Interactive Micro-Computer Software for the Production of Engineering Drawings

1986

Brian R. Garvey
University of Central Florida

Find similar works at: https://stars.library.ucf.edu/rtd

University of Central Florida Libraries http://library.ucf.edu

Part of the Engineering Commons

STARS Citation

Garvey, Brian R., "Interactive Micro-Computer Software for the Production of Engineering Drawings" (1986). Retrospective Theses and Dissertations. 4929.
https://stars.library.ucf.edu/rtd/4929

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
INTERACTIVE MICRO-COMPUTER SOFTWARE
FOR THE PRODUCTION OF ENGINEERING DRAWINGS

BY

BRIAN R. GARVEY
B.C.E., Georgia Institute of Technology, 1983

THESIS
Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Graduate Studies Program of the College of Engineering
University of Central Florida
Orlando, Florida

Spring Term
1986
ABSTRACT

This project has developed a microcomputer-based interactive Graphics Software System for the production of engineering drawings. The drawings, stored in a digital format, can be recalled and updated much faster than by using conventional drawing methods. This allows for increased production and a standardized drawing format.

The system was designed to be easy to use so that with little training, a user can create, edit, plot, and save drawings for later retrieval. In addition, an open architecture was used in the design of the drawing database and program command structure so that the system can be easily expanded upon as the need arises.

The software, developed for the IBM Personal Computer using the TURBO PASCAL Compiler, has been implemented and hard copy results are presented in order to demonstrate the system’s capabilities.
ACKNOWLEDGEMENTS

Special thanks must be extended to the staff of Geodesy Professional Services for their patience while this system was being developed and debugged; the staff of L. T. Ray and Associates, Inc. for their assistance in the production of this document; and to Dr. C. S. Bauer for his guidance on this project. Without their assistance, this work would not have been possible. Also in this thesis, references are made to several products:

IBM is a registered trademark of International Business Machines, Inc. MS-DOS is a registered trademark of Microsoft, Inc. TURBO PASCAL and TURBO GRAPHIX TOOLBOX are registered trademarks of Borland International, Inc.
# TABLE OF CONTENTS

**LIST OF FIGURES** ........................................ iv

**INTRODUCTION** ........................................... 1

- Background ............................................. 1
- Objectives ............................................ 4
- General Description .................................... 9

**CHAPTER I - DATABASE MANAGEMENT** ..................... 13

- Organization .......................................... 13
- Data Structure ......................................... 17
- Expandability .......................................... 19

**CHAPTER II - DRAWING FUNCTIONS** ....................... 22

- Introduction .......................................... 22
- Command Descriptions ................................ 23

**CHAPTER III - USER INTERFACE** .......................... 79

- Introduction .......................................... 79
- Screen Layout .......................................... 82
- Command Selection ..................................... 86
- Keyboard Pointing ..................................... 89
- Workplace Layout ...................................... 92

**CHAPTER IV - CONCLUSIONS** .............................. 95

- System Implementation ............................... 95
- Implementation Problems ............................. 97
- Possible Improvements ................................ 99

**APPENDICES** ............................................. 101

- A - Development Environment ........................ 101
- B - Program Overlay Structure ....................... 104
- C - Program Listing ................................... 106
- D - Results ............................................ 196

**REFERENCES** ............................................. 203
<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>System Diagram</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>Drawing World Limits and Coordinate System Orientation</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>World to Screen Mapping Relationship</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>Object Data Structure</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>Block Diagram of Program Control Structure</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>Results of Scaling Operation</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>Methods of Describing Circular Arcs in a Drawing</td>
<td>42</td>
</tr>
<tr>
<td>8</td>
<td>Offset Command Examples</td>
<td>50</td>
</tr>
<tr>
<td>9</td>
<td>Extend and Modify Command Examples</td>
<td>52</td>
</tr>
<tr>
<td>10</td>
<td>Results of Break Command</td>
<td>55</td>
</tr>
<tr>
<td>11</td>
<td>Window Movement on a Zoomed Drawing</td>
<td>58</td>
</tr>
<tr>
<td>12</td>
<td>Typical Area Calculation</td>
<td>66</td>
</tr>
<tr>
<td>13</td>
<td>Screen Layout for User Interface</td>
<td>83</td>
</tr>
<tr>
<td>14</td>
<td>Drawing Commands on Each Command Menu</td>
<td>87</td>
</tr>
<tr>
<td>15</td>
<td>Crosshair Movement with the Numeric Keypad</td>
<td>90</td>
</tr>
<tr>
<td>16</td>
<td>Sample Drafting Workstation Layout</td>
<td>93</td>
</tr>
<tr>
<td>17</td>
<td>Program Overlay Structure</td>
<td>105</td>
</tr>
</tbody>
</table>
INTRODUCTION

Background

The computer has had an important role in engineering for well over twenty years. Its role however, has not been directed toward the center of the business, the engineering and design process. Computer usage instead has been directed toward service, in areas such as numerical processing, project management and control, and financial management. Even in engineering areas that traditionally use the computer, i.e., structural mechanics, and electrical system calculations, the program input is often copied manually from a drawing to the computer, and the results are manually incorporated into the design.

Recent advances in computer graphics, however, will move computer use from the perimeter to the center of the engineering and design process. With a properly integrated system, the computer will be displaying graphics, performing design functions, controlling data management, and handling communications, as well as all numerical computations.

Perhaps the most significant computer applications for the engineer are those which revolve around computer-aided design (CAD). One such application of computer-aided
design lies within the production of engineering drawings through the use of microcomputer graphics. The origins of computer-aided drafting can be traced through the history of interactive computer graphics. Computer graphics started with the early teletype and line printer hard copy devices of the 1940s. Computer driven CRTs (cathode-ray tubes) began to display output in the 1950s. Graphics, however, made little progress until 1962, when Dr. Ivan E. Sutherland published a paper describing a computer system that enabled users to draw. His paper proved to many readers that interactive computer graphics was feasible.

By the mid-1960s large computer graphic research projects were underway at MIT, General Motors, Bell Telephone Laboratories, and Lockheed Aircraft. This early research and the systems that were produced were mainframe or minicomputer based. The threshold cost for the use of this technology was greater than $200,000. This did not include the hidden costs of implementation, including a systems manager, operating personnel, training, site preparation, the development of a symbol library, a database, and custom software. An investment of $500,000 was not unrealistic.

The advent of the 8-bit microprocessor enabled and encouraged system manufacturers to develop lower priced
single user systems. In addition, developments in output-device technology and software in the 1970s have made microcomputer-aided drafting feasible. Currently more than a dozen software developers are marketing significant design and drafting systems for desktop computers. These are not expensive dedicated systems or bundled hardware and software configurations that are device dependent; they run on standard off-the-shelf microcomputers and peripherals. Because of this new microcomputer-based software, computer-aided drafting is now economically feasible for thousands of smaller engineering firms. The demands from this new market will greatly accelerate the applications of microcomputer graphics and computer-aided design.

The purpose of this thesis then, is to describe and document the development and implementation of a simple computer-aided drafting software system. This system has not been designed to compete with the increasing number of commercially available software packages. It is presented with the hope that the techniques used may be further expanded upon for the continued development of computer-aided design tools for the engineer.
Objectives of This System

This drawing system was developed with several objectives in mind. These objectives can be defined by answering the following questions:

What will the system do?
What functions will it serve?
Who will use the system?
What are the benefits of its use?
What are the drawbacks of its use?

This section presents answers to these questions in order to define the scope of this project, and to demonstrate the possible applications for this system.

The drawing system is a software application program which allows the user to manipulate two-dimensional graphic elements in a database. These graphic elements or objects consist of lines, circles, arcs, rectangles, symbols, and text. The user adds, deletes, and moves these objects in the database through a menu-driven user interface.

To interactively see the changes in the database as they occur, the user has a drawing window through which the objects can be seen. Control of the window is also available on the menu to enlarge a portion of these objects, so
that finer details can be seen; or reduce the size of the objects so that they can be seen in relation to each other.

For continued work over a long period of time, functions have been added to save this database into a disk file so that it may be recalled and modified whenever necessary. The program will also translate this data into a suitable command language for driving a pen plotter to produce a hard copy output of this database as a scaled drawing. Finally, these functions for editing, storage, retrieval, and output must be bundled together in a simple format so that someone with no knowledge of computers, graphics, or programming can use the system to its fullest extent.

The obvious function of a system such as this would be for the replacement of the manual techniques currently used for the production of drawings. Although the physical drawing speed for the man-computer combination is admittedly slower than that of a skilled drafter using conventional methods, if you account for the training time necessary for production, the drawing system has a clear advantage. A skilled draftsperson producing a finished production quality drawing in one hour might have had two to three years of experience in drawing and lettering. This involves a considerable investment of time and money
for training. With the computer, a person may produce that same original drawing in up to twice that time, however they may only require two to three months of training to reach a production quality level. When considering the economics including the supply of and demand for skilled drafters, the advantages of a computer drawing system can be seen.

Furthermore, the manual vs. computerized drawing time gap is further decreased when part of a drawing already exists in the computer. Many drafting functions involve redrawing existing details onto new drawings, a very time-consuming task. Using the drawing system, however, a portion of an existing drawing can be instantly copied into a new drawing. This reproductive ability is the main argument for the use of a computerized drawing system.

The software is designed to be used by anyone who has a need for producing small-sized, accurately scaled drawings. Although the software can be easily modified for larger drawings the current maximum allowable drawing size is 11" x 17". Some of the applicable engineering related industries which would have use for a system such as this include:
Land Surveyors: For property boundary sketches
Civil Engineers: For site plans
Architects: For floor plan diagrams
Land Planners: For building layouts
Landscape Architects: For landscape design

Although the system is tailored for civil engineering related applications, it could be used for any type of engineering drawings, from electrical circuit diagrams to machinery detailing. The only modifications that would be necessary would be to change the standard symbol library that is built into the software.

One of the benefits from using a system such as this one include the way that you may store your drawings. Using conventional methods, a drawing must be stored in some sort of bulky filing cabinet where it is subjected to the wear and tear of everyday use as other drawings are added and replaced. With a computerized system, the drawings are stored in a digital format and can be saved on floppy diskettes or on an internal drive which can hold hundreds of drawings.

Another advantage is that each drawing produced is an original that can be reproduced into several other original drawings. Each of these copies can be modified slightly to
produce five original drawings. Using conventional methods, however, unless the drawing is redrawn, each reproduction produces a degradation in the quality of the new drawing produced.

A final benefit from a system such as this would be the consistent quality of the drawings produced. Using manual methods, it is often obvious where one draftsperson stopped and another one continued on the same drawing. Lettering and line quality vary between draftspersons with differing experience. With a software system, however, the final drawing is produced by a single plotter using a consistent format with no distinction between the abilities of the operators.

As with any new system, however, there are always drawbacks which must also be recognized. One of the major drawbacks of computer drawing systems is that a user must convert their thoughts and ideas about what must be done into a command which can be recognized by a computer. Retraining of personnel is necessary. In addition, the user is sometimes forced to wait on the computer as the screen is being redrawn, or as the software is searching for a selected object. This hardware performance lapse can interrupt the user's train of thought and result in a slower production time.
Another problem that must be dealt with results from the storage of the drawings as data files. Although the files are easy to store and require much less space, they can also be lost quite easily. While it is not easy to erase an entire drawing on paper, a few accidental keystrokes can destroy hours of work instantaneously. Care must be taken in the storage of drawings to safeguard against accidental data loss.

Finally, as with all automation, the computerized production of drawings is now machine dependent and the production can be stopped at any time by a hardware failure. Care must be taken on the part of the user to have backup systems to continue production while the inoperative hardware is repaired.

General Description of System

The software developed for this project can be divided into three separate modules; the drawing module, the graphics utility module, and the plotter driver. The drawing module interacts with the user and the drawing data stored in memory. This data in memory can also be saved as a disk file.

The graphics utility module acts as an interpreter between the drawing module and the computer. This software
performs many of the low level functions necessary for graphic display. For a more detailed explanation of this software refer to Appendix A.

The plotter driver consists of a software procedure that accesses the drawing data and translates it into a command which can be understood by a specific pen plotter. This command is sent to the plotter as an ASCII text string.

For this project, plotter drivers were developed for two different pen plotters; the Hewlett Packard 7475A, and the Houston Instruments DMP-29. Source code listings of all the software are provided in Appendix C. The interaction between the software and hardware can be seen in Figure 1.

The hardware for the drawing system consists of a computer containing an appropriate graphics display controller, graphics monitor, keyboard, and disk drive. A pen plotter is cabled to the computer through a serial communications port. All input from the user to the system is received from the keyboard. This input includes command selection and pointing, object selection and data input. It is processed by the drawing module and responses are sent to the monitor.
Figure 1. System Diagram.
The hardware for the implementation of this project consisted of an IBM Personal Computer with 640 K RAM and an 8087 numerical co-processor. Mass storage included a 10 MB fixed disk, and one double-sided, double density diskette drive. The monitor was a standard IBM Color Graphics monitor driven by an IBM Color Graphics Card. A Houston Instrument DMP-29 Pen Plotter was used to output the results reproduced in Appendix D. For further information regarding the hardware configuration, refer to Appendix A.
An engineering drawing is a collection of several distinct graphic objects. Once it has been defined, an object on a drawing can be positioned, moved, and erased. Each time a change is made to objects in a drawing, the drawing database must be updated to reflect this change. This can be referred to as the data management.

The drawing data is stored on a disk, and is read into the system's random access memory (RAM) when a drawing is loaded by the user. The entire collection of objects contained within that drawing are placed into memory at that time. The software, then, contains no paging to a disk file for additional drawing data. Therefore the maximum allowable number of drawing objects is limited by the system's RAM capacity.

To create a drawing, the objects must be positioned within some sort of rectangular coordinate system. Using a coordinate system, each object can be mapped to a particular location on the drawing. When an object is moved, the system must simply update the object's
coordinates. The limits of this coordinate system are defined by size of the drawing that the user is currently working on. For example, when the user is working on an 8.5" x 14" drawing, the coordinate system's limits are defined from (0,0) to (8.5,14). These boundaries define the drawing world, so they are referred to as the minimum and maximum world limits. For an 11" x 17" drawing the world limits range from (0,0) to (17,11). These limits, and their orientation, can be seen in Figure 2.

Since it would be too difficult to achieve any sort of detail when mapping an entire drawing to the computer screen, the user is allowed to zoom in on a smaller portion of the entire world. This subset of the maximum drawing world is called the current world. The current world is then mapped to the drawing window on the computer screen. The relationship between the maximum world, current world, and drawing window can be viewed in Figure 3. When a new current world is to be displayed within the drawing window, the existing window is eased and the current world limits are scaled to the new drawing window. Each object within the drawing is redrawn with this new magnification and clipped to fit within the drawing window.
Figure 2. Drawing World Limits and Coordinate System Orientation
Figure 3. World To Screen Mapping Relationship
The objects in a drawing are broken down into six distinct graphic entities; lines, circles, arcs, text, rectangles, and symbols. These entities make up the six different types of objects in a drawing. Each of these object types have different requirements for description. For example, a line is described by the coordinates of its two endpoints, while a circle is described by its radius point coordinates and its radius distance. Given below is a list of the information necessary to completely describe each of the object types.

**Line**
- an integer to describe the type of line desired (dashed, bold, etc.)
- the X coordinate of the first endpoint
- the Y coordinate of the first endpoint
- the X coordinate of the second endpoint
- the Y coordinate of the second endpoint

**Text**
- a flag indicating whether the text is normal or bold
- a character string containing the text
- the X coordinate of the text starting point
- the Y coordinate of the text starting point
- the height of the text (we have assumed a standard height to width ratio)
- the angle of the text (measured from horizontal)
Circle - the X coordinate of the radius point
      the Y coordinate of the radius point
      the radius distance

Arc - the X coordinate of the starting point
       the Y coordinate of the starting point
       the X coordinate of the radius point
       the Y coordinate of the radius point
       the angle subtended by the arc

Rect - (sides are assumed to be horizontal and vertical)
       X coordinate at the upper left hand corner
       Y coordinate of the upper left hand corner
       X coordinate of the lower right hand corner
       Y coordinate of the lower right hand corner

Symbol - A symbol is a specialized element which is defined by the system. The symbols currently defined consist of a north arrow, a leader arrow, a small circle, a small box, and a small cross. Each of these objects require the following data to be described.

North Arrow - its starting coordinates and direction angle

Leader Arrow - three coordinate points

Circle, Box and Cross - center point coordinates
A data structure had to be defined so that each of the drawing objects are contained in an array. The object data was described as a record in Pascal consisting of two integers, and six real numbers. Figure 4 shows a table explaining how each of the object types is mapped to fit within the record.

You may notice in Figure 4 that with this data structure there is no provision for the character string containing the text that is to be displayed. To solve this problem the first real number in the text data structure is used as an index pointer to locate the text in a separate array of character strings. The complete data for a drawing, consists of two arrays, an array of objects and an array of text strings. The first record in the object array contains the overall drawing data including the drawing size and the drawing scale. In its simplest form, the drawing module acts as an editor that loads, stores, and manipulates these arrays and graphically displays the result.

**Expandability**

To conclude this section on data management, some techniques are suggested for the expansion of this data set at a later date in time. Possible enhancements of the data
<table>
<thead>
<tr>
<th>OBJECT TYPE</th>
<th>INT 1</th>
<th>INT 2</th>
<th>REAL 1</th>
<th>REAL 2</th>
<th>REAL 3</th>
<th>REAL 4</th>
<th>REAL 5</th>
<th>REAL 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINE</td>
<td>1</td>
<td>TYPE</td>
<td>FROM</td>
<td>FROM</td>
<td>TO X</td>
<td>TO Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1-9</td>
<td>X</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td>2</td>
<td>BOLD=1</td>
<td>ARRAY</td>
<td>INDEX#</td>
<td>START</td>
<td>START</td>
<td>SIZE</td>
<td>ANGLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>Y</td>
<td>X</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIRCLE</td>
<td>3</td>
<td>RAD X</td>
<td>RAD Y</td>
<td>RADIUS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARC</td>
<td>4</td>
<td>START</td>
<td>START</td>
<td>RADIUS</td>
<td>RADIUS</td>
<td>ANGLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>Y</td>
<td>X</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECT</td>
<td>5</td>
<td>UPPER</td>
<td>UPPER</td>
<td>LOWER</td>
<td>LOWER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LEFT</td>
<td>LEFT</td>
<td>RIGHT</td>
<td>RIGHT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>Y</td>
<td>X</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYMBOL</td>
<td>6</td>
<td>1</td>
<td>START</td>
<td>START</td>
<td>ANGLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NORTH</td>
<td></td>
<td></td>
<td>X</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARROW</td>
<td></td>
<td>4</td>
<td>1st X</td>
<td>2nd X</td>
<td>3rd X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEADER</td>
<td></td>
<td></td>
<td>1st Y</td>
<td>2nd Y</td>
<td>3rd Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARROW</td>
<td></td>
<td>2</td>
<td>CENT</td>
<td>CENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIRCLE</td>
<td></td>
<td>3</td>
<td>CENT</td>
<td>CENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOX</td>
<td></td>
<td>5</td>
<td>CENT</td>
<td>CENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CROSS</td>
<td></td>
<td></td>
<td>CENT</td>
<td>CENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4. Drawing Object Data Structure.**
could include additional data types, additional symbol types, larger object arrays, and larger text arrays. Since the data type is described using an integer, new types of objects may be added to the data list without having to modify any existing data structures. The new object type would just have an unused type descriptor for the first integer value (i.e., 7, 8, 9, etc.). The same method could be applied to the symbol type descriptor for additional symbols.

Finally, since both the drawing objects and drawing text are defined as arrays, they can be enlarged by simply changing dimensions of their declarations in the variable section of the source code. The constants maxobj and maxtext have been provided in the code to facilitate this expansion.
CHAPTER II

DRAWING FUNCTIONS

Introduction

When the program is started, several initialization procedures must be invoked before the user is able to select a drawing command. The first procedure clears a section of memory for use by the drawing data. After this memory has been cleared, a procedure is called to initialize the routines within the graphics utility module. This includes the reading, from disk, of the custom fonts used to display characters on the screen.

When the graphics utilities module has been initialized, the program sets values for the variables within the drawing module including world size, window size, and drawing scale. After the drawing module initialization, the program enters its normal operating state within a procedure called menu. Menu is the control procedure that allows the user to branch to the various drawing commands. Program control will remain within the menu procedure until the QUIT command is specified by the user. When the QUIT command is received, the menu
procedure is terminated and a text initialization procedure is called to return the computer’s display to its normal mode. A block diagram displaying this flow can be seen in Figure 5.

This section will present a descriptive explanation and overview of each of the drawing commands available to the user. The information provided includes a description of the command’s operation, an explanation of the source code routine that invokes the function, and a description of the command’s result. The functions are presented in the same order as they appear in the drawing command menu; the primary menu functions are explained first, then the secondary menu functions, followed by an explanation of the individual function keys used by the software.

**COMMAND DESCRIPTIONS**

**DWG COMMAND**

**DESCRIPTION:** The DWG command is a mnemonic representing the functions that involve the physical manipulation of the drawing files within the system’s memory and those contained on a disk. These functions include LOAD which reads a file from a disk into memory, STORE for saving the file in memory onto a disk, and RENAME which will change the filename of the current drawing in memory.
BEGIN

DATA INITIALIZATION

GRAPHICS UTILITY MODULE INITIALIZATION

DRAWING MODULE INITIALIZATION

MENU CONTROL PROCEDURE

TEXT MODE INITIALIZATION

END

Figure 5. Block Diagram of Program Control Structure
The procedure which is called to execute the DWG command is called getdwg. The following data is passed to procedure getdwg:

- **dwgname**: the name of the current file in memory
- **object**: the array of drawing objects in memory
- **onscreen**: an index containing the objects currently displayed on the screen
- **textline**: the array of text strings in the drawing
- **minx, maxx, miny, and maxy**: the current minimum and maximum X and Y limits of the world being displayed
- **crossx, crossy**: the world coordinate position of the drawing crosshair

The Getdwg procedure has three subprocedures that it uses; loadwg, storedwg, and rename. Each of these performs its respective function as described below.

The loadwg subprocedure loads the drawing data consisting of the object array and textline array from a diskette and into memory. After asking the user for the filename to be loaded, it will sequentially read into memory every object and every text line contained in the two files, filename.DWG and filename.DTX. The routine will then set new current world limits to accommodate the new drawing, update the text information on the screen and then display the new drawing within the viewing window.
The storedwg sub-procedure writes the current object and textline data in memory to the disk files; filename.DWG and filename.DTX. The filename used is the one that is currently stored within the program's memory. If no name is currently stored, then the user is prompted for one. Storedwg has no affect on the drawing data in memory or the viewing window.

The rename subprocedure simply changes the name of the filename variable stored within the program's memory. It prompts the user for the new filename desired and the old filename is overwritten.

**BORDER COMMAND**

**DESCRIPTION:** The BORDER command allows the user to select the size of the drawing that is being worked on. The current drawing sizes that may be selected at this point are 8.5" x 14" and 11" x 17". A drawing size may be selected before any object is drawn or it may be selected after several objects are already on the drawing, thus changing its size.

**CODING:** Procedure getborder is called to execute the BORDER command. The following data is passed to procedure getborder:
object: the array of drawing objects in memory
onscreen: the index containing the objects currently displayed on the screen

minx, maxx, miny, maxy: the current minimum and maximum
X and Y limits of the world being displayed
crossx, crossy: the world coordinate position of the drawing crosshair

After the user selects the drawing size, the maximum world coordinate limits are calculated so that the entire drawing can be displayed on the screen with the correct aspect ratio. These maximum limit values are assigned to the variables minx, maxx, miny, and maxy. These limits are used to rescale the drawing window. The crosshair world coordinates are also updated so that they are positioned in the center of the screen. Finally, the object array is redrawn so that the entire drawing, with its new border, is displayed on the screen.

SCALE COMMAND

DESCRIPTION: The SCALE command allows the user to change the current scale for the drawing that is being edited. Since the world coordinates are defined in inches, it is often useful to have some sort of scale factor which will convert the user units to world units. All numeric input
from the user to the system is multiplied by this scale factor. The SCALE command can also be used to rescale the current objects that are displayed on the screen. By increasing or decreasing the scale factor, the objects displayed can be enlarged or reduced in a drawing.

**CODING:** Procedure `changescale` is called to execute the SCALE function. The following data is sent to procedure `changescale`:

- `scale`: the scale factor variable to convert user units to inches
- `object`: the array of drawing objects in memory
- `onscreen`: the index containing the objects currently displayed on the screen
- `minx`, `maxx`, `miny`, `maxy`: the current minimum and maximum X and Y limits of the world being displayed
- `crossx`, `crossy`: the world coordinate position of the drawing crosshair

After the user enters the new scale that is desired, the screen is updated to display this new scale factor. The user is then asked if the entire drawing needs to be rescaled. If a total rescale is desired, the subprocedure `rescale` is called. The `rescale` subprocedure calculates a scale adjustment factor `(oldscale/newscale)` and multiplies this factor by all of the X and Y coordinate values for the objects in the drawing.
Once every object has been rescaled, the procedure will update the world limits and display the entire drawing on the screen.

RESULTS: If a drawing is currently at 1" = 30 feet and the scale is changed to 1" = 50 feet the net effect would be that the drawing would get smaller as each object is rescaled closer to the origin. This can be seen in Figure 6. Conversely, if the scale factor were reduced (i.e., 1" = 50 ft. to 1" = 30 ft.), the rescaled objects on the screen would appear to get larger and move away from the origin.

COGO COMMAND

DESCRIPTION: The COGO command allows the user to access previously defined figures, stored in coordinate geometry data files. The files created using a software system called Geodesy COGO, contain the coordinates of individual points that are grouped together as figures. Using this command the user is able to directly convert these geometric figures into objects which are displayed in the drawing. This command acts as an interface then, to share the data between the Geodesy COGO program and the drawing program.
Old Scale: 1" = 30'

New Scale: 1" = 50'

Figure 6. Results of Scaling Operation.
CODING: Since the Geodesy COGO software system is a proprietary program belonging to Geodesy Professional Services, a detailed explanation of Geodesy COGO or its data base structure will not be included in this thesis. For further information on this system, refer to the Geodesy COGO user's Manual referenced in the bibliography.

RESULTS: After a figure is copied from the COGO files into a drawing, its elements can be manipulated just like any other drawing object.

TEXT COMMAND

DESCRIPTION: The TEXT command is used to place alphanumeric data within a drawing. The user is asked to specify the starting point of the text, the text orientation, the text height, and the alphanumeric character string to be displayed. The text location is pointed to on the screen by the movement of the crosshairs. This location is the lower left-hand corner of the first character of the text string.

The text orientation may either be pointed to on the screen so that the text will be displayed along a line from the starting location to the pointed location, or the text orientation may be entered as an angle in degrees above or below the horizontal. For angles above horizontal the
angle entered would be positive, and for angles below horizontal the angle would be negative. The text height can be selected from one of five predetermined sizes ranging from 0.11 inches to 0.42 inches.

After its information has been entered, the text will be displayed on the screen showing its true size in reference to the drawing. The user is then given the option of keeping this text, or removing it and entering new text.

**CODING:** The TEXT command is implemented by calling procedure addtext. Procedure addtext contains three subprocedures. The first subprocedure getextinfo performs all of the input/output operations to determine the location, angle and height of the text. Showprompt, the second subprocedure, displays a prompt symbol to the user indicating the location, angle and size of the text. The final subprocedure, addtotext takes the alphanumeric text string and places it in the data base.

The addtext procedure performs the following functions. After calling procedure Getextinfo to determine the text position, orientation, and size, it prompts the user to enter the text string and displays the text prompt on the screen using the global procedure displaytext. If
the user wishes to keep the text, then the procedure prompts the user for another line of text immediately below the first line. If the first line is unsatisfactory, then the user may re-enter the text or end the procedure.

RESULTS: Since the program is not designed to display alphanumeric characters on the screen at any size and orientation, the text is displayed as a rectangle. This display rectangle defines the perimeter of the text string. Although this is an inconvenience from a human factors standpoint, it greatly enhances the display speed of an image. To view the text string represented by the display rectangle the user must use the VIEW command.

VIEW COMMAND

DESCRIPTION: The VIEW command is used to display the alphanumeric text string that is represented by a display rectangle on the screen. Since the program is not designed to display alphanumeric characters on the screen at any size and orientation, a display rectangle is used to indicate the presence of alphanumeric text in a drawing. Since it is often desirable to read this text, the VIEW command is quite useful.

After choosing the VIEW command, the user is asked to select the text to be viewed. This is done by moving the
crosshairs to a point along the display rectangle and pressing the ENTER key. After doing so, the program searches for the text string and displays it to the user along with the text height and other characteristics.

CODING: The VIEW command is executed by calling procedure Viewtext. After being called, Viewtext prompts the user for the text to be displayed by calling the procedures selectpoint and selectobject. Selectpoint prompts the user for a point on the screen and passes it to selectobject which searches for an object in the database adjacent to that point. Once an adjacent object has been found, if it is text, its contents and characteristics will be displayed to the user on the screen. If the object is not text then the message "No Text Selected" is displayed and the command ends.

RESULTS: If the item selected on the screen is indeed a display box representing a text string in the drawing database, its contents will be displayed on the bottom line of the screen. For example, if the box selected represented a text string for a dimension of 45.32 feet and that text was 0.14 inches tall, the bottom line of the screen would read

\[0.14\] 45.32 feet
This information will remain on the screen until the user presses any key. At this point the VIEW command ends and normal program operation will continue.

**LINE COMMAND**

**DESCRIPTION:** The LINE command is used to place line segments in the drawing. The program has nine different line types available to the user.

<table>
<thead>
<tr>
<th>Normal</th>
<th>Bold</th>
<th>Dashed</th>
<th>Right-of-Way</th>
<th>Centerline</th>
<th>Broken</th>
<th>Chain Link Fence</th>
<th>Wood Fence</th>
<th>Wall</th>
</tr>
</thead>
</table>

After selecting the LINE command the program prompts the user for the desired line type. After the line type has been selected, the user is asked to select the starting point of the new line segment with the crosshairs. When the starting point has been chosen, the user may either point to the desired location of the end point with the crosshairs, or he may enter a numerical angle and distance to that end point. The angle entered would have the following reference frame.
The distance would be measured along the line from the starting point to the ending point, in scaled units. After the information has been entered, the line is displayed in the drawing and the program asks for additional input to continue this line. The starting point of the new line is assumed to be the ending point of the previous line. In this manner, chains of continuous line segments can be created. If no further lines are desired then the user simply presses the ENTER key to exit from the command and return to the drawing menu.

CODING: The LINE command is executed by procedure Addline. The Addline procedure has two distinct sections. The first section determines the line type and starting point of the line, and the second section consists of a loop which prompts the user for the end point, either by selection or angle and distance information, and then displays that line. After the line is displayed its ending point becomes the starting point of the new line and the loop repeats itself. Procedure Addline also calls the two global procedures Addtobuffer and Displayline. Addtobuffer is used throughout the drawing program to add a newly created object into the drawing database. Procedure Displayline is also used to display a line on the screen when it is given the line type and the line endpoints.
The LINE command can be used to place a single line segment, or a series of line segments, onto a drawing. Each of the lines created will be displayed with a given line type. These line types may also be changed at a later point in time with the Change Line command.

CIRCLE COMMAND

DESCRIPTION: The CIRCLE command is used to place circles in the drawing. After selecting the CIRCLE command the program prompts the user for the center or radius point of the circle. This is done by moving the crosshairs around the drawing to select a desired location. After the radius point has been chosen, the user may either point to a location which is on the circle, or he may enter a radius distance for the circle. The radius distance would be measured from the center of the circle to a point on the circle in scaled units.

After the radius point and radius are known, the circle is displayed in the drawing, and the function ends.

CODING: The CIRCLE command is executed by procedure Addcircle. The Addcircle procedure has three separate parts. The first part prompts the user for the center
point of the circle and then branches to either one of the two remaining sections depending upon the user input. When asked to select a point on the circle or enter a radius, if the user presses a key to move the crosshairs then the program assumes that a point on the circle will be selected and control branches to the second section. If however, any input is received from the keyboard other than a crosshair control signal, then the program assumes that the circle’s radius will be entered numerically and control passes to the third section.

The second section of code is used when a point on the circle is being pointed to on the screen. This section will calculate the distance between the center point and the point on the circle. This is the radius distance. A circle is then displayed on the screen at proper location with the desired radius.

The third section of code is used when the radius is to be entered from the keyboard. This section will read the radius in from the keyboard, analyze it to make sure that it is in a proper numerical format, convert it from a scaled distance to a distance in drawing units, and then finally display the circle at the desired location with the entered radius.
RESULTS: A circle may be added to a drawing in one of two ways. You may either select a center point and a point on the circle as shown below on the left, or you may select a center point and enter a radius distance as shown below on the right.

![Diagram of circle drawing methods]

ARC COMMAND

DESCRIPTION: The ARC command is used to place circular arcs in the drawing. A circular arc is defined as a section of a circle. After selecting the ARC command the program prompts the user for the starting point and ending point of the arc. When the user has pointed to the arc's start and endpoints, the program will ask the user to either select the radius point, or enter a radius. If the radius point is selected then an arc is drawn from the start point to the endpoint around the given radius point. If a radius distance is entered, then the program will calculate an appropriate radius point and then draw the arc from the start point to the endpoint, about the calculated radius point.
CODING: Procedure Addarc is used to create arcs in a drawing. The procedure is divided up into three separate parts, an initial data-gathering section with a conditional branch, and two additional parts that represent the possible results of the branch.

The first section prompts for the arc's starting point and ending point using the selectpoint procedure. After these points have been input, the program branches to either one of the two remaining sections depending upon user input. When the program asks to select the center point or enter the arc's radius, if the user presses a key to move the crosshairs then the program assumes that the user is going to select the location of the center point with the crosshairs, and control branches to the second section.

If any input from the keyboard is received other than a crosshair control signal, then the program assumes that the arc's radius will be entered numerically and control passes to the third section.

The second section of code is used when the arc's center point is being pointed to on the screen. This section will calculate the angle between the starting point, the center point, and the radius point. This is
referred to as the angle subtended by the arc. Once this angle has been calculated the arc can be displayed on the drawing by passing the starting point, the ending point, and the angle subtended to the displayarc procedure.

The final section of code is used when the radius is to be entered from the keyboard. This section will read the radius in from the keyboard, analyze it to make sure that it is in the proper numerical format, convert it from a scaled distance to a distance in drawing units and then calculate the location of the arc's radius point, and the angle subtended by the arc. The arc is then displayed, using the displayarc procedure described above.

RESULTS: An arc may be described on a drawing by one of two ways. Both of the ways require a starting point and ending point of the arc. The first method allows the user to select an arc center point as shown in Figure 7a. The second method allows the user to enter a desired radius to be fit between the start and endpoints. Given only the radius, however, two curves can be fit between the start and endpoints so we must also specify a curve direction. A positive radius (R>0) will describe a clockwise arc as shown in Figure 7b, and a negative radius (R<0) will describe a counterclockwise arc as shown in Figure 7c.
(a) Arcs specified by a starting point, an ending point, and a center point.

