TWO DIMENSIONAL COMPUTER AIDED DRAFTING SYSTEM

A Dissertation submitted in partial fulfilment of the requirements for the Degree of

MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE & TECHNOLOGY

PRAMOD GUPTA

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI

DECEMBER 1990

CERTIFICATE

This is to certify that the thesis entitled **"Two Dimensional Computer Aided Drafting System"**, being submitted by me to JawaharLal Nehru University in partial fulfilment of the requirements for the award of the degree of **Master of Technology** is a record of original work done by me under the supervision of **Dr. P. C. Saxena**, Associate professor, School of Computer and Systems Sciences during the Monsoon semester, 1990.

The results reported in this thesis have not been submitted in part or full to any other University or Institution for the award of any degree etc.

(PRAMOD GUPTA)

Prof. N. P. Mukherjee
Dean,
School of Computer and
Systems Sciences,
J. N. U., New Delhi.

Dr. P. C. Saxena
Associate Professor,
School of Computer and
Systems Sciences,
J. N. U., New Delhi.

ACKNOWLEDGEMENTS

I take this opportunity to express my gratitude to Dr. P. C. Saxena for introducing me to this project work. It could be completed only due to his patience and effort in dealing with various technical problems.

I would like to thank Prof.N.P.Mukherjee,Dean,SC&SS, JNU for providing me the opportunity to undertake this project. I would also like to thank the authorities of our school for providing me the necessary facilities to complete my project.

This project would not have been completed without the help of many people. Of all these, I am particularly indebted to my colleague, Mr. Pramod Varma K., who helped me through all my day to day problems.

I would like to convey my thanks to all those people, who tested the software and passed their critical comments to improve it.

CONTENTS

1· INTRODUCTION

During the last few years, all activities connected with computers have experienced an enormous upswing. This is due, in particular, to the advances in the field of semiconductor electronics. The wide scale use of integrated circuit chips has revolutionised the way we work. With the circuit element becoming smaller and smaller, i.e. the transition to integrated circuits, the price of hardware has reduced to amazingly low level. This has definitely been an impulse to the expansion of computer technology in divergent areas. Some of the distinguishable areas are - Artificial Intelligence, Numerical Computation, Decision Making, Automated Process Control, Optimization, Games and Computer Graphics.

While two decades ago, Computer graphics was still considered to be a special application for computer, today it can be regarded as being completely integrated into computer science. It is an effective medium of communication between man and computer as it produces images whose appearances and motions make them quite unlike any other form of computer output. The dynamics of display which can be achieved on the screen is also of significance for the design. It is a necessary condition for some technical applications, for example, when simulating dynamic processes. The three areas in which computer graphics is used are :

1. Animation
2. Visualization for instruction purposes
3. Design

The area of animation has become known because of new methods of producing animated pictures. Computers are used now, to simulate technical objects, landscapes and movies. Computer graphics has proven to be an excellent tool in decreasing their cost of production.

The task of computer graphics in the field of instruction is the visualization of instructional material, as is possible in the field of Mathematics, Physics and other areas. The new method permits the use of pictures to have a better overview and to make the material easier to understand.

The transition from mechanical plotter to graphics terminals is a development characteristic of the last few years. They open the way to unlimited use of color, dynamic display, and interactive use. Traditionally, much of what drafters do on the job is laborious and time consuming. Because of this, techniques have been developed to cut down on the amount of time required to perform drafting tasks. The use of computer in design and drafting is one of the most significant engineering developments which have come about over the years to increase productivity.

1.1 MOTIVATION FOR INTERACTIVE COMPUTER GRAPHICS

One of the most renowned scholars in computer graphics area, Ivan Sutherland [13] wrote :

" *Whereas a microscope enables us to examine the structure of a subminiature world and a telescope reveals the structure of the universe at large, a computer display enables us to examine the structure of a man-made mathematical world simulated entirely within an electronic mechanism.*

Computer displays have become of major importance to two groups of people. One group has a pictorial problem in the day to day work, for which they *would like computer help. The other group using computer display is interested in gaining insight into complex natural and mathematical phenomena. These users simulate physical situations of various kinds in the computer and use display devices to present the results of simulation. All these people, interested in educating themselves or others, use computer display as one of the many tools for gaining deeper understanding of the problem.* "

Interactive computer graphics help to carry out tasks that would otherwise be prohibitively expansive to perform. For example, architects can explore alternative solutions to design problems at an interactive graphics terminal. In this way, they can test many more solutions than would be possible without the computer. Architectural drawings graphics made little progess because the computers of that period were incapable of interactive use. The graphics terminal had to be invented and manufactured before they could be used for interactive graphics. Only towards the end of the decade, with the development of machines like MIT's TX-0 and TX-1, did interactive computer graphics become feasible.

The beginning of research in the application of computers to architectural problems coincides roughly with the introduction of the second generation digital computers in early 60s. In 1962, Ivan Sutherland developed SKETCHPAD, which paved the way for interactive graphics. Commercial CAD was introduced in 1964, when IBM made its first graphics system available.

From 1965 to 1970, the fundamentals did not change. An expensive mainframe and an expensive vector refresh terminal were still the only equipments available for CAD use. In 1965, Control Data Corporation (CDC) supplied one of the first Computer aided design system.

The period, 1970-75, was marked by the use of two newly developed technologies - the mini computers and the storage graphics screen. Softwares that were not specific to one particular application were developed, and turnkey systems (made up of a minicomputer, a storage screen and basic software) were marketed. The first complete turnkey require lettering that is neat and attractive, line work that is clear, scalework that is accurate, and dimensioning that must total out correctly. **Computer Aided Drafting** allows architectural drafters to produce plans that meet all the requirements set forth above.

1.2 DEFINITION OF COMPUTER AIDED DRAFTING

The use of computer in the process of generation, representation and manipulation of graphic data is known as Computer Aided Drafting. Its very commonly used acronym is CAD.

CAD involves the use of computer as a tool in making, checking, correcting and revising drawings. It combines the best abilities of computers to form a powerful drafting and designing tool for drafters.

Computer aided drafting encompasses the interacive processing of graphical data for design purposes. The screen of the graphics terminal is used as an electronic drafting board. A CAD system makes a user much more productive by providing proper tools for image creation and manipulation.

1.3 A BRIEF HISTORY OF COMPUTER AIDED DRAFTING

Computers have been used in design engineering since early 1950s. During that period, interactive computer interactive graphics system was made available in 1970 by Applicon Incorporated.

The period, 1975-80, was not marked by any great technological breakthrough but the improved reliability of turnkey systems was an important factor to begin widespread use of CAD systems.

CAD systems are now being used in a wide range of engineering applications. Changing technologies and a new approach to CAD software are some of the reasons for this. In early 1980s new possibilities arose from the introduction of microcomputers and raster screens. The introduction of workstations and personal computers led to the development of new softwares to run on these computers. This period saw the use of CAD move out of sophisticated areas to those involved in everyday engineering applications.

1.4 COMPUTER AIDED DRAFTING SYSTEMS

Since the introduction of Computer aided drafting in engineering applications, a large number of CAD systems have been developed to meet the specific requirements. Some of the CAD systems are given below (in lexical order) -

1.4.1 ADAGE 4370 WORK STATION

It is a high performance, high resolution, full three dimensional graphics system for mainframe users. It is designed to attach directly to IBM mainframes. 4370 permits local display, modification and manipulation of complex, highly structured images. A network of high-speed microprocessors reduces execution time and speeds up user interaction with computer.

1.4.2 APPLE III

Apple III is a fully integrated system with a built-in 143k byte disk drive, 256k of main memory and built-in disk controller for handling upto four floppy disk drives. It has got improved multicolor capability and 16 shades of grey for vivid graphics presentation.

1.4.3 ADVANCED GRAPHICS WORKSTATION (AGW)

Auto-trol Technology Corporation announced the availability of AGW. It is the first turnkey CAD system that offers the speed, power and capacity of 32-bit processor at low cost. AGWs can be linked together, making it the first CAD system that can provide high-performance local area network of dedicated computers with distributed processing.

1.4.4 CADAM

The Computer Graphics Augmented Design and Manufacturing system, known as CADAM is a high function, general purpose design and drafting package containing analytical and conceptual design aids for 3-D drafting. The CADAM software may be divided into an interactive and a batch portion. The interactive portion allows the user to construct geometrical figures which may later be input to batch routines.

1.4.5 CAD2D

CAD2D, developed by ManTech International, is a software system for automated drafting and design. In order to support a wide variety of application areas, CAD2D is designed to interface to a variety of graphic display, input devices and graphic output equipments. CAD2D system allows the user to generate text in a variety of fonts and lines with any desired pattern or width.

1.4.6 CALCOMP

A product of California Computer Products, it supports multiple workstations. Out of two CRT monitors available at each workstation, one is alphanumeric and other is graphic. The alphanumeric monitor displays the information such as program status, lists of symbols available for drawing and so on. The graphics monitor utilizes raster technology and Calcomp picture processor to display the drawings.

1.4.7 CHIPS

CHIPS is an advanced, minicomputer-based system for VLSI design work. This turnkey system features 32-bit precision to support VLSI work, color graphics terminal and software aids to enhance design implementation.

1.4.8 CASCADE II

CASCADE II is a multistation Computer aided drafting system which provides general purpose 2-D facilities. The CASCADE II configuration is built on an Apple IIE computer system with 80k RAM. It can be used as a standalone system or can be connected through a network to a central fixed-disk that can support upto eight workstations. Any program written for Apple II plus can be used on CASCADE II.

1.4.9 DIMENSION III

This is an interactive computer graphics system where a designer works directly at a video display terminal to create 3-D designs. It is used in the fields of architectural, engineering and construction (AEC). The created design becomes a part of the database from which material report and other documentations can be obtained. It was developed by Calma Company.

1.4.10 HEWLETT PACKARD SYSTEM

HP Engineering Graphics system / 45 (EGS/45) offers general drawing, PC board layout and schematic drawing capabilities to help electronic circuit and printed-circuit board designers. EGS/45 is compatible with the HP 9845 family of desktop computer systems. The drawing area can be thought of as containing upto 256 overlapping transparent layers.

1.4.11 KAD II

KAD II was released by Kohinoor Rapidograph Inc. to directly use on the drawing board. It is a microprocessor-based system to produce a virtually limitless library of lettering styles, symbols and designs used in architectural drawings.

1.4.12 LEXIDATA MODEL 8400/D

8400/D is a high performance disk-based graphics workstation that supports multiple configurations of black and white or color graphics. The standalone product incorporates a powerful raster graphics subsystem and 32-bit microprocessor. An Extended Graphics Operating System (EGOS) is included in the display processor subsystem to handle drawing functions. Library of graphics subroutines that is available on the system can be used by C or FORTRAN application programs.

1.4.13 PERKIN - ELMER DISTRIBUTED SYSTEM

Users of Perkin-Elmer system have both the power and responsiveness of a local computer and access to centralized data base of drawing information. The basic system includes one to four workstations of interactive graphic terminals and an 80 MB disk for local drawing storage. Each workstation includes a supermini computer, high performance floating point processor and a graphic terminal.

1.4.14 PRIME MEDUSA SYSTEM

The system is modularly designed by Prime Computers. Features of the PRIME workstation include : Raster scan technology, high resolution 1280 x 1024 pixel monitor, video processor, video display monitor and alphanumeric terminal. Some of the intelligent workstation operations are - pan, zoom and selective erase.

1.4.15 PRODUCER DRAFTING SYSTEM

It is a turnkey system with three work stations where drafters can work concurrently. At the interactive station, drafters can see their drawings as they create them. At the plotter station, high-quality hardcopy can be taken in a variety of line weights and colors. At the electronic drawing station, the drafter can take existing drawings and quickly revise them. A library which contains thousands of commonly used symbols and figures is a standard feature of the system.

1.4.16 STICKS

STICKS is a dynamic, symbolic IC design package which allows designers to use symbols far less complex than actual circuit elements when laying out ICs. STICKS uses an automatic spacing system to layout the circuit in as small a space as design rules allow. This leaves the designer to concentrate on other designing aspects of chip design.

1.4.17 TEKTRONIX

The complete configuration of Tektronix system includes a desktop computer, dynamic graphics memory and 2-D drafting software. The Tektronix package can accomodate different drafting standards. Use of Graphics Model Exchange (GMX) file format allows users to exchange graphic data between the drafting package and other Tektronix softwares.

1.4.18 UNIGRAPHICS

UNIGRAPHICS is a standalone minicomputer based turnkey system developed by MCAUTO co.. The system is unique in its machine-independence and runs on several models of standard minicomputers. Several different images of the model can be displayed on the screen at one time. Different views can be displayed at different scales.

1.4.19 UNISCAD

Security and integrity of the design data base can be protected with the UNIVAC's UNISCAD system. It is a simple and easy-to-use system where access to drawings can be restricted on the basis of passwords. The system is composed machine independent graphics environment. TIGS allows the user to have a device-independent graphics interface and terminals to suit his requirements. Both TIGS and UNIPLOT were released by Conrtol Data Corporation (CDC).

1.5 ADVANTAGES OF COMPUTER AIDED DRAFTING

To make changes on the traditional drafting board may cause frustration for several reasons. To help overcome this frustration, the CAD system relieves the drafter and designer from tedious manual drafting. The various advantages of using a computer in drafting are as follows -

1. The time for product design and subsequent engineering changes has been redued.
2. Draftings can be revised and changed much more quickly and accurately than by hand.
3. The facility of modification without delay facilitates a much freer work style which includes phases of experimenting to a much larger extent than before.
4. Compared with manually prepared drawings, computer produced drawings are superior in quality of lettering, scaling accuracy and overall appearance.

5. A drawing that has been prepared and stored can, at any time, be recalled and readily modified. The modified drawing might be utilized for a completely different application.
6. More design cycles can be carried out within the limitations of time duration.

1.6 COMMAND LANGUAGE

Command language is a set of rules by which a user and the computer carry out their conversation. The command language of each graphic system determines what type of input data is accepted and how these input data are utilized. One desirable characteristic of command language is to minimize the number of actions required by a user to achieve the desired result. The description of all the commands is given in the source listing of 'JNU_HELP.PAS' module which is attached in Appendix of this report.

This chapter explains the data structures used to maintain the information provided by the user. While choosing the data structures, it was kept in mind to reduce the memory requirement, easy handling of data and future development of the package. All the facilities provided to create a layout are introduced in this chapter. Detailed algorithms to achieve various tasks are also given in this chapter.

2.1 FEATURES OF THE SYSTEM

This subsection covers the different facilities provided to draw layouts. These facilities are categorized as :

1. Position Generators
2. Geometry Generators
3. Geometry Modifiers
4. Display functions
5. Disk functions

The position generators are those functions which help the user to move to desired location for drawing. The geometry generators enable a graphic shape to be drawn on the monitor. Some of the geometry generators are line, circle and arc. Geometry modifiers are used to facilitate modifications in the drawing. These include - erase, move, rotate etc.

Display functions help to view minute details of different parts of the drawing more clearly.

The proper combination and execution of each generator and modifier listed above enables an individual to prepare an engineering drawing.

2.1.1 POSITION GENERATORS

Cursor :

Cursor is required to indicate location on the screen. The X, Y co-ordinates of the current cursor position will be displayed on the top right hand corner of the screen. There are 3 different types of cursor :

(1) Dot cursor :

There will be only a dot for the cursor.

(2) Small cursor :

The small cursor is an enlarged plus sign.

(3) Big cursor :

It is in the form of a crosswire, with a vertical line which stretches from top to bottom and one horizontal line which stretches from left to right of the drawing screen.

The size of the cursor can be changed by pressing the plus (+) sign on the keypad . The plus (+) sign acts as a toggle and changes the size of the cursor. By default, the size of the cursor is an enlarged plus sign.

2. PROJECT DESCRIPTION

Movement of the Cursor :

There will be a warning beep if one attempts to move the cursor beyond the screen boundary. The speed of the cursor can be increased by pressing the <PgUp> key. Similarly, the speed of the cursor can be decreased by pressing the <PgDn> key.

The cursor has variable speeds.

Minimum speed	: Movement by lowest units.(i.e. 1 mm).
Maximum speed	: Depends on the dimensions of layout under view and can move from one end of the screen to another.

2.1.2 GEOMETRY GENERATORS

Dot :

This is the basic drawing entity. Move the cursor to the desired position to mark a dot.

Line :

A line may be drawn if two end points of the line are specified. As the cursor is moved around, a dummy elastic line appears on the screen between the first point and the current cursor position.

Line styles :

The system provides five predefined line styles to create drawing entities. All the possible line styles are shown in the top right hand corner of the screen. The cursor blinks at the present line style. The user can selects a new line style by moving the cursor up and down. Once a particular line style is selected, all the subsequent drawing entities are drawn in the same line style.

Rectangle :

A rectangle is drawn when the position of two of its diagonal corners are marked by the user. As the cursor is moved around to draw the rectangle, a dummy elastic rectangle appears on the screen.

Circle :

To draw a circle, the centre of the circle and its radius are specified. Moving the cursor increases or decreases the radius of the circle.

Arc :

When an arc is required, information regarding where the arc starts and ends and the centre of curvature is required. An arc will be drawn from the first point to the end point in anti-clockwise direction.

Text :

Text is defined as a string of alphanumeric characters that is entered for the purpose of being shown on the screen and on the plot of drawing. This distinguishes it from other alphanumeric inputs entered for name of commands or numeric values for angle.

To place text anywhere in the layout, the user has to specify the string and the SCALE of the text. A temporary rectangle will appear on the screen which indicates the size the text will occupy. The size can be changed by pressing a suitable key. The user can move this rectangle in the layout using the direction keys.

2.1.3 GEOMETRY MODIFIERS

Changes and corrections are always made in any design process. A design system must have ways of modifing the layout. The window clipping method is used to find the drawing entities that are to be manipulated. The finder routine can find all the elements that fall within the rectangle and makes a list of them for manipulation process. Only those elements, which completely fall within the window are selected. The system then ask the user to select the elements which he wishes to modify.

Move (Shift) :

The user can move the drawing entity from one screen loction to another using this facility. Size, shape and orientation of the symbol do not change. The user has the option to shift:

1. The entire display
2. A single entity (a line, an arc etc.)
3. A group of entities

The selected entities are moved, depending on the distance between reference point and the displacement point, and the relative angle made by the line joining them.

Rotation :

The change in location is measured by the angle between a line from centre of rotation to the original point and the line from the centre to the transformed point. Each point on the object moves in a circular path around the centre of rotation. The user specifies the entity to be rotated, centre of rotation and the angle by which the entity has to be rotated in anticlockwise direction.

Erase :

A temporary rectangle is drawn to encompass the entities to be erased. The user has the option to unselect any of the entities present in the temporary rectangle.

Break (Cut) a line :

A part of any length of a line, which is already present on the drawing, can be erased by specifing the two end points of the part which is to be erased. Each part is then treated as an independent line' for subsequent operations.

Before a line can be broken, it has to be identified. The window clipping method is used to find out all the lines that fall completely within the rectangle. The system then asks the user to identify a particular line to be broken into pieces.

2.1.4 DISPLAY FUNCTIONS

Pan :

Pan changes the window to be viewed at the same viewport. It is a convenient approach to view different parts of a drawing that is too large to be viewed on the screen at one time. The active window can be dragged in any direction to display the other parts of the layout. It does not change the scale of the drawing viewed on the screen. All the commands can be used on to the new displayed area of the layout.

Distance between points :

To find the distance between two points, move the cursor to the first point to select it. Then move the cursor to the second point. The distance between the points is displayed at the top of the screen.

Zoom - Actual / Window :

The ability to zoom in or out from a given part of the drawing area partially overcomes the physical limitations of the screen size. Zoom will change the size of display, but it will not change the scale of the drawing. Changing the window size of a portion of display requires the two diagonal points of the window. This will make a dummy box around the portion to be magnified.

It is desired to enlarge the display so that work can be accomplished more accurately. New drawing entities can be drawn in the magnified window. All the changes made in the window reflects on the overall drawing. If the magnification is not enough, the user can zoom in on a zoom.

2.1.5 DISK FUNCTIONS

Save :

The save option allows the user to save, whatever he has drawn, on a permanent storage which can be used for future purposes. This file is known as design file, which is a list of graphic elements and intelligent information that define the design. Whenever one adds or deletes an element from the design, the element is added to or removed from this file.

Hard°copy :

The image on the display screen can be preserved as a permanent paper copy through the use of a printer. The term 'Hard Copy' emphasizes the distinction between the physical presence of the image on paper and the 'soft' temporary visual image on the screen. The output is produced much more quickly and less expensively.

2 DATA STRUCTURES

2.1 DATA STRUCTURE FOR INTERACTION

As a user is allowed to draw different types of ·awing entitites and to perform modifications on them, it was ·lt that the best way to handle this situation was to store ıe information in linked lists. Initially it was found ›propriate to store lines in one linked list, circles in ıother and so on. But later on all these linked lists are ·rged in one linked list of variable node size. In each node, character is stored which tells whether the node is for a ne, or for a dot or for something else. At the same time, it so tells the number of bytes in that node. A combined linked st is made to use the memory optimally and to avoid blockage memory in developing linked lists.

Follwing paragraphs give the information regarding nodes of the linked list -

1. Node for **world** :

The world is the area specified by the user. The fields in this node are -

'W' - defines that the node is for world.

W status - always one.

X1, Y1 - (x,y) co-ordinates of the lower left corner of the world.

X2, Y2 - (x,y) co-ordinates of the upper right corner of the world.

Style - specifies the line style in which the rectangle corresponding to the area is to be drawn. Always 1.

NxtPtr - Pointer to the next node in linked list.

2. Node for **Dot** :

The fields in the node are -

'D' - Defines that this node is for a Dot.

D Status - determines the status of the Dot. This bit is 1 (one) if the dot is present in the layout and if it is erased, the bit becomes 0 (zero). When the information is stored in the file, it will be stored only for those dots which have their D_Status equal to 1 (one).

X, Y - (x,y) co-ordinates of the point.

NxtPtr - Pointer to the next node in linked list.

3. Node for **Line** :

The fields in the node are -

'L' - Defines that this node is for a line.

L_Status - determines the status of the line. This bit is 1 (one) if the line is present in the layout and if it is erased, the bit becomes 0 (zero). When the information is stored in the file, it will be stored only for those lines which have their L Status equal to 1 (one).

X1, Y1 - (x,y) co-ordinates of the end point of the line which is close to the origin.

X2, Y2 - (x,y) co-ordinates of the other end point of the line.

Style - specifies the line style in which the line is to be drawn.

NxtPtr - Pointer to the next node in linked list.

4. Node for **Circle** :

The fields in this node are -

'C' - defines that the node is for a circle.

C_Status - determines the status of the circle. This bit is 1 (one) if the circle is present in the layout and if it is erased, the bit becomes 0

(zero). When the information is stored in the file, it will be stored only for those circles which have their C_Status equal to 1 (one).

Xc, Yc - (x,y) co-ordinates of the centre point of the circle.

R - Radius of the circle. (in world co-ordinates)

Style - specifies the line style in which the circle is to be drawn.

NxtPtr - Pointer to the next node in linked list.

5. Node for **Arc** :

The fields in the node are -

'A' - defines the node for an arc.

A_Status - determines the status of the arc. This bit is 1 (one) if the arc is present in the layout and if it is erased, the bit becomes 0 (zero). When the information is stored in the file, it will be stored only for those arcs which have their A_Status equal to 1 (one).

Xc, Yc - (x,y) co-ordinates of the centre point of the circle from which arc is to be cut.

X1, Y1 - (x,y) co-ordinates of the first point on the arc.

