURE X4_PLOT_HQ ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( 'XX$' , X_1 + 136 , Y_1 + 295 )
    ; FLUSH_TEXT
    END

; PROCEDURE X5_PLOT_HQ ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( 'X$' , X_1 + 143 , Y_1 + 295 )
    ; FLUSH_TEXT
    END

; PROCEDURE X6_PLOT_HQ ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( '111$' , X_1 + 110 , Y_1 + 295 )
    ; FLUSH_TEXT
    END

; PROCEDURE X7_PLOT_HQ ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( '11$' , X_1 + 136 , Y_1 + 295 )
    ; FLUSH_TEXT
    END

; PROCEDURE X8_PLOT_HQ ( X_1 , Y_1 : INTEGER )

  ; BEGIN
      START_STRING_AT ( '1$' , X_1 + 143 , Y_1 + 295 )
    ; FLUSH_TEXT
    END

; PROCEDURE PLOT_HQ
            ( A_B , C_D , X : CHAR ; X_1 , Y_1 : INTEGER )
```

```
  ; BEGIN
    ; CASE A_B
      OF
        'B' : B_PLOT ( X_1 , Y_1 )
      END
    ; CASE C_D
      OF
        'C' : C_PLOT_HQ ( X_1 , Y_1 )
      ; 'D' : D_PLOT_HQ ( X_1 , Y_1 )
      ; 'E' : E_PLOT_HQ ( X_1 , Y_1 )
      END
    ; CASE X
      OF
        '1' : X1_PLOT_HQ ( X_1 , Y_1 )
      ; '2' : X2_PLOT_HQ ( X_1 , Y_1 )
      ; '3' : X3_PLOT_HQ ( X_1 , Y_1 )
      ; '4' : X4_PLOT_HQ ( X_1 , Y_1 )
      ; '5' : X5_PLOT_HQ ( X_1 , Y_1 )
      ; '6' : X6_PLOT_HQ ( X_1 , Y_1 )
      ; '7' : X7_PLOT_HQ ( X_1 , Y_1 )
      ; '8' : X8_PLOT_HQ ( X_1 , Y_1 )
      END
    END

; PROCEDURE TRANSFER_SYMBOL_TEXT ( A_B , C_D , X : CHAR )
  ; FORWARD

; PROCEDURE BUILD_SYMBOL_TEXT
 ( VAR FIRST_TIME_THRU : BOOLEAN )

  ; VAR Y_N , YN , A_B , C_D , X : CHAR
    ; X_1 , Y_1 : INTEGER

  ; BEGIN
      IF FIRST_TIME_THRU
      THEN BEGIN
        LOG_DEVICE_ASSIGN ( 'O' , 'A' , 0 )
      ; OUTPUT_LINE
          (
        '(:10:)DO YOU WISH TO BUILD A MILITARY SYMBOL?
           (:10:)(:13:)$')
      ; OUTPUT_LINE ( '(:10:)TYPE (Y)YES OR N(NO)
       (:10:)(:13:)$' )
      ; REPEAT
        ; INPUT ( Y_N )
        ;
        UNTIL ( Y_N = 'Y' ) OR ( Y_N = 'N' )
      ; IF Y_N = 'N'
        THEN BYE
        ELSE BEGIN
        ; ERASE_PAGE
        ; FIX ( 300000 )
        ; BASIC_SYMBOL
        ; FUNCTION_SYMBOL
```

```
        ; UNIT_SIZE
        ; FIRST_TIME_THRU := FALSE
        END
      END
    ; LOG_DEVICE_ASSIGN ( 'O' , 'A' , 1 )
    ; ERASE_PAGE
    ; FIX ( 300000 )
    ; LOG_DEVICE_ASSIGN ( 'O' , 'A' , 0 )
    ; OUTPUT_LINE
        ( '(:10:)SELECT UNIT SYMBOL BY TYPING A OR B
            (:10:)(:13:)$' )
    ; REPEAT
      ; INPUT ( A_B )
      ;
      UNTIL ( A_B = 'A' ) OR ( A_B = 'B' )
      OR ( A_B = 'X' )
    ; IF A_B <> 'X'
      THEN BEGIN
      ; PLOT_UNIT ( A_B )
      ; ERASE_PAGE
      ; FIX ( 300000 )
      ; OUTPUT_LINE
          (
         '(:10:)(:10:)(:10:)SELECT FUNTIONAL SYMBOL BY
          (:10:)(:13:)$')
      ; OUTPUT_LINE
          ( '(:10:)TYPING CORRESPONDING LETTER
           (:10:)(:13:)$' )
      ; REPEAT
        ; INPUT ( C_D )
        UNTIL ( C_D = 'C' ) OR ( C_D = 'D' )
         OR ( C_D = 'E' )
      ; IF C_D <> 'X'
        THEN BEGIN
          PLOT_FUNCTION ( C_D )
        ; ERASE_PAGE
        ; FIX ( 30000 )
        ; OUTPUT_LINE
            ( '(:10:)SELECT UNIT SIZE BY TYPING
            (:10:)(:13:)$' )
        ; OUTPUT_LINE ( '(:10:)CORRESPONDING NUMBER
           (:10:)(:13:)$' )
        ; REPEAT ; INPUT ( X ) UNTIL
            ( X IN [ '1' .. '8' ] )
        ; PLOT_STRENGTH ( X )
        ; ERASE_PAGE
        ; FIX ( 300000 )
        ; TRANSFER_SYMBOL_TEXT ( A_B , C_D , X )
        END
      END
    END