(b) Arc specified by a starting point, an ending point, and a positive radius distance describing a clockwise arc.

(c) Arc specified by a starting point, an ending point, and a negative radius distance describing a counterclockwise arc.

Figure 7. Methods of Describing Circular Arcs in a Drawing
RECT COMMAND

DESCRIPTION: The RECT command is used to place rectangles in a drawing. The program prompts the user for the upper left-hand corner of the rectangle, and the lower right-hand corner of the rectangle. After these points are entered, it draws a rectangle between these points with horizontal and vertical lines.

CODING: The RECT command is executed by called procedure Addrect. This procedure prompts the user for the two diagonal corner points using the selectpoint procedure, and then displays the rectangle using the displayrect procedure. The rectangle is added as an object in the drawing database using the addtobuffer procedure.

RESULTS: When the diagonal end points (X1,Y1) and (X2,Y2) are given to the rectangle command the following rectangle will be displayed on the screen.

```
\begin{tikzpicture}
  \draw[very thick] (X1,Y1) rectangle (X2,Y2);
\end{tikzpicture}
```

The RECT command is quite useful for placing borders around text or diagrams in a drawing.
SYMBOL COMMAND

DESCRIPTION: The SYMBOL command is used to place system defined symbols in a drawing. These symbols have been coded into the software and cannot be changed without recoding. The predefined symbols consist of the following:

- North Arrow
- Small Circle
- Small Box
- Cross
- Leader Arrow

Each symbol is located on the drawing by the user specifying the location of either 1, 2, or 3 points. For example, the Circle, Box, and Cross symbols require only one point to be completely located and the symbol is centered at that point. The North Arrow requires two points to be located on a drawing, a location point for the base of the arrow, and a direction point to indicate the direction that the arrow should point. The Leader Arrow is located by 3 points, a location point for the start of the arrow, a break point to the middle of the arrow and an arrow point for the tip.
CODING: The SYMBOL command is executed by the Addsymbol procedure. This procedure is responsible for the acquisition of the location data for the symbols. After the point information for the desired symbol is known, the procedure calls another global procedure called displaySymbol which is responsible for drawing each particular symbol on the screen. The new symbol is also added to the drawing's database by using the procedure addtobuffer.

RESULTS: When a point location is specified for a circle, box or cross it will be displayed on the drawing centered on that location. For example, in the pictures below, the circle, box, and cross symbols are shown along with that specifier point.

![Circle](image1)

![Box](image2)

![Cross](image3)

The North Arrow, specified by two points can be placed on a drawing in the following ways.

![North Arrow](image4)
The Leader Arrow can be described with 3 points as follows:

FILLET COMMAND

DESCRIPTION: The FILLET command is used to place circular fillets between skew lines on a drawing. The user is asked to select each of the lines to be used, and the radius of the fillet to be placed between the lines. If a radius of zero is entered, then the lines are connected at their intersection point, and any excess line past the intersection is removed.

CODING: The FILLET command is executed by procedure Fillet. This procedure contains three sub-procedures Calcintersect, Ptinsegment and Calctangentpts. Calcintersect is used to calculate the intersection point of the two lines selected by the user. Ptinsegment is used to determine which part of the line to keep and which to
throw away. Procedure Calctangentpts is used to calculate the beginning and endpoints of the circular fillet to be placed between the two lines.

The main Fillet procedure prompts the user for the two lines that are used. Once the lines have been selected, their intersection point is calculated and the procedure decides which line segments are needed and which are not. After this has been done the original lines are erased from the screen and the new endpoints of the line are calculated. The modified lines are then redrawn and the arc between them is displayed.

RESULTS: The FILLET command is used to connect unrelated line segments or it may be used to fit curves between lines. For example, the lines shown below can be connected by using the FILLET command.
In addition, circular curves may be placed between the lines, as shown below:

OFFSET COMMAND

DESCRIPTION: The OFFSET command is used to add additional objects to the drawing parallel to existing objects. For example, lines can be offset with additional line segments parallel to the original, circles can be offset with other circles so that they are concentric, arcs can be offset so each additional arc has the same center point, and finally rectangles can be offset so that each new rectangle either contains, or is contained, by the previous rectangle. The OFFSET command asks the user to select the object to be offset, as well as indicating the offset direction and distance. The new object is then displayed on the screen and added to the drawing database.

CODING: The OFFSET command is implemented by procedure Offset. This procedure contains four (4) subprocedures, Offsetline, Offsetcircle, Offsetarc, and Offsetrect. The Offsetline procedure is used to calculate the endpoints of
a new line, parallel to a given line, at a specific offset distance. The Offsetcircle routine is used to calculate a new circle concentric to a given circle. Likewise, the Offsetarc and Offsetrect procedures are used to calculate parallel offset arcs and rectangles. The main procedure, Offset, consists of prompts to determine the object to be offset, the offset direction, and the offset distance.

After this information has been determined, the control branches to one of the four sub-procedures which calculates and displays the newly offset object.

RESULTS: An offset can be calculated for a line, circle, arc, and rectangle. For each of the objects shown in Figure 8, the original object is shown dashed. Figure 8a shows typical line offsets. Figure 8b shows typical circle offsets. Finally, Figures 8c and 8d give examples of typical arc and rectangle offsets.

EXTEND COMMAND

DESCRIPTION: The EXTEND command is used to extend or shorten any type of line. The new endpoint is a point along the line selected by the user. For example, if you want to lengthen a line on the drawing, or shorten a line on the drawing then you would use the EXTEND command. The command prompts the user to select the line to be extended
(a) Typical LINE Offsets

(b) Typical CIRCLE Offsets

(c) Typical ARC Offsets

(d) Typical RECTANGLE Offsets

Figure 8. Offset Command Examples.
and then to select the extended endpoint location. The new length for the line is then calculated, the old line is erased, and the new line is displayed.

CODING: The procedure Extendline is called to execute the EXTEND command. After prompting the user for the line and the new endpoint, the line endpoints are ordered and the new line length is calculated. After erasing the old line, the new line is drawn. The database is then modified with the new line endpoints and the command ends.

RESULTS: Figure 9a shows how a line can be lengthened or shortened by specifying a new endpoint with this command. The selected new endpoint does not have to be located directly along the line, since its perpendicular projector is calculated for the new length.

MODIFY COMMAND

DESCRIPTION: The MODIFY command is used to physically change the endpoint of a line. While the EXTEND command is used to extend a line along its existing direction to a new point, the MODIFY command actually changes the direction of a line by replacing its current endpoint with a new endpoint. The program prompts the user to select the line to be modified, and to select the new endpoint location. Since a selected line has two existing endpoints, the one that is closest to the newly selected endpoint is replaced.
(a) Examples of EXTEND Command use.

(b) Examples of MODIFY Command use.

Figure 9. Extend and Modify Command Examples
CODING: The Modifyline procedure is used to execute the MODIFY command. After prompting the user for the line to be modified and the new desired endpoint, the line is erased and the distance between each existing endpoint and the desired endpoint is calculated. The existing endpoint that is closest to the new endpoint is replaced with the new endpoint and the line is redrawn. Finally the drawing database is updated with the line’s new endpoints and the command terminates.

RESULTS: Figure 9b shows how a line endpoint can be changed using the MODIFY command. This can be contrasted to the results of the EXTEND command in Figure 9a. The difference between these two commands is significant. While the EXTEND command preserves the existing direction of the line, the MODIFY command does not.

BREAK COMMAND

DESCRIPTION: The BREAK command is used to break an object into smaller objects for partial deletion or removal. By using the BREAK command, a circle can be broken into an arc, an arc can be broken into two arcs, and a rectangle can be broken into four lines. For example, the line below can be broken twice into three separate lines so that a middle section can be deleted.
After selecting the BREAK command the program prompts the user to select the object to be broken. The selected object is then broken at its selection point.

CODING: The BREAK command is implemented by procedure Break. This procedure contains four subprocedures: Breakline, Breakcircle, Breakarc, and Breakrect. These procedures are used to break lines, circles, arcs, and rectangles, respectively. Each of these procedures is given the object that is to be broken and the X and Y coordinates of the point at which it is to be broken. Smaller objects are then calculated in place of the larger object and these new objects are placed in the drawing database. The original object is also erased from the screen and the two new objects are redrawn in its place.

RESULTS: The results of the BREAK command are shown in Figure 10. Individual lines are broken into two new lines in Figure 10a. Figure 10b shows the effect of breaks on circles. Arcs and rectangles are also broken in Figures 10c and 10d.
Figure 10. Results of BREAK Command.
DESCRIPTION: The WINDOW command is used to move the viewing window around on the drawing. When the user zooms in on a particular section of a drawing, the entire drawing is no longer visible in the viewing window on the screen. The viewing window can be moved around on the drawing while the user remains at a particular zoom magnification, using the WINDOW command. The WINDOW command allows the user to move the viewing window left, right, up, down or in any diagonal direction, using the arrow keys in a manner similar to the movement of the drawing crosshair on the screen.

CODING: The WINDOW command is coded as procedure Panwindow. After the Panwindow procedure has been called, the program calculates a distance that the window is to be moved. This distance is designed to be one-third of the current viewing area. For example, if the window is moved to the left it would move over a distance equal to one-third of its width as shown below.
The procedure consists of a loop which reads the keyboard input buffer for codes representing the arrow and diagonal keys. If a proper key is depressed then the view screen boundaries are updated and the window is redrawn. There is also a procedure to check the window boundaries to insure that the user has not moved off the drawing. The loop continues to ask the user to move the window until the ENTER key is pressed and the command terminates.

RESULTS: The results of the WINDOW command can be seen in Figure 11. This figure illustrates what would happen if the window was moved diagonally up and to the left several times before ending the command. A similar result would be expected for each of the other diagonal window movements as well as the horizontal and vertical window movements.

ZOOM COMMAND

DESCRIPTION: The ZOOM command is used to magnify a portion of the drawing to be viewed. When a drawing is first read in from a file, it is displayed in the viewing window so that the entire drawing can be seen. Since this does not allow the user to see finer details in the drawing, the ZOOM facility has been added to enlarge a small area of the drawing so that much more detail can be achieved.
Figure 11. Window Movement on a Zoomed Drawing.
After selecting the ZOOM command, the program prompts the user to select an Upper left corner, All, Half, or Twice. This is referring to the ZOOM modes available. The user may either specify a viewing rectangle to be enlarged to fill the screen by selecting the upper left and lower right corners of the section to be enlarged, or the user may zoom out to see the drawing by pressing the "A" key for all. In addition, the user may zoom out to see all of the drawing at one half of its current size by pressing the "H" key for half, or the drawing may be enlarged to twice its current size by pressing the "T" key for twice.

CODING: The ZOOM command is executed by procedure Zoom. After displaying the prompt to the user, the program reads in input characters from the keyboard and branches to one of four different sections with an extended if-then-else statement.

If an arrow key has been pressed the program assumes that a rectangle will be indicated to set the new viewing limits. The user is then prompted for the upper left, and lower right corners of this rectangle and the new viewing window is calculated from these values. If an A key is pressed then the current viewing window limits are set to the maximum drawing limits so that the entire drawing is displayed on the screen.
If an H key is pressed then the objects in the viewing window should be half as large as the current display. The viewing window limits are enlarged then so that the window size is twice as large as its previous size. Finally, if a T key is pressed, the viewing window is reduced so that the objects are displayed at twice their current size.

After the new viewing window limits have been calculated the screen is redrawn and the drawing crosshairs are positioned at the center of the screen.

**RESULTS:** By using the ZOOM command the objects can be displayed either larger or smaller by zooming in and zooming out. Of the four zoom modes available to the user, selecting a Viewing Rectangle and Zoom Twice are used for zooming in or enlarging the objects in the viewing window. The Zoom All and Zoom Half modes are used for zooming out or reducing the size of the objects displayed in the viewing window.

**MOVE COMMAND**

**DESCRIPTION:** The MOVE command is used to translate objects on a drawing. There are three possible ways that MOVE can be used: the user may move all of the objects in a drawing; the user may move an individual object in a drawing; or the
user may move several selected objects in a drawing. The maximum number of objects that can be selected and moved is 100.

The program prompts the user to select the object or objects to be moved, or All. If the "A" key is pressed then the entire drawing's contents are translated by a specified amount. However, if any of the arrow keys are pressed then the program assumes that the objects to be moved will be selected by the user. The program continues to prompt the user for new objects to be moved until the ENTER key is pressed, and then the user is asked to select the "From" point and the "To" point for the move. These indicate the direction and the distance for the translation. After the objects have been translated and the display is updated, the user is asked to move the objects again, and the process is repeated until the objects are in their desired location on the drawing.

**CODING:** The MOVE command is coded as procedure Moveobject. This procedure contains three subprocedures: Translateobject, Getdelta, and Addtomovebuffer. Translateobject is used to update the locations of the objects in the drawing database. It is given the object number and the X and Y displacement and it will find the object in the database and update its location. Subprocedure Getdelta is used to find the X and Y
displacement of the objects. This procedure asks the user for the "From" point and the "To" point for the move and the distance between these points is the displacement.

Since more than one object can be moved at one time there must be some way that the program can remember the objects that need to be moved. That is the function of subprocedure Addtomovebuffer. As each object is selected, it is added to a move buffer with this procedure. The move buffer is a first in-first out stack that is enlarged as each new object to be moved is selected. This stack contains the numbers of all the objects to be moved.

The actual moveobject procedure prompts the user for the move mode, either selecting objects or moving all. If the move all is desired, then the displacement is obtained using getdelta, and then the entire drawing is updated.

If the objects are to be selected, then the program begins a selection loop which prompts the user and then adds the selected object to the move buffer until all of the desired objects are selected. The objects to be moved are then erased from the screen, the drawing database is updated and then the objects are redrawn.

RESULTS: The MOVE command relocates an object or series of objects on the drawing along a given X and Y displacement. The user may continue to respecify this displacement until
the object or objects are located at a desired point on the drawing. An effective use of the MOVE command would be for the positioning of notes on a drawing. The user could place a series of text notes on a drawing using the TEXT command and then at a later point place them at a specific point in the drawing using the MOVE command.

ERASE COMMAND

DESCRIPTION: The ERASE command is used to delete or remove an object from the drawing. After selecting the ERASE command the user is prompted to select the object to be erased. An object may be selected by positioning the drawing crosshairs over the objects using the arrow keys, and then pressing the ENTER key. The selected object is then erased from the screen. The user is then asked if he wishes to restore the object. If a restoration is requested, the object is redrawn on the screen. If not, the object is removed from the drawing database, and the command ends.

CODING: The ERASE command is coded as procedure Erase. This procedure prompts the user to select the object to be erased and that object is searched for in the drawing database using procedure Selectobject. If the object selected exists in memory, it is erased from the screen. If the user wishes to restore the erased object, it is
redrawn on the screen using procedure Redrawobject. If a restoration is not desired, then the object data is removed from the drawing database using procedure Removefrombuffer.

RESULTS: Any object placed in the drawing can be erased. When the program asks the user to restore the object after it has been erased, it is important to note that this is the last chance that the user will have to retrieve this object. After the ERASE command ends, the object will be lost forever. Care must be exercised when using the ERASE command.

DISTANCE COMMAND

DESCRIPTION: The DISTANCE command is used to interact with the drawing to obtain distance measurements. It is the software equivalent of placing a ruler against a drawing to measure distances. After selecting the DISTANCE command, the user will be prompted to select a "From" point on the screen with the drawing crosshairs. The user is then requested to select a "To" point on the screen. This is the point to which you wish the distance to be measured. After locating the "To" point, the scaled distance between two points is displayed on the bottom line of the screen. The distance will continue to be displayed until any key is pressed to end the command.
CODING: The DISTANCE command is coded as procedure Getdist. This procedure prompts the user for the "From" point and "To" point and then calculates the distance between those points in drawing units (inches). This distance is then multiplied by the drawing scale to obtain scaled units. These scaled units are then displayed to the user.

RESULTS: The DISTANCE command displays the distance between any two selected points on the screen to the user. This displayed distance will remain on the screen until any key is pressed. The distance is then removed from the screen and the command ends.

AREA COMMAND

DESCRIPTION: The AREA command is used to interact with the drawing to obtain the area of any closed polygon created by selected points on the drawing. After selecting the AREA command, the user is prompted to select Area points. These are points on the drawing that will form the perimeter of the polygon whose area is to be measured. Up to 50 points may be selected. After the perimeter has been selected, the polygon's area is calculated in square units and displayed to the user. The displayed area will remain on the screen until any key is pressed.
CODING: The AREA command is coded as procedure Getarea. This procedure contains two sub-procedures: Getareapolygon and Calcarea. Getareapolygon is used to prompt the user for the area points and then places them in an array which is passed to procedure Calcarea. This procedure calculates the area enclosed by the area polygon in drawing units, and then converts this into scaled units by multiplying the area by the drawing scale squared. It is this scaled area that is displayed to the user.

RESULTS: Figure 12 illustrates an example of using the AREA command to calculate the area enclosed by an irregular set of objects. It must be noted that this area calculation will only be approximate, depending upon the location of the area polygon endpoints.

PLOT COMMAND

DESCRIPTION: The PLOT command is used to create a hard copy output of the drawing with a pen plotter. It is essentially a translator that reads each object from the drawing database, translates it into a proper plotter command, and sends that command to the pen plotter.

After selecting the Plot command the program will ask the user to choose a plotter pen number for each of the objects that can be in a drawing. This allows for multicolor plots, or plots with differing line weights.
Figure 12. Typical Area Calculation.
When the pens have been selected, the program proceeds to plot the drawing.

**CODING:** The PLOT command is coded as procedure Plot. Since each plotter has a different command language a separate plot procedure is needed for each different plotter. The two plotter drivers developed for this project were for the Hewlett Packard HP 7475A six-pen plotter, and the Houston Instrument DMP-29 eight-pen plotter. The procedures for these plotter drivers are stored in the files PLOTHP.PAS and PLOTHI.PAS. One of these two plot procedures is read into the program as an included file when the program is compiled. For more information on compile procedures, please refer to Appendix A. The Plot procedure contains eight main subprocedures responsible for the plotting of a drawing, Getpendata, Plotborder, Plotline, Plotext, Plotcircle, Plotarc, Plotrect, and Plotsymbol.

After selecting the Plot procedure, the user is prompted for all of the pen choices by procedure Getpendata. Once the pens have been selected, the plotter is initialized with paper size, and velocity data, and procedure Plotborder is called to plot the drawing border and titleblock information, finally, the plot procedure begins its object plot loop.
This loop reads in each object from the drawing database and sends that object to a particular procedure depending upon the object's type. Lines are sent to procedure Plotline, text is sent to procedure Plottext, etc. Each of these subprocedures further breaks down the object into components which can be drawn by the plotter. This loop repeats until all of the objects have been plotted and the command ends.

**RESULTS:** The PLOT command will produce a hard copy output of a drawing. Detailed examples of drawing plots are given in Appendix D. In addition, all figures included in this document were drawn with this software and plotted. Once plotting has begun, there is no provision for the user to stop the plotting. It is, therefore, important to make sure that the pens and paper have been properly loaded and the plotter is turned on before beginning the PLOT command.

**CHANGE LINE COMMAND**

**DESCRIPTION:** The CHANGE LINE command is used to change the current typesetting of a line that is in the drawing. Knowing the eight different line types allowed in the drawing, (Normal, Bold, Dashed, Right-of-Way, Centerline, Broken, Chainlink fence, Wood fence, and Wall), the user
may elect to change a selected line of one type to another type. For example, a Bold line may be changed to a Dashed line.

After selecting the CHANGE LINE command, the user is asked to select the line to be changed. Once it has been chosen, the program prompts the user to select the new line type from the list displayed. The current line is erased and then redrawn with the new type.

**CODING:** The CHANGE LINE command is coded as procedure `Chline`. This procedure prompts the user for the line to be changed with the Selectpoint and Selectobject procedures it then assigns an integer number to the new line type that is desired. The selected object is erased, the line in the drawing database is given the new line type number, and it is redrawn using procedure Redrawobject.

**RESULTS:** A line type can be changed from one type to another with this command. The CHANGE LINE command will only change the type of the particular line that is selected. It will not, for example, change all of the Bold lines in a drawing to dashed lines. Each change must be done individually.
CHANGE TEXT COMMAND

DESCRIPTION: The CHANGE TEXT command is used to change the size or weight of an alphanumerical text string in a drawing. Since five different text sizes are available to the user (0.11", 0.14", 0.21", 0.28" and 0.42"), and two different text weights are allowed (normal and bold), the user may change the size and weight of a piece of text using this command. For example, text that is 0.14" tall and has a normal weight, may be changed to 0.11" tall and given a bold weight.

After selecting the CHANGE TEXT command, the user is asked to select the text to be changed. The program displays the size and weight characteristics on the screen’s prompt line. The user is then asked to choose the new text size and weight. Once this is complete, the program erases the old text and redraws it with its new characteristics.

CODING: The CHANGE TEXT command is coded as procedure Chtext. This procedure prompts the user for the text to be changed with the Selectpoint and Selectobject procedures, and then assigns a set of numbers for the text’s size and weight characteristics. The selected text is updated with the new size and weight characteristics and then it is redrawn using procedure Redrawobject.
RESULTS: After the text has been selected, its current characteristics are displayed to the user as shown in the examples below.

[0.14]  [0.28B]

The example on the left is the message for normal text 0.14" tall and the example on the right is for bold text 0.28" tall. These characteristics remain displayed on the prompt line until any key has been pressed on the keyboard.

After the user selects the new height and weight characteristics, the text is redrawn on the screen, and the command terminates.

QUIT COMMAND

DESCRIPTION: The QUIT command is used to exit the drawing program and return to the operating system. After electing to QUIT, the user is asked, "Do you wish to save current drawing?" This is the last chance that the user has to save the drawing in the system's memory to a disk file. Once the program has ended, the drawing will be lost forever unless it is saved. After the drawing program ends, the operating system prompt is visible in the upper left-hand corner of the screen.
CODING: The main body of the drawing program consists of a loop that branches out to all of the available drawing commands. That loop is controlled by a boolean flag called Finished. Finished is set to be false at the beginning of the program, and is set at true only by selecting the QUIT command. Once Finished has been set to true, the loop terminates, and the program execution ends.

RESULTS: The QUIT command terminates the operation of the drawing program. After the program has ended, the only way to restart the drawing system is to invoke its starting command again. Since this causes the program to initialize all of the drawing memory that it can access, any information, whether it was a drawing, or garbage, is erased. Therefore it is imperative that the current drawing being worked on is saved before exiting the program.

F1 - CHANGE MENU COMMAND

DESCRIPTION: The drawing program contains two command menus. Any command may be selected from any menu, even if the command is not currently displayed on the menu. It is often useful to toggle back and forth between menus to serve as a reminder of the commands available and how to select them.
When the drawing program begins, the first menu is displayed on the screen. If the user presses F1, the first function key, this menu will be erased and the second menu will be displayed. If the key is pressed again, the first menu will be redisplayed.

CODING: The command menu is displayed on the screen by the procedure Displaymenu. This procedure uses a boolean flag called Menul to determine which menu to display. If Menul is true, then the first menu will be displayed, if not, the second menu is shown.

After pressing the F1 key the boolean variable Menul is complemented, the current menu is erased, and then the new menu is displayed.

RESULTS: Repeated pressing of the F1 key will toggle the display of the drawing command menus. The menu displayed, however, has no control over the program commands which can be selected. For example, even though the CHANGE TEXT command is displayed on the second menu, it can be selected while viewing the first menu. The menus, then, are simply a list of available commands that the user may select.

F2 - CHANGE COLOR COMMAND

DESCRIPTION: The color of the drawing currently being displayed can be changed by pressing F2, the second
function key. After starting the program, the drawing will be displayed as white. This can be changed to any one of fifteen colors by pressing the F2 key. Each time the key is pressed the color will change to the next available color in the list shown below.

White
Green
Cyan
Red
Magenta
Brown
Light Grey
Dark Grey
Light Blue
Light Green
Light Cyan
Light Red
Light Magenta
Yellow

This list is a continuous loop, so if the current color is Yellow and F2 is pressed again the color will return to White. Because of the resolution available on the IBM Color Graphics card, only one color can be displayed on the screen at one time.

**CODING:** The CHANGE COLOR procedure is coded as procedure Changecolor. The display color is accessed by the Pascal command Setforegroundcolor where the color is any integer between 1 and 15. The changecolor procedure takes the
current drawing color number, increments it by one. If the new drawing color number is less than 16 then the color is changed. If not, then the color number is set to 1.

RESULTS: Pressing the F2 function key causes the color of the screen to be changed through a looped list of available colors. The background of the drawing will always remain black, and the color of the objects displayed on the screen will change. It is recommended that the user select a color that is easy on the eyes, and that the color be changed occasionally to prevent eyestrain.

F9 - WINDOW INTERRUPT COMMAND

DESCRIPTION: The F9 key is a window interrupt function that can be called whenever the system is asking the user to select a point or an object on the screen. This window interrupt allows the user to perform a combination of zoom or window control functions from within the command in which it was called. The options available to the user in the interrupt routine are a \texttt{Zoomall}, to see the entire drawing; a \texttt{Zoomhalf} to see the drawing at one-half of its current size; a \texttt{Window centered} option, to center the viewing window about the current crosshair location; and a \texttt{Zoomtwice} option, to enlarge the drawing to twice its current size. In addition, each of the zoom options will center the new window around the current crosshair.
position. This allows the user to zoom into a specific point on the screen quickly and easily. After the interrupt routine ends, the program returns to the command in which the interrupt was called, and normal program execution continues.

CODING: The interrupt routine is coded as subprocedure Zoom within procedure Selectpoint. This interrupt routine is quite similar to the procedure selected by the ZOOM command, with the addition of the centering about the current crosshair location, and the lack of the zoom rectangle. Since it is a subprocedure within another program procedure, this window interrupt is not a true software interrupt from the system's point of view. However, from the user's reference frame there is no difference, and the procedure appears to be a true interrupt.

RESULTS: Suppose, for example, that the user desired to erase an object in the drawing, and selected the ERASE command. When the prompt to select the object to be erased is displayed on the screen, the user discovers that the object is not displayed in the viewing window, but is located on the drawing just past the right-hand side of the viewing window.
Without the use of window interrupt, the user would have to end the ERASE command, select the WINDOW command, move the window to the right, exit the WINDOW command, restart the ERASE command, and select the object to be erased that is now displayed on the screen. The use of the window interrupt greatly simplifies this problem.

To use the interrupt, the user simply moves the crosshairs to the right-hand edge of the screen, presses the F9 key, and chooses the centered option. The screen is redrawn with its center at the indicated crosshair location, and the object to be erased is displayed. The interrupt returns to the point at which it was called within the ERASE command, and the user may conveniently select the object to erase it.
CHAPTER III
USER INTERFACE

Introduction

While a task like word processing is very well suited for microcomputers because of its use of the keyboard and text display, the production of engineering drawings is not. For a word processor, most input uses the keyboard directly and the operator is usually well trained in typing and keyboard input. A draftsperson's primary tools, however, consist of pens, scales, and triangles. It is highly unlikely that the user has spent any length of time behind a keyboard, and much less likely that they possess any typing skills. It is, therefore, critical that a computerized drafting system contain a well-designed user interface for it to be effective in increasing productivity.

The user interacts with the software through a conceptual model of the program's operation that the user creates. This model is what the user sees and usually does not reflect what really happens in the program with the variables, stacks, and procedures. It is very important
that the user interface supports the creation of a simple conceptual model that predicts the behavior of the system. The simpler the model, the easier the program is to use.

It is not an easy task to create a workable model of a drafting system for engineering drawings. The primary reason for this is because the users are out of their element. For the word processor mentioned earlier, it is very easy to display text on a computer screen. The computer, keyboard, and screen have been designed for rapid analysis and display of numerical and text data.

The display of graphics, however, is not as easy on the computer. Redrawing a picture at an enlarged size requires far more time than displaying a page of text on the screen. For this reason, the model for a drafting system must be carefully designed.

The user has been trained to look at an entire drawing at one time to gain an overall impression of its elements in relation to each other. On a computer, however, he is forced to see the entire drawing at a very poor resolution, or a small section of the drawing at very good resolution. The eye movement and focusing commands, that are automatic when a person looks at a drawing on paper, must now be designed into the drafting system.
To do this, two important factors must be considered, ease of use and time. Since it requires no effort for a drafter to look at a detail on the left side of the drawing and then focus on a detail on the opposite side, the corresponding commands to do this with computer system should be just as easy to use. In addition, response time is a very important factor. It requires almost no time at all for a person to focus their eyes from one detail to another on paper, but it may require several seconds for a computer to erase its current screen and redraw the drawing with new a detail visible. If the system does not respond quickly, then productivity will suffer and the system will not be considered easy to use.

The drawing program's operation has been designed with these requirements in mind. The user interacts with the program by selecting simple two letter commands from a menu. Each command prompts the user for the next required input. The user is also allowed to exit a command without finishing it if an error has been made. Finally, all changes to a drawing are immediately updated on the screen for the user to see.

This chapter will discuss several concepts and their relationship to the user interface of the drawing program. The concepts discussed include screen layout, command selection, keyboard pointing, and workplace layout.
Screen Layout

The display screen is divided into four primary sections in the drawing program: the File and View Line; the Command Menu; the user Prompt Line; and the Viewing Window. Each of the sections, displaying different kinds of information to the user and their locations on the screen, can be seen in Figure 13.

The File and View Line contains information about the drawing being edited, and its view. Specifically, it lists the name of the file being worked on, the magnification of the drawing displayed in the Viewing Window and the drawing scale. In the File and View Line shown in Figure 13, the name of the file currently being worked on is TESTNAME, the current window magnification is 4, and the drawing scale is 1" = 30 ft.

The window magnification number is the ratio of the size of the current drawing window to the entire drawing. A window magnification of 4 indicates that the objects currently in the window are 4 times larger than they would be if the entire drawing was displayed. When the entire drawing is displayed in the Viewing Window, the magnification is 1. The maximum allowable window magnification is 30.
Figure 13. Screen Layout for User Interface.
The drawing scale tells the user how the distances entered for objects in the drawing will be plotted. In the given example scale of 1" = 30', a distance represented as 30 feet on a drawing would be plotted as 1 inch on the paper. Separate sections of drawings may be drawn to different scales, but only the current scale being used will be displayed on the File and View Line. While the window magnification changes as the user zooms in and zooms out of a drawing, the scale remains constant, unless it is changed with the SCALE command.

The Command Menu displays a list of the drawing commands available to the user. The drawing program contains two command menus allowing almost fifty drawing commands, and the user may change from one menu to the other by pressing the F1 key. The command selected by the user does not have to be displayed on the current menu. For more information on the drawing commands, please refer to the following section discussing Command Selection.

The user Prompt Line is the line on the display where all text interaction between the user and the system occurs. Whenever the system prompts the user for input, it is done on this line. In addition, all input that the user enters to the system is also displayed on this line. For example, when the system desires the user to enter the
radius of a circle that is being created, it will prompt
the user with a message to "Enter Circle Radius:". This
message will appear on the prompt line. The user in turn
enters a number to represent the desired circle radius.
The number entered is displayed on the prompt line
immediately after the prompt so the result would be:

Enter Circle Radius: 50

Finally, the Viewing Window is used to graphically
display the drawing being worked on. This is the window
through which the drawing can be seen. All changes to a
drawing visible in the Viewing Window will be updated as
they occur, so that the user can evaluate the results of
his work. The magnification of this window can be changed
using the ZOOM command.

In conclusion, the Screen Layout in the drawing
program has been designed to display a large amount of
information in a very small area. The main focus of the
user's field of view is directed toward the Viewing Window
of the Screen. The prompts from the system can be found
immediately below the Viewing Window for quick glances
between the two areas. The drawing commands which are used
less frequently are located to the right of the Viewing
Window. Finally, the seldom used information including
filename, window size, and drawing scale, are displayed along the top of the screen above the Viewing Window.

Command Selection and Keyboard Control

The user controls the operation of the drawing program by selecting from a list of available drawing commands displayed on the Command Menu. A listing of the available commands on their respective menus can be found in Figure 14. For a discussion of each of these individual commands please refer to Chapter II.

Each command is selected by pressing a two-letter combination shown as the capital letters in the command as it is displayed in the drawing Command Menu. For example, to select the OFFSET Command, the user will notice that it is displayed in the menu as OffSet. Therefore, to execute the command, the user is required to press the O and S keys on the keyboard.

Two letter mnemonics were chosen for drawing commands because of their ease in typing. Some of the other options considered were typing in the entire command, moving a command pointer up and down through the menu to select a highlighted command, or selecting function keys for commands. Typing the entire command was eliminated because
<table>
<thead>
<tr>
<th>DWg</th>
<th>ChLine</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOrder</td>
<td>ChText</td>
</tr>
<tr>
<td>SCale</td>
<td></td>
</tr>
<tr>
<td>COgo</td>
<td></td>
</tr>
<tr>
<td>TeXt</td>
<td></td>
</tr>
<tr>
<td>View</td>
<td></td>
</tr>
<tr>
<td>LINE</td>
<td></td>
</tr>
<tr>
<td>Circle</td>
<td></td>
</tr>
<tr>
<td>ARC</td>
<td></td>
</tr>
<tr>
<td>REct</td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td></td>
</tr>
<tr>
<td>FILlet</td>
<td></td>
</tr>
<tr>
<td>Offset</td>
<td></td>
</tr>
<tr>
<td>EXTend</td>
<td></td>
</tr>
<tr>
<td>MoDIFY</td>
<td></td>
</tr>
<tr>
<td>BReak</td>
<td></td>
</tr>
<tr>
<td>WIndow</td>
<td></td>
</tr>
<tr>
<td>ZooM</td>
<td></td>
</tr>
<tr>
<td>MoVe</td>
<td></td>
</tr>
<tr>
<td>ERase</td>
<td></td>
</tr>
<tr>
<td>DIST</td>
<td></td>
</tr>
<tr>
<td>AreA</td>
<td></td>
</tr>
<tr>
<td>PLOT</td>
<td></td>
</tr>
<tr>
<td>QUIT</td>
<td></td>
</tr>
</tbody>
</table>

(a) Primary Menu

(b) Secondary Menu

Figure 14. Drawing Commands on the Command Menus.
of the lack of typing experience of the user. The pointer to select a highlighted command was ruled out because of the time it would take to move the pointer from the first command in the list to the last. It would be much faster to simply press two keys. Although the function key option was a good idea in terms of required keystrokes, it was eliminated because of the space on the screen required to describe the command to function key mapping. For example, displaying \texttt{F10=Fillet} would require a minimum of ten spaces on the screen for the Command menu. This is almost double the six spaces currently used. Therefore, if the command menu is larger, then the Viewing Window would have to be smaller, and that is unacceptable.

The function keys are used, however, to execute some system commands that are not displayed on the menu. These system commands include: \texttt{F1} - to change the command menu currently being displayed; \texttt{F2} - to change the color displayed on the screen; and \texttt{F9} - to change the Viewing Window while executing a command.