Theta - angle (in degrees) which gives the displacement from (X1,Y1) point in anticlockwise direction.

Style - specifies the line style in which the arc is to be drawn.

NxtPtr - Pointer to the next node in linked list.

6. Node for Text :

The fields in the node are -

'T' - Defines the node for a text string.

T_Status - determines the status of the text. This bit is 1 (one) if the text is present in the layout and if it is erased, the bit becomes 0 (zero). When the information is stored in the file, it will be stored only for those text string which have their T_Status equal to 1 (one).

X, Y - (x,y) co-ordinates of the point from where the first charcter of the text starts. It is the (x,y) co-ordinates of the lower left corner of the rectangle which encloses the charcter.

Size - Specifies the character size of the text.

Style - specifies the line style in which the text is to be written.

Length - Gives the length of the string i.e. number of the charcters in the string.

StrPtr - Pointer to the string which is to be drawn.

NxtPtr - Pointer to the next node in linked list.

2.2.2 DATA STRUCTURE FOR THE FILE - DATA.FIL

When the user requests to save the layout, all the intelligent information about the different entities present in the layout, are stored in this file. When the Layout Editor is executed next time and this file is present in the project directory, it gets loaded in the main linked list. As this file stores the information about different entities, it is not a file of records. The information is stored in sequence in chunks of bytes. The first byte of the chunk specify the number of bytes in that chunk.

Following paragraphs mention the fields for different entities, which are stored in the file :

1. Fields for **world** :

'W' - defines that the next few bytes are for layout area.

W_status - always one.

X1, Y1 - (x,y) co-ordinates of the lower left corner of the world.

X2, Y2 - (x,y) co-ordinates of the upper right corner of the world.

Style - specifies the line style in which the rectangle corresponding to the area is to be drawn. Always 1.

2. Fields for Dot :

'D' - Defines that the next few bytes are for a Dot.

D Status - Always 1 to indicate that the dot is present in the layout.

X, Y - (x,y) co-ordinates of the point.

3. Field for Line :

'L' - Defines that the next few bytes are for a line.

L Status - Always 1 to indicate that the line is present in the layout.

X1, Y1 - (x,y) co-ordinates of the end point of the line which is close to the origin.

X2, Y2 - (x,y) co-ordinates of the other end point of the line.

Style - specifies the line style in which the line is to be drawn.

4. Fields for Circle :

'C' - defines that the next few bytes are for a circle.

C Status - Always 1 to indicate that the circle is present in the layout.

Xc, Yc - (x,y) co-ordinates of the centre point of the circle.

R - Radius of the circle. (in world co-ordinates)

Style - specifies the line style in which the circle is to be drawn.

5. Fields for Arc :

'A' - defines the the next few bytes are for an arc.

A_Status - Always 1 to indicate that the arc is present in the layout.

Xc, Yc - (x,y) co-ordinates of the centre point of the circle from which arc is to be cut.

X1, Y1 - (x,y) co-ordinates of the first point on the arc.

Theta - angle (in degrees) which gives the displacement from (X1,Y1) point in anticlockwise direction.

Style - specifies the line style in which the arc is to be drawn.

6. Fields for Text :

'T' - Defines the next few bytes are for a text string.

T_Status - Always 1 to indicate that the text string is present in the layout.

X, Y - (x,y) co-ordinates of the point from where the first charcter of the text starts. It is the (x,y) co-ordinates of the lower left corner of the rectangle which encloses the charcter.

Size - Specifies the character size of the text.

Style - specifies the line style in which the text is to be written.

Length - Gives the length of the string i.e. number of the charcters in the string.

Actual text string is also stored in the file immediately after the above mentioned information.

2.3 ALGORITHMS

This section includes algorithms for all the facilities available in the system.

2.3.1 ALGORITHMS FOR POSITION GENERATORS

CURSOR POSITION

Step 1. : Store the x and y co-ordinate values of current cursor position in local variables.

Step 2. : Find out the step size by which cursor position can be changed.

Step 3. : Depending upon the value of arrow key pressed, modify the cursor co-ordinate values.

Step 4. : IF the modified cursor position goes out of the screen, give a 'beep' sound and restore the previous cursor position.
ELSE modify the cursor position by drawing it to new position and erase it from the old position.

Step 5. : Display the co-ordinates of cursor position at the right hand corner of the screen.

CURSOR SPEED

Step 1. : Find out the present cursor speed.

Step 2. : (a) If 'PgUp' key is pressed, increase the cursor speed by a factor of 10.

(b) If 'PgDn' key is pressed, reduce the cursor speed by a factor of 10.

Step 3. : (a) If the modified cursor speed is more than the maximum possible speed on the screen, reduce it to maximum possible speed.

(b) If the modified cursor speed is less than the minimum possible speed on the screen, increase it to minimum possible speed.

2.3.2 ALGORITHMS FOR GEOMETRY GENERATORS

DOT

Step 1. : Select RAM screen as active screen to draw.

Step 2. : Display a dot at current cursor position.

Step 3. : Copy the contents of RAM screen on to inactive screen.

Step 4. : Make an entry for this dot in the linked list.

LINE

Step 1. : Store the present line style in a local variable and set a new line style to draw elastic lines.

Step 2. : Take present cursor position as the first end point on the line.

Step 3. : As the cursor is moved using arrow keys, draw a dummy line from the cursor position to the first end point.

Step 4. : Repeat step 3 until the user presses <Enter> or <Esc> key.

Step 5. : (a) If <Esc> key is pressed, abandon the line command and quit the routine.

(b) If <Enter> key is pressed, take current cursor position as the second end point.

Step 6. : Select RAM screen as active screen to draw.

Step 7. : Restore the line style from local variable and draw a line between two end points.

Step 8. : Copy the contents of RAM screen on to inactive screen.

Step 9. : Make an entry for this line in the linked list.

RECTANGLE

Step 1. : Store the present line style in a local variable and set a new line style to draw elastic rectangle.

Step 2. : Take present cursor position as the first end point of a diagonal.

Step 3. : As the cursor is moved using arrow keys, draw a dummy rectangle using current cursor position and the first point as the end points of a diagonal.

Step 4. : Repeat step 3 until the user presses <Enter> or <Esc> key.

Step 5. : (a) If <Esc> key is pressed, abandon the rectangle command and quit the routine.

(b) If <Enter> key is pressed, take current cursor position as second end point of a diagonal.

Step 6. : Select RAM screen as active screen to draw.

Step 7. : Restore the line style from local variable and draw a rectangle using these two end points of diagonal.

Step 8. : Copy the contents of RAM screen on to inactive screen.

Step 9. : Make an entry for this rectangle in the linked list.

CIRCLE

Step 1. : Store the present line style in a local variable and set a new line style to draw elastic circle.

Step 2. : Take present cursor position as the center point.

Step 3. : As the cursor is moved using arrow keys, draw a dummy circle using the distance between current cursor position and the centre point as the radius.

Step 4. : Repeat step 3 until the user presses <Enter> or <Esc> key.

Step 5. : (a) If <Esc> key is pressed, abandon the circle command and quit the routine.

(b) If <Enter> key is pressed, take current cursor position as a point on the circle.

Step 6. : Select RAM screen as active screen to draw.

Step 7. : Restore the line style from local variable and draw a circle.

Step 8. : Copy the contents of RAM screen on to inactive screen.

Step 9. : Make an entry for this circle in the linked list.

ARC

Step 1. : Store the present line style in a local variable and set a new line style to draw elastic circle.

Step 2. : Take present cursor position as the center of curvature of the arc.

Step 3. : As the cursor is moved using arrow keys, draw a dummy circle using the distance between current cursor position and the centre point as the radius of curvature.

Step 4. : Repeat step 3 until the user presses <Enter> or <Esc> key.

Step 5. : (a) If <Esc> key is pressed, abandon the arc command and quit the routine.

(b) If <Enter> key is pressed, take current cursor position as the first end point of the arc.

Step 6. : Ask the user to mark the second point of the arc, which should be either on the circle or as close as possible.

Step 7. : Select RAM screen as active screen to draw.

Step 8. : Restore the line style from local variable and draw an arc from first point to second point in anticlockwise direction.

Step 9. : Copy the contents of RAM screen on to inactive screen.

Step 10. : Make an entry for this arc in the linked list.

TEXT

Step 1. : Store the present line style in a local variable and set a new line style to draw elastic box.

Step 2. : Ask the user to enter text string.

Step 3. : If <Esc> key is pressed, abandon the text command and quit the routine.

Step 4. : Display a dummy box to indicate the area which is going to be occupied by the text.

Step 5. : If the user wants to change the character size of text, allow him to change it and goto step 4.

Step 6. : Allow the user to use arrow keys to change the location of text.

Step 7. : If <Esc> key is pressed, abandon the text command and quit the routine.

Step 8. : Select RAM screen as active screen to draw.

Step 9. : write the text in present character size.

Step 10. : Copy the contents of RAM screen on to inactive screen.

Step 11. : Make an entry for this text in the linked list.

2.3.3 ALGORITHMS FOR GEOMETRY MODIFIERS

An element (drawing entity) must be located first, before it can be modified. Window clipping method is used to find out the elements falling completely within the window. Since all such elements can be found out together, it is possible to perform the geometrical modifications on more than one entity at the same time.

ALGORITHM FOR WINDOW CLIPPING PROCEDURE

Step 1. : Store the present line style in a local variable and set a new line style to draw elastic box.

Step 2. : Ask the user to enter one of the diagonal points of the window.

Step 3. : If <Esc> key is pressed, quit the routine.

Step 4. : Take present cursor position as the first diagonal point.

Step 5. : As the cursor is moved using arrow keys, draw a dummy rectangle using current cursor position and the first point as the end points of a diagonal.

Step 6. : Repeat step 5 until the user presses <Enter> or <Esc> key.

Step 7. : If <Esc> key is pressed, abandon the command and quit the routine.

Step 8. : Find out all the elements which lie completely within the window.

Step 9. : Make a linked list of all such elements and make the selection bit of them equal to 1.

Step 10. : Ask the user to unselect those entities which

Step 11. : Pass this linked list for modification process.

MOVE

Step 1. : Use window clipping algorithm to find out the entities which user wants to move.

Step 2. : Ask the user to enter the reference point.

Step 3. : Ask the user to enter the displacement point.

Step 4. : If <Esc> key is pressed, abandon the command and quit the routine.

Step 5. : Depending upon the distance and angle between reference point and displacement point calculate the new co-ordinates for all the entities selected for moving.

Step 6. : Make all these changes in the main linked list.

Step 7. : Clear the screen and draw the entities of the modified linked list again.

ERASE

Step 1. : Use window clipping algorithm to find out the entities which user wants to erase.

Step 2. : Make the status bit equal to zero of all those entities which are selected for erasing. Zero implies that they are no longer in the layout.

Step 3. : Clear the screen and draw the entities of the modified linked list again.

ROTATION

Step 1. : Use window clipping algorithm to find out the entities which user wants to rotate.

Step 2. : Ask the user to enter the reference point about which he wants to rotate the entities.

Step 3. : If <Esc> key is pressed, abandon the command and quit the routine.

Step 4. : Ask the user to enter the angle for rotation in degrees.

Step 5. : Depending upon the reference point and the angle, calculate the new co-ordinate values for all the entities selected for rotation.

Step 6. : Make all these changes in the main linked list.

Step 7. : Clear the screen and draw the entities of the modified linked list.

BREAK A LINE

Step 1. : Use window clipping algorithm to find out the line which user wants to break.

Step 2. : Ask him to enter the first and second cut points on the line (or as close to the line as possible).

Step 3. : Take the projection of these two points on the line and calculate the co-ordinates of the foot of the perpendiculars.

Step 4. : If any of the projection does not fall on the line, abandon the command and quit the routine.

Step 5. : IF one of the projection falls on a end point of the line, modify the end co-ordinate of existing line.

OTHERWISE break the line into two independent lines.

Step 6. : Make an entry in linked list for the newly created line.

Step 7. : Clear the screen and draw the entities of the modified linked list.

2.3.4 ALGORITHMS FOR DISPLAY FUNCTIONS

ZOOM IN

Step 1. : Store the present line style in a local variable and set a new line style to draw elastic rectangle.

Step 2. : Ask the user to enter the first diagonal point.

Step 3. : As the cursor is moved using arrow keys, draw a dummy rectangle using current cursor position and the first point as the end points of a diagonal.

Step 4. : Repeat step 3 until the user presses <Enter> or <Esc> key.

: If <Esc> key is pressed, abandon the zoom command and quit the routine.

: Calculate the modified world co-ordinates.

: Clear the screen and change the world co-ordinates on active screen.

: Restore the line style and draw the entities of the linked list.

: Change the active world co-ordinates to the area of layout.

: Clear the screen and change the co-ordinates for active screen.

: Draw the entities present in layout once again.

BETWEEN POINTS

Store the present line style in a local variable and set a new line style to draw elastic lines.

Take the present cursor position as the reference point relative to which the distance is to be calculated.

As the cursor is moved using arrow keys, draw a dummy line using current cursor position and the reference point as the end points of the line.

Step 4. : Calculate the distance between two end points of the dummy line.

Step 5. : Display the distance on top of the screen.

Step 6. : Goto step 3 until <Enter> key is pressed to terminate the command.

PAN

Step 1. : Ask the user to enter the reference point.

Step 2. : Allow him to use arrow keys to indicate displacement.

Step 3. : As the cursor is moved on screen, draw a dummy line using current cursor position and the first point as the end points of the line.

Step 4. : Goto step 2 until <Enter> key or <Esc> key is pressed.

Step 5. : If <Esc> key is pressed, abandon the command and quit the routine.

Else using the relative distance and angle between points calculate the modified world co-ordinates.

Step 6. : Clear the screen and change the co-ordinates for active screen.

Step 7. : Draw the entities present in layout once again.

2.3.5 ALGORITHMS FOR DISK FUNCTIONS

SAVE

Step 1. : Ask the user to confirm the choice of saving the modified layout.

Step 2. : If user wants to save the layout goto step 3. Else quit the routine.

Step 3. : Scan the nodes of the linked list.

Step 4. : If the status bit of an entity is found to be one, save it in the 'data.fil' file in the format already outlined in section 2.2.
Status bit is equal to zero indicates that the entity is no longer in the layout, hence do not save it.

Step 5. : Goto step 3 until the end of the linked list is reached.

HARD COPY

Step 1. : Ask the user to enter the print scale factor.

Step 2. : Check if it is in the valid range [1..1000] (arbitrarily selected range)

Step 3. : As the complete layout cannot be printed on printer in the specified scale, divide the layout in grid.

Step 4. : Find the number of cells on grid.

Step 5. : Clear the screen and draw a cell on t
in given print scale.

Step 6. : dump the screen image on the printer.

Step 7. : Repeat step 5 and step 6 for all the c

3. MODULAR DESCRIPTION

This chapter includes a brief description about various modules of the project. The function of each subroutine is explained with the help of input / output parameters and control flow. Depending upon the role of each subroutine, the whole software is divided into five modules.

3.1 THE MAIN MODULE

This is the controlling program which starts the software. It includes all the files which have relevant routines. This module asks the user to enter the project name and creates a subdirectory using it. All the files related to this project, are created in this subdirectory. If the layout for a given project is already created, this module passes the control to relevant routines to draw the layout on the screen.

3.1.1 Function Make_new_dir : boolean

Parameters : None

Function : This routine creates a new subdirectory by the name stored in 'Dir' variable.

Glb References : None

Local routines : None

Returns : True : If the subdirectory can be created.
False : otherwise.

3.1.2 Function Create_new_dir : boolean

Parameters : None

Function : It offers the user the choice to start a new project. If the user wishes to do so, it passes the control to create a new subdirectory.

Glb References : None

Local routines : None

Returns : True : if the user wants to start a project.
False : otherwise.

3.1.3 procedure General_data

Parameters : None

Function : It asks the user to enter the area. It also writes it in the file 'Ps_wrld.Dat'.

Glb References : None

Local routines : None

Returns : None

3.2 THE GLOBAL MODULE

This file contains the type declarations required to store the information for all the drawing entites. It also includes the global variable declarations for the system. The description for all the subroutines is given in the next few paragraphs.

3.2.1 Procedure ClearInkeyBuffer

Parameters : None

Function : It clears the keyboard input buffer by reading the charcters.

Glb References : None

Local routines : None

Returns : None

3.2.2 Function fileExist(filename : string80) : boolean

Parameters : fileName : name of the file to be checked.

Function : Checks if the file with the given name exists on the drive.

Note : If the file is present on other Drive/ Dir, then the full path should be passed in filename.

Glb References : None

Local routines : None

Returns : True : if file is present.

False : if file is not present.

3.2.3 Procedure Beep

Parameters : None

Function : This is sound routine for the editor, where a combination of different sound frequencies are used.

Glb References : None

Local routines : None

Returns : None

3.2.4 Procedure Short_Beep

Parameters : None

Function : One sound frequency is used for a short beep sound.

Glb References : None

Local routines : None

Returns : None

3.2.5 Function Angle_rtn(X_Cent, Y_Cent, X, Y : real): real

Parameters : X_Cent, Y_Cent : (x,y) co-ordinates of the point about which angle is to be measured.

X, Y : (x,y) co-ordinates of the point for which angle is to be measured.

Note : All values are in world co-ordinate system.

Function : It calculates the angle for a point (X,Y) taking another point (X_Cent, Y_Cent) as origin.

Glb References : None

Local routines : None

Returns : returns angle value in degrees.

3.2.6 Procedure Swap_screen_rtn

Parameters : None

Function : Copies the contents of RAM screen on to the displayed screen.

Control Flow : Selects the RAM screen as active screen, then uses 'CopyScreen' to copy its contents onto inactive screen (Screen 1). It then selects screen 1 as active screen.

Glb References : None

Local routines : None

Returns : None

3.2.7 Procedure Draw_txt_rtn(Loc_TxtStr : string60; var X, Y : real; size : real)

Parameters : Loc_TxtStr : Text, which is to be written.

X,Y : (x,y) co-ordinates of the start point.

Size : character size of Text.

Function : Draws the alphanumeric text string.

Control Flow : It draws the string character by character. For each character,it reads the corresponding string from 'CHAR.DAT' file present in the working directory and draws the charcter.

Glb References : None

Local routines :

procedure DrawChar_rtn(Loc_Chstr : string60)

parameters : Loc_ChStr : an equivalent string for the character to be drawn

function : Draws the character. The various combinations in the string are -

Un : Draw a line upward (n * size) long

Dn : Draw a line downward (n * size) long

Ln : Draw a line leftward (n * size) long

Rn : Draw a line rightward (n * size) long

En : Draw a line diagonally up - rightward (n * size) long

Fn : Draw a line diagonally down-rightward (n * size) long

Gn : Draw a line diagonally down- leftward (n * size) long

Hn : Draw a line diagonally up - leftward (n * size) long

returns : none

Returns : None

3.2.8 Procedure Draw_List(flag : boolean)

Parameters : flag : if true, then Line Style is set before an entity is drawn.

Function : Draws the elements present in the linked list

Control Flow : Draws only those elements(entities) which have their status as 1. It starts drawing the elements from the top of the linked list and stops when end of linked list is reached.

Glb References : Draw_txt_rtn

Local routines : None

Returns : None

3.2.9 Procedure save_list

Parameters : None

Function : Saves the entities present in the layout in the file 'DATA.FIL'. This file is created in the subdirectory which has the name same as the project name. The file gets overwritten everytime 'save' function is selected in the editor module.

Control Flow : This module starts saving the entities from the top of the linked list. If an entity has its status field value equal to 1, it is saved in the file.

Glb References : None

Local routines : None

Returns : None

3.2.10 Procedure GetMem_Rtn(Var TempPtr : PtrtoString; MemReq : word)

Parameters : TempPtr : Pointer, which will point to the continuous free memory block (MemReq bytes long) available in the Heap.

MemReq : Memory required in bytes.

Function : Allocates continuous memory block in the heap to the pointer

Control Flow : If MemReq bytes are available in the heap, the routine assigns it to the pointer else it saves the linked list in the 'Data.Fil' and halts the program with a warning message.

Glb References : Save_List , beep

Local routines : None

Returns : A pointer, pointing to the free memory block.

3.2.11 Procedure DWReal(No : Integer; x1, y2, x2, y1 : real)

Parameters : No: Index of selected world [1..MaxWorldsGlb]

x1,y1 : (x,y) co-ordinates of lower left vertex

x2,y2 : (x,y) co-ordinates of upper right vertex

Function : Defines a world co-ordinate system. Vertices are determined taking aspect ratio of the screen into account. (i.e. if a square is drawn in the layout it should look like a square on the screen too.)

Glb References : None

Local routines : None

Returns : None

3.2.12 Procedure init_draw_data(flag : boolean)

Parameters : flag : if true then the Global world co-ordinates are read from the file. If it is false, they are ignored.

Function : Reads the file 'DATA.FIL' present in the project directory.

Control Flow : This routine reads 'DATA.FIL' and makes the main linked list of all the elements present in the file. Once this file is created, it will be loaded in the link list whenever the package is used for same project.

Glb References : GetMem_rtn , Draw_List

Local routines : None

Returns : None

3.2.13 Procedure init_world

Parameters : None

Function : This routine reads 'Ps_wrld.FIL' only if 'Data.fil' is not present in the project directory. It reads the world co-ordinates from the file and initializes the vertices of the world indexed as 1. It draws the layout only after reading the file and storing the elements in the main linked list.

Glb References : GetMem_rtn , Draw_list

Local routines : None

Returns : None

3.2.14 Procedure init_cursor

Parameters : None

Function : This routine initializes the values for all the pointers. It gets the size of each record type and stores the values in global variables. It initializes the line style of elastic box which appears in some of the commands like zoom_in, rectangle, turn, move, erase. It also initializes the cursor size and cursor step.

Glb References : None

Local routines : None

Returns : None

3.2.15 Procedure Draw_csr_rtn(Csr_X,Csr_y : real; Csr_Size : Byte)

Parameters : Csr_x, Csr_x : (x, y) co-ordinates where cursor is to be drawn. co-ordinates are in world co-ordinate system.

Csr_Size : size of the cursor [0..3]

Function : Draws the cursor at (Csr_x, Csr_y) position on displayed screen

Control Flow : It first stores the present line style in temporary variable. It then copies the contents of RAM screen on the displayed screen. After this, the cursor is drawn in the passed cursor size.

Glb References : Swap_screen_rtn

Local routines : None

Returns : None

3.2.16 Function Quit_rtn: boolean

Parameters : None

Function : Asks the user to verify his choice of quitting.

Glb References : None

Local routines : None

Returns : True : if user wants to quit
otherwise false.

3.2.17 Procedure Cursor_pos_rtn(var Csr_X, Csr_Y : real; ch_pos : char)

Parameters : Csr_X, Csr_Y : (x,y) co-ordinates of current cursor position.

ch_pos : character representing the direction in which cursor is to be moved

Function : Changes the (x,y) co-ordinates of cursor according to the direction passed. If the modified cursor co-ordinates are out of active window co-ordinates, it does not change the co-ordinate values of the cursor and gives a 'beep' sound.

Glb References : Beep

Local routines : None

3.2.18 Procedure pg_rtn(ch_pg : char)

Parameters : ch_pg : character passed to change the cursor speed.

Function : Increases cursor step by a factor of 10 if ch_pg is 'U'. Decrease cursor step by a factor of 10 if ch_pg is 'D'. It does not increase / decrease the cursor step once the upper limit / lower limit is reached.

