GRAPHIC EDITOR
BY
ABDUL WADOOD KHAN

A project report submitted to the QUAID-E-AZAM University, Islamabad, in partial fulfillment of requirement of Postgraduate Diploma in computer sciences.

COMPUTER CENTRE
QUAID-E-AZAM UNIVERSITY
ISLAMABAD
MAY, 2002.

COMPUTER CENTRE
QUAID-E-AZAM UNIVERSITY
ISLAMABAD

FINAL APPROVAL

Certified that we have read the project report submitted by Abdul Wadood Khan. And in our judgement this work is of sufficient standard to warrant its acceptance by Quaid-e-Azam University, Islamabad for the Post graduate Diploma in computer sciences.

COMMITTEE:-

1. EXTERNAL EXAMINER ------------------------

2. SUPERVISOR ---------------------------------------
   Mr. Nazim-ud-Din
   Deputy Director Computer Center
   Quaid-e-Azam University Islamabad.

3. DIRECTOR ---
   Dr. Ghulam Muhammad
   Computer Center,
   Quqid-e-Azam University Islamabad.

PROJECT BRIEF

PROJECT TITLE:	Graphic Editor
UNDERTAKEN BY:	Abdul Wadood Khan
SUPERVISED BY:	Mr. Nazim-ud-Din
OBJECTIVE:	To develop a graphics editor based on Geometrical figures
DATE OF COMMENCEMENT:	March 2002
DATE OF CMPLETION:	June 2002
SOURCE LANGUAGE:	C++ 2.0
OPERATING SYSTEM:	MS DOS 6.0
SYSTEM USED:	IBM 300 GL(PC)

ACKNOWLEDGEMENT

I am thankful to Almighty Allah, Who helped me to complete this project. Working on this project was of great interest for me because it was an experience of passing through a maze-one end leading towards several openings. All openings seemed workable at the first glance. Later it used to dawn upon me that I had a little room for maneuvering.

Initial design of the project could not be completed. Because I had envisaged a vicious project-not in the sense of complexity but the time and my knowledge of the language(c++) were little.

Anyhow then I slashed my project and left the incomplete modules, as a room for improvement is always there.

I am greatly indebted to Mr.Nazim-ud-Din, whom I always found at hand for my guidance. Infect it was him who helped me to conceive the project.

I am also thankful to the staff members of the Computer Center for their help.

Abdul Wadood Khan

Index

CHAPTER 6 Project as it is

CHAPTER 1

INTRODUCTION

1.1 Introduction

Today there is hardly a field of life, which is not affected by graphics. Reason is simple and very old. A picture or a figure conveys which would otherwise require hours of oral explanations or lengthy pages of written messages. Still there is a chance that sound and words may not completely depict the mind of the creator of the message.

On the other hand a figure or a sketch may influence the subject, in the intended and desired way, more easily. Provided the subject has the basic intelligence to decode the pictorial message. For instance traffic signals are universally recognized and understood, messages/instructions regarding traffic-flow-control.

1.2 Computer Graphics

Computer has invaded every sphere of life, may it be commercial, personal, professional or ethical spheres. And as more than ninety percent of computers are used by the users –who are not professionals. Therefore it has become all the more significant to develop a universal language, on part of computer professionals so that their software are better understood, used and marketed.

And it is very interesting that the modern man of twenty-first-century could not develop a new universally understood code of language of his times. He had to revert back to the language of signs and symbols of the stone-age-man.

This is the core reason why graphics user interface (GUI) paradigm, so thoroughly and extensively, is being incorporated in all the new software. Which has increased the importance and use of the computer graphics, as a subject and as a profession in the field of computer sciences.

1.3 Graphics Editor

There may be a variety of the graphics editors. But the underlying feature is to enable the user to design, employing different tools with varying degrees of maneuverability. Tools may be geometric figures, free hand sketching, colors selection etc.

Besides, geometric objects, an option of free hand sketching are also incorporated in the software. The software is menu driven, with keyboard as the input device. While the free hand sketching is using mouse as input device.

CHAPTER 2

OBJECT ORIENTED PROGRAMMING (OOP)

2.1 INTRODUCTION

We live in a world of objects. These objects exist in nature, in human made entities, in commerce and trade, and in the products we use. These can be categorized, described, organized, combined, manipulated and created. For instance doors, windows, paints, colors, glass, walls, roofs, ceilings, frames, hinges, door knobs, etc, all are objects which may be created and manipulated to design and create yet another object ' a house'.

Therefore it is no surprise that an object oriented view would be proposed for the creation of the computer software. It is an abstraction, which enables us to world in ways that help us to understand and navigate it.

Object oriented approach (OOP) was first introduced in the late sixties. And now the approach has evolved into a full-fledged engineering discipline, known as object oriented software engineering.

2.2 OBJECT ORIENTED PARADIGM

The object-oriented approach demands an evolutionary approach towards engineering. Following are the steps, which are followed in a spiral path, starting from the beginning of software to the end.

- Identify candidate classes
- Look up classes in library
- Extract classes if available
- Engineer classes if unavailable
    - Object oriented analysis
    - Object oriented design
    - Programming
    - Testing
- Put new classes in the library
- Construct nth iteration of the system

2.3 CLASSES AND OBJECTS

In object oriented concept, a class is a logical construct, which encapsulates the data and the procedural abstractions required to describe the contents and behavior of some real world entity. A wall of procedures (functions) separates data and the procedural abstractions (operations, methods or services). Which means data (i.e. the attributes) can only be approached through the functions or the procedural abstractions. Which achieves the concept of information hiding, i.e. the attributes of a class is hidden from rest of the software.

By defining all objects that exist within a class inherit it's attributes and the operations that are available to manipulate they attributes. A super-class is a collection of classes, and a subclass is a specialized instance of a class.

2.4 ATTRIBUTES AND DOMAIN

Attributes are attached to classes and objects, and they describe the class or object in some way. And an attribute may have a domain. Which means that an attribute can take on a value defined by a domain.

For example 'automobile' is a class. Which has an attribute 'color'. While color has a domain of red, white, blue, yellow, orange etc. Thus the attribute 'color' can take on any value from within the domain of colors.

2.5 OPERATIONS, METHODS, AND SERVICES

An object encapsulates data (represented as a collection of attributes) and the algorithms that process the data. These algorithms are called operations, methods or services.

2.6 ENCAPSULATION, INHERITANCE, AND POLYMORPHISM

Encapsulation, inheritance and polymorphism are the three characteristics, which differentiate object-oriented paradigm, in a considerable way, from the structured paradigm. Rather these three characteristics are the strong points of the OOD (object oriented design)

Encapsulation

As object oriented class and objects spawned from the class, which encapsulate data and the operations that work on the data in a single package. This concept of encapsulation provides a number of very important benefits:

1. The internal implementation details of data and procedures are hidden from the outside world. Which reduces the propagation of side effects when changes are incorporated in the software.
2. Data structures and the operations that manipulate them are merged in a single named entity-class. This facilitates component 'Reuse'. And reuse, reuse, reuse is the call of the day.
3. Interfaces among encapsulated objects are simplified. Because an object which sends a message need not be concerned with the details of the internal data structures. Thereby interfacing is simplified.

Inheritance

Inheritance is one of the key elements, which differentiates between the conventional and object-oriented systems. Let us see it's utility through an example.

A subclass Y inherits all the attributes and operations associated with it's super-class X. This means that all data structures and algorithms originally designed and implemented for X are immediately available for Y. And no further work need to dine. That is 'Reuse' has been accomplished directly.

Furthermore, any change to the data and operations contained within a super-class is immediately inherited by all subclasses that have inherited from the super-class. Therefore the class hierarchy becomes a mechanism through which changes (at high level) can be immediately propagated through a system.

It is important to note that, at each level of the class hierarchy, new attributes and operations may be added to those that have been inherited from higher levels on the hierarchy. In fact, whenever a new class has to be created, the software engineer has a number of options:

- The class can be designed and built from scratch. Which means inheritance is not used.

- The class hierarchy can be searched for determining, if a class higher in the hierarchy contains most of the required attributes and operations. The new class inherits from the higher class and additions may then be added, as required.

- The class hierarchy can be restructured so that the new class can inherit the required attributes and operations.

- Characteristics of an existing class can be over-ridden and private versions of attributes and operations are implemented for the new class. Over-ridding occurs when attributes and operations are inherited in the normal manner but are then modified to the specific needs of the new class.

Polymorphism

Polymorphism is a characteristic that greatly reduces the effort required extending an existing object oriented system. To understand polymorphism, consider a conventional application that must draw three different types of graphs, line graph, pie chart, and histogram.

Ideally once data are collected for a particular type of graph, the graph should draw itself. To accomplish, this in a procedural way. It is necessary to develop drawing modules for each type of graph. And within the design of each graph type, control logic similar to the following would have to be embedded:

```
Case of graph-type:
        If graph-type = line-graph then Draw-Line-Graph (data);
        If graph-type = pie-chart then Draw-Pie-Chart (data);
        If graph-type = histogram then Draw-Histogram (data);
End case:
```

On the other hand, same problem adopting object-oriented design requires that all of the graphs become sub-classes of a general class say 'graph'. Using a concept called over-lading, each subclass defines an operation called 'Draw'. An object can send a draw message to any one of the object instantiated from anyone of the subclasses. The object receiving the message will invoke it's own 'Draw' operation to create the appropriate graph. Therefore, the design is reduced to:

```
Graph-type draw
```

Furthermore, whenever a new graph is to be added to the system, a subclass is created with it's own, draw' operation. And no changes are required within any object that wants a graph drawn, because message 'graph-type draw' remains unchanged.

To summarize, polymorphism enables a number of different operations to have the same name. Which in turn de-couples the objects from one another, making each more independent.

CHAPTER 3

OBJECT-ORIENTED DESIGN

3.1 INTRODUCTION

What is the relevant object? How do they relate to one another? How do objects behave in the context of the system? How do we specify or model a problem so that we can create an effective design?

Each of these questions is answered within the context of object-oriented analysis (OOA). Which is the first technical activity that is performed as part of object-oriented software engineering.

Coad and Yourdon consider this issue when they write:

" OOA, object –oriented analysis is based upon concepts that we first learned in Kindergarten: Objects and attributes, classes and members, wholes and parts. Why it has taken so long to apply these concepts to the analysis and specification of information systems is anyone's guess"

3.2 OBJECT-ORIENTED ANALYSIS

The objective of the object-oriented analysis is to develop a model that describes computer software as it works to satisfy a set of customer-defined requirements. And this objective is achieved by defining all classes that are relevant to the problem to be solved, the operations and attributes associated with them, the relationships between them, and the behavior they exhibit. To accomplish this, a number of tasks must occur:

1. Basic user requirements must be communicated between the customer and the software engineer.
2. Classes must be identified. (i.e. attributes and methods are defined)
3. A class hierarchy must be specified.
4. Object to object relationships (object connections) should be represented.
5. Object behavior must be modeled.
6. Tasks 1 through 5 are reapplied iteratively until the model is complete.

3.3 A UNIFIED APPROACH TO OOA

Over the past three decades a lot of independent approaches have evolved around the methods to implement OOA to achieve an analysis model. Each model has it's own strong and weak points.

However, over the last decade Grady Booch, James Rumbaugh, and Ivan Jacobson have collaborated to combine the best features of their individual object-oriented analysis and design methods into a unified method. The result called the 'Unified Modeling Language (UML)', has become widely used throughout the software industry.

In UML, a system is represented using five different 'views' that describe the system, distinctly from different perspectives. Each view is defined by a set of diagrams:

User Model View

This view represents the system from the user's (called actors in UML) perspective. This important analysis representation describes a usage scenario from the end-user's perspective.

Structural Model View

Data and functionality are viewed from inside the system. Which means static structures (classes, objects, and relationships) is modeled.

Implementation Model View

The structural and behavioral aspects of the system are represented as they are to be built.

Behavioral Model

This analysis model represents the dynamic or behavioral aspects of the system

Environmental Model View

The structural and behavioral aspects of the environment in which the system is to be implemented are represented.

3.4 THE OOA PROCESS ON THE SOFTWARE

The OOA PROCESS DOES NOT BEGIN WITH A CONCERN FOR OBJECTS. Rather, it begins with an understanding of the manner in which the system will be used. My software was designed to be user-interactive.

3.4.1 USE-CASES

Following were the user's requirements as envisaged by me.

1. It should be a graphic editor, enabling the user to draw different figures,
2. The system should be menu-driven.
3. Following are the geometric figures
   Line
   Arc
   Rectangle
   Square
   Ellipse
   Cone
   Circle
   Pie-slice
   Triangle
4. There should be a facility to fill the figure with user defined color.
5. In case a figure is drawn at the wrong location. The system should be able to erase the previous figure and translate it to the user-defined location.
6. There must be a provision to rotate the drawn figure.
7. The system should provide the user with a facility to scale the drawn figure according to the user defined scaling factor.

8. The help line should be able to guide any user to effectively design, making use of the system.
9. System should provide the user with a facility to change the set up of the screen.
10. The system must be able to save a drawing as file on the hard disk.
11. System should be able to open an already saved file on the hard disk.
12. System must have a facility to enable the user to find the drawing coordinates of a point (pixel) on the drawing area.
13. Free hand drawing facility should also be there to complement the drawing.

3.4.2 CLASS-RESPONSIBILITY-COLLABORATOR MODELING (CRC)

Following classes were identified on the basis of the user scenarios.

Classes

1. Menu class
2. Line class
3. Arc class
4. Ellipse class
5. Rectangle class
6. Square class
7. Cone class
8. Pie-slice class
9. Triangle class
10. Circle class
11. Setup class
12. Mouse class
13. Free hand class

Class Responsibilities

1. **Figures classes**

Figure classes would be the same for all the geometric figures with little variations according to the requirements of the concerned figure

i. Provide user interface.
ii. Accept user values as attributes or data.
iii. Should have the following operations or services.
  a) Draw the figures.
  b) Rotate the figures
  c) Scale the figures.
  d) Transfer the figures.
  e) Erase the old figures before translation
  f) Send control to a common point among all the classes.
  g) Ask the user for the screen coordinates
  h) Set the drawing area coordinates
  I) Display messages for asking the user to input the screen coordinates
  k) Calculate the rotation coordinates

I) Calculate the scaling coordinates
j) Calculate the translation coordinates

2. Position class

It should enable the user to find the drawing coordinates in the drawing area, with the help of mouse.

3. Menu Class

The menu class should be responsible for the following operations:

a) Display the main menu.
b) Highlight the line to which the control is shifted.
c) Display the menu selected by the user:

File menu
Translation menu
Scaling menu
Rotation menu
Mouse message
Help menu
Setup menu

d) Highlight the line of the menu selected, to which the control has been shifted.
e) Close the opened menu.
f) Shift control to the FIGURES.

4.Mouse Class

The mouse class is responsible for the following operations:

a) Activate the mouse.
b) Display the mouse pointer.
c) Restrict the mouse to a defined area.
d) Display the coordinates of a pixel.
e) Display the messages for the mouse activation period.