After a drawing command in the command menu has been selected, all other commands in the menu are erased so that the user always knows what command the system is currently executing. This is quite useful for the inexperienced user who is not yet familiar with all of the commands and their operation.
Keyboard Pointing

All pointing and object selection on a drawing is accomplished by moving a pair of drawing crosshairs within the Viewing Window. These crosshairs are moved by pressing the arrow keys on the computer's numeric keypad. To use the arrow keys you must disable the NUMBER LOCK key. Each of the crosshair control keys highlighted in Figure 15 can move the drawing crosshairs in a different direction. The arrow keys are used to move the crosshairs left, right, up and down, and the HOME, END, PGUP, and PGDN keys are used to move the crosshairs diagonally.

The crosshairs have been designed to accelerate across the screen so that the longer you hold a key down the faster it moves along the screen. As soon as another key is pressed the crosshair speed is reset to its minimum. This allows for faster crosshair movement without having to fumble with crosshair velocity settings. The drawing crosshair also has a wrap-around feature which allows you to move it off one side of the Viewing Window and it will appear on the other side. This allows for faster movement from one side of the Viewing Window to another.

To point to a specific location on the screen, the user must move the drawing crosshairs to that point by
CROSSHAIR CONTROL KEYS

Home = Diagonally UP and LEFT
↕ = UP
Pg Up = Diagonally UP and RIGHT
← = LEFT
→ = RIGHT
End = Diagonally DOWN and LEFT
↓ = DOWN
Pg Dn = Diagonally DOWN and RIGHT

Figure 15. Crosshair Movement with the Numeric Keypad
pressing the crosshair control keys. Once the crosshairs are at the desired location on the drawing, the user simply presses the ENTER key to indicate that the pointing is complete. The program will then process this input and continue the command.

To select a specific object in the drawing, the user must move the crosshairs so that they are located on or very close to the desired object with the crosshair control keys. After the crosshairs are properly positioned, the user must press the ENTER Key and the object is selected from the drawing database. Objects cannot be selected if they are not displayed in the Viewing Window.

Pointing on a drawing is a very simple task. Only the crosshair control keys and the ENTER key are required to move the crosshairs in the drawing. All other Keyboard input is ignored. If the program prompts the user to select a point or an object and the user wants to terminate the command, he may do so by pressing the ENTER key without having pressed any of the crosshair control keys.

Finally, since the crosshair moves at a fixed increments on the screen, the user may be able to point and select objects more accurately by zooming in on a portion of the drawing. If moving the crosshair one increment when
viewing the entire drawing amounts to one-tenth of an inch, then when the user is working at window magnification 10, that increment will be reduced to one one-hundredth of an inch, and the accuracy will be increased by a factor of 10.

Workplace Layout

Since the workplace layout cannot be designed into the program, it is an often overlooked feature that can lead to decreased productivity with the drawing program. Too often a computer is located in a small area that has only enough room for the computer and a small amount of desk space for some paper documents. It is the purpose of this section to make some recommendations for a working configuration of the computer and its peripherals to function as a drafting workstation.

This workstation consists of four principle components: the computer, including the keyboard, system unit and display monitor; the printer; the pen plotter; and applicable books and manuals. In addition, workspace is required for simultaneous viewing of several typed documents or papers, as well as a space for viewing larger existing drawings. A workstation containing all these features can be seen in Figure 16.
Figure 16. Sample Drafting Workstation Layout
The key to an efficient drafting workstation design is to locate the equipment that is used the most, closest to the user, while at the same time providing enough workspace for viewing drawings and reference material. This clearspace concept is very important. If a user has to spend time sorting through papers and drawings because he does not have enough workspace to lay them side by side, productivity will suffer just as much as it would with a poorly designed program. To achieve maximum productivity with a drafting system, it is just as important for the user to design an efficient workspace layout as it is for the system designer to create an efficient program.
CHAPTER IV
CONCLUSIONS

System Implementation

The drawing program was developed for, and is currently being used at, the offices of Geodesy Professional Services. This company is a relatively small Land Surveying Office. Their drawings primarily range in three sizes: 8.5" x 14", 11" x 17", and 24" x 36". Of these three sizes, it is estimated that 25% of the drawings produced are 8.5" x 14"; 25% are 11" x 17", and, 50% of these drawings are 24" x 36". The drafting system, when introduced, had the potential to produce 50% of the drawings created by the office.

Training of the employees consisted of approximately one week of instruction with the first two days demonstrating the system and its commands, and the latter three days consisting of supervised practice on the system. This was just to obtain a working knowledge of the system and its commands. It is estimated, on the average, that it took approximately three months to reach a point where the drawing system's maximum productivity could be reached.
Even though the system had a potential to be used for 50% of the office's drawings, its current use is closer to 30% to 40% of the total drawings produced. This is due to several factors. First of all, as was expected, the time from blank paper to finished drawing is about one and one-half times as long with the drawing system, as it would be with an experienced draftsman. Given improvements in the efficiency of the software and advances in the speed of the hardware, it is expected that this gap will be closed within three to five years.

The drawing system's time savings comes from its ability to copy and change existing drawings quickly. Throughout the lifetime of a construction project, many drawings must be repeatedly updated and produced. For example, for a single building, the drawings produced throughout the job may include a Boundary Survey, a Site Plan, a Foundation Survey, and a Final Survey. Each of these drawings are usually enhancements of, or modifications to, a previous drawing.

When done by manual methods, each drawing must be redrawn from scratch. Using the drafting system however, the current drawing can be copied and improvements can be made to that copy. When the drawing is finally plotted it will have been produced in a fraction of the time that it takes to produce the drawing manually.
The drawings produced with the drafting system at Geodesy Professional Services are only those which will require repeated updating. That is why only 30 - 40% of the possible 50% of total drawings produced are drawn using the drafting system. Once again, as the system is refined, and hardware advances are introduced, it is expected that this number will increase.

Implementation Problems

The major problem that was discovered in the full-time use of the system was the speed at which the drawing system worked. The primary complaint was the amount of time required to redraw the display using a ZOOM or WINDOW command, and the time it takes to select an object on the screen. Because of the design of the system, both of the complaints are caused by the same problem, the system's redrawobject procedure.

This procedure goes through each object in the database, checks to see if it is within the drawing window, clips the object to the window boundary, and then displays the object on the screen. It repeats this procedure with every object in the database, until the entire drawing has been checked. The speed problem lies in the calculations for clipping, and in the display of the objects on the
screen. No further code optimizations can be seen, so further improvements must lie in hardware upgrades, or database modifications.

Since the problem is caused by trying to display too much information in too short of a period of time, it is suggested that some sort of layering tag be placed on the objects in the drawing database to limit the amount of information that needs to be displayed at one time. For example, the user could turn off the layer that contained all of the drawing text. In this way, no text would be displayed until that layer was turned on again, and the amount of objects to be displayed in a redraw would be reduced, thereby reducing the time required.

Another complaint with the system was its lack of color. Since the system is designed to utilize the IBM Personal Computer's Color Graphics Adapter in its high resolution mode, the user can only display one color on the screen at a time. This makes it difficult to display contrasts between objects on the screen, and can slow productivity. It is especially critical as the number of objects in a drawing increases. Possible solutions for this problem include switching to one of the lower resolution modes available. This would allow the user to display drawings with four (4) colors but they would be
displayed at a lower pixel resolution. (320 x 200 as opposed to 640 x 200). Another solution would be to make a hardware upgrade to a more advanced graphics card that could display more colors without the resolution loss. Both of these solutions, however, would require recoding of the system.

Possible Improvements

In addition to the previously discussed improvements including Layering, Display Speed Reduction, and Multi-color Displays, some suggested improvements to the system include the addition of a user-defined symbol file. This file would allow the user to create symbols which can be stored, edited, and copied. This would let the user save collections of objects as symbols that could be recalled and placed into a drawing at any location or orientation. Additional drawing commands to accomplish this would include Make Symbol, Copy Symbol, and Edit Symbol.

Another suggested improvement would be a file translator function that could convert drawing files into a neutral format that could be read by other drawing systems.

A third improvement would be a function that would allow a block of ASCII text, stored as a file, to be read into the drawing program and added to the drawing as text
data. In that way, drawing notes could be created and edited on a word processor and the finished result could be added to the drawing.

Finally there always exists the opportunity to expand the program to access the multitude of available input and output devices including digitizers, mice, lightpens, additional pen plotters, and dot matrix printer plotting, as well as drivers for laser printers.
APPENDIX A

PROGRAM DEVELOPMENT ENVIRONMENT

The program was developed using TURBO PASCAL Version 3.0 which closely follows the definition of Standard Pascal as defined by K. Jenson and N. Wirth in the Pascal User Manual and Report. Any minor differences between TURBO PASCAL and Standard Pascal are described in Appendix D of the TURBO PASCAL Reference Manual listed in the bibliography.

In addition to the TURBO PASCAL Compiler, the software utilizes a series of source code extensions of TURBO PASCAL known as the TURBO GRAPHIX TOOLBOX to add additional graphics procedures and functions. Both TURBO PASCAL and TURBO GRAPHIX TOOLBOX are registered trademarks of Borland International Inc.

Since the TURBO GRAPHIX TOOLBOX contained many functions and procedures that were not used in this project, it was extensively edited to reduce its overall size. This was a concern because of TURBO PASCAL’s limit of only 64 K of object code for a program. The original TURBO GRAPHIX TOOLBOX consisted of over 2,000 lines of source code. This was edited to slightly under 1,200 lines to reduce the program’s object code size.
In addition to the removal of non-essential functions, the clipping procedures in the TURBO GRAPHIX TOOLBOX were replaced with procedures that were developed to be more applicable to the floating point database used in this project. This modified version of the TURBO GRAPHIX TOOLBOX is included in the program listings in Appendix C under the source code file name TGUTIL.SYS. This is referred to as the graphics utility module.

The remaining source listings in Appendix C include the files SCREEN.PAS and SCREEN2.PAS. These files make up what is known as the program's drawing module. As mentioned before, the graphics utility module is contained in the source file TGUTIL.SYS. Two plotter drivers are also included in the source code, PLOTHP.PAS, for the Hewlett Packard 7475A six-pen plotter and PLOTHI.PAS for the Houston Instruments DMP-29 eight-pen plotter.

The main source code file SCREEN.PAS reads in the additional files, TGUTIL.SYS, SCREEN2.PAS, and either PLOTHP.PAS or PLOTHI.PAS, at compile time.

The computer creates a set of object code files consisting of a main file and several overlay files. An explanation of the object code files and their overlays can be found in Appendix B.
To run the drawing program, it is necessary to initialize the asynchronous communications port COM1 on the computer with the proper baud rate and control parameters that are required by the plotter. After the communications port has been initialized, the program may be started by entering the command SCREEN. This begins the execution of the file SCREEN.COM. To perform both of these functions with a single command it is recommended that the user create a batch file that contains the commands to initialize the communications port, and begin the program's execution.
APPENDIX B

PROGRAM OVERLAY STRUCTURE

To work around TURBO PASCAL's small memory mode which allows a maximum of 64K of object code, the program used an overlay system. This technique collects a number of subprograms (procedures and functions) into one or more files separate from the main program file. These subprograms are then loaded automatically, one at a time, into the same area in memory, as they are needed. There is, however, a space/time tradeoff.

Since these portions of the program must be read in from a disk file, there is a slight delay in the execution of the program. The placement of the overlay procedures is critical to not force the user to wait on the program as the overlay is being read from the disk. To prevent this lag in the drawing program, overlay files are being read while the user is usually preoccupied with command selection.

Figure 17 is a diagram which illustrates the overlay structure used for the drawing program.
Figure 17. Program Overlay Structure.

All remaining Procedures and Functions that are not listed above, are included in the file SCREEN.COM.
APPENDIX C

PROGRAM SOURCE CODE LISTINGS

This Appendix contains the listings for the files SCREEN.PAS, SCREEN2.PAS, PLOTHP.PAS, PLOTHI.PAS, and TGUTIL.SYS.

The files SCREEN.PAS and SCREEN2.PAS make up the program's drawing module. The files PLOTHP.PAS and PLOTHI.PAS present examples of two different plotter drivers. TGUTIL.SYS is an edited and modified version of the code provide in the TURBO GRAPHIX TOOLBOX package. This is used as the program's graphics utilities module.
SOURCE LISTING FOR FILE

SCREEN.PAS
program draw;  (* This loads the graphics utilities contained *)
{$I tgutil.sys}  (* in the TURBO GRAPHIX TOOLBOX Ver 1.02. *)
(* Program Compiled using TURBO PASCAL Ver 3.0 *)
const pixxmin=0; pixxmax=575; pixymin=10; pixymax=190; maxobj=500; maxtext=200;
type dwnentity=record
desc:integer;  (* Record type for each object in a dwg *)
code:integer;
data:array[1..6] of real;
end; (*record*)
line:string[80];
txtln:string[80];
onarray=array[0..maxobj] of integer;
textarray=array[1..maxtext] of txtln;
objectarray=array[0..maxobj] of dwnentity;
filename:string[10];
var object:objectarray;
textline:textarray;
onscreen:onarray;
finished,menu:boolean;
dwgfilenname:filename;
scale,colorvar:integer;
maxworldx,maxworldy,minworldx,minworldy,windowsz,crossx,crossy,
currentminx,currentminy,currentmaxx,currentmaxy:real;
{$I screen2.pas}  (* will read in utility procedures *)
{$I plothp.pas}  (* reads in plot procedure *)
overlay procedure panwindow(var object:objectarray;var onscreen:onarray;
    var crossx,crossy,minx,maxx,miny,maxy:real);
var ch:char;
diffx,diffy:real;
done,error:boolean;
begin
erasemenu;
gotoXY(74,18);write('Window');
if windowsz=1 then begin
    beep;
    writeprompt('CANNOT MOVE - WINDOW ALREADY DISPLAYED AT MAX SIZE');
delay(1500);
end else begin
    writeprompt('Press ARROWS to move window or ENTER to end');
diffx:=(maxx-minx)/3;
diffy:=(maxy-miny)/3;
done:=false; error:=false;
repeat
    read(kbd,ch);
    if (ch=#27) and keypressed then begin
        done:=true;
    end else begin
        case ch of
            #75:begin
                minx:=minx-diffx; maxx:=maxx-diffx; (*left*)
            end;
            #77:begin
                minx:=minx+diffx; maxx:=maxx+diffx; (*right*)
            end;
            #72:begin
                miny:=miny-diffy; maxy:=maxy-diffy; (*up*)
            end;
            #80:begin
                miny:=miny+diffy; maxy:=maxy+diffy; (*down*)
            end;
            #71:begin
end;
minx:=minx-diffx; maxx:=maxx-diffx;  
miny:=miny-diffy; maxy:=maxy-diffy;  
end;

e73:begin
minx:=minx+diffx; maxx:=maxx+diffx;  
miny:=miny+diffy; maxy:=maxy+diffy;  
end;

e97:begin
minx:=minx-diffx; maxx:=maxx-diffx;  
miny:=miny+diffy; maxy:=maxy+diffy;  
end
else begin
beep;
error:=true;
end;
end; {case}
if not error then begin
checkwindow(minx,maxx,miny,maxy);
changeresizewindow(crossx,crossy,minx,maxx,miny,maxy);
end;
writeprompt('Press ARROWS to move window or ENTER to end (if arrow pressed)
if ch=#13 then done:=true;
until done;
end;  
end; {panwindow}

--------------------------------------------------------------------------------
overlay procedure zoom(var object:objectarray;var onscreen:onscreenarray;
var crossx,crossy,minx,maxx,miny,maxy:real); 
var move,done:boolean;
ch:char;
begin
done:=false;
erase cursor;
gotoxy(74,19);write('Zoom');
writeprompt('Select UPPER LEFT Corner, All, Half, or Twice');
read(kbd,ch);ch:=upcase(ch);
if (ch='#27) and keypressed then begin
read(kbd,ch);
selectpoint('Select UPPER LEFT Corner :',object,onscreen,
minx,maxx,miny,maxy,crossx,crossy;x1,y1,move);
displaymark(x1,y1); 
selectpoint('Select LOWER RIGHT Corner :',object,onscreen,
minx,maxx,miny,maxy,crossx,crossy;x2,y2,move);
if move then begin
displaymark(x2,y2);
if abs(x2-x1)<abs(y2-y1) then x2:=x1+4*(y2-y1)/3
else y2:=y1+3*(x2-x1)/4;
end; {if move}
end {not all}
else if ch='A' then begin {zoom all}
x1:=minworldx; y1:=minworldy; x2:=maxworldx; y2:=maxworldy;
end
else if ch='H' then begin
if windowsz>=2 then begin
x1:=minx-(maxx-minx)/2;
y1:=miny-(maxy-miny)/2;
x2:=maxx+(maxx-minx)/2;
y2:=maxy+(maxy-miny)/2;
checkwindow(x1,x2,y1,y2);
end;
else if ch='L' then begin
if windowsz>=2 then begin
x1:=minx-(maxx-minx)/2;
y1:=miny-(maxy-miny)/2;
x2:=maxx+(maxx-minx)/2;
y2:=maxy+(maxy-miny)/2;
checkwindow(x1,x2,y1,y2);
end   begin
    x1:=minworldx; y1:=minworldy; x2:=maxworldx; y2:=maxworldy;
    end; {else}
end
else if ch='T' then begin
    x1:=minx+(maxx-minx)/4;
    y1:=miny+(maxy-miny)/4;
    x2:=maxx-(maxx-minx)/4;
    y2:=maxy-(maxy-miny)/4;
    checkwindow(x1,x2,y1,y2);
end
else done:=true;
if ((y2 <> y1) and (x2<>x1)) and not done then begin
    minx:=x1; miny:=y1; maxx:=x2; maxy:=y2;
    windowsz:=(maxwolrdx-minworldx)/(maxx-minx);
    if windowsz>30 then begin
        windowsz:=30;
        xcent:=(minx+maxx)/2; ycent:=(miny+maxy)/2;
        minx:=xcent-((maxworldx-minworldx)/windowsz)/2;
        miny:=ycent-((maxworldy-minworldy)/windowsz)/2;
        maxx:=minx+(maxworldx-minworldx)/windowsz;
        maxy:=miny+(maxworldy-minworldy)/windowsz;
    end;
    changewindow(crossx,crossy,minx,maxx,miny,maxy);
    titles(dwfile,windowsz,scale);
    redraw(object,onscreen);
    crossx:=(maxx+minx)/2; crossy:=(maxy+miny)/2;
    end; {if window}
end; {zoom}

overlay procedure addsymbol(var minx,maxx,miny,maxy,crossx,crossy:real; var object:objectarray; var onscreen:onarray);
var xstart,ystart,xend,yend,xbrk,ybrk,angle:real;
move,error: boolean;
stype: integer;
ch:char;
begin
    error:=false;
    errormenu;
    gotoxy(74,12);write('SYmbol:');
    repeat
        writeprcmpt('Enter Symbol type: N=north C=circ:le B=box A=arrow X=cr:oss '); read(kbd,ch); ch:=upcase(ch);
        case ch of
            'N':stype:=1;
            'C':stype:=2;
            'B':stype:=3;
            'A':stype:=4;
            'X':stype:=5;
            '#13':stype:=0;
        end; {case}
    until ch in ['N','C','B','A','X',#13];
    if stype<>0 then begin
        selectpoint('Select Symbol LOCATION :',object,onscreen,minx,maxx,miny,maxy,crossx,crossy,xstart,ystart,move);
        if ch = 'N' then begin
            displaymark(xstart,ystart);
            selectpoint('Select Symbol DIRECTION :',object,onscreen,minx,maxx,miny,maxy,crossx,crossy,xend,yend,move);
            if move then begin
                angle:=inverseazimuth(xstart,ystart,xend,yend);
                addtobuffer(object,onscreen,stype,xstart,ystart,xend,yend,angle,0);
                displaysymbol(stype,xstart,ystart,xend,yend,angle,0);
            end;
        end {north}
else if ch = 'A' then begin
  displaymark(xstart,ystart);
  selectpoint('Select BREAK Point : ',object,onscreen,minx,maxx,mini,maxy,
  crossx,crossy,xbrk,ybrk,move);
  if move then begin
    displaymark(xbrk,ybrk);
    selectpoint('Select BREAK Point : ',object,onscreen,minx,maxx,mini,maxy,
    crossx,crossy,xend,yend,move);
    if move then begin
      addtobuffer(object,onscreen,6,stype,xstart,ystart,xend,yend,move);
      displaysymbol(stype,xstart,ystart,xend,yend,xbrk,ybrk);
      end;
    end;
  end;
end; {arrow}
else begin {one pt input}
  repeat
    addtobuffer(object,onscreen,6,stype,xstart,ystart,xend,yend,move);
    displaysymbol(stype,xstart,ystart,xend,yend,move);
    selectpoint('Select Symbol LOCATION : ',object,onscreen,minx,maxx,mini,maxy,
    crossx,crossy,xstart,ystart,move);
    until not move;
  end;
end; {stype<>0}
end; {addsymbol}

overlay procedure addline(var minx,maxx,mini,maxy,crossx,crossy:real;
  var object:objectarray; var onscreen:onscreenarray);

begin
  erasemenu;
  gotoXY(74,8);write('LiNe');
  repeat
    error:=false;
    writeprompt('Select Line TYPE : N B D R C K X O W --> ');
    read(kbd,ch);
    ch:=toupper(ch);
    case ch of
      'N':lntype:=1;
      'B':lntype:=2;
      'D':lntype:=3;
      'R':lntype:=4;
      'C':lntype:=5;
      'K':lntype:=6;
      'X':lntype:=7;
      'O':lntype:=8;
      'W':lntype:=9;
    else begin
      error:=true;
      beep;
      end;
    end;
  until not error;
  x2:=0; y2:=0;
  selectpoint('Select FROM Point : ',object,onscreen,minx,maxx,mini,maxy,crossx,crossy,x1,y1,move);
  displaymark(x1,y1);
  repeat
    if (x2<>0) and (y2<>0) then begin
      x1:=x2; y1:=y2;
      end;
    repeat
      error:=false;
    until not error;
  end;
writeprompt('Select TO Point or Enter ANGLE (ie A45) :');
read(kbd,ch); ch:=upcase(ch);
if (ch='27) and keypressed then begin (To pt will be pointed to)
    read(kbd,ch); (remove second ch from input buffer)
    selectpoint('Select TO Point :',object,onscreen,
        minx,maxx,miny,maxy,crossx,crossy,x2,y2,move);
    displaymark(x2,y2);
    if (x1=x2) and (y1=y2) then begin end
    else begin
        addtobuffer(object,onscreen,1,intype,x1,y1,x2,y2,0,0);
        displayline(intype,x1,y1,x2,y2);
    end; (if line drawn);
end (selected To pt)
else if ch='A' then begin (angle & dist will be entered)
write(ch);
    readln(input);
    convertreal(input,angle,error);
    if not error then begin
        angle:=correct360(angle); (conv from neg to pos)
        angle:=correct360(450-angle); (conv to azimuth reference)
        getreal('Enter LINE LENGTH :',dist,done);
        if not done then begin
            dist:=convertscale(dist);
            calcnextpt(0,0,0,x1,y1,0,angle,dist,x2,y2);
            addtobuffer(object,onscreen,1,intype,x1,y1,x2,y2,0,0);
            displayline(intype,x1,y1,x2,y2);
            if ptonscreencrossx=crossx; crossy=crossy;
        end (pt onscreen)
        else begin
            crossx:=(minx+maxx)/2; crossy:=(miny+maxy)/2;
        end (pt not on screen)
        end (not done)
    else begin (else end routine)
        error:=false;
        move:=false;
        end (else)
end (entered angle)
else begin
    if ch='13' then begin
        writeprompt('Invalid Input'); beep; delay(500);
        end; (bad input)
    error:=false; (else end routine)
    move:=false;
    end (else)
until not error;
until not move;
end (addline)

overlay procedure addcircle(var
    minx,maxx,miny,maxy,crossx,crossy:real;
    var object:objectarray; var onscreen:onarray);
var
    xcent,ycent,xon,yon,radius:real;
    ch:char;
    move,error: boolean;
    input:line;
begin
    erasemenu;
    gotoxy(74,9); write('Circle');
    selectpoint('Select CENTER POINT :',object,onscreen,minx,maxx,miny,maxy,
        crossx,crossy,xcent,ycent,move);
    displaymark(xcent,ycent);
    writeprompt('Select POINT on CIRCLE or Enter RADIUS :');
    read(kbd,ch);
    if (ch='27) and keypressed) then begin
        read(kbd,ch); (remove 2nd ch from buffer)
selectpoint('Select POINT on CIRCLE :', object, onscreen, minx, maxx, miny, maxy, crossx, crossy, xon, yon, move);
if move then begin
  displaymark(xon, yon);
  radius:=invdist(xcent, ycent, xon, yon);
  addtobuffer(object, onscreen, 3, 0, xcent, ycent, radius, 0, 0, 0);
  displaycircle(xcent, ycent, radius);
end; {if move}
end {if selecting point on circle}
else begin
  write(ch);
  readln(input);
  input:=concat(ch, input);
  convertreal(input, radius, error);
  if (not error) and (radius > 0) then begin
    radius:=convertscale(radius);
    addtobuffer(object, onscreen, 3, 0, xcent, ycent, radius, 0, 0, 0);
    displaycircle(xcent, ycent, radius);
  end; {not error}
end; {else enter radius}
end; {addcircle}

overlay procedure addarc(var minx, maxx, miny, maxy, crossx, crossy: real;
var object: objectarray; var onscreen: onarray);
var xstart, ystart, xend, yend, xcent, ycent, angle,
    radius, midx, midy, chordist, offset: real;
    ch: char;
    move, error, ccw: boolean;
    input: line;
begin
  error:=false;
  ccw:=false;
  radius:=0;
  erasemenu;
gotoxy(74, 10);write('ARC ');
selectpoint('Select START POINT :', object, onscreen, minx, maxx, miny, maxy,
  crossx, crossy, xstart, ystart, move);
displaymark(xstart, ystart);
selectpoint('Select END POINT :', object, onscreen, minx, maxx, miny, maxy,
  crossx, crossy, xend, yend, move);
if move then begin
  displaymark(xend, yend);
  writeprompt('Select CENTER POINT or Enter RADIUS : ');
  read(kbd, ch);
  if (ch = #27) and keypressed then begin
    read(kbd, ch); {remove 2nd ch from input buffer}
    selectpoint('Select CENTER POINT :', object, onscreen, minx, maxx, miny, maxy,
      crossx, crossy, xcent, ycent, move);
    if move then begin
      displaymark(xcent, ycent);
      angle:=angleright(xstart, ystart, xcent, ycent, xend, yend);
      addtobuffer(object, onscreen, 4, 0, xstart, ystart, xcent, ycent, angle, 0);
      displayarc(xstart, ystart, xcent, ycent, angle);
    end {if move}
  end {if selecting centerpoint}
else begin {radius will be entered}
  write(ch);
  readln(input);
  input:=concat(ch, input);
  convertreal(input, radius, error);
  if radius=0 then error:=true;
  if radius<0 then ccw:=true else ccw:=false;
  radius:=abs(radius);
  if not error then begin
    radius:=convertscale(radius);
    chordist:=invdist(xstart, ystart, xend, yend);
  end {radius > 0}
end {addarc}
if chordist>2*radius then begin
  beep;
  writeprompt('ARC CANNOT BE FITTED BETWEEN GIVEN POINTS');
  delay(1500);
  end
else begin
  angle:=degree(arcsin(chordist/(2*radius))*2);
  offset:=radius*cos(rad(angle/2));
  midx:=(xstart+xend)/2;
  midy:=(ystart+yend)/2;
  if not ccw then begin
    calcnextpt(xstart,ystart,xend,yend,midx,midy,5,90,offset,xcent,ycent);
    addtobuffer(object,onscreen,4,0,xstart,ystart,xcent,ycent,angle,0);
    displayarc(xstart,ystart,xcent,ycent,angle);
    end
  end
else begin
    calcnextpt(xstart,ystart,xend,yend,midx,midy,6,90,offset,xcent,ycent);
    addtobuffer(object,onscreen,4,0,xend,yend,xcent,ycent,angle,0);
    displayarc(xend,yend,xcent,ycent,angle);
    end
end;{else}
end;{not error}
end;{addarc}
(c ******************************************)
overlay procedure addtext(var minx,maxx,miny,maxy,crossx,crossy:real;
var object:objectarray; var onscreen:onarray);
var
  textptx,textpty,topx,topy,angle,height:real;
  count,bold:integer;
  ch:char;
  input: line;
  boldflag:boolean;

  ********************** procedure gettextinfo(var minx,maxx,miny,maxy,crossx,crossy,
  textptx,textpty,angle,height:real; var bold:boolean);
  var
    error,move,done:boolean;
    anglex,angley:real;
    temp: integer;
    ch:char;

begin
  selectpoint('Select TEXT STARTING POINT: ',object,onscreen,
    minx,maxx,miny,maxy,crossx,crossy,textptx,textpty,move);
  displaymark(textptx,textpty);
  repeat
    error:=false;
    writeprompt('Enter Angle (ie A30) or Select Angle Point: ');
    read(kbd,ch);ch:=upcase(ch);
    if (ch=#27)and keypressed then begin
      selectpoint('Select Angle Point: ',object,onscreen,minx,maxx,miny,maxy,
        crossx,crossy,textptx,textpty,angle,height);{angle will be pointed to}
      read(kbd,ch);{remove second ch from input buffer}
    end
  if move then begin
    displaymark(anglex,angley);
    angle:=inverseazimuth(textptx,textpty,anglex,angley);
    end
  if selected angle
  else if ch='A' then begin
    angle:=correct360(angle);{conv from neg to pos}
    angle:=correct360(450-angle);{conv to azimuth reference}
  end
  end
end; {not error}
end {entered angle}
else angle:=90;
until not error;
bold:=ask('Do you want bold text?');
repeat
getinteger('Enter Text Size 1=11" 2=14" 3=21" 4=28" 5=42":',
temp,done);
until (temp>0) and (temp<5);
case temp of
 1:height:=0.105;
 2:height:=0.14;
 3:height:=0.21;
 4:height:=0.28;
 5:height:=0.42;
end; {case}
end; {gettextinfo}

procedure showprompt(var textptx,textpty,angle,height,top:{,topy:real); begin
calcnextpt(0,0,0,textptx,textpty,0,correct360(angle-90),height,topx,topy);
drawline(textptx,textpty,topx,topy);
end; {showprompt}

procedure addtotext(input:txtln;var count:integer); var stored:boolean;
begin
stored:=false;
count:=1;
repeat
  count:=succ<count>
  if textline[count]='' then begin
    textline[count]:=input;
    stored:=true;
  end;
  until </textline[count]=false or (count=maxtext);
if stored=false then begin
  writeprompt('TOO MUCH TEXT IN MEMORY'); beep;
delay(1500);
end;
end; {addtotext}

procedure addtobuffer(objection,onscreen,2,bold,count,textptx,textpty);

begin
erasemenu;gotoXY(74,6);write('Text');
gettextinfo(minx,maxx,miny,maxy,crossx,crossy,textptx,textpty,angle,height,
  boldflag);
count:=1;
if boldflag then bold:=1 else bold:=0;
repeat
  input:='';
  writeprompt('Enter Text:'); clear;
  showprompt(textptx,textpty,angle,height,topx,topy);
  readln(input);
  if input<>'' then begin
    displaytext(0,textptx,textpty,height,angle,input);
    if ask('Do you wish to keep text') then begin
      addtotext(input,count); (add input to text array)
      addtobuffer(object,onscreen,2,bold,count,textptx,textpty,height,angle,0); displaytext(count,textptx,textpty,height,angle,"'";
      calcnextpt(0,0,0,textptx,textpty,0,correct360(angle+90),1.5*height,
        textptx,textpty);
    end {keerptext}
  end else begin
    setcolorblack;
    showprompt(textptx,textpty,angle,height,topx,topy);
    displaytext(0,textptx,textpty,height,angle,input);
  end; {else begin}
end;
overlay procedure getdwg(var dwgname:filename; var object:objectarray; var onscreen:onarray; var textline:textarray; var minx,maxx, miny, maxy, crossx, crossy:real; var scale:integer);

overlay procedure rename(var dwgname:filename);

overlay procedure loadwg(var dwgname:filename; var object:objectarray; var onscreen:onarray; var textline:textarray; var minx,maxx, miny, maxy, crossx, crossy:real; var scale:integer);

function goodfile(checkfile:dfilename):boolean;

begin
  writeprompt('Enter name of file to be loaded : ');
  readln(firstname);
  if firstname <> '' then begin
    clearmemory(object,onscreen,textline);
    dwgfilenm:=firstname + '.dwg';
    txtfilenm:=firstname + '.dtx';
    if goodfile(dwgfilenm) then begin
      assign(dwgfile,dwgfilenm);
      reset(dwgfile);
      count:=1;
      read(dwgfile,newobj);
      scale:=newobj.code;
      object[0].code:=newobj.code;
      object[0].desc:=newobj.desc;
      case object[0].desc of
        begin
          maxpgx:=8.5; maxpgy:=14;
        end;
    end;
end;
if objectCOJ.desc=1 then begin
  maxworldy:=maxpgy; maxworldx:=maxworldy*4/3;
end
else begin
  maxworldx:=17; maxworldy:=maxworldx*3/4;
end;

minworldx:=0; minworldy:=0;
minx:=minworldx; maxx:=maxworldx; miny:=minworldy; maxy:=maxworldy;
while not eof(dwgfile) do begin
  read(dwgfile,newobj);
  object[count] := newobj;
  count := succ(count);
end;

minworldx:=0; minworldy:=0;
minx:=minworldx; maxx:=maxworldx; miny:=minworldy; maxy:=maxworldy;
while not eof(txtfile) do begin
  read(txtfile,newtxt);
  textline[count] := newtxt;
  count := succ(count);
end;

overlay procedure changescale(var scale:integer; var object:objectarray;
  var onscreen:onarray; var minx,maxx,miny,maxy,crossx,crossy:real);
var done:boolean;
oldscale:integer;