Glb References : None

Local routines : None

Returns : None

3.2.19 Procedure Disp_co-ordinate(Csr_X, Csr_Y : real)

Parameters : Csr_X, Csr_Y : (x, y) co-ordinate values to be displayed.

Function : writes the values at position (1 , 64) on both RAM screen and displayed screen.

Glb References : None

Local routines : None

Returns : None

3.2.20 Procedure Disp_relative(Csr_X, Csr_Y : real)

Parameters : Csr_X, Csr_Y : (x, y) values to be displayed.

Function : Writes the values at position (1, 38) on both RAM screen and displayed screen.

Glb References : None

Local routines : None

Returns : None

3.2.21 Procedure Clear_comm

Parameters : None

Function : Clears the first line from both RAM screen and displayed screen.

Glb References : None

Local routines : None

Returns : None

3.2.22 Procedure comm_line(x, y : integer; Str : string60)

Parameters : x, y : (x, y) co-ordinates on the screen where string is to be written.

Str : String which is to be written.

Function : Writes the string at (x, y) position on both RAM screen and displayed screen.

Glb References : None

Local routines : None

Returns : None

3.2.23 Procedure plus_rtn

Parameters : None

Function : Increases the cursor size by 1 from the present size. Once the maximum size is reached and this routine is called again, cursor size becomes minimum.

Glb References : None

Local routines : None

Returns : Changed value of cursor size.

3.2.24 Function Choice_rtn(var X, Y : real; Var Ch_choice : char): boolean

Parameters : X, Y : (x, y) co-ordinates of current cursor position in world co-ordinates.

Ch_choice : Character corresponding to the key pressed.

Function : If the pressed key is one of the arrow keys, move the cursor in that direction provided the cursor co-ordinates do not fall outside the active world co-ordinates.

Glb References : Cursor_pos_rtn , Pg_rtn
Plus_rtn

Local routines : None

Returns : True : if valid key is pressed
False : otherwise.

3.3 THE DRAW MODULE

This module contains all the routines pertaining to different drawing entities like arc, text, circle, rectangle etc.. It also includes routines to allow the user to zoom in on a smaller area of the drawing. It includes all the controlling procedures for geometry modifiers. When the user wants to rotate or move the drawing entities, the new co-ordinate values are calculated using the routines which are given below.

3.3.1 Procedure Dot_rtn

Parameters : None

Function : Displays a dot at present cursor position.

Control Flow : It copies the contents of RAM screen on to the displayed screen and then displays a dot. After this the contents of displayed screen are copied onto the RAM screen. It then creates an element for this dot in the main linked list.

Glb References : Swap_screen_rtn, GetMem_rtn

Local routines : None

Returns : None

3.3.2 Procedure rearrange(x1, y1, x2, y2 : real)

Parameters : x1, y1 : (x, y) co-ordinate of the first point.

x2, y1 : (x, y) co-ordinate of the second point.

Function : It rearranges the co-ordinates of the end points of the line in such a manner so that the point close to origin gets assign to (LineBuf.X1, LineBuf.Y1).

Control Flow : Compares the passed values X1 and X2. The lower value gets assign to LineBuf.X1 and the corresponding value of Y gets assign to LineBuf.Y1. In case X1 and X2 are same then check is performed on Y1 and Y2. The lower value of them is assigned to LineBuf.Y1 and corresponding value of X gets assigned to LineBuf.X1. The other point is assigned to (LineBuf.X2, LineBuf.Y2)

Glb References : None

Local routines : None

Returns : None

3.3.3 Procedure Line_rtn

Parameters : None

Function : Draws a line on the RAM screen and on the displayed screen in the present line style. The end point co-ordinates are rearranged before making a node for line in the main linked list.

Glb References : Draw_csr_rtn , GetMem_rtn ,
Disp_cordinate , Rearrange ,
Disp_relative , Clear_comm ,
Swap_screen_rtn.

Local routines : None

Returns : None

3.3.4 Procedure Rect_rtn

Parameters : None

Function : Draws a rectangle on both RAM screen and Displayed screen in the current line style. The four sides of the rectangle are saved as four lines instead of a rectangle. This is done to facilitate the turning facility to turn a rectangle.

Glb References : Draw_Csr_rtn , GetMem_rtn , Disp_cordinate , Rearrange , Disp_relative , Clear_comm .

Local routines : None

Returns : None

3.3.5 Procedure Circ_rtn

Parameters : None

Function : Draws a circle on both RAM screen and displayed screen in the current line style. The radius of the circle is calculated in world co-ordinate system. An entry for this is made in the main linked list.

Glb References : Draw_csr_rtn , choice_rtn , Disp_cordinate , GetMem_rtn , Disp_relative , Clear_comm , Swap_screen_rtn .

Local routines : None

Returns : None

3.3.6 Procedure Arc_rtn

Parameters : None

Function : Draws an arc on both RAM screen and the displayed screen. The routine prompts the user to enter the first and the second points. The arc is drawn in anticlockwise direction from the first point to the second point. An entry for the arc is made in main linked list.

Glb References : Draw_csr_rtn , Comm_line ,
Disp_cordinate , Choice_rtn ,
Disp_relative , GetMem_rtn ,
Swap_screen_rtn , Clear_comm ,
Angle_rtn.

Local routines : None

Returns : None

3.3.7 Procedure Draw_Square_rtn(x1, y1, x2, y2 : real)

Parameters : x1, y1 : (x, y) co-ordinates of one of the end point of a diagonal.

x2, y2 : (x, y) co-ordinates of the other end point of same diagonal.

Function : It draws a rectangle in black and white colors alternatively to give the effect of blinking.

Glb References : None

Local routines : None

Returns : None

3.3.8 Procedure Zoom_rtn

Parameters : None

Function : Zooms in either on a small area of layout or Zooms out to the full layout on the screen.

Control Flow : It prompts to enter the choice - whether a small area is to be zoomed or the full layout should be displayed on the screen.If the full layout is to be zoomed then it makes the active world co-ordinates equal to the area of layout and redraws the full linked list on to the screen. In case a small window is to be zoomed in, it askes the user to mark the area which is to be zoomed. It modifies the active world co-ordinates according to the area marked and redraws the complete linked list.

Glb References : Swap_screen_rtn , clear_comm ,
Disp_cordinate , Draw_list ,
Disp_relative , comm_line ,
draw_csr_rtn , DWreal .

Local routines : None

Returns : None

3.3.9 Procedure Text_rtn

Parameters : None

Function : Draws a character string on both display screen and the RAM screen. The character size is taken from the global variable 'Txt_sixe_glb'. Once the character size is changed, it will be effective for the subsequent text command, until it is changed again by using the proper command. A node for this string is also created in the main linked list.

Glb References : Swap_screen_rtn , Edit_field ,
Disp_cordinate , Short_beep ,
Draw_csr_rtn , Choice_rtn ,
Draw_txt_rtn , GetMem_rtn .

Local routines :

Procedure Size_rtn;

Parameters : None

Function : Allows the user to change the character size for text.

Glb References : Edit_field

Local routines : None

Returns : None

Return : None

3.3.10 Function Select_Entry (s_ch : char; x1_Loc, y1_Loc, x2_Loc, y2_Loc : real): boolean

Parameters : s_ch : char [L, D, C, A, T] corresponding to the entity.

x1_Loc, y1_Loc : (x,y) co-ordinates of the lower left corner of the rectangle.

x2_Loc, y2_Loc : (x,y) co-ordinates of the upper right corner of the rectangle.

Function : Checks if the passed drawing entity falls completely inside the window.

Glb Reference : None

Local routines : None

Returns : True : if the entity falls completely inside the rectangle

False : otherwise.

3.3.11 Procedure Selection_rtn(x1_s, y1_s, x2_s, y2_s : real; flag : boolean)

Parameters : x1_s, y1_s : (x, y) co-ordinates of the lower left corner of the rectangle.

x2_s, y2_s : (x, y) co-ordinates of the upper right corner of the rectangle.

Function : It selects all those entities which fall completely inside the rectanlge formed by (x1_s, Y1_s) and (x2_S, y2_s) and makes a temporary linked list of all such entities.

The linked list can be referred by pointer - ListPtr. The sel_status field of each element in the linked list is initialized to 1. Once an element is unselected for the operation, this field is made equal to 0.

Glb References : Select_entry , GetMem_rtn

Local routines : None

Returns : None

3.3.12 Function choice_to_select : Char

Parameters : None

Function : Allows the user to unselect any of the entity present in the temporary linked list pointed by pointer - ListPtr.

Control Flow : It asks the user to unselect any of the entity. It displays each and every entity and asks him to make his choice. When an entity is unselected the Sel_status(SelectStatus) field of that entity is made equal to 0. The turning, moving or erasing operation is performed only on those elements which have their sel_status field equal to 1.

Glb References : Short_beep , ClearinkeyBuffer , Comm_line , Draw_csr_rtn .

Local routines : None

Returns : <Esc> key : when the user want to break the command by pressing < Esc > key.

<Ret> key : otherwise.

3.3.13 Procedure Dist_rtn

Parameters : None

Function : Displays the distance between two points on the layout. The distance is measured in world co-ordinate system.

Glb References : Disp_cordinate , Clear_comm , Draw_csr_rtn , Disp_dist .

Local routines :

procedure disp_dist(len : real);

Parameters : len : Distance which is to be displayed.

Function : It displays the distance on both displayed screen and the RAM screen at position (38,1)

Glb References : None

Local routines : None

Returns : None

Return : None

3.3.14 Procedure move_cal(m_c : char ; x_m, y_m : real)

Parameters : m_c : char [D, L, C, A, T] corresponding to the entity to be moved.

x_m, y_m : Displacements in x and y axis for each entity.

Function : It modifies the (x, y) co-ordinates of the entities that are present in the temporary linked list pointed by ListPtr. The modification in co-ordinates are directly reflected in the main linked list pointed by HeadPtr.

Glb References : None

Local routines : None

Returns : None

3.3.15 Procedure move_entity(var x_m, y_m : real)

Parameters : x_m, y_m : Displacement in x and y axis for each entity.

Function : Modifies the co-ordinates of all the elements which are present in the linked list pointed by the pointer - ListPtr and have the select status field equal to 1.

Control Flow : It scans the linked list pointed by ListPtr. When it finds the select status field of an element equal to 1, it's co-ordinates are modified in the main linked list.

Glb References : move_cal

Local routines : None

Returns : None

3.3.16 Procedure Erase_entity

Parameters : None

Function : It removes the entities from the layout.

Control Flow : It scans the linked list pointed by the ListPtr completely. When it finds the select status field of an element equal to 1 (i.e. the element has been selected to erase from the layout), it modifies the status field of this element equal to zero in the main linked list (i.e. the element will not get drawn for the subsequent calls to draw_list).

Glb References : None

Local routines : None

Returns : None

3.3.17 Procedure Turn_cal(ch_t : char; XRef, YRef, Theta: real)

Parameters : ch_t: character [D, L, C, A, T] corresponding to the entity to be turned.

XRef, YRef : (x, y) co-ordinates of the point about which entities have to be turned.

Theta : Angle by which entities have to be turned.

Function : It modifies the co-ordinates of the entity depending upon the character ch_t passed to the routine.

Control Flow : It adds the value of the angle by which the entity to be turned, to the angle of entity with respect to the horizontal line. Using the modified value of the angle, it calculates the new co-ordinates.

Glb References : Angle_rtn

Local routines : None

Returns : None

3.3.18 Procedure Turn_entity(XRef, YRef, Theta : real)

Parameters : XRef, YRef : (x, y) co-ordinate of the point about which entities in the linked list pointed by ListPtr, have to be turned.

Theta : Angle by which entities have to be turned in anticlockwise direction.

Function : It modifies the (x, y) co-ordinates of the entities.The co-ordinates are modified according to the value of (XRef, YRef) and Theta.

Control Flow : It scans the linked List completely. When it finds the select status field of an entity equal to 1, it modifies (x, y) co-ordinates. These changes are made directly in the main linked list pointed by HeadPtr.

Glb References : Turn_cal

Local routines : None

Returns : None

3.3.19 Procedure Free_select

Parameters : None

Function : It releases the memory occupied by the temporary linked list pointed by ListPtr.

Glb References : None

Local routines : None

Returns : None

3.3.20 Procedure Move_rtn(choice : char)

Parameters : choice : ['M', 'N', 'E']

'M' - The procedure is called to execute move command.

'N' - The procedure is called to execute turn command.

'E' - The procedure is called to execute erase command.

Function : This is the controlling program for three commands - move, erase and turn.

Control Flow : It prompts the user to enclose the entities in a rectangular area, for which the command is to be executed. It then finds out all the entities which are completely falling in the rectangle. If any entity is found completely inside it, it calls the respective procedure to execute the command. If this procedure was selected for move command, it asks to enter the displacement. If the procedure was called to execute turn command, it asks to enter the point about which the entities have to be turned and the angle which is measured in anticlockwise direction.

Glb References : Disp_cordinate , Free_select ,
Disp_relative , Turn_entity ,
Selection_rtn , Short_beep ,
Draw_csr_rtn , Edit_field ,
Erase_entity , Draw_List ,
move_entity , comm_line .

Local routines : None

Returns : None

3.3.21 Procedure Select_Lines(x1_s, y1_s, x2_s, y2_s :real)

Parameters : x1_s, y1_s : (x, y) co-ordinates of the lower left corner of the rectangle.
x2_s, y2_s : (x, y) co-ordinates of the upper right corner of the rectangle.

Function : It selects all those lines which can be enclosed completely by the rectangle formed by (x1_s, y1_s) and (x2_s, y2_s).

Control Flow : This routine is used to cut a line. It scans the linked list pointed by HeadPtr. When it finds an entry corresponding to a line, it calls the select_entry routine to check whether it falls completely in the rectangle. If the line can be enclosed by the rectangle, it pushes it on a linked list pointed by ListPtr. The select status field of each line is then initialized to zero.

Glb References : Select_entity , GetMem_rtn

Local routines : None

Returns : None

3.3.22 Function Select_per_line: Char

Parameters : None

Function : It selects the particular line which is to be cut.

Control Flow : All the lines which fall in the rectangle are already present on a temporary linked list pointed by ListPtr. This module asks the user to select the particular line by flashing each line present in the temporary linked list. Once a particular line is selected to cut, the select status field of that line is made one and it does not flashes the remaining lines for the choice.

Glb References : Short_beep , ClearInKeyBuffer

Local routines : None

Returns : <Ret> key : if the line is selected by the user.

<Esc> key : if the user wants to break the command.

3.3.23 Function CutLineValue(loc_x, loc_y : real; var x, y : real): boolean

Parameters : loc_x, loc_y : (x, y) co-ordinates of the point specified by the user.

x, y : (x, y) co-ordinates of the point on the line.

Function : It calculates the co-ordinates of the point on the line which is the foot of the projection from the point (loc_x, loc_y).

Control Flow : This routine first finds the angle of the line, then it calculates the angle of the dummy line from (LineBuf.X1, LineBuf.Y1) to (Loc_x, Loc_Y). After this, it calculates the co-ordinate of the foot of the projection from point (Loc_X, Loc_Y) to the line. It then checks whether the foot falls on the line or Not.

Glb References : Angle_rtn

Local routines : None

Returns : True : if the calculated point is on the line. The co-ordinates of the point are returned in (X,Y).

False : otherwise.

3.3.24 Function Cut_Line(xc1, yc1, xc2, yc2 : real; style : byte): boolean

Parameters : xc1, yc1 : (x, y) co-ordinates of the first point specified by the user.

xc2, yc2 : (x, y) co-ordinates of the other point specified by the user.

Style : Present line style.

Function : It breaks the selected line in two lines.

Control Flow : If the projections of the specified points fall on the line, the co-ordinates of feet of projection are returned in (X1, Y1) and (X2, Y2). It then rearranges these points so that the point closer to the origin is stored in (X1, Y1). If any one of the point is one of the end points of the line, it does not make a new entity in the main linked list but updates the co-ordinates of the line. Otherwise it makes a new node in the linked list as well as modifies the co-ordintes of the existing line in the linked list.

Glb References : CutLineValue , GetMem_rtn

Local routines : None

Returns : True : if it is possible to cut the line.
False : otherwise.

3.3.25 Procedure Cut_rtn

Parameters : None

Function : It allows the user to cut a line in two lines. If the procedure is called repetitively, the line can be cut in as many parts as many the user wishes.

Control Flow : It first asks the user to enclose the line, which is to be cut, in a rectangle. All the lines which fall completely in the rectangle are selected. The user then specifies the particular line. It then asks the user to mark two points on the line (or as close as possible). The portion which falls between these two points is erased from the line.

Glb References : Select_per_line , Free_select ,
Swap_screen_rtn , Short_beep ,
Disp_cordinate , Choice_rtn ,
Draw_csr_rtn , Comm_line ,
Disp_relative , Clear_comm ,
Select_lines , Draw_list .

Local routines : None

Returns : None

3.3.26 Procedure Style_rtn

Parameters : None

Function : Changes the line style. Once a new line style is selected, all the subsequent drawings will take place in the new line style.

Glb References : Draw_square_rtn , Draw_csr_rtn , Swap_screen_rtn .

Local routines : None

Returns : None

3.3.27 Procedure Drag_rtn

Parameters : None

Function : It changes the active window.

Control Flow : It asks the user to enter the reference point about which the active window is to be dragged. Then it asks the user to enter the displacement in the desired direction. The routine changes the values of the active world co-ordinates. If the full layout is displayed on the screen, it does not allow to drag the layout in any direction. Similarly once a boundary of the layout is reached, the layout can not be dragged further in that direction.

Glb References : Comm_line , Short_beep , choice_rtn , Draw_csr_rtn ,

Disp_cordinate , Disp_relative ,
draw_list , Clear_comm .

Local routines : None

Returns : None

3.3.28 Procedure save_choice_rtn

Parameters : None

Function : It asks the user to confirm his choice of saving the modified layout. It saves the modified layout in 'DATA.Fil' file in the project directory.

Glb References : Short_beep , Save_blk_name ,
Save_list .

Local routines : None

Returns : None

3.3.29 Procedure Edit_rtn

Parameters : None

Function : It is the controlling program which gives calls to different modules to serve the user request. After serving the request, the control comes back to this routine again.

Glb References : Plus_rtn , Draw_csr_rtn ,
Dist_rtn , Disp_cordinate ,
Line_rtn , Cursor_pos_rtn ,
Rect_rtn , Creation_rtn ,

Circ_rtn ,
Drag_rtn ,
Text_rtn ,
Zoom_rtn ,
Move_rtn ,
Quit_rtn ,
Dot_rtn ,
Style_rtn ,
Block_rtn ,
Cut_rtn ,
Pg_rtn ,
Arc_rtn .

Local routines : None

Returns : None

3.4 THE HELP MODULE

This module contains the help screens to help the user in drafting editor. The screens contain the description for all the commands available in the editor.

3.5 THE HARDCOPY MODULE

This module of the project includes all the routines required to take hardcopy of a layout. The description of various routines present in this module is mentioned below.

3.5.1 Procedure HardCopy_rtn

Parameters : None

Function : It is the controlling routine to get the hardcopy of a layout. This routine makes calls to other routines to actually print the layout.

Glb References : None

Local routines :

procedure print_rtn

Parameters : None

Function : This routine asks the user to enter the scaling factor for the hardcopy. Once a valid scale factor is provided, the whole layout is divided into a rectangular grid and each cell is printed on the printer.

Glb References : beep, Draw_list

Local routines : None

Returns : None

procedure Scrdump

Parameters : None

Function : It dumps the part of the layout which is displayed on the screen, on the printer.

Glb References : None

Local routines : None

Returns : None

procedure compute_matrix(fac : integer; var Del_x, Del_y : real)

Parameters : fac : Print scale factor

Del_x : No. of cells on the grid in x-axis.

Del_y : No. of cells on the grid in y-axis.

Function : Depending upon the scale factor and the world co-ordinates of the layout, it calculates the total number of rectangular cells required to divide the layout to print.

Glb References : None

Local routines : None

Returns : No. of cells on x-axis and no. of cells on y-axis

Returns : None

CONCLUSION

Inspite of the considerable progress which computer graphics has experienced within the last few years, it should not, by any means, be considered as having reached the stage of full maturity. Thus, the last part which is devoted to future prospects remains indispensable.

This report has explored some of the issues involved in developing device independent graphics software. The graphics system is a package of function, all of which have been discussed in chapter two and three. The report has highlighted the need for a well designed programmer's model of the graphics system. Such a model can be implemented fairly easily for a range of different displays.

To achieve absolute portability of applications on a wide scale, the stress should be given to develop a standard graphics package. A wide degree of portability will not only decrease the programming costs, it will also leave the user free to choose a computer system suitable to his requirements.

BIBLIOGRAPHY

1. Artwick, Bruce A. : *Applied concepts in Microcomputer Graphics*. Prentice-Hall, Englewood cliffs, N.J., 1980.
2. Bethune, James D. and Kee, Bonnie A. : *An introduction to Computer Aided Drafting*. Prentice-Hall, Englewood cliffs, N.J., 1980.
3. Chasen, Sylvan H. : *Geometric Principles and Procedures for computer graphic applications*. Prentice-Hall, Englewood cliffs, N.J., 1978.
4. Enderle, G. and Kansy, K. : *Computer Graphics Programming : GKS - The Graphics standard*. Springer-Verlag,Berlin, 1984.
5. Giloi, Wolfgang K. : *Interactive Computer Graphics : Data structures, Algorithms, Languages*. Prentice-Hall, Englewood cliffs, N.J., 1978.
6. Goetsh, David L. : *Introduction to Computer-Aided Drafting*. Prentice-Hall, Englewood cliffs, N.J., 1983.
7. Litchen, Larry : " Computer Aided Design Applications on Microcomputer ", IEEE *Computer Graphics and applications*, Oct. 1984, p. 25.
8. Marshall, George R. : *Computer graphics in applications*. Prentice-Hall, Englewood cliffs, N.J., 1987.

9. Newman, William M. and Sproull, Robert F. : *Principles of Interactive Computer Graphics*, Second edition.McGraw-Hill, New York.