5.Free Hand Class

Free hand class's responsibilities are:

a) Activate the mouse
b) Display the mouse pointer
c) Restrict the mouse to a defined area
d) Ask for the user's drawing color
e) Display messages
f) Hide the mouse while drawing, with left button pressed
g) Draw in the user defined color

6.Setup Class

It has the following responsibilities:

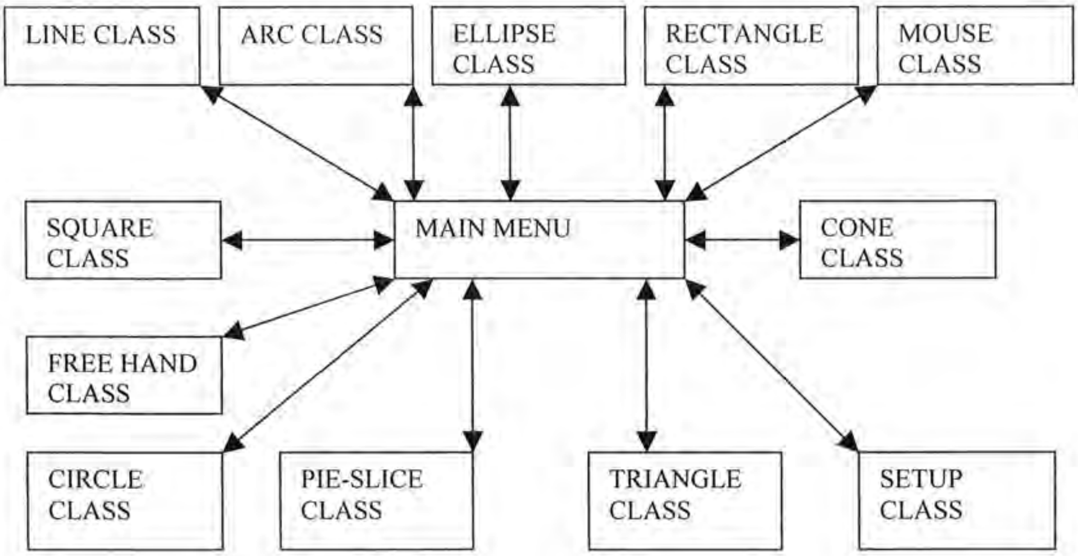
a) Ask the user's choice for the background color of the drawing area
b) Set the desired background color of the drawing area
c) Ask the user for the background color of the messages area
d) Set the desired background color of the messages area
e)

Class Collaboration

In my software, all the classes work independently. Which means that hierarchical distribution was not needed. Which also means that the concepts of Inheritance and polymorphism were not incorporated. If incorporated they would have reduced the code to a considerable degree. However due to the lack of my knowledge about these concepts in the language C++ I could not benefit from them.

3.5 DEFINING STRUCTURES

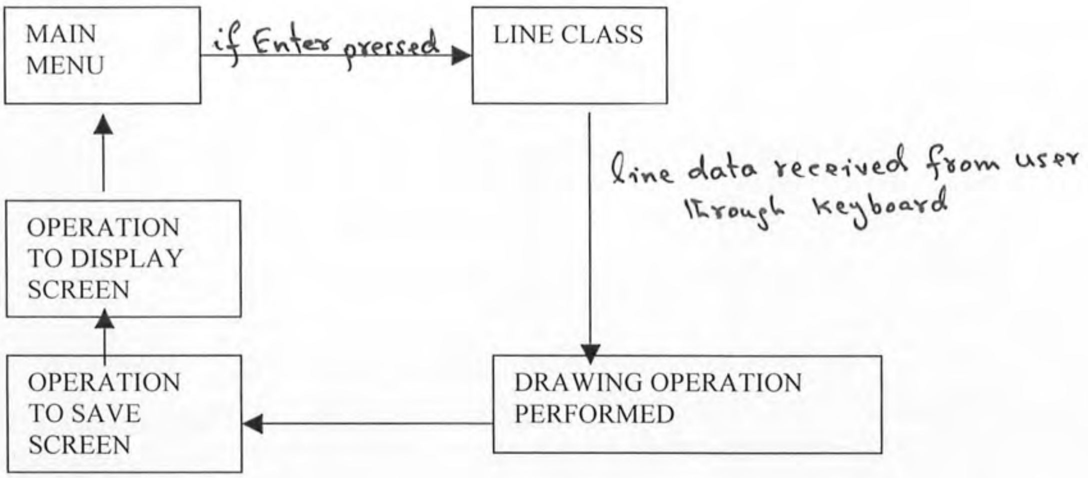
Following is the class structure of the software.



3.6 OBJECT BEHAVIOR MODEL

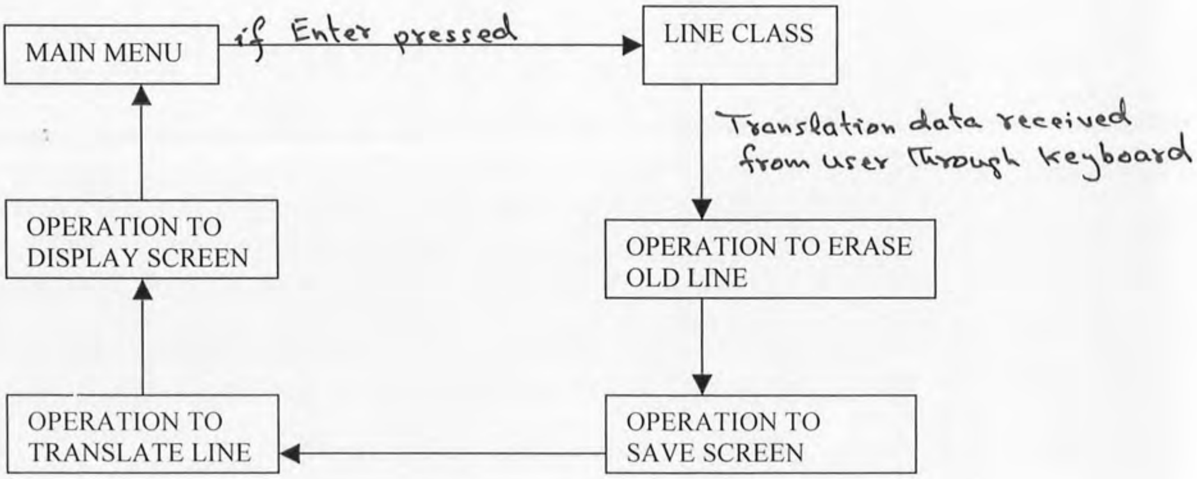
1. **To draw an object**

Following is the object behavior model for all the geometrical objects


MAIN MENU
if Enter pressed
LINE CLASS
line data received from user through keyboard
OPERATION TO DISPLAY SCREEN
OPERATION TO SAVE SCREEN
DRAWING OPERATION PERFORMED

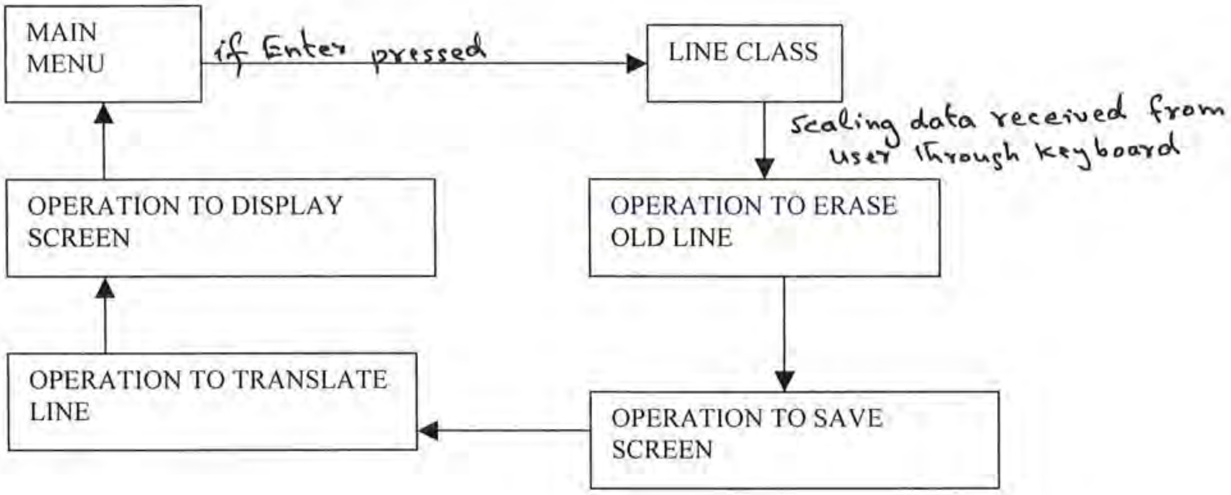
2. **To translate an object**

Following is the object behavior model for the translation of a line as an object. And this model is the same for all other geometrical objects to be translated.


MAIN MENU
if Enter pressed
LINE CLASS
Translation data received from user through keyboard
OPERATION TO DISPLAY SCREEN
OPERATION TO ERASE OLD LINE
OPERATION TO TRANSLATE LINE
OPERATION TO SAVE SCREEN

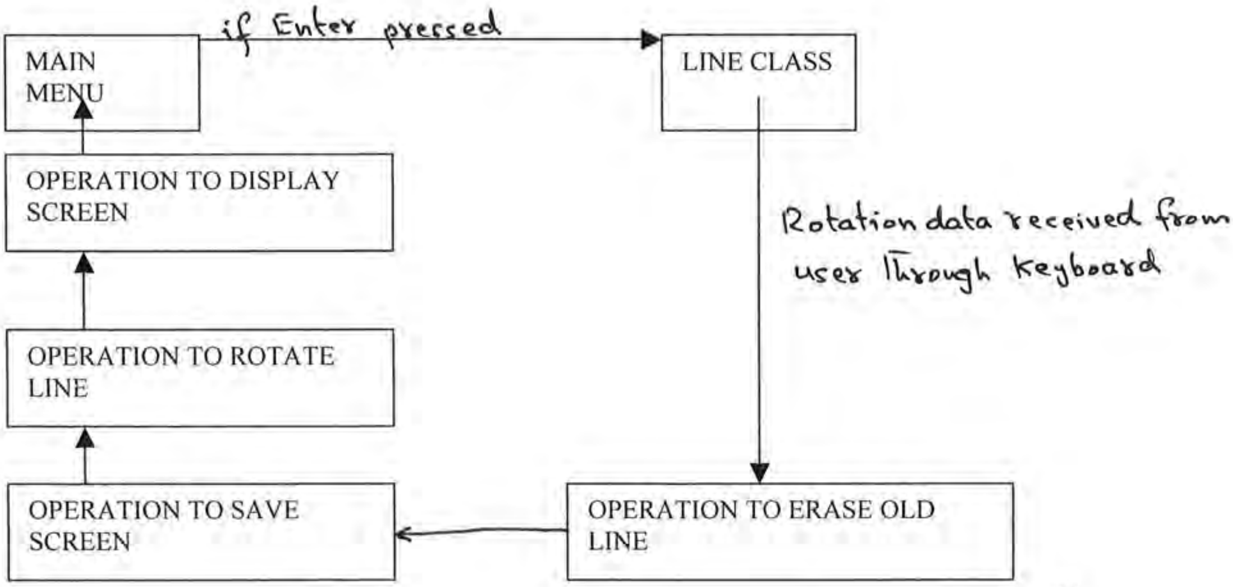
3. To scale an object

Following is the object behavior model for the scaling a line, as an object. And this model is same for all other geometrical figures.

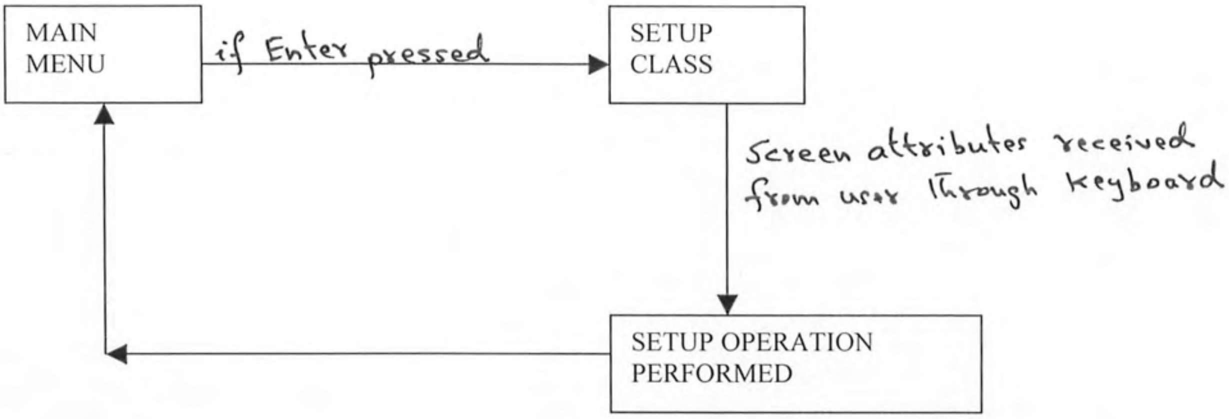

MAIN MENU
if Enter pressed
LINE CLASS
Scaling data received from user through keyboard
OPERATION TO DISPLAY SCREEN
OPERATION TO ERASE OLD LINE
OPERATION TO TRANSLATE LINE
OPERATION TO SAVE SCREEN

4. To rotate an object

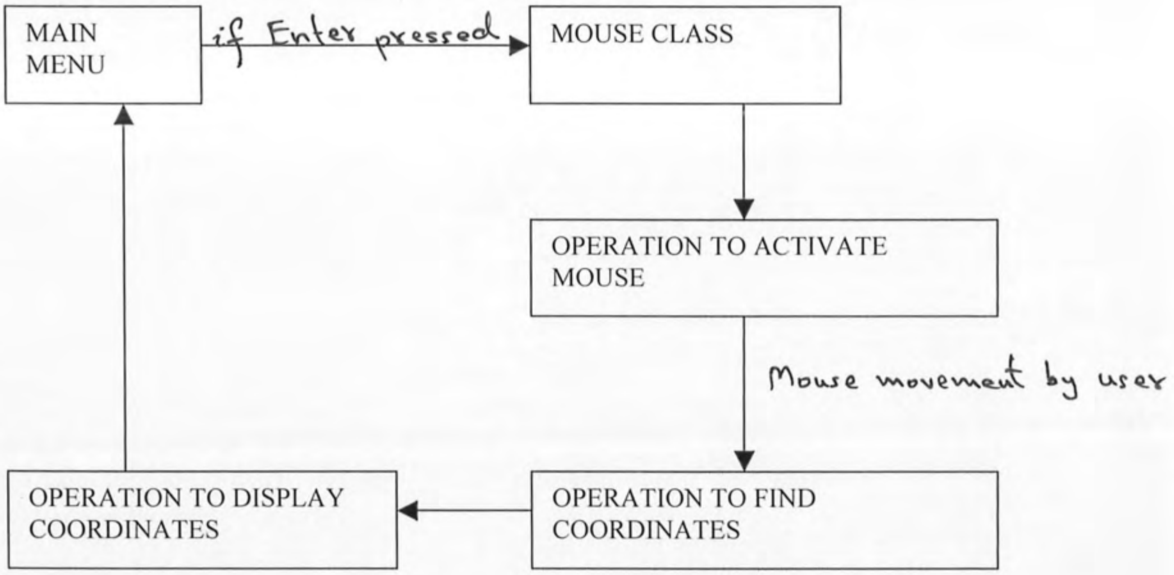
Following is the object behavior model for rotation of line, as an object. Which is true for all other geometrical figures.