procedure rescale(var scale,oldscale:integer; var object:objectarray;
  var onscreen:onarray; var minx,maxx,miny,maxy,crossx,crossy:real);
var scaleFactor: real;
  count: integer;
begin
  scaleFactor := oldscale / scale;
  count := 0;
repeat
  count := succ(count);
  if object[count].desc <> 0 then begin
    with object[count] do begin
      case desc of
        1, 4, 5: begin
          dataC1J := dataC1J * scaleFactor;
          dataC2J := dataC2J * scaleFactor;
          dataC3J := dataC3J * scaleFactor;
          dataC4J := dataC4J * scaleFactor;
        end;
        {line, arc, rect rescale}
        2: begin
          dataC2J := dataC2J * scaleFactor;
          dataC3J := dataC3J * scaleFactor;
        end;
        {text rescale}
        3: begin
          dataC1J := dataC1J * scaleFactor;
          dataC2J := dataC2J * scaleFactor;
        end; {circle rescale}
        6: begin
          if code = 4 then begin
            dataC1J := dataC1J * scaleFactor;
            dataC3J := dataC3J * scaleFactor;
            dataC5J := dataC5J * scaleFactor;
          end; {arrow}
          else begin
            dataC1J := dataC1J * scaleFactor;
          end;
        end; {symbol rescale}
    end; {case}
  end; {with}
until count = maxobj;
end; {rescale}
(**) begin
  eraseMenu;
gotoXY(74, 4); write('SCale');
oldscale := scale;
repeat
  getInteger('Enter NEW SCALE : ', scale, done);
  until (scale > 0) or done;
if not done then begin
  title(dwgfile, windowsz, scale);
  object[0].code := scale;
  if ask('Do you wish to change existing drawing?') then
    rescale(scale, oldscale, object, onscreen, minx, maxx, miny, maxy, crossx, crossy);
  end; {not done}
end; {changeScale}
(**) overlay procedure addrect(var minx, maxx, miny, maxy, crossx, crossy: real;
  var object: objectarray; var onscreen: onarray);
var x1, y1, x2, y2: real;
var move: boolean;
begin
  eraseMenu;
gotoXY(74, 11); write('REct');
selectPoint('Select Upper Left CORNER : ', object, onscreen,
  minx, maxx, miny, maxy, crossx, crossy, x1, y1, move);
displayMark(x1, y1);
selectPoint('Select Lower Right CORNER : ', object, onscreen,
  minx, maxx, miny, maxy, crossx, crossy, x2, y2, move);
if move then begin
displayRect(x1, y1, x2, y2);
end;
**Overlay Procedure `getdist`**

```pascal
overlay procedure getdist(var minx, maxx, miny, maxy, crossx, crossy: real);
var x1, y1, x2, y2: real;
  done: boolean;
  ch: char;
begin
  erase;
  gotoxy(74, 22); write('Dist');
  selectpoint('Select first Point :', object, onscreen, minx, maxx, miny, maxy, crossx, crossy, x1, y1, done);
  displaymark(x1, y1);
  selectpoint('Select second Point :', object, onscreen, minx, maxx, miny, maxy, crossx, crossy, x2, y2, done);
  if done then begin
    displaymark(x2, y2);
    x1 := invdist(x1, y1, x2, y2) * scale;
    gotoxy(9, 25); write('DIST = ', x1:10:3,' Press any key to cont');
    read(kbd, ch);
  end;
end;{if done}
end;{getdist}
```
overlay procedure modifyline(var minx, maxx, miny, maxy, crossx, crossy: real;
var object: objectarray; var onscreen: onarray);
var xcheck, ycheck, newx, newy, x1, y1, x2, y2: real;
move: boolean;
objno: integer;
begin
erase menu;
goto XY(74, 16); write('Modify');
selectpoint('Select LINE to be modified : ', object, onscreen,
    minx, maxx, miny, maxy, crossx, crossy, xcheck, ycheck, move);
if (objno <> 0) and (object[objno].desc = 1) then begin
    x1 := object[objno].data[1]; y1 := object[objno].data[2];
    x2 := object[objno].data[3]; y2 := object[objno].data[4];
    selectpoint('Select new line endpoint location : ', object, onscreen,
        minx, maxx, miny, maxy, crossx, crossy, newx, newy, move);
    if invdist(x1, y1, newx, newy) < invdist(x2, y2, newx, newy) then begin
        x1 := newx; y1 := newy;
    end {replace pt1}
else begin
    x2 := newx; y2 := newy;
end {replace pt2}
    with object[objno] do begin
    end {with}
    redrawobject(object, objno); {redraw new line}
end {line found}
else begin
    write(prompt('No LINE Selected'));
    beep;
delay(1500);
end {else}
end {modifyline}

overlay procedure extendline(var minx, maxx, miny, maxy, crossx, crossy: real;
var object: objectarray; var onscreen: onarray);
var xcheck, ycheck, newx, newy, x1, y1, x2, y2, angle, dist: real;
move: boolean;
objno: integer;
begin
erase menu;
goto XY(74, 15); write('Extend');
selectpoint('Select LINE to be extended : ', object, onscreen,
    minx, maxx, miny, maxy, crossx, crossy, xcheck, ycheck, move);
if (objno <> 0) and (object[objno].desc = 1) then begin
    x1 := object[objno].data[1]; y1 := object[objno].data[2];
    x2 := object[objno].data[3]; y2 := object[objno].data[4];
    selectpoint('Select extended endpoint location : ', object, onscreen,
        minx, maxx, miny, maxy, crossx, crossy, newx, newy, move);
    setcolor black;
    redrawobject(object, objno); {erase old line}
    setcolor white;
    if invdist(x1, y1, newx, newy) < invdist(x2, y2, newx, newy) then begin
        x1 := newx; y1 := newy;
    end {replace pt1}
else begin
    x2 := newx; y2 := newy;
end {replace pt2}
    with object[objno] do begin
    end {with}
    redrawobject(object, objno); {redraw new line}
end {line found}
else begin
    write(prompt('No LINE Selected'));
    beep;
delay(1500);
end {else}
end {extendline}
with object[objno] do begin
end; (with)
redrawobject(object, objno); (redraw new line)
end (line found)
else begin
writeprompt('No LINE Selected');
beep;
delay(1500);
end; (else)
end; (extendline)

overlay procedure moveobject(var minx, maxx, miny, maxy, crossx, crossy: real;
  var object: objectarray; var onscreen: onarray);

var
  xcheck, ycheck, deltax, deltay: real;
  move: boolean;
  ch: char;
  objno, count: integer;
  movebuffer: movearray;

begin
  getdelta(var minx, maxx, miny, maxy, crossx, crossy, x1, y1, move);
  if move then begin
    deltax := x2 - x1;
    deltay := y2 - y1;
  end;
end; (getdelta)

procedure translateobject(var object: objectarray;
  objno: integer; deltax, deltay: real);
begin
  with object[objno] do begin
    case desc of
      1, 4, 5: begin
      end; (line, arc, rect transl)
      2: begin
      end; (text transl)
      3: begin
      end; (circle transl)
      6: begin
        if code = 4 then begin
        end (arrow)
        else begin
          end; (all other symbols)
      end; (symbol transl)
  end; (case)
end; (with)
end; (transobj)

procedure addtomovebuffer(var movebuffer: movearray; objno: integer);
begin
movebuffer[0]:=succ(movebuffer[0]);
movebuffer[movebuffer[0]]:=objno;
end;
(******************************)
begin
eraseMenu;
gotoXY(74,20);write('Move');
writePrompt('Select object to be moved (or All):');
read(kbd,ch):ch:=upcase(ch);
if (ch=#27) and keypressed then begin {objects will be selected}
movebuffer[0]:=0;
read(kbd,ch): {remove second ch from input buffer}
repeat
  selectpoint('Select object to be moved: ',object,onscreen,
    minx,maxx,miny,maxy,crossx,crossy,xcheck,ycheck,movex);
  if move then begin
    selectobject(xcheck,ycheck,objno,object,onscreen);
    if objno<>0 then addtomovebuffer(movebuffer,objno)
  end; {else}
end; {if move}
until not move;
if movebuffer[0]>0 then begin
  repeat
    getdelta(minx,maxx,miny,maxy,crossx,crossy,deltax,deltay,movex);
    count:=0;
    repeat
      count:=succ(count);
      setColorBlack;
      redrawobject(object,movebuffer[count]);
      setColorWhite;
      translateobject(object,movebuffer[count],deltax,deltay);
    until count=movebuffer[0];
    count:=0;
    repeat
      count:=succ(count);
      redrawobject(object,movebuffer[count]);
    until count=movebuffer[0];
  until not ask('Move object(s) again?');
end; {moveobjs}
end; {selected objects}
else if ch='A' then begin {move all}
  getdelta(minx,maxx,miny,maxy,crossx,crossy,deltax,deltay,movex);
  if move then begin
    count:=0;
    repeat
      count:=succ(count);
      if object[count].desc<>0 then
        translateobject(object,count,deltax,deltay);
    until count=maxobj;
    zoomall(minx,maxx,miny,maxy,crossx,crossy,object,onscreen);
  end; {if move}
end; {move all}
end; {moveobject}
(**************************************************************************)
overlay procedure getarea(var minx,maxx,miny,maxy,crossx,crossy:real);
type arearray=array[1..50] of array[1..2] of real;
var areapt:arearray;
done:boolean;
(**************************************************************************)
procedure getareapolygon(var minx,maxx,miny,maxy,crossx,crossy:real;
  var areapt:arearray; var done:boolean);
var count:integer;
move: boolean;
begin
  done:=false;
  selectpoint('Select 1st Area Point:', object, onscreen,
    minx, maxx, miny, maxy, crossx, crossy, areapt[1,1], areapt[1,2], move);
  displaymark(areapt[1,1], areapt[1,2]);
  count:=2;
  repeat
    selectpoint('Select Next Area Point:', object, onscreen,
      minx, maxx, miny, maxy, crossx, crossy, areapt[count,1], areapt[count,2], move);
    displaymark(areapt[count,1], areapt[count,2]);
    count:=succ(count);
  until (count=50) or (not move);
  areapt[count,1]:=areapt[1,1]; areapt[count,2]:=areapt[1,2];
  if count<=3 then done:=true;
end;

procedure calcarea(var areapt: arearray);
var
  area: real;
  count: integer;
  ch: char;
begin
  count:=1;
  area:=areapt[count,1]* (areapt[succ(count),2]-areapt[count,2]);
  repeat
    count:=succ(count);
    area:=area+(areapt[count,1]* (areapt[succ(count),2]-areapt[pred(count),2]));
  until (areapt[succ(count),1]=areapt[1,1])
    and (areapt[succ(count),2]=areapt[1,2]);
  count:=succ(count);
  area:=area+ (areapt[count,1]*(areapt[count,2]-areapt[pred(count),2]));
  area:=abs(area/2);
  area:=area*scale*scale; {convert to scaled units}
  writeprompt('Area = '); write(area:10:3, ' sq units');
  if scale <> 1 then write(' ', area/4.356E04:10:3, ' Ac');
end;

begin{getarea}
  erasemenu;
  gotoxy(74,23); write('Area >');
  getareapolygon(minx, maxx, miny, maxy, crossx, crossy, areapt, done);
  if not done then calcarea(areapt);
end;

overlay procedure offset(var minx, maxx, miny, maxy, crossx, crossy: real;
  var object: objectarray; var onscreen: onarray);

procedure calcnextpt(var x1, y1, x2, y2, x3, y3: real;
  var lntype: integer; var x3, y3: real; var dircode: integer;
  var again: boolean);
begin
  if angleright(x2,y2,x1,y1,dirx,diry)<=anglemleft(x2,y2,x1,y1,dirx,diry)
    then dircode:=5
  else dircode:=6;
  repeat
calcnexpt(x1, y1, x2, y2, x3, y3, dircode, dist, x3, y3);
calcnexpt(x1, y1, x2, y2, x2, y2, dircode, dist, x4, y4);
displayline(lntype, x3, y3, x4, y4);
  addtobuffer(object, onscreen, 1, lntype, x3, y3, x4, y4, 0, 0);
  again:=ask('Offset again ');
end;

******
if again then begin
  getreal('Enter offset distance :',dist,done);
  if not done then begin
    dist:=convertscale(dist);
    x1:=x3; y1:=y3; x2:=x4; y2:=y4;
    end
  else again:=false;
  end;
  until not again;
end;(offsetline)

procedure offsetcircle(var object:objectarray; var onscreen:onarray;
                        x1,y1,rad1,dirx,diry,dist:real);
  var rad2:real;
  again:boolean;
  begin
    if invdist(x1,y1,dirx,diry)<=rad1 then dist:=0-dist; (offset inside)
    rad2:=rad1+dist;
    repeat
      if rad2<=0 then begin
        writeprompt('New Radius is less than 0 ');
        beep;delay(1500);
        end
      else begin
        displaycircle(x1,y1,rad2);
        addtobuffer(object,onscreen,3,0,x1,y1,rad2,0,0,0);
        end;
      again:=ask('Offset again ');
      if again then rad2:=rad2+dist;
    until not again;
  end;{offsetcircle}

procedure offsetarc(var object:objectarray; var onscreen:onarray;
                   s1x,s1y,rx,ry,angle,dirx,diry,dist:real);
  var rad1,rad2,s2x,s2y:real;
  again:boolean;
  begin
    rad1:=invdist(rx,ry,s1x,s1y);
    if invdist(rx,ry,dirx,diry)<=rad1 then dist:=0-dist;
    rad2:=rad1+dist;
    repeat
      if rad2<=0 then begin
        writeprompt('New Radius is less than 0 ');
        beep;delay(1500);
        end
      else begin
        calcnextpt(rx,ry,s1x,s1y,rx,ry,angle,1,0,rad2,s2x,s2y);
        displayarc(s2x,s2y,rx,ry,angle);
        addtobuffer(object,onscreen,4,0,s2x,s2y,rx,ry,angle,0);
        end;
      again:=ask('Offset again ');
      if again then begin
        rad2:=rad2+dist;
        s1x:=s2x; s1y:=s2y;
      end;
    until not again;
  end;{offsetarc}

procedure offsetrect(var object:objectarray; var onscreen:onarray;
                     x1,y1,x2,y2,dirx,diry,dist:real);
  var x3,y3,x4,y4:real;
  exist:boolean;
  begin
    exist:=true;
    if (dirx=x1) and (dirx=x2) and (diry=y1) and (diry=y2) then dist:=0-dist; (change sign for rect inside)
repeat
  \( x_3 = x_1 \) \( - \text{dist} \); \( y_3 = y_1 \) \( - \text{dist} \);
  \( x_4 = x_2 \) \( + \text{dist} \); \( y_4 = y_2 \) \( + \text{dist} \);
  if \( (x_3 < x_4) \) and \( (y_3 < y_4) \) then begin
    displayrect\( (x_3, y_3, x_4, y_4) \);
    addtobuffer\( \text{object, onscreen, 5, 0, x_3, y_3, x_4, y_4, 0, 0} \);
    end \{ \text{rect exists} \}
  else begin
    \( \text{exist} = \text{false} \);
    writepromot\( \text{('OFFSET RECTANGLE DOES NOT EXIST')} \);
    beep; delay\( (1500) \);
    end \{ \text{else} \}
  again := \text{ask ('Offset again ')};
  if again then begin
    \( x_1 = x_3 \); \( y_1 = y_3 \); \( x_2 = x_4 \); \( y_2 = y_4 \);
    end;
  until not again;
end \{ \text{offsetrect} \}

\( \text{********************} \) begin
  \text{erasemenu}; gotoXY\( (74, 14) \); write\( ('OffSet') \);
  \text{selectpoint ('Select object to be offset : ')}, \text{object, onscreen, minx, maxx, miny, maxy, crossx, crossy, objx, objy, move} ;
  \text{selectobject (objx, objy, objno, object, onscreen)} ;
  if \( \text{objno <> 0} \) and \( \text{object(objno).desc in [1, 3, 4, 5]} \) then begin
    \text{selectpoint ('Select offset direction : ')}, \text{object, onscreen, minx, maxx, miny, maxy, crossx, crossy, dirx, diry, move} ;
  if move then begin
    getreal\( ('Enter offset distance : ') \), \text{dist, done} ;
    if not done then begin
      \text{dist = convertscale(dist)} ;
      with \text{object[objno]} do begin
        case desc of
          1: offsetline\( \text{object, onscreen, code, data[1], data[2], data[3], data[4], dirx, diry, dist} \);
          3: offsetcircle\( \text{object, onscreen, data[1], data[2], data[3], dirx, diry, dist} \);
          4: offsetarc\( \text{object, onscreen, data[1], data[2], data[3], data[4], data[5], dirx, diry, dist} \);
          5: offsetrect\( \text{object, onscreen, data[1], data[2], data[3], data[4], dirx, diry, dist} \);
        end ; \{ \text{case} \}
      end ; \{ \text{with} \}
      end ; \{ \text{if done} \}
    end ; \{ \text{objno offset} \}
  else begin
    writepromot\( \text{('OBJECT CANNOT BE OFFSET ')} \);
    beep; delay\( (1500) \);
    end ; \{ \text{else} \}
end \{ \text{offset} \}

\( \text{****************************} \)
overlay procedure fillet\( \text{(var minx, maxx, miny, maxy, crossx, crossy: real; var object: objectarray; var onscreen: onarray)} \);
var 11x, 11y, 12x, 12y, x1, y1, x2, y2, x3, y3, x4, y4, xi, yi, radius: real;
objno1, objno2: integer;
move, exist: boolean;
\( \text{****************************} \)
procedure calcintersect\( \text{(var x1, y1, x2, y2, x3, y3, x4, y4, xi, yi: real; var exist: boolean)} \);
var a1, a2, b1, b2, c1, c2: real; ch: char;
begin
  \text{exist = true} ;
  a1 := y1 \( - y_2 \); \( b1 := x_2 - x_1 \); \( c1 := x_2 \times y_1 - x_1 \times y_2 \);
  a2 := y3 \( - y_4 \); \( b2 := x_4 \times x_3 \); \( c2 := x_4 \times y_3 - x_3 \times y_4 \);
  if \( (a2 \times b1) = (a1 \times b2) \) then \text{exist = false} \quad \text{(lines are parallel)}
else begin
  \[ x_i = \frac{c_1b_2 - c_2b_1}{a_1b_2 - a_2b_1}; \]
  \[ y_i = \frac{a_1c_2 - a_2c_1}{a_1b_2 - a_2b_1}; \]
end;
end;\{calcinter\}

function ptinsegment(newx1,newy1,newx2,newy2,newxck,newyck:real):boolean;
var ch:char;
begin
  if newx1 > newx2 then switch(newx1,newx2);
  if newy1 > newy2 then switch(newy1,newy2);
  if ((newxck>=newx1) and (newxck<=newx2)) or ((newyck>=newy1) and (newyck<=newy2))
  then ptinsegment:=true
  else ptinsegment:=false;
end;\{ptinseg\}

procedure calc:tangentpts(x1,y1,xi,yi,x3,y3,radius:real;objno1,objno2:integer;
var object:objectarray; var onscreen:onarray);
var xt1,yt1,xt2,yt2,xr,yr,delta,tangent:real;
begin
  \[ \text{delta} = 180 - \text{vertex}(x_1,y_1,x_i,y_i,x_3,y_3); \]
  \[ \text{tangent} = \text{radius} \times \tan(\text{rad}(\text{delta}/2)); \]
  if \text{angleright}(x_1,y_1,x_i,y_i,x_3,y_3) < \text{angleleft}(x_1,y_1,x_i,y_i,x_3,y_3)
  then begin
    switch(x_1,x_3);
    switch(y_1,y_3);
  end;
  if \text{radius}>0 then begin
    calcnextpt(x_1,y_1,x_i,y_i,x_1,0,tangent,xt1,yt1);
    calcnextpt(x_i,y_i,x_3,x_3,y_1,1,0,tangent,xt2,yt2);
    calcnextpt(x_1,y_1,x_t1,y_t1,1,5,90,radius,xr,yr);
    \text{displayarc}(xt1,yt1,xr,yr,delta,0);
  end (\text{rad}=0)
  else begin
    xt1:=xi; yt1:=yi; xt2:=xi; yt2:=yi;
  end;
  with object[objno1] do begin
    data[1]:=x_1; data[2]:=y_1; data[3]:=xt1; data[4]:=yt1;
  end;\{with\}
  with object[objno2] do begin
    data[1]:=x_3; data[2]:=y_3; data[3]:=xt2; data[4]:=yt2;
  end;\{with\}
  redrawobject(object,objno1);
  redrawobject(object,objno2);
end;\{calc:tanpts\}

begin
  erasemenu;
gotoxy(74,13);write('Fillet');
repeat
  \text{objno1}:=0;
  selectpoint('Select 1st LINE : ',object,onscreen,
    minx,maxx,miny,maxy,crossx,crossy,lx,ly,move);
  if move then selectobject(lx,ly,objno1,object,onscreen);
  until (objno1<>0) or (not move);
  if ((objno1<>0) and (object[objno1].desc=1)) then begin
    repeat
      \text{objno2}:=0;
      \text{objno1}:=0;
  end else begin
  \text{objno2}:=0;
  \text{objno1}:=0;
end;\{calc:tanpts\}
selectpoint('Select 2nd LINE : ',object,onscreen,
   minx,maxx,miny,maxy,crossx,crossy,12x,12y,move);
if move then selectobject(12x,12y,objno2,object,onscreen);
until (objno2<>0) or (not move);
if (objno2<>0) and (object(objno2).desc=1) then begin
   with object(objno1) do begin
      x1:=data[1]; y1:=data[2]; x2:=data[3]; y2:=data[4];
   end;
   with object(objno2) do begin
      x3:=data[1]; x4:=data[2]; x4:=data[3]; y4:=data[4];
   end;
end;
if ((objno2<>0) and (object(objno2).desc=1)) then begin
   with object(objno1) do begin
      x1:=data[1]; y1:=data[2]; x2:=data[3]; y2:=data[4];
   end;
   with object(objno2) do begin
      x3:=data[1]; x4:=data[2]; x4:=data[3]; y4:=data[4];
   end;
end;

calcintersect(x1,y1,x2,y2,x3,y3,x4,y4,xi,yi,exist);
if exist then begin
   if ptinsegment(x2,y2,xi,yi,11x,11y) then begin
      xl:=x2; y1:=y2;
   end; {switch points}
   if ptinsegment(x4,y4,xi,yi,12x,12y) then begin
      x3:=x4; y3:=y4;
   end; {switch points}
end;
getreal('Enter fillet RADIUS : ',radius,exist);
setcolorblack;
redrawobject(object,objno1); {erase existing lines on screen}
redrawobject(object,objno2); {erase existing lines on screen}
setcolorwhite;
if exist then radius:=0; {no number was entered}
radius:=convertscale(radius);
calctangentpts(x1,y1,xi,yi,x3,y3,radius,objno1,objno2,object,onscreen);
end; {second line selected}
end; {first line selected}
end; {fillet}

(******************************************************************************)
overlay procedure break(var minx,maxx,miny,maxy,crossx,crossy:real;
   var object:objectarray; var onscreen:onarray);
var xbk,ybk:real;
objno:integer;
move,done:boolean;
(******************************************************************************)
procedure breakline(var object:objectarray; var onscreen:onarray;
   x1,y1,x2,y2,xbk,ybk:real; objno:integer);
var xtemp1,ytemp1,xtemp2,ytemp2,distireal;
begin
   dist:=invdist(x1,y1,xbk,ybk)-(1/32);
calcnxtpt(x1,y1,x2,y2,x1,y1,1,0,dist,xtemp1,ytemp1);
calcnxtpt(x2,y2,x1,y1,1,0,dist,xtemp2,ytemp2);
setcolorblack;
displayline(object(objno).code,x1,y1,x2,y2);
setcolorwhite;
displayline(object(objno).code,xtemp1,ytemp1,xtemp2,ytemp2);
with object(objno) do begin
   data[3]:=ytemp1; data[4]:=ytemp1;
end;
addtobuffer(object,onscreen,1,object(objno).code,xtemp2,ytemp2,x2,y2,0,0);
end; {breakline}
(******************************************************************************)
procedure breakcircle(var object:objectarray; var onscreen:onarray;
   rx,ry, radius,xbk,ybk:real; objno:integer);
var xtemp1,ytemp1,xtemp2,ytemp2,distireal;
begin
   calcnxtpt(rx,ry,x2,y2,rx,ry,0.5,dist,xtemp1,ytemp1);
calcnxtpt(rx,ry,x2,y2,rx,ry,0.5,dist,xtemp2,ytemp2);
setcolorblack;
displaycircle(rx,ry,radius);
setcolorwhite;
displayarc(xtempl,ytempl,rx,ry,359);
with object[objno] do begin
  desc:=4;
data[1]:=xtempl; data[2]:=ytempl; data[3]:=rx; data[4]:=ry; data[5]:=359;
end;{with}
end;{breakcircle}

procedure breakarc(var object:objectarray; var onscreen:onarray;
  sx,sy,rx,ry,angle,xbk,ybk:real; objno:integer);
var xtemp1,ytemp1,xtemp2,ytemp2,radius,newangle:real;
beg
  radius:=invidist(sx,sy,rx,ry);
calcnextpt(rx,ry,xbk,ybk,rx,ry,5,0.5,radius,xtempl,ytempl);
calcnextpt(rx,ry,xbk,ybk,rx,ry,6,0.5,radius,xtemp1,ytemp1);
  setcolorblack;
displayarc(sx,sy,rx,ry,angle);
  setcolorwhite;
  newangle:=angleright(sx,sy,rx,ry,xtemp2,ytemp2);
displayarc(sx,sy,rx,ry,newangle);
  object[objno].data[5]:=newangle;
displayarc(xtemp1,ytemp1,rx,ry,(angle-newangle-1));
  addtobuffer(object,onscreen,4,0,xtemp1,ytemp1,rx,ry,(angle-newangle-1),0);
end;{breakarc}

procedure breakrect(var object:objectarray; var onscreen:onarray;
  ulx,uly,lrx,lry:real; objno:integer);
beg
  with object[objno] do begin
    desc:=1; code:=1;
data[1]:=ulx; data[2]:=uly; data[3]:=lrx; data[4]:=uly;
end;{with}
  addtobuffer(object,onscreen,1,1,lrx,uly,lrx,lry,0,0);
  addtobuffer(object,onscreen,1,1,lrx,lry,ulx,lry,0,0);
  addtobuffer(object,onscreen,1,1,ulx,lry,ulx,uly,0,0);
  writelnprompt('Rectangle has been broken into 4 LINES'); delay(500);
end;{breakrect}

begin
  {break}
erasemenu;gotoXY(74,17);write('BReak');
selectpoint('Select object to be broken : ',object,onscreen,
  minx,maxx,miny,maxy,crossx,crossy,xbk,ybk,move);
selectobject(xbk,ybk,objno,object,onscreen);
if ((objno<>0) and (object[objno].desc in [1,3,4,5])) then begin
  with object[objno] do begin
    case desc of
      1:breakline(object,onscreen,data[1],data[2],data[3],data[4],xbk,ybk,objno);
      3:breakcircle(object,onscreen,data[1],data[2],data[3],data[4],xbk,ybk,objno);
      4:breakarc(object,onscreen,
        data[1],data[2],data[3],data[4],data[5],xbk,ybk,objno);
      5:breakrect(object,onscreen,data[1],data[2],data[3],data[4],objno);
    end;{case}
  end;{with}
else begin
  writelnprompt('OBJECT CANNOT BE BROKEN'); beep; delay(500);
end;{else}
end;{break}

overlay procedure chline(var minx,maxx,miny,maxy,crossx,crossy:real;
  var object:objectarray; var onscreen:onarray)
var xcheck,ycheck:real;
error,mover:boolean;
objno,intype:integer;
ch:char;
beg
erasemenu;gotoXY(74,2);write('ChLine');
selectpoint ("Select LINE to be changed : ", object, onscreen,
  minx, maxx, miny, maxy, crossx, crossy, xcheck, ycheck, move);
selectobject (xcheck, ycheck, objno, object, onscreen);
if (objno <> 0) and (object[objno].desc=1) then begin
  repeat
    error:=false;
    writeprompt ("Select New LINE TYPE : N B D R C K X O W \rightarrow ");
    read(kbd, ch); write(ch);
    ch:=upcase(ch);
    case ch of
      'N': lntype:=1;
      'B': lntype:=2;
      'D': lntype:=3;
      'R': lntype:=4;
      'C': lntype:=5;
      'K': lntype:=6;
      'X': lntype:=7;
      'O': lntype:=8;
      'W': lntype:=9;
    else begin
      error:=true; beep;
    end;
  end;
  until not error;
  setcolorblack;
  redrawobject (object, objno); { erase old line }
  setcolorwhite;
  object[objno].code:=lntype;
  redrawobject (object, objno); { redraw new line }
  end (line found)
else begin
  writeprompt ("No LINE Selected");
  beep;
  delay (1500);
end;
end;{chline}

overlay procedure chtext (var minx, maxx, miny, maxy, crossx, crossy: real;
  var objarray: objarray; var onscreen: onarray);
var
  xc, yc, height: real;
  move, bold: boolean;
  objno, temp: integer;
  ch: char;
begin
  erasmenu; gotoXY(74, 3); write('ChText');
  selectpoint ("Select TEXT to be changed : ", object, onscreen,
    minx, maxx, miny, maxy, crossx, crossy, xcheck, ycheck, move);
  selectobject (xcheck, ycheck, objno, object, onscreen);
  if (objno <> 0) and (object[objno].desc=2) then begin
    writeprompt ('');
    write ('[', object[objno].data[4]:4:2);
    if object[objno].code=1 then write('B ') else write('B ');
    write(textline[trunc(object[objno].data[1])]);
    repeat until keypressed;
    read(kbd, ch);
    repeat
      getinteger ('Enter NEW Text Size 1=.11"  2=.14"  3=.21"  4=.28"  5=.42": ',
        temp, move);
      until (temp>0) and (temp<=5);
    case temp of
      1: height:=0.105;
      2: height:=0.14;
      3: height:=0.21;
      4: height:=0.28;
      5: height:=0.42;
    end;{case}
    if ask ('Do you want bold text ?') then object[objno].code:=1
      else object[objno].code:=0;
    setcolorblack;
    redrawobject (object, objno);
setcolorwhite;
redrawobject(object,objectno);
end {text found}
else begin
writeprompt('No Text Selected');
beep;
delay(1500);
end {else}
end {chtext}

<********************************>
procedure menu(var minx,maxx,miny,maxy,crossx,crossy:real;
var object:objectarray;var onscreen:onarray;var finished:boolean;
var dwgfile:filename; var scale:integer>;

var input:string;
ch1,ch2:char;
X1,Y1,X2,Y2:real;
mov,done:boolean;
objectno:integer;

begin
finished:=false;
repeat
displaymenu;
writeprompt('SELECT COMMAND : ');
read(kbd,chl);
chl:=upcase(chl);
write(chl);
read(kbd,ch2>);
ch2:=upcase(ch2);
write(ch2);
input:=concat(ch1,ch2);
if input='DW' then
  getdwg(dwgfile,object,onscreen,iextline,minx,maxx,miny,maxy,crossx,crossy,scale)
else if input = 'BO' then
  getborder(object,onscreen,minx,maxx,miny,maxy,crossx,crossy)
else if input = 'SC' then
  changescale(scale,object,onscreen,minx,maxx,miny,maxy,crossx,crossy)
else if input = 'CO' then
  getcogo(minx,maxx,miny,maxy,crossx,crossy,object,onscreen)
else if input = 'TX' then
  addtext(minx,maxx,miny,maxy,crossx,crossy,object,onscreen)
else if input = 'RE' then
  addrect(minx,maxx,miny,maxy,crossx,crossy,object,onscreen)
else if input = 'SY' then
  addsymbol(minx,maxx,miny,maxy,crossx,crossy,object,onscreen)
else if input = 'FL' then
  fillet(minx,maxx,miny,maxy,crossx,crossy,object,onscreen)
else if input = 'OS' then
  ofset(minx,maxx,miny,maxy,crossx,crossy,object,onscreen)
else if input = 'EX' then
  extendline(minx,maxx,miny,maxy,crossx,crossy,object,onscreen)
else if input = 'MD' then
  modifyline(minx,maxx,miny,maxy,crossx,crossy,object,onscreen)
else if input = 'BR' then
  break(minx,maxx,miny,maxy,crossx,crossy,object,onscreen)
else if input = 'WI' then
else if input = 'ZM' then
  zoom(object, onscreen, crossx, crossy, minx, maxx, miny, maxy)
else if input = 'MV' then
  moveobject(minx, maxx, miny, maxy, crossx, crossy, object, onscreen)
else if input = 'ER' then
  erase(minx, maxx, miny, maxy, crossx, crossy, object, onscreen)
else if input = 'DI' then
  getdist(minx, maxx, miny, maxy, crossx, crossy)
else if input = 'AA' then
  getarea(minx, maxx, miny, maxy, crossx, crossy)
else if input = 'PL' then
  plot(minx, maxx, miny, maxy, crossx, crossy, object, onscreen)
else if ((input[1] = #27) and (input[2] = #59)) then begin
  menu1 := not menu1; erasemenu; displaymenu;
  end
else if input = 'CL' then
  chline(minx, maxx, miny, maxy, crossx, crossy, object, onscreen)
else if input = 'CT' then
  chtext(minx, maxx, miny, maxy, crossx, crossy, object, onscreen)
else if ((input[1] = #27) and (input[2] = #60)) then begin
  colorvar := succ(colorvar); if colorvar > MaxForeground then colorvar := 1;
  setforegroundcolor(colorvar);
  end
else if input = 'QU' then begin
  gotoxy(74, 25); write('QUIT');
  finished := true;
  if ask('Do you wish to save current drawing') then
    storedwg(dwgfile, object, textline);
  writeprompt('QUIT FUNCTION');
  end
else beep;
until finished;
end;

