(1411)

On the Implementation of a Constraint Based Object Oriented Graphics Database Model Using C++

Dissertation submitted to The Jawaharlal Nehru University
in partial fulfilment of the requirements
for the award of the degree of
MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE

72p+fig.

S KARUNAKAR REDDY

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110 067
JANUARY 1994

CERTIFICATE

This is to certify that the dissertation entitled **"On the Implementation of a Constraint Based Object Oriented Graphics Database Model Using C++"**, which is being submitted by **S. KARUNAKAR REDDY** to the **School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi** for the award of degree of **Master of Technology** in **Computer Science**, is a record of bonafide project work carried out by him under the guidance and supervision of **Dr. S. Balasundaram.**

This work is original and has not been submitted, in part or full to any other University or Institution for the award of any degree.

5-1-94

Prof. K.K. BHARADWAJ
(DEAN, SC&SS)

5/1/94

Dr. S. BALASUNDARAM
(SUPERVISOR)

TO MY MOTHER

CONTENTS

INTRODUCTION

INTRODUCTION

1.1 Data Modeling

A database is a collection of stored data together with their description and inter relationships. A Database Management System (DBMS) provides users with a conceptual representation of data that does not include details of how the data is stored. It is here the 'Data Modeling' enters the scene. Underlying the structure of a database is the concept of a datamodel, a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. A data model provides the conceptual representation of the data and hence hides the storage details from the users. The users are blissfully oblivious of the way the data is stored but are aware of the framework of concepts used to express the semantics of database.

The various data models that have been proposed can be grouped into three groups : object-based logical models, record based logical models, and physical data models.

1.2 Object Orientation

The 1990s will undoubtedly be called the decade of object-orientation. The need for this novel paradigm is rather simple. Users are demanding more functionality from their computing systems. They are asking for simpler, easier-to-use computing

environments. Increased functionality and easier to use computing environments come at a price, however. They demand more complex underlying systems. This means more lines of code to be organised, managed, and maintained.

Object Orientation can be loosely described as the software modeling and development (engineering) disciplines that make it easy to construct complex system from individual components.

The intuitive appeal of object orientation is that it provides better concepts and tools with which to model and represent the real world as closely as possible. The advantages in programming and data modeling are many. As pointed out by Ledbetter and Cox (1985) "....object (oriented) programming allows a more direct representation of the real world model in the code. The result is that the normal radical transformation from system requirements (defined in user's terms) to system specification (depend in computer terms) is greatly enhanced.

Object orientation provides better paradigms and tools for (i) Modeling the real world as close to a user's perspective as possible (ii) Interacting easily with a computational environment, using familiar metaphors (iii) Constructing reusable software components and easily extensible libraries of software modules (iv) Easily modifying and extending implementations of components without having to re-code everything from scratch.

Object orientation attempts to satisfy the needs of end users and as well as those of developers of software products.

This is accomplished via real world modeling capabilities. In object-oriented programming the central entities are data (vs procedure or subroutines). The data objects communicate or invoke one another through sending messages to each other. Collection of objects that respond to the same messages are implemented through classes. A class describes and implements all the methods that capture the behavior of its 'instances'. The implementation is totally hidden or encapsulated within the class. In other words, implementations can be extended and modified without affecting the users of the class in any way. Thus, classes contribute both to the modeling of the real world and to software extensibility and reusability. Classes implement a very fundamental concept in object orientation namely abstract data types.

A class is like a module. With object orientation, these modules can be used not only in different applications but also to extend of specialize a class. This is achieved through inheritance. In addition, it allows software modules to share code and representation.

The three main computer science areas that have been irreversibly influenced and affected by object orientation are programming languages, database management system and interfaces.

1.3 Object Orientation in Data Bases

The forerunners of database management system were generalized file routine systems. These file management systems

were able to discretely perform common operations on files, independent of their data. Some examples of these routines include, sorting, file maintenance and report generation. Then came the Network model, then the Hierarchical and finally the Relational databases came into existence. Those to follow viz. the Semantic data models and the Complex object models were mostly research proposals, projects or prototypes. The Complex object-models and the Semantic data models laid a firm foundation for the development of a number of object-oriented databases, both in research and the market place. Concepts such as complex objects, object identity, inheritance and set and tuple valued attributes propagated thus powerful object-oriented database system. Each object oriented database was influenced by one or more of the complex and semantic data modeling alternatives.

The object oriented logical models are used in describing data at the conceptual and view levels. They are characterized by the fact that they provide fairly flexible structuring capabilities and allow data constraints to be specified explicitly. There are many different models, and more are likely to come. Some of the more widely known ones are the entity-relationship model, the object oriented model etc.

The entity relationship model has gained acceptance in database design and is widely used in practice. The object oriented model, includes many of the concepts of the entity-relationship mode, but represents executable code as well as data. It is rapidly gaining acceptance in practice.

A number of research efforts are in progress in creating object-oriented advanced DBMSs based on the object-oriented paradigm. Another direction of object oriented database research is being concentrated on the extensibity of relational system. Prototype efforts such as POSTGRES and STARBUST attempt to provide extensions to the relational system. In the third category of efforts such as EXODUS, system designers are experimenting using a "Toolkit" approach i.e., software tools and libraries to generate a database system specific to the application domain.

1.4 Graphical Data Bases and Existing Packages

The first international standard on computer graphics, the Graphics Kernel System (GKS) [ISO 1985] appeared in 1985. It provides a functional interface between the application program and multiple input and output devices. It is a two-dimensional system. This was followed by a related standard for three-dimensional graphics GKS-3D [ISO 1987] in 1987.

Both GKS and GKS-3D standards enable graphic data to be grouped into segments. Segments can be manipulated as single entities. That is, segments can be deleted, made visible or invisible or highlighted. Once a segment is created its contents cannot be modified. That is, existing primitives in the segment cannot be deleted or modified and new primitives cannot be inserted. Transformations can be applied on segments. These operations affect the entire segment, thereby affecting all the primitives within the segment.

A picture may be constructed from a number of disjoint segments. This is because a segment cannot include or contain references to other segments, thus providing a single-level or flat graphics data structure. Therefore a complex picture is flattened. In contrast to segments, output primitives cannot be addressed or manipulated as individual entities.

Since the structure of a complex picture is hierarchical, not flattened, it is desirable to retain the original hierarchical structure at the graphical level and therefore the segment model of GKS is not sufficient. The Programmer's Hierarchical Interactive Graphics System (PHIGS) standard [ISO 1986] overcomes these problems.

PHIGS is designed to support applications which need highly dynamic displays. The graphical data in PHIGS is organized into structures which consists of a sequence of structure elements. Structures may refer to other structures. This enables handling of hierarchical relationship among graphics data and manipulating geometrically related objects. It provides functions for editing the contents of a structure. But the editing functions are very limited, they only allow structure elements in a structure to be deleted, inserted or replaced. PHIGS does not allow editing the contents of individual elements. For example, it is not possible to edit a point of a polyline or to add a new point to a polyline. To achieve this the entire element must be regenerated.

Both the GKS and PHIGS, entities which are not defined as separate cannot be accessed and manipulated as individual entities. Also, it is not possible to assign names to ouput primitives and to manipulate them as individual items. That is, both systems distinguish between output primitives and segments/structures. It is not possible to assign attributes to a collection of graphics primitives by name or identifier. For example, if we have to assign color "red" to a group of primitives, we have to open a segment/structure and then set the attribute color to "red" and then invalidate the attribute by assigning a color that is different from "red". We cannot use the identifier for this purpose. Also, it is not possible to assign attributes such as visibility, highlighting, etc. to individual primitives. For example, if an individual primitive has to be made visible or invisible, we have to create a segment/structure consisting of only one primitive.

Further, it is not possible to impose constraints on graphical data. For example, we cannot restrict the character size to a minimum for legibility.

The creation and manipulation of individual objects, their geometry and attributes are very important in any graphics systems of today. For this it is necessary to be able to address and access individual objects. Further, it is also important to establish relationship among graphical objects and define operations on them. Object-oriented paradigm as a basic concept provides these features and so it is becoming very popular.

The emergence of object-oriented languages and systems have added a new dimension in the field of computer graphics. The Smalltalk-80 [Goldberg 1983], [Goldberg 1984] is a typical representation for supporting graphics in object-oriented programming environment. It is a raster graphics system and supports direct pixel manipulation. All graphics primitives are objects and can be called by name and refereed by other objects. They can be directly assigned attributes and can be manipulated individually. In addition to graphics primitives, Smalltalk-80 also provides special classes to generate windows, menus and dialog boxes. These features are particularly relevant to the development of user interfaces [Barth 1986], [Myers 1989]. Smalltak-80 graphics systems do not support hierarchical picture structure and therefore it has to be implemented with available object-oriented programming tools.