; PROCEDURE TRANSFER_SYMBOL_TEXT
```

```
; VAR FIRST_TIME_THRU : BOOLEAN
  ; Y_N , YN : CHAR
  ; X_1 , Y_1 : INTEGER
  ; DONE : BOOLEAN

; BEGIN
  ; FIRST_TIME_THRU := FALSE
  ; OUTPUT_LINE
      ( '(:10:)DO YOU WISH TO TRANSFER MILITARY
            (:10:)(:13:)$' )
  ; OUTPUT_LINE
      ( '(:10:)SYMBOL TO OVERLAY TYPE Y OR N ?
          (:10:)(:13:)$' )
  ; REPEAT ; INPUT ( Y_N ) UNTIL ( Y_N = 'Y' )
      OR ( Y_N = 'N' )
  ; IF ( Y_N = 'Y' )
    THEN BEGIN
      DONE := FALSE
    ; REPEAT
      ; BAD_INTEGER := TRUE
      ; WHILE BAD_INTEGER
        DO BEGIN
        ; OUTPUT_LINE
            ( '(:10:)TYPE 3 DIGIT X COORDINATE
             (:10:)(:13:)$' )
        ; OUTPUT_LINE ( 'AND PUSH RETURN(:10:)(:13:)$' )
        ; ACCEPT_INT ( X_1 )
        END
      ; ERASE_PAGE
      ; FIX ( 300000 )
      ; BAD_INTEGER := TRUE
      ; WHILE BAD_INTEGER
        DO BEGIN
        ; OUTPUT_LINE
            ( '(:10:)TYPE 3 DIGIT Y COORDINATE
             (:10:)(:13:)$' )
        ; OUTPUT_LINE
            ( 'TYPE CARRIAGE RETURN TO END NUMBER
              (:10:)(:13:)$' )
        ; ACCEPT_INT ( Y_1 )
        END
      ; OUTPUT_LINE ( '(:10:) ARE COORDINATES $' )
      ; OUTPUT_DEC ( X_1 )
      ; OUTPUT_DEC ( Y_1 )
      ; OUTPUT_LINE ( ' CORRECT ?(:10:)(:13:)$' )
      ; OUTPUT_LINE
          ( '(:10:) TYPE Y(YES) OR N(NO)
              (:10:)(:13:)$' )
      ; INPUT ( YN )
      ; IF YN = 'Y'
        THEN DONE := TRUE
        ELSE BEGIN DONE := FALSE ; ERASE_PAGE
           ; FIX ( 300000 ) END
      UNTIL DONE
```

```
      ; ERASE_PAGE
      ; FIX ( 300000 )
      ; IF A_B = 'A'
        THEN PLOT ( A_B , C_D , X , X_1 * 2 , Y_1 * 2 )
        ELSE PLOT_HQ ( A_B , C_D , X , X_1 * 2 , Y_1 * 2 )
      ; FLUSH_TEXT
      ; OUTPUT_LINE
          (
          '(:10:)DO YOU WISH TO BUILD ANOTHER MILITARY
             (:10:)(:13:)$'
          )
      ; OUTPUT_LINE
          ( '(:10:)SYMBOL?  TYPE (Y)YES OR (N)NO.
              (:10:)(:13:)$' )
      ; REPEAT
        ; INPUT ( Y_N )
        ;
        UNTIL ( Y_N = 'Y' ) OR ( Y_N = 'N' )
      ; IF Y_N = 'Y'
        THEN BUILD_SYMBOL_TEXT ( FIRST_TIME_THRU )
        ELSE BYE
      END
      ELSE OUTPUT_LINE
          (
          '(:10:)DO YOU WISH TO BUILD ANOTHER MILITARY
              (:10:)(:13:)$'
          )
    ; OUTPUT_LINE
      ('(:10:)SYMBOL? TYPE (Y)YES OR (N)NO
            (:10:)(:13:)$' )
    ; REPEAT ; INPUT ( Y_N ) ; UNTIL ( Y_N = 'Y' )
    OR ( Y_N = 'N' )
    ; IF Y_N = 'Y'
      THEN BUILD_SYMBOL_TEXT ( FIRST_TIME_THRU )
      ELSE BYE
    END