 ******************************************
begin (main)
clearmemory(object, onscreen, textline);
inigraphic;
maxworldx := 17; maxworldy := maxworldx * 3/4;
minworldx := 0; minworldy := 0;
currentminx := minworldx; currentmaxx := maxworldx;
currentminy := minworldy; currentmaxy := maxworldy;
changewindow(crossx, crossy, currentminx, currentmaxx, currentminy, currentmaxy);
finished := false;
dwgfile := 'NO NAME';
windowsz := 1.0;
scale := 1;
titles(dwgfile, windowsz, scale);
menu1 := true;
menu(currentminx, currentmaxx, currentminy, currentmaxy, crossx, crossy, object, onscreen, finished, dwgfile, scale);
leavegraphic;
end.
SOURCE LISTING FOR FILE
SCREEN2.PAS
procedure writeprompt(promptline: line);
var count: integer;
begi
  count := length(promptline) + 9;
gotoXY(9,25);
write('>
gotoXY(9,25);
write(promptline);
repeat
  count := succ(count);
  write(' ');
until count >= 73;
gotoXY(length(promptline) + 9, 25);
end;

procedure titles(dwgfile: filename; windowsz, scale: real);
begi
  gotoXY(1, 1);
  write(' )
  gotoXY(1, 1);
  write('DWG->', dwgfile);
gotoXY(26, 1);
write(' UCF DRAW');
gotoXY(47, 1);
write('WINDOW: ', windowsz: 7: 3);
gotoXY(64, 1);
write('SCALE: ', trunc(scale));
gotoXY(1, 25);
write('PROMPT: ');
end;

function convertscale(scaledist: real): real;
begi
  convertscale := scaledist / scale;
end;

procedure clearmemory(var object: objectarray; var onscreen: onarray;
  var textline: textarray);
var count: integer;
begi
  count := 0;
  object[0]. desc := 0; object[0]. code := 1; onscreen[0] := 0;
  repeat
    count := succ(count);
    object[count]. desc := 0;
  until count = maxobj;
  count := 0;
  repeat
    count := succ(count);
    textline[count] := '
  until count = maxtext;
end;

procedure storedwg(var dwgname: filename;
  var object: objectarray; var textline: textarray);
type dfilename = string[15];
var dwgfile: file of dwgenreality;
txtfile: file of txtfile;
dwgfilename, textfilename: dfilename;
1: integer;
done: boolean;
begin (storedwg)
if dwgname = ' NO NAME' then begin
  writeprompt('Input name of SAVE file : ');
  readln(dwgname);
  if dwgname='' then done:=true;
end;
if not done then begin
  titles(dwgname,windowsz,scale);
  dwgfilenm:= dwgname + '.dwg';
  txtfilenm:= dwgname + '.dtx';
  assign(dwgfile,dwgfilenm);
  rewrite(dwgfile);
  write(dwgfile,objectCOl>;
  I:=1;
  while I<(maxobj + 1) do begin
    if object[I].desc <> 0 then write(dwgfile,object[I]);
    I:=I + 1;
  end;
  close(dwgfile);
  assign(txtfile,txtfilenm);
  rewrite(txtfile);
  I:= 1;
  while I<<(maxtext + 1) do begin
    if length(textline[I]) <> 0 then write(txtfile,textline[I]);
    I:=I + 1;
  end;
  close(txtfile);
  writeprompt('File Store Complete');delay(1500);
end; {if then}
end; (storedwg)

procedure beep;
begin
  sound <600>
  delay(200);
  nosound;
end; (beep)

function ask(question:line;boolean;
type charset=set of char;
var -input:stringC5;
  promptset:charset;
  c:h: char;
begin (ask)
  promptset:=[Y','N'];
  repeat
    writeprompt(question);
    write(' Y/N : ');
    readln(input);
    if input <> '' then begin
      ch:= copy(input,1,1);
      ch:= upcase(ch);
      if not (ch in promptset) then begin
        writeprompt(' Invalid input -- Please reenter');
        beep;
        delay(1500);
        end;
      end;
      input<>' ';
    else ch:='N';
    until ch in promptset;
    if ch='Y' then ask:=true else ask:=false;
  end; (ask)

procedure getinteger(prompt:line; var promptvar:integer; var done:boolean);
var
  input:stringC5;
  code:integer;
begin 
done:=false; code:=0; 
repeat 
  writepromp(prompt); 
  readln(input); 
  if input=' ' then begin 
    done:=true; 
    code:=0; 
  end 
else begin 
  val(input,promptvar,code); 
  if code <> 0 then begin 
    writepromp(' Invalid Entry -- Please Re-input ');
    beep; 
    delay(1500); 
  end;
end; 
until code=0; 
end;getinteger}
<******************************************>
procedure convertreal(input:line; var convert:real; var error:boolean); 
begin 
  code:=0; error:=false; 
  val(input,convert,code); 
  if code <> 0 then begin 
    writepromp(' Invalid Entry -- Please Re-input ');
    beep; 
    delay(1500); 
    error:=true; 
  end;
end;{convertreal}
<*****************************~************'
procedure getreal(prompt:line; var promptvar:real; var input:line; 
  error:boolean; 
begin 
  repeat 
    done:=false; 
    error:=false; 
    writepromp(prompt); 
    readln(input); 
    if input=' ' then done:=true 
  else convertreal(input,promptvar,error); 
  until not error; 
end;{getreal}
<******************************************>
procedure switch(var first,second:real); 
begin 
  temp:=first; first:=second; second:=temp; 
end;{switch}
<*******************************************>'
function ponscreen(x,y:real):boolean; 
begin 
  if (x=currentmaxx) and (x>=currentminx) and (y=currentmaxy) and (y>=currentminy) 
  then ponscreen:=true
  else ponscreen:=false; 
end; 
<************************************************>
procedure checkwindow(var minimumx,maximumx,minimumy,maximumy:real); 
begin 
  if minimumx<minworldx then begin 
    maximumx:=maximumx+(minworldx-minimumx); 
    minimumx:=minworldx; 
  end; 
  (CHECKS TO SEE IF WINDOW MOVEMENT)
if minimumy < minworldy then begin
  maximumy := maximumy + (minworldy - minimumy);
  minimumy := minworldy;
end;
if maximumx > maxworldx then begin
  minimumx := minimumx - (maximumx - maxworldx);
  maximumx := maxworldx;
end;
if maximumy > maxworldy then begin
  minimumy := minimumy - (maximumy - maxworldy);
  maximumy := maxworldy;
end;
end;(*ckwndw)

procedure displaymark(x, y: real);
var
  pixx, pixy: integer;
begin
  pixx := windowx <x>;
pixy := windowy <y>;
setwindowmodeoff;
drawpoint(pixx-1, pixy);
drawpoint(pixx+1, pixy);
drawpoint(pixx, pixy-1);
drawpoint(pixx, pixy+1);
setwindowmodeon;
end;(*displaymark)

procedure displaymenu;
begin
  gotoxy(74,2); write('DWg');
gotoxy(74,4); write('Scale');
gotoxy(74,6); write('Text');
gotoxy(74,8); write('Line');
gotoxy(74,10); write('Arc');
gotoxy(74,12); write('Symbol');
gotoxy(74,14); write('Set');
gotoxy(74,16); write('Modify');
gotoxy(74,18); write('Window');
gotoxy(74,20); write('Move');
gotoxy(74,22); write('Dist');
gotoxy(74,24); write('Plot');
end;(*displaymenu1)

procedure displaymenu2;
begin
  gotoxy(74,2); write('ChLine');
gotoxy(74,25); write('Quit');
end;(*displaymenu2)

procedure erasemenu;
var
  line: integer;
begin
  line := 1;
  repeat
    line := succ(line);
    gotoxy(74, line);
    write('');
  until line = 25;
end;(*erasemenu)

********************~*********************>
procedure displaymenu;
begin
  gotoxy(74,3); write('Border');
gotoxy(74,5); write('DoGo');
gotoxy(74,7); write('View');
gotoxy(74,9); write('Circle');
gotoxy(74,11); write('Rect');
gotoxy(74,13); write('Fillet');
gotoxy(74,15); write('Extend');
gotoxy(74,17); write('Break');
gotoxy(74,19); write('Zoom');
gotoxy(74,21); write('Erase');
gotoxy(74,23); write('Area');
gotoxy(74,25); write('Quit');
end;(*displaymenu)

********************~*********************>
procedure displaymenu2;
begin
  gotoxy(74,3); write('ChText');
gotoxy(74,7); write('Quit');
end;(*displaymenu2)

procedure erasemenu;
var
  line: integer;
begin
  line := 1;
  repeat
    line := succ(line);
    gotoxy(74, line);
    write('');
  until line = 25;
end;(*erasemenu)

********************~*********************>
procedure displaymenu;
begin
  gotoxy(74,3); write('Border');
gotoxy(74,5); write('DoGo');
gotoxy(74,7); write('View');
gotoxy(74,9); write('Circle');
gotoxy(74,11); write('Rect');
gotoxy(74,13); write('Fillet');
gotoxy(74,15); write('Extend');
gotoxy(74,17); write('Break');
gotoxy(74,19); write('Zoom');
gotoxy(74,21); write('Erase');
gotoxy(74,23); write('Area');
gotoxy(74,25); write('Quit');
end;(*displaymenu)

********************~*********************>
procedure displaymenu2;
begin
  gotoxy(74,3); write('ChText');
gotoxy(74,7); write('Quit');
end;(*displaymenu2)

procedure erasemenu;
var
  line: integer;
begin
  line := 1;
  repeat
    line := succ(line);
    gotoxy(74, line);
    write('');
  until line = 25;
end;(*erasemenu)
procedure changewindow(var crossx, crossy: real; minx, maxx, miny, maxy: real);
begin
  definewindow(1, 0, 10, xmaxglb-9, ymaxglb-9);
  defineworld<1, minx, maxx, miny, maxy>;
  selectworld<1>;
  selectwindow<1>;
  setbackground<0>;
  drawborder;
  crossx:=(minx+maxx)/2; crossy:=(miny+maxy)/2;
end:(changewindow)

function invdist(fromx, fromy, tox, toy: real): real;
begin
  invdist:=sqrt(sqr(tox-fromx)+sqr(toy-fromy));
end:invdist)

function correct360(angle: real): real;
begin
  if angle<0 then angle:=angle+360;
  if angle>=360 then repeat
    angle:=angle-360;
    until angle<360;
  correct360:=angle;
end:

function degree(angle rad: real): real;
begin
  degree:=angle rad*180/3.14159;
end:

function rad(angle: real): real;
begin
  rad:= angle*3.14159/180;
end:

function arcsin(sina: real): real;
begin
  if sina=0 then arcsin:=0
  else if sina=1 then arcsin:=3.14159/2
  else if sina=-1 then arcsin:=-3.14159/2
  else arcsin:= arctan(1/(sqrt(sqr(1/sina)-1)));
end:

function inverseazimuth(fromx, fromy, tox, toy: real): real;
begin
  latitude:= fromy - toy;
  departure:= tox - fromx;
  if latitude=0 then
    if departure>0 then inverseazimuth:=90 else inverseazimuth:=270;
  if departure=0 then
    if latitude>0 then inverseazimuth:=0 else inverseazimuth:=180;
  if departure>0 then
    inverseazimuth:=90-degree(arctan(latitude/departure));
  if departure<0 then
    inverseazimuth:=270-degree(arctan(latitude/departure));
end: (function inverseazimuth)

function angleright(x1, y1, x2, y2, x3, y3: real): real;
begin
  start:=inverseazimuth(x2, y2, x1, y1);
  stop:=inverseazimuth(x2, y2, x3, y3);
  temp:=stop - start;
  angleright:=correct360(temp);
end;
function angleleft(x1, y1, x2, y2, x3, y3: real): real;
var start, stop, temp: real;
begin
  start := inverseazimuth(x2, y2, x1, y1);
  stop := inverseazimuth(x2, y2, x3, y3);
  temp := start - stop;
  angleleft := correct360(temp);
end;

function vertex(x1, y1, x2, y2, x3, y3: real): real;
var temp: real;
begin  // (returns the smaller angle between 3 pts)
  temp := angleright(x1, y1, x2, y2, x3, y3);
  if temp > 180 then
    vertex := 360 - temp
  else
    vertex := temp;
end; vertex

procedure calcnextpt(fromx, fromy, tox, toy, atx, aty: real; dircode: integer; tempangle, dist: real; var newx, newy: real);
var azimuth: real;
begin  // calcnextpoint
  case dircode of
    0: azimuth := tempangle;
    1: azimuth := inverseazimuth(fromx, fromy, tox, toy);
    2: azimuth := inverseazimuth(tox, toy, fromx, fromy);
    3: azimuth := inverseazimuth(tox, toy, fromx, fromy) + tempangle;
    4: azimuth := inverseazimuth(tox, toy, fromx, fromy) - tempangle;
    5: azimuth := inverseazimuth(fromx, fromy, tox, toy) + tempangle;
    6: azimuth := inverseazimuth(fromx, fromy, tox, toy) - tempangle;
  end; {case}
  azimuth := correct360(azimuth);
  newy := aty - (cos(rad(azimuth)) * dist);
  newx := atx + (sin(rad(azimuth)) * dist);
end; {calcnextpt}

procedure addonscreen(var onscreen: onarray; objno: integer);
begin
  onscreen[onscreen[count] := objno;
end; {addonscreen}

procedure addtobuffer(var object: objectarray; var onscreen: onarray; item, int1: integer; real1, real2, real3, real4, real5, real6: real);
var count: integer;
begin
  stored := false;
count := 0;
repeat
  count := succ(count);
  if object[count].desc = 0 then begin
    with object[count] do begin
      desc := item;
      code := int1;
      stored := true;
    end; {with}
  end; {if object unused}
until (stored = true) or (count = maxobj);
if stored then addonscreen(onscreen, count)
else begin
  writelnprompt('TOO MANY OBJECTS IN DRAWING MEMORY');
  beep;
end;
begin; \{ else \}
end; \{ add to buffer \}

\begin{verbatim}
procedure removefrombuffer(var object: objectarray; var onscreen: onarray;
    objno: integer);
var count1, count2: integer;
begin
    with object[objno] do begin
        desc := 0; code := 0;
    end; \{ with \}
    count1 := 0;
    repeat
        count1 := succ(count1);
        if onscreen[count1] = objno then begin
            count2 := count1;
            repeat
                onscreen[count2] := onscreen[succ(count2)];
                count2 := succ(count2);
            until count2 >= onscreen[0];
            onscreen[0] := pred(onscreen[0]);
        end; \{ removed obj on screen \}
    until count1 >= onscreen[0];
end; \{ removefrombuffer \}
\end{verbatim}

\begin{verbatim}
procedure displayline(lntype: integer; fromx, fromy, tox, toy: real);
var
dist, tempdist, new1x, newly, new2x, new2y: real;
count: integer;
\end{verbatim}

\begin{verbatim}
procedure drawpost(fromx, fromy, tox, toy, postx, posty: real; chainlink: boolean);
var
    new1x, new2x, new3x, new4x, new1y, new2y, new3y, new4y: real;
begin
    calcnextpt(fromx, fromy, tox, toy, postx, posty, 3, 45, 0.04, new1x, new1y);
    calcnextpt(fromx, fromy, tox, toy, postx, posty, 3, 135, 0.04, new2x, new2y);
    calcnextpt(fromx, fromy, tox, toy, postx, posty, 4, 135, 0.04, new3x, new3y);
    calcnextpt(fromx, fromy, tox, toy, postx, posty, 4, 45, 0.04, new4x, new4y);

    if chainlink then begin
        drawline(new1x, newly, new3x, new3y);
        drawline(new4x, new4y, new2x, new2y);
    end; \{ chainlink post \}
    else begin
        drawline(new1x, newly, new2x, new2y);
        drawline(new2x, new2y, new3x, new3y);
        drawline(new3x, new3y, new4x, new4y);
        drawline(new4x, new4y, new1x, newly);
    end; \{ wooden post \}
end; \{ drawpost \}
\end{verbatim}

\begin{verbatim}
begin
    setlinestyle(0);
    dist := invdist(fromx, fromy, tox, toy);
    case lntype of
        1: begin \{ normal \}
            drawline(fromx, fromy, tox, toy);
            end;
        2: begin \{ bold \}
            drawline(fromx, fromy, tox, toy);
            end;
        3: begin \{ dash \}
            setlinestyle(2);
            drawline(fromx, fromy, tox, toy);
            end;
        4: begin \{ R/W \ --- - ---- \}
            setlinestyle(117);
            drawline(fromx, fromy, tox, toy);
            end;
\end{verbatim}
5:begin {CL-Centerline}---<---
setlinestyle(3);
drawline(fromx,fromy,tox,toy);
end;
6:begin {broken}
calcnextpt(fromx,fromy,tox,toy,fromx,fromy,1,0,dist/2,newlx,newly);
calcnextpt(fromx,fromy,tox,toy,newlx,newly,2,0,0.06,new2x,new2y);
drawline(fromx,fromy,new2x,new2y);
calcnextpt(fromx,fromy,tox,toy,new2x,new2y,1,0,0.12,new1x,newly);
drawline(new1x,newly,newlx,toy);
end;
7:begin {chainlink}
drawline(fromx,fromy,tox,toy);
if ptonscreen(fromx,fromy) then
  drawpost(fromx,fromy,tox,toy,fromx,fromy,true);
if ptonscreen(tox,toy) then drawpost(fromx,fromy,tox,toy,tox,toy,true);
tempdist:=l.O;
while tempdist<dist-0.5 do begin
  calcnextpt(fromx,fromy,tox,toy,fromx,fromy,1,0,tempdist,newlx,newly);
  if ptonscreen(newlx,newly) then
    drawpost(fromx,fromy,tox,toy,newlx,newly,true);
  tempdist:=tempdist+l.O;
end; {while}
end;
8:begin {woodfence}
drawline(fromx,fromy,tox,toy);
if ptonscreen(fromx,fromy) then
  drawpost(fromx,fromy,tox,toy,fromx,fromy,false);
if ptonscreen(tox,toy) then drawpost(fromx,fromy,tox,toy,tox,toy,false);
tempdist:=l.O;
while tempdist<dist-0.5 do begin
  calcnextpt(fromx,fromy,tox,toy,fromx,fromy,1,0,tempdist,newlx,newly);
  if ptonscreen(newlx,newly) then
    drawpost(fromx,fromy,tox,toy,newlx,newly,false);
  tempdist:=tempdist+l.O;
end; {while}
end;
9:begin {wall}
calcnextpt(fromx,fromy,tox,toy,fromx,fromy,3,90,0.015,newlx,newly);
calcnextpt(fromx,fromy,tox,toy,fromx,fromy,3,90,0.015,new2x,new2y);
drawline(new1x,newly,new2x,new2y);
calcnextpt(fromx,fromy,tox,toy,fromx,fromy,4,90,0.015,newlx,newly);
calcnextpt(fromx,fromy,tox,toy,fromx,fromy,4,90,0.015,new2x,new2y);
drawline(new1x,newly,new2x,new2y);
else begin
  writeprompt('Invalid Line Type');
  beep;
end;
end; {case}
setlinestyle(0);
end; {displayline}
*******************************************************************************
procedure displaytext(index:integer; startx,starty,height,angle:real;
  txt1n:line);
var topx,topy,temp1x,temp1y,temp2x,temp2y:real;
slen,pixht,scale:integer;
begin
if index=0 then slen:=length(txt1n) else slen:=length(textline[index]);
calcnextpt(0,0,0,0,startx,starty,0,correct360(angle-90),height,topx,topy);
pixht:=windowy(starty)-windowy(topy);
if ((angle=90) and (pixht>=6)) then begin
  scale:=round(pixht/6);
  if index=0 then drawtextw(startx,starty-(height/2),scale,txt1n)
  else drawtextw(startx,starty-(height/2),scale,textline[index]);
end; {horizontal text}
calcnextpt(0,0,0,startx,starty,0,angle,slen*6/7*height,temp1x,temp2y);
temp1x:=temp2x+(topx-startx);
temp1y:=temp2y+(topy-starty);
<calcnextpt(0,0,0,topx,topy,0,angle,slen*6/7*height,temp1x,temp1y);>
drawline(startx,starty,topx,topy);
drawline(startx,starty,temp2x,temp2y);
drawline(temp2x,temp2y,temp1x,temp1y);
drawline(temp1x,temp1y,topx,topy);
end;(displaytext)

<**************************************************>
procedure displaycircle(radx,rady,radius:real);
var xtemp1,ytemp1,xtemp2,ytemp2,d1,d2,d3,d4:real;
begin
xtemp1:=radx; ytemp1:=rady-radius;
angle:=0;
repeat
angle:=angle+6;
ytemp2:=rady - (cos(3.14159*angle/180)*radius);
xtemp2:=radx + (sin(angle*3.14159/180)*radius);
drawline(xtemp1,ytemp1,xtemp2,ytemp2);
d1:=2 * radx * xtemp1;
d2:=2 * radx * xtemp2;
d3:=2 * rady * ytemp1;
d4:=2 * rady * ytemp2;
drawline(d1,ytemp1,d2,ytemp2);
drawline(xtemp1,d3,xtemp2,d4);
drawline(d1,d3,d2,d4);
xtemp1:=xtemp2; ytemp1:=ytemp2;
until angle>90;
end;{displaycircle}

<**************************************************>
procedure displayarc(startx,starty,radx,rady,angle:real);
var tempangle,xtemp1,ytemp1,xtemp2,ytemp2,radius:real;
checkangle:integer;
begin
xtempl:=startx; ytempl:=starty;
radius:=invdist(radx,rady,startx,starty);
tempangle:=inverseazimuth(radx,rady,startx,starty);
checkangle:=round(angle);
if checkangle>=12 then begin
repeat
tempangle:=tempangle+6;
checkangle:=checkangle-6;
ytemp2:=rady - (cos(rad(tempangle))*radius);
xtemp2:=radx + (sin(rad(tempangle))*radius);
drawline(xtemp1,ytemp1,xtemp2,ytemp2);
tempangle:=tempangle+6;
checkangle:=checkangle-6;
ytemp2:=correct360(tempangle);
xtemp2:=radx + (sin(rad(tempangle))*radius);
drawline(xtemp1,ytemp1,xtemp2,ytemp2);
until checkangle<=12;
calcnextpt(radx,rady,startx,starty,radx,rady,5,angle,radius,xtemp2,ytemp2);
drawline(xtemp1,ytemp1,xtemp2,ytemp2);
end;(displayarc)

<**************************************************>
procedure displaysymbol(sytype:integer; pt1x,pt1y,pt2x,pt2y:real);
begin
drawsquare(pt1x,pt1y,pt2x,pt2y,false);
end;(displayrect)

<**************************************************>
procedure displaysymbol(sytype:integer; pt1x,pt1y,pt2x,pt2y,angle,pt3y:real);
procedure displaynorth(stx, sty, angle:real);
begin
  calcnextpt(0, 0, 0, stx, sty, 0, angle, 0.625, b1x, b1y);
  calcnextpt(0, 0, 0, stx, sty, 0, angle, 1.0, b2x, b2y);
  calcnextpt(0, 0, 0, stx, sty, 0, angle, 1.25, b3x, b3y);
  calcnextpt(0, 0, 0, stx, sty, 0, angle, 2.25, endx, endy);
  calcnextpt(endx, endy, stx, sty, b1x, b1y, 6, 90, 0.125, n1x, n1y);
  calcnextpt(endx, endy, stx, sty, b2x, b2y, 6, 90, 0.125, n2x, n2y);
  calcnextpt(endx, endy, stx, sty, b3x, b3y, 6, 90, 0.125, a1x, a1y);
  calcnextpt(0, 0, 0, 0, n1x, n1y, 0, angle, 0.25, n2x, n2y);
  calcnextpt(0, 0, 0, n4x, n4y, 0, angle, 0.125, n3x, n3y);
  drawline(stx, sty, b1x, b1y); drawline(b2x, b2y, endx, endy);
  drawline(endx, sty, b3x, b3y, n1x, n1y);
  drawline(endx, sty, b3x, b3y, n2x, n2y);
  drawline(endx, sty, b3x, b3y, n3x, n3y);
  drawline(endx, sty, b3x, b3y, a1x, a1y);
  drawline(n1x, n1y, n2x, n2y);
  drawline(n3x, n3y, n4x, n4y);
  end; {displaynorth}

procedure displaysymcir(radx, rady:real);
begin
  radius:=0.05;
  xtemp1:=radx; ytemp1:=rady-radius;
  angle:=0;
  repeat
    angle:=angle+30;
    ytemp2:=rady - (cos(3.14159*angle/180)*radius);
    xtemp2:=radx + (sin(angle*3.14159/180)*radius);
    drawline(xtemp1, ytemp1, xtemp2, ytemp2);
    d1:=2 * radx * xtemp1;
    d2:=2 * radx * xtemp2;
    d3:=2 * rady * ytemp1;
    d4:=2 * rady * ytemp2;
    drawline(d1, d1, d2, d2);
    drawline(d3, d3, d4, d4);
    xtemp1:=xtemp2;
  until angle>90;
end; {displaysymcir}

procedure displaybox(ptx, pty:real);
begin
  drawline(ptx-0.03, pty-0.03, ptx+0.03, pty-0.03);
  drawline(ptx+0.03, pty-0.03, ptx+0.03, pty+0.03);
  drawline(ptx+0.03, pty+0.03, ptx-0.03, pty+0.03);
  drawline(ptx-0.03, pty+0.03, ptx-0.03, pty-0.03);
end; {displaybox}

procedure displayarrow(pt1x, pt1y, pt2x, pt2y, bendx, bendy:real);
begin
  drawline(pt1x, pt1y, bendx, bendy);
  drawline(bendx, bendy, pt2x, pt2y);
  calcnextpt(pt2x, pt2y, bendx, bendy, pt2x, pt2y, 6, 90, 0.25, pt1x, pt1y);
  drawline(pt2x, pt2y, pt1x, pt1y);
end; {displayarrow}

procedure displaycross(pt1x, pt1y:real);
begin
  drawline(pt1x, pt1y-0.0625, pt1x, pt1y+0.0625);
  drawline(pt1x-0.0625, pt1y, pt1x+0.0625, pt1y);
end; {displaycross}
end; (displaycross)

begin

end;

********************

procedure redrawobject(var object:objectarray; objno:integer);

begin

  if object[objno].desc <> 0 then begin

    case object[objno].desc of

      1: displayline(code, data[1], data[2], data[3], data[4], data[5], data[6], data[7]);
      2: displaytext(trunc(data[1], data[2], data[3], data[4], data[5], data[6]));
      3: displaycircle(data[1], data[2], data[3], data[4]);
      4: displayarc(data[1], data[2], data[3], data[4], data[5]);
      5: displayrectangle(data[1], data[2], data[3], data[4]);
      6: displaysymbol(code, data[1], data[2], data[3], data[4], data[5], data[6], data[7]);
    end;

  end;

end; (if)

end; (case)

end; (redrawobject)

********************

function showcircle(x, y, rad:real):boolean;

var

  dist1, dist2, dist3, dist4:real;

begin

  dist1 := invdist(x, y, currentminx, currentminy);
  dist2 := invdist(x, y, currentmaxx, currentminy);
  dist3 := invdist(x, y, currentmaxx, currentmaxy);
  dist4 := invdist(x, y, currentminx, currentmaxy);

  if ((not ptonscreen(x, y)) and (dist1 > rad) and (dist2 > rad) and (dist3 > rad) and (dist4 > rad)) or ((dist1 < rad) and (dist2 < rad) and (dist3 < rad) and (dist4 < rad)) then showcircle := false
  else showcircle := true;

end; (showcircle)

********************

function showtext(index:integer; x1, y1, height, angle:real):boolean;

var

  x2, y2:real;

begin

  x2 := x1 + x1 * cos(angle) + y1 * sin(angle);
  y2 := y1 - x1 * sin(angle) + y1 * cos(angle);

  if clipreal(currentminx, currentmaxx, currentminy, currentmaxy, x1, y1, x2, y2) then showtext := true else showtext := false;

end; (showtext)

********************

procedure displayborder(bordercode:integer);

var

  xmin, xmax, ymin, ymax, offset:real;

begin

  if bordercode <> 0 then begin

    if bordercode = 1 then begin

      xmin := 0.5;
      xmax := 8.0;
      ymin := 1.0;
      ymax := 15.5;
      offset := 0;
    end;

  end;

end; (bordercode=1)
if bordercode = 2 then begin
  xmin:=1.0; xmax:=16.0;
  ymin:=0.5; ymax:=10.5;
  offset:=7.5;
  end; (bcode=2)
drawline(xmin,ymin,xmax,ymin);
drawline(xmax,ymin,xmax,ymax);
drawline(xmax,ymax,xmin,ymax);
drawline(xmin,ymax,xmin,ymin);
drawline(xmin+offset,ymax,xmin+offset,ymax-2);
drawline(xmin+offset,ymax-2,xmax,ymax-2);
drawline(xmax-5,ymax-2,xmax-5,ymax);
drawline(xmax-2.5,ymax-2,xmax-2.5,ymax);
displaytext(0,xmin+offset,ymax-0.425,0.105,90,'DATE:');
displaytext(0,xmin+offset,ymax-0.275,0.105,90,'SCALE:');
displaytext(0,xmin+offset,ymax-0.125,0.105,90,'REVISED:');
drawline(xmax-2.25,ymax-0.625,xmax-0.25,ymax-0.625);
end; (if<>0)
end; (displayborder)
(*************************************************************************)
begin
  count:=0;
  writeprompt('Please Wait while Screen is being redrawn:');
displayborder(object[0].desc);
onscreen[0]:=~0;
  repeat
    count:=succ(count);
    if object[count].desc <> 0 then begin
      with object[count] do begin
        case desc of
          1:begin
            x1:=data[1]; y1:=data[2];
            x2:=data[3]; y2:=data[4];
            if clipreal(currentminx,currentmaxx,currentminy,currentmaxy,x1,y1,x2,y2)
              then begin
                addonscreen(onscreen,count);<
                writeprompt('Displaying Line');write(count);
                displayline(code,data[1],data[2],data[3],data[4]);
              end;
          end;
          2:begin
            if showtext(trunc(data[1]),data[2],data[3],data[4],data[5]) then begin
              displaytext(trunc(data[1]),data[2],data[3],data[4],data[5],');'
              writeprompt('Displaying Text');write(count);
              addonscreen(onscreen,count);
            end;if showtext
          end;
          3:if showcircle(data[1],data[2],data[3]) then begin
            writeprompt('Displaying circle');write(count);
            addonscreen(onscreen,count);
            displaycircle(data[1],data[2],data[3]);
          end;
          4:begin
            rad:=invdist(data[1],data[2],data[3],data[4]);
            if showcircle(data[3],data[4],rad)
              then begin
                displayarc(data[1],data[2],data[3],data[4],data[5])
                addonscreen(onscreen,count);
              end;
            writeprompt('Displaying arc');write(count);
          end;
          5:begin
            displayrect(data[1],data[2],data[3],data[4]);
            writeprompt('Displaying Rect');write(count);
            addonscreen(onscreen,count);
          end;
        end of case
      end;
    end;
if ((ptonscreen(data(1J),data(2J)) or (ptonscreen(data(3J),data(4J))) or (ptonscreen(data(5J),data(6J))))
then begin
  writeln('Displaying symbol '); write(count);
  addonscreen(onscreen,count);
  displaysymbol(code,data(1J),data(2J),data(3J),data(4J),data(5J),data(6J));
else desc:=0;
end; {case}
end; {with}
end; {if}
until count=maxobj;
end; {redraw}

procedure selectobject(crossx,crossy:real; var objno:integer; var object:objectarray; var onscreen:onarray);

var pixx,pixy,count:integer;
found:boolean;

begin
  if pd(pixx,pixy) then found:=true;
end; {checkpixels}

procedure erasepixels(var pixx,pixy:integer);

var found:boolean;

begin
  found:=false;
ycount:=-1;
repeat
  xcount:=succ(xcount);
  until (xcount>=2) or found;
  ycount:=succ(ycount);
  until (ycount>=2) or found;
  setcolorblack;
  setwindowmodeoff;
  drawpoint(pixx,pixy);
  setwindowmodeon;
  setcolorwhite;
end; {checkpixels}

procedure checkobject(var object:objectarray; objno:integer);

begin
  if object[objno].desc <> 0 then begin
    with object[objno] do begin
      case desc of
      1:begin
        if (code=7) or (code=8) then displayline(1,data(1J),data(2J),data(3J),data(4J));
      else writeln('Displaying symbol '); write(count);
      end;
      2:displaytext(trunc(data(1J),data(2J),data(3J),data(4J),data(5J),''');
      3:displaycircle(data(1J),data(2J),data(3J));
      4:displayarc(data(1J),data(2J),data(3J),data(4J),data(5J));
      5:displayrect(data(1J),data(2J),data(3J),data(4J));
      6:displaysymbol(code,data(1J),data(2J),data(3J),data(4J),data(5J),data(6J));
      else desc:=0;
    end; {case}
  end; {with}
end; {if}
end; {checkobject}

begin {selectobject}
  pixx:=windowx(crossx);
  pixy:=windowy(crossy);
  erasurepixels(pixx,pixy);
  found:=false;
  objno:=0;
  count:=1;
  writeprompt('Checking for selected object:');
  repeat
    writeprompt('Checking Object');
    write(onscreen[count]);
    checkobject(object,onscreen[count]);
    checkpixels(found,pixx,pixy);
    if not found then count:=succ(count);
  until (count>onscreen[0]) or found;
  if found then begin
    objno:=onscreen[count];
    case object(objno).desc of
      1: writeprompt('Line Selected');
      2: writeprompt('Text Selected');
      3: writeprompt('Circle Selected');
      4: writeprompt('Arc Selected');
      5: writeprompt('Rect Selected');
      6: writeprompt('Symbol Selected');
    end; {case}
    delay(500); 
  end {if found}
  else begin
    writeprompt('No Object Selected');
    beep;
    delay(500);
  end {else}
end {selectobject}

procedure selectpoint(prompt:line; var object:objectarray;
  var onscreen:onarray; var minx,maxx,miny,maxy,crossx,crossy,
  tempx,tempy:real; var movement:boolean);

var ch:char;
  done:boolean;

procedure drawcrosshair(crossx,crossy:real);

var pixx,pixy:integer;

procedure drawcrosspoint(x,y:integer);
begin(crosspoint)
  if PD(x,y) then begin
    setcolorblack;
    drawpoint(x,y);
    setcolorwhite;
  end
  else drawpoint(x,y);
end(crosspoint)

begin(drawcrosshair)

pixx:=windowx(crossx);
pixy:=windowy(crossy);
if (pixx>=pixxmin+4) and (pixx<=pixxmax-4) and (pixy>=pixymin+3) and
  (pixy<=pixymin-3) then begin
  setwindowmodeoff;
  drawcrosspoint(pixx,pixy);
  drawcrosspoint(pixx-4,pixy);
  drawcrosspoint(pixx-3,pixy);
  drawcrosspoint(pixx+3,pixy);
  drawcrosspoint(pixx+4,pixy);
  drawcrosspoint(pixx,pixy-3);
  drawcrosspoint(pixx,pixy+3);
end {if PD(x,y)
procedure zoom(var object: objectarray; var onscreen: onarray;
  var crossx, crossy, minx, maxx, miny, maxy: real);
var move, done: boolean;
ch: char;
x1, y1, x2, y2, xcent, ycent, dummy1, dummy2: real;

begin
  if ch = 'A' then begin
    x1 := minworldx; y1 := minworldy; x2 := maxworldx; y2 := maxworldy;
  end
  else if ch = 'H' then begin
    if windowsz > 2 then begin
      orientworld(crossx, crossy, minx, maxx, miny, maxy);
      x1 := miny - (maxy - miny) / 2;
      y1 := maxy + (maxy - miny) / 2;
      x2 := maxx + (maxx - minx) / 2;
      y2 := minx - (maxx - minx) / 2;
      checkwindow(x1, x2, y1, y2);
    end
    else begin
      x1 := minworldx; y1 := minworldy; x2 := maxworldx; y2 := maxworldy;
    end
  end
  else if ch = 'T' then begin
    orientworld(crossx, crossy, minx, maxx, miny, maxy);
    x1 := minx + (maxx - minx) / 4;
    y1 := maxy + (maxy - miny) / 4;
    x2 := maxx - (maxx - minx) / 4;
    y2 := maxy - (maxx - minx) / 4;
    checkwindow(x1, x2, y1, y2);
  end
  else if ch = 'C' then begin
    orientworld(crossx, crossy, minx, maxx, miny, maxy);
    x1 := minx; y1 := miny;
    x2 := maxx;
    y2 := maxy;
    checkwindow(x1, x2, y1, y2);
  end
  else done := true;
  if (y2 > y1) and (x2 > x1) and not done then begin
    minx := x1; miny := y1;
    maxx := x2; maxy := y2;
    windowsz := (maxworldx - minworldx) / (maxx - minx);
    if windowsz > 30 then begin
      windowsz := 30;
      xcent := (minx + maxx) / 2;
      ycent := (miny + maxy) / 2;
      minx := xcent - ((maxworldx - minworldx) / windowsz) / 2;
      miny := ycent - ((maxworldy - minworldy) / windowsz) / 2;
      maxx := xcent + ((maxworldx - minworldx) / windowsz) / 2;
      maxy := ycent + ((maxworldy - minworldy) / windowsz) / 2;
    end
    orientworld(crossx, crossy, minx, maxx, miny, maxy);
    windowsz := (maxworldx - minworldx) / (maxx - minx);
    if windowsz > 30 then begin
      windowsz := 30;
      xcent := (minx + maxx) / 2;
      ycent := (miny + maxy) / 2;
      minx := xcent - ((maxworldx - minworldx) / windowsz) / 2;
      miny := ycent - ((maxworldy - minworldy) / windowsz) / 2;
      maxx := xcent + ((maxworldx - minworldx) / windowsz) / 2;
      maxy := ycent + ((maxworldy - minworldy) / windowsz) / 2;
    end
  end
  end;

end;
end;
changenewindow(dummy1, dummy2, minx, maxx, miny, maxy);
titles(dwgfile, windowsz, scale);
redraw(object, onscreen);
drawcrosshair(crossx, crossy);
end;(if window)
end;(zoom)