10. Ryan, Daniel L. : *Principles of Automated Drafting.* Marcel Dekker, New York, 1984.

11. Scott, Joan E. : *Introduction to Interactive Computer Graphics.* John Wiley & Sons, New York, 1982.

12. Sutherland, Ivan : " Computer Display ", *Scientific American*, June, 1970, p. 132.

APPENDIX

```
program Computer_Aided_Drafter;

Uses
  Dos, Crt, ps_const, Ps_fun, Ps_sys, Ps_decab,
  GDriver, GKernel, GShell, GWindow, Ps_Edtf,
  Ps_Glb, Printer;

Var
  error         : integer;
  Chk_flag      : boolean;
  S9            : string[ 9 ];
  Present_Dir   : string[ 40 ];
  f_world       : file of structure;

{$I HardCopy.pas }
{$I Draw.pas }
{$I Help.pas }

(* ######################################## *)

  Function Make_new_dir: boolean;
  begin
    {$I-}
    MkDir( Drive + Path + Dir );
    {$I+}
    if IOResult <> 0 then
    begin
      writeln;
      writeln(' Unable to create Project Directory ...
      writeln(' Press a key to return to DOS. ....' );
      Make_new_dir := False;
      repeat
      until KeyPressed
    end
    else
      Make_new_dir := true;
  end;

(* ######################################## *)

  Function Create_new_dir : boolean;
  var
    c     :  char;
  begin
    Create_new_dir := False;
    GotoXY( 1, 24 );
    c := ' ';
    write( ' Do you want to start a new Project ( Y / |
    repeat
      c := UpCase( ReadKey );
    until c in [ 'Y', 'N' ];
    if c = 'Y' then
    begin
      chk_flag := Make_new_dir;
      if chk_flag = true then
         Create_new_dir := true;
    end
  end;

(* ######################################## *)

 procedure General_data;
 begin
   ClearScreen;
   Assign( f_world, Drive + Path + Dir + 'Ps_wrld.Dat'
   rewrite( f_world );
   StruBuf.Typ := 'W';
   StruBuf.x1  := 0 ;
   StruBuf.y1  := 0 ;
   GotoXY( 2, 20 );
  write( 'Enter the length of construction area ( in mm. ) : ' );
  GotoXY( 63, 20 );
  readln( StruBuf.x2 );
  GotoXY( 2, 22 );
  write( 'Enter the width of construction area ( in mm. ) : ' );
  GotoXY( 63, 22 );
  readln( StruBuf.y2 );
  Write( f_world, StruBuf );
  close( f_world );
end;


Begin
  write(' ENTER THE PROJECT NAME  : A:\');
  readln( Dir );
  {$I-}
  GetDir( 0, Present_Dir );
  {$I+}
  {$I-}
  s9 := Drive + Path + Dir;
  ChDir( s9 );
  {$I+}
  Error := IOResult;
  if Error <> 0 then
  begin
    chk_flag  := Create_new_dir ;
    if chk_flag = false then
       Halt;
  end;
  ChDir( Present_Dir );
  Dir := concat( Dir + '/' );
  InitGraphic;
  SetAspect( 1 );
  init_cursor;
  if FileExist( Drive + Path + Dir + 'Data.fil' ) then
  begin
    GotoXY( 1, 25 );
    write( '     Please Wait! Computing ............. ');
    init_draw_data( true );
    GotoXY( 1, 25 );
    ClrEol;
  end
  else
  begin
     if not FileExist( Drive + Path + Dir + 'Ps_wrld.dat' ) then
     begin
       short_beep;
       General_data;
     end;
     ClearScreen;
     GotoXY( 1, 25 );
     write( '     Please Wait! Computing ............. ');
     Init_world;
     GotoXY( 1, 25 );
     ClrEol;
  end;
  CopyScreen;
  Edit_rtn;
  save_choice_rtn;
  LeaveGraphic
End.
```

```
unit Ps_Glb;

interface
uses
  Dos, Crt, GDriver, GKernel, GWindow, Gshell, Ps_Edtf, Ps_const;

const
  Plus = #43;
  DotKey = #46;
  Drive = 'A:';
  Path = '/';

Type
  String80 = String[ 80 ];
  String60 = String[ 60 ];
  String12 = String[ 12 ];
  PtrtoString = ^String;
  Dot = Record
          Typ            : char;
          D_status       : byte;
          x, y           : real;
          NxtPtr         : PtrtoString
        end;

  Line = Record
           Typ                    : char;
           L_status               : byte;
           x1, y1, x2, y2         : real;
           Style                  : 0..255;
           NxtPtr                 : PtrtoString
        end;

  Circle = Record
             Typ               : char;
             C_status          : byte;
             Xc, Yc, R         : real;     { radius is in world value }
             Style             : 0..255;
             NxtPtr            : PtrtoString
           end;

  Arc = Record
          Typ                           : char;
          A_status                      : byte;
          Xc, Yc, X1, Y1, Theta         : real;
          Style                         : 0..255;
          NxtPtr                        : PtrtoString
        end;

  Txt = Record
           Typ                    : char;
           T_status               : byte;
           X, Y, Size             : real;
           Style, length          : 0..255;
           StrPtr, NxtPtr         : PtrtoString
         end;

  Structure = Record
                Typ            : char;
                x1, y1, x2, y2 : real
              end;

  Select = Record
             Sel_status        : byte;
             EntPtr, NxtPtr    : PtrtoString
           end;

Var
  World_Limit_Glb, Active_World_Glb     : Array[0..3] of real;
  Csr_size_Glb, Size_of_lineBuf,
  size_of_DotBuf, size_of_Ptr,
  size_of_CirBuf, size_of_ArcBuf,
  Elastic_style, Size_of_TxtBuf,
  Size_of_Select                         : byte;
  Csr_X_Glb, Csr_Y_Glb, Csr_Step_Glb,
  Txt_size_Glb                           : Real;
  ch, ch_print                           : char;
  DotBuf                                 : Dot;
  LineBuf                                : Line;
  CirBuf                                 : Circle;
  ArcBuf                                 : Arc;
  TxtBuf                                 : Txt;
  SelectBuf                              : Select;
  DotPtr, LinePtr, HeadPtr, TempPtr,
  CirPtr, ArcPtr, TxtPtr, TxtStrPtr,
  EntPtr, SelectPtr, DummyPtr, ListPtr,
  BlkPtr, PathPtr                        : PtrtoString;
  StruBuf                                : Structure;
  TxtStringGlb                           : String60;
  BlockStrGlb                            : String[ 8 ];
  CreatStrGlb                            : String[ 8 ];
  no_of_ent                              : longint;
  f                                      : text;
  Dir                                    : String[ 9 ];


Procedure ClearInkeyBuffer;
Function fileExist( filename : string80 ) : boolean;
Procedure Beep;
Procedure Short_Beep;
Function Angle_rtn( X_Cent, Y_Cent, X, Y : real):real; { returns angle in deg
 }
Procedure Swap_screen_rtn;
Procedure Draw_txt_rtn( Loc_TxtStr : string60; var x, y : real; size : real )
;
Procedure Draw_List( flag : boolean );
Procedure save_list;
Procedure GetMem_Rtn( Var TempPtr : PtrtoString; MemReq : word );
Procedure DWReal( No : Integer; x1, y2, x2, y1 : real );
Procedure init_draw_data( flag : boolean );
Procedure init_world;
Procedure init_cursor;
Procedure Draw_csr_rtn( Csr_X, Csr_y : real; Csr_Size : Byte );
Function  Quit_rtn: boolean;
Procedure Cursor_pos_rtn( var Csr_X, Csr_Y : real; ch_pos : char );
Procedure pg_rtn( ch_pg : char );
Procedure Disp_cordinate( Csr_X, Csr_Y : real );
Procedure Disp_relative( Csr_X, Csr_Y : real );
Procedure Clear_comm;
Procedure comm_line( x, y : integer; Str : string60 );
Procedure plus_rtn;
Function  Choice_rtn( var  X, Y : real; Var Ch_choice : char ): boolean;

implementation

(* ####################################################### *)
Procedure ClearInkeyBuffer;
var        127
begin
  if keyPressed then
    repeat
      Ch_Ch := Readkey
    until not keyPressed
end;

(* ####################################################### *)

Function fileExist( filename : string80 ) : boolean;
var
```

```
  f                                    : file;
  temp                                 : integer;
begin
  Assign( f, filename );
  {$I-}
  reset( f );
  {$I+};
  temp := IOResult;
  if temp <> 0 then
    fileExist := false
  else
  begin
    fileExist := true;
    close( f )
  end
end;

(* ###################################################### *)

Procedure Beep;
begin
  sound(900);
  delay(200);
  sound(1200);
  delay(100);
  sound(900);
  delay(200);
  nosound;
  ClearInkeyBuffer
end;

(* #######################################################*)
Procedure Short_Beep;
begin
  sound(700);
  delay(100);
  nosound;
end;

(* ######################################################*)

Function Angle_rtn( X_Cent, Y_Cent, X, Y : real):real; { returns angle in deg
 }
var
  dx, dy, Angle_value                  : real;
begin
  dx := x - x_cent;
  dy := y - y_cent;
  if dx = 0 then
  begin
    if y >= y_cent then
      Angle_value := 90
    else
      Angle_value := 270
  end
  else
    if dx > 0.0 then
      if dy >= 0 then
        Angle_value :=  ArcTan( ( y - y_cent ) / ( x - x_cent )) * ( 180 / Pi
 )
      else
        Angle_value := ( ArcTan( ( y - y_cent ) / ( x - x_cent ))* ( 180 / Pi
 ))
                       + 360
  else
    Angle_value := ( ArcTan( ( y - y_cent ) / ( x - x_cent )) * ( 180 / Pi )
)
                   + 180;
  Angle_rtn := Angle_value
end;
```

```

(* ####################################################### *)

Procedure Swap_screen_rtn;

begin
  SelectScreen( 2 );
  CopyScreen;
  SelectScreen( 1 )
end;

(* ####################################################### *)

Procedure Draw_txt_rtn( Loc_TxtStr : string60; var x, y : real; size : real )
;
var
  i             StrLen                                  : byte;
  chStr                                  : string60;
  charFile                               : text;

  procedure DrawChar_rtn( Loc_ChStr : string60 );
  var
    i_Dr, j_Dr, num                      : integer;
    ch_Draw                              : char;
  begin
    i_Dr := length( Loc_ChStr );
    j_Dr := 1;
    while j_Dr <= i_Dr do
    begin
      case Loc_ChStr[ j_Dr ] of
        'B' : begin
                ch_Draw := Loc_ChStr[ j_Dr + 1 ];
                num := ord( Loc_ChStr[ j_Dr + 2 ]) - ord( '0' );
                inc( j_Dr, 3 );
                case ch_Draw of
                  'U' : Y := Y + ( num * size );
                  'D' : Y := Y - ( num * size );
                  'L' : X := X - ( num * size );
                  'R' : X := X + ( num * size );
                  'E' : begin
                          X := X + ( num * size );
                          Y := Y + ( num * size )
                        end;
                  'F' : begin
                          X := X + ( num * size );
                          Y := Y - ( num * size )
                        end;
                  'G' : begin
                          X := X - ( num * size );
                          Y := Y - ( num * size )
                       end;
                  'H' : begin
                          X := X - ( num * size );
                          Y := Y + ( num * size )
                        end
                end
              end;
        'N' : begin
                ch_Draw := Loc_ChStr[ j_Dr + 1 ];
                num := ord( Loc_ChStr[ j_Dr + 2 ] ) - ord( '0' );
                inc( j_Dr, 3 );
                case ch_Draw of
                  'U' : DrawLine( X, Y, X, Y + ( num * size ));
                  'D' : DrawLine                              'L' : DrawLine( X
X - ( num * size ), Y);
                  'R' : DrawLine( X, Y, X + ( num  * size ), Y);
                  'E' : DrawLine( X, Y, X + ( num  * size ),
                                  Y + ( num  * size ));
                  'F' : DrawLine( X, Y, X + ( num  * size ),
                                  Y - ( num  * size ));
                  'G' : DrawLine( X, Y, X - ( num  * size ),
                                  Y - ( num  * size ));
                  'H' : DrawLine( X, Y, X - ( num  * size ),
                                  Y + ( num  * size ))
                end
              end;
        'U', 'D', 'L',
        'R', 'E', 'F',
        'G', 'H'      : begin
                          num := ord( Loc_ChStr[ j_Dr + 1 ]) - ord( '0' );
                          case Loc_ChStr[ j_Dr ] of
                            'U' : begin
                                    DrawLine( X,Y,X,Y + ( num * size ));
                                    Y := Y + ( num * size )
                                  end;
                            'D' : begin
                                    DrawLine( X, Y - ( num * size ), X, Y );
                                    Y := Y - ( num * size )
                                  end;
                            'L' : begin
                                    DrawLine( X - ( num * size ), Y, X, Y );
                                    X := X - ( num * size )
                                  end;
                            'R' : begin
                                    DrawLine( X + ( num  * size ), Y, X, Y );
                                    X := X + ( num * size )
                                  end;
                            'E' : begin
                                    DrawLine( X, Y, X+( num  * size ),
                                              Y + ( num  * size ));
                                    X := X + ( num * size );
                                    Y := Y + ( num * size )
                                  end;
                            'F' : begin
                                    DrawLine( X + ( num  * size ),
                                              Y - ( num  * size ), X, Y );
                                    X := X + ( num * size );
                                    Y := Y - ( num * size )
                                  end;
                            'G' : begin
                                    DrawLine( X - ( num  * size ),
                                              Y - ( num  * size ), X, Y );
                                    X := X - ( num * size );
                                    Y := Y - ( num * size )
                                  'H' : begin
                                    DrawLine( X - ( num  * size ),
                                              Y + ( num  * size ), X, Y );
                                    X := X - ( num * size );
                                    Y := Y + ( num * size )
                                  end
                          end;
                          inc( j_Dr, 2 )
                        end;
        else
          inc( j_Dr )
      end  { end of case }
    end
  end;


begin
  Assign( CharFile, 'Char.dat' );
  StrLen := TxtBuf.Length;
 { Length( Loc_TxtStr );  }
  i := 1;
  reset( CharFile );
  while i <= StrLen do
  begin
  reset( CharFile );
  for j := 1 to ( ord( Loc_TxtStr[ i ] ) - ord( ' ' ) ) do
   readln( CharFile );
  readln( CharFile, ChStr );
  DrawChar_rtn( ChStr );
  inc( i );
  X := X +  ( 7 * Size )
 end;
 Close( CharFile )
end;

(* ###################################################### *)

Procedure Draw_List( flag : boolean );
var
 LocChar                            : Char;
 LocDotBuf                          : Dot;
 LocFlag                            : boolean;
 LocPtr1, LocEntPtr, LocDPtr        : PtrtoString;
begin
 TempPtr := HeadPtr;
 LocEntPtr := Ptr( Seg( LocDotBuf ), Ofs( LocDotBuf ));
 repeat
  LocFlag := true;
  move( TempPtr^, LocChar, 1 );
  Case LocChar of
   'D' : begin                      { of DOT }
       move( TempPtr^, DotPtr^, Size_Of_DotBuf );
       TempPtr := DotBuf.NxtPtr;
       if DotBuf.D_status <> 0 then
        DrawPoint( DotBuf.X, DotBuf.Y )
      end;
   'L' : begin                      { of LINE }
       move( TempPtr^, LinePtr^, Size_Of_LineBuf );
       TempPtr := LineBuf.NxtPtr;
       if flag then
        SetLineStyle( LineBuf.Style );
       if LineBuf.L_status <> 0 then
        DrawLine( LineBuf.X1, LineBuf.Y1, LineBuf.X2, LineBuf.Y2 )
      end;
   'C' : begin                      { of Circle }
       move( TempPtr^, CirPtr^, Size_Of_CirBuf );
       TempPtr := CirBuf.NxtPtr;
       if flag then
        SetLineStyle( CirBuf.Style );
       if CirBuf.C_status <> 0 then
        DrawCircleDirect( WindowX( CirBuf.Xc ), WindowY( CirBuf.Yc ),
                         WindowX(CirBuf.Xc + CirBuf.R) - WindowX(Cir
Buf.Xc),
                         true )
      end;
   'A' : begin                      { of Arc }
       move( TempPtr^, ArcPtr^, Size_Of_ArcBuf );
       TempPtr := ArcBuf.NxtPtr;
       if flag then
        SetLineStyle( ArcBuf.Style );
       if ArcBuf.A_status <> 0 then
        DrawCircleSegment( ArcBuf.Xc, ArcBuf.Yc, ArcBuf.X1, ArcBuf.Y1
,
                  end;
   'T' : begin                      { of Text }
       move( TempPtr^, TxtPtr^, Size_Of_TxtBuf );
       TempPtr := TxtBuf.NxtPtr;
       move( TxtBuf.StrPtr^, TxtstrPtr^, TxtBuf.length + 1 );
       if flag then
        SetLineStyle( TxtBuf.style );
       if TxtBuf.T_status <> 0 then
        Draw_txt_rtn( TxtStringGlb, TxtBuf.x, TxtBuf.y, TxtBuf.size )
      end;
   'W' : begin                   { of Area of layout }
          move( TempPtr^, LinePtr^, Size_Of_LineBuf );
          TempPtr := LineBuf.NxtPtr;
          if flag then
            SetLineStyle( LineBuf.Style );
          if LineBuf.L_status <> 0 then
            DrawSquare( LineBuf.X1, LineBuf.Y1, LineBuf.X2, LineBuf.Y2,
                 False )

        end
  end                            { of case }
 until TempPtr = Nil
end;

(* ############################################## *)

Procedure save_list;
var
 locChar                          : char;
 i_save                           : integer;
 f_Get                            : file;
begin
  Assign( f_Get, Drive + Path + Dir + 'Data.fil' );
  rewrite( f_Get, 1 );
  TempPtr := HeadPtr;
  repeat
   move( tempPtr^, LocChar, 1 );
   case LocChar of
    'W', 'S', 'L' : begin
             move( TempPtr^, LinePtr^, Size_of_LineBuf );
             if LineBuf.L_status <> 0 then
              BlockWrite( f_Get, TempPtr^,
                    Size_of_LineBuf - Size_of_Ptr );
             TempPtr := LineBuf.NxtPtr;
            end;
         'D' : begin
             move( TempPtr^, DotPtr^, Size_of_DotBuf );
             if DotBuf.D_status <> 0 then
              BlockWrite( f_Get, TempPtr^,
                    Size_of_DotBuf - Size_of_Ptr );
             TempPtr := DotBuf.NxtPtr;
            end;
         'C' : begin
             move( TempPtr^, CirPtr^, Size_of_CirBuf );
             if CirBuf.C_status <> 0 then
              BlockWrite( f_Get, TempPtr^,
                    Size_of_cirBuf - Size_of_Ptr );
             TempPtr := CirBuf.NxtPtr;
            end;
         'A' : begin
             move( TempPtr^, ArcPtr^, Size_of_ArcBuf );
             if ArcBuf.A_status <> 0 then
              BlockWrite( f_Get, TempPtr^,
                    Size_      467
l;
         'T' : begin
             move( TempPtr^, TxtPtr^, Size_of_TxtBuf );
             if TxtBuf.T_status <> 0 then
              BlockWrite( f_Get, TempPtr^,
                    Size_of_TxtBuf - 2 * Size_of_Ptr );
             move( TxtBuf.strPtr^, TxtStrPtr^, TxtBuf.Length  +
1 );
             TxtStringGlb := copy( TxtStringGlb, 1, TxtBuf.Lengt
h );
             if TxtBuf.T_status <> 0 then
              BlockWrite( f_Get, TxtStrPtr^, TxtBuf.Length + 1
);
             TempPtr := TxtBuf.NxtPtr;
            end;
```

```

      end;
    until tempPtr = Nil;
    close( f_get );
  end;


(* ####################################################### *)

Procedure GetMem_Rtn( Var TempPtr : PtrtoString; MemReq : word );
var
  LocChar                                 : char;

begin
  if maxAvail > MemReq then
    GetMem( TempPtr, MemReq )
  else
  begin
    beep; Beep; Beep;
    GotoXY( 1, 10 );
    writeln( ' Warning ! No more memory available. Halting the system. ');
    writeln( '    Saving your data to the disk, O.K. ! ');
    save_list;
    repeat
    until KeyPressed;
    LeaveGraphic;
    HALT
  end
end;
(* ####################################################### *)

Procedure DWReal( No : Integer; x1, y2, x2, y1 : real );
var
  ratio, Dx, Dy                           : real;
  WR                                      : real;
begin
  Dx := Abs(x2 - x1 );
  Dy := Abs(y2 - y1 );
  WR := 1.54;
  if Dx > dy then
  begin
    DefineWorld( No, x1, y1 + ( dx / wr ), x2, y1 );
    Active_World_Glb[ 0 ] := x1;
    Active_World_Glb[ 3 ] := y1+(dx / wr);
    Active_World_Glb[ 2 ] := x2 ;
    Active_World_Glb[ 1 ] := y1
  end
  else
  begin
    DefineWorld( No, x1, y2, x1 + ( dy * wr ), y1 );
    Active_World_Glb[ 0 ] := x1;
    Active_World_Glb[ 3 ] := y2;
    Active_World_Glb[ 2 ] := x1 + ( dy * wr) ;
    Active_World_Glb[ 1 ] := y1
  end;
  Csr_X_Glb := ( Active_World_Glb[ 0 ] + Active_World_Glb[ 2 ] ) / 2.0;
  Csr_Y_Glb := ( Active_World_Glb[ 1 ] + Active_World_Glb[ 3 ] ) / 2.0
end;

(* ################################################# *)

Procedure init_draw_data( flag : boolean );
var
  f_data                                  : file;
  f_ch                                    : char;
  TEmpPtr1                                : PtrtoString;
  LocFlag                                 : boolean;
begin
  Assign( f_data, Drive + Path + Dir + 'data.fil' );
  reset( f_data, 1 );
  while not EOF( f_data ) do
  begin
          BlockRead( f_data, f_ch, 1 );
    case f_ch of
      'W' : begin
              EntPtr := Ptr( seg( LineBuf ), ofs( LineBuf ) + 1 );
              BlockRead( f_data, EntPtr^, Size_of_LineBuf - Size_of_ptr - 1 )
;
              if flag then
              begin
                World_Limit_Glb[ 0 ] := LineBuf.x1 - ( 0.05 * LineBuf.x2 );
                World_Limit_Glb[ 1 ] := LineBuf.y1 - ( 0.05 * LineBuf.y2 );
                World_Limit_Glb[ 2 ] := LineBuf.x2 + ( 0.05 * LineBuf.x2 );
                World_Limit_Glb[ 3 ] := LineBuf.y2 + ( 0.05 * LineBuf.y2 );
                DWreal( 1, World_Limit_Glb[ 0 ], World_Limit_Glb[ 3 ], World_
Limit_Glb[ 2 ],
                   World_Limit_Glb[ 1 ]);
                DefineWindow( 1, 0, 8, 79, 191 );
                World_Limit_Glb[ 0 ] := Active_World_Glb[ 0 ];
                World_Limit_Glb[ 1 ] := Active_World_Glb[ 1 ];
                World_Limit_Glb[ 2 ] := Active_World_Glb[ 2 ];
                World_Limit_Glb[ 3 ] := Active_World_Glb[ 3 ];
                SelectWorld( 1 );
                SelectWindow( 1 )
              end;
              GetMem_rtn( TempPtr, Size_of_LineBuf );
              LineBuf.Typ := 'W';
              LineBuf.NxtPtr := HeadPtr;
              Move( LinePtr^, TempPtr^, Size_of_LineBuf );
              SetLinestyle( LineBuf.Style );
              DrawSquare( World_Limit_Glb[ 0 ], World_Limit_Glb[ 1 ], World_L
imit_Glb[ 2 ],
                          World_Limit_Glb[ 3 ], false )
            end;
      'L': begin
              EntPtr := Ptr( seg( LineBuf ), ofs( LineBuf ) + 1 );
              BlockRead( f_data, EntPtr^, Size_of_LineBuf - Size_of_ptr - 1 )
;
              GetMem_rtn( TempPtr, Size_of_LineBuf );
              LineBuf.Typ := 'L';
              LineBuf.NxtPtr := HeadPtr;
              Move( LinePtr^, TempPtr^, Size_of_LineBuf );
              SetLinestyle( LineBuf.Style );
            end;
      'D' : begin
              EntPtr := Ptr( seg( DotBuf ), ofs( DotBuf ) + 1 );
              BlockRead( f_data, EntPtr^, Size_of_DotBuf - Size_of_ptr - 1 );
              GetMem_rtn( TempPtr, Size_of_DotBuf );
              DotBuf.Typ := 'D';
              DotBuf.NxtPtr := HeadPtr;
              Move( DotPtr^, TempPtr^, Size_of_DotBuf );
            'C' : begin
              EntPtr := Ptr( seg( CirBuf ), ofs( CirBuf ) + 1 );
              BlockRead( f_data, EntPtr^, Size_of_CirBuf - Size_of_ptr - 1 );
              GetMem_rtn( TempPtr, Size_of_CirBuf );
              CirBuf.Typ := 'C';
              CirBuf.NxtPtr := HeadPtr;
              Move( CirPtr^, TempPtr^, Size_of_CirBuf );
              SetLinestyle( CirBuf.Style );
            end;
      'A' : begin
              EntPtr := Ptr( seg( ArcBuf ), ofs( ArcBuf ) + 1 );
              BlockRead( f_data, EntPtr^, Size_of_ArcBuf - Size_of_ptr - 1 );
              GetMem_rtn( TempPtr, Size_of_ArcBuf );
              ArcBuf.Typ := 'A';
              ArcBuf.NxtPtr := HeadPtr;
              Move( ArcPtr^, TempPtr^, Size_of_ArcBuf );
              SetLinestyle( ArcBuf.Style );
            end;
      'T' : begin
              EntPtr := Ptr( seg( TxtBuf ), ofs( TxtBuf ) + 1 );
              BlockRead( f_data, EntPtr^, Size_of_TxtBuf - 1 - 2* Size_of_Ptr
 );
              GetMem_rtn( TempPtr, Size_of_TxtBuf );
              TxtBuf.Typ := 'T';
              TxtBuf.NxtPtr := HeadPtr;
              GetMem_rtn( TempPtr1, TxtBuf.Length  + 1 );
              BlockRead( f_data, TxtStrPtr^, TxtBuf.Length + 1 );
              Move( TxtStrPtr^, TempPtr1^, TxtBuf.Length  + 1 );
              TxtBuf.StrPtr := TempPtr1;
              Move( TxtPtr^, TempPtr^, Size_of_TxtBuf );
              SetLinestyle( TxtBuf.Style );
            end;
    end;    { end of case }
    HeadPtr := TempPtr;
  end;  { end of while }
  close( f_data );
  Draw_list( true );
end;

(* ####################################################### *)

 Procedure init_world;
 var
   i                                      : byte;
   f_world                                : file of structure;

 begin
   Assign( f_world, Drive + Path + Dir + 'Ps_wrld.Dat');
   reset( f_world );
   while not EOF( f_world ) do
   begin
     read( f_world, StruBuf );
     Getmem_rtn( TempPtr, Size_of_LineBuf );
     LineBuf.Typ := StruBuf.Typ;
     LineBuf.Style := 0;
     LineBuf.x1 := StruBuf.x1;
     LineBuf.y1 := StruBuf.y1;
     LineBuf.x2 := StruBuf.x2;
     LineBuf.y2 := StruBuf.y2;
     LineBuf.L_status := 1;
     LineBuf.NxtPtr := HeadPtr;
     move( LinePtr^, TempPtr^, Size_of_lineBuf );
     HeadPtr := TempPtr ;
     if StruBuf.Typ = 'W' then
     begin
       World_Limit_Glb[ 0 ] := StruBuf.x1 - ( 0.05 * StruBuf.x2 );
       World_Limit_Glb[ 1 ] := StruBuf.y1 - ( 0.05 * StruBuf.y2 );
       World_Limit_Glb[ 2 ] := StruBuf.x2 + ( 0.05 * StruBuf.x2 );
       World_Limit_Glb[ 3 ] := StruBuf.y2 + ( 0.05 * StruBuf.y2 )
     end
   end;    { end of while }
   close( f_world );
   DWreal( 1, World_Limit_Glb[ 0 ], World_Limit_Glb[ 3 ], World_Limit_Glb[ 2
],
                   World_Limit_Glb[ 1 ]);
   DefineWindow( 1, 0, 8, 79, 191 );
   World_Limit_Glb[ 0 ] := Active_World_Glb[ 0 ];
   World_Limit_Glb[ 1 ] := Active_World_Glb[ 1 ];
   World_Limit_Glb[          WorId_Limit_Glb[ 3 ] := Active_World_Glb[ 3 ];
   SelectWorld( 1 );
   SelectWindow( 1 );
   SetClippingOn;
   DrawBorder;
   GotoXY( 1, 25 );
   write(' wait ! computing Data ...... ');
   Draw_List( true );
  GotoXY( 1, 25 );
  ClrEol;
end;

(* ###################################################### *)

Procedure init_cursor;
begin
  Elastic_Style := 2;
  TxtStringGlb := '
  ';
  BlockStrGlb := '        ';
  CreatStrGlb := '        ';
  Txt_size_Glb := 1;
  Csr_Step_Glb := 10.0;
  Csr_Size_Glb := 2;
  Size_of_Ptr := SizeOf( HeadPtr );
  Size_of_LineBuf := SizeOf( LineBuf );
  Size_of_DotBuf := SizeOf( DotBuf );
  Size_of_CirBuf := SizeOf( CirBuf );
  Size_of_ArcBuf := SizeOf( ArcBuf );
  Size_of_TxtBuf := SizeOf( TxtBuf );
  Size_of_Select := SizeOf( Select );
  LinePtr := Ptr( seg( LineBuf ), ofs( LineBuf ) );
  DotPtr := Ptr( seg( DotBuf ), ofs( DotBuf ) );
  CirPtr := Ptr( seg( CirBuf ), ofs( CirBuf ) );
  ArcPtr := Ptr( seg( ArcBuf ), ofs( ArcBuf ) );
  TxtPtr := Ptr( seg( TxtBuf ), ofs( TxtBuf ) );
  SelectPtr := Ptr( seg( SelectBuf ), ofs( SelectBuf ) );
  TxtStrPtr := Ptr( seg( TxtStringGlb ), ofs( TxtStringGlb ) );
  HeadPtr := Nil;
  TempPtr := Nil
end;

(* ###################################################### *)

Procedure Draw_csr_rtn( Csr_X, Csr_y : real; Csr_Size : Byte );
var
  x, y                                : integer;
  TempStyle                           : word;
begin
  TempStyle := GetLineStyle;
  SetLineStyle( 0 );
  Swap_Screen_rtn;
  Case Csr_Size of
    0 : ;        { blank cursor }
    1 : DrawPoint( Csr_X, Csr_Y );
    2 : begin
          SetWindowModeOff;
          x := WindowX( Csr_X );
          y := WindowY( Csr_Y );
          DrawPoint( x, y );
          DrawLineClipped( x + 5, y, x + 11, y );
          DrawLineClipped( x - 10, y, x - 4, y );
          DrawLineClipped( x, y - 2, x, y - 5 );
          DrawLineClipped( x, y + 2, x, y + 5 );
          SetWindowModeOn
        end;
    3 : begin
          DrawLine( Csr_X, Active_World_Glb[ 1 ],
                    Csr_X, Active_World_Glb[ 3 ]  );
          DrawLine( Active_World_Glb[ 0 ], Csr_Y,
                    Active_World_Glb[ 2 ], Csr_Y )
        end
  end;
        end;

(* ###################################################### *)