if Enter pressed
MAIN MENU
LINE CLASS
OPERATION TO DISPLAY SCREEN
Rotation data received from user through keyboard
OPERATION TO ROTATE LINE
OPERATION TO SAVE SCREEN
OPERATION TO ERASE OLD LINE

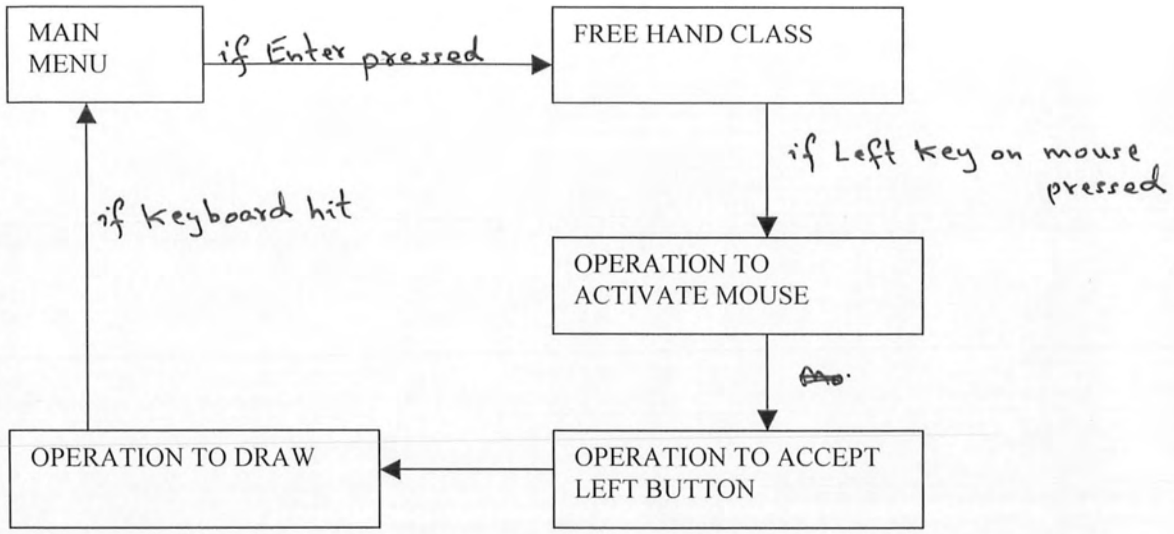
5.To set screen attributes


MAIN MENU
if Enter pressed
SETUP CLASS
Screen attributes received from user through keyboard
SETUP OPERATION PERFORMED

6.To determine position


MAIN MENU
if Enter pressed
MOUSE CLASS
OPERATION TO ACTIVATE MOUSE
Mouse movement by user
OPERATION TO FIND COORDINATES
OPERATION TO DISPLAY COORDINATES

7.To draw with free hand


MAIN MENU
if Enter pressed
FREE HAND CLASS
if Left key on mouse pressed
OPERATION TO ACTIVATE MOUSE
OPERATION TO ACCEPT LEFT BUTTON
OPERATION TO DRAW
if Keyboard hit

CHAPTER 4

OBJECT ORIENTED DESIGN PROCESS

4.1 INTRODUCTION

Object oriented design transforms the analysis mode created; using object oriented analysis, into a design model that serves as a blueprint for software engineering or construction.

UML (unified modeling language) is organized into two major design activities: System design and Object design. The primary objective of UML system design is to represent the software architecture. The conceptual architecture is concerned with the structure of the static class model and the connections between the components of the model.

UML object design focuses on a description of objects and their interactions with one another. And then System and Object design in UML are extended to consider the design of the user interfaces. The user model view of the analysis model drives the user interface design process, providing a scenario that is elaborated through iteration to become a set of interface classes.

4.2 THE SYSTEM DESIGN PROCESS

System design develops the architectural detail required to build a system or a product.

4.2.1 Partitioning the analysis model

The software is partitioned into **three layers**:

Presentation Layer,
Data base Layer, and
Application Layer.

The Presentation Layer

The main menu is the top layer. In which the main menu is the main-system. And it has the following **sub-systems**:

File (communicates with the file class)
Translation (communicates with any of the figure classes)
Scaling (communicates with any of the figure classes)
Rotation (communicates with any of the figure classes)
Help (communicates with the help class)
Position (communicates with the mouse class)
Free hand (communicates with the free hand class)
Line (communicates with the line class)
Arc (communicates with the arc class)
Pie-slice (communicates with the pie-slice class0
Rectangle (communicates with the rectangle class)
Square (communicates with the square class)
Triangle (communicates with the triangle class)
Circle (communicates with the circle class)
Cone (communicates with the cone class)
Setup (communicates with the setup class)

Messages sent from the presentation layer

As the architecture of my system is an open system, so messages may be sent to any lower layer, from any layer. In a closed architecture the messages may be sent only to the adjacent layers.

Messages are the means by which object interact. A message stimulates some behavior to occur in the receiving object. The behavior is accomplished when an operation is executed. General format of the message is

Message: [destination, operation, and parameters]

Message: [file class, open file menu, name of file]
Message: [translate, open translation menu, enter-key-signal]
Message: [scaling, open scale menu, enter-key-signal]
Message: [rotate, open rotation menu, enter-key-signal]
Message: [help class, open help menu, enter-key-signal]
Message: [mouse classes, activate, enter-key-signal]
Message: [free-hand-class, activate, enter-key-signal]
Message: [line-class, take coordinates, enter-key-signal]
Message: [arc-class, take coordinates, enter-key-signal]
Message: [pie-slice-class, take coordinates, enter-key-signal]
Message: [rectangle-class, take coordinates, enter-key-signal]
Message: [square-class, take coordinates, enter-key-signal]
Message: [triangle-class, take coordinates, enter-key-signal]
Message: [circle-class, take coordinates, enter-key-signal]
Message: [cone-class, take coordinates, enter-key-signal]

The Database Layer

Following are the **subsystems** of the database layer, which is the middle layer between the presentation layer and the application layer.

Line data (communicates with the line class)
Arc data (communicates with the arc class)
Pie-slice data (communicates with the pie-slice class)
Rectangle data (communicates with the rectangle class)
Square data (communicates with the square class)
Triangle data (communicates with the triangle class)
Circle data (communicates with the circle class)
Cone data (communicates with the cone class)

Scale data (communicates with the figure classes)
Rotate data (communicates with the figure classes)
Translate data (communicates with the figure classes)
Free hand data (communicates with the free hand class)

Setup data (communicates with the setup class)
Mouse data (communicates with the mouse class)
Open file data (communicates with the file class)
Close file data (communicates with the file class)
Save file data (communicates with the file class)

Messages sent from the database layer

Message: [line data, set drawing data, data input from keyboard]
Message: [arc data, set drawing data, data input from keyboard]
Message: [pie-slice data, set drawing data, data input from keyboard]
Message: [rectangle data, set drawing data, data input from keyboard]
Message: [square data, set drawing data, data input from keyboard]
Message: [triangle data, set drawing data, data input from keyboard]
Message: [circle data, set drawing data, data input from keyboard]
Message: [cone data, set drawing data, data input from keyboard]

Message: [rotate data, set rotation data for line, input from keyboard]
Message: [rotate data, set rotation data for arc, input from keyboard]
Message: [rotate data, set rotation data for rectangle, input from keyboard]
Message: [rotate data, set rotation data for square, input from keyboard]
Message: [rotate data, set rotation data for triangle, input from keyboard]
Message: [rotate data, set rotation data for circle, input from keyboard]
Message: [rotate data, set rotation data for pie-slice, input from keyboard]
Message: [rotate data, set rotation data for cone, input from keyboard]

Message: [scale data, set scaling data for line, input from keyboard]
Message: [scale data, set scaling data for arc, input from keyboard]
Message: [scale data, set scaling data for rectangle, input from keyboard]
Message: [scale data, set scaling data for square, input from keyboard]
Message: [scale data, set scaling data for triangle, input from keyboard]
Message: [scale data, set scaling data for circle, input from keyboard]
Message: [scale data, set scaling data for pie-slice, input from keyboard]
Message: [scale data, set scaling data for cone, input from keyboard]

Message: [translate data, set translation data for line, input from keyboard]
Message: [translate data, set translation data for arc, input from keyboard]
Message: [translate data, set translation data for rectangle, input from keyboard]
Message: [translate data, set translation data for square, input from keyboard]
Message: [translate data, set translation data for triangle, input from keyboard]
Message: [translate data, set translation data for circle, input from keyboard]
Message: [translate data, set translation data for pie-slice, input from keyboard]
Message: [translate data, set translation data for cone, input from keyboard]

Message: [free hand data set mouse coordinates, mouse movement]
Message: [position data, set mouse coordinates, mouse movement]
Message: [open file data, read file name, input from keyboard]
Message: [close file data, read file name, input from keyboard]
Message: [save file data, read file name, input from keyboard]

The Application Layer

Following are the **subsystems** of the application layer

Draw line (communicates with line class)
Draw arc (communicates with the arc class)
Draw pie-slice (communicates with the pie-slice class)
Draw rectangle (communicates with the rectangle class)
Draw Square (communicates with the square class)
Draw circle (communicates with the circle class)
Draw triangle (communicates with the triangle class)
Draw cone (communicates with the cone class)

Translate line (communicates with line class)
Translate arc (communicates with the arc class)
Translate pie-slice (communicates with the pie-slice class)
Translate rectangle (communicates with the rectangle class)
Translate square (communicates with the square class)
Translate circle (communicates with the circle class)
Translate triangle (communicates with the triangle class)
Translate cone (communicates with the cone class)

Rotate line (communicates with line class)
Rotate arc (communicates with the arc class)
Rotate pie-slice (communicates with the pie-slice class)
Rotate rectangle (communicates with the rectangle class)
Rotate square (communicates with the square class)
Rotate circle (communicates with the circle class)
Rotate triangle (communicates with the triangle class)
Rotate cone (communicates with the cone class)

Scale line (communicates with line class)
Scale arc (communicates with the arc class)
Scale pie-slice (communicates with the pie-slice class)
Scale rectangle (communicates with the rectangle class)
Scale Square (communicates with the square class)
Scale circle (communicates with the circle class)
Scale triangle (communicates with the triangle class)
Scale cone (communicates with the cone class)

Open file (communicates with the file class)
Close file (communicates with the file class)
Save file (communicates with the file class)
Activate position (communicates with the help class)
Activate free hand (communicates with the free hand class)

Messages sent from the application layer

Message: [draw line, draw, coordinates and color]
Message: [draw arc, draw, coordinates and color]
Message: [draw pie-slice, draw, coordinates and color]
Message: [draw rectangle, draw, coordinates and color]
Message: [draw square, draw coordinates and color]
Message: [draw triangle, draw, coordinates and color]
Message: [draw circle, draw, coordinates and color]
Message: [draw cone, draw, coordinates and color]

Message: [translate line, translate, distance and direction]
Message: [translate arc, translate, distance and direction]
Message: [translate pie-slice, translate, distance and direction]
Message: [translate rectangle, translate, distance and direction]
Message: [translate square, translate, distance and direction]
Message: [translate triangle, translate, distance and direction]
Message: [translate circle, translate, distance and direction]
Message: [translate cone, translate, distance and direction]

Message: [rotate line, rotate, angle]
Message: [rotate arc, rotate, angle]
Message: [rotate pie-slice, rotate, angle]
Message: [rotate rectangle, rotate, angle]
Message: [rotate, square, rotate, angle]
Message: [rotate, circle, rotate, angle]
Message: [rotate triangle, rotate, angle]
Message: [rotate cone, rotate, angle]

Message: [open-file, open, file-name and drive name]
Message: [close-file, close, drawing area coordinates]
Message: [save-file, save, file-name and drive name]
Message: [activate-mouse, activate, mouse movement]
Message: [activate free hand, activate, (mouse-position, left-key-signal)]

4.2.2 INTER SUBSYSTEM COMMUNICATION

Following is the subsystem collaboration table.

Contract	Type	Collaborators	Classes	Operation(s)	Message
Request	User/system	• File • Open file data	File	• Open menu • Set data	User presses Enter key
Request	User/system	• File • Close file data	File	• Open menu • Set data	User presses Enter key
Request	User/system	• File • Save file data	File	• Open menu • Set data	User presses Enter key
Internal trigger	Peer-to-peer	• Open file data • Open file	File	• Open the file	Internal call to the object
Internal trigger	Peer-to-peer	• Close file data • Close file	File	• Close the file	Internal call to the object
Internal trigger	Peer-to-peer	• Save file data • Save file	File	• Save the file	Internal call to the object
Request	User/system	• Position • Activate position	Mouse	• Set screen • Set mouse • Set cursor • Display coordinates	User presses enter key
Request	User/system	• Free hand • Free hand data	Free hand	• Set screen • Save screen • Set mouse coordinates • Activate mouse	User presses enter key
Request	User/system	• Free hand data • Activate free hand	Free hand	• Set data • Fill the pixel • Display pixel	User presses Enter key
Request	User/system	• Line • Line data	Line	• Receive data • Set data	User inputs through keyboard
Internal trigger	Peer-to-peer	• Line data • Draw line	Line	• Save screen • Draw • Display screen	Internal call to object
Request	User/system	• Line data	Line	• Save screen	User Inputs

		• Translate line		• Erase line • Translate line • Display screen	through keyboard
Request	User/system	• Line data • Rotate line	Line	• Save screen • Erase line • Rotate line • Display screen	User inputs through keyboard
Request	User/system	• Line data • Scale data	Line	• Save screen • Erase line • Scale line • Display screen	User inputs through keyboard
Request	User/system	• Arc • Arc data	Arc	• Receive data • Set data	User inputs through keyboard
Internal trigger	Peer-to-peer	• Arc data • Draw arc	Arc	• Save screen • Draw • Display screen	Internal call to object
Request	User/system	• Arc data • Translate arc	Arc	• Save screen • Erase arc • Translate arc • Display screen	User Inputs through keyboard
Request	User/system	• Arc data • Rotate arc	Arc	• Save screen • Erase arc • Rotate arc • Display screen	User inputs through keyboard
Request	User/system	• Arc data • Scale data	Arc	• Save screen • Erase arc • Scale arc • Display screen	User inputs through keyboard
Request	User/system	• Pie-slice • Pie-slice data	Pie-slice	• Receive data • Set data	User inputs through keyboard
Internal trigger	Peer-to-peer	• Pie-slice data • Draw pie-slice	Pie-slice	• Save screen • Draw • Display screen	Internal call to object
Request	User/system	• Pie-slice data • Translate pie-slice	Pie-slice	• Save screen • Erase pie-slice	User Inputs through keyboard

		slice		slice • Translate pie-slice • Display screen	
Request	User/system	• Pie-slice data • Rotate pie-slice	Pie-slice	• Save screen • Erase pie-slice • Rotate pie-slice • Display screen	User inputs through keyboard
Request	User/system	• Pie-slice data • Scale data	Pie-slice	• Save screen • Erase pie-slice • Scale pie-slice • Display screen	User inputs through keyboard
Request	User/system	• Rectangle • Rectangle data	Rectangle	• Receive data • Set data	User inputs through keyboard
Internal trigger	Peer-to-peer	• Rectangle data • Draw Rectangle	Rectangle	• Save screen • Draw • Display screen	Internal call to object
Request	User/system	• Rectangle data • Translate Rectangle	Rectangle	• Save screen • Erase Rectangle • Translate Rectangle • Display screen	User Inputs through keyboard
Request	User/system	• Rectangle data • Rotate Rectangle	Rectangle	• Save screen • Erase Rectangle • Rotate Rectangle • Display screen	User inputs through keyboard
Request	User/system	• Rectangle data • Scale data	Rectangle	• Save screen • Erase Rectangle • Scale Rectangle • Display screen	User inputs through keyboard
Request	User/system	• Triangle • Triangle data	Triangle	• Receive data	User inputs through