Small talk-80[4,5] operates in 2-D screen coordinates unlike GKS and PHIGS which operate in world coordinates. Also, unlike GKS/PHIGS it does not provide functions for device independent graphics. Smalltalk-80 does not maintain a database of structures as in PHIGS. Unlike Smalltalk-80, GKS and PHIGS use floating point coordinate systems. In general, Smalltalk-80 does not support the range of functionalities that is expected from a graphics standard. Therefore, it cannot be called complete graphics system and it cannot compete with the graphics standards GKS and PHIGS. They address different needs and applications, and therefore no single graphics package meets all the needs. It is therefore necessary to synthesize between the existing object-

oriented graphics and functionalities provided by graphics standards.

1.5 Constraints

Constraints play a very important role in computer graphics specially in supporting declarative style of graphics programming. However the idea of imposing constraints on graphical data is not new. Sutherland's Sketchpad [6] system was the first to realize the importance of constraints. In this system lines, for example, can be consternated to be horizontal, perpendicular or parallel. Borning's Thiglab[7, 8] advanced a step further by incorporating general constraints and applying them to interactive graphics simulation. ThingLab-II[9, 10] is a constraint programming system to construct user interfaces. The constraints are used to map between application objects and graphical objects on the display screen, to maintain consistency between multiple views of data, window layout and animation. These are implemented in Smalltalk-80. In Nelson's JUNO graphics system[11], constraints are allowed to be specified by selecting icons and points. There are four types of constraints possible; making two lines of equal length, two lines parallel, and a line horizontal or vertical. The novel feature of this system is that interactive editing of any displayed image results in modification of the text of the program accordingly.

Constraints can be classified into one-way and multi-way constraints. In multi-way constraints, when two or more objects

are attached by a constraint then modification of any object implies the modification of some or all the other objects again satisfying the constraints whereas one-way constraints allow modification of only one object so that the other objects are also modified satisfying the constraint. Both Sketchpad and Thiglab are examples of multi-way constraint-based systems. For one-way constraint-based system we refer to[12,13]. Multi-way constraints are more difficult to implement than one-way constraints. Also they can introduce ambiguity at the design level. The theory of constraints hierarchy was introduced by Borning et al[14] to eliminate this ambiguity. Vander Zanden[15] proposed a new constraint-solving algorithm to solve multi-way constraints in real time at the user interface development environment. Leler's Bertrand[16] is a constraint language that is used to solve systems of constraints.

In the constraint-based system mentioned above, constraints are used to control the layout of objects on the screen. These systems allow constraints to be imposed on the geometry of objects and on the relationship between two or more objects. Also, these systems do not consider constraints that are important in this framework. As these systems are not database systems they do not impose constraints when new objects are inserted in or deleted from the database, or are modified.

1.6 Declarative Approach

In any graphics application, pictures are typically represented procedurally as a series of calls to graphics drawing

routines (or commands) which manipulate pixel areas of raster device. In the declarative approach one specifies what has to appear on the screen and consequently what modifications are required, rather than how the picture is displayed and modifications are performed. The picture contents are described in terms of objects that are to appear on the screen. The user manipulates objects of the picture, rather than manipulating pixel areas. For example, if an object has to be moved from one portion of the screen to another, in the declarative approach one needs to assert the new location rather than to erasure the object and to redraw it.

The main advantage of the declarative approach is that the modifications of the picture contents are easier to perform as compared to the procedural approach. For example, in the case of modification of height of a pyramid the user has to calculate the new location of the fifth vertex in a procedural approach. If the fifth vertex lies on the base, then two actions can be taken. One is checking before modification and the other is to allow modification and sending an error message at the time of displaying. In the declarative approach, we represent a pyramid by specifying the size of the base and its height. With this approach it is much easier to ensure that height never becomes zero.

In PHIGS the graphical data is organized into structures which are collection of structure elements. It provides functions for editing the contents of structures. That is, the

structure elements in a structure can be deleted or inserted. But it does not allow editing the contents of individual elements. For example, it is not possible to modify the number of points in a polyline. To achieve this the entire element must be regenerated. Entities can be accessed and manipulated only when they are defined as separate entities. Also, it does not allow constraints to be imposed on graphical data. For example, we cannot restrict the character size to a minimum for legibility.

EXPLANATION

2.1 Motivation

Many graphics system (mentioned in section 1.4) address different needs and application and therefore no single system meets all the requirements. They cannot be called complete systems and cannot compete with the graphics standard. Thus it is necessary to synthesize the existing object oriented graphics and functionalities provided by graphics standards.

This is an effort to integrate the declarative approach to graphics and object oriented data modeling techniques to form a fruitful symbiosis for constraint based graphics database system[18,19]. Efforts are made to imbibe rich modeling contents to describe graphics data and allows sharing of representation. It also consists of successful implementation of integrity constraints. This implementation is done in C++.

We present a general purpose constraint-based graphics database system. In our scheme all graphical entities are treated as objects, which can be addressed and manipulated independently. Objects are supported as instances of classes, which in turn, are organized into 'is-a' hierarchy(refer to the HIERARCHY FIGURE). The declarative approach to graphics allows us to represent and manipulate objects at the same level of abstraction at which the user thinks but not at the level of pixels or drawing routines. It allows us to specify the contents of the picture in terms of objects that are to appear on the screen. The contents of the picture are stored in the database. This enables us to share representation of pictures. Further, in

our scheme graphics data is specified in screen coordinates. We allow constraints to be imposed on individual objects. Our work can be successfully extended for scientific modeling and computer-aided design in mechanical engineering.

In our package, each graphical object can addressed by name of identifier. Attributes can be assigned to objects as part of their own inherent information. Therefore the values of the attributes can be inquired directly. Attributes of an object can be selected and manipulated individually. That is, they can be deleted, modified, transformed etc. Thus, even primitives can be edited. An object may be referred by any number of objects. Objects with the same structure (attributes) and behavior (operation) are regarded as instances of one class. Classes are organized in a class hierarchy called 'is-a' hierarchy. The root of the hierarchy is SHAPE.

Class SHAPE has display attributes such as color, linestyle, in addition to spatial attributes. Primitives such as line, rectangle, triangle, polygon, text, circles etc. are created as object which are instances of subclasses of classes of respective names. Even HELP is also a class, which acts as a help command for the user. The user constantly uses this class or its instances to navigate through the picture. We provide separate classes for each type of primitive. Class hierarchy allows sharing of methods by related primitives. It also helps in setting default values for undefined attributes. Operations such as scale, rotate and translate are inherited by the subclasses

through class hierarchy. Operations such as calculation of area, perimeter etc., can also be performed.

2.2 Working

The package when activated, displays a help menu on the screen (refer to the PRIMARY MENU figure). The names of different primitives which can be drawn are displayed at the top of the-screen, horizontally, starting from LeftTop to RightTop. They are HELP, CIRKL, TRIA, RECT, POLY, and TEXT. Also there is a vertical pull down menu against HELP, starting from LeftTop to LeftBottom. The different help functions that are available on this pulldown menu are draw, move, rotate, scale etc.

Any primitive can be selected by pressing the arrow Keys appropriately, and RETURNing. The pull down help menu is replaced by a secondary help menu which displays the different types of values to be entered (refer to the SECONDARY MENU figure). If for example, a primitive CIRKL and the function DRAW is a selected, then the pull down menu asks for the 'location', 'Radius', 'line color', and 'fill color', in the secondary help menu (again refer to the SECONDARY MENU figure attached). When the appropriate values are entered, a circle appears on the screen and again the secondary pull down help menu is replaced by the primary pull down help menu (refer to the figure with the circle, attached). Now if we want to move the circle, we have to select the function MOVE by using the 'arrow keys' and RETURNing it. Again a secondary pull down menu, asking for the new location, appears in place of the primary help menu. Once the new

location is supplied, the circle is moved to the new location and the menu is replaced with the primary help menu. In this way all the primitives can be drawn and any other transformation can be done on these. (Different pictures have been drawn, whose figures are attached, entitled figure 1, figure 2, figure 3) This is in the first phase.