(***************************************************************************)
procedure erasecrosshair(crossx, crossy:real);
var pixx, pixy:integer;
(**************************************************************************)
procedure erasecrosspoint(x, y:integer);
begin
 erasecrosspoint;
 if (not PD(x, y)) then drawpoint(x, y)
 else begin
   setcolorwhite;
   drawpoint(x, y);
   setcolorblack;
  end;
end;
(*************************************************************************)
procedure erasecrosshair;
begin
 pixx:=windowx(crossx);
pixy:=windowy(crossy);
setcolorblack;
if (pixx>pixxmin+4) and (pixx<pixxmax-4) and (pixy>pixymin+3) and
(pixy<pixymax-3) then begin
  setwindowmodeoff;
  erasecrosspoint(pixx, pixy);
  erasecrosspoint(pixx-4, pixy);
  erasecrosspoint(pixx-3, pixy);
  erasecrosspoint(pixx+3, pixy);
  erasecrosspoint(pixx+4, pixy);
  erasecrosspoint(pixx, pixy-3);
  erasecrosspoint(pixx, pixy-2);
  erasecrosspoint(pixx, pixy+2);
  erasecrosspoint(pixx, pixy+3);
  setwindowmodeon;
  end;(if within limits)
setcolorwhite;
end;
(*************************************************************************)
procedure lef(t(var min, maxx, crossx, crossy:real; var ch:char);
var factor, pkfactor:real;
done:boolean;
begin
 factor:=0.05/windowsz;
pkfactor:=0.5/windowsz;
repeat
  done:=true;
  erasecrosshair(crossx, crossy);
  crossx:=crossx-factor;
  if crossx<=minx then crossx:=maxx;
  drawcrosshair(crossx, crossy);
  read(kbd, ch);
  if (ch=27) and keypressed then begin
    read(kbd, ch);
    if ch=75 then begin
      done:=false;
      if factor<pkfactor then factor:=factor*1.15;
    end
  else if ch=67 then
    zoom(object, onscreen, crossx, crossy, minx, maxx, miny, maxy);
  end;(ch=27 and kp)
  until done;
end;(left)
procedure right(var minx,maxx,crossx,crossy:real; var ch:char);
  var factor,pkfactor:real;
    done:boolean;
begin
  factor:=0.05/windowsz;
  pkfactor:=0.5/windowsz;
  repeat
    done:=true;
    erasecrosshair(crossx,crossy);
    crossx:=crossx+factor;
    if crossx>=maxx then crossx:=minx;
    drawcrosshair(crossx,crossy);
    read(kbd.ch);
    if (ch=#27) and keypressed then begin
      read(kbd.ch);
      if ch=#77 then begin
        done:=false;
        if factor<pkfactor then factor:=factor*1.15;
      end
      else if ch=#67 then zoom(object,onscreen,crossx,crossy,minx,maxx,miny,maxy);
    end;
  until done;
end;//right

procedure up(var miny,maxy,crossx,crossy:real; var ch:char);
  var factor,pkfactor:real;
  done:boolean;
begin
  factor:=0.05/windowsz;
  pkfactor:=0.5/windowsz;
  repeat
    done:=true;
    erasecrosshair(crossx,crossy);
    crossy:=crossy-factor;
    if crossy<=miny then crossy:=maxy;
    drawcrosshair(crossx,crossy);
    read(kbd.ch);
    if (ch=#27) and keypressed then begin
      read(kbd.ch);
      if ch=#72 then begin
        done:=false;
        if factor<pkfactor then factor:=factor*1.15;
      end
      else if ch=#67 then zoom(object,onscreen,crossx,crossy,minx,maxx,miny,maxy);
    end;
  until done;
end;//up

procedure down(var miny,maxy,crossx,crossy:real; var ch:char);
  var factor,pkfactor:real;
  done:boolean;
begin
  factor:=0.05/windowsz;
  pkfactor:=0.5/windowsz;
  repeat
    done:=true;
    erasecrosshair(crossx,crossy);
    crossy:=crossy+factor;
    if crossy>=maxy then crossy:=miny;
    drawcrosshair(crossx,crossy);
    read(kbd.ch);
    if (ch=#27) and keypressed then begin
      read(kbd.ch);
if ch=#80 then begin
  done:=false;
  if factor<pkfactor then factor:=factor*1.15;
end
else if ch=#67 then
  zoom(object,onscreen,crossx,crossy,minx,maxx,miny,maxy);
end;
until done;
end;(down)

procedure upleft(var minx,maxx,miny,maxy,crossx,crossy:real;var ch:char);
var factor,pkfactor:real;
done:boolean;
beginn
  factor:=0.05/windowsz;
  pkfactor:=0.5/windowsz;
  repeat
    done:=true;
    erasetcrosshair(crossx,crossy);
    crossy:=crossy-factor;
    crossx:=crossx-factor;
    if crossy<=miny then crossy:=maxy;
    if crossx<=minx then crossx:=maxx;
    drawcrosshair(crossx,crossy);
    read(kbd,ch);
    if (ch=#27) and keypressed then begin
      read(kbd,ch);
      if ch=#71 then begin
        done:=false;
        if factor<pkfactor then factor:=factor*1.15;
      end
      else if ch=#67 then
        zoom(object,onscreen,crossx,crossy,minx,maxx,miny,maxy);
    end;
  until done;
end;{upleft}

procedure upright(var minx,maxx,miny,maxy,crossx,crossy:real;var ch:char);
var factor,pkfactor:real;
done:boolean;
beginn
  factor:=0.05/windowsz;
  pkfactor:=0.5/windowsz;
  repeat
    done:=true;
    erasetcrosshair(crossx,crossy);
    crossy:=crossy+factor;
    crossx:=crossx+factor;
    if crossy<=miny then crossy:=maxy;
    if crossx<=minx then crossx:=maxx;
    drawcrosshair(crossx,crossy);
    read(kbd,ch);
    if (ch=#27) and keypressed then begin
      read(kbd,ch);
      if ch=#73 then begin
        done:=false;
        if factor<pkfactor then factor:=factor*1.15;
      end
      else if ch=#67 then
        zoom(object,onscreen,crossx,crossy,minx,maxx,miny,maxy);
    end;
  until done;
end;{upright}

procedure dwnleft(var minx,maxx,miny,maxy,crossx,crossy:real;var ch:char);
var factor,pkfactor:real;
done:boolean;
begi
factor:=0.05/windowsz;
pkfactor:=0.5/windowsz;
repeat
done:=true;
erasecrosshair(crossx,crossy);
crossy:=crossy+factor;
crossx:=crossx+factor;
if crossy>=maxy then crossy:=miny;
if crossx<=minx then crossx:=maxx;
drawcrosshair(crossx,crossy);
read(kbd,ch);
if (ch=#27) and keypressed then begin
  read(kbd,ch);
  if ch=#79 then begin
    done:=false;
    if factor<pkfactor then factor:=factor*1.15;
  end;
  if ch=#67 then zoom(object,onscreen,crossx,crossy,minx,maxx,miny,maxy);
end;
until done;
end;

(************************
procedure downright(var minx,maxx,miny,maxy,crossx,crossy:real;var ch:char);
var factor,pkfactor:real;
begin
factor:=0.05/windowsz;
pkfactor:=0.5/windowsz;
repeat
done:=true;
erasecrosshair(crossx,crossy);
crossy:=crossy+factor;
crossx:=crossx+factor;
if crossy>=maxy then crossy:=miny;
if crossx>=maxx then crossx:=minx;
drawcrosshair(crossx,crossy);
read(kbd,ch);
if (ch=#27) and keypressed then begin
  read(kbd,ch);
  if ch=#81 then begin
    done:=false;
    if factor<pkfactor then factor:=factor*1.15;
  end;
  if ch=#67 then zoom(object,onscreen,crossx,crossy,minx,maxx,miny,maxy);
end;
until done;
end;

(************************
begin (selectpoint)
done:=false; movement:=false;
drawcrosshair(crossx,crossy);
repeat
writeprompt('');write(prompt); {clear line and write prompt}
read(kbd,ch);
if (ch=#27) and keypressed then begin
  movement:=true;
  read(kbd,ch);
case ch of
  #75: left(minx,maxx,crossx,crossy,ch);
  #77: right(minx,maxx,crossx,crossy,ch);
  #72: up(miny,maxy,crossx,crossy,ch);
  #80: down(miny,maxy,crossx,crossy,ch);
#71: upleft(minx,maxx,miny,maxy,crossx,crossy,ch);
#73: upright(minx,maxx,miny,maxy,crossx,crossy,ch);
#77: dwnleft(minx,maxx,miny,maxy,crossx,crossy,ch);
#78: dwnright(minx,maxx,miny,maxy,crossx,crossy,ch);
#67: zoom(object,onscreen,crossx,crossy,minx,maxx,miny,maxy);
else beep;
end;(case)
end;{then}
if ch=#13 then done:=true; {enter key has been pressed}
until done;
erasecrosshair(crossx,crossy);
tempx:=crossx;tempy:=crossy;
end;<selectpoint>
<******************************************>
procedure zoomall(var minx,maxx,miny,maxy,crossx,crossy:real; var object:objectarray; var onscreen:onarray);
begin
minx:=minworldx; miny:=minworldy; maxx:=maxworldx; maxy:=maxworldy;
changewindow(crossx,crossy,minx,maxx,miny,maxy);
windowsz:=1;
titles(dwgfile,windowsz,scale);
redraw(object,onscreen);
crossx:=(maxx+minx)/2; crossy:=(maxy+miny)/2;
end;{zoomall}
<*****************************************> overlay
procedure getborder(var object:objectarray;var onscreen:onarray; var minx,maxx,miny,maxy,crossx,crossy:real);
begin
sizecode:integer;
done: boolean;
maxpgx,maxpgy:real;
begin
erasemenus;
gotoxy(74,3);write('BOrder');
repeat
getinteger('Enter DWG size 1 = 8.5x14 2 = 11x17 :',sizecode,done);
until (sizecode>=0) and (sizecode<=2) or done;
if not done then begin
case sizecode of
1:begin
maxpgx:=8.5; maxpgy:=14;
end;
0,2:begin
maxpgx:=17; maxpgy:=11;
end;
end;{case}
if sizecode=1 then begin
maxworldx:=maxpgy;maxworldy:=maxworldx/4/3;
end
else begin
maxworldx:=17; maxworldy:=maxworldx*3/4;
end;
minworldx:=0;minworldy:=0;
minx:=minworldx; maxy:=maxworldy; miny:=minworldy;
object01.desc:=sizecode;
changewindow(crossx,crossy,minx,maxx,miny,maxy);
redraw(object,onscreen);
end;{if not done}
end;{getborder}
<******************************************>
overlay procedure getcogo(var minx,maxx,miny,maxy,crossx,crossy:real; var object:objectarray; var onscreen:onarray);
const maxcogopts=999; maxcogofig=100;
type ptype = (PI,PCR,PCL,TP,TS,SC,CS,ST,CR,SS);
coord = record
  pointnum:integer; {number of coordinate}
  north : real; {northing coord of point }
east : real;  
  (easting coord of point)
pttype : ptype;  
  (type of point)
end;  
  (coord record)
figure = record
  name : integer;
  figpoint : array[1..100] of integer;
  figtype : array[1..100] of ptype;
end;  
  (figure record)
pointarray = array[1..maxcogopts] of coord;
figarray = array[1..maxcogofig] of figure;
cogotype=(CPI, CBK, CPCR, CPCL, CRP, CPT);
polypttype=array[1..100] of cogotype;
var figurepoint:plotarray;
point:pointarray;
fig:figarray;
pointtype:polypttype;
minimumx,maximumy:real;
figno,maxfigpts:integer;
done:boolean;

function getoption:char;
var error:boolean;
ch:char;
begin
  repeat
    error:=false;
    writeprompt('Press either R, T, or ENTER to Rotate Translate or End :');
    read(kbd,ch);ch:=upcase(ch);
    until (ch='R') or (ch='T') or (ch=#13);
  getoption:=ch;
end;  

function gocdfile(checkfile:filestring):boolean;
begin
  assign(testfile,checkfile);
  ( clears for existence of disk file)
  reset(testfile);
  ( clears for existence of disk file)
goodfile:= ( ioresult = 0);
close(testfile);
end; (goodfile)
procedure clearmemory(var point:pointarray;var fig:figarray);
var I,count:integer;
begin
I:=1;
while I<succ(maxcogopts) do begin
point[I].pointnum:=0;
point[I].north:=0;
point[I].east:=0;
point[I].pttype:=PI;
I:=succ(I);
end;
I:=1;
while I<=maxcogofig do begin
fig[I].name:=0;
count:=0;
repeat
   count:=succ(count);
   fig[I].figpoint[count]:=0;
   fig[I].figtype[count]:=PI;
until count=100;
I:=succ(I);
end;
end; (clearmemory)
begin (load)
   error:=false;
   if filename<>'' then begin
      clearmemory(loadpoint,loadfig);
      pointname:=filename + '.pts';
      figname:=filename + '.fig';
      if goodfile(pointname) then begin
         assign(pointfile,pointname);
         reset(pointfile);
         while not eof(pointfile) do begin
            read(pointfile,newpoint);
            loadpoint[newpoint.pointnum]:=newpoint;
         end;
         close(pointfile);
         assign(figfile,figname);
         reset(figfile);
         while not eof(figfile) do begin
            read(figfile,newfig);
            loadfig[newfig.name]:=newfig;
         end;
         close(figfile);
      end;
      if if then 
end; (load)
begin (getfiledata)
done:=false;
repeat
   filename:='';
   error:=false;
   repeat
      writeprompt('Enter Cogo file disk drive A, B, or C: ');
      readln(ch);ch:=upcase(ch);
      until (ch='A') or (ch='B') or (ch='C');
writeprompt('Enter COGO file name : ');
readln(filename);
if filename='' then done:=true
else begin
filename:=concat(ch,'\',filename);
load(filename,point,fig,error);
end;
until not error;
end;

 overlay procedure findfigure(var point:pointarray; var fig:figarray; var figno:integer; var done:boolean);
 var error:boolean;
count:integer;
begin
repeat
repeat
error:=false;
getinteger('Enter FIGURE NO. to be transferred : ',figno,done);
if not done then begin
if fig[figno].name<>figno then begin
error:=true;
writeprompt(' FIGURE DOES NOT EXIST');
beep;delay(1500);
end;
end;
until not error;
if not done then begin
count:=0;
repeat
if fig[figno].figpoint[count]>0 then begin
if point[fig[figno].figpoint[count]].pointnum=0 then begin
error:=true;
writeprompt(' POINTS IN FIGURE HAVE NO COORDINATES');
beep;delay(1500);
end;
end;
end;
until not error;
end;
end;

 overlay procedure createdatarray(var figurepoint:plotarray; var pointtype:polypttype; var point:pointarray; var fig:figarray; var figno:integer; var minimum,maximum:real; var maxfigpts:integer);
 var count:integer;
 figptno:integer;
begin
minimum:=1.0E30;
maximum:=-1.0E30;
count:=0;
repeat
figptno:=fig[figno].figpoint[count];
if figptno<>0 then begin
if figptno<0 then begin
figurepoint[count,1]:=point[fig[figno].east];
figurepoint[count,2]:=point[fig[figno].north];
case fig[figno].figtype[count] of
PI : pointtype[count]:=SPI;
...
PCR :pointtype[countJ:=CPCR;
PCL :pointtype[countJ:=CPCL;
TP :pointtype[countJ:=CPT;
CR :pointtype[countJ:=CRP;
else pointtype[countJ:=CPI;
end;{case}
if point[figptno].east<minimumx then minimumx:=point[figptno].east;
if point[figptno].north>maximumy then maximumy:=point[figptno].north;
end;{else}
end;{case}
if point[figptno].east<minimumx then minimumx:=point[figptno].east;
if point[figptno].north>maximumy then maximumy:=point[figptno].north;
end;{else}
end;{case}
figptno<>0>
until
figptno=0;
maxfigpts:=pred(countJ);
end;{createdatarray>
overlay procedure converttoworld(var figurepoint:plotarray;
    maxfigpts:integer;minimumx,maximumy:real);{SETS COGO COORDS TO WORLD COORD ORIGIN AND}
begin
    count:=0;
    repeat
        count:=succ(countJ);
        figurepoint[count,1J:=(figurepoint[count,1J]-minimumx)/scale + 1.5;
        figurepoint[count,2J:=abs(maximumy-figurepoint[count,2J]/scale + 1.5;
        until count=maxfigpts;
    end;{converttoworld>
overlay procedure rotatefig(var figurepoint:plotarray; maxfigpts:integer);{CONVERTS TO PROPER SCALE WITH 1.5 INCH INDENTATION}
begin
    getreal('Enter ROTATION ANGLE + = CW - = CCW ',angle,done);
    if not done then begin
        setcolorblack;
        drawpoly(figurepoint,maxfigpts);
        setcolorwhite;
        rotatepolygon(figurepoint,maxfigpts,angle);
        drawpoly(figurepoint,maxfigpts);
    end;{notdone}
end;{rotatefig>
overlay procedure translatefig(var figurepoint:plotarray; maxfigpts:integer; var minx,maxx,miny,maxy,crossx,crossy:real);
begin
    selectpoint('Select FROM Point :',object,onscreen,minx,maxx,
        miny,maxy,crossx,crossy,x1,y1,move);
    selectpoint('Select TO Point :',object,onscreen,minx,maxx,
        miny,maxy,crossx,crossy,x2,y2,move);
    if move then begin
        deltay:=y2-y1;
        setcolorblack;
        drawpoly(figurepoint,maxfigpts);
        setcolorwhite;
        translatepolygon(figurepoint,maxfigpts,deltax,deltay);
        drawpoly(figurepoint,maxfigpts);
    end;{if move}
end;{translatefig>
overlay procedure installfig(var figurepoint:plotarray;
    var pointtype:polypttype;maxpts:integer;
    var object:objectarray;var onscreen:onarray);{SETS COGO COORDS TO WORLD COORD ORIGIN AND}
begin
    angle:=real;
if ask('Do you wish to keep figure?') then begin
  setcolorblack;
drawpoly(figurepoint,maxfigpts);
setcolorwhite;
count:=0;
repeat
  count:=succ(count);
  if (pointtype[count]=CPI) or (pointtype[count]=CPT) then begin
    if pointtype[succ(count)]<>CBK then begin
      addtobuffer(object,onscreen,colorpoint[count],1,
                   figurepoint[count],2,figurepoint[count+1,1],
                   figurepoint[count+1,2],figurepoint[count+1,2],0,0);
displayline(1,figurepoint[count],1,figurepoint[count],2,
                  figurepoint[count+1,1],figurepoint[count+1,2],0);
    end;
  end
  else if pointtype[count]=CPCR then begin
    angle:=angleright(figurepoint[count],1,figurepoint[count],2,
                       figurepoint[count+1,1],figurepoint[count+1,2],
                       figurepoint[count+2,1],figurepoint[count+2,2],
                       figurepoint[count+2,2],figurepoint[count,2],
                       angle,1,angle,0);
displayarc(figurepoint[count],1,figurepoint[count],2,
               figurepoint[count+1,1],figurepoint[count+1,2],
               angle,figurepoint[count+1,2],figurepoint[count+1,2],
               angle,figurepoint[count+2,2],figurepoint[count+2,2],
               angle);
count:=succ(count);
end
else if pointtype[count]=CPCL then begin
  angle:=angleright(figurepoint[count+2,1],figurepoint[count+2,2],
                    figurepoint[count+1,2],figurepoint[count+1,1],
                    figurepoint[count+1,1],figurepoint[count+1,2],
                    figurepoint[count+2,1],figurepoint[count+2,2],
                    angle,figurepoint[count+1,2],figurepoint[count+1,2],
                    angle,figurepoint[count+2,2],figurepoint[count+2,2],
                    angle);
count:=succ(count);
end
else if pointtype[count]=CBK then begin
  end;
end (if ask)
else begin
  setcolorblack;
drawpoly(figurepoint,maxfigpts);
setcolorwhite;
end (else)
end; (else)
end; (installfigure)

(***************************************************************************)
begin(getcogo)

erasemenu;
gotoXY(74,5);write('CDgO');
getfiledata(point,fig,dona);
if not done then findfigure(point,fig,figno,dona);
if not done then findfigure(point,fig,figno,dona);
createdatarray(figurepoint,pointtype,point,fig,figno,
minimumx,maximumx,maxfigpts);
drawpoly(figurepoint,maxfigpts);
repeat
  case getoption of
    'R':rotatefig(figurepoint,maxfigpts);
    'T':translatefig(figurepoint,maxfigpts,minx,maxx,miny,maxy,crossx,crossy);
    'G':gdone:=true;
end (case)
until done;
installfigure(figurepoint,pointtype,maxfigpts,object,onscreen);
end (notdone)
end; (getcaga)
(*******************************************************************************)
SOURCE LISTING FOR FILE
PLOTHP.PAS
overlay procedure plot(var minx,maxx,miny,maxy,crossx,crossy:real);
  var object:objectarray; var onscreen:onnarray);
var border,count,maxpen,currentpen:integer;
  pen:penarray;
  resolution:real;               (PLOT ROUTINE FOR HP 7475 A)
  ch:char;

(*--------------------------------------------------------------------------)
procedure pentopt(xl,yl:real);
begin
  writeln(aux,'PA ');
  writeln(aux,'PU ');
  if border = 1 then
    writeln(aux,round((yl-1)/resolution),',',round((xl-0.5)/resolution),';'
  )
end;{pentopt}

(*--------------------------------------------------------------------------)
procedure drawline(x1,y1,x2,y2:real);
begin
  pentopt(x1,y1);
  writeln(aux,'PO ');
  if border=1 then
    writeln(aux,round((y2-1)/resolution),',',round((x2-0.5)/resolution),';'
  )
else
  writeln(aux,round((x2-1)/resolution),',',round((10.5-y2)/resolution),';'
end;{drawline}

(*--------------------------------------------------------------------------)
procedure drawlineb(x1,y1,x2,y2:real);
var inc:real;
begin
  inc:=0.010;
  drawline(x1,y1,x2,y2);
  drawline(x2+inc,y2-inc,x1+inc,y1-inc);
  drawline(x1-inc,y1-inc,x2-inc,y2-inc);
end;{drawline}

(*--------------------------------------------------------------------------)
procedure chpen(penno:integer);
begin
  if penno<>currentpen then begin
    writeln(aux,'SPO;');
    writeln(aux,'SP',penno,';');
    currentpen:=penno;
  end;if
end;{chpen}

(*--------------------------------------------------------------------------)
overlay procedure getpendata(var pen:penarray);
(*--------------------------------------------------------------------------)
overlay procedure getpen(prompt:line; var pen:integer);
var xloc,yloc,code:integer;
input:line;
begin
  pen:=0;
xloc:=wherex;
yloc:=wherey;
repeat
  gotoxyturbo(xloc,yloc);gotoxy(xloc,yloc); clreol;
  write('','prompt'::': ');
  readln(input);
  if input=''' then pen:=1 else begin
    val(input,pen,code);
  end;
end;{getpen}
if code<>0 then pen:=0;
end;(else)
until (pen>=1) and (pen=maxpen);
gotoxyturbo(1,yloc+1); gotoxy(1,yloc+1);
end;(getpen)
(************************************************************************)
procedure getpens(prompt1:line;var pen1:integer;prompt2:line;var pen2:integer);
var xloc,yloc:integer;
begin
xloc:=wherex;
yloc:=wherey;
getpen(prompt1,pen1);
gotoxy(xloc+33,yloc);
gotoxyturbo(xloc+33,yloc);
getpen(prompt2,pen2);
end;(getpens)
(************************************************************************
begin(getpendata)
repeat
clearscreen; gotoxyturbo(3,2); gotoxy(3,2);
writeln('SELECT PENS 1-',maxpen,' FOR: ');
gotoxyturbo(1,3); gotoxy(1,3);
getpen('Border & Title Block',pen1);
getpen('Normal Lines ',pen2);
getpen('Bold Lines ',pen3);
getpen('Dashed Lines ',pen4);
getpen('R/W Lines ',pen5);
getpen('Center Lines ',pen6);
getpen('Broken Lines ',pen7);
getpen('Chain Link Lines ',pen8);
getpen('Wood Fence Lines ',pen9);
getpen('Double Lines ',pen10);
getpen('Size 1 Text = 0.11" ',pen11);
getpen('Size 2 Text = 0.14" ',pen12);
getpen('Size 3 Text = 0.21" ',pen13);
getpen('Size 4 Text = 0.28" ',pen14);
getpen('Size 5 Text = 0.42" ',pen15);
getpen('Circles ',pen16);
getpen('Arrows ',pen17);
getpen('Rectangles ',pen18);
getpen('North Arrow ',pen19);
getpen('Circle Symbol ',pen20);
getpen('Cross Symbol ',pen21);
until ask('Is this information correct ');
repeat
until ask('Is the plotter turned on and ready ');
end;(getpendata)
(************************************************************************
overlay procedure plotline(lntype:integer; fromx,fromy,tx,toy:real);
var dist,tempdist,newlx,newly,new2x,new2y,new3x,new3y,new4x,new4y:real;
count:integer;
(************************************************************************
procedure drawpost(fromx,fromy,tx,toy,postx,posty:real; chainlink:boolean);
var newx1,newx2,newx3,newx4,newy1,newy2,newy3,newy4:real;
begin

calcnxtpt(fromx,fromy,tx,toy,postx,posty,3,45,0.04,newx1,newy1);
calcnxtpt(fromx,fromy,tx,toy,postx,posty,3,135,0.04,newx2,newy2);
calcnxtpt(fromx,fromy,tx,toy,postx,posty,4,135,0.04,newx3,newy3);
calcnxtpt(fromx,fromy,tx,toy,postx,posty,4,45,0.04,newx4,newy4);
if chainlink then begin
drawline(newx1,newy1,newx3,newy3);
drawline(newx4,newy4,newx2,newy2);
end;
else begin
drawline(newx1,newy1,newx2,newy2);
drawline(newx2,newy2,newx3,newy3);
drawline(newx3,newy3,newx4,newy4);
drawline(newx4,newy4,newx1,newy1);
end;
begin
  dist: = invdist(fromx, fromy, tox, toy);
  case intype of
  i: begin (normal)
    chpen(pen[2]);
    writeln(aux, 'LT;'); (set to line type 0)
    drawline(fromx, fromy, tox, toy);
    end;
  2: begin (bold)
    chpen(pen[3]);
    writeln(aux, 'LT;'); (set to line type 0)
    drawlineb(fromx, fromy, tox, toy);
    end;
  3: begin (dash)
    chpen(pen[4]);
    writeln(aux, 'LTS;'); (set to line type 5)
    drawline(fromx, fromy, tox, toy);
    end;
  4: begin (R/W  ---- - ----)
    chpen(pen[5]);
    writeln(aux, 'LT6;'); (set to line type 8)
    drawline(fromx, fromy, tox, toy);
    end;
  5: begin (CL-Centerline  ---- - ----)
    chpen(pen[6]);
    writeln(aux, 'LT5;'); (set to line type 9)
    drawline(fromx, fromy, tox, toy);
    end;
  6: begin (broken)
    chpen(pen[7]);
    writeln(aux, 'LT;'); (set to line type 0)
    calcnxtpt(fromx, fromy, tox, toy, fromx, fromy, 1, 0, dist/2, new1x, new1y);
    calcnxtpt(fromx, fromy, tox, toy, new1x, new1y, 2, 0, 0.06, new2x, new2y);
    drawline(fromx, fromy, new2x, new2y);
    calcnxtpt(fromx, fromy, tox, toy, new2x, new2y, 1, 0, 0.12, new1x, new1y);
    drawline(new1x, new1y, tox, toy);
    end;
  7: begin (chainlink)
    chpen(pen[8]);
    writeln(aux, 'LT;'); (set to line type 0)
    drawline(fromx, fromy, tox, toy);
    drawpost(fromx, fromy, tox, toy, fromx, fromy, true);
    drawpost(fromx, fromy, tox, toy, tox, toy, true);
    tempdist:=0.5;
    while tempdist<dist-0.5 do begin
      calcnxtpt(fromx, fromy, tox, toy, fromx, fromy, 1, 0, tempdist, new1x, new1y);
      drawpost(fromx, fromy, tox, toy, new1x, newly, true);
      tempdist:=tempdist+0.5;
    end; (while)
    end;
  8: begin (woodfence)
    chpen(pen[9]);
    writeln(aux, 'LT;'); (set to line type 0)
    drawline(fromx, fromy, tox, toy);
    drawpost(fromx, fromy, tox, toy, fromx, fromy, false);
    drawpost(fromx, fromy, tox, toy, tox, toy, false);
    tempdist:=0.5;
    while tempdist<dist-0.5 do begin
      calcnxtpt(fromx, fromy, tox, toy, fromx, fromy, 1, 0, tempdist, new1x, newly);
      drawpost(fromx, fromy, tox, toy, new1x, newly, false);
      tempdist:=tempdist+0.5;
    end; (while)
    end;
end; (wooden post)
end; (drawpost)
(xxxxxxxxxxxxxxxxxxxxxx)
begin
dist: = invdist(fromx, fromy, tox, toy);
case intype of
i: begin (normal)
    chpen(pen[2]);
    writeln(aux, 'LT;'); (set to line type 0)
    drawline(fromx, fromy, tox, toy);
    end;
2: begin (bold)
    chpen(pen[3]);
    writeln(aux, 'LT;'); (set to line type 0)
    drawlineb(fromx, fromy, tox, toy);
    end;
3: begin (dash)
    chpen(pen[4]);
    writeln(aux, 'LTS;'); (set to line type 5)
    drawline(fromx, fromy, tox, toy);
    end;
4: begin (R/W  ---- - ----)
    chpen(pen[5]);
    writeln(aux, 'LT6;'); (set to line type 8)
    drawline(fromx, fromy, tox, toy);
    end;
5: begin (CL-Centerline  ---- - ----)
    chpen(pen[6]);
    writeln(aux, 'LT5;'); (set to line type 9)
    drawline(fromx, fromy, tox, toy);
    end;
6: begin (broken)
    chpen(pen[7]);
    writeln(aux, 'LT;'); (set to line type 0)
    calcnxtpt(fromx, fromy, tox, toy, fromx, fromy, 1, 0, dist/2, new1x, new1y);
    calcnxtpt(fromx, fromy, tox, toy, new1x, new1y, 2, 0, 0.06, new2x, new2y);
    drawline(fromx, fromy, new2x, new2y);
    calcnxtpt(fromx, fromy, tox, toy, new2x, new2y, 1, 0, 0.12, new1x, new1y);
    drawline(new1x, new1y, tox, toy);
    end;
7: begin (chainlink)
    chpen(pen[8]);
    writeln(aux, 'LT;'); (set to line type 0)
    drawline(fromx, fromy, tox, toy);
    drawpost(fromx, fromy, tox, toy, fromx, fromy, true);
    drawpost(fromx, fromy, tox, toy, tox, toy, true);
    tempdist:=0.5;
    while tempdist<dist-0.5 do begin
      calcnxtpt(fromx, fromy, tox, toy, fromx, fromy, 1, 0, tempdist, new1x, new1y);
      drawpost(fromx, fromy, tox, toy, new1x, newly, true);
      tempdist:=tempdist+0.5;
    end; (while)
    end;
8: begin (woodfence)
    chpen(pen[9]);
    writeln(aux, 'LT;'); (set to line type 0)
    drawline(fromx, fromy, tox, toy);
    drawpost(fromx, fromy, tox, toy, fromx, fromy, false);
    drawpost(fromx, fromy, tox, toy, tox, toy, false);
    tempdist:=0.5;
    while tempdist<dist-0.5 do begin
      calcnxtpt(fromx, fromy, tox, toy, fromx, fromy, 1, 0, tempdist, new1x, newly);
      drawpost(fromx, fromy, tox, toy, new1x, newly, false);
      tempdist:=tempdist+0.5;
    end; (while)
    end;
overlay procedure plottext(bold,index:integer;startx,starty,height,angle:real;
newline:line);

var dummyx,dummyy,xslope,yslope:real;
et:char;

begin {plottext}
writeln(aux,'CS;');
if bold=0 then writeln(aux,'LB',newline,etx,');'
else
begin
changezero(var input:line);
writeln(aux,'LB',textline[index],etx,');'
end; {else}
end; (plottextln)

9:begin (wall)
chpen(pen[10]);
writeln(aux,'LT;'); {set to line type 0}
calcnexptpt (fromx,fromy,tox,toy,fromx,fromy,3,90,0.015,newx,newy);
calcnexptpt (fromx,fromy,tox,toy,tox,toy,3,90,0.015,new2x,new2y);
drawline(newx,newy,new2x,new2y);
calcnexptpt (fromx,fromy,tox,toy,fromx,fromy,4,90,0.015,newx,newy);
calcnexptpt (fromx,fromy,tox,toy,tox,toy,4,90,0.015,new2x,new2y);
drawline(newx,newy,new2x,new2y);
end;
else begin
writeprompt('Invalid Line Type '); beep;
end; (case)
 writeln(aux,'LT;'); {set to line type 0}
end; (plotline)

(**************************************************************************)
overlay procedure plottext(bold,index:integer;startx,starty,height,angle:real;
newline:line);

begin {plottextln}
writeln(aux,'CS;');
if bold=0 then writeln(aux,'LB',newline,etx,');'
else 
begin
changezero(var input:line);
writeln(aux,'LB',textline[index],etx,');'
end; {else}
end; (plottextln)

(*-----------------------------------------------------------------------*
procedure plottext(bold,index:integer;startx,starty,height,angle:real;
newline:line);

var dummyx,dummyy,xslope,yslope:real;
et:char;

begin {plottext}
writeln(aux,'CS;');
if bold=0 then writeln(aux,'LB',newline,etx,');'
else
begin
changezero(var input:line);
writeln(aux,'LB',textline[index],etx,');'
end; {else}
end; (plottextln)

(**************************************************************************)
overlay procedure plottext(bold,index:integer;startx,starty,height,angle:real;
newline:line);

begin {plottextln}
writeln(aux,'CS;');
if bold=0 then writeln(aux,'LB',newline,etx,');'
else
begin
changezero(var input:line);
writeln(aux,'LB',textline[index],etx,');'
end; {else}
end; (plottextln)}
plottextln(index, startx, starty, height, angle, newline);
end
else begin (plot bold text)
  if height = 0.105 then chpen(pen[12])
  else if height = 0.14 then chpen(pen[14])
  else if height = 0.21 then chpen(pen[16])
  else if height = 0.28 then chpen(pen[18])
  else if height = 0.42 then chpen(pen[20])
  else chpen(1);
plottextln(index, startx, starty, height, angle, newline);
plottextln(index, startx+0.010, starty, height, angle, newline);
if height>0.14*2-4
begin
plottextln(index, startx, starty-1.01, height, angle, newline);
plottextln(index, startx, starty+c.010, height, angle, newline);
end;  (extra bold)
end; (plottext)