				• Set data	keyboard
Internal trigger	Peer-to-peer	• Triangle data • Draw Triangle	Triangle	• Save screen • Draw • Display screen	Internal call to object
Request	User/system	• Triangle data • Translate Triangle	Triangle	• Save screen • Erase Triangle • Translate Triangle • Display screen	User Inputs through keyboard
Request	User/system	• Triangle data • Rotate Triangle	Triangle	• Save screen • Erase Triangle • Rotate Triangle • Display screen	User inputs through keyboard
Request	User/system	• Triangle data • Scale data	Triangle	• Save screen • Erase Triangle • Scale Triangle • Display screen	User inputs through keyboard
Request	User/system	• Circle • Circle data	Circle	• Receive data • Set data	User inputs through keyboard
Internal trigger	Peer-to-peer	• Circle data • Draw Circle	Circle	• Save screen • Draw • Display screen	Internal call to object
Request	User/system	• Circle data • Translate Circle	Circle	• Save screen • Erase Circle • Translate Circle • Display screen	User Inputs through keyboard
Request	User/system	• Circle data • Rotate Circle	Circle	• Save screen • Erase Circle • Rotate Circle • Display screen	User inputs through keyboard
Request	User/system	• Circle • Scale data	Circle	• Save screen • Erase Circle • Scale Circle	User inputs through keyboard

				• Display screen	
Request	User/system	• Square • Square data	Square	• Receive data • Set data	User inputs through keyboard
Internal trigger	Peer-to-peer	• Square data • Draw Square	Square	• Save screen • Draw • Display screen	Internal call to object
Request	User/system	• Square data • Translate Square	Square	• Save screen • Erase Square • Translate Square • Display screen	User Inputs through keyboard
Request	User/system	• Square data • Rotate Square	Square	• Save screen • Erase Square • Rotate Square • Display screen	User inputs through keyboard
Request	User/system	• Square data • Scale data	Square	• Save screen • Erase Square • Scale Square • Display screen	User inputs through keyboard
Request	User/system	• Cone • Cone data	Cone	• Receive data • Set data	User inputs through keyboard
Internal trigger	Peer-to-peer	• Cone data • Draw Cone	Cone	• Save screen • Draw • Display screen	Internal call to object
Request	User/system	• Cone data • Translate Cone	Cone	• Save screen • Erase Cone • Translate Cone • Display screen	User Inputs through keyboard
Request	User/system	• Cone data • Rotate Cone	Cone	• Save screen • Erase Cone • Rotate Cone • Display screen	User inputs through keyboard

Request	User/system	• Cone data • Scale data	Cone	• Save screen • Erase Cone • Scale Cone • Display screen	User inputs through keyboard

4.3 OBJECT DESIGN PROCESS

4.3.1 INTRODUCTION

A design description of an object (an instance of a class or subclass) can take one of two forms:

- **A protocol description**:

That establishes the interface of an object by defining each message that the object can receive and the related operation that the object performs when it receives the message.

- **An implementation description**:

That shows implementation details for each operation implied by a message that is passed to an object. It provides the internal details that are required for implementation.

As I will code the design in C++, therefore I am going for the 'implementation description'.

4.3.2 PROGRAM COMPONENTS

```
PACKAGE program-component-open file
        TYPE
            File name. string (8)
        PROC read, load, display.
        PACKAGE BODY program-component-open file
            PROC operation. Read (file name)
                  Read. File-name. Keyboard
                  Compare. File names
                        IF (matched) THEN load. File-name. Buffer
                        ELSE
                              DISPLAY. Message = 'no file found'.
                                        EXIT
              END
              PROC operation. load (file)
                    Load. File. Buffer
              END
              PROC operation. Display (file)
                    DISPLAY. File. Monitor
              END
END program-component-open file
```

```
PACKAGE program-component-close file
        TYPE
              Int. drawing-area-coordinates
        PROC  get, close
        PACKAGE BODY program-component-close file
                PROC operation. Get (drawing area coordinates)
                      Pass (coordinates) TO operation. Close
                END
                PROC operation. Close (file)
                      FOR (coordinates)
                          Put-pixel. White
                END
END program-component-close file
```

```
PACKAGE program-component-save file
        TYPE file-name. String (8), drive char (1)
        PROC read, save
        PACKAGE BODY program-component-save file
                PROC operation. Read (file-name)
                      Read. File-name. Keyboard
                      Read. Drive-name. Keyboard
                END
                PROC operation. Save (file)
                      Compare. File-names
                      IF (not matched) THEN save. File ON drive
                      ELSE
                          DISPLAY. Message = 'file already exists, overwrite?'
                                 IF yes THEN save. File ON drive
                                 ELSE EXIT
                END
END program-component-save file
```

```
PACKAGE program-component-position
        TYPE int. position-coordinates
         PROC get, display
         PACKAGE BODY program-component-position
                PROC operation. Get (position coordinates)
                      FOR (not keyboard hit)
                           Get. Coordinates
                      PASS. Coordinates TO operation. Display
                END
                PROC operation. Display (coordinates)
                      FOR (not keyboard hit)
                            Display. Coordinates. Monitor
                END
END program-component-position
```

```
PACKAGE program-component-free hand
           TYPE int. position-coordinates
            PROC get, write, display
            PACKAGE BODY program-component-free hand
                    PROC operation. Get (position coordinates)
                          FOR (not keyboard hit)
                                Get. Coordinates
                           PASS (coordinates) TO operation. Write
                     END
                     PROC operation. Write (pixel)
                           FOR (not keyboard hit)
                                  FOR (left key)
                                       PASS (write. Pixel. Coordinates)
                                       TO  operation. Display
                      END
                      PROC operation. Display (pixel)
                            Display. Pixel
                      END
END program-component-free hand
```

```
PACKAGE program-component-line
      TYPE PRIVATE int. center coordinates
                      Int. drawing coordinates
                      Int. translation coordinates
       PROC clear-message-area
             Make-message-area
             Make-drawing-area
             Save-screen
             Display-screen
             Get-center coordinates
             Set-drawing-coordinates
             Set-translation-coordinates
             Draw-line
             Translate-line
             Erase-line
        PACKAGE BODY program-component-line
                 PROC operation. Clear-message-area
                       PUBLIC int. message-area-coordinates = 'DEFINED '
                          FOR (message-area-coordinates)
                                  Write. Pixel. White
                 END
                 PROC operation. Make-message-area
                       PUBLIC TYPE int. message-area-coordinates = 'DEFINED'
                          Make-boundary. Message-area-coordinates
                 END
                 PROC operation. Make-drawing-area
                      PUBLIC int. drawing-area-coordinates = 'DEFINED'
                          Make-boundary. Drawing-area-coordinates
                 END
                 PROC operation. Save-screen (drawing-area)
                        PUBLIC int. drawing-area-coordinates = 'DEFINED'
                          FOR (drawing-area-coordinates)
                                  Array = get. Pixel. Color
                 END
                 PROC operation. Display-screen (drawing-area)
                         PUBLIC int. drawing-area-coordinates = 'DEFINED'
                                  FOR (drawing-area-coordinates)
                                      Put. Pixel. Color = array
                 END
        PROC operation. Get-center-coordinates
                PUBLIC int. x, y
                    DISPLAY. Message = 'Enter center coordinates'
                    GET. X, y
                    CALCULATE. Center-coordinates
                    DISPLAY. Message = 'Enter color'
                    GET. Color
        END
        PROC operation. Set-drawing-coordinates
                PUBLIC
                    CALCULATE. Drawing-coordinates
        END
        PROC operation. Set-translation-coordinates
                PUBLIC
                    CALCULATE. Translation-coordinates
        END
        PROC operation. Draw-line
                PUBLIC
                    FOR (drawing-coordinates)
                        Write. Pixel. Color
        END
        PROC operation. Translate-line
                PUBLIC
                    FOR (translation-coordinates)
                        Write. Pixel.. color
        END
        PROC operation. Erase-line
                PUBLIC
                    FOR (drawing-coordinates)
                        Write. Pixel. White
        END
END program-component-line
```

NOTE: The above component is true for all geometrical figures, with the corresponding changes to the concerned figure.

CHAPTER 5

MATHEMATICAL PROCEDUTRES AND ALGORITHMS

5.1 INTRODUCTION

As my software's core working area is the geometric figures. For which mathematical routines were developed. After developing their mathematical routines, correspondingly their algorithms were developed. And these algorithms were then coded in C++.

In this chapter along with the necessary mathematical background are presented. Where x stands for the columns and y for the rows. I have also made use of the C++ built-in graphics functions.

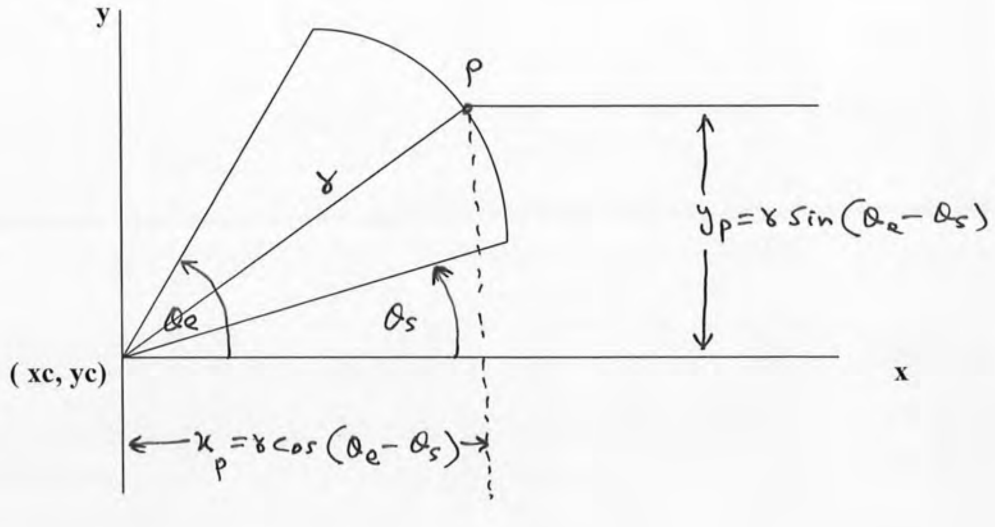
5.2 LINE

A (ax, ay)
B (bx, by)

Where ax & ay are the starting coordinates
And bx, by are the ending coordinates.

```
1.  START
2.  INPUT starting coordinates (ax, ay)
3.  INPUT ending coordinates (bx, by)
4.  INPUT color
5.  FOR (starting to ending coordinates)
            WRITE. Position. Color
6. END
```

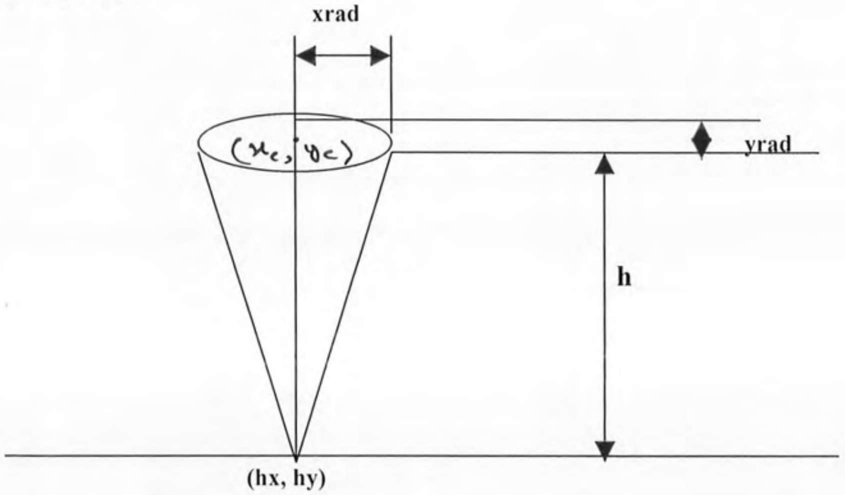
5.3 ARC


y
P
r
θ_e
θ_s
$y_p = r \sin(\theta_e - \theta_s)$
(xc, yc)
x
$x_p = r \cos(\theta_e - \theta_s)$

R = radius
Xc, yc = center coordinates
= starting angle
= ending angle

```
1.  START
2.  INPUT radius (r)
3.  INPUT center-coordinates (xc, yc)
4.  INPUT starting angle (  )
5.  INPUT ending angle (  )
6.  INPUT color
7.  CALCULATE
          xp = rcos (              )
          yp = rsin (              )
           x = xc + xp
           y = yc + yp
           WRITE. X. Y. Color

8. END
```

5.4 CONE

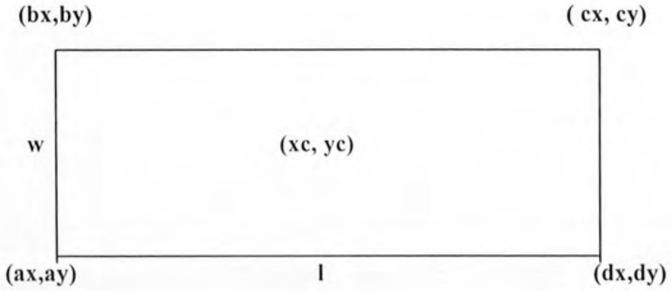


xc,yc = center coordinates
h = height
xrad = semi-major axis of base
yrad = semi-minor axis of base
yrad = 1/3 xrad

```
1.  START
2.  INPUT center coordinates (xc, yc)
3.  INPUT height (h)
4.  INPUT base-color
5.  INPUT height color
6.  INPUT radius of base (xrad)
7.  CALCULATE
              Yrad = 1/3 (xrad0
              Hx = xc
              Hy = h + yc
8.  SET. Heightcolor
9.  DRAW. Height
               FOR (Hx. Hy. Yrad)
                      WRITE. heightcolor
```

```
10. CALCUALTE
                    X= Hx-xrad
                    Y= Hy
11. SET. Basecolor
12. DRAW BASE
                    FOR (X, Y)
                    WRITE. X. Y. basecolor

13. END
```

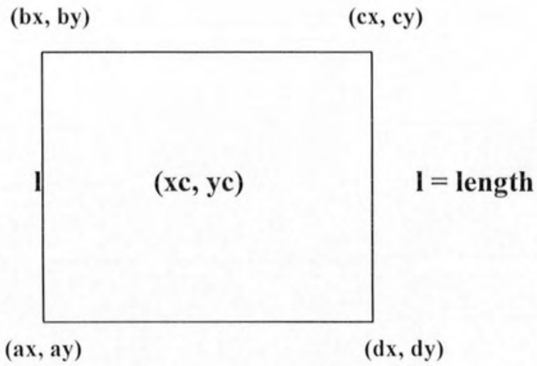
5.5 RECTANGLE



w = width
L = length

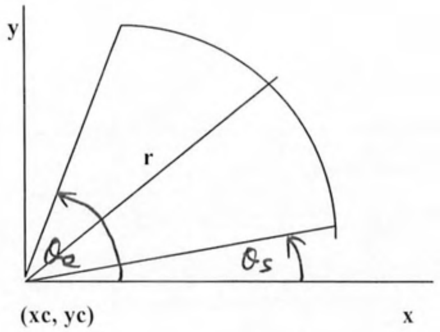
```
1.  START
2.  INPUT length (l)
3.  INPUT width (w)
4.  INPUT color
5.  INPUT center coordinates (xc, yc)
6.  CALCULATE
                    Ax = xc- (l/2)
                    Ay = yc- (w/2)
                    Bx = ax
                    By = yc- w/2
                    Cx = by
                    Dx = cx
                    Dy = ay
7.  DRAW rectangle
                      FOR (X-coordinates)
                            FOR (Y-coordinates)
                                   WRITE. color
8.END
```