```

```
 Function Quit_rtn: boolean;
 var
   ch_quit                                : char;
 begin
   GotoXY( 1, 25 );
   write(' Do you want to quit ( Y or N ) ?  ');
   ch_quit := ' ';
   repeat
     ch_quit := upcase( ReadKey )
   until ch_quit in [ 'Y', 'N' ];
   GotoY( 1, 25 );
   ClrEol;
   Quit_rtn := ( ch_quit = 'Y' )
 end;

(* ##################################################### *)

Procedure Cursor_pos_rtn( var Csr_X, Csr_Y : real; ch_pos : char );
var
  x, y                                     : real;
begin
  case ch_pos of
    'R' : x := Csr_X + Csr_Step_Glb;
    'U' : y := Csr_Y + Csr_Step_Glb;
    'L' : x := Csr_X - Csr_Step_Glb;
    'D' : y := Csr_Y - Csr_Step_Glb
   end;
   case ch_pos of
     'R', 'L' : if ( x < Active_World_Glb[ 0 ] ) or ( x > Active_World_Glb[ 2 ] ) then
                  Beep
                else
                  Csr_X := x;
     'U', 'D' : if ( y < Active_World_Glb[ 1 ] ) or ( y > Active_World_Glb[ 3 ] ) then
                  Beep
                else
                  Csr_Y := y
   end
end;

(* ##################################################### *)

Procedure pg_rtn( ch_pg : char );
begin
  case ch_pg of
    'U' : begin
            csr_step_Glb := csr_step_Glb * 10.0;
            if (Csr_step_Glb > (Active_World_Glb[2] - Active_World_Glb[0])) or
               (Csr_step_Glb > (Active_World_Glb[3] - Active_World_Glb[1])) then
              csr_step_Glb := csr_step_Glb / 10.0
          end;
    'D' : if csr_step_Glb <> 1.0 then
            csr_step_Glb := csr_step_Glb / 10.0
  end
end;

(* ##################################################### *)

Procedure Disp_cordinate( Csr_X, Csr_Y : real );
begin
  GotoXY( 64, 1 );
  write( Csr_x:7:0, ' , ', Csr_Y:7:0 );
  SelectScreen( 2 );

  GotoXY( 64, 1 );
  write( Csr_x:7:0, ' , ', Csr_Y:7:0 );
  SelectScreen( 1 )
```

```
end;

(* ##################################################
Procedure Disp_relative( Csr_X, Csr_Y : real );
begin
  GotoXY( 38, 1 );
  write( '( ',Csr_x:7:0, ' , ', Csr_Y:7:0, ' )' );
  SelectScreen( 2 );

  GotoXY( 38, 1 );
  write( '( ', Csr_x:7:0, ' , ', Csr_Y:7:0,' )' );
  SelectScreen( 1 )
end;

(* ##################################################### *)

Procedure Clear_comm;
begin
  GotoXY( 1, 1 );
  ClrEol;
  SelectScreen( 2 );

  GotoXY( 1, 1 );
  ClrEol;
  SelectScreen( 1 )
end;

(* ##################################################### *)

Procedure comm_line( x, y : integer; Str : string60 );
begin
  GotoXY( x, y );
  ClrEol;
  write( str );
  SelectScreen( 2 );

  GotoXY( x, y );
  ClrEol;
  write( str );
  SelectScreen( 1 )
end;

(* ##################################################### *)

Procedure plus_rtn;
begin
  inc( Csr_Size_Glb );
  Csr_Size_Glb := ( Csr_Size_Glb mod 4 )
end;

(* ##################################################### *)

Function Choice_rtn( var  X, Y : real; Var Ch_choice : char ): boolean;
begin
  Choice_rtn := True;
  ch_Choice := ReadKey;
  case ch_Choice of
    #0   : begin
             ch_Choice := ReadKey;
             Case ch_Choice of
                 Front  : Cursor_pos_rtn( X, Y, 'R' );
                 Up     : Cursor_pos_rtn( X, Y, 'U' );
                 Back   : Cursor_pos_rtn( X, Y, 'L' );
                 Down   : Cursor_pos_rtn( X, Y, 'D' );
                 PgUp   : begin
                            Pg_rtn( 'U' );
                            Choice_rtn := false
                          end;
                 PgDn   : begin
```

```
                           Pg_rtn( 'D' );
                           Choice_rtn := false
                         end;
                 else
                   Choice_rtn := false;
             end
           end;
    Plus : Plus_rtn;
    else
      Choice_rtn := false;
  end
end;
 (* ########################################## *)
end.
```

```
(* ##################################################### *)
Procedure Dot_rtn;
begin
  Swap_screen_rtn;
  DrawPoint( Csr_x_Glb, Csr_y_Glb );
  CopyScreen;
  GetMem_rtn( TempPtr, Size_of_DotBuf );
  DotBuf.Typ := 'D';
  DotBuf.x  := Csr_x_Glb;
  DotBuf.y  := Csr_y_Glb;
  DotBuf.D_status := 1;
  DotBuf.NxtPtr := HeadPtr;
  move( DotPtr^, TempPtr^, Size_of_DotBuf );
  HeadPtr := TempPtr
end;

(* ##################################################### *)

Procedure rearrange( x1, y1, x2, y2 : real );
begin
  if x1 () x2 then
  begin
    if x1 ( x2 then
    begin
      LineBuf.x1  := x1;
      LineBuf.y1  := y1;
      LineBuf.x2  := x2;
      LineBuf.y2  := y2;
    end
    else
    begin
      LineBuf.x1  := x2;
      LineBuf.y1  := y2;
      LineBuf.x2  := x1;
      LineBuf.y2  := y1;
    end
  end
  else
  begin
    if y1 ( y2 then
    begin
      LineBuf.x1  := x1;
      LineBuf.y1  := y1;
      LineBuf.x2  := x2;
      LineBuf.y2  := y2;
    end
    else
    begin
      LineBuf.x1  := x2;
      LineBuf.y1  := y2;
      LineBuf.x2  := x1;
      LineBuf.y2  := y1;
    end
  end;
end;

(* ##################################################### *)

Procedure Line_rtn;
var
  Temp_style                          : word;
  X_Line, Y_Line                      : real;
  Csr_Line_Flag                       : boolean;
  ch_I                                : char;
begin
  Temp_Style := GetLineStyle;
  X_Line := Csr_X_Glb;
  Y_line := Csr_Y_Glb;
  Csr_Line_flag := true;
  SetLineStyle( Elastic_style );
  repeat
    if csr_Line_flag then
    begin
      Draw_csr_rtn( X_Line, Y_Line, csr_size_Glb );
      DrawLine( X_Line, Y_Line, Csr_X_Glb, Csr_Y_Glb );
      Disp_Cordinate( X_Line, Y_Line );
      Disp_relative( X_Line - Csr_X_Glb, Y_Line - Csr_Y_Glb );
      Csr_Line_Flag := true
    end;
    csr_line_flag := Choice_rtn( x_Line, y_Line, ch_l )
  until ch_l in [ Escape, Return ];
  SetLineStyle( Temp_Style );
  if ch_l <> Escape then
  begin
    Swap_Screen_rtn;
    DrawLine( Csr_X_Glb, Csr_Y_Glb, x_Line, y_Line );
    CopyScreen;
    Getmem_rtn( TempPtr, Size_of_LineBuf );
    LineBuf.Typ := 'L';
    LineBuf.Style := Temp_Style;
    Rearrange( Csr_X_Glb, Csr_Y_Glb, x_Line, y_Line );
    LineBuf.L_status := 1;
    LineBuf.NxtPtr := HeadPtr;
    move( LinePtr^, TempPtr^, Size_of_lineBuf );
    HeadPtr := TempPtr
  end;
  Csr_X_Glb := x_Line;
  Csr_Y_Glb := y_line;
  Clear_comm
end;

(* ##################################################### *)

Procedure Rect_rtn;
var
  Temp_style, i                         : word;
  X_Box, Y_Box                          : real;
  Rect_flag                             : boolean;
  ch_B                                  : char;
begin
  Temp_Style := GetLineStyle;
  X_Box := Csr_X_Glb;
  Y_Box := Csr_Y_Glb;
  Rect_flag := true;
  SetLineStyle( Elastic_style );
  repeat
    if Rect_flag then
    begin
      Draw_csr_rtn( X_Box, Y_Box, csr_size_Glb );
      DrawSquare( X_Box, Y_Box, Csr_X_Glb, Csr_Y_Glb, False );
      Disp_Cordinate( X_Box, Y_Box);
      Disp_relative( X_Box - Csr_X_Glb, Y_Box - Csr_Y_Glb );
      Rect_flag := true
    end;
    Rect_flag := Choice_rtn( x_Box, y_Box, ch_B );
  until ch_B in [ Escape, Return ];
  SetLineStyle( Temp_Style );
  if ch_B <> Escape then
  begin
    Swap_Screen_rtn;
    DrawSquare( Csr_X_Glb, Csr_Y_Glb, x_Box, y_Box, False );
    CopyScreen;
    LineBuf.Typ := 'L';
    LineBuf.Style := Temp_Style;
    if (( Csr_x_Glb - x_Box ) = 0 ) or (( Csr_y_Glb - y_Box ) = 0 ) then
    begin
      Rearrange( Csr_X_Glb, Csr_Y_Glb, x_Box, y_Box );
      LineBuf.NxtPtr :=         I40          HeadPtr := TempPtr
    end
    else
    begin

      for i := 1 to 4 do
      begin
        Getmem_rtn( TempPtr, Size_of_LineBuf );
        LineBuf.L_status := 1;
        case i of
          1 : begin
                Rearrange( Csr_X_Glb, Csr_Y_Glb, Csr_x_Glb, y_Box );
              end;
          2 : begin
                Rearrange( X_Box, Csr_Y_Glb, X_Box, y_Box );
              end;
          3 : begin
                Rearrange( Csr_X_Glb, Csr_Y_Glb, X_Box, Csr_Y_Glb );
              end;
          4 : begin
                Rearrange( Csr_X_Glb, Y_Box, X_Box, Y_Box );
              end;
        end;
        LineBuf.NxtPtr := HeadPtr;
        move( LinePtr^, TempPtr^, Size_of_lineBuf );
        HeadPtr := TempPtr
      end
    end
  end;
  Csr_X_Glb := x_Box;
  Csr_Y_Glb := y_Box;
  Clear_Comm
end;

(* ##################################################### *)

Procedure Circ_rtn;
var
  Temp_style                              : word;
  X_C, Y_C                                : real;
  Cir_flag                                : boolean;
  ch_C                                    : char;
  Radius_C                                : integer;
begin
  Temp_Style := GetLineStyle;
  X_C := Csr_X_Glb;
  Y_C := Csr_Y_Glb;
  Cir_flag := true;
  SetLineStyle( Elastic_style );
  repeat
    if cir_flag then
    begin
      Draw_csr_rtn( X_C, Y_C, csr_size_Glb );
      CirBuf.R := Sqrt( Sqr( x_C - Csr_X_Glb ) + Sqr( Y_C - Csr_Y_Glb ) );
      radius_C := WindowX( Csr_X_Glb + CirBuf.R ) - WindowX( Csr_X_Glb );
      DrawCircleDirect( WindowX( Csr_X_Glb ), WindowY( Csr_Y_Glb ), radius_C
,
                        true );
      Disp_Cordinate( X_C, Y_C );
      Disp_relative( X_C - Csr_X_Glb, Y_C - Csr_Y_Glb );
    end;
    cir_flag := Choice_rtn( x_C, y_C, ch_C )
  until ch_C in [ Escape, Return ];
  SetLineStyle( Temp_Style );
  if ch_C <> Escape then
  begin
    Swap_Screen_rtn;
    CirBuf.R := Sqrt( Sqr( x_C - Csr_X_Glb ) + Sqr( y_C - Csr_Y_Glb ) );
    Radius_C := WindowX( Csr_X_Glb + CirBuf.R ) - WindowX( Csr_X_Glb );
    DrawCircleDirect( WindowX( Csr_X_                                   radius_C ,
ue );
```

```
    CopyScreen;
    Getmem_rtn( TempPtr, Size_of_CirBuf );
    CirBuf.Typ := 'C';
    CirBuf.Style := Temp_Style;
    CirBuf.Xc := Csr_X_Glb;
    CirBuf.Yc := Csr_Y_Glb;
    CirBuf.C_status := 1;
    CirBuf.NxtPtr := HeadPtr;
    move( CirPtr^, TempPtr^, Size_of_CirBuf );
    HeadPtr := TempPtr
  end;
  Csr_X_Glb := x_C;
  Csr_Y_Glb := y_C;
  Clear_comm
end;

(* ######################################################*)

Procedure Arc_rtn;
Var
  Temp_radius                           : integer;
  Arc_Flag                              : Boolean;
  X1, Y1, X2, Y2, Radius_A, Theta1,
  Theta2                                : real;
  ch_A                                  : char;
  TempStyle                             : word;
begin
  TempStyle := GetLineStyle;
  ArcBuf.Xc := Csr_X_Glb;
  ArcBuf.Yc := Csr_Y_Glb;
  X1 := Csr_X_Glb;
  Y1 := Csr_Y_Glb;
  SetLineStyle( Elastic_Style );
  Arc_flag := true;
  repeat
    Short_Beep;
    comm_line(1, 25,' Enter first point on the Arc ');
    if Arc_flag then
    begin
      Draw_Csr_rtn( x1, y1, csr_size_Glb );
      Disp_cordinate( x1, y1 );
      Disp_relative( X1 - Csr_X_Glb, Y1 - Csr_Y_Glb );
      radius_A := Sqrt( Sqr( X1 - Csr_X_Glb ) + Sqr( Y1 - Csr_Y_Glb ) );
      Temp_radius := WindowX( Csr_X_Glb + radius_A ) - WindowX( Csr_X_Glb );
      DrawCircleDirect( WindowX( Csr_X_Glb ), WindowY( Csr_Y_Glb ), Temp_radi
us,
                        true )
    end;
    Arc_flag := Choice_rtn( x1, y1, ch_A )
  until ch_A in [ Escape, Return ];
  Arc_flag := true;
  X2 := X1;
  Y2 := Y1;
  if ch_A <> Escape then
  begin
    Short_Beep;
    comm_line(1, 25,' Enter the Second point on the Arc');
    repeat
      Arc_flag := Choice_rtn( x2, y2, ch_A );
      if Arc_flag then
      begin
        Draw_Csr_rtn( x2, y2, csr_size_Glb );
        Disp_cordinate( x2, y2 )
      end;
      DrawCircleDirect( WindowX( Csr_X_Glb ), WindowY( Csr_Y_Glb ),
                        Temp_radius, true )
    until ch_A in [ Escape, Return ];
    if ( ch_A <> Escape ) and (( x1 <> x2 ) or ( y1 <> y2 )) then
                    Theta1 := Angle_rtn( Csr_X_Glb, Csr_Y_Glb, X1, Y1 );
      Theta2 := Angle_rtn( Csr_X_Glb, Csr_Y_Glb, X2, Y2 );
      if Theta1  <= Theta2  then
        ArcBuf.Theta := Theta2 - Theta1
      else
        ArcBuf.Theta := 360 - ( Theta1 - Theta2 );
      ArcBuf.Typ := 'A';
      ArcBuf.Xc := Csr_X_Glb;
      ArcBuf.style := TempStyle;
      ArcBuf.Yc := Csr_Y_Glb;
      ArcBuf.X1 := X1;
      ArcBuf.Y1 := Y1;
      ArcBuf.A_status := 1;
      GetMem_rtn( TempPtr, Size_of_ArcBuf );
      ArcBuf.NxtPtr := HeadPtr;
      move( ArcPtr^, TempPtr^, Size_of_ArcBuf );
      HeadPtr := TempPtr;
      Swap_Screen_rtn;
      SetLineStyle( TempStyle );
      DrawCircleSegment( Csr_X_Glb, Csr_Y_Glb, X1, Y1, 1, 1, ArcBuf.Theta,
                         1, '', 0, 0 );
      CopyScreen
    end
  end;
  comm_line(1, 25,'                                    ');
  Csr_X_Glb := X2;
  Csr_Y_Glb := Y2;
  SetLineStyle( TempStyle );
  Clear_comm
end;

(* ####################################################### *)

Procedure Draw_Square_rtn( x1, y1, x2, y2 : real );
begin
  repeat
    SetColorBlack;
    DrawSquare( x1, y1, x2, y2, False );
    SetColorWhite;
    DrawSquare( x1, y1, x2, y2, False )
  until KeyPressed
end;


(* ####################################################### *)

Procedure Zoom_rtn;
var
  ch_z                                  : char;
  x1, y1, x2, y2                        : real;
  TempStyle                             : word;
  Zoom_flag                             : boolean;
begin
  GotoXY( 1, 25 );
  Short_beep;
  write(' Zoom to Actual size or Zoom a window ( A/W ) ? ');
  repeat
    ch_z := upcase( ReadKey )
  until ch_z in [ 'A', 'W', Escape ];
  TempStyle := GetLineStyle;
  if ch_z = 'A' then
  begin
    DWreal( 1, World_Limit_Glb[ 0 ], World_Limit_Glb[ 3 ], World_Limit_Glb[ 2
  ],
                    World_Limit_Glb[ 1 ]);
    SelectWorld( 1 );
    SelectWindow( 1 );
    ClearScreen;
    SetLineStyle( 0 );
    SetClippingOn;
    DrawBorder;
```

```
      GotoXY( 1, 25 );
      ClrEol;
      Write( ' Please wait ! Computing ...... ');
      draw_list( true );
      GotoXY( 1, 25 );
      ClrEol;
      Clear_Comm;
      CopyScreen
    end
    else
      if ch_z = 'W' then
      begin
        x1 := Csr_X_Glb;
        y1 := Csr_Y_Glb;
        SetLineStyle( Elastic_style );
        Short_beep;
        comm_line(1, 25,' Enter first point ');
        Zoom_flag := true;
        repeat
          zoom_flag := Choice_rtn( x1, y1, ch_z );
          if zoom_flag then
          begin
            Draw_Csr_rtn( x1, y1, csr_size_Glb );
            Disp_cordinate( x1, y1 )
          end
        until ch_z in [ Return, Escape ];
        zoom_flag := true;
        x2 := x1;
        y2 := y1;
        if ch_z <> Escape then
        begin
          Short_beep;
          comm_line(1, 25,' Enter the diagonal point ');
          repeat
            zoom_flag := Choice_rtn( x2, y2, ch_z );
            if zoom_flag then
            begin
              Draw_Csr_rtn( x2, y2, csr_size_Glb );
              Disp_cordinate( x2, y2 );
              DrawSquare( x1, y1, x2, y2, false );
              Disp_relative( X2 - X1, Y2 - Y1 )
            end
          until ch_z in [ Return, Escape ];
          if ch_z <> Escape then
          begin
            if ( x1 <> x2 ) and ( y1 <> y2 ) then
            begin
              if x1 < x2 then
                if y1 < y2 then
                  DWReal( 1, x1, y2, x2, y1 )
                else
                  DWReal( 1, x1, y1, x2, y2 )
              else
                if y1 < y2 then
                  DWReal( 1, x2, y2, x1, y1 )
                else
                  DWReal( 1, x2, y1, x1, y2 );
            end;
            SelectWorld( 1 );
            SelectWindow( 1 );
            ClearScreen;
            SetLineStyle( 0 );
            SetClippingOn;
            DrawBorder;
            GotoXY( 1, 25 );
            ClrEol;
            Write( ' Please wait ! Computing  ...... ');
            draw_list( true );
            GotoXY( 1, 25 );
            ClrEol;
          Clear_Comm;
          CopyScreen;
        end
        else
          Swap_screen_rtn;
      end
      else
        Swap_screen_rtn;
    end;
  Comm_line(1, 25,'                                                            ');
  SetLineStyle( TempStyle )
end;

(* ######################################################## *)
Procedure Text_rtn;
var
  ch_T                                  : char;
  TempStyle                             : word;
  Index, i                              : Byte;
  x1, y1                                : real;
  TxtString                             : string[60];
  Txt_flag, Edited, out                 : boolean;
  SizeStr                               : string[ 5 ];

  Procedure Size_rtn;
  var
    ch_s                                : char;
    Loc_Size                            : real;
    Result                              : integer;
  begin
    Txt_flag := true;
    GotoXY( 1, 25 );
    ClrEol;
    Short_beep;
    write( ' Enter Text Size ( Present size : ', Txt_size_Glb:5:1 ,' ) : ');
    Edited := false;
    Index := 1;
    GotoXY( 45, 25 );
    write( SizeStr );
    GotoXY( 45, 25 );
    ch_s := Edit_field( SizeStr, 0, 5, false, index, 45, 25, Edited, 0 );
    if ch_s <> Escape then
    begin
      val( SizeStr, Loc_size, Result );
      if Loc_Size > 0.0 then
        Txt_size_Glb := Loc_size
    end;
    SelectWorld( 1  );
    SelectWindow( 1 )
  end;

begin
  TempStyle := GetLineStyle;
  SizeStr := '     ';
  SetLineStyle( Elastic_style );
  x1 := Csr_X_Glb;
  y1 := Csr_Y_Glb;
  Txt_flag := True;
  GotoXY( 1, 25 );
  Short_beep;
  write(' Enter Text : ');
  Edited := False;
  Index := 1;
  GotoXY( 15, 25 );
  write( TxtStringGlb );
  GotoXY( 15, 25 );
  ch_T := Edit_Field( TxtStringGlb, 2, 60, False, index, 15, 25, Edited, 0 );
  Out := false;
  i := 60;
  while (i >= 1) and (not Out ) do
  begin
    if TxtStringGlb[i] <> ' ' then
           491      else
      Dec( i );
  end;
  SelectWorld( 1  );
  SelectWindow( 1 );
  TxtString := '';
  TxtString := Copy( TxtStringGlb, 0, i);
  GotoXY( 1, 25 );
  ClrEol;
  ch_T := Return;
  if ( ch_T <> Escape ) and ( Length( TxtString ) <> 0 ) then
  begin
    repeat
      if Txt_flag then
      begin
        Draw_Csr_rtn( x1, y1, Csr_Size_Glb );
        Disp_cordinate( x1, y1 );
        DrawSquare( x1, y1, x1 + ( 7 * Length( TxtString ) * Txt_size_Glb ),
                    y1 - ( 9 * Txt_size_Glb ), false )
      end;
      Txt_flag := choice_rtn( x1, y1, ch_T );
      if upcase( ch_T ) = 'S' then
      begin
        Size_rtn;
        Txt_flag := true
      end
    until ch_T in [ Return, Escape ];
    SetLineStyle( 0 );
    if ch_T <> Escape then
    begin
      Swap_screen_rtn;
      Csr_X_Glb := x1;
      Csr_Y_Glb := y1;
      TxtBuf.x := x1;
      TxtBuf.y := y1;
      TxtBuf.Length := Length( TxtString );
      Draw_txt_rtn( TxtString, x1, y1, Txt_size_Glb );
      CopyScreen;
      GetMem_rtn( TempPtr, Length( TxtString ) + 1 );
      move( TxtstrPtr^, TempPtr^, Length( TxtString ) + 1 );
      TxtBuf.strPtr := TempPtr;
      GetMem_rtn( TempPtr, Size_Of_TxtBuf );
      TxtBuf.Typ := 'T';
      TxtBuf.size := Txt_size_Glb;
      TxtBuf.style := 0;
      TxtBuf.Length := Length( TxtString );
      TxtBuf.T_status := 1;
      TxtBuf.NxtPtr := HeadPtr;
      move( TxtPtr^, TempPtr^, Size_of_txtBuf );
     HeadPtr := TempPtr
    end
  end;
  SetLineStyle( TempStyle )
end;

(* ########################################### *)

Function Select_Entry( s_ch : char;
                       x1_Loc, y1_Loc, x2_Loc, y2_Loc : real ): boolean;
var
  LocPtr1                                : PtrtoString;
  LocChar                                : char;
  LocSel_Flag                            : boolean;
begin
  Select_Entry := false;
  case s_ch of
    'L' : begin
```

```
                      ( LineBuf.x2 > x1_Loc ) and ( LineBuf.x2 < x2_Loc ) and
              ( LineBuf.y2 > y1_Loc ) and ( LineBuf.y2 < y2_Loc ) then
             Select_Entry := true;
         end;
   'D' : begin
           if ( DotBuf.x > x1_Loc ) and ( DotBuf.x < x2_Loc ) and
              ( DotBuf.y > y1_Loc ) and ( DotBuf.y < y2_Loc ) then
             Select_Entry := true;
         end;
   'T' : begin
           if ( TxtBuf.x > x1_Loc ) and ( TxtBuf.x < x2_Loc ) and
              ( TxtBuf.y > y1_Loc ) and ( TxtBuf.y < y2_Loc ) and
              ( ( TxtBuf.x + ( TxtBuf.Length * TxtBuf.Size * 7 ) ) < x2_Loc
) and
              ( ( TxtBuf.y - ( 9 * TxtBuf.Size ) ) > y1_Loc ) then
             Select_Entry := true;
         end;
   'C' : begin
           if ( CirBuf.xc > x1_Loc ) and ( CirBuf.xc < x2_Loc ) and
              ( CirBuf.yc > y1_Loc ) and ( CirBuf.yc < y2_Loc ) and
              ( ( CirBuf.xc + CirBuf.R ) < x2_Loc ) and
              ( ( CirBuf.xc - CirBuf.R ) > x1_Loc ) and
              ( ( CirBuf.yc - CirBuf.R ) > y1_Loc ) and
              ( ( CirBuf.yc + CirBuf.R ) < y2_Loc ) then
             Select_Entry := true;
         end;
   'A' : begin
           if ( ArcBuf.xc > x1_Loc ) and ( ArcBuf.xc < x2_Loc ) and
              ( ArcBuf.yc > y1_Loc ) and ( ArcBuf.yc < y2_Loc ) and
              ( ArcBuf.x1 > x1_Loc ) and ( ArcBuf.x1 < x2_Loc ) and
              ( ArcBuf.y1 > y1_Loc ) and ( ArcBuf.y1 < y2_Loc ) then
            Select_Entry := true;
         end;
  end;  { end of case }
end;

(* ###################################################### *)

Procedure Selection_rtn( x1_s, y1_s, x2_s, y2_s : real; flag : boolean );
var
  f_sel                                 : file;
  ch_s                                  : char;
  flag_sel                              : boolean;
  Loc_x, Loc_y                          : real;
begin
  DummyPtr := Nil;
  ListPtr := Nil;
  TempPtr :=  HeadPtr;
  no_of_ent := 0;
  repeat
    move( TempPtr^, ch_s, 1 );
    case ch_s of
      'W',
      'L' : begin
              move( TempPtr^, LinePtr^, Size_of_LineBuf );
              if LineBuf.L_Status <> 0 then
              begin
                                         begin
                  flag_sel := Select_entry( 'L', x1_s, y1_s, x2_s, y2_s );
                  if flag_sel then
                  begin
                    if not( ( flag = false ) and ( ch_s in ['W', 'S' ] )) the
n
                    begin
                      SelectBuf.EntPtr := TempPtr;