; PROCEDURE USER_PROG

  ; VAR FIRST_TIME_THRU : BOOLEAN

  ; BEGIN
      SET_UP_SCREEN
    ; FIRST_TIME_THRU := TRUE
    ; BUILD_SYMBOL_TEXT ( FIRST_TIME_THRU )
    END

; PROCEDURE INITIALIZE

  ; BEGIN
      INIT_SVC
    ; SET_SVC1_INPUT
    ; OUTPUT_COUNT := 0
    ; HEIGHT_CHAR := 30
```

```
    ; WIDTH_CHAR := 30
    ; OUTPUT_CHAR ( P_ON )
    ; USER_PROG
    ; PEN_UP
    ; OUTPUT_CHAR ( P_OFF )
    ; FLUSH_TEXT "EMPTY THE OUTPUT BUFFER"
    END

; BEGIN INITIALIZE END
```

APPENDIX C

PROCEDURE DICTIONARY FOR INPUT/OUTPUT ROUTINES

The following procedures are used to input and output data between the Chromatics, Soltec and Interdata. Some of the procedures use Supervisor Calls (SVC) to provide the program interface to the operator system. The SVC instructions are executed by programs to request OS/32 MT services.

1. SVC1 This procedure is an interface with the operating system. A module called PASDRIVR receives the SVC1_Block and generates an assembly language supervisor call number 1 using the information in the SVC1_Block passed to it.
2. SET_SVC1_INPUT This procedure defines the parameters of SVC1. The functions read, wait and image are used and the buffer stores one character at a time.
3. INPUT This procedure actually does input. SVC1 is called passing SVC1 parameter block and passes printable characters.
4. INIT_SVC This procedure sets up the output block called SVC1_OUT. SVC1_OUT uses the write and image function and it inputs into an array called an outline.
5. SHIP_OUT This procedure set the ending address to the starting address and the length of the field to be transferred.

6. OUTPUT This procedure increases length of output by one and places characters into array called outline.

7. OUTPUT_LINE This procedure makes calls on procedure OUTPUT, one character at a time until it reaches 133 characters or the '$' is reached.

8. FLUSH_TEXT This procedure is used to force the output buffer to write if anything is in it.

9. OUTPUT_INT This procedure puts out a number of a maximum of three digits.

10. OUTPUT_LINE This procedure is used in procedure START_STRING_AT_OUTPUT_LINE to set character height and width for the plotter and outputs string until NL, 80 characters or '$' is reached.

11. FIX This procedure is a time delay loop to allow the Chromatics time to process commands from the Interdata. Procedure Fix must follow commands which use fill, complex fill, and erase page. The number of iterations which FIX must loop depends on usage of Interdata and complexity of fill command.

MILITARY INTERACTIVE SYMBOL DESIGN PACKAGE

by

JUSTIN GUY BALLOU, III

B.S., Embry-Riddle Aeronautical University, 1975

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

ABSTRACT

On the battlefield of the future a varied and complex combination of people, cultures and languages in a vast geographical area could place the United States Army in a difficult communication situation. Because of a lack of a common language on the battlefield, the commander of a multi-national army will find it extremely difficult to communicate time sensitive requirements. One potential solution could be the use of high resolution graphics using common military graphic symbols.

The purpose of this project was to develop a small demonstration interactive graphics package which will produce military symbols on an overlay. The user will build the desired military sumbol on a terminal screen in three interactive phases. First, the geometric figure used for the basic symbol is selected from the screen menu. Next, the user must select the branch symbol from the menu which corresponds with the basic symbol. Finally, the user selects the unit size symbol from the menu. Once the desired military symbol has been constructed the military symbol is transferred to an overlay using six digit coordinates for a 1:50000 map.

The software package interprets key strokes from the Chromatics 1999 color graphics terminal to device dependent commands which are used by the Chromatics and Soltec plotter. The source code which interprets key strokes is written in sequential PASCAL for the Interdata 8/32.