5.6 SQUARE



```
1.  START
2.  INPUT cnter coordinates (xc, yc)
3.  INPUT length (l)
4.  INPUT color
5.  CALCULATE
            $Ax = xc - l/2$
            $Ay = yc + l/2$
            $Bx = ax$
            $By = yc - l/2$
            $Cx = xc + l/2$
            $Cy = by$
            $Dx = cx$
            $Dy = ay$
6.  DRAW square
            FOR (X-coordinates)
                    FOR (Y-coordinates)
                            WRITE. color


7.  END
```

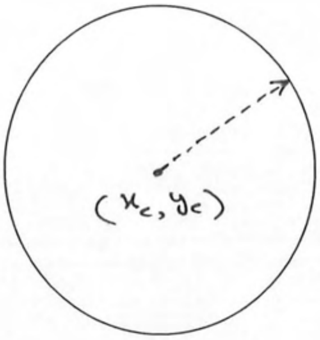
5.7 PIE-SLICE



R = radius
= starting angle
= ending angle
xc, yc = center coordinates

```
START
INPUT radius
INPUT starting angle
INPUT ending angle
INPUT color
CLACULATE
            Yrad = radius
DRAW pie-slice
            FOR (starting angle, ending angle, Yrad)
                  WRITE. color
END
```

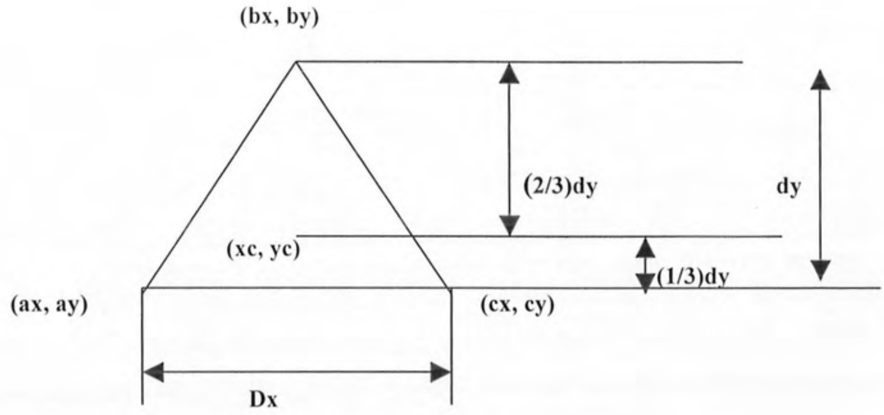
5.8 CIRCLE



R = radius
Xc, yc = center coordinates

```
1. START
2. INPUT color
3. INPUT radius
4. CALCULATE
        FOR (angle= 0 TO 360)
            X= radius (cosine of angle)
            Y= radius (sine of angle)
5. DRAW circle
        FOR (X, Y)
            WRITE. color
6. END
```

5.9 TRIANGLE



Dy = height
Dx = width
Xc, yc = center coordinates

```
1. START
2. INPUT center coordinates (xc, yc)
3. INPUT width (dx)
4. INPUT height (dy)
5. INPUT color
6. CLACULATE
                Ax = xc – (1/2)dx
                Ay = yc + (1/3)dy
                Bx = xc
                By = yc – (2/3)dy
                Cx = xc + (1/2)dx
                Cy = yc + (1/30dy
7. SET. color
7. DRAW triangle
                FOR (ax, ay, bx, by, cx, cy)
                        WRITE. color
8. END
```

CHAPTER 6

PROJECT AS IT IS

6.1 INTRODUCTION

In this chapter the software is described as it is. Afterwards the room for improvement is discussed. The deficiencies in the initial design are also discussed

6.2 SOFTWARE AS IT IS

The software is menu driven. At the start the main menu is displayed. In which on the top line following are the objects:

- Figures
- File
- Scale
- Help
- Translation
- Rotation
- Mouse
- Free hand
- Setup

Figure is the common point to which the control is transferred after the execution an object (component). Whenever background color is white and the foreground color is red, the control will be on figures, in other words in the hands of users.

If the user presses the down arrow key he can select the geometrical objects to draw. Once an object is selected it is highlighted. And the system asks the users to input the needed values for selected object. The user is supposed to interact with the system through the keyboard.

Scaling means to enlarge an object or to shrink an object to the predefined factors.
It has a submenu that allows the user to select the factor scaling through a multiple-choice option. After selecting the desired option the system asks the user to select the object to be scaled.

Translation means to move an object in one of the four directions, up, down, left, right. It has a submenu that asks the user to select the direction. After selecting the direction the system would asks the user to select the object to be translated. Upon selection of object the needful is done, after erasing the previous figure.

Rotation means to rotate through predefined angles in counter clockwise direction. It has a submenu, which asks the user to select the angle. After selecting the angle the system would asks the user to select the object to be translated. Upon selection of object the needful is done, after erasing the previous figure.

In the setup the user is provided with a facility to change the following:

- Background color of the drawing area.
- Background color of the message area and
- Text color of the screen writing.

The software is equipped with a help line. Which can be called by selecting the HELP from the main menu. The help line has eighteen pages of explanations; one page dedicated to an object.

As the software is used, for the purpose of designing, with the help of geometrical objects. For which the user must know the exact location on the drawing area. The MOUSE, when selected, enables the user to do so, with the help of mouse in his hands. The pixels' coordinates are displayed as the mouse is moved within the drawing area.

Free hand drawing utility is also provided in the system. Which can be activated if the user selects the FREEHAND from the main menu. The mouse pointer will appear on the drawing area. And will move with the movement of the mouse. When the left button on the mouse is pressed the mouse pointer will disappear. Which is an indication of the fact that now the user can draw, by dragging the mouse. Before activating the free hand drawing facility the system would ask the user to choose the drawing color. For this purpose the colors along with their code numbers are displayed on the screen.

6.3 DEFFICIENCIES

The software could not be completed as per the original design. Reasons were two. First my little knowledge about the language C++. Second, time. For which the following components of the design could not be coded:

- Rotation
- Scaling
- Setup
- File saving
- File retrieval

6.4 ROOM FOR IMPROVEMENT

A room for improvement is always there. It is true here. Personally I think that the code of the software could have considerably be slashed, had I employed the strong concepts of C++. Which are the inheritance and polymorphism. Secondly, there could be a design in which the help can be called at any point during the execution of the software. Presently it is not so.

6.5 USER INTERFACE

The software is equipped with the user interface in the shape of proper messages displayed at the needed places. Furthermore, throughout the process of drawing the user is not left to guess. Rather he is guided at every step. The screen outputs of the different messages are part of this chapter.

6.6 DIVISION OF THE MAIN SCREEN

The main screen of the display monitor is divided in the following areas:

- Drawing area
- Message area
- Tips area
- Top line
- Figures area

Drawing Area

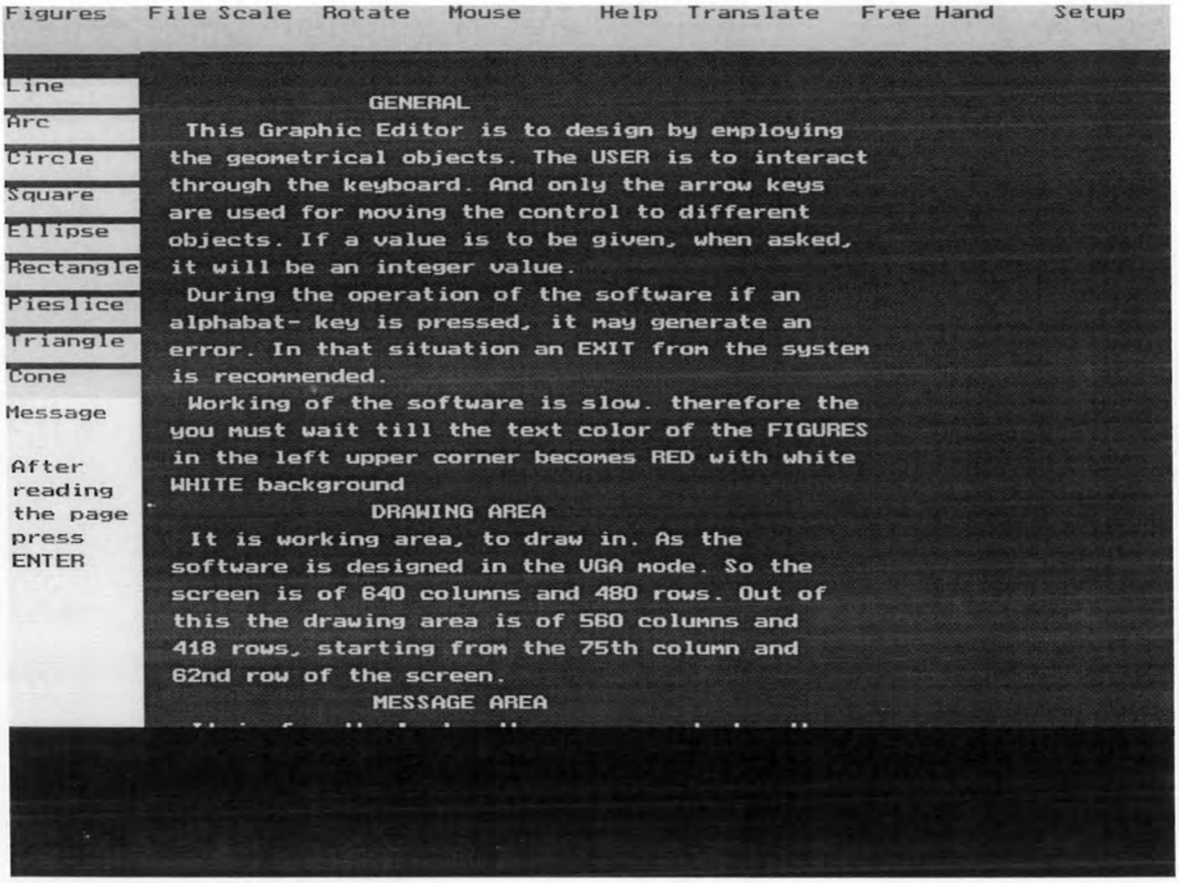
The drawing area is a portion of the screen, reserved for the user to draw his designing. The software is designed in the VGE mode. In which the screen has 640 columns and 480 rows. The drawing area starts at the 75th column and 62nd row, with a dimension of 560 columns and 418 rows. It is this area in which the mouse movement is also restricted while drawing with the free hand facility.

Message Area

The message area is reserved for the purpose of displaying messages, needed while interacting with the user. It starts at the 75th column and 25th row, with a dimension of 560 columns and 37 rows.

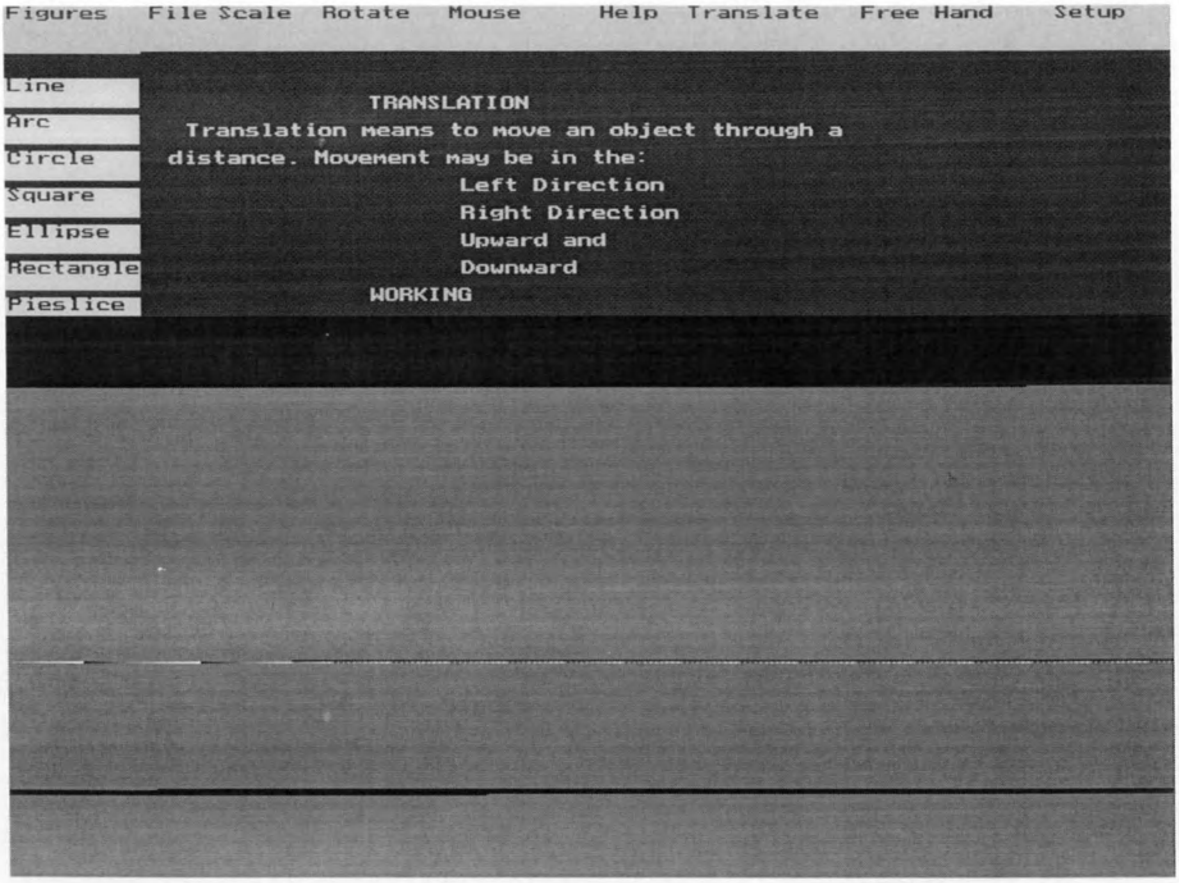
Tips Area

This is the area of the screen where different tips would appear at the required instances of operation of the software. These tips guide the user about the steps needed at different places. It starts at 216th row and zero column of the screen, with a dimension of 74 columns and 264 rows.



```
Message

 After
 reading
 the page
 press
 ENTER
```

```
                 GENERAL
 This Graphic Editor is to design by employing
the geometrical objects. The USER is to interact
through the keyboard. And only the arrow keys
are used for moving the control to different
objects. If a value is to be given, when asked,
it will be an integer value.
 During the operation of the software if an
alphabat- key is pressed, it may generate an
error. In that situation an EXIT from the system
is recommended.
 Working of the software is slow. therefore the
you must wait till the text color of the FIGURES
in the left upper corner becomes RED with white
WHITE background
              DRAWING AREA
 It is working area, to draw in. As the
software is designed in the VGA mode. So the
screen is of 640 columns and 480 rows. Out of
this the drawing area is of 560 columns and
418 rows, starting from the 75th column and
62nd row of the screen.
              MESSAGE AREA
```