********************************************
overlay procedure plotcircle(radx, rady, radius:real);
begin
  chpen(pen[21]);
  penopt(radx, rady);
  writeln(aux,'PA');  (specify absolute coords)
  writeln(aux,'CI ',round(radius/resolution),',',5');  (draw circle)
end; (plotcircle)

********************************************
overlay procedure plotarc(startx, starty, radx, rady, angle:real);
begin
  chpen(pen[22]);
  penopt(startx, starty);  (goto start
  of arc and put pen down)
  writeln(aux,'PB');
  if border=1 then
  write(aux,'AA ',round((rady-1)/resolution),',',
  round((radx-0.5)/resolution),',');
  else
  write(aux,'AA ',round((radx-1)/resolution),',',
  round((10.5-rady)/resolution),',');
  writeln(aux,'AA ',round(angle),',');
end; (plotarc)

********************************************
overlay procedure plotrect(pt1x, pt1y, pt2x, pt2y:real);{pl 1ct-rect}
begin
  chpen(pen[23];
  drawline(pt1x, pt1y, pt2x, pt1y);
  drawline(pt2x, pt1y, pt2x, pt2y);
  drawline(ptlx, pt2y, pt1x, pt2y);
end; (plotrect)

********************************************
overlay procedure plotsymbol(sytype:integer;
pt1x, pt1y, pt2x, pt2y, angle, pt3y:real);
procedure plotnorth(stx, sty, angle:real);
var endx, endy, bx1, by1, b2x, b2y, b3x, b3y, a1x, a1y, a2x, a2y,
n1x, n1y, n2x, n2y, n3x, n3y, n4x, n4y, n5x, n5y, n6x, n6y:real;
begin
  chpen(pen[24]);
  calcnextpt(0, 0, 0, stx, sty, 0, angle, 0.625, blx, b1y);
  calcnextpt(0, 0, 0, stx, sty, 0, angle, 1.0, b2x, b2y);
  calcnextpt(0, 0, 0, stx, sty, 0, angle, 1.25, b3x, b3y);
  calcnextpt(0, 0, 0, stx, sty, 0, angle, 2.25, endx, endy);
  calcnextpt(endx, endy, stx, sty, blx, b1y, 6, 90, 0.125, n4x, n4y);
  calcnextpt(endx, endy, stx, sty, blx, b1y, 5, 90, 0.125, n1x, n1y);
  calcnextpt(endx, endy, stx, sty, b2x, b2y, 6, 90, 0.125, n6x, n6y);
  calcnextpt(endx, endy, stx, sty, b2x, b2y, 5, 90, 0.125, n3x, n3y);
  calcnextpt(endx, endy, stx, sty, b3x, b3y, 6, 90, 0.125, a2x, a2y);
end;
calcnextpt (endx, endy, stx, sty, b3x, b3y, 5, 90, 0.125, alx, aly);
calcnextpt (0, 0, 0, nix, nly, 0, angle, 0.25, n2x, n2y);
calcnextpt (0, 0, 0, n4x, n4y, 0, angle, 0.125, n5x, n5y);
drawline (stx, sty, b3x, b3y); drawline (b2x, b2y, endx, endy);
drawline (alx, aly, b3x, b3y); drawline (a1x, a2y, a2y);
drawline (nix, nly, n3x, n3y); drawline (n4x, n4y, n6x, n6y);
drawline (n2x, n2y, n4x, n4y); drawline (n5x, n5y, n3x, n3y);
end; {plotnorth}

(* ***************************************************************************)
procedure plot symc ir (radx, rad y: real);
var radius: real;
begin
  chpen (pen [261]);
  radius := 0.05;
  pentopt (radx, rad y);
  writeln (aux, 'PA'); {specify absolute coords}
  writeln (aux, 'CI ', round (radius/resolution), ',30'); {draw circle}
end; {plot symc ir}

(* ***************************************************************************)
procedure plot box (ptx, pty: real);
begin
  chpen (pen [271]);
  drawline (ptx-0.03, pty-0.03, ptx+0.03, pty-0.03);
  drawline (ptx+0.03, pty+0.03, ptx-0.03, pty+0.03);
  drawline (ptx-0.03, pty+0.03, ptx+0.03, pty-0.03);
end; {plotbox}

(* ***************************************************************************)
procedure plotarrow (ptx1, pty1, pt2x, pt2y, bendx, bendy: real);
begin
  chpen (pen [253]);
  drawline (ptx1, pty1, bendx, bendy);
  drawline (bendx, bendy, pt2x, pt2y);
  calcnextpt (pt2x, pt2y, bendx, bendy, pt2x, pt2y, 6, 4, 0.2, ptx1, pty1);
  drawline (pt2x, pt2y, ptx1, pty1);
end; {plotar row}

(* ***************************************************************************)
procedure plot cross (ptx, pty: real);
begin
  chpen (pen [281]);
  drawline (ptx, pty-0.0625, ptx, pty+0.0625);
  drawline (ptx-0.0625, pty, ptx+0.0625, pty);
end; {plotcross}

(* ***************************************************************************)
begin
  case sytype of
    1: plotnorth (ptx, pty, angle);
    2: plot symc ir (ptx, pty);
    3: plot box (ptx, pty);
    4: plot arrow (ptx, pty, pt2x, pt2y, angle, pt3y);
    5: plot cross (ptx, pty);
  end; {case}
end; {plot symbol}

(* ***************************************************************************)
procedure plot border (bordercode: integer);
var xmin, xmax, ymin, ymax, offset, incr: real;
(* ***************************************************************************)
procedure titleblock (xmin, xmax, ymin, ymax, offset: real);
(* ***************************************************************************)
procedure plot logo (x, y, rad: real);
begin
  chpen (pen [211]);
  drawline (x-0.399*rad, y+0.917*rad, x+0.399*rad, y-0.917*rad);
  drawline (x-0.917*rad, y-0.399*rad, x+0.917*rad, y+0.399*rad);
  drawline (x-0.993*rad, y+0.472*rad, x+0.472*rad, y-1.094*rad);
  drawline (x-0.432*rad, y+0.993*rad, x-0.475*rad, y+1.094*rad);
plotcircle(x, y, rad);
plotarc(x-0.399*rad, y-0.917*rad, x-0.420*rad, y-0.183*rad, 120.76);
end; {plotlogo}

begin {titleblock}
  chpen (pen[13]);
  plottext(1, 0, xmin+offset, ymax-1.25, 0.29, 90, ' GEODESY');
  chpen (pen[12]);
  plottext(1, 0, xmin+offset, ymax-1.20, 0.105, 90, ' Professional Services Inc.');
  chpen (pen[11]);
  plottext(0, 0, xmin+offset, ymax-0.875, 0.105, 90, ' P.O. Box 22346');
  plottext(0, 0, xmin+offset, ymax-0.325, 0.105, 90, ' Lk. Buena Vista, Fl. 32830');
  plottext(0, 0, xmin+offset, ymax-0.075, 0.105, 90, ' (305) 239-0462');
  plottext(0, 0, xmin+offset, ymax-0.025, 0.105, 90, ' SCALE');
  plottext(0, 0, xmin+offset, ymax-0.075, 0.105, 90, ' REVISED');
  plottext(0, 0, xmax-2.4, ymax-1.67, 0.105, 90, ' Certified correct in');
  plottext(0, 0, xmax-2.4, ymax-1.504, 0.105, 90, ' accordance with Chap.');
  plottext(0, 0, xmax-2.4, ymax-1.336, 0.105, 90, ' 21 HH-6, F.A.C. and');
  plottext(0, 0, xmax-2.4, ymax-1.168, 0.105, 90, ' Chapter 472, Florida');
  plottext(0, 0, xmax-2.4, ymax-1.0, 0.105, 90, ' Statutes.');
  plottext(0, 0, xmax-2.4, ymax-0.875, 0.105, 90, ' Richard H. Garvey');
  plottext(0, 0, xmax-2.4, ymax-0.625, 0.105, 90, ' Florida P.L.S. 381');
  chpen (pen[23]);
  drawline(xmin, ymin+incr, xmax, ymin+incr);
  drawline(xmax, ymax-2*incr, xmin, ymax-2*incr);
  drawline(xmax-2*incr, ymax, xmin-2*incr, ymax);
  titlablock(xmin, xmax, ymin, ymax, offset);
end; {titleblock}

if bordercode<>0 then begin
  if bordercode = 1 then begin
    xmin = 0.5; xmax = 8.0;
    ymin = 1.0; ymax = 13.5;
    offset = 0;
  end; {bc ode = 1}
  if bordercode = 2 then begin
    xmin = 1.0; xmax = 16.0;
    ymin = 0.5; ymax = 10.5;
    offset = 7.5;
  end; {bc ode = 2}
  incr = 0.10;
  chpen (pen[11]);
  drawline(xmin, ymin, xmax, ymax);
  drawline(xmax, ymax, xmin+incr, ymin+incr);
  drawline(xmin+2*incr, ymax, xmax, ymax+2*incr);
  drawline(xmax-2*incr, xmin, xmin+2*incr, ymin);
  drawline(xmax, ymax, ymin, xmin, ymax);
  drawline(xmin, ymax, xmin+incr, xmax, ymax+incr);
  drawline(xmax, ymax-2*incr, xmin, ymin-2*incr);
  drawline(xmin, ymin, ymax, xmin, xmax);
  drawline(xmin+2*incr, ymax, xmin+2*incr, ymax);
  drawline(xmin-offset, ymax, xmin+offset, ymax-2);
  drawline(xmin+2*incr, ymax-2, xmax, ymax-2);
  drawline(xmin+2.5, ymax-2, xmax-2.5, ymax);
  titlablock(xmin, xmin, ymin, ymax, offset);
end; {if<>0}
end; {plotborder}

begin (plot)
erasemenu;
gotoXY(74,24);write('Plot');
count:=0;
border:=object[0].desc;
maxpen:=6;
if ask('Do you wish to begin plotting') then begin
  getpendata(pen);
  resolution:=0.00098;
  writeln(aux,'IN'); (select plotter)
  writeln(aux,'PS1'); (large paper)
  writeln(aux,'SP1'); (get pen 1)
  writeln(aux,'VS10'); (set vel=10cm/sec)
  plotborder(border);
  repeat
    count:=succ(count);
    if object[count].desc <> 0 then begin
      gotoxy(1,25);cleol; write('Plotting object ',count);
      with object[count] do begin
        case desc of
          1:plotline(code,data[1],data[2],data[3],data[4]);
          2:plottext(code,trunc(data[1]),data[2],data[3],data[4],data[5],''');
          3:plotcircle(data[1],data[2],data[3],data[4],data[5]);
          4:plotarc(data[1],data[2],data[3],data[4],data[5],data[6]);
          5:plotrect(data[1],data[2],data[3],data[4],data[5],data[6]);
          6:plotsymbol(code,data[1],data[2],data[3],data[4],data[5],data[6]);
          else desc:=0;
        end; (case)
      end; (with)
    end; (if)
  until count=maxobj;
  writeln(aux,'SP0'); (Reset plotter)
  cleascreen;
  zoomall(minx,maxx,miny,maxy,crossx,crossy,object,onscreen);
end; (if plot)
end; (plot)
SOURCE LISTING FOR FILE

PLOTHI.PAS
overlay procedure plot(var minx, maxx, miny, maxy, crossx, crossy: real;
   var object: objectarray; var onscreen: onarray);

procedure pentopt(x1, y1: real);
begin
  writeln(aux, 'U ');
  writeln(aux, 'A ');
  if border=1 then
    writeln(aux, mround((y1-1)/resolution), ',', mround((x1-0.5)/resolution), ' ')
  else
    writeln(aux, mround((x1-1)/resolution), ',', mround((0.5-y1)/resolution), ' ');
end; (pentopt)

procedure drawline(x1, y1, x2, y2: real);
begin
  pentopt(x1, y1);
  writeln(aux, 'D ');
  if border=1 then
    writeln(aux, mround((y2-1)/resolution), ',', mround((x2-0.5)/resolution), ' ')
  else
    writeln(aux, mround((x2-1)/resolution), ',', mround((0.5-y2)/resolution), ' ');
end; (drawline)

procedure drawlineb(x1, y1, x2, y2: real);
var inc: real;
begin
  inc:=0.010;
  drawline(x1, y1, x2, y2);
  drawline(x2+inc, y2-inc, x1+inc, y1-inc);
  drawline(x1+inc, y1+inc, x2+inc, y2+inc);
  drawline(x2-inc, y2+inc, x1-inc, y1+inc);
  drawline(x1-inc, y1-inc, x2-inc, y2-inc);
end; (drawline)

procedure chpen(penno: integer);
begin
  if penno=currentpen then begin
    writeln(aux, 'P', penno, ' ');
    currentpen:=penno;
  end;
end; (chpen)

overlay procedure getpendata(var pen: penarray);

procedure getpen(prompt: line; var pen: integer);
var xloc, yloc, code: integer;
inputline;
begin
  if pen<0 then pen:=0;
  xloc:=whereex;
  yloc:=wherey;
  repeat
...
gotoxyturbo(xloc,yloc);gotoxy(xloc,yloc); clreol;
write('','prompt',':');
readln(input);
if input = ' ' then pen:=1 else begin
  val(input,pen,code);
  if code<>0 then pen:=0;
  end;
until (pen<>1) and (pen<=maxpen);
gotoxyturbo(1,yloc+1); gotoxy(1,yloc+1);
end;(getpen)

procedure getpens(prompt1:line;var pen1:integer;prompt2:line;var pen2:integer);
var
  xloc,yloc:integer;
begin
  xloc:=wherex; yloc:=wherey;
  getpen(prompt1,pen1);
  gotoxy(xloc+33,yloc);gotoxyturbo(xloc+33,yloc);
  getpen(prompt2,pen2);
end;

procedure plotline(intype:integer; fromx,fromy,tox,toy:real);
begin
  if intype=0 then begin
    tempdist:=sqrt((tox-fromx)^2+(toy-fromy)^2);
    if tempdist<0.02 then return;
    count:=0;
    dist:=sqrt((tox-fromx)^2+(toy-fromy)^2);
    while dist>0.01 do begin
      tempdist:=sqrt((tox-fromx)^2+(toy-fromy)^2);
      if tempdist<0.02 then return;
      count:=count+1;
      dist:=dist-0.01;
    end;
    if intype=3 then begin
      new1x:=fromx-tempdist/2;
      new2x:=tox+tempdist/2;
      new1y:=fromy-tempdist/2;
      new2y:=toy+tempdist/2;
      writeln('Warning: line too short');
      writeln('Try use shorter pens');
    end;
  end;
end;

overlay procedure drawline(intype:integer; fromx,fromy,tox,toy:real; chainlink:boolean);
var
  tempdist,new1x,new1y,new2x,new2y:new1x,newyny,new2x,new2y:real;
begin
  if intype=0 then begin
    tempdist:=sqrt((tox-fromx)^2+(toy-fromy)^2);
    if tempdist<0.02 then return;
    count:=0;
    dist:=sqrt((tox-fromx)^2+(toy-fromy)^2);
    while dist>0.01 do begin
      tempdist:=sqrt((tox-fromx)^2+(toy-fromy)^2);
      if tempdist<0.02 then return;
      count:=count+1;
      dist:=dist-0.01;
    end;
    if intype=3 then begin
      new1x:=fromx-tempdist/2;
      new2x:=tox+tempdist/2;
      new1y:=fromy-tempdist/2;
      new2y:=toy+tempdist/2;
      writeln('Warning: line too short');
      writeln('Try use shorter pens');
    end;
  end;
end;
else begin
  drawline(newlx,newly,new2x,new2y);
doend;
else begin
  drawline(new2x,new2y,new3x,new3y);
doend;
else begin
  drawline(new3x,new3y,new4x,new4y);
doend;
else begin
  drawline(new4x,new4y,newlx,newly);
doend;
end;(wooden post)
end;(drawpost)

(******************************************************************************)
begin
  dist:invdist(fromx,fromy,tox,toy);
c:asalntyp
  of
  l:beign
    {normal}
      chpen(penC2J);
      writeln(aux,'0' );
      (set to line type 0)
      drawline(fromx,fromy,tox,toy);
doend;
  drawline(fromx,fromy,tox,toy);
end;
2:beign {bold}
  chpen(penC3J);
  writeln(aux,'0' );
  (set to line type 0)
  drawline(fromx,fromy,tox,toy);
doend;
3:beign {dash}
  chpen(penC4J);
  writeln(aux,'0' );
  (set to line type 5)
  drawline(fromx,fromy,tox,toy);
doend;
4:beign {R/W --- - - ---}
  chpen(penC5J);
  writeln(aux,'0' );
  (set to line type 0)
  drawline(fromx,fromy,tox,toy);
doend;
5:beign {CL-Centerline --- - - ---}
  chpen(penC6J);
  writeln(aux,'0' );
  (set to line type 9)
  drawline(fromx,fromy,tox,toy);
doend;
6:beign {broken}
  chpen(penC7J);
  writeln(aux,'0' );
  (set to line type 0)
  calcnexptpt(fromx,fromy,tox,toy,fromx,fromy,1,0,dist/2,newlx,newly);
  calcnexptpt(fromx,fromy,tox,toy,newlx,newly,2,0,0.06,new2x,new2y);
  drawline(fromx,fromy,new2x,new2y);
  calcnexptpt(fromx,fromy,tox,toy,new2x,new2y,1,0,0.12,newlx,newly);
  drawline(newlx,newly,tox,toy);
doend;
7:beign {chainlink}
  chpen(penC8J);
  writeln(aux,'0' );
  (set to line type 0)
  drawline(fromx,fromy,tox,toy);
  drawpost(fromx,fromy,tox,toy,fromx,fromy,true);
  drawpost(fromx,fromy,tox,toy,tox,toy,true);
  tempdist:=0.5;
  while tempdist<dist-0.5 do begin
    calcnexptpt(fromx,fromy,tox,toy,fromx,fromy,1,0,tempdist,newlx,newly);
    drawpost(fromx,fromy,tox,toy,newlx,newly,true);
    tempdist:=tempdist+0.5;
  end;(while)
doend;
8:beign {woodfence}
  chpen(penC9J);
  writeln(aux,'0' );
  (set to line type 0)
  drawline(fromx,fromy,tox,toy);
  drawpost(fromx,fromy,tox,toy,fromx,fromy,false);
  drawpost(fromx,fromy,tox,toy,tox,toy,false);
  tempdist:=0.5;
  while tempdist<dist-0.5 do begin
    end;(while)
calcnextpt(fromx, fromy, tox, toy, fromx, fromy, 1, 0, tempdist, newlx, newly);
drawpost(fromx, fromy, tox, toy, newlx, newly, false);
tempdist := tempdist + 0.5;
end; {while}
end;
if begin
chpen(pen[10]);
writeln(aux,"LO"); {set to line type 0}
calcnextpt(fromx, fromy, tox, toy, fromx, fromy, 3, 90, 0.015, newlx, newly);
calcnextpt(fromx, fromy, tox, toy, tox, toy, 3, 90, 0.015, newlx, newly);
drawline(newlx, newly, newlx, newly);
calcnextpt(fromx, fromy, tox, toy, fromx, fromy, 4, 90, 0.015, newlx, newly);
calcnextpt(fromx, fromy, tox, toy, tox, toy, 4, 90, 0.015, newlx, newly);
drawline(newlx, newly, newlx, newly);
end;
else begin
writeln(prompt,"Invalid Line Type");
beep;
end;
end; {case}
writeln(aux,"LO"); {set to line type 0}
end; {plotline}

begin
overlay procedure plottext(index:integer; startx, starty, height, angle:real; italic:boolean; newline;line);

var dummyx, dummyy:real;
x_slope, y_slope, size:integer;
icode:string[21];

function changezero(input;line):line;
var zpos:integer;
begin
repeat
zpos := pos(’0’, input);
if zpos<0 then begin
delete(input,zpos,1);
insert(’0’, input, zpos);
end;
until zpos=0;
changezero := input;
end; {changezero}

begin
pentcpt(startx, starty);
if angle>90 then writeln(aux,"V1"); {slow down for angled text}
calcnextpt(0, 0, 0, startx, starty, 0, angle, 1, dummyx, dummyy);
x_slope := mround((dummyx - startx)/resolution);
y_slope := mround((dummyy - starty)/resolution);
size := mround(height/0.035);
if italic then icode := ’I’ else icode := ’NI’;
if border=1 then
write(aux,"S(S’,size,’’,icable,’’,GO,X’,y_slope,’’,Y’,x_slope,’’)");
else
write(aux,"S(S’,size,’’,icable,’’,GO,X’,y_slope,’’,Y’,0-y_slope,’’)");
if index=0 then writeln(aux,newline,"_");
else writeln(aux,changezero(textline[index]),","_ ");
end; {plottextln}

begin
chpen(pen[11]);
if bold=0 then
if height = 0.105 then chpen(pen[11])
else if height = 0.14 then chpen(pen[13])
else if height = 0.21 then chpen(pen[15])
else if height = 0.28 then chpen(pen[17])
else if height = 0.42 then chpen(pen[19])
else chpen(1);
plottextln(index, startx, starty, height, angle, italic, newline);
end else begin (plot bold text)
if height = 0.105 then chpen(pen[12])
else if height = 0.14 then chpen(pen[14])
else if height = 0.21 then chpen(pen[16])
else if height = 0.28 then chpen(pen[18])
else if height = 0.42 then chpen(pen[20])
else chpen(1);
plottextln(index, startx, starty, height, angle, italic, newline);
plottextln(index, startx+0.010, starty, height, angle, italic, newline);
if height>0.14 then begin
plottextln(index, startx, starty-0.010, height, angle, italic, newline);
plottextln(index, startx, starty+0.010, height, angle, italic, newline);
end; (extra bold)
end; (plot bold text)
writeln(aux,'V2 ->; {plot text}
writeln(aux,'A ->; <specify absolute coords}
if border=1 then
writeln(aux,'CC ','mround((rady-1)/resolution),",","
  mround((radx-0.5)/resolution),"'");
else
writeln(aux,'CC ','mround((radx-1)/resolution),",","
  mround((10.5-rady)/resolution),"'");
writeln(aux, 'CC ','mround((radius/resolution),",","
  mround((10.5-rady)/resolution),"'"); (draw circle)
end; (plotcircle)
(***********************************************************************)
overlay procedure plotarc(startx, starty, radx, rady, angle:real);
begin
chpen(pen[22]);
pentopt(startx, starty); (goto start of arc and put pen down)
writeln(aux,'D ');
if border=1 then
writeln(aux,'CA ','mround((rady-1)/resolution),",","
  mround((radx-0.5)/resolution),"'");
else
writeln(aux,'CA ','mround((radx-1)/resolution),",","
  mround((10.5-rady)/resolution),"'");
writeln(aux, '-', 'mround(angle),"'");
end; (plotarc)
(***********************************************************************)
overlay procedure plotrect(ptlx, pty1, pt2x, pt2y:real);
begin
chpen(pen[23]);
writeln(aux,'V4 ');
drawline(ptlx, pty1, pt2x, pt1y);
drawline(pt2x, pty1, pt2x, pt2y);
drawline(pt2x, pt2y, pt1x, pt2y);
drawline(pt1x, pt2y, pt1x, pty1);
writeln(aux,'V2 ');
end; (plotrect)
(***********************************************************************)
overlay procedure plotsymbol(sytype:integer;
ptlx, pty1, pt2x, pt2y, angle, pt3y:real);
procedure plotnorth(stx, sty, angle:real);
var endx, endy, blx, bly, b2x, b2y, b3x, b3y, a1x, a1y, a2x, a2y,
n1x, n1y, n2x, n2y, n3x, n3y, n4x, n4y, n5x, n5y, n6x, n6y: real;

begin
  chpen (pen[241]);
  calcnextpt (0, 0, 0, 0, stx, sty, 0, angle, 0.625, b1x, b1y);
  calcnextpt (0, 0, 0, 0, stx, sty, 0, angle, 1.0, b2x, b2y);
  calcnextpt (0, 0, 0, 0, stx, sty, 0, angle, 1.25, b3x, b3y);
  calcnextpt (0, 0, 0, 0, stx, sty, 0, angle, 2.125, endx, endy);
  calcnextpt (endx, endy, stx, sty, b1x, b1y, 6, 90, 0.125, n4x, n4y);
  calcnextpt (endx, endy, stx, sty, b2x, b2y, 6, 90, 0.125, n5x, n5y);
  calcnextpt (endx, endy, stx, sty, b3x, b3y, 6, 90, 0.125, n3x, n3y);
  calcnextpt (endx, endy, stx, sty, b3x, b3y, 6, 90, 0.125, a2x, a2y);
  calcnextpt (endx, endy, stx, sty, b3x, b3y, 6, 90, 0.125, a1x, a1y);
  calcnextpt (0, 0, 0, 0, n1x, n1y, 0, angle, 0.25, n2x, n2y);
  calcnextpt (0, 0, 0, 0, n4x, n4y, 0, angle, 0.125, n5x, n5y);
  drawline (stx, sty, b1x, b1y);
  drawline (b2x, b2y, endx, endy);
  drawline (a1x, a1y, a2x, a2y);
  drawline (n2x, n2y, n4x, n4y);
  drawline (n3x, n3y, n5x, n5y);
end; plotnorth;

procedure plotsymcir (radx, rady: real);

begin
  chpen (pen[241]);
  radius := 0.05;
  writeln (aux, 'A '); { specify absolute coords }
  if border = 1 then
    writeln (aux, 'C ', mround ((rady-1)/resolution), ', ',
      mround ((radx-0.5)/resolution), ' ')
  else
    writeln (aux, 'C ', mround ((radx-1)/resolution), ', ',
      mround ((10.5-rady)/resolution), ' ');
  writeln (aux, mround (radius/resolution), ', '); { draw circle }
end; plotsymcir;

procedure plotbox (ptx, pty: real);

begin
  chpen (pen[271]);
  drawline (ptx-0.03, pty-0.03, ptx+0.03, pty-0.03);
  drawline (ptx+0.03, pty-0.03, ptx+0.03, pty+0.03);
  drawline (ptx-0.03, pty+0.03, ptx-0.03, pty+0.03);
  drawline (ptx+0.03, pty+0.03, ptx+0.03, pty+0.03);
end; plotbox;

procedure plotarrow (ptlx, pty1, pt2x, pt2y, bendx, bendy: real);

begin
  chpen (pen[254]);
  drawline (ptlx, pty1, bendx, bendy);
  drawline (bendx, bendy, pt2x, pt2y);
  calcnextpt (pt2x, pt2y, bendx, bendy, pt2x, pt2y, 6, 4, 0.25, ptx, pty);
  drawline (pt2x, pt2y, ptx, pty);
end; plotarrow;

procedure plotcross (ptlx, pty1: real);

begin
  chpen (pen[281]);
  drawline (ptlx, pty1-0.0625, ptx, pty1+0.0625);
  drawline (ptlx-0.0625, pty1, ptx+0.0625, pty1);
end; plotcross;

begin
  case sytype of
    1: plotnorth (ptlx, pty1, angle);
    2: plotsymcir (ptlx, pty1);
    3: plotbox (ptlx, pty1);
  end;
4: plotarrow(ptlx, ptly, pt2x, pt2y, angle, pt3y);
5: plotcross(ptlx, ptly)
end; {case}

{******************************************}
procedure plotborder (borderc: integer);
var
  xmin, xmax, ymin, ymax, offset, inc: real;
{*************************}
procedure titleblock (xmin, xmax, ymin, ymax, offset: real);
{******************}
procedure plotlogo (x, y, rad: real);
  begin
    c:hp, (penC21J);
    drawline(x-0.399*rad, y+0.917*rad, x+0.399*rad, y-0.917*rad);
    drawline(x-0.917*rad, y-0.399*rad, x+0.917*rad, y+0.399*rad);
    drawline(x+0.432*rad, y-0.993*rad, x+0.476*rad, y+0.993*rad);
    plotcircle(x, y, rad);
    plotarc(x-0.557*rad, y-0.817*rad, -0.73*rad, y+1.678*rad, 39.973);
    plotarc(x+0.557*rad, y+0.817*rad, -0.73*rad, y-1.678*rad, 39.973);
    plotarc(x-0.399*rad, y+0.917*rad, x+0.420*rad, y+0.183*rad, 130.76);
    plotarc(x+0.399*rad, y-0.917*rad, x-0.420*rad, y-0.183*rad, 130.76);
  end; {plotlogo}
{******************}
begin {titleblock}
  c:hp, (penC14J);
  plottext(1, 0, xmin+offset, ymax-1.575, 0.8, 90, true, 'GEODESY');
  c:hp, (penC11J);
  plottext(1, 0, xmin+offset, ymax-1.35, 0.105, 90, true, 'Professional Services Inc.');
  plotcircle(x, y, rad);
  plotarc(x-0.557*rad, y-0.817*rad, -0.73*rad, y+1.678*rad, 39.973);
  plotarc(x+0.557*rad, y+0.817*rad, -0.73*rad, y-1.678*rad, 39.973);
  plotarc(x-0.399*rad, y+0.917*rad, x+0.420*rad, y+0.183*rad, 130.76);
  plotarc(x+0.399*rad, y-0.917*rad, x-0.420*rad, y-0.183*rad, 130.76);
end; {titleblock}
{*************************}
drawline(xmin, ymin, xmax, ymin);
drawline(xmax, ymin+incr, xmin, ymin+incr);
drawline(xmin, ymin+2*incr, xmax, ymin+2*incr);
drawline(xmax, ymin, xmin, ymax);
drawline(xmax-incr, ymin, xmax, ymax-incr);
drawline(xmax-2*incr, ymin, xmax-2*incr, ymax);
drawline(xmax-incr, ymax, xmax-incr, ymin);
drawline(xmax-2*incr, ymax, xmax-2*incr, ymin);
drawline(xmin, ymax, xmin+2*incr, ymax+2*incr);
drawline(xmax, ymax, xmin, ymax);
drawline(xmax, ymax, xmin+incr, ymin+incr);
drawline(xmax-2*incr, ymax, xmax-2*incr, ymin);
drawline(xmin, ymax, xmin+incr, ymin+incr);
drawline(xmin+incr, ymin, xmin+incr, ymax);
drawline(xmin+2*incr, ymax, xmin+2*incr, ymin);
drawlineb(xmin-offset, ymax, xmin-offset, ymax-2);
drawlineb(xmax-5, ymax-2, xmax-5, ymax);  
drawlineb(xmax-2.5, ymax-2, xmax-2.5, ymax);
writeln(aux, 'V2');  
titleblock(xmin, xmax, ymin, ymax, offset);
end;  

end; (plotborder)

begin (plot)
  erasemenu;
  gotoXY(74, 24); write('Plot');
  count:=0;
  border:=object[01].desc;
  maxpen:=8;
  currentpen:=1;
  if ask('Do you wish to begin plotting?') then begin
    getpendata(pen);
    resolution:=0.005;
    writeln(aux, 'EF');  
    writeln(aux, 'EC');
    writeln(aux, 'V2');  
    writeln(aux, 'H');  
    plotborder(border);
    repeat
      count:=succ(count);
      if object[count].desc <> 0 then begin
        writeprompt(''); write('Plotting object '', count);
        with object[count] do begin
          case desc of
            1:plotline(code, data[1], data[2], data[3], data[4]);
            2:plottext(code, trunc(data[1]), data[2], data[3], data[4], data[5], false, '');
            3:plotcircle(data[1], data[2], data[3]);
            4:plotarc(data[1], data[2], data[3], data[4], data[5]);
            5:plotrect(data[1], data[2], data[3], data[4], data[5]);
            6:plotsymbol(code, data[1], data[2], data[3], data[4], data[5], data[6]);
            else desc:=0;
          end; (case)
        end; (with)
      end; (if)
      until count=maxobj;
    writeln(aux, 'Z');  
    zoomall(minx, maxx, miny, maxy, crossx, crossy, object, onscreen);
  end; (if plot)
end; (plot)
SOURCE LISTING FOR FILE
tgutil.sys
const MaxWorldsGlb=4;
MaxWindowsGlb=16;
MaxPiesGlb=10;
MaxPlotGlb=100;
StringSizeGlb=80;
HeaderSizeGlb=10;
RamScreenGlb:boolean=true;
CharFile:string[StringSizeGlb]='4x6.fon';
MaxProcsGlb=27;
MaxErrsGlb=7;

type wrkstring=array[StringSizeGlb] of string;
WorldType=record
  x1,y1,x2,y2:real;
end;
WindowTitle=record
  x1,y1,x2,y2:integer;
  header:wrkstring;
  drawn,top:boolean;
  size:integer;
end;
worlds=Array[1..MaxWorldsGlb] of WorldType;
windows=Array[1..MaxWindowsGlb] of WindowType;
PlotArray=Array[1..MaxPlotGlb,1..2] of real;
character=Array[1..31] of byte;
CharArray=Array[32..126] of char;
PieType=record
  area:real;
  text:wrkstring;
end;
PileArray=Array[1..MaxPiesGlb] of PieType;
BackgroundArray=Array[0..7] of byte;
LineStyleArray=Array[0..7] of boolean;

var X1WldGlb,X2WldGlb,Y1WldGlb,Y2WldGlb,AxGlb,AyGlb,BxGlb,ByGlb:real;
X1RefGlb,X2RefGlb,Y1RefGlb,Y2RefGlb:integer;
LineStyleGlb,MaxWorldGlb,MaxWindowGlb,WindowNdxGlb:integer;
X1Glb,X2Glb,Y1Glb,Y2Glb:integer;
XTextGlb,YTextGlb,VStepGlb:integer;
PileGlb,DirectModeGlb,ClippingGlb,AxisGlb,HatchGlb:boolean;
MessageGlb,BkGlb,HeaderGlb,TopGlb,GrafModeGlb:boolean;
CntGlb,ColorGlb:byte;
ErrCodeGlb:byte;
LineStyleArrayGlb:LineStyleArray;
ErrorProc:array[0..MaxProcsGlb] of 'WrkString;
ErrorCode:array[0..MaxErrsGlb] of 'WrkString;
const XMaxGlbl=79;  
YMaxGlbl=639;  
YMaxGlbl=199;  
IVStepGlbl=2;  
ScreenSizeGlbl=8191;  
HardwareGrafBasa=SB800;  
FontLoaded:boolean=fal9a;  
MinForeground:integer=0;  
MinBackground:integer=0;  
MaxForeground:integer=15;  
MaxBackground:integer=0;  
AspectFactor=0.44;  
SavaStateGlbl:integer=10;  
ForegroundColorGlbl:integer=15;  

var ScreenGlbl:ScreenPointer;  
ConOutPtrSave:integer;  
Font:IBMFont;  
Stack:Stacks;  
DisplayType:(IBMPCjr,IBMCGA,IBMega,NoDisplay);  

procedure error(ErrProc,ErrCode:integer); forward;  