                      GetMem_Rtn( DummyPtr, Size_of_Select );
                      inc( no_of_ent );
```

```
          end
          else
           flag_sel := false;
          if ch_s = 'L' then
          begin
           SetColorBlack;
           DrawLine( LineBuf.x1, LineBuf.y1, LineBuf.x2, LineBuf.y
2 );
           SetColorWhite;
           DrawLine( LineBuf.x1, LineBuf.y1, LineBuf.x2, LineBuf.y
2 )
          end
          else
          begin
           SetColorBlack;
           DrawSquare( LineBuf.x1, LineBuf.y1, LineBuf.x2, LineBuf
.y2,
               false );
           SetColorWhite;
           DrawSquare( LineBuf.x1, LineBuf.y1, LineBuf.x2, LineBuf
.y2,
               false )
          end;
         end;
        end;
       end;
       TempPtr := LineBuf.NxtPtr
      end;
   'D' : begin
       move( TempPtr^, DotPtr^, Size_of_DotBuf );
       if DotBuf.D_Status <> 0 then
       begin
        flag_sel := Select_entry( 'D', x1_s, y1_s, x2_s, y2_s );
        if flag_sel then
        begin
         SelectBuf.EntPtr := TempPtr;
         GetMem_Rtn( DummyPtr, Size_of_Select );
         SetColorBlack;
         DrawPoint( DotBuf.x, DotBuf.y );
         SetColorWhite;
         DrawPoint( DotBuf.x, DotBuf.y );
         inc( no_of_ent )
        end;
       end;
       TempPtr := DotBuf.NxtPtr
      end;
   'C' : begin
       move( TempPtr^, CirPtr^, Size_of_CirBuf );
       if CirBuf.C_Status <> 0 then
       begin
        flag_sel := Select_entry( 'C', x1_s, y1_s, x2_s, y2_s );
        if flag_sel then
        begin
         SelectBuf.EntPtr := TempPtr;
         GetMem_Rtn( DummyPtr, Size_of_Select );
         inc( no_of_ent );
         SetColorBlack;
                 DrawCircleDirect( WindowX( CirBuf.Xc ), Win
dowY( CirBuf.Yc ),
                 W                Buf.Xc),
                 true );
         SetColorWhite;
         DrawCircleDirect( WindowX( CirBuf.Xc ), WindowY( CirBuf.Yc
),
                 WindowX(CirBuf.Xc + CirBuf.R) - WindowX(Cir
Buf.Xc),
                 true )
        end;
       end;
```

```
              TempPtr := CirBuf.NxtPtr
            end;
      'A' : begin
              move( TempPtr^, ArcPtr^, Size_of_ArcBuf );
              if ArcBuf.A_Status <> 0 then
              begin
                flag_sel := Select_entry( 'A', x1_s, y1_s, x2_s, y2_s );
                if flag_sel then
                begin
                  SelectBuf.EntPtr := TempPtr;
                  GetMem_Rtn( DummyPtr, Size_of_Select );
                  inc( no_of_ent );
                end;
              end;
              TempPtr := ArcBuf.NxtPtr
            end;
      'T' : begin
              move( TempPtr^, TxtPtr^, Size_of_TxtBuf );
              if TxtBuf.T_Status <> 0 then
              begin
                move( TxtBuf.StrPtr^, TxtStrPtr^, TxtBuf.Length );
                flag_sel := Select_entry( 'T', x1_s, y1_s, x2_s, y2_s );
                if flag_sel then
                begin
                  SelectBuf.EntPtr := TempPtr;
                  GetMem_Rtn( DummyPtr, Size_of_Select );
                  inc( no_of_ent );
                  DrawSquare( TxtBuf.x, TxtBuf.y, TxtBuf.x + ( 7 * TxtBuf.Length * TxtBuf.Size ) ,
                                TxtBuf.y - ( 9 * TxtBuf.size ), false );
                end;
              end;
              TempPtr := TxtBuf.NxtPtr
            end
    end;  { end of case }
    if flag_sel then
    begin
      SelectBuf.Sel_status := 1;
      SelectBuf.NxtPtr := ListPtr;
      move( SelectPtr^, DummyPtr^, Size_of_Select );
      ListPtr := DummyPtr;
      flag_sel := false
    end;
  until TempPtr = Nil;
end;

(* ######################################################### *)

Function choice_to_select : Char;
var
  ch_ch, LocChar                          : char;
  j_c                                     : longint;
  LocBuf                                  : Select;
  LocPtr                                  : PtrtoString;
  first_el                                : boolean;
begin
  j_c := 1;
  LocPtr := Ptr( Seg( LocBuf ), Ofs( LocBuf ));
  first_el := true;
  GotoXY( 1, 25 );
  write( no_of_ent,
  ' entities found ! Do you want to unselect anyone ( Y/N ) ?  ');
    repeat
    ch_ch := Upcase( ReadKey )
  until ch_ch in [ Escape, Return, 'Y', 'N' ];
  if ch_ch = Escape then
  begin
    choice_to_select := Escape;
    exit
```

```
end
else
begin
  if ch_ch = 'Y' then
  begin
    DummyPtr := ListPtr;
    repeat
      Comm_line(1, 25,' Do You want to unselect this entity ( Y or N ) ? ');
      Short_beep;
      move( DummyPtr^, LocPtr^, Size_of_select );
      move( LocBuf.EntPtr^, LocChar, 1 );
      case LocChar of
        'A' : comm_line(1, 1,'The entity selected is ****** Arc ********');
        'C' : comm_line(1, 1,'The entity selected is ****** Circle ********');
        'D' : comm_line(1, 1,'The entity selected is ****** Dot ********');
        'T' : comm_line(1, 1,'The entity selected is ****** Text ********');
        'L' : comm_line(1, 1,'The entity selected is ****** Line ********');
      end;
      Case LocChar of
        'D' : begin                      { of DOT }
                move( LocBuf.EntPtr^, DotPtr^, Size_Of_DotBuf );
                repeat
                  repeat
                    SetColorBlack;
                    DrawPoint( DotBuf.X, DotBuf.Y );
                    SetColorWhite;
                    DrawPoint( DotBuf.X, DotBuf.Y );
                  until KeyPressed;
                  ch_ch := Upcase( ReadKey );
                  ClearInkeyBuffer;
                until ch_ch in [Escape, Return, 'Y', 'N'];
              end;
        'L' : begin                      { of LINE }
                move( LocBuf.EntPtr^, LinePtr^, Size_Of_LineBuf );
                repeat
                  repeat
                    SetColorBlack;
                    DrawLine( LineBuf.X1, LineBuf.Y1, LineBuf.X2, LineBuf.Y2 );
                    SetColorWhite;
                    DrawLine( LineBuf.X1, LineBuf.Y1, LineBuf.X2, LineBuf.Y2 )
                  until KeyPressed;
                  ch_ch := Upcase( ReadKey );
                  ClearInkeyBuffer;
                until ch_ch in [Escape, Return, 'Y', 'N'];
              end;
        'C' : begin                      { of Circle }
                move( LocBuf.EntPtr^, CirPtr^, Size_Of_CirBuf );
                repeat
                  repeat
                    SetColorBlack;
                    DrawCircleDirect( WindowX( CirBuf.Xc ), Window

                                 WindowX(CirBuf.Xc + CirBuf.R) - WindowX(CirBuf.Xc),
                                 true );
                    SetColorWhite;
                    DrawCircleDirect( WindowX( CirBuf.Xc ), WindowY( CirBuf.Yc ),
                                 WindowX(CirBuf.Xc + CirBuf.R) - WindowX(CirBuf.Xc),
                                 true )
                  until KeyPressed;
                  ch_ch := Upcase( ReadKey );
                  ClearInkeyBuffer;
```

```
                 until ch_ch in [Escape, Return, 'Y', 'N'];
               end;
         'A' : begin                           { of Arc }
                 move( LocBuf.EntPtr^, ArcPtr^, Size_Of_ArcBuf );
                 repeat
                   repeat
                     SetColorBlack;
                     Draw_csr_rtn( ArcBuf.X1, ArcBuf.Y1, 2 );
                     SetColorWhite;
                     Draw_csr_rtn( ArcBuf.X1, ArcBuf.Y1, 2 );
                   until KeyPressed;
                   ch_ch := Upcase( ReadKey );
                   ClearInkeyBuffer;
                 until ch_ch in [Escape, Return, 'Y', 'N'];
               end;
         'T' : begin                           { of Text }
                 move( LocBuf.EntPtr^, TxtPtr^, Size_Of_TxtBuf );
                 move( TxtBuf.StrPtr^, TxtstrPtr^, TxtBuf.length );
                 repeat
                   repeat
                     SetColorBlack;
                     DrawSquare( TxtBuf.x, TxtBuf.y, txtBuf.x + ( 7 * TxtBuf.Length * TxtBuf.size ) ,
                                 TxtBuf.y - ( 9 * TxtBuf.size ), false );
                     SetColorWhite;
                     DrawSquare( TxtBuf.x, TxtBuf.y, txtBuf.x + ( 7 * TxtBuf.Length * TxtBuf.size ),
                                 TxtBuf.y - ( 9 * TxtBuf.size ), false )
                   until KeyPressed;
                   ch_ch := Upcase( ReadKey );
                   ClearInkeyBuffer;
                 until ch_ch in [Escape, Return, 'Y', 'N'];
               end;
         'W' : begin                           { of Area of layout }
                 move( LocBuf.EntPtr^, LinePtr^, Size_Of_LineBuf );
                 repeat
                   repeat
                     SetColorBlack;
                     DrawSquare( LineBuf.X1, LineBuf.Y1, LineBuf.X2, LineBuf.Y2,
                               False );
                     SetColorWhite;
                     DrawSquare( LineBuf.X1, LineBuf.Y1, LineBuf.X2, LineBuf.Y2,
                               False )
                   until KeyPressed;
                   ch_ch := Upcase( ReadKey );
                   ClearInkeyBuffer;
                 until ch_ch in [Escape, Return, 'Y', 'N'];
               end
        end;                                   { of               if ch_ch = '
then
          LocBuf.Sel_status := 0;
        move( LocPtr^, DummyPtr^, Size_of_select );
        DummyPtr := LocBuf.NxtPtr;

      until ( ch_ch in [ Escape, Return ] ) or (  DummyPtr = nil );
    end;
    if ch_ch = Escape then
      choice_to_select := Escape
    else
      choice_to_select := return
  end;
end;

(* ##################################################### *)
Procedure Dist_rtn;
var
  TempStyle                             : byte;
  ch_D                                  : char;
  Dist_flag                            : boolean;
  X_d, Y_d                             : real;

  procedure disp_dist( len : real );
  begin
    GotoXY( 38, 1);
    write('Distance = ', len:5:1 );
    SelectScreen( 2 );

    GotoXY( 38, 1);
    write('Distance = ', len:5:1 );
    SelectScreen( 1 )
  end;

begin
  TempStyle := GetLineStyle;
  x_d := Csr_X_Glb;
  y_d := Csr_Y_Glb;
  SetLineStyle( Elastic_style );
  Dist_flag := true;
  repeat
    if Dist_flag then
    begin
      Draw_Csr_rtn( x_d, y_d, csr_size_Glb );
      Disp_cordinate( x_d, y_d );
      DrawLine( X_d, y_d, Csr_X_Glb, Csr_Y_Glb );
      Disp_Dist( Sqrt( Sqr( x_d - Csr_X_Glb ) + Sqr( y_d - Csr_Y_Glb )));
    end;
    Dist_flag := Choice_rtn( x_d, y_d, ch_D );
  until ch_D in [ Return, Escape ];
  SetLineStyle( TempStyle );
  Clear_Comm
end;

(* ####################################################### *)
Procedure move_cal( m_c : char ; x_m, y_m : real);
begin
  case m_c of
    'D' : begin
            DotBuf.X := DotBuf.X + x_m;
            DotBuf.Y := DotBuf.Y + y_m;
          end;
    'L' : begin
            LineBuf.X1 := LineBuf.X1 + x_m;
            LineBuf.Y1 := LineBuf.Y1 + y_m;
            LineBuf.X2 := LineBuf.X2 + x_m;
            LineBuf.Y2 := LineBuf.Y2 + y_m;
          end;
    'C' : begin
            CirBuf.Xc := CirBuf.XC + x_m;
            CirBuf.Yc := CirBuf.YC + y_m;
          end;
    'A' : begin
            ArcBuf.Xc := ArcBuf.Xc + x_m;
            ArcBuf.Yc := ArcBuf.Yc + y_m;
            ArcBuf.X1 := ArcBuf.X1 + x_m;
            ArcBuf.Y1 := ArcBuf.Y1 + y_m;
          end;
    'T' : begin
            TxtBuf.X := TxtBuf.X + x_m;
            TxtBuf.Y := txtBuf.Y + y_m;
          end;
  end;
end;

(* ####################################################### *)
Procedure move_entity( var x_m, y_m : real);
var
  ch_mo                                : char;
  LocBuf                               : Select;
```

```
begin
 DummyPtr := ListPtr;
 LocPtr := Ptr( Seg( LocBuf ), ofs( LocBuf ) );
 repeat
  move( DummyPtr^, LocPtr^, Size_of_select );
  if LocBuf.Sel_Status () 0 then
  begin
   move( LocBuf.EntPtr^, ch_mo, 1 );
   case ch_mo of
    'D' : begin
        move( LocBuf.EntPtr^, DotPtr^, Size_Of_DotBuf );
        move_cal( 'D', x_m, y_m );
        move( DotPtr^, LocBuf.EntPtr^, size_of_DotBuf )
       end;
    'L' : begin
        move( LocBuf.EntPtr^, LinePtr^, Size_Of_LineBuf );
        move_cal( 'L', x_m, y_m );
        move( LinePtr^, LocBuf.EntPtr^, size_of_LineBuf )
       end;
    'C' : begin
        move( LocBuf.EntPtr^, CirPtr^, Size_Of_CirBuf );
        move_cal( 'C', x_m, y_m );
        move( CirPtr^, LocBuf.EntPtr^, size_of_cirBuf )
       end;
    'A' : begin
        move( LocBuf.EntPtr^, ArcPtr^, Size_Of_ArcBuf );
        move_cal( 'A', x_m, y_m );
        move( ArcPtr^, LocBuf.EntPtr^, size_of_ArcBuf )
       end;
    'T' : begin
        move( LocBuf.EntPtr^, TxtPtr^, Size_Of_TxtBuf );
        move_cal( 'T', x_m, y_m );
        move( TxtPtr^, LocBuf.EntPtr^, size_of_TxtBuf )
       end;
    'W' : begin
        move( LocBuf.EntPtr^, LinePtr^, Size_Of_LineBuf );
        move_cal( 'L', x_m, y_m );
        move( LinePtr^, LocBuf.EntPtr^, size_of_LineBuf )
       end;
   end;
  end;
  DummyPtr := LocBuf.NxtPtr;
 until dummyPtr = Nil
end;

(* ########################################################## *)

Procedure Erase_entity;   { have to be modified }
var
 ch_mo                 : char;
 LocBuf                : Select;
 LocPtr                : PtrtoString;
begin
 DummyPtr := ListPtr;
 LocPtr := Ptr( Seg( LocBuf ), ofs( LocBuf ) );
 repeat
  move( DummyPtr^, LocPtr^, Size_of_select );
  if LocBuf.Sel_Status () 0 then
  begin
   move( LocBuf.EntPtr^, ch_mo, 1 );
   case ch_mo of
    'D' : begin
        DotBuf.D_status := 0;
        move( DotPtr^, LocBuf.EntPtr^, size_of_DotBuf )
       end;
    'L' : begin
        move( LocBuf.EntPtr^, LinePtr^, Size_Of_LineBuf );
        LineBuf.L_status := 0;
```

```
                move( LinePtr^, LocBuf.EntPtr^, size_of_LineBuf )
              end;
        'C' : begin
                move( LocBuf.EntPtr^, CirPtr^, Size_Of_CirBuf );
                CirBuf.C_status := 0;
                move( CirPtr^, LocBuf.EntPtr^, size_of_cirBuf )
              end;
        'A' : begin
                move( LocBuf.EntPtr^, ArcPtr^, Size_Of_ArcBuf );
                ArcBuf.A_status := 0;
                move( ArcPtr^, LocBuf.EntPtr^, size_of_ArcBuf )
              end;
        'T' : begin
                move( LocBuf.EntPtr^, TxtPtr^, Size_Of_TxtBuf );
                TxtBuf.T_status := 0;
                move( TxtPtr^, LocBuf.EntPtr^, size_of_TxtBuf )
              end;
        'W' : begin
                move( LocBuf.EntPtr^, LinePtr^, Size_Of_LineBuf );
                LineBuf.L_status := 0;
                move( LinePtr^, LocBuf.EntPtr^, size_of_LineBuf )
              end;
      end;
    end;
    DummyPtr := LocBuf.NxtPtr;
  until dummyPtr = Nil
end;
(* ##################################################### *)
Procedure Turn_cal( ch_t : char; XRef, YRef, Theta : real);
var
  Theta1, R                               : real;
begin
  case ch_t of
    'D' : begin
            Theta1 := Angle_rtn( XRef, YRef, DotBuf.X, DotBuf.Y );
            R := Sqrt( Sqr( XRef - DotBuf.X ) + Sqr( YRef - DotBuf.Y ));
            DotBuf.X := XRef + R * cos( (Theta + Theta1) * Pi / 180.0 );
            DotBuf.Y := YRef + R * sin( (Theta + Theta1) * Pi / 180.0 );
          end;
    'L' : begin
            Theta1 := Angle_rtn( XRef, YRef, LineBuf.X1, LineBuf.Y1 );
            R := Sqrt( Sqr( XRef - LineBuf.X1 ) + Sqr( YRef - LineBuf.Y1 ));
            LineBuf.X1 := XRef + R * cos( (Theta + Theta1) * Pi / 180.0 );
            LineBuf.Y1 := YRef + R * sin( (Theta + Theta1) * Pi / 180.0 );
            Theta1 := Angle_rtn( XRef, YRef, LineBuf.X2, LineBuf.Y2 );
            R := Sqrt( Sqr( XRef - LineBuf.X2 ) + Sqr( YRef - LineBuf.Y2 ));
            LineBuf.X2 := XRef + R * cos( (Theta + Theta1) * Pi / 180.0 );
            LineBuf.Y2 := YRef + R * sin( (Theta + Theta1) * Pi / 180.0 );
          end;
    'C' : begin
            Theta1 := Angle_rtn( XRef, YRef, CirBuf.Xc, CirBuf.Yc );
            R := Sqrt( Sqr( XRef - CirBuf.Xc ) + Sqr( YRef - CirBuf.Y
CirBuf.Xc := XRef + R * cos( (Theta + Theta1) * Pi / 180.0 );
            CirBuf.Yc := YRef + R * sin( (Theta + Theta1) * Pi / 180.0 );
          end;
    'A' : begin
            Theta1 := Angle_rtn( XRef, YRef, ArcBuf.Xc, ArcBuf.Yc );
            R := Sqrt( Sqr( XRef - ArcBuf.Xc ) + Sqr( YRef - ArcBuf.Yc));
            ArcBuf.Xc := XRef + R * cos( (Theta + Theta1) * Pi / 180.0 );
            ArcBuf.Yc := YRef + R * sin( (Theta + Theta1) * Pi / 180.0 );
            Theta1 := Angle_rtn( XRef, YRef, ArcBuf.X1, ArcBuf.Y1 );
            R := Sqrt( Sqr( XRef - ArcBuf.X1 ) + Sqr( YRef - ArcBuf.Y1 ));
            ArcBuf.X1 := XRef + R * cos( (Theta + Theta1) * Pi / 180.0 );
            ArcBuf.Y1 := YRef + R * sin( (Theta + Theta1) * Pi / 180.0 );
          end;
    'T' : begin
            Theta1 := Angle_rtn( XRef, YRef, TxtBuf.X, TxtBuf.Y );
            R := Sqrt( Sqr( XRef - TxtBuf.X ) + Sqr( YRef - TxtBuf.Y));
            TxtBuf.X := XRef + R * cos( (Theta + Theta1) * Pi / 180.0 );
            TxtBuf.Y := YRef + R * sin( (Theta + Theta1) * Pi / 180.0 );
          end;
  end;
end;

(* ##################################################### *)

Procedure Turn_entity( XRef, YRef, Theta : real);
var
  ch_mo                                 : char;
  LocBuf                                : Select;
  LocPtr, LocPtr1                       : PtrtoString;
  Theta1, R                             : real;
begin
  DummyPtr := ListPtr;
  LocPtr := Ptr( Seg( LocBuf ), ofs( LocBuf ) );
  repeat
    move( DummyPtr^, LocPtr^, Size_of_select );
    if LocBuf.Sel_Status <> 0 then
    begin
      move( LocBuf.EntPtr^, ch_mo, 1 );
      case ch_mo of
        'D' : begin
                move( LocBuf.EntPtr^, DotPtr^, Size_Of_DotBuf );
                Turn_cal( 'D', XRef, YRef, Theta );
                move( DotPtr^, LocBuf.EntPtr^, size_of_DotBuf )
              end;
        'L' : begin
                move( LocBuf.EntPtr^, LinePtr^, Size_Of_LineBuf );
                Turn_cal( 'L', XRef, YRef, Theta );
                move( LinePtr^, LocBuf.EntPtr^, size_of_LineBuf )
              end;
        'C' : begin
                move( LocBuf.EntPtr^, CirPtr^, Size_Of_CirBuf );
                Turn_cal( 'C', XRef, YRef, Theta );
                move( CirPtr^, LocBuf.EntPtr^, size_of_cirBuf )
              end;
        'A' : begin
                move( LocBuf.EntPtr^, ArcPtr^, Size_Of_ArcBuf );
                Turn_cal( 'A', XRef, YRef, Theta );
                move( ArcPtr^, LocBuf.EntPtr^, size_of_ArcBuf )
              end;
        'T' : begin
                move( LocBuf.EntPtr^, TxtPtr^, Size_Of_TxtBuf );
                Turn_cal( 'T', XRef, YRef, Theta );
                move( TxtPtr^, LocBuf.EntPtr^, size_of_TxtBuf )
              end;
        'W' : begin
              end;
      end;
    end;
    DummyPtr := LocBuf.NxtPtr;
  until dummyPtr = Nil
end;

(* ##################################################### *)

Procedure Free_select;
var
  LocBuf                                : Select;
  LocPtr                                : PtrtoString;
begin
  LocPtr := Ptr( Seg( LocBuf ), ofs( LocBuf ) );
  DummyPtr := ListPtr;
  repeat
    move( DummyP                     FreeMem( DummyPtr, Size_of_Select );
    DummyPtr := LocBuf.NxtPtr;
  until DummyPtr = Nil
end;
```

```

(* ############################################################ *)

Procedure Move_rtn( choice : char );
var
  ch_m                                  : char;
  x1, y1, x2, y2, x_dis, y_dis, Theta   : real;
  TempStyle                             : word;
  Move_flag, Edited                     : boolean;
  Index                                 : byte;
  AngleStr                              : String[3];
  Result                                : integer;
begin
  No_of_ent := 0;
  TempStyle := GetLineStyle;
  AngleStr := '   ';
  x1 := Csr_X_Glb;
  y1 := Csr_Y_Glb;
  SetLineStyle( Elastic_style );
  comm_line(1, 25, ' Enter first point ');
  Short_beep;
  Move_Flag := true;
  repeat
    Move_flag := Choice_rtn( x1, y1, ch_m );
    if Move_flag then
    begin
      Draw_Csr_rtn( x1, y1, csr_size_Glb );
      Disp_cordinate( x1, y1 );
    end;
  until ch_m in [ Return, Escape ];
  Move_flag := true;
  x2 := x1;
  y2 := y1;
  if ch_m <> Escape then
  begin
    comm_line(1, 25, ' Enter the diagonal point ');
    Short_beep;
    repeat
      Move_flag := Choice_rtn( x2, y2, ch_m );
      if Move_flag then
      begin
        Draw_Csr_rtn( x2, y2, csr_size_Glb );
        Disp_cordinate( x2, y2 );
        Disp_relative( X2 - X1, Y2 - Y1 );
        DrawSquare( x1, y1, x2, y2, false )
      end;
    until ch_m in [ Return, Escape ];
    Clear_Comm;
    if ch_m <> Escape then
    begin
      if ( x1 <> x2 ) and ( y1 <> y2 ) then
      begin
        if x1 < x2 then
          if y1 < y2 then
            Selection_rtn( x1, y1, x2, y2, true )   { true to select z and s}
          else
            Selection_rtn( x1, y2, x2, y1, true )
        else
          if y1 < y2 then
            Selection_rtn( x2, y1, x1, y2, true )
          else
            Selection_rtn( x2, y2, x1, y1, true );
      end;
      if no_of_ent > 0 then
      begin
        if choice_to_select <> Escape then
        begin
          case choice of
          'M' : begin
                  comm_line(1, 25, ' Enter the reference point ');
            Short_beep;
            Move_flag := true;
            repeat
              Move_flag := Choice_rtn( x1, y1, ch_m );
              if Move_flag then
              begin
                Draw_Csr_rtn( x1, y1, csr_size_Glb );
                Disp_cordinate( x1, y1 );
              end;
            until ch_m in [ Return, Escape ];
            Move_flag := true;
            x2 := x1;
            y2 := y1;
            if ch_m <> Escape then
            begin
              comm_line(1, 25,' Enter the displacement point ');
              Short_beep;
              repeat
                Move_flag := Choice_rtn( x2, y2, ch_m );