In the second phase, once different primitives are drawn i.e. once the picture is complete, the user is free to store it. The picture is stored in a segment. A new file is created and the different primitives with their respective attributes are stored in the file. The user is free to redraw the stored picture, and even perform any future modifications.

In the third phase, the important service the package offers is the inherent constraints it offers while drawing the primitives. There are different types of constraints. Under the intra-object constraints, one cannot draw a circle with radius less than or equal to zero. Also a circle cannot be drawn with its location near the border of the screen and a radius which makes it cross the border i.e., once the location of the circle is fixed, the maximum value of radius gets automatically fixed. These are one type of constraints. Also, we can impose inter object constraints (i.e. one way and multiway constraints). For example once the location and radius of a circle are fixed, the location and radius of the second circle are partially fixed, if we define that the two circles say, form the two wheels of a bicycle.

2.3 Code

The package consists of five files, one calling the other four. It also includes some library header file. The Library Files which have been used are <graphics.h>, <conio.h>, <math.h>, <stdlib.h>, <stdio.h>, <iostream.h>, <string.h>, <dos.h>.For obvious reasons (because the whole effort itself is to build a graphics package), the 'Graphics File' has been used.'Conio.h' has been included because it declares various functions used in calling the DOS console I/O routines. And 'Dos.h' has been included because it defines various constants and gives declarations needed for DOS and 8086-specific calls. The 'iostream.h' declares the basic C++ (version 2.0) streams (I/O) routines. The 'math.h' file has been included to calculate the different values of the 'primitives'.'stdio.h' defines types and macros needed for the Standard I/O package defined in Kernighan and Ritchie and extended under UNIX system V. It defines the standard I/O predefined streams stdin,stdout,stdprn, and stderr, and declares stream-level I/O routines.'stdlib.h' declares several commonly used routines:conversion routines, search/sort routines, and other miscellany.'string.h' has been included because it declares several string- and memory-manipulation routines.

The five files that have been built and used are, the main file, the menufile, the primitive file, the erase file and the shape file. The mainfile includes the rest four. The menu file consists of different menu being defined in a hierarchy. The primary pull down help menu and the secondary pull down help menu

are defined separately. The primitive file consists of different primitives. The erase file consists of functions that help in erasing the menus. The shape file consists of the root class SHAPE defined, with its attributes and methods.

The class SHAPE is defined as follows :

```
        location : POINT
        line color : String
        fill color : String
        line style : Integer
        line width : Integer
    operations
        set (s:SHAPE) ----> SHAPE
        draw(s:SHAPE) ----> SHAPE

class POLYGON
        points : Integer
        location : POINT
        line color : Integer
        fill pattern : Integer
        fill color   : Integer

    operations
        draw (p:POLYGON) -----> action
        move (p:POLYGON) -----> action
        enlarge (p:POLYGON) -----> action
        rotate (p:POLYGON) -----> action
        erase (p:POLYGON) -----> action
        paste (p:POLYGON) -----> action
```

change line color (p:POLYGON) -----> action

change fill color (p:POLYGON) -----> action

change fill style (p:POLYGON) -----> action

class CIRKL

location : POINT

radius : Integer

line color : Integer

fill color : Integer

operations

draw (c:CIRKL) ----> action

move (c:CIRKL) ----> action

enlarge (c:CIRKL) ----> action

rotat (c:CIRKL) ----> action

erase (c:CIRKL) ----> action

paste (c:CIRKL) ----> action

change line color (c:CIRKL) ----> action

change fill color (c:CIRKL) ----> action

NOTE : The other primitives RECT and TRIA are defined similar to CIRKL.

class LINE

location : point

line color : Integer

line style : Integer

line width : Integer

operations

draw (l:LINE) ----> action

move (l:LINE) ----> action

enlarge (l:LINE) ----> action

rotate (l:LINE) ----> action

erase (l:LINE) ----> action

paste (l:LINE) ----> action

class TEXT

location : point

direction : Integer

color : Integer

char size : Integer

operations

write (t:TEXT) ---> action

move (t:TEXT) ---> action

change color (t:TEXT) ---> action

Such a representation can be easily be extended to 3D. For this every class and primitives have to be defined appropriately in the 3D and the attributes and operations get transformed appropriately.

REMARK : The user is expected to learn only a few commands. To select, one has to bring the cursor to the appropriate and RETURN. To move the cursor, the 'arrow key' are used. To come out of the package one has to press 'ESC'.

As far as storing the picture is concerned, the 'store' function is selected after drawing each figure, and that primitive is stored. The attributes of the primitive with an appropriate code for respective primitive is stored in a segment. More than one segment can be opened at a time, and the primitives can be appropriately stored. The 'redraw' function is to be selected to draw a stored picture. The menu will display the names of different stored pictures. Once the required picture is selected, the package draws the picture. Now modifications can be performed on this picture and again the modified picture is stored. This can be repeated for any number of times.

Next the menu creation and menu erasure are done in different files. Menu creation is through calling the functions that displays various rectangular blocks with appropriate text written in it viz. 'HELP', 'CIRKL' etc and, 'Draw', 'Move' etc, in different colors as defined. Erasing of Menu is done using the principle of redrawing the menu with the text in black color so that it disappears in the background color. Here it is obvious that the screen color is 'black'.

The menu creation file consists of collection of library functions for displaying rectangles and outputting text in each rectangular box. For example,

```
rectangle(x1, y1, x2, y2);
outtextxy(x1+n, y1+n, "TEXT");
```

first creates a rectangle with (x1,y1) being the left-top corner and (x2,y2) being right-bottom corner of the rectangle and writes the text TEXT starting at a point (x1+n, y1+n) (where 'n' is an integer) such that the text fits properly into the box.

Again, if the color of the rectangle is not specified, the rectangle is drawn in the default (white) color. But to have a different color, the function,

```
setcolor(color);
```

is to be called, just before the rectangle is drawn. The system, when encounters this function, changes the default (white) color to the new color and all the figures drawn are and text written is, in that color until it encounters a similar function assigning different color.

To move the curser on the menu, 'arrow keys' are used. Cursor jumps from one rectangular box to the other (or in other words from one text to the other). In order to differentiate a selected text from a non-selected text, RED and GREEN colors have been used. The selected text is always in GREEN and the non selected will be in RED. Hence, for example, if 'CIRKL' is selected from the primitives and 'Draw' is selected from the functions, 'CIRKL' and 'Move' are displayed in GREEN and the rest text i.e. 'RECT', 'TRIA' etc. primitives and 'Move', 'Rotat' etc. functions are displayed in RED. Hence, the mere, RETURNing will force the package to be ready to 'draw' a 'circle'.

The 'shape file' is the most important file, next to the 'primitive file'. It consist of the basic functions which use the library functions that just assign the different styles and fonts to the text being written and the figures being drawn. For example, the main function in the shape file (which consists of SHAPE class only), is Draw () which is defined as follows :

```
Draw()
{
 setcolor(color);
 fillcolor(fcolor);
 settextstyle(font, direction, charsize);
 setfillstyle(pattern, color);
 setcharsize(multx, divx, multy, divy);
 };
```

The function setcolor() assigns the defined 'color' to all the figures and text. Setfillcolor() fills the figures with the color 'fcolor'. Settextstyle() sets the text style in which the arguments cater to the obvious needs. Setfillstyle() goes with fillcolor() by filling the figure with the defined color and in a pre-defined style. Setcharsize() sets the size of the characters of the text both in the horizontal and vertical direction. The next important collections of functions defined in shapefile is set(). These functions assign the values of the variable to its appropriate variable. For example,

Set (x1, y1, w, Red, Blue) is a function for the Class CIRKL. (x1, y1) is the location, w, the radius, Red, the color of the outline, and Blue, the fillcolor. The set function is defined as follows,

```
set (x1,y1,w,   Red,    Blue)
  {
     X1=x1;   Y1=y1;    W=w;
     Color=Red;   fcolor=Blue ;
  };
```

Here the values x1, y1,w, Red, Blue are assigned to the appropriate variables, which are used in the function Draw().

Hence, the functions set() and Draw() are called together to draw any required primitive.

The whole set of functions when arranged in an order, look as follows :

```
(1) set(x1, y1, w, Red, Blue)
    { X1=n1; Y1=y1; W=w; Color=Red; fcolor=Blue;}

(2) shape :: Draw()
    {
      setcolor (color);
      fillcolor(fcolor);
      - - - - - -
      - - - - - -
    }

(3) circle (X1, Y1,W);
```

The first function i.e. set() assigns values to the variables, the second function i.e. Shape :: Draw() calls the 'color' functions and the third i.e. circle(), draws the circle, at the location X1,Y1 (or x1,y1) with the radius equal to W (or w), the outline color being color (Red) and filling color being 'fcolor' (Blue). Similar methods are used for other functions like 'Move', 'Rotat' etc, and for other primitives too.