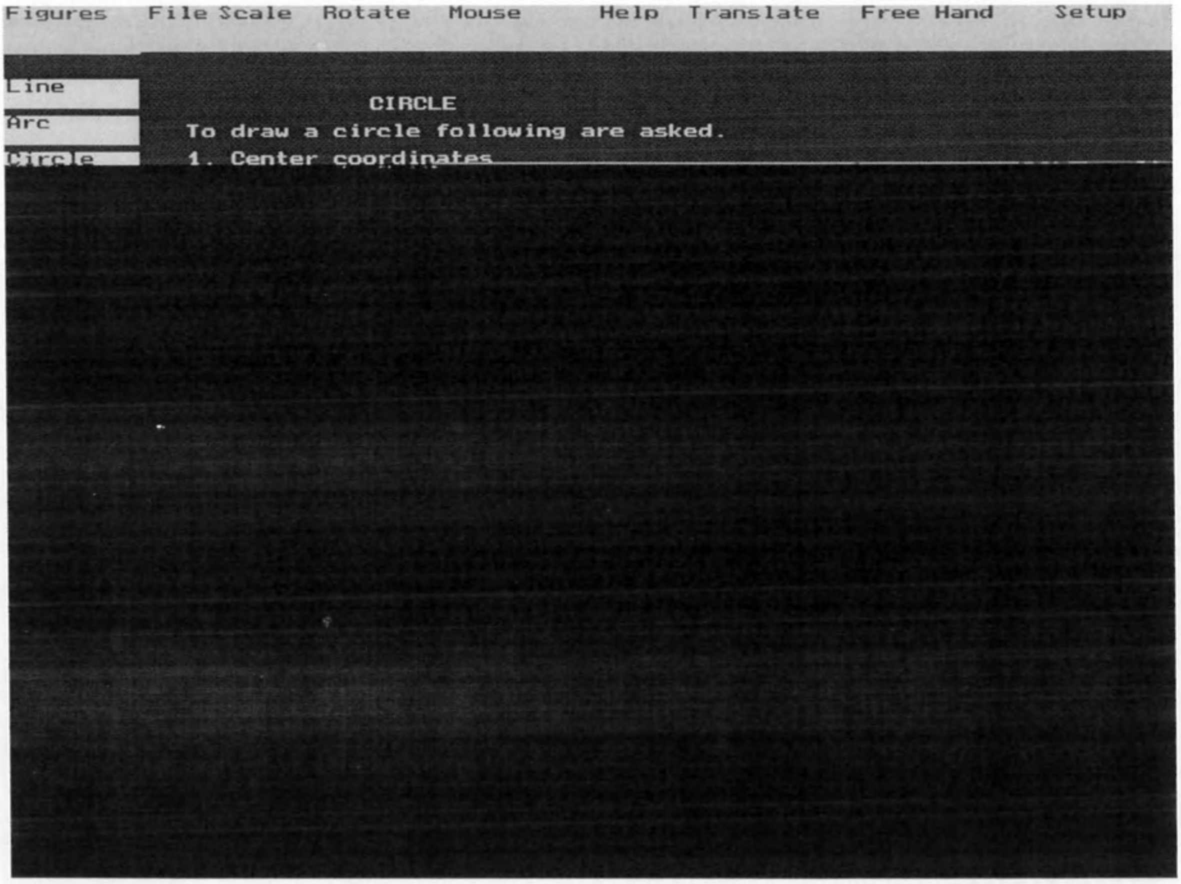


TRANSLATION

Translation means to move an object through a distance. Movement may be in the:

Left Direction
Right Direction
Upward and
Downward

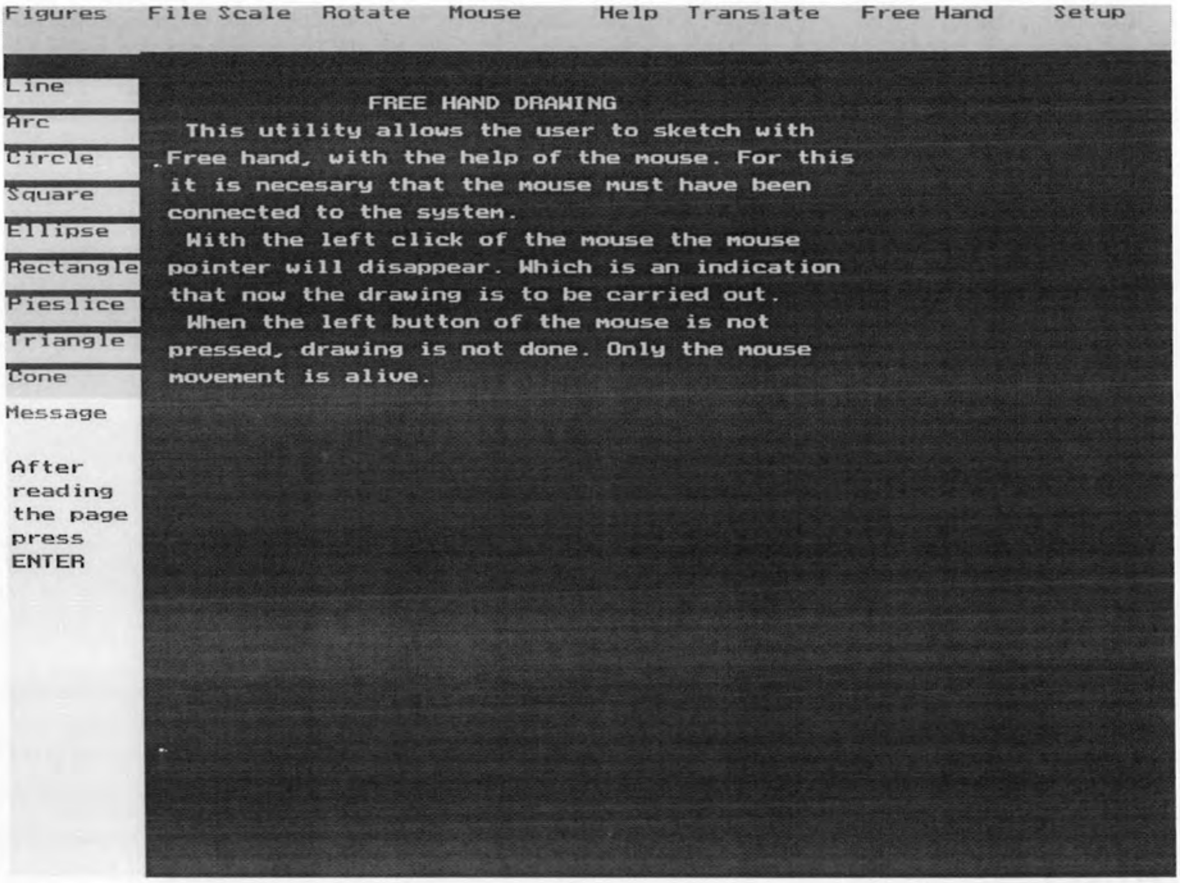
WORKING



CIRCLE

To draw a circle following are asked.

1. Center coordinates



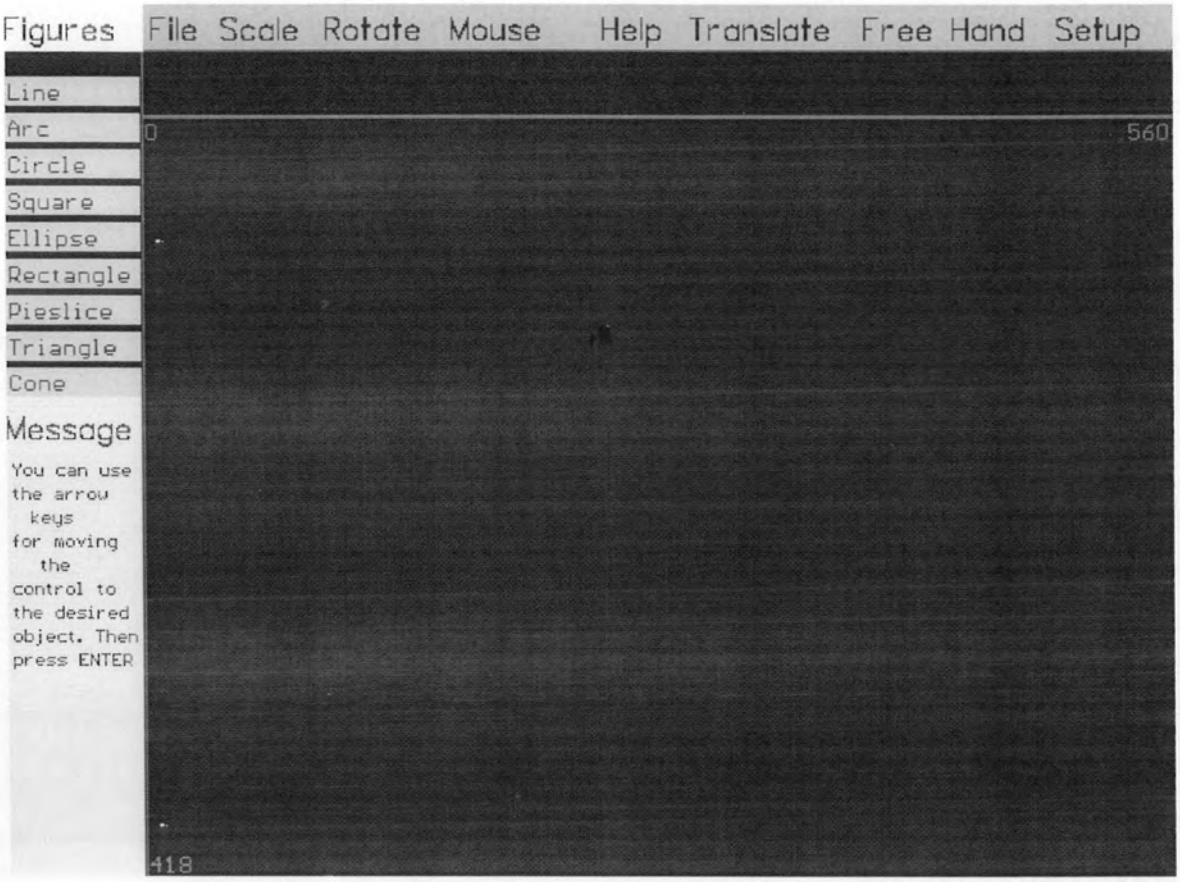
After reading the page press ENTER

FREE HAND DRAWING

This utility allows the user to sketch with Free hand, with the help of the mouse. For this it is necesary that the mouse must have been connected to the system.

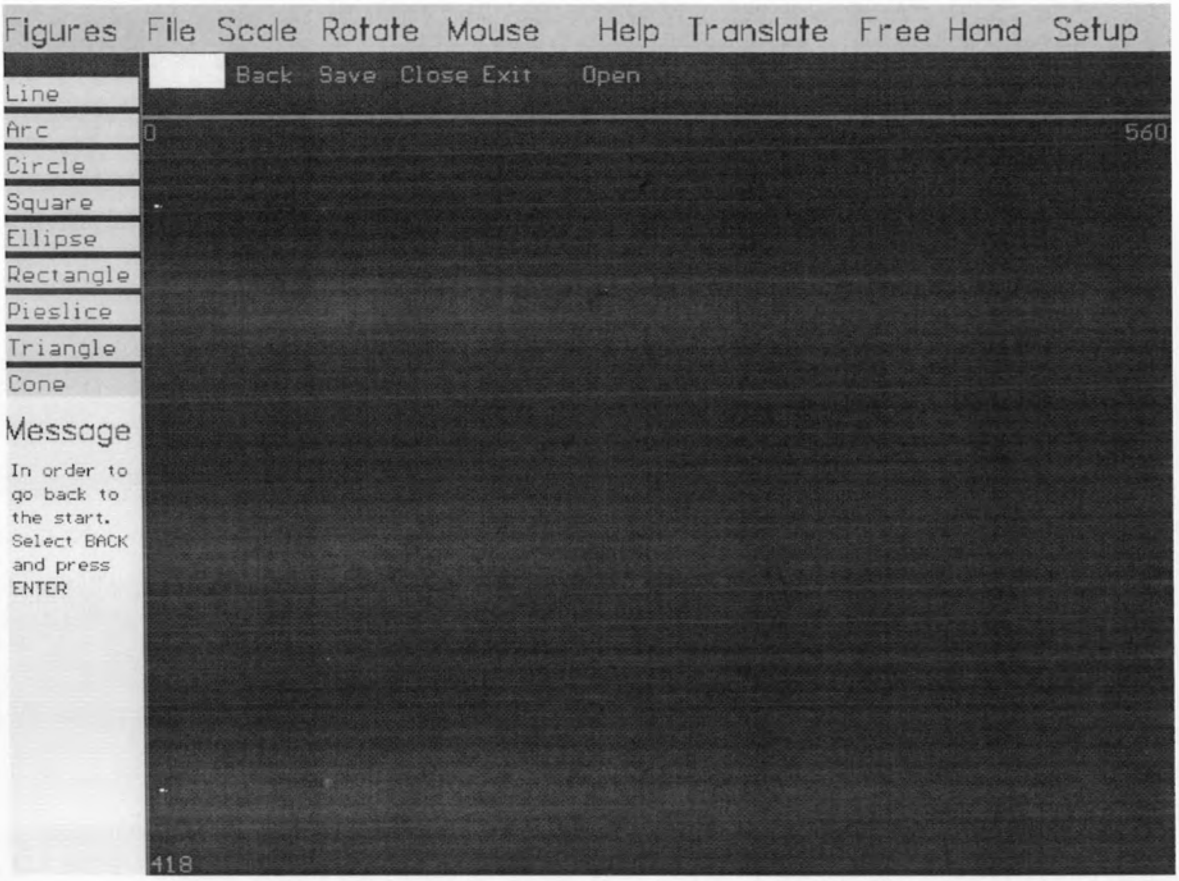
With the left click of the mouse the mouse pointer will disappear. Which is an indication that now the drawing is to be carried out.

When the left button of the mouse is not pressed, drawing is not done. Only the mouse movement is alive.