function HardwarePresent: boolean;  
var i,EquipFlag:integer;  
Info,EGASwitch:byte;  
HP:boolean;  
regs:record case integer of  
1:(ax,bx,cx,dx,bp,si,di,ds,es,flgs:integer);  
2:(al,ah,bl,bh,cl,ch,dl,dh:byte);  
end;  

begin  
HP:=false;  
DisplayType:=NoDisplay;  
with regs do begin  
intr($11,regs);  
EquipFlag:=AX;
an:=$l:..::;
bl:=SlO;
intr<SlO,regs>;
EGASwitch:=CL;
Info:=BH;
end;
if mem[#F000:$FFFE]=$FD then ( PCjr )
begin
MinForeground:=O;
MaxForeground:=15;
MinBackground:=O;
MaxBackground:=15;
DisplayType:=IBMPcJr;
HP:=true;
end
else if ((EquipFlag and 52) in [0,16,32]) and (Info=0) then
begin
MinForeground:=O;
MaxForeground:=15;
MinBackground:=O;
MaxBackground:=15;
DisplayType:=IBMEGA;
HP:=true;
end
if not HP then
if ((EquipFlag and 48) in [16,32] ( CGA )) or
((EquipFlag and 52)=4 ( EGA but not active )) and
(EGASwitch in [4,5,10,11]) ( EGA is mono, CGA for color ) then
begin
MinForeground:=O;
MaxForeground:=15;
MinBackground:=O;
MaxBackground:=15;
DisplayType:=IBMCGA;
HP:=true;
end;
HardwarePresent:=HP;
end;
procedure AllocateRAMScreen;
var test:integer;
begin
new(ScreenGlb);
while ofs(ScreenGlb^)<=O do
begin
dispose(ScreenGlb);
new(test);
new(ScreenGlb);
end;
end;
function BaseAddress(Y: integer): integer;
begin
BaseAddress:=(Y and 1) shl 13 + (Y and -2) shl 5 + (Y and -2) shl 3;
end;
procedure LeaveGraphic;
var regs:record case integer of
  1:(ax,bx,cx,dx,pp,si,dl,ds,es,flgs: integer);
  2:(al,ah,bl,bh,cl,ch,dl,dh: byte);
begin
  regs.ax:=SaveStateGlb;
  intr($10,regs);
  if GrafModeGlb then ConOutPtr:=ConOutPtrSave;
  GrafModeGlb:=false;
end;
procedure DC(C: byte);
begin
  inline(#BA/#9E/9 /#B7/#00/#D1/#E3/#D1/#E3/#D1/#E3/#D1/#E3/#C7/# Font /#8A/#16/
    XTextGlGl /#FE/#8A/#B6/#00/#88/#FA/#8A/#16/ YTextGlGl /#4A/#D1/#02/
    #D1/#E2/#D1/#E2/#A1/ GrafBase /#8E/#C0/#85/#08/#B1/#0D/#88/#C2/#25/
    #01/#00/#C3/#E0/#88/#F0/#88/#C2/#25/#FE/#B1/#03/#D3/#E0/#C3/
    #F0/#FE/#C9/#D3/#E0/#03/#F0/#03/#F7/#8A/#07/#26/#88/#04/#43/#42/
    #FE/#CD/#75/#D7);    
end;

procedure DisplayChar(C: byte);
begin
  if C<8 then
    begin
      if XTextGlGl>1 then XTextGlGl:=XTextGlGl-1;
    end
  else if C=10 then
    begin
      if YTextGlGl<25 then YTextGlGl:=YTextGlGl+1;
    end
  else if C=13 then XTextGlGl:=1
  else
    begin
      DC(C);
      if XTextGlGl<80 then XTextGlGl:=XTextGlGl+1;
    end;
end;

procedure SetIBMPalette(PaletteNumber,Color:integer);
var regs:record case integer of
  1: (ax,bx,cx,dx,bp,si,di,ds,es,flgs: integer);
  2: (al,ah,bl,bh,cl,ch,dl,dh: byte);
end;
begin
  with regs do
    begin
      if PaletteNumber<>2 then
        begin
          ah:=$0B;
          bl:=Color;
          bh:=PaletteNumber;
        end
      else
        begin
          ax:=$1000;
          bl:=1;
          bh:=Color;
        end;
      intr($10,regs);
    end;
end;

procedure SetForegroundColor(Color: Integer);
begin
  case DisplayType of
    IBMPCjr: SetIBMPalette(1,1-(Color and 1));
    IBMCGA: SetIBMPalette(0,Color);
    IBMEGA: SetIBMPalette(2,Color);
  end;
  ForegroundColorGlGl:=Color;
end;

procedure SetBackgroundColor(Color: Integer);
begin
case DisplayType of
  IBMPCjr,
  IBMEGA: SetIBMPalette(0,Color);
end;
if DisplayType=IBMEGA then SetIBMPalette(2,ForegroundColorGlb);
end;

procedure ClearScreen;
begin
  fillchar(mem[grafBase:0000],(ScreenSizeGlb+1) Shl 1,0);
end;

procedure EnterGraphic;
var reg: record case integer of
  1:(ax,bx,cx,dx,bp,si,di,es,flags: integer);
  2:(al,ah,bl,bh,cl,ch,dl,dh: byte);
end;
var regs:reg;
FontFile: file of IBMFont;
begin
  if not FontLoaded then
  begin
    Assign(FontFile,'BxB.FON');
    {$I+} Reset(FontFile); {$I-}
    if IOResult=0 then
    begin
      Read(FontFile,Font);
      Close(FontFile);
    end
    else FillChar(Font,SizeOf(Font),0);
    FontLoaded:=true;
  end;
  regs.ax:=#0f00;
  intr($10,regs);
  if (regs.al<4) or (SaveStateGlb=10) then SaveStateGlb:=regs.al;
  regs.ax:=#0006;
  intr($10,regs);
  SetForegroundColor(MaxForeground);
  if not GrafModeGlb then ConOutPtrSave:=ConOutPtr;
  ConOutPtr:=ofs(DisplayChar);
  GrafModeGlb:=true;
end;

procedure DP<X,Y:integer>:
begin
  inline $#B6/#B1/#B3/#B2/#81/#8D/#8A/#EA/#88/#F8/#89/#F2/#F1/#E3/#E2/#F7/88/
  $8D/#E1/#F3/#F4/#8F/#8B/#8C/#86/
  GrafBase /$80/$53/ColorGlb /$FF/$75/$20/$0B/17/$EB/
  $05/$F6/$D2/$26/$20/$17);
end;

function PD(x,y:integer):boolean;
begin
  PD:=(ColorGlb=0) xor (mem[grafBase:BaseAddress(y) + x shr 3]
    and (128 shr (x and 7)) <> 0);
end;

procedure SetBackgroundS<Background:BackgroundArray>;
var i:integer;
begin
  for i:=Y1RefGlb to Y2RefGlb do
    FillChar(mem[grafBase:BaseAddress(i)+X1RefGlb1,X2RefGlb-X1RefGlb+1,
          Background[i] and 7])
end;
procedure SetBackgroundColor(byt:byte);
var bk:BackgroundArray;
begin
  fillchar(bk,8,byt);
  SetBackgroundColor(bk);
end;

procedure DrawStraight(x1,x2,y:integer); (Draw a horizontal line from x1,y to x2,y)
var i,x:integer;
  DirectModeLoc:boolean;
begin
  if not ((x1<0) or (x1>XMaxGlbl shl 3+7)) and not ((x2<0) or (x2>XMaxGlbl shl 3+7)) and ((y>=0) and (y<=YMaxGlbl)) then
  begin
    DirectModeLoc:=DirectModeGlbl;
    DirectModeGlbl:=true;
    if x1>x2 then
      begin
        x:=x1;
        x1:=x2;
        x2:=x;
      end;
    if x2-x1<=16 then
      begin
        x1:=x1 to x2 do dp(x,y)
      end;
    else
      begin
        x1:=x1+8;
        for i:=(x1-8) to (x1 and -8) do dp(i,y);
        for i:=(x2 and -8) to x2 do dp(i,y);
        FillChar(Mem[GrafBase:BaseAddress(Y)+<Xl Shr 3>],
          (x2 Shr 3)-(x1 Shr 3),ColorGlbl);
      end;
    DirectModeGlbl:=DirectModeLoc;
  end;
end;

procedure GotoXYTurbo(X,Y:integer);
begin
  GotoXY(X,Y); {This will call Turbo's GotoXY}
end;

procedure GotoXY(X,Y:integer); (Further calls to GotoXY will call this procedure)
begin
  if not GrafModeGlbl then GotoXYTurbo(X,Y);
  XTextGlbl:=X;
  YTextGlbl:=Y;
end;

procedure ClrEOLTurbo;
begin
  ClrEOL; {This will call Turbo's ClrEOL}
end;
procedure ClrEOL:  ( Further calls to ClrEOL will call this procedure )
var temp:integer;

begin
  if not GrafModeGlb then ClrEOLTurbo
  else
    begin
      temp:=XTextGlb;
      for XTextGlb:=temp to 80 do DC<32>;
      XTextGlb:=temp;
    end;
  end;

procedure error ( declared in GRAPHIX.SYS: (ErrProc,ErrCode:integer) );
type string2=string[2];
var NLevels,PCValue,XLoc,YLoc:integer;
  ch:char;

function HexString(byt:byte):string2;
  const hex:array [0..15] of char='0123456789ABCDEF';

begin
  HexString:=hex[byt shr 4] + hex[byt and 15];
end;

begin
  if not (ErrProc in [0..MaxProcsGlb]) then
    begin
      LeaveGraphic;
      writeln('FATAL ERROR 1: illegal procedure number ',ErrProc);
      halt;
    end;
  if not (ErrCode in [0..MaxErrsGlb]) then
    begin
      LeaveGraphic;
      writeln('FATAL ERROR 2: illegal error code ',ErrCode);
      halt;
    end;
  ErrCodeGlb:=ErrCode;
  if BrkGlb then LeaveGraphic;
  if MessageGlb or BrkGlb then
    begin
      XLoc:=XTextGlb;
      YLoc:=YTextGlb;
      GotoXY(1,24);
      ClrEOL;
      writeln('Turbo Graphix error #',ErrCode, ' in procedure #',ErrProc);
      if MessageGlb then
        begin
          ClrEOL;
        end;
    end;
  if MessageGlb and BrkGlb then
    begin
      WriteLn;
      WriteLn('Traceback:');
      NLevels:=0;
      repeat
        inline($89/$EB/$88/$BE/ NLevels /$09/$C9/$74/$05/$6E/
        $00/$E2/$F8/$8B/$46/$02/$89/$DD/$89/$86/ PCValue );
        if PCValue<>0 then
          writeln('PCGlb: ',HexString(hi(PCValue-1)),HexString(lo(PCValue-1)));
        NLevels:=NLevels+1;
      until (NLevels>20) or (PCValue=0);  ( Trace back no more than 20 levels )
halt;
end
else if BrkGlb ( and not MessageGlb ) then halt
else if MessageGlb then
begin
write( ' . Hit enter: ' );
repeat
read(Kbd,Ch);
until (Ch='M') or (Ch='C');
if Ch='C then
begin
 LeaveGraphic;
 halt;
end;
GotoXY(XLoc,YLoc);
end;
end;

procedure SetWindowModeOff;
begin
 DirectModeGlb:=true;
end;

procedure SetWindowModeOn;
begin
 DirectModeGlb:=false;
end;

procedure SetClippingOn;
begin
 ClippingGlb:=true;
end;

procedure SetMessageOn;
begin
 MessageGlb:=true;
end;

procedure SetHeaderOff;
begin
 HeaderGlb:=false;
end;

procedure SetHeaderToTop;
begin
 TopGlb:=true;
end;

procedure RemoveHeader(i:integer);
begin
 if i in [1..MaxWindowsGlb] then
 with window[i] do
 begin
 drawn:=false;
top:=true;
header:='';
 end
 else error(22,2);
 end;

procedure SetColorWhite;
begin
 ColorGlb:=255;
end;

procedure SetColorBlack;
begin
  ColorGlb:=0;
end;

procedure SetAspect(aspect:real);
begin
  if aspect<>0.0 then AspectGlb:=abs(aspect)*AspectFactor;
end;

procedure SetLinestyle(ls:integer);
var i: integer;
const lsarray [0..4] of byte=(#FF,#BB,#F8,#E4,#EE);
begin
  if not (ls in [0..4]) then ls:=ls and #FF + $100;
  LineStyleGlb:=ls;
  if ls<5 then ls:=lsa[ls];
  for i:=0 to 7 doLineStyleArrayGlb[7-i]:=(ls shr i) and 1<>0;
  CntGlb:=7;
end;

procedure SelectScreen(i:integer);
begin
  if RamScreenGlb and (i=2) then GrafBase:=Seg(ScreenGlb)
  else GrafBase:=HardwareGrafBase;
end;

procedure DefineWorld(i:integer;
  X_1,Y_1,X_2,Y_2:real);
begin
  if ((X_1<X_2) and (Y_1<Y_2)) and (i in [1..MaxWorldsGlb]) then
    with world[i] do
      begin
        x1:=X_1; y1:=Y_2; x2:=X_2; y2:=Y_1;
        if i>MaxWorldGlb then MaxWorldGlb:=i;
      end;
  else if i in [1..MaxWorldsGlb] then error(1,3)
  else error(1,2);
end;

procedure SelectWorld(i:integer);
begin
  if (i in [1..MaxWorldGlb]) then
    begin
      with world[i] do
      begin
        XLWldGlb:=x1;
        YLWldGlb:=y1;
        X2WldGlb:=x2;
        Y2WldGlb:=y2;
      end;
    else error(2,2);
  end;

procedure ReDefineWindow(i,X_1,Y_1,X_2,Y_2:integer);
begin
  if (i in [1..MaxWindowsGlb]) and (X_1<X_2) and (Y_1<Y_2) and
      (X_1>=0) and (X_2<=XMaxGlb) and (Y_1>=0) and (Y_2<=YMaxGlb) then
    with window[i] do
    begin
      x1:=X_1;
      y1:=Y_1;
      x2:=X_2;
      y2:=Y_2;
      if i>MaxWindowGlb then MaxWindowGlb:=i;
    end
  else if i in [1..MaxWindowsGlb] then error(3,3)
else error(3,2);
end;

procedure DefineWindow(i,X₁,Y₁,X₂,Y₂:integer);
begin
ReDefineWindow(i,X₁,Y₁,X₂,Y₂);
with window[i] do
begin
header:='';
top:=true;
drawn:=false;
end;
end;

procedure SelectWindow(i:integer);
begin
if (i in [1..MaxWindowGlbl]) then
with window[i] do
begin
WindowNxGlbl:=i;
X1RefGlbl:=x1;
Y1RefGlbl:=y1;
X2RefGlbl:=x2;
Y2RefGlbl:=y2;
BxGlbl:=(x2-x1) shl 3+7)/(X2WldGlbl-X1WldGlbl);
ByGlbl:=(y2-y1)/(Y2WldGlbl-Y1WldGlbl);
AxGlbl:=(x1 shl 3)-X1WldGlbl*BxGlbl;
AyGlbl:=y1-Y1WldGlbl*ByGlbl;
if AxisGlbl then
begin
AxisGlbl:=false;
X1Glbl:=0;
Y1Glbl:=0;
X2Glbl:=0;
Y2Glbl:=0;
end;
else error(4,2);
end;
end;

function WindowX(x:real):integer;
var temp:real;
begin
  temp:=AxGlbl+BxGlbl*x;
  if temp>3.2760E04 then temp:=1.0E04;
  if temp<-3.2760E04 then temp:=-1.0E04;
  WindowX:=trunc(temp);
end;

function WindowY(y:real):integer;
var temp:real;
begin
  temp:=AyGlbl+ByGlbl*y;
  if temp>3.2760E04 then temp:=1.0E04;
  if temp<-3.2760E04 then temp:=-1.0E04;
  WindowY:=trunc(temp);
end;

procedure InitGraphic;
var file:file of CharArray;
tfile:text;
test:*integer;
temp:WrkString;
i:integer;
begin
GotoXY(1,1);
if not HardwarePresent then
begin
  ClrScr;
  GotoXY(1,2);
  writeln('Fatal error: graphics hardware not found or not properly activated');
  halt;
end;
MessageGlb:=True;
BrkGlb:=False;
GrafModeGlb:=False;
GetMem(ErrorProc[0],16);
GetMem(ErrorCode[0],24);
ErrorProc[0]:='InitGraphic';
ErrorCode[0]:='ERROR MSG missing';
assign(tfila,'error.msg');
($I-) reset(tfila); ($I+)
if ioresult=0 then
begin
  for i:=0 to MaxProcsGlb do
  begin
    readln(tfila,temp);
    if i<>0 then GetMem(ErrorProc[i],length(temp)+1);
    ErrorProc[i]:=temp;
  end;
  for i:=0 to MaxErrsGlb do
  begin
    readln(tfila,temp);
    if i<>0 then GetMem(ErrorCode[i],length(temp)+1);
    ErrorCode[i]:=temp;
  end;
  readln(tfila,PcGlb);
  close(tfila);
end
else
begin
  GetMem(ErrorProc[i],14);
  ErrorProc[i]:='**UNKNOWN**';
  for i:=2 to MaxProcsGlb do
    ErrorProc[i]:=ErrorProc[i-1];
  for i:=1 to MaxErrsGlb do
    ErrorCode[i]:=ErrorProc[i];
  error(O,1);
end;
for i:=1 to MaxWorldsGlb do
  DefineWorld(i,0,0,XScreenMaxGlb,YMaxGlb);
MaxWorldGlb:=1;
for i:=1 to MaxWindowsGlb do
begin
  DefineWindow(i,0,0,XMaxGlb,YMaxGlb);
  with stack[i] do
  begin
    W.Size:=0;
    Contents:=Nil;
  end;
  RemoveHeader(i);
end;
MaxWindowGlb:=1;
if CharFile<>'' then
begin
  assign(fil,CharFile);
  ($I-) reset(fil); ($I+)
  if ioresult=0 then read(fil,CharSet)
  else error(O,1);
  close(fil);
end;
BrkGlb:=true;
if RamScreenGlb then
begin
AllocateRAMScreen;
SelectScreen(2);
ClearScreen;
end;
SelectScreen(1);
WindowNdxGlb:=1;
SelectWorld(1);
SelectWindow(1);
SetColorWhite;
SetClippingOn;
SetAspect(AspectRatio);
DirectModeGlb:=false;
PieGlb:=false;
SetMessageOn;
SetHeaderOff;
SetHeaderToTop;
ErrCodeGlb:=0;
SetLineStyle(0);
V8StepGlb:=IV8StepGlb;
EnterGraphic;
X1Glb:=0;
X2Glb:=0;
Y1Glb:=0;
Y2Glb:=0;
AxisGlb:=false;
HatchGlb:=false;
Sel ectScre n<1 > ;
WindowNdxGlb:=-1;
SalectWorld<l>;
SelectWindow<l>;
SetClippingOff;
SatAspact<AspactFactcr>;
DirectModeGlbs=falsa;
PiaGlbs=falsa;
SetMass•gaOn;
SatHaadarOff;
SatHaadarToTop;
ErrCodeGlb:=-0;
SatLinaStyla<O>;
VStapGlb:•I~StapGlb;
EntarGraphic;
x1Glb:-0;
x2Glb1=-0;
y1Glb:•O;
y2Glb1=-0;
AxisGlb:=-falsa;
HatchGlb:=falsa;
and;
<******************************>
function clipreal(<clipxl,clipxr,clipyb,clipyt):real;
var x1,y1,x2,y2:real>:boolean;
label return;
type edgen=(left,right,bottom,top); outcode=set of edge;
var c,c1,c2:outcode;
x,y:real;
tempclip:boolean;
<COHEN-SUTHERLAND CLIPPING ALGORITHM>
begin{clipreal)
(tempclip:=false)
beginclipreal
(tempclip:=false)
code(x1,y1,c1); code(x2,y2,c2);
while (c1<>CJ) or (c2<>CJ) do begin
if (c1*c2)<C] then goto return;
c:=c1; if c=C[ then c:=c2;
if left in c then begin {crosses left edge}
y:=y1+(y2-y1)•(clipxl-x1)/(x2-x1);
x:=clipxl;
end
else if right in c then begin {crosses right edge}
y:=y1+(y2-y1)•(clipxr-x1)/(x2-x1);
x:=clipxr;
end
else if bottom in c then begin {crosses bottom edge}
x:=x1+(x2-x1)•(clipyb-y1)/(y2-y1);
y:=clipyb;
end
end
end
end
end
else if top in c then begin (crosses top edge)
  x := x1 + (x2 - x1) * (clipyt - y1) / (y2 - y1);
  y := clipyt;
end;
if c = c1 then begin
  x1 := x; y1 := y; code(x, y, c1);
end else begin
  x2 := x; y2 := y; code(x, y, c2);
end;
tempclip := true;
return
<clipreal> := tempclip;
end;
function clip(var x1, y1, x2, y2: integer): boolean;
var
  xr1, yr1, xr2, yr2, pxmin, pxmax, pymin, pymax: real;
  tempclip: boolean;
begin
  xr1 := x1; yr1 := y1; xr2 := x2; yr2 := y2;
  pxmin := 0; pxmax := 575; pymin := 10; pymax := 190;
  tempclip := clipreal(pxmin, pxmax, pymin, pymax, xr1, yr1, xr2, yr2);
  if tempclip then begin
    x1 := round(xr1); y1 := round(yr1); x2 := round(xr2); y2 := round(yr2);
  end;
clip := tempclip;
end;
procedure DrawPoint(xr, yr: real);
var
  x, y: integer;
begin
  if DirectModeGlb then dp(trunc(xr), trunc(yr))
else begin
  x := WindowX(xr);
  y := WindowY(yr);
  if ClippingGlb then begin
    if (x >= X1RefGlb shl 3) and (x < X2RefGlb shl 3 + 7) then
      if (y >= Y1RefGlb) and (y < Y2RefGlb) then dp(x, y);
    else dp(x, y);
  end;
end;
procedure DrawLine(x1, y1, x2, y2: real);
var
  X1Loc, Y1Loc, X2Loc, Y2Loc: integer;
begin
  X1Loc := x1;
  Y1Loc := y1;
  XStep := 1;
  YStep := 1;
  if x1 > x2 then XStep := -1;
  if y1 > y2 then YStep := -1;
  DeltaX := abs(x2 - x1);
  DeltaY := abs(y2 - y1);
  if DeltaX = 0 then direction := 0;
  else direction := 1;
  while not ((x = x2) and (y = y2)) do
begin
  if LinestyleGlb=0 then dp(x,y)
  else
    begin
      CntGlb:=(CntGlb+1) and 7;
      if LineStyleArrayGlb[CntGlb] then dp(x,y);
    end;
  if direction<0 then
    begin
      y:=y+YStep;
      direction:=direction+DeltaX;
    end
  else
    begin
      x:=x+XStep;
      direction:=direction-DeltaY;
    end;
end;
begin
  if DirectModeGlb then
    DrawLineDirect(trunc(x1),trunc(y1),trunc(x2),trunc(y2))
  else
    begin
      X1Loc:=WindowX(x1);
      Y1Loc:=WindowY(y1);
      X2Loc:=WindowX(x2);
      Y2Loc:=WindowY(y2);
      if clip(X1Loc,Y1Loc,X2Loc,Y2Loc) then
        DrawLineDirect(X1Loc,Y1Loc,X2Loc,Y2Loc);
    end;
end;
procedure DrawLineClipped(x1,y1,x2,y2:integer);
begin
  if clip(x1,y1,x2,y2) then DrawLine(x1,y1,x2,y2);
end;
procedure DrawSquare(X1,Y1,X2,Y2:real;
  fill:boolean);
var
  i,x1loc,y1loc,x2loc,y2loc:integer;
  DirectModeLoc:boolean;
procedure DS(x1,x2,y:integer);
begin
  if LineStyleGlb=0 then DrawStraight(x1,x2,y)
  else DrawLine(x1,y,x2,y);
end;
procedure DSC(x1,x2,y:integer);
begin
  if clip(x1,y,x2,y) then DS(x1,x2,y);
end;
procedure DrawSqr(x1,y1,x2,y2:integer;
  fill:boolean);
var
  i:integer;
begin
  if not fill then
    begin
      DS(x1,x2,y1);
      DrawLine(x2,y1,x2,y2);
      DS(x1,x2,y2);
      DrawLine(x1,y2,x1,y1);
    end;
end;
begin
  if DirectModeGlb then DrawSqr(trunc(X1),trunc(Y1),trunc(X2),trunc(Y2),fill
else
  begin
    DirectModeLoc:=DirectModeGlb;
    DirectModeGlb:=true;
    x1loc:=WindowX(X1);
    y1loc:=WindowY(Y1);
    x2loc:=WindowX(X2);
    y2loc:=WindowY(Y2);
    if not fill then
      begin
        DSC(x1loc,x2loc,y1loc);
        DrawLineClipped(x2loc,y1loc,x2loc,y2loc);
        DSC(x1loc,x2loc,y2loc);
        DrawLineClipped(x1loc,y2loc,x1loc,y1loc);
      end
    else
      for i:=y1loc to y2loc do DSC(x1loc,x2loc,i);
    DirectModeGlb:=DirectModeLoc;
  end;
end;

procedure DrawAscii(var x,y:integer;
  size,chr:byte);
var
  x1ref,x2ref,xpos,ypos,xstart,ystart,xend,yend,xx,yy: integer;
  charbyte: byte;
begin
  x1ref:=x1RefGlb shl 3;
  x2ref:=x2RefGlb shr 3+7;
  for ypos:=0 to 5 do
    begin
      CharByte:=CharSet[ch,(7-ypos) shr 11 shr ((ypos and 1) shr 2)) and $0F;
      for xpos:=0 to 3 do
        if (CharByte shr (3-xpos)) and 1<>0 then
          begin
            xstart:=x+xpos*size;
            xend:=xstart+size-1;
            ystart:=y+1+(ypos-2)*size;
            yend:=ystart+size-1;
            if ClippingGlb then
              begin
                if xstart<x1ref then xstart:=x1ref;
                if xend>x2ref then xend:=x2ref;
                if xstart<Y1RefGlb then ystart:=Y1RefGlb;
                if yend>Y2RefGlb then yend:=Y2RefGlb;
              end;
            for yy:=ystart to yend do
              for xx:=xstart to xend do
                dp(xx,yy);
          end;
    end;
  x:=x+size*6;
end;

procedure DrawText(x,y,scale:integer;
  txt:wrkstring);
var
  LineStyleLoc,code,AsciiValue,StringLen,i,SymbolScale,SymbolCode:integer;
  DirectModeLoc:boolean;
begin
  DirectModeLoc:=DirectModeGlb;
  DirectModeGlb:=true;
  LineStyleLoc:=LineStyleGlb;
  SetLineStyle(0);
  StringLen:=length(txt);
  i:=1;
  while i<=StringLen do
    begin
      AsciiValue:=ord(txt[i]);
      if AsciiValue=27 then
        begin
          SymbolScale:=scale;
          i:=i+1;
          if i<=StringLen then
            begin
              val(txt[i],SymbolCode,code);
              if (i+2<=StringLen) and (ord(txt[i+1])=64) then
                begin
                  val(txt[i+2],SymbolScale,code);
                  i:=i+2;
                end;
              case SymbolCode of
                1:dp(x,y);
                2:dp(x,y);
                3,4:dp(x,y);
                5:dp(x,y);
                6:dp(x,y);
                7:dp(x,y);
                8:dp(x,y);
                end;
              x:=x+3*SymbolScale;
              SymbolScale:=scale;
            end;
        end;
        else DrawAscii(x,y,Scale,AsciiValue);
        i:=i+1;
      end;
  DirectModeGlb:=DirectModeLoc;
  SetLineStyle(LineStyleLoc);
end;

procedure DrawTextW(<x,y:real;
  scale:integer;
  txt:wrkstring>);
begin
  if DirectModeGlb then DrawText(Trunc(<x>),Trunc(<y>),Scale,txt)
  else DrawText(WindowX(<x>),WindowY(<y>),Scale,txt);
end;

procedure DrawBorder;
var ClipLoc,DirectModeLoc:boolean;
  x11,x12:integer;
begin
  DrawHeaderBackground(y1,y2:integer);
  var i:integer;
  begin
    for i:=y1 to y2 do DrawStraight(x11,x12,i);
  end;
end;

procedure DrawHeader;
var Y1Hdr,Y2Hdr,y11,y12:integer;
begin
  with window[WindowNdxGlb] do
    begin
      // Draw header code here.
    end;
end;
if drawn then
  if top then
    begin
      ReDefineWindow(WindowNdxGlb, X1RefGlb, Y1RefGlb+HeaderSizeGlb, X2RefGlb, Y2RefGlb);
      SelectWindow(WindowNdxGlb);
    end
  else
    begin
      ReDefineWindow(WindowNdxGlb, X1RefGlb, Y1RefGlb, X2RefGlb, Y2RefGlb+HeaderSizeGlb);
      SelectWindow(WindowNdxGlb);
    end;
  if TcpGlb then
    begin
      y11:=Y1RefGlb+HeaderSizeGlb;
      y12:=Y2RefGlb;
      Y1Hdr:=Y1RefGlb;
      Y2Hdr:=Y1RefGlb+HeaderSizeGlb-1;
    end
  else
    begin
      y11:=Y1RefGlb;
      y12:=Y2RefGlb-HeaderSizeGlb;
      Y1Hdr:=Y2RefGlb-HeaderSizeGlb+1;
      Y2Hdr:=Y2RefGlb;
    end;
  top:=TopGlb;
  ReDefineWindow(WindowNdxGlb, X1RefGlb, y11, X2RefGlb, y12);
  SelectWindow(WindowNdxGlb);
  DrawHeaderBackground(Y1Hdr, Y2Hdr);
  ColorGlb:=255-ColorGlb;
  DrawText(xl1+2+(x12-xl1-length(header)*6) shr 1, y11, xl2, Y1Hdr+3, header);
  DrawSquare(xl1, y11, xl2, Y2Hdr, false);
  ColorGlb:=255-ColorGlb;
  DrawSquare(xl1, Y1RefGlb, xl2, Y2RefGlb, false);
  drawn:=true;
end;
begin
  DirectModeLoc:=DirectModeGlb;
  DirectModeGlb:=true;
  ClipLoc:=ClippingGlb;
  ClippingGlb:=false;
  x11:=X1RefGlb shl 3;
  x12:=X2RefGlb shl 3+7;
  with window[WindowNdxGlb] do
    if ((HeaderGlb and (length(header)>0)) and (y2-y1)HeaderSizeGlb) and
        (length(header)*6)<abs(x12-xl1)-4) then
      DrawHeader
    else
      begin
        drawn:=false;
        DrawSquare(x11, Y1RefGlb, x12, Y2RefGlb, false);
      end;
  DirectModeGlb:=DirectModeLoc;
  ClippingGlb:=ClipLoc;
end;

**********************************************************
(* TURBO GRAPHIX version 1.03A *)
(* Polygon modification module *)
(* Module version 1.00A *)
procedure RotatePolygonAbout(Var A:PlotArray;
NPoints:integer;
Theta,XO,YO:real);
var C,S,X,Ph:real;
i:integer;
begin
if NPoints>=2 then
begin
Ph:=Pi/180.0*Theta;
C:=cos(Ph);
S:=sin(Ph);
for i:=1 to NPoints do
begin
X:=XO+C*(A[i,1]-XO)-S*(A[i,2]-YO);
A[i,2]:=YO+S*(A[i,1]-XO)+C*(A[i,2]-YO);
A[i,1]:=X;
end;
end
else error(8,4);
end;
procedure RotatePolygon(Var A:PlotArray;
NPoints:integer;
Theta:real);
var XO,YO:real;
i:integer;
begin
XO:=0.0;
YO:=0.0;
for i:=1 to NPoints do
begin
XO:=XO+A[i,1];
YO:=YO+A[i,2];
end;
RotatePolygonAbout(A,NPoints,Theta,XO/NPoints,YO/NPoints);
end;
procedure TranslatePolygon(Var A:PlotArray;
N:integer;
DeltaX,DletaY:real);
var i:integer;
begin
N:=abs(N);
if N>=2 then
for i:=1 to N do
begin
A[i,1]:=A[i,1]+DeltaX;
A[i,2]:=A[i,2]+DeltaY;
end
else error(9,4);
end;
APPENDIX D

PROGRAM RESULTS

This Appendix provides reduced copies of the actual views seen and the results produced as a drawing is edited and plotted. All plotted drawings contained in this Appendix were produced at the offices of Geodesy Professional Services, Inc. and have been used with their permission.
DESCRIPTION: (Per Plat) Lot 61, MONTGOMERY SQUARE, as recorded in Plat Book 24, Pages 8 & 9, Public Records of Seminole County, Florida.

BOUNDARY SURVEY for
The Babcock Company
631 Palm Springs Drive
Altamonte Springs, FL

Certified correct in accordance with Chap. 21, MH-G, F.A.C. and Chapter 472, Florida Statutes.

Richard H. Garvey
Florida P.L.S. 3811
DESCRIPTION (Per Plat)
Lots 26 thru 29, CRANE'S ROOST VILLAS as recorded in Plat Book 23, Pages 74 thru 77, Public Record of Seminole County, Florida.
DESCRIPTION:

Lot 5, LESS the East 13 feet and LESS the North 12 feet thereof, AND Lot 4 LESS the East 16 feet and LESS the South 17 feet thereof TOGETHER WITH Easements for Ingress and Egress over the West 30 feet of the East 42.82 feet of Lot 6 LESS the South 26 feet thereof AND OVER the West 20 feet of the East 32.82 feet of the South 26 feet of Lot 5 AND OVER the West 20 feet of the East 33 feet of the North 12 feet of Lot 5 AND OVER the South 8 feet of the Vesterly 15 feet of Lot 6 AND OVER the North 12 feet of the Vesterly 15 feet of Lot 5; SUBJECT TO an Easement for Ingress and Egress over the Vesterly 15 feet of Lot 5 LESS the North 12 feet thereof AND OVER the West 20 feet of the East 33 feet of the North 12 feet of Lot 6 AND OVER the South 8 feet of the Vesterly 15 feet of Lot 5; SUBJECT TO an Easement for Ingress and Egress over the South 1.6 feet of the North 13.6 feet of Lot 5 TOGETHER WITH an Easement for Maintenance and Overhang over the South 2.4 feet of the North 12 feet of Lot 5; ALL LYING ANO BEING IN Block "E", EAGLE SPRINGS PARK, as recorded in Plat Book "P", Page 47, Public Records of Orange County, Florida, containing 0.1888 acres.

Property is

located in

a Flood Hazard Area per Map

No.20/70-0200 B dated 12/18/81

1860025

GEODESY
Professional Services Inc.
415 Montgomery Place
Suite 120
Altamonte Springs, FL 32714
(305) 682-0071
DATE: January 22, 1986
SCALE: 1" = 20'
REVISED: 1993

BOUNDARY SURVEY
Certified to:
M. S. SINGAPURI
and
Florida National Bank
and
TICO Title Insurance Co.

Certified correct in accordance with Chap. 21 14th F.A.C. and Chapter 472, Florida Statutes.

Richard M. Garvey
Florida P.L.S. 3811
REFERENCES