The main file 'includes' the rest four files, and consists of the definition of the different classes. This file acts as the mother file having an infinite loop in which it calls the methods of individual classes through its respective objects.

CONCLUSION

CONCLUSION

Our aim of a fruitful symbiosis for a constraints based graphic database system is achieved to a major extent.

Our package obeys the orders of the users to the extent of drawing pictures, performing different types of transformations (viz. Scale, Move, Rotate etc), storing the pictures, recalling the same, editing the pictures, re-storing the same etc. Intra-object constraints are inherently imposed on the objects.

We did face some problems before we started and while going through the project. The foremost one was, whether to use windows or not. Though the package, if created in windows, will have obvious advantages of easy use, easy modification and a less chance of getting stuck. But the factor which made us not to go for windows, is that our package will not run if windows is not available to the user. Hence we did the entire project using just Borland C++. But we are trying to create our package in windows too, which, in addition to others, facilitates the use of a mouse too.

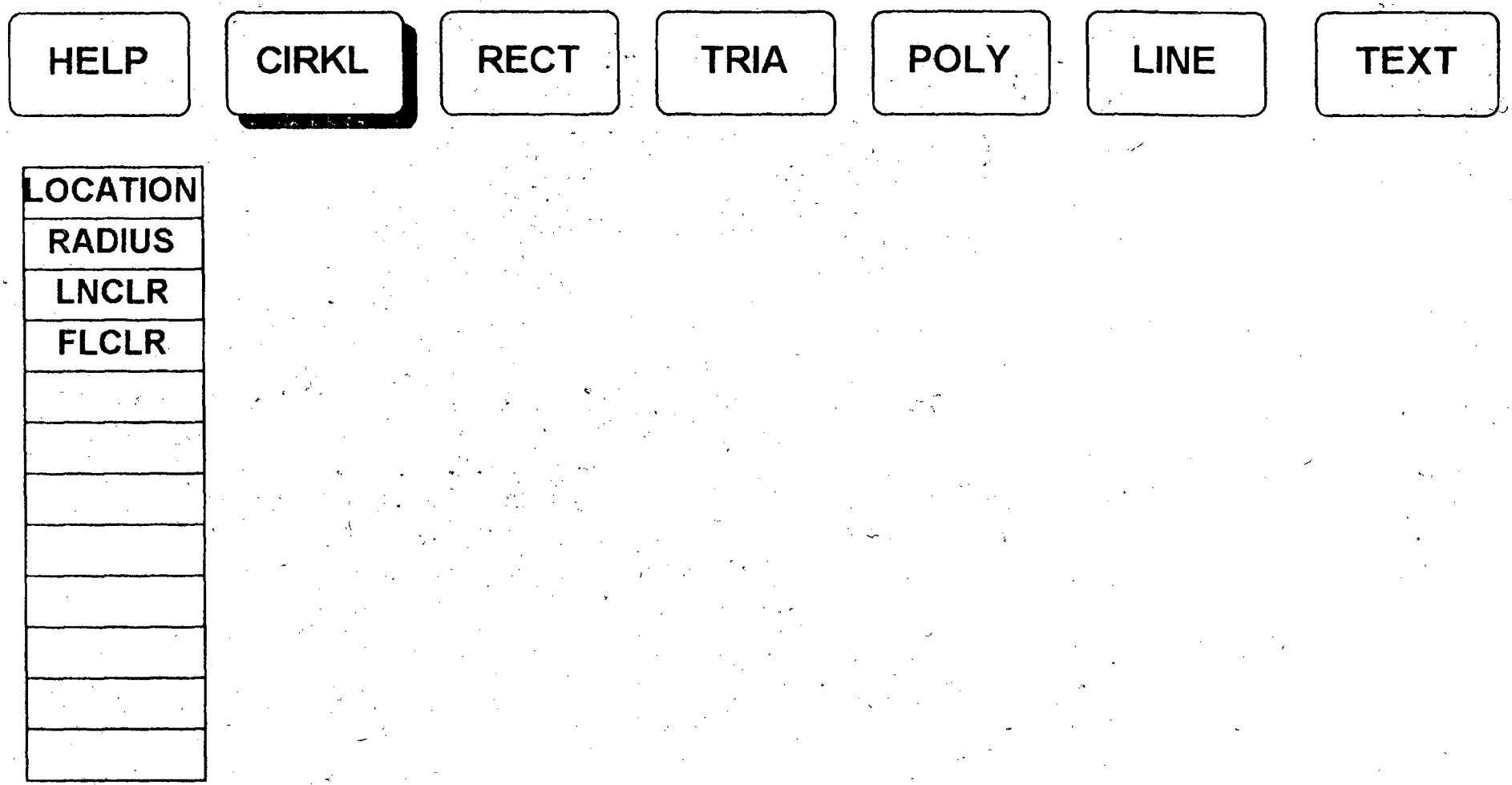
Creating multiple segments was the next important problem which still haunts us. In our package one can open a single segment, enter different values in it and close it before we open a second one. We are trying to crack this problem too, when we use windows for our package.

'Intra object constraints' is an easier concept compared to Inter object constraints (which include one-way, multi-way constraints), which is our third important problem. Even certain inter object constraints like, if a single primitive in a segment is moved, then all the other primitives move in a similar relative distance and in a relative direction, are easier compared to, imposing the constraints of location and dimensions on a second primitive once the first primitive is drawn. For example, while drawing a bicycle, the radius of both the wheels remain same. Hence, once the front wheel is drawn, the radius of the second wheel gets automatically fixed (unless the bicycle is not an ordinary commercial bicycle but a fashionable old cycle). This problem is still a hard nut to crack.

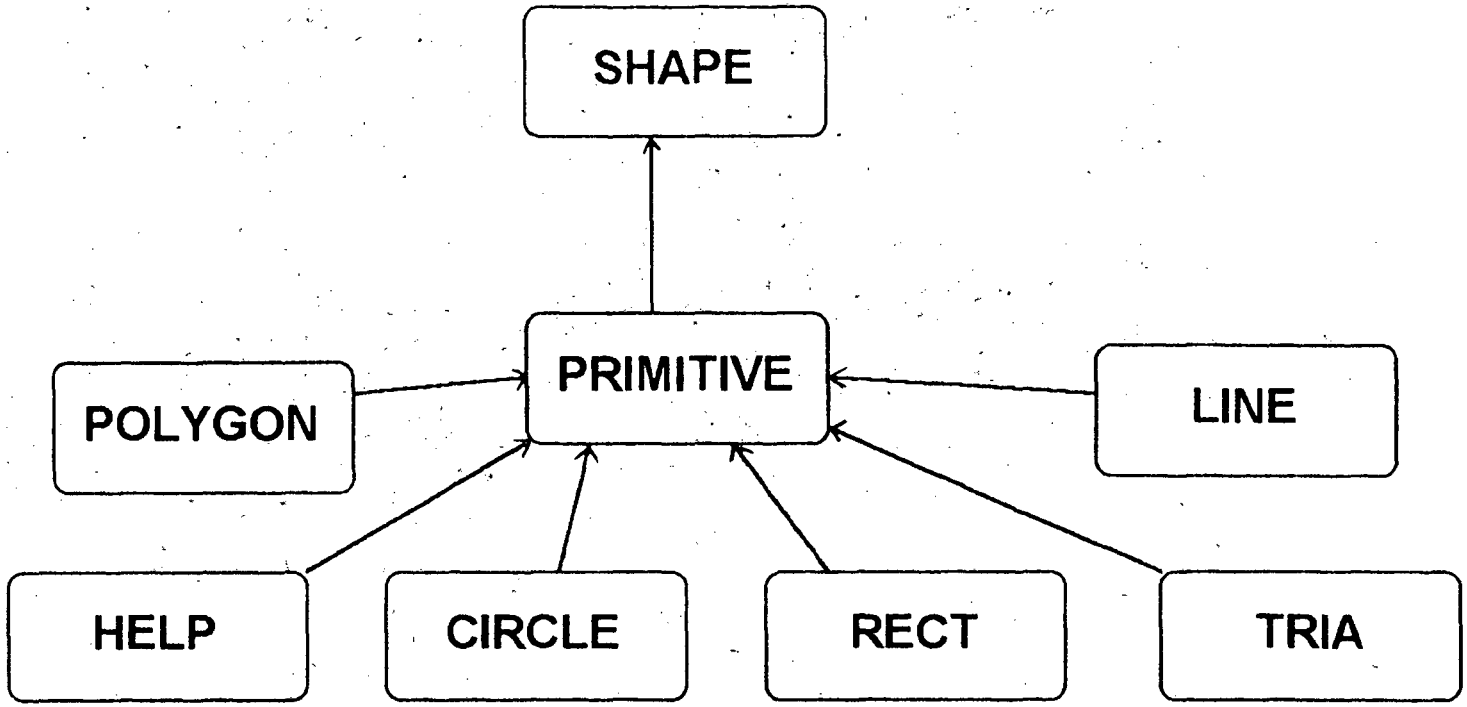
This package, when it overcomes the above three problems, can become an important competitor to certain commercially existing packages like GKS-2D, PHIGS etc. We still entertain the idea of making it so.