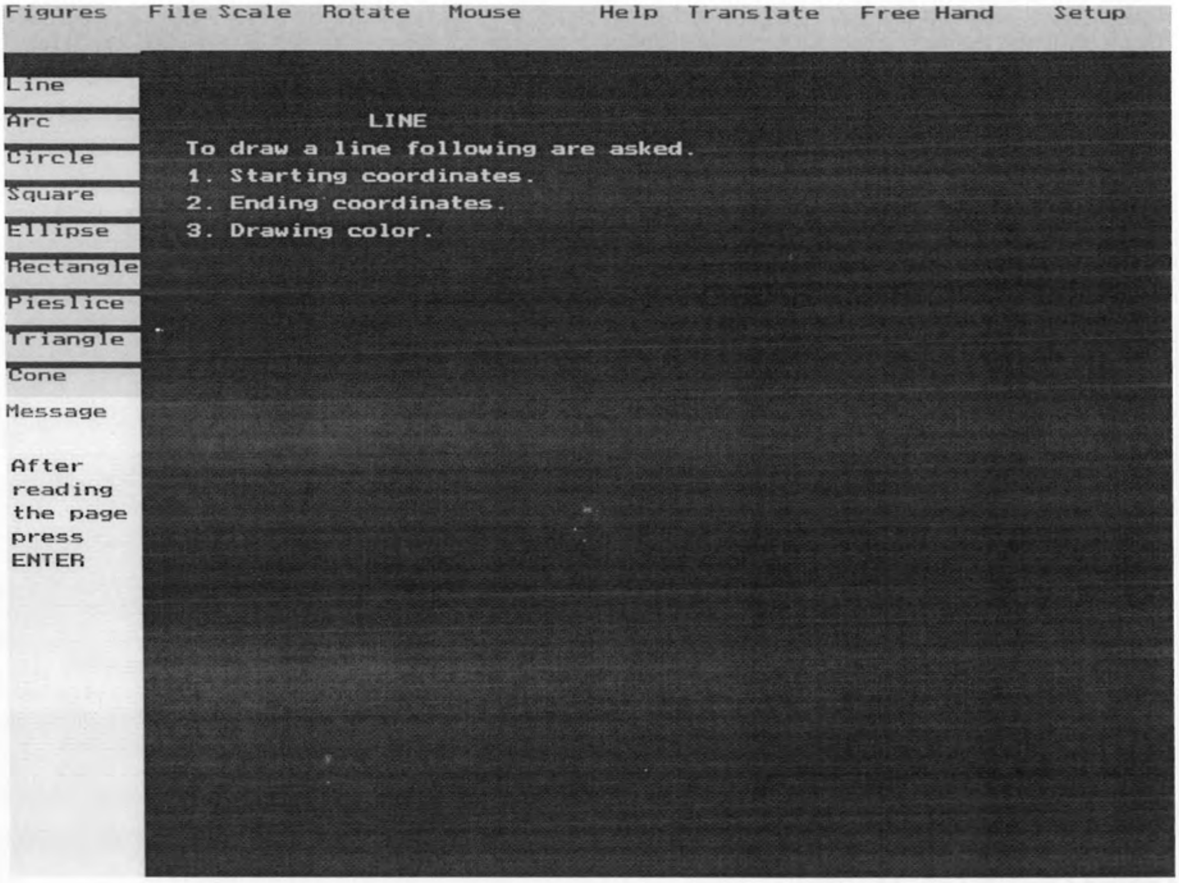


Message

You can use the arrow keys for moving the control to the desired object. Then press ENTER

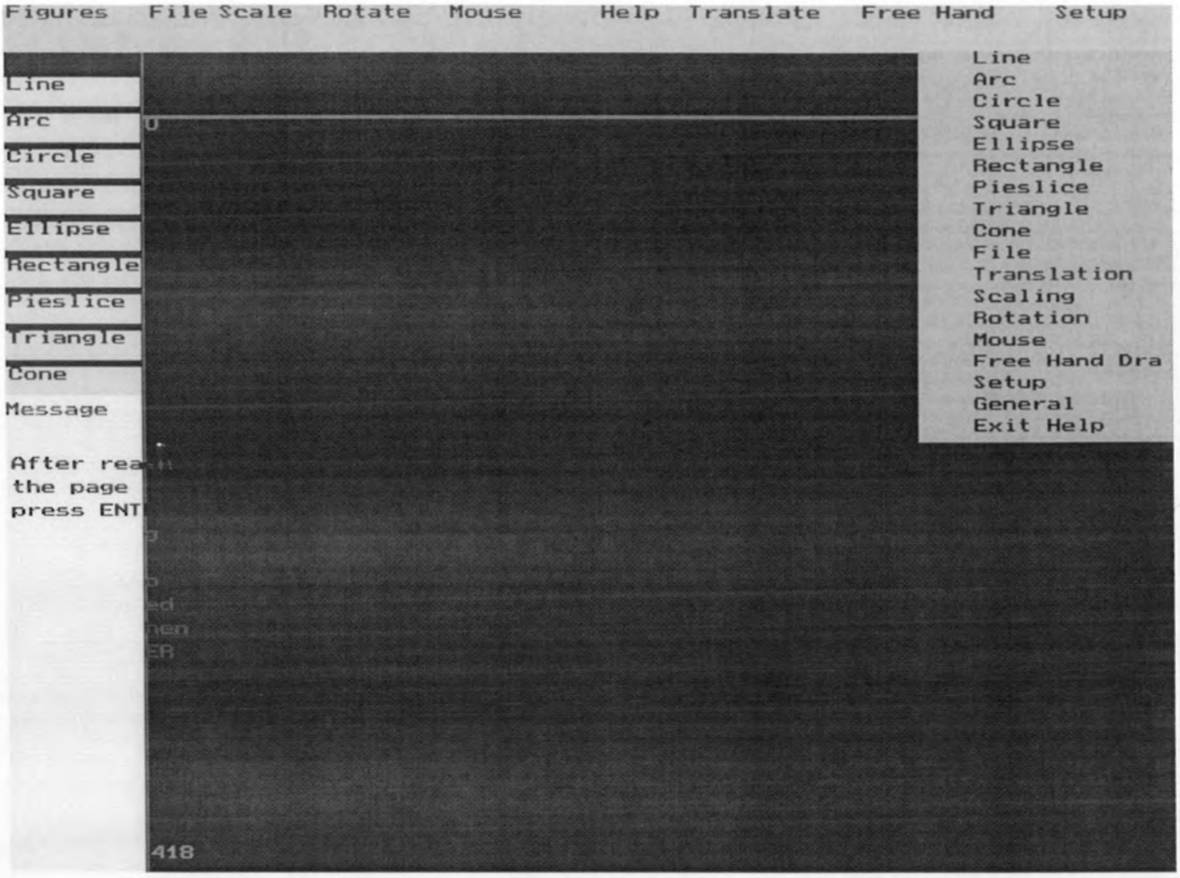


In order to go back to the start. Select BACK and press ENTER

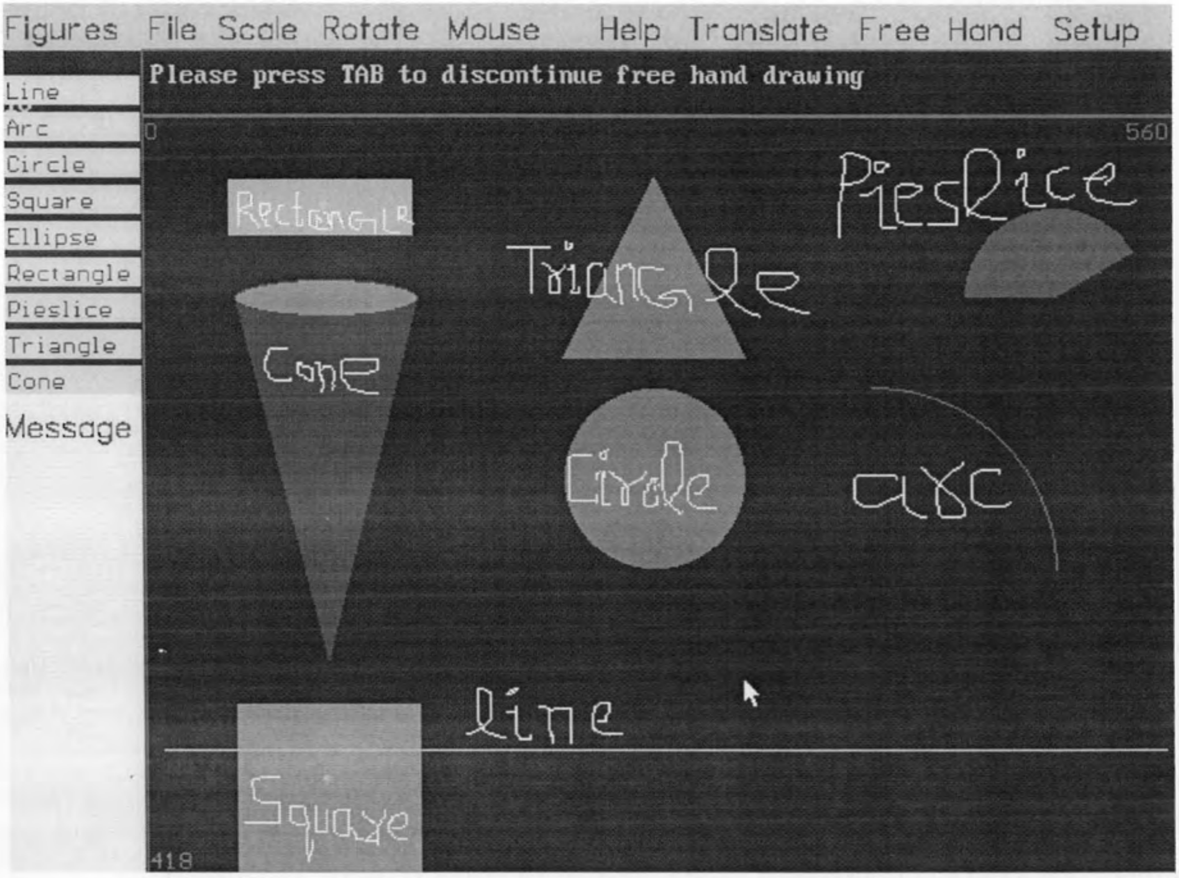


```
                    LINE
To draw a line following are asked.
1. Starting coordinates.
2. Ending coordinates.
3. Drawing color.
```

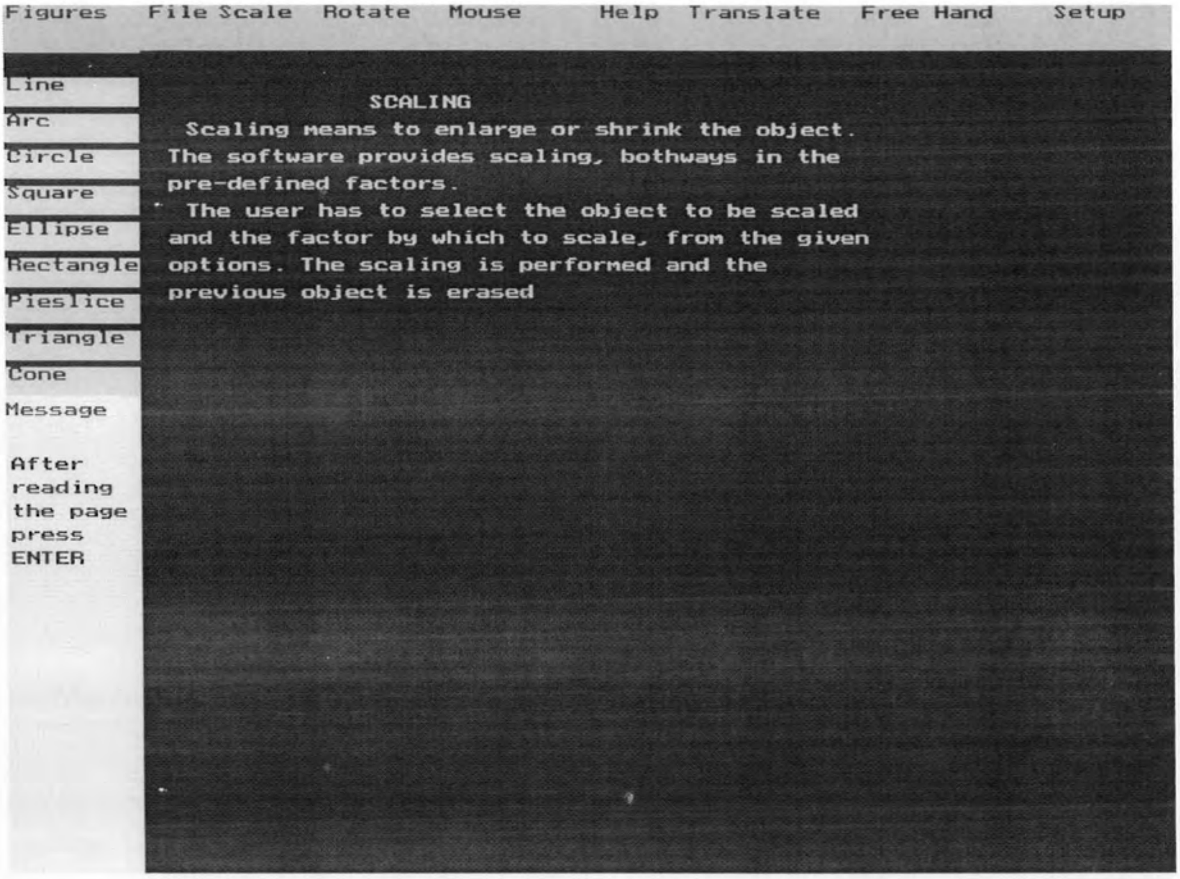
```
Message

 After
 reading
 the page
 press
 ENTER
```