                if Move_flag then
                begin
                  Draw_Csr_rtn( x2, y2, csr_size_Glb );
                  Disp_cordinate( x2, y2 );
                  Disp_relative( X2 - X1, Y2 - Y1 );
                  DrawLine( x1, y1, x2, y2 )
                end;
              until ch_m in [ Return, Escape ];
              if ch_m <> Escape then
              begin
                x_dis := x2 - x1;
                y_dis := y2 - y1;
                move_entity( x_dis, y_dis );
                ClearScreen;
                SetLineStyle( 0 );
                DrawBorder;
                Draw_List( true );
                CopyScreen;
                Free_select
              end
              else
              begin

                Free_select
              end
            end
            else
            begin

              Free_select
            end
          end;  { end of 'm '}
    'N'  : begin
            comm_line(1, 25,' Enter the reference point ');
            Short_beep;
            Move_flag := true;
            repeat
              Move_flag := Choice_rtn( x1, y1, ch_m );
              if Move_flag then
              begin
                Draw_Csr_rtn( x1, y1, csr_size_Glb );
                Disp_cordinate( x1, y1 );
              end;
            until ch_m in [ Return,                           Move_flag :=

            if ch_m <> Escape then
            begin
              GotoXY( 1, 25 );
              ClrEol;
              write( ' Enter value for Angle ( AntiClockWise ) : ');
              Short_beep;
```

```
                  Edited := false;
                  Index := 1;
                  GotoXY( 45, 25 );
                  write( AngleStr );
                  GotoXY( 45, 25 );
                  ch_m := Edit_field( AngleStr, 0, 3, false, index, 45, 25, Ed
ed, 0 );
                  SelectWorld( 1 );
                  SelectWindow( 1 );
                  if ch_m <> Escape then
                  begin
                    val( AngleStr, Theta, Result );
                    Turn_entity( x1, y1, Theta );
                    ClearScreen;
                    SetLineStyle( 0 );
                    DrawBorder;
                    Draw_List( true );
                    CopyScreen;
                    Free_select
                  end
                  else
                    Free_select;
                end
                else
                  Free_select;
              end;  { end  of 'N' }
        'E' : begin
                Erase_entity;
                ClearScreen;
                SetLineStyle( 0 );
                DrawBorder;
                Draw_List( true );
                CopyScreen;
                Free_select
              end;
        end;   { end of case  }
      end;
    end;
  end;
end;
Swap_screen_rtn;
SetLineStyle( tempStyle );
comm_line(1, 25,'                                                       ');
comm_line(1, 1,'                                                        ');
d;

 ######################################################## *)

ocedure Select_Lines( x1_s, y1_s, x2_s, y2_s : real );
r
f_sel                                 : file;
ch_s                                  : char;
flag_sel                              : boolean;
Loc_x, Loc_y                          : real;
gin
DummyPtr := Nil;
ListPtr := Nil;
TempPtr :=  HeadPtr;
no_of_ent := 0;
repeat
  move( TempPtr^, ch_s, 1 );
  case ch_s of
    'W' : begin
            move( TempPtr^, LinePtr^, Size_of_LineBuf );
            TempPtr := LineBuf.NxtPtr
          end;
    'L' : begin
            move( TempPtr^, LinePtr^, Size_of_LineBuf );
            if LineBuf.L_Status <> 0 then
            b
```

```
                flag_sel := Select_entry( 'L', x1_s, y1_s, x2_s, y2_s );
                if flag_sel then
```

```
                  begin
                    SelectBuf.EntPtr := TempPtr;
                    GetMem_Rtn( DummyPtr, Size_of_Select );
                    inc( no_of_ent );
                    SetColorBlack;
                    DrawLine( LineBuf.x1, LineBuf.y1, LineBuf.x2, LineBuf.y2 );
                    SetColorWhite;
                    DrawLine( LineBuf.x1, LineBuf.y1, LineBuf.x2, LineBuf.y2 )
                  end;
              end;
              TempPtr := LineBuf.NxtPtr
            end;
      'D' : begin
              move( TempPtr^, DotPtr^, Size_of_DotBuf );
              TempPtr := DotBuf.NxtPtr
            end;
      'C' : begin
              move( TempPtr^, CirPtr^, Size_of_CirBuf );
              TempPtr := CirBuf.NxtPtr
            end;
      'A' : begin
              move( TempPtr^, ArcPtr^, Size_of_ArcBuf );
              TempPtr := ArcBuf.NxtPtr
            end;
      'T' : begin
              move( TempPtr^, TxtPtr^, Size_of_TxtBuf );
              TempPtr := TxtBuf.NxtPtr
            end
    end;  { end of case }
    if flag_sel then
    begin
      SelectBuf.Sel_status := 0;
      SelectBuf.NxtPtr := ListPtr;
      move( SelectPtr^, DummyPtr^, Size_of_Select );
      ListPtr := DummyPtr;
      flag_sel := false
    end;
  until TempPtr = Nil;
end;


(* ####################################################### *)
Function Select_per_line: Char;
var
  ch_ch, LocChar                        : char;
  j_c                                   : longint;
  LocBuf                                : Select;
  LocPtr                                : PtrtoString;
  first_el                              : boolean;
begin
  j_c := 1;
  LocPtr := Ptr( Seg( LocBuf ), Ofs( LocBuf ));
  first_el := true;
  GotoXY( 1, 25 );
  write( no_of_ent,' Lines found ! Select perticular Line to be cut ');
  Short_beep;
  delay( 300 );
  DummyPtr := ListPtr;
  repeat
        GotoXY( 1, 25 );
        ClrEol;
        write(' Is this the line to be cut ( Y or N ) ? ');
        Short_beep;
        move( DummyPtr^, LocPtr^, Size_of_select );
        move( LocBuf.EntPtr^, LinePtr^, Size_Of_LineBuf );
        repeat
          repeat
            SetColorBlack;
            DrawLine( LineBuf.X1, LineBuf.Y1, LineBuf.X2, LineBuf.Y2 );
                        DrawLine( LineBuf.X1, LineBuf.Y1, LineBuf.X2, LineBuf.Y2 )
          until KeyPressed;
          ch_ch := Upcase( ReadKey );
          ClearInkeyBuffer;
        until ch_ch in [Escape, Return, 'Y', 'N'];
        if ch_ch = 'Y' then
          LocBuf.Sel_status := 1;
        move( LocPtr^, DummyPtr^, Size_of_select );
        DummyPtr := LocBuf.NxtPtr;
  until ( ch_ch in [ Escape, 'Y' ] ) or (  DummyPtr = nil );
  if ch_ch = 'Y' then
    Select_per_line := return
  else
  if ( ch_ch = Escape ) or ( DummyPtr = Nil ) then
    Select_per_line := Escape
end;

(* ####################################################### *)
Function CutLineValue( loc_x, loc_y : real; var x, y : real ): boolean;
var
  R, ThetaL, ThetaPt                         : real;
begin
  CutLineValue := true;
  R := Sqrt( sqr( LineBuf.X1- Loc_x ) + sqr( LineBuf.Y1- loc_y ) );
  ThetaL := Angle_rtn( LineBuf.X1, LineBuf.y1, LineBuf.x2, LineBuf.y2 );
  ThetaPt := Angle_rtn( LineBuf.X1, LineBuf.y1, loc_x, loc_y );
  x := LineBuf.x1 + ( R * cos( (ThetaPt - ThetaL)* Pi / 180  ) * cos( ThetaL * P
i / 180 ) );
  y := LineBuf.y1 + ( R * cos( (ThetaPt - ThetaL)* Pi / 180  ) * sin( ThetaL * P
i / 180) );
  if LineBuf.Y1 < LineBuf.Y2 then
  begin
    if ( x < LineBuf.X1 ) or ( x > LineBuf.X2 ) or ( y > LineBuf.y2 ) or
       ( y < LineBuf.y1 ) then
       CutLineValue := false;
  end
  else
    if ( x < LineBuf.X1 ) or ( x > LineBuf.X2 ) or ( y < LineBuf.y2 ) or
       ( y > LineBuf.y1 ) then
       CutLineValue := false;
end;

(* ####################################################### *)
Function Cut_Line( xc1, yc1, xc2, yc2 : real; style : byte):boolean;
var
  LocBuf                                : Select;
  LocPtr                                : PtrtoString;
  Temp1, Temp2, x1, y1, x2, y2          : real;
begin
  Cut_Line := true;
  LocPtr := Ptr( Seg( LocBuf ), Ofs( LocBuf ));
  DummyPtr := ListPtr;
  repeat
    move( DummyPtr^, LocPtr^, Size_of_select);
    if LocBuf.Sel_Status = 0 then
      DummyPtr := LocBuf.NxtPtr
    else
    begin
      move( LocBuf.EntPtr^, LinePtr^, Size_of_LineBuf );
      if CutLineValue( xc1, yc1, x1, y1 ) then
      begin
        if CutLineValue( xc2, yc2, x2, y2 ) then
               temp1 := x1;
          temp2 := y1;
          if x1 > x2 then
          begin
            x1 := x2;
          y1 := y2;
          x2 := temp1;
          y2 := temp2;
        end
        else
        if x1 = x2 then
        begin
          if y1 ) y2 then
          begin
            x1 := x2;
            y1 := y2;
            x2 := temp1;
            y2 := temp2;
          end;
        end;
        if x1 = LineBuf.X1 then
        begin
          if y1 = LineBuf.y1 then
          begin
            LineBuf.x1 := x2;
            LineBuf.y1 := y2;
            move( LinePtr^, LocBuf.EntPtr^, size_of_LineBuf )
          end
          else
          begin
            if y1 ) y2 then
            begin
              temp1 := x1;
              temp2 := y1;
              y1 := y2;
              x1 := x2;
              y2 := temp2;
              x2 := temp1;
            end;
            temp1 := LineBuf.x2;
            temp2 := LineBuf.y2;
            LineBuf.x2 := x1;
            LineBuf.y2 := y1;
            move( LinePtr^, LocBuf.EntPtr^, size_of_LineBuf );
            GetMem_rtn( TempPtr, Size_of_LineBuf );
            LineBuf.x1 := x2;
            LineBuf.y1 := y2;
            LineBuf.x2 := Temp1;
            LineBuf.y2 := Temp2;
            LineBuf.NxtPtr := HeadPtr;
            LineBuf.Typ := 'L';
            LineBuf.L_status := 1;
            move( LinePtr^, TempPtr^, size_of_LineBuf );
            HeadPtr := TempPtr;
          end
        end
        else
        if x2 = LineBuf.X2 then
        begin
          if y2 = LineBuf.Y2 then
          begin
            LineBuf.x2 := x1;
            LineBuf.y2 := y1;
            move( LinePtr^, LocBuf.EntPtr^, size_of_LineBuf )
          end
        end
        else
        begin
          Temp1 := LineBuf.x2;
          Temp2 := LineBuf.y2;
          LineBuf.x2 := x1;
          LineBuf.y2 := y1;
          move( LinePtr^, LocBuf.EntPtr^, size_of_LineBuf );
          GetMem_rtn( TempPtr, Size_of_LineBuf );
          LineBuf.x1 := x2;
```

```
            LineBuf.y1 := y2;
            LineBuf.x2 := Temp1;
                      LineBuf.NxtFtr := HeadFtr;
            LineBuf.Typ := 'L';
            LineBuf.L_status := 1;
            move( LinePtr^, TempFtr^, size_of_LineBuf );
            HeadFtr := TempFtr;
          end
        end
        else
          Cut_line := false;
      end
      else
        Cut_line := false;
    end;
  until ( DummyPtr = Nil ) or ( LocBuf.Sel_Status = 1 );
end;

(* ##################################################### *)
Procedure Cut_rtn;
var
  ch_c                                  : char;
  x1, y1, x2, y2                        : real;
  TempStyle                             : word;
  Cut_flag, Edited                      : boolean;
  Index                                 : byte;
  Result                                : integer;
begin
  TempStyle := GetLineStyle;
  x1 := Csr_X_Glb;
  y1 := Csr_Y_Glb;
  SetLineStyle( Elastic_style );
  comm_line(1, 25,' Enter first point ');
  Short_beep;
  Cut_flag := true;
  repeat
    Cut_flag := Choice_rtn( x1, y1, ch_c );
    if Cut_flag then
    begin
      Draw_Csr_rtn( x1, y1, csr_size_Glb );
      Disp_cordinate( x1, y1 );
    end;
  until ch_c in [ Return, Escape ];
  Cut_flag := true;
  x2 := x1;
  y2 := y1;
  if ch_c <> Escape then
  begin
    comm_line(1, 25,' Enter the diagonal point ');
    Short_beep;
    repeat
      cut_flag := Choice_rtn( x2, y2, ch_c );
      if cut_flag then
      begin
        Draw_Csr_rtn( x2, y2, csr_size_Glb );
        Disp_cordinate( x2, y2 );
        Disp_relative( X2 - X1, Y2 - Y1 );
        DrawSquare( x1, y1, x2, y2, false )
      end;
    until ch_c in [ Return, Escape ];
    Clear_Comm;
    if ch_c <> Escape then
    begin
      if ( x1 <> x2 ) and ( y1 <> y2 ) then
      begin
        if x1 < x2 then
          if y1 < y2 then
            Select_Lines( x1, y1, x2, y2 )
          else
          Select_Lines( x1, y2, x2, y1 )
      else
        if y1 ( y2 then
          Select_Lines( x2, y1, x1, y2 )
        else
          Select_Lines( x2, y2, x1, y1 );
        495        if no_of_ent ) 0 then
    begin
      ch_c := Select_per_line;
      if ch_c () Escape then
      begin
        comm_line(1, 25,' Enter the first cut point ');
        Short_beep;
        Cut_flag := true;
        repeat
          Cut_flag := Choice_rtn( x1, y1, ch_c );
          if Cut_flag then
          begin
            Draw_Csr_rtn( x1, y1, csr_size_Glb );
            Disp_cordinate( x1, y1 );
          end;
        until ch_c in [ Return, Escape ];
        Cut_flag := true;
        x2 := x1;
        y2 := y1;
        if ch_c () Escape then
        begin
          comm_line(1, 25,' Enter the second cut point ');
          Short_beep;
          repeat
            cut_flag := Choice_rtn( x2, y2, ch_c );
            if cut_flag then
            begin
              Draw_Csr_rtn( x2, y2, csr_size_Glb );
              Disp_cordinate( x2, y2 );
              Disp_relative( X2 - X1, Y2 - Y1 )
            end;
          until ch_c in [ Return, Escape ];
          Clear_Comm;
          if ch_c () Escape then
          begin
            if Cut_Line( x1, y1, x2, y2, Tempstyle ) then
            begin
              ClearScreen;
              SetLineStyle( 0 );
              DrawBorder;
              Draw_List( true );
              CopyScreen;
              Free_select
            end
            else
            begin
              Comm_line(1, 25, 'invalid  cut point      ');
              Short_beep;
              Free_select;
            end
          end
          else
            Free_select;
        end
        else
          free_Select;
      end
      else
        Free_select;
    end;
  end;
end;
Swap_screen_rtn;
```

```
  SetLineStyle( tempStyle );
  comm_line(1, 25,'                                             ');
end;

(* ##################          Procedure Style_rtn;
var
  TempStyle, Loc_Style                  : word;
  ch_s                                  : char;
begin
  Draw_csr_rtn( Csr_X_Glb, Csr_Y_Glb, 0 );
  CopyScreen;
  TempStyle := GetLineStyle;
  SetWindowModeOff;
  DefineWindow( 2, 40, 0, XMaxGlb, 125 );
  DefineHeader( 2, 'LINE   STYLES');
  SelectWorld( 1 );
  SelectWindow( 2 );
  SetHeaderOn;
  SetLineStyle( 0 );
  DrawBorder;
  SetBackGround( 255 );
  SetColorBlack;
  SetLineStyle( 0 );
  DrawLine( 380, 20, 620, 20 );
  SetLineStyle( 3 );
  DrawLine( 380, 40, 620, 40 );
  SetLineStyle( 1 );
  DrawLine( 380, 60, 620, 60 );
  SetLineStyle( 4 );
  DrawLine( 380, 80, 620, 80 );
  SetLineStyle( 2 );
  DrawLine( 380, 100, 620, 100 );
  GotoXY( 42, 15 );
  Write( ' Use Keys :            ');
  GotoXY( 54, 15 );
  DC( 24 );
  GotoXY( 56, 15 );
  DC( 25 );
  GotoXY( 58, 15 );
  DC( 27 );
  GotoXY( 59, 15 );
  DC( 45 );
  SetColorWhite;
  Drawline( 470, 115, 470, 110 );
  GotoXY( 62, 15 );
  write( 'and Esc ');
  SetLineStyle( 0 );
  Loc_Style := TempStyle;
  case Loc_Style of
    0  : Loc_style  := 0;
    3  : Loc_style  := 1;
    1  : Loc_style  := 2;
    4  : Loc_style  := 3;
    2  : Loc_style  := 4
  end;
  repeat
    case Loc_Style of
      0 : Draw_Square_rtn( 360, 15, 375, 25 );
      1 : Draw_Square_rtn( 360, 35, 375, 45 );
      2 : Draw_Square_rtn( 360, 55, 375, 65 );
      3 : Draw_Square_rtn( 360, 75, 375, 85 );
      4 : Draw_Square_rtn( 360, 95, 375, 105 )
    end;
    ch_s := ReadKey;
    if ch_s = #0 then
      ch_s := ReadKey;
    case ch_s of
      Up    : if Loc_Style = 0 then
               Loc_Style := 4
```

```
             else
               dec( Loc_Style );
      Down : begin
               inc( Loc_style );
               if Loc_Style > 4 then
                 Loc_Style := 0
             end
    end
  until ch_s in [ Return, Escape ];
  SetColorWhite;
  SetBackGround( 0 );
      640    SelectWorld( 1 );
  SelectWindow( 1 );
  Swap_Screen_rtn;
  Draw_Csr_rtn( Csr_X_Glb, Csr_Y_Glb, Csr_Size_Glb );
  if ch_s <> Escape then
  begin
    case Loc_Style of
      0  : SetLineStyle( 0 );
      1  : SetLineStyle( 3 );
      2  : SetLineStyle( 1 );
      3  : SetLineStyle( 4 );
      4  : SetLineStyle( 2 )
    end
  end
  else
    SetLineStyle( TempStyle );
end;

(* ##################################################### *)

Procedure Drag_rtn;
var
  x1_d, y1_d, x2_d, y2_d, x_dif, y_dif,
  wld_x, wld_y                            : real;
  drag_flag                               : boolean;
  ch_D                                    : char;
  TempStyle                               : byte;
begin
  x1_d := Csr_X_Glb;
  y1_d := Csr_Y_Glb;
  TempStyle := GetLineStyle;
  SetLineStyle( Elastic_Style );
  comm_line(1, 25,'Enter the reference point : ');
  Short_beep;
  Drag_Flag := true;
  repeat
    Drag_flag := choice_rtn( x1_d, y1_d, ch_D );
    if drag_flag then
    begin
      Draw_Csr_rtn( x1_d, y1_d, Csr_Size_Glb );
      Disp_cordinate( x1_d, y1_d )
    end
  until ch_D in [ Escape, Return ];
  if ch_D = Return then
  begin
    x2_d := x1_d;
    y2_d := y1_d;
    comm_line(1, 25,'Use arrow keys to show the displacement : ');
    Short_beep;
    Drag_Flag := true;
    repeat
      Drag_flag := choice_rtn( x2_d, y2_d, ch_D );
      if drag_flag then
      begin
        Draw_Csr_rtn( x2_d, y2_d, Csr_Size_Glb );
        Disp_cordinate( x2_d, y2_d );
        DrawLine( x1_d, y1_d, x2_d, y2_d );
        Disp_relative( X2_d - X1_d, Y2_d - Y1_d );
      end
    until ch_D in [ Escape, Return ];
    if ch_D <> Escape then
    begin
      x_dif := x1_d - Active_World_Glb[ 0 ];
      y_dif := y1_d - Active_World_Glb[ 1 ];
      wld_x := Active_World_Glb[ 2 ] - Active_World_G                 wld_y := Active_W
orld_Glb[ 3 ] - Active_World_Glb[ 1 ] ;
      Active_World_Glb[ 0 ] := x2_d - x_dif;
      Active_World_Glb[ 1 ] := y2_d - y_dif;
      Active_World_Glb[ 2 ] := Active_World_Glb[ 0 ] + wld_x;
      Active_World_Glb[ 3 ] := Active_World_Glb[ 1 ] + wld_y;
      if Active_World_Glb[ 0 ] < World_Limit_Glb[ 0 ] then
      begin
        Active_world_Glb[ 0 ] := World_Limit_Glb[ 0 ];
        Active_world_Glb[ 2 ] := World_Limit_Glb[ 0 ] + wld_x
      end
      else
      if Active_World_Glb[ 2 ] > World_Limit_Glb[ 2 ] then
      begin
        Active_world_Glb[ 2 ] := World_Limit_Glb[ 2 ];
        Active_world_Glb[ 0 ] := World_Limit_Glb[ 2 ] - wld_x
      end;
      if Active_World_Glb[ 1 ] < World_Limit_Glb[ 1 ] then
      begin
        Active_world_Glb[ 1 ] := World_Limit_Glb[ 1 ];
        Active_world_Glb[ 3 ] := World_Limit_Glb[ 1 ] + wld_y
      end
      else
      if Active_World_Glb[ 3 ] > World_Limit_Glb[ 3 ] then
      begin
        Active_world_Glb[ 3 ] := World_Limit_Glb[ 3 ];
        Active_world_Glb[ 1 ] := World_Limit_Glb[ 3 ] - wld_y
      end;
      DefineWorld( 1, Active_world_Glb[ 0 ], Active_world_Glb[ 3 ],
                      Active_world_Glb[ 2 ], Active_world_Glb[ 1 ] );
      ClearScreen;
      SetLineStyle( 0 );
      SelectWorld( 1 );
      SelectWindow( 1 );
      DrawBorder;
      GotoXY( 1, 25 );
      ClrEol;
      write(' Please wait ! computing ...... ');
      draw_list( true );
      GotoXY( 1, 25 );
      ClrEol;
      CopyScreen;
      Csr_X_Glb := x2_d;
      Csr_Y_Glb := y2_d
    end
  end;
  comm_line(1, 25,'                                                 ');
  Clear_comm;
  SetLineStyle( TempStyle )
end;

(* ###################################################### *)

Procedure save_choice_rtn;
var
  ch_choice                           : char;
begin
  GotoXY( 1, 25 );
  ClrEol;
  Write( ' Do you want to save changes ( Y or N ) ? ');
  Short_beep;
  repeat
  if ch_choice = 'Y' then
  begin
    GotoXY( 1, 25 );
    ClrEol;
    Write( ' wait ! saving the data .... ');
    Short_beep;
    save_list
  end
end;

(* ###################################################### *)
 Procedure Edit_rtn;
 var
   Draw_Csr_Flag, Quit_flag             : boolean;
 begin
   Draw_Csr_Flag := true;
   Quit_Flag  := False;
   Repeat
     if Draw_csr_flag then
     begin
       Draw_csr_rtn( Csr_X_Glb, Csr_Y_Glb, Csr_Size_Glb );
       Disp_cordinate( Csr_X_Glb, Csr_Y_Glb )
     end;
     Draw_Csr_Flag := true;
     ch := upcase( ReadKey );
     case ch of
       #0    : begin
                 ch := ReadKey;
                 Case ch of
                   Front : Cursor_pos_rtn( Csr_X_Glb, Csr_Y_Glb, 'R' );
                   Up    : Cursor_pos_rtn( Csr_X_Glb, Csr_Y_Glb, 'U' );
                   Back  : Cursor_pos_rtn( Csr_X_Glb, Csr_Y_Glb, 'L' );
                   Down  : Cursor_pos_rtn( Csr_X_Glb, Csr_Y_Glb, 'D' );
                   PgUp  : begin
                             Pg_rtn( 'U' );
                             Draw_Csr_Flag := false;
                           end;
                   PgDn  : begin
                             Pg_rtn( 'D' );
                             Draw_Csr_Flag := false;
                           end;
                   F1    : Help_rtn;
                   F5    : HardCopy_rtn;
                   F0    : Save_choice_rtn;
                   else            { else of case }
                     Draw_Csr_Flag := false;
                 end;
               end;
       Plus  : Plus_rtn;
       'D'   : Dist_rtn;
       DotKey: Dot_rtn;
       'L'   : Line_rtn;
       'R'   : Rect_rtn;
       'C'   : Circ_rtn;
       'A'   : Arc_rtn;
       'S'   : Style_rtn;
       'T'   : Text_rtn;
       'G'   : Drag_rtn;
       'Z'   : Zoom_rtn;
       'M'   : Move_rtn( 'M' );
       'E'   : Move_rtn( 'E' );       { erase_rtn }
       'N'   : Move_rtn( 'N' );       { turn_rtn  }
       'B'   : Cut_rtn;
       'Q'   : Quit_flag := Quit_rtn;
     else              { else of case }
       Draw_Csr_Flag := false;
     end;
   until Quit_flag
 end;