PRIMARY MENU

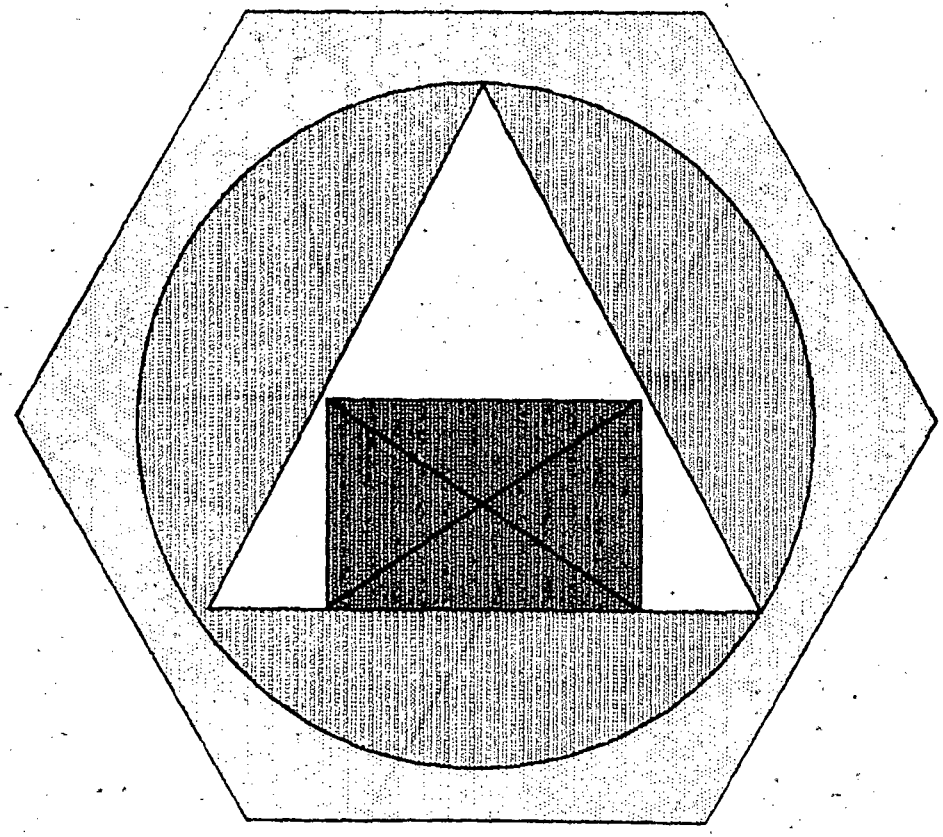
SECONDARY MENU



HIERARCHY FIGURE



ALL TOGTHER



VIEW

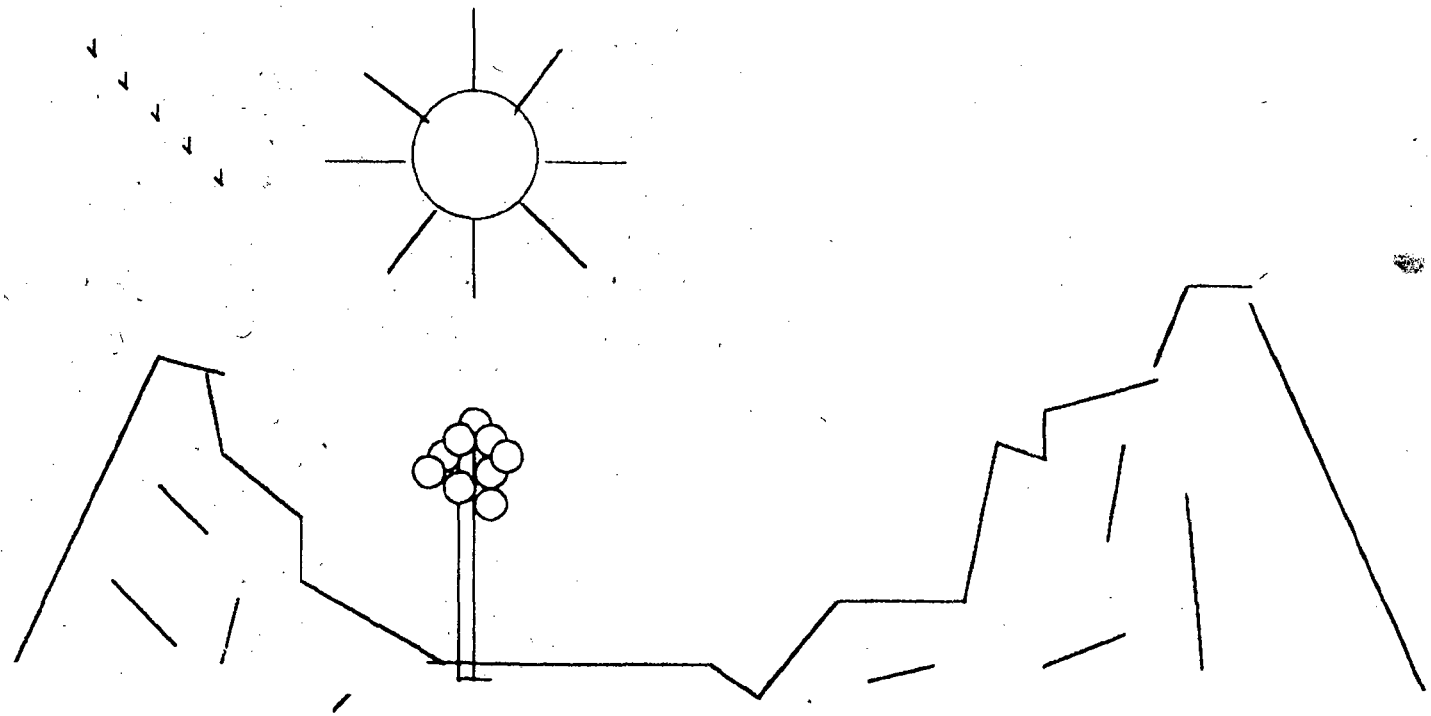


FIGURE 1