After rea
the page
press ENT



Please press TAB to discontinue free hand drawing

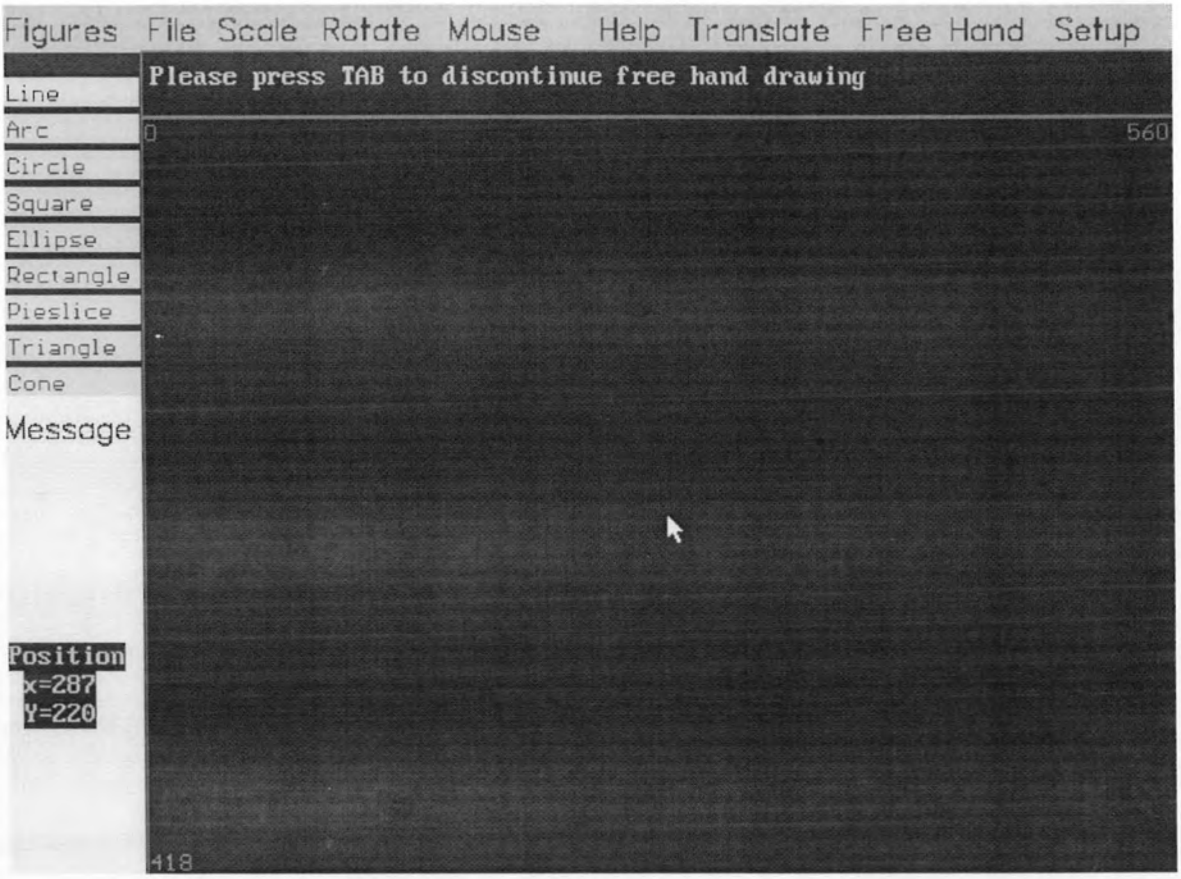


After reading the page press ENTER

SCALING

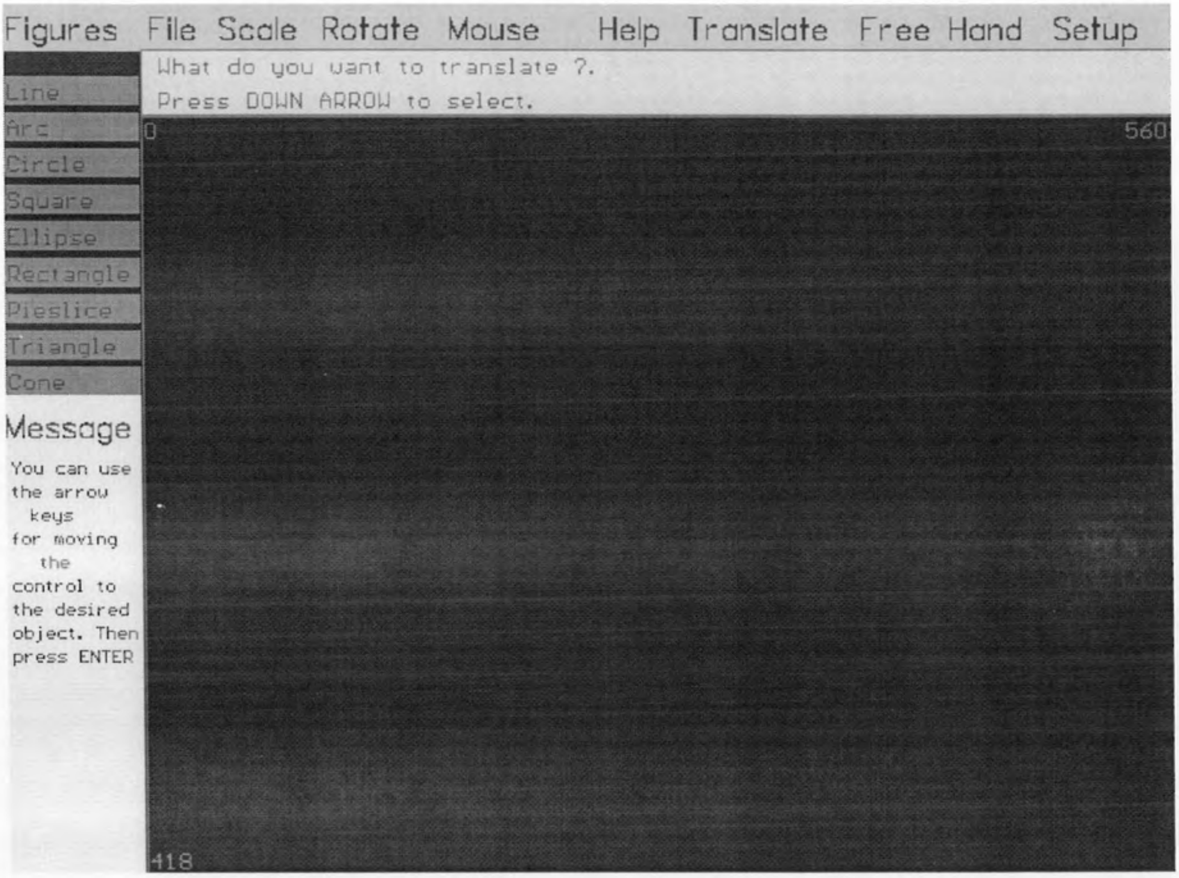
Scaling means to enlarge or shrink the object. The software provides scaling, bothways in the pre-defined factors.

The user has to select the object to be scaled and the factor by which to scale, from the given options. The scaling is performed and the previous object is erased



Please press TAB to discontinue free hand drawing

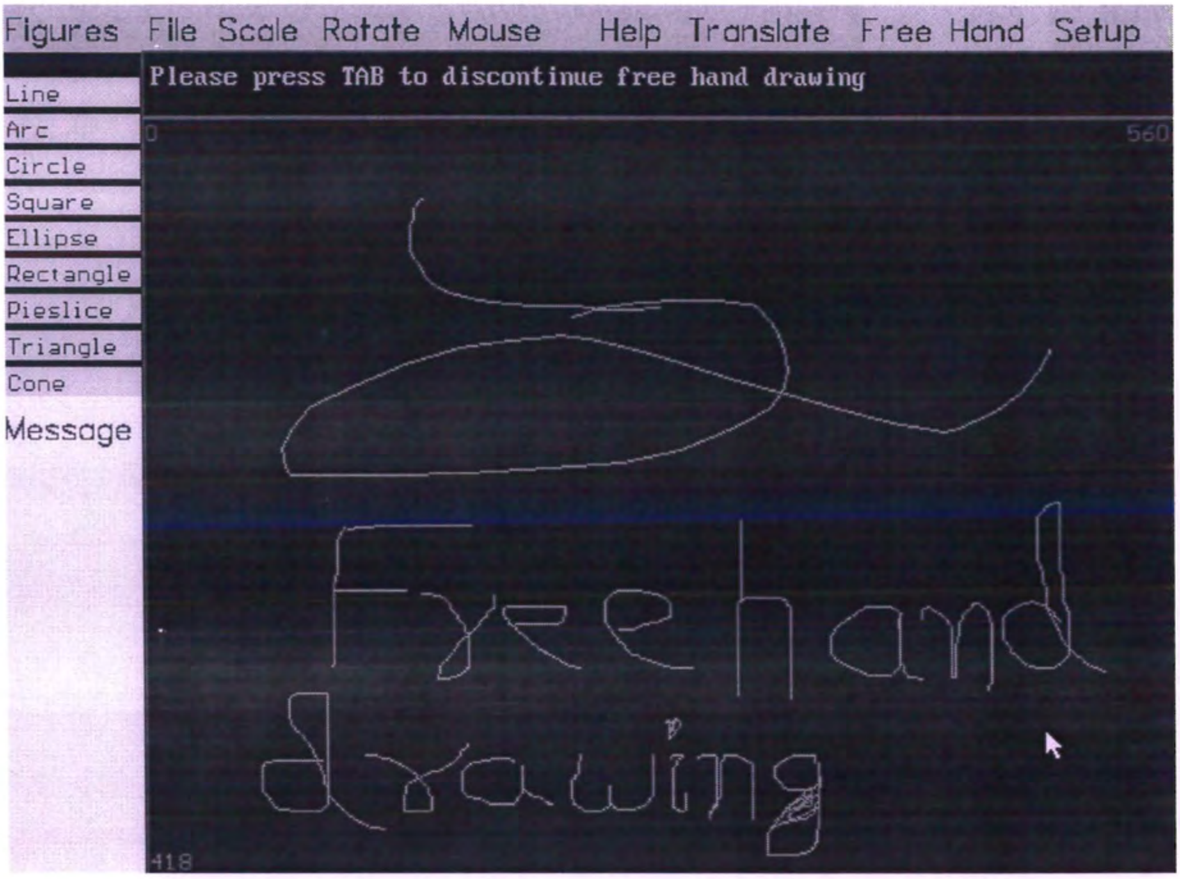
Position
x=287
Y=220



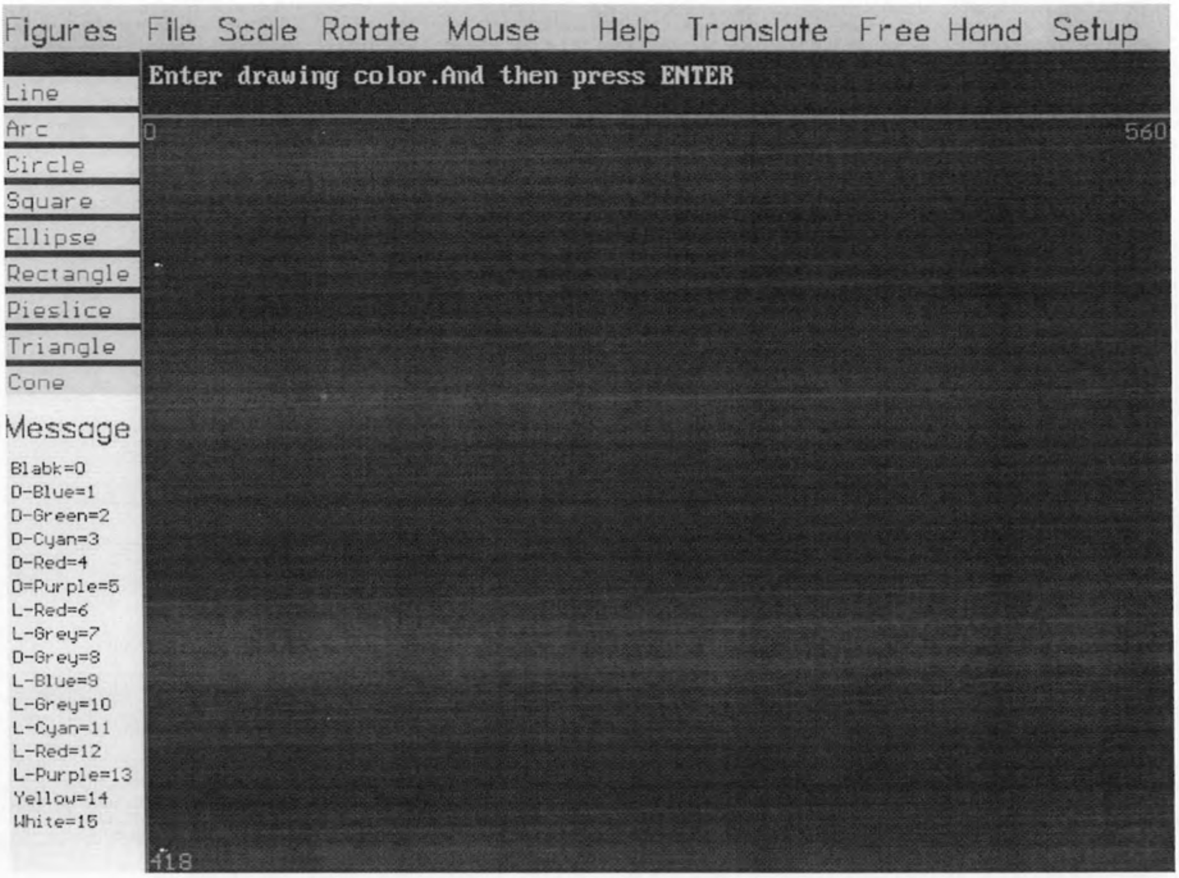
What do you want to translate ?.

Press DOWN ARROW to select.

You can use the arrow keys for moving the control to the desired object. Then press ENTER

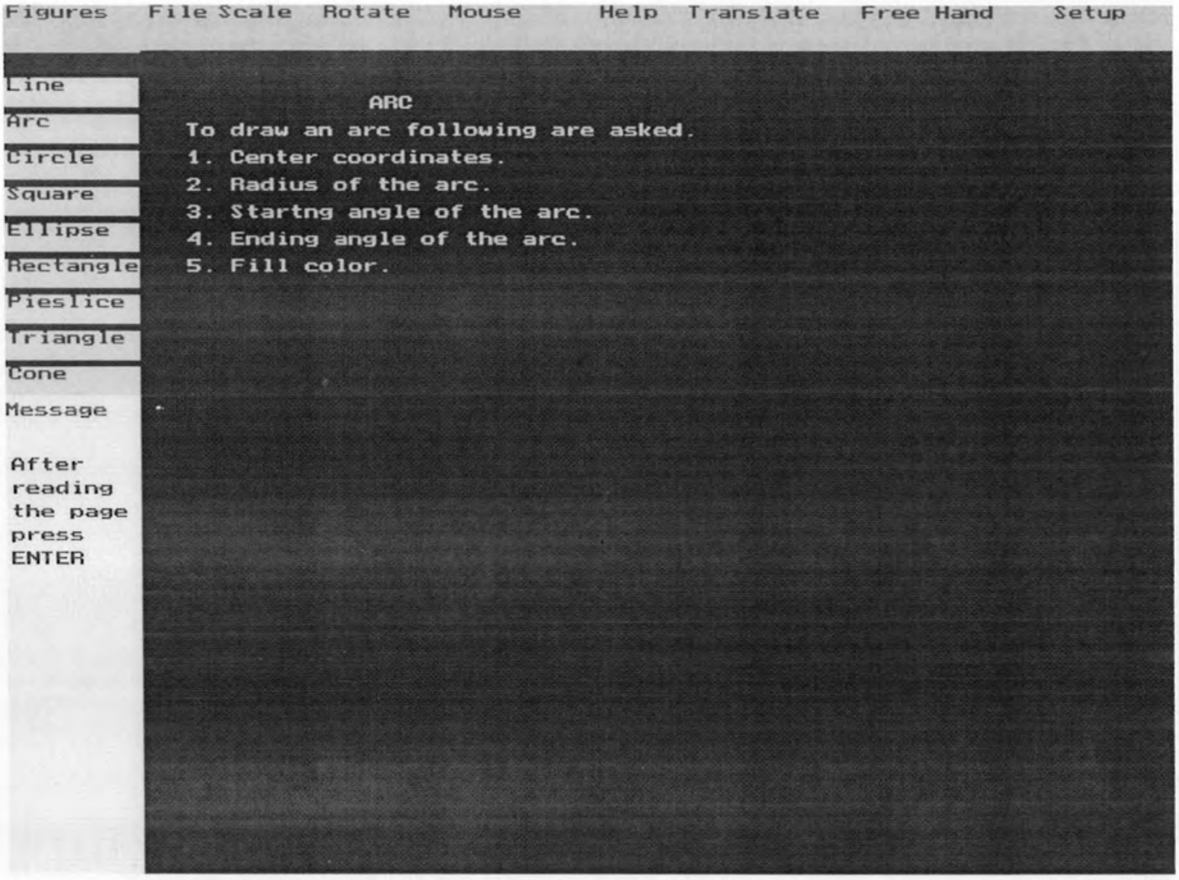


Please press TAB to discontinue free hand drawing



```
Enter drawing color.And then press ENTER
```

```
Blabk=0
D-Blue=1
D-Green=2
D-Cyan=3
D-Red=4
D=Purple=5
L-Red=6
L-Grey=7
D-Grey=8
L-Blue=9
L-Grey=10
L-Cyan=11
L-Red=12
L-Purple=13
Yellow=14
White=15
```



After reading the page press ENTER

ARC

To draw an arc following are asked.

1. Center coordinates.
2. Radius of the arc.
3. Startng angle of the arc.
4. Ending angle of the arc.
5. Fill color.