```

```
{ This procedure gives the help routine for the design and layout editor.
  All the commands & cursor facilities }

procedure help_rtn;
const
  first_page = 1;
  last_page = 4;
var
  screen_no                          : integer;


procedure page1;
begin
  ClearScreen;
  GoToXY( 2, 3 );
  writeln('                                  COMPUTER AIDED DRAFTER  ' );
  writeln('                              ----------------------------          ' );
  writeln( ' PAGE 1' );
  writeln( ' -------' );
  writeln;
  writeln( ' ARC         :   Move cursor to the centre of the circle from which arc is to '
 );
  writeln( '               drawn. Press "A". Use arrow  keys  to draw  dummy circles of '
 );
  writeln( '               different radii.  Press (Enter)  at  the first point of arc. '
 );
  writeln( '               Use arrow keys to  move  to  the  second point of arc. Press '
 );
  writeln( '               (Enter)' );
  writeln;
  writeln( ' BREAK LINE :  Press "B". Specify the block containing the line  to be cut. '
 );
  writeln( '               Mark the first and second cut points on the line. ' );
  writeln;
  writeln( ' CURSOR      :  "PgUp" increases cursor  speed. "PgDn"  decreases the speed. '
 );
  writeln( ' MOVEMENTS      "+" increases cursor size. Arrow keys can be used to move it '
 );
  writeln( '               in all the directions.' );
  writeln;
  writeln( ' CIRCLE      :  Move the cursor to the centre of the circle. Press "C". Move '
);
  writeln( '               the cursor to see circles of different radius. Press (Enter) '
);
  writeln( '               to have the circle of desired radius.' );
  writeln;
  DrawBorder;
end;

procedure page2;
begin
  ClearScreen;
  GoToXY( 2, 3 );
  writeln( ' PAGE 2' );
  writeln( ' -------' );
  writeln;writeln;
  writeln( ' DISTANCE    :  To find the distance between tow  points, press  "D" at  the
);
  writeln( '               first point and move the cursor to the second point. ' );
  writeln;
  writeln( ' DOT         :  Press "." and move the cursor. Dot will  get  drawn wherever
);
  writeln( '               you press  "."' );
  writeln;
  writeln( ' DRAG        :  Press "G". Enter the reference point. Enter the displacement
);
  writeln( '               point in any direction.' );
  writeln;
  writeln( ' ERASE       :  Press "E". Enclose the entities in  a  dummy rectangle using
);
```

```
  writeln( '                      arrow keys.Objects that are not to be erased can be selected '
);
  writeln( '                      one by one ');
  writeln;
  writeln( ' HARDCOPY    :   Press  "F5".  Enter  the  print  scale  factor.  Follow  the '
);
  writeln( '                      instructions as and when they appear on the screen. ');
  writeln;
  writeln( ' HELP        :   Press "F1" to see the description of various commands.');
  writeln;
  DrawBorder;
end;

procedure page3;
begin
  ClearScreen;
  GoToXY( 2, 3 );
  writeln( ' PAGE 3' );
  writeln( ' -------' );
  writeln;writeln;
  writeln( ' LINE        :   Press "L" at  the  first  end  point of the line.  Move  the '
);
  writeln( '                      cursor to the second end point of the line. Press "Enter" to '
);
  writeln( '                      end the command.' );
  writeln;
  writeln( ' LINE STYLE :   Press "S". Choose the desired line style. Press "Enter". ' );
  writeln;writeln;
  writeln( ' MOVE        :   Press "M". Select  the  entities  in  a  window. Enter  the '
);
  writeln( '                      reference point and the displacement point to move entities.'
);
  writeln;
  writeln( ' RECTANGLE   :   Press "R" at the first  corner. Enter the other point of the '
);
  writeln( '                      same diagonal.' );
  writeln;writeln;
  writeln( ' SAVE        :   Press "F10" to save the drawn layout. This option overwrites '
);
  writeln( '                      the previously saved layout. ');
  writeln;
  DrawBorder;
end;

  procedure page4;
  begin
    ClearScreen;
    GoToXY( 2, 3 );
    writeln( ' PAGE 4' );
    writeln( ' -------' );
    writeln;writeln;
    writeln( ' TEXT        :   Press "T". Enter the text string. Press "S" to  give size o
 ');
    writeln( '                      the text. The area which is going to be occupied by the tex
 ');
    writeln( '                      is  shown  by  a dummy  rectangle. Place  it at  the  prope
 ');
    writeln( '                      location and press "Enter".' );
    writeln;writeln;
    writeln( ' TURN        :   Press "N". Select  the  entities  in  a  window.  Enter  th
 ');
    writeln( '                      reference point and angle  by  which they have to be rotate
 ');
    writeln( '                      in anti-clokwise direction.' );
    writeln;writeln;
    writeln( ' ZOOM        :   Press "Z".  Enter  "A"  or  "W"  as  per  choice  of  zooming
to' );
    writeln( '                      actual  size  or  a  window.  Mark  the  window  to  be  zoo
ed' );
    writeln; writeln;
    writeln( ' QUIT        :   Press "Q" to quit the system. ');
    DrawBorder;
  end;

procedure help_menu( screen_no : integer );
begin
  if screen_no = 1 then
    page1;
  if screen_no = 2 then
    page2;
  if screen_no = 3 then
    page3;
  if screen_no = 4 then
    page4;
  GoToXY( 2, 25 );
  write( ' Use :   F9 / F10  --  Backward / Forward          Esc  --  Exit ' );
end;

begin
  screen_no := first_page;
  help_menu( screen_no );
  repeat
    ch := UpCase( ReadKey );
    case ch of
      #0  :  begin
               ch := UpCase( ReadKey );
               case  ch  of
                 F9  :  begin
                          if screen_no <> first_page then
                            screen_no := screen_no - 1;
                          help_menu( screen_no );
                        end;
                 F0  :  begin
                          if screen_no <> last_page then
                            screen_no := screen_no + 1;
                          help_menu( screen_no );
                        end;
               end;
             end;
    end;
  until ch = Escape;
end;
```

```
Procedure HardCopy_rtn;

Var
  prnt_fact_glb                         : longint;


     procedure Scrdump;
     begin
         Inline ($cd/
                 $05
                )
     end;


     procedure compute_matrix( fac : integer; var Del_x, Del_y : real );
     const
       wr = 1.42;

     begin
       Del_x := round( ( 224.5 * fac ) + 0.5 ) + 100;
       Del_Y := round( ( Del_x / Wr ) + 0.5 );
     end;


 procedure print_rtn;
 var
  factor, No_of_rows, No_of_cols        : integer;
  X_loop, Y_loop, grid_no, Result       : integer;
  Delta_X, Delta_Y, X, Y                : real;
  edited, exiting                       : boolean;
  ch_p                                  : char;
  index                                 : byte;
  PrntFactStr                           : String[4];

 begin                                  { print_rtn }
  exiting := false;
  Prntfactstr :='0200';
  Prnt_fact_Glb := 0;
  repeat                                { read the scaling factor }
    edited := false;
    index := 1;
    GoToXY( 1, 25 );
    write( 'Specify the scaling factor  : ', Prnt_fact_Glb );
    ch_p := edit_field( PrntfactStr, 1, 4, false, index, 31, 25, edited, 0 );
    if ch_p <> Escape then
    begin
      val( PrntFactStr, Prnt_fact_Glb, Result);
      if ( Prnt_fact_Glb <= 0 ) or ( Prnt_fact_Glb > 1000 ) then
      begin
        beep;
        GotoXY( 1, 1 );
        writeln( 'Invalid scaling factor ! valid range - 1..1000' );
        repeat
        until KeyPressed;
      end;
    end
    else
      exit;
  until ( Prnt_fact_Glb > 0 ) and ( Prnt_fact_Glb <= 1000 );

  Compute_matrix ( Prnt_fact_Glb, Delta_X, Delta_Y );
  No_of_rows := round( ( abs( ( world_limit_glb[ 1 ] - world_limit_glb[ 3 ] )
 / Delta_Y )  + 0.5 ));
  No_of_cols := round( ( abs( ( world_limit_glb[ 0 ] - world_limit_glb[ 2 ] )
 / Delta_X )  + 0.5 ));
  grid_no := 0;
  Y := round( world_limit_glb[ 1 ] + 0.5 );
  Y_loop := 1;
  X_loop := 1;
  begin
    X := round( world_limit_glb[ 0 ] + 0.5 );
    For X_loop := 1 to No_of_cols do   { inner loop }
    begin
      if not exiting then
      begin
        grid_no := grid_no + 1;
        DefineWorld( 3, X, Y + Delta_Y, X + Delta_X, Y );
        SelectWorld( 3 );
        SelectWindow( 1 );
        ClearScreen;
        GoToXY( 1, 1 );
        write( 'Row  ', Y_loop, '   Column  ', X_loop, '   Matrix grid number
', grid_no );
        GoToXY( 1, 25 );
        write( 'Co-ordinates of lower left corner   X = ', X:7:0, '       Y =
 ', Y:7:0 );
        DrawBorder;
        Draw_list( true );
        Scrdump;
        X := X + Delta_X;
        ClearScreen;
        GotoXY( 1, 1 );
        Beep;
        writeln(' Adjust paper for next page ....');
        write(' Press (P) to continue print,  (Esc) to stop. ');
        repeat
          ch_p := UpCase( ReadKey );
        until ch_p in [ Escape, 'P' ];
        if ch_p = Escape then
          Exit;
      end
    end;                               { end of inner loop }
    Y := Y + Delta_Y;
  end;                                 { end of outer loop }
end;

begin
  exec( 'Graphics.com', '');
  GotoXY(1, 15 );
  beep;
  writeln(' Please make sure that printer is ON and READY...');
  writeln(' Press (Return) ...');
  readln;
  print_rtn;
  SelectWorld( 1 );
  SelectWindow( 1 );
  ClearScreen;
  SetLineStyle( 0 );
  SetClippingOn;
  DrawBorder;
  GotoXY( 1, 25 );
  ClrEol;
  Write( ' Please wait ! Computing  ...... ');
  draw_list( true );
  GotoXY( 1, 25 );
  ClrEol;
  Clear_Comm;
  CopyScreen;
end;
```