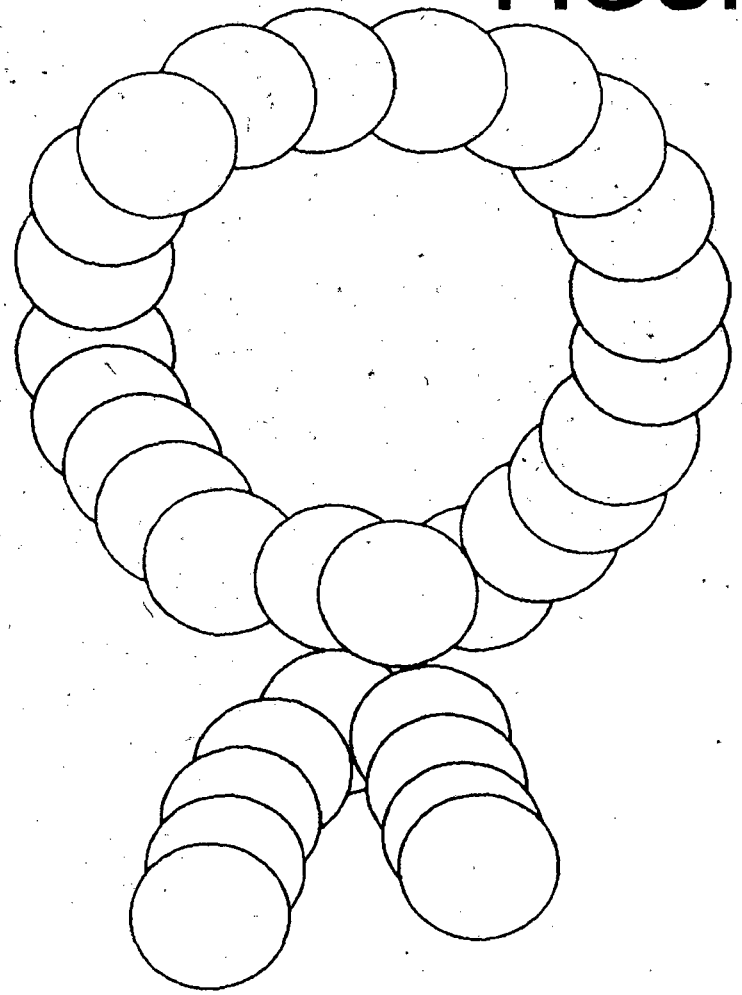


FIGURE 2

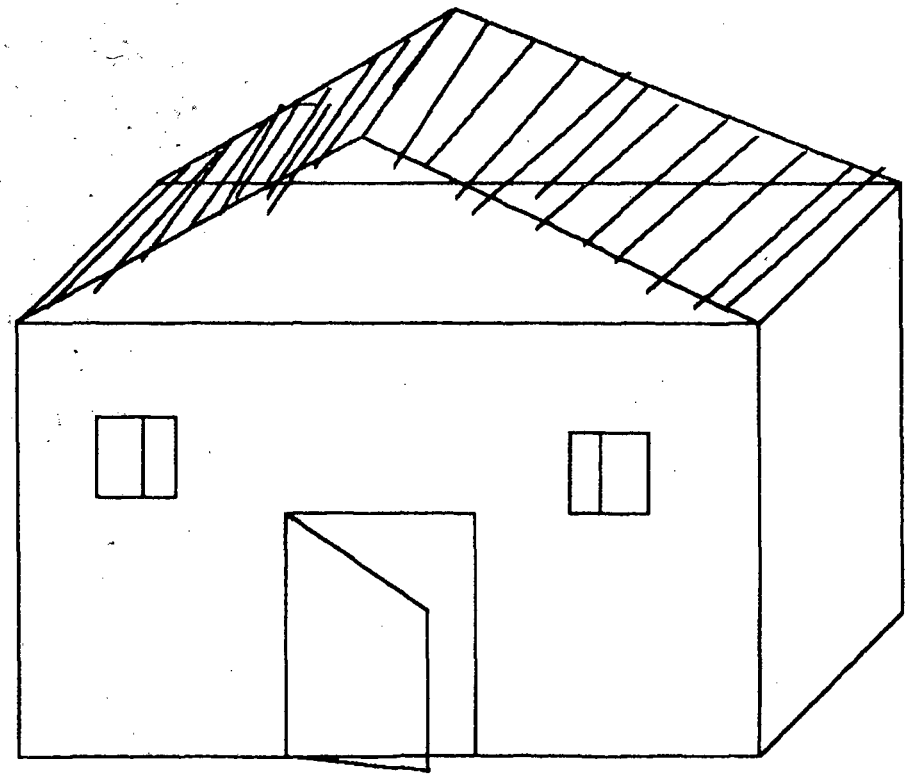


FIGURE 3

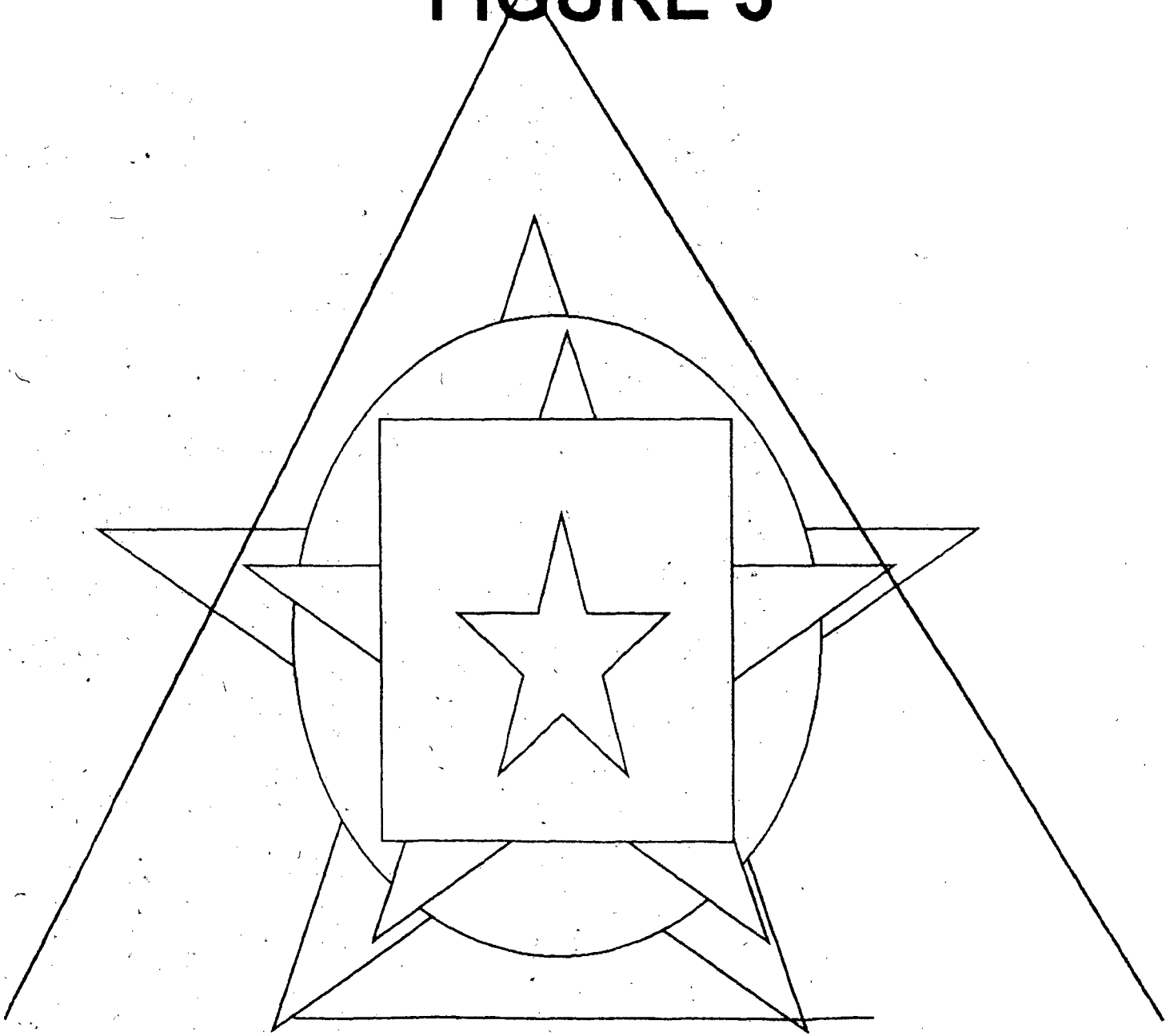
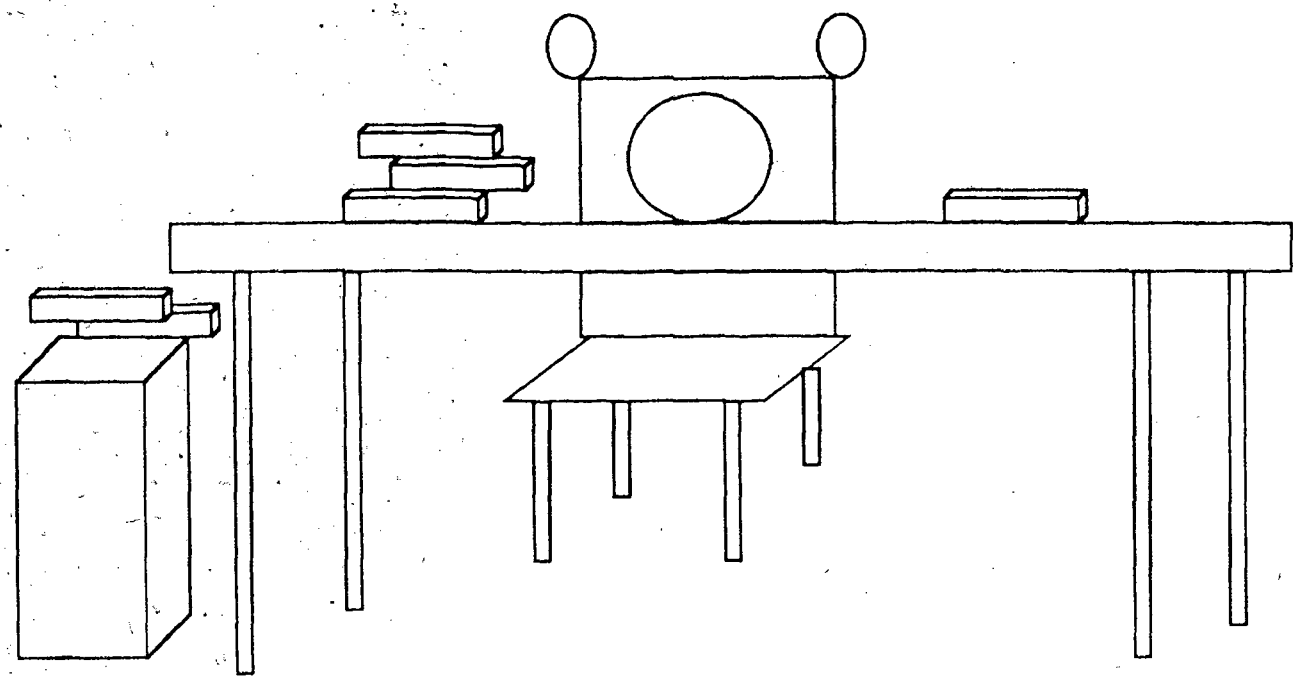


FIGURE 4



REFERENCES

REFERENCES

1. P. Wisskirchen, *Object-Oriented Graphics: From GKS and PHIGS to Object-Oriented Graphics*, Springer-Verlag, New York (1990).

2. J. Banerjee, H. Chou, J.F. Garza, W.Kim, D. Woelk, N. Ballou, and H.Kim, "Data Model Issues for Object-Oriented Applications," *ACM Trans. on Office Information Systems* 5(1), pp. 3-26 (Jan 1987).

3. O. M. Nierstrasz, "A survey of Object-Oriented Concepts," pp. 3-21 in *Object-Oriented Concepts, Database and Applications*, ed. W.Kim and F. Lockhovsky, ACM Press and Addison-Wesley, New York (1989).

4. A. Goldberg, D. Robson, *Samltalk-80: the Language and Its Implementation*, Addison-Wesley, Reading, MA (1983).

5. A. Boldberg, *Smalltalk-80: The Interactive Programming Environment*, Addison-Wesley, Reading, MA (1984).

6. I. E. Sutherland, "Sketchpad: A Man-Machine Graphical Communication System," *AFIPS Spring Joint Computer Conf.*, pp. 329-346 (1963).

7. A. Borning. "Thinglab: A constraint-Oriented Simulation Laboratory," Technical Report SSL-79-3, Xerox Palo Alto Research Centre, Palo Alto, Calif. (July 1979).

8. A Borning, *The Programming Language Aspects of ThingLab: A constraint-Oriented Simulation Laboratory*, ACM Trans. on Programming Languages and Systems 3(4), pp. 353-387 (Oct 1981).

9. A Borning and R. Duisberg, *Constraint-Based Tools for Building User Interface*, ACM Trans. on Graphics 5(4), pp. 345-374 (Oct 1986).

10. J.H. Maloney, A Borning, B.N. Freeman-Benson, *Constraint Technology for User-Interface Construction in ThingLab II*, Proc. ACM Conf. on Object-Oriented Programming Systems, Languages and Applications in SIGPLAN Notices, pp. 381-389 (Oct. 1989).

11. G. Nelson, *Juno, A Constraint-Based Graphics System*, Conf. Proc. ACM SIGGRAPH 19(3), PP. 235-243 (1985).

12. P. Barth, *An Object-Oriented Approach to Graphical Interfaces*, ACM Trans. on Graphics 5(2), pp. 142-172 (1986).

13. P.A. Szekely nd B.A. Myers, *A User Interface Toolkit Based on Graphical Objects and Constraints*, Proc. ACM Conf. Object- Oriented Programming, Systems Languages and Applications in SIGPLAN Notices 23(11), pp. 36-45 (Nov 1988).

14. A. Borning, R. Duisberg, B.N. Freeman-Benson, A. Kramer and M. Woolf, *Constraint Hierarchies*, pp. 48-60 in ACM Conf. on Object-Oriented Programming Systems, Languages and Applications, ACM SIGPLAN Notices (Oct. 1987).

15. B.V. Zanden, *Constraint Grammers: A New Model for Specifying Graphical Applications*, Proc. Conf. Human Factors in Computing Systems (SIGCHI 89), pp. 325-330 (April 1989).

16. W. Leler, *Constraint Programming Languages: Their Specification and Generation*, Addison-Wesley, Reading, MA (1988).

17. J.D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading MA 1984).

18. Neelam Bhalla and S. Balasundaram, *Object-Oriented Data Modeling for Graphics Databases: a Declarative Approach*, Computer Graphics Forum 10(1991), pp. 297-308.

19. Neelam Bhalla, **On Constraint-based Object-Oriented Graphics Database and Algebraic Query Model for Object-Oriented Databases**, Ph.D Thesis(1992),School of Computer and Systems Sciences,J.N.U., New Delhi.

APPENDIX

```
//MAIN FILE

#include <graphics.h>
#include <conio.h>
#include <fstream.h>

#include "PRIMITIVE.H"

void main()
{
char z;
int gdriver,gmode;                    // INITIALISING
gdriver = DETECT;                     // GRAPHICS
initgraph(&gdriver, &gmode,           // MODE
          "c:\\borlandc\\bgi");

circles c1 , ec1;                     //  These are
rect  r1 , er1;                       //  the different
tria  t1 , et1;                       //  classes and
texts t[1];                           //  their
polygon p1;                           //  respective
lines l1;                             //  objects.

mm.submenu();
mm.mainmenu();
mr.help_menu2();
mm.setscreen();

int shlr = 1, shud = 1;
  while ((z =  getche()) != 'k')      // Here the programme starts.
        {
           switch (z)
           {
//         case('R'):
           case  (77)  :   shud = 1;   //Right  Arrow  Button is
                                       //activated here
                  if (shlr == 1)

                  ++shlr;

              else if (shlr == 7)
                shlr = 1;    else
                  ++shlr;
           switch(shlr)
            {
            case (1) :mr.help_menu1();mm.help_menu1();
               break;
            case (2) :mr.help_menu1();mm.help_menu2();break;
            case (3) :mr.help_menu1();mm.help_menu2();break;
            case (4) :mr.help_menu1();mm.help_menu2();break;
            case (5) :mr.help_menu1();mm.help_menu4();break;
            case (6) :mr.help_menu1();mm.help_menu5();break;
            case (7) :mr.help_menu1();mm.help_menu5();break;
            }
                  break;
```

```
//          case('L'):
            case  (75)  :   shud  = 1;    //Left  Arrow  Button  is
                                          //activated here
                        if (shlr == 1)

                           shlr = 7;
                            else
                           if (shlr == 1)
                           shlr = 7; else
                           --shlr;

            switch(shlr)
            {
            case (1) :mr.help_menu1();mm.help_menu1();
               break;
            case (2) :mr.help_menu1();mm.help_menu2();break;
            case (3) :mr.help_menu1();mm.help_menu2();break;
            case (4) :mr.help_menu1();mm.help_menu2();break;
            case (5) :mr.help_menu1();mm.help_menu4();break;
            case (6) :mr.help_menu1();mm.help_menu5();break;
            case (7) :mr.help_menu1();mm.help_menu5();break;

            }
                      break;

//          case('U'):
            case  (72):  if  (shlr ==1)    //Up  Arrow  Button  is
                                          //activated here
              if (shud == 1) shud = 10 ;
                 else --shud;
                 else
               if (shlr == 2)
                 if (shud ==1) shud = 8;
                 else --shud;
                 else
               if (shlr == 3)
                 if (shud ==1) shud = 8;
                 else --shud;
                     else
                 if (shlr == 4)
                 if (shud ==1) shud = 8;
                 else --shud;
                 else
               if (shlr == 5)
                 if (shud ==1) shud = 10;
                 else --shud;
                 else
                if (shlr == 6)
                 if (shud == 1) shud = 7;
                 else --shud;
        break;
```

```
//          case ('D'):

        case  (80):  if (shlr ==1 )     //Down  Arrow  Button  is
                                        //activated here

          if (shud==10) shud = 1;       //This fixes the limit for
                else  ++shud;           //secondary menu of HELP

          else

          if (shlr == 2)                //This fixes the limit for
            if (shud ==8) shud =1;      //secondary menu of CIRKL
                else ++shud;

          else

          if (shlr == 3)
            if (shud ==8) shud =1;      //This fixes the limit for
                else ++shud;            //secondary menu of TRIA

          else

          if (shlr == 4)
            if (shud ==8) shud =1;      //This fixes the limit for
                else ++shud;            //secondary menu of RECT

            else

          if (shlr == 5)
            if (shud ==10) shud = 1;    //This fixes the limit for
            else ++shud;                //secondary menu of POLYGON

            else

          if (shlr == 6)
            if (shud == 7) shud = 1;    //This fixes the limit for
            else ++shud;                //secondary menu of LINE

            break;

case('C'):
case ('c'):mm.clear_menu();mm.submenu();
           mm.mainmenu();mr.help_menu2();break;
```

```
    case   13:                          //  ENTER   button   is
                                        //  activated here

switch(shlr)
{
 case(1):
  switch(shud)
    {
  case(1):mr.help_menu2();mm.help_menu1();
          break;
  case(2):mr.help_menu2();mm.help_menu1();
          break;
  case(3):mr.help_menu2();mm.help_menu1();
          break;
  case(4):mr.help_menu2();mm.help_menu1();
          break;
  case(5):mr.help_menu2();mm.help_menu1();
          break;
 case(11):mr.help_menu2();mm.help_menu1();mr.openfile();
          break;
 case(12):mr.help_menu2();mm.help_menu1();mr.closefile();
          break;
  default:mr.help_menu2();mm.help_menu1();
          break;
          shud = 1;

     } break;
```

```
  case(2):
    switch(shud)
    {
// This deals with CIRCLE and different transformations on it.
   case(1): mr.help_menu2();mr.help_menu1();c1.draw_menu();
            c1.draw();mr.cerasemenu();mr.help_menu2();break;
   case(2): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
            mr.hor_menu();c1.hrdraw();mr.gen_erase_menu();
            mr.help_menu2();break;
   case(3): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
            mr.eng_menu();c1.edraw();mr.gen_erase_menu();
            mr.help_menu2();break;
   case(4): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
            mr.rotat_menu();c1.rotdraw();mr.gen_erase_menu();
            mr.help_menu2();break;
   case(5): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
            mr.erase_menu();c1.erdraw();mr.gen_erase_menu();
            mr.help_menu2();break;
   case(6): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
            mr.paste_menu();c1.paste();mr.gen_erase_menu();
            mr.help_menu2();break;
   case(7): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
            mr.chng_lnclr();c1.chng_linecolor();mr.gen_erase_menu();
            mr.help_menu2();break;
   case(8): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
            mr.chng_flclr();c1.chng_fillcolor();mr.gen_erase_menu();
            mr.help_menu2();break;
/*     case(9): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
         c1.addfile();mr.gen_erase_menu();
         mr.help_menu2();break;*/
      }break;
```

```
    case(3):
      switch(shud)
      {
// This deals with RECTANGLE and different transformations on it.

    case(1): mr.help_menu2();mr.help_menu1();r1.drawmenu();
             r1.draw();mr.rerasemenu();mr.help_menu2();break;
    case(2): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
             mr.hor_menu();r1.hrdraw();mr.gen_erase_menu();
             mr.help_menu2();break;
    case(3): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
             mr.eng_menu();r1.edraw();mr.gen_erase_menu();
             mr.help_menu2();break;
    case(4): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
             mr.rotat_menu();r1.rotdraw();mr.gen_erase_menu();
             mr.help_menu2();break;
    case(5): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
             mr.erase_menu();r1.erdraw();mr.gen_erase_menu();
             mr.help_menu2();break;
    case(6): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
             mr.paste_menu();r1.paste();mr.gen_erase_menu();
             mr.help_menu2();break;
    case(7): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
             mr.chng_lnclr();r1.chng_linecolor();mr.gen_erase_menu();
             mr.help_menu2();break;
    case(8): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
             mr.chng_flclr();r1.chng_fillcolor();mr.gen_erase_menu();
             mr.help_menu2();break;

        }break;
```

```
  case(4):
     switch(shud)
      {
// This deals with TRIANGLE and different transformations on it.
      case(1): mr.help_menu2();mr.help_menu1();t1.drawmenu();
               t1.draw();mr.terasemenu();mr.help_menu2();break;
      case(2): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
               mr.hor_menu();t1.hrdraw();mr.gen_erase_menu();
               mr.help_menu2();break;
      case(3): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
               mr.eng_menu();t1.edraw();mr.gen_erase_menu();
               mr.help_menu2();break;
      case(4): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
               mr.rotat_menu();t1.rotdraw();mr.gen_erase_menu();
               mr.help_menu2();break;
      case(5): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
               mr.erase_menu();t1.erdraw();mr.gen_erase_menu();
               mr.help_menu2();break;
      case(6): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
               mr.paste_menu();t1.paste();mr.gen_erase_menu();
               mr.help_menu2();break;
      case(7): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
               mr.chng_lnclr();t1.chng_linecolor();mr.gen_erase_menu(
               mr.help_menu2();break;
      case(8): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
               mr.chng_flclr();t1.chng_fillcolor();mr.gen_erase_menu(
               mr.help_menu2();break;
      }break;
```

```
case(5):
  switch(shud)
  {
// This deals with POLYGON and different transformations on it.
  case(1): mr.help_menu2();mr.help_menu1();p1.drawmenu();
           p1.draw();mr.perasemenu();mr.help_menu2();break;
  case(2): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
           mr.hor_menu();p1.hdraw();mr.gen_erase_menu();
           mr.help_menu2();break;
  case(4): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
           mr.rotat_menu();p1.rotdraw();mr.gen_erase_menu();
           mr.help_menu2();break;
  case(5): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
           mr.erase_menu();p1.erdraw();mr.gen_erase_menu();
           mr.help_menu2();break;
  case(6): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
           mr.paste_menu();p1.paste();mr.gen_erase_menu();
           mr.help_menu2();break;
  case(7): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
           mr.chng_lnclr();p1.chng_linecolor();mr.gen_erase_menu();
           mr.help_menu2();break;
  case(8): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
           mr.chng_flclr();p1.chng_fillcolor();mr.gen_erase_menu();
           mr.help_menu2();break;
  case(9): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
           mr.chng_flptrn();p1.chng_fillpattern();mr.gen_erase_menu(
           mr.help_menu2();break;
 case(10): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
           mr.chng_flstyl();p1.chng_linestyle();mr.gen_erase_menu();
           mr.help_menu2();break;
    }break;
```

```
   case(6):
     switch(shud)
     {
// This deals with LINE and different transformations on it.

   case(1): mr.help_menu2();mr.help_menu1();l1.drawmenu();
            l1.draw();mr.lerasemenu();mr.help_menu2();break;

   case(2): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
            mr.hor_menu();l1.hrdraw();mr.gen_erase_menu();
            mr.help_menu2();break;

// case(3): mr.help_menu2();l1.edraw();mr.gen_erase_menu();
//          mr.help_menu2();break;

   case(4): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
            mr.rotat_menu();l1.rotdraw();mr.gen_erase_menu();
            mr.help_menu2();break;

   case(5): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
            mr.erase_menu();l1.erdraw();mr.gen_erase_menu();
            mr.help_menu2();break;

   case(6): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
            mr.paste_menu();l1.paste();mr.gen_erase_menu();
            mr.help_menu2();break;

   case(7): mr.help_menu2();mr.help_menu1();mr.gen_erase_menu();
            mr.chng_lnclr();l1.chng_linecolor();mr.gen_erase_menu();
            mr.help_menu2();break;
            }break;
```

```
   case(7):

        switch(shud)
        {
// This deals with TEXT and different transformations on it.

        case(1): mr.help_menu2();mr.help_menu1();t[1].drawmenu();
         t[1].drawtext();mr.xerasemenu();mr.help_menu2();break;

//      case(2): mr.help_menu2();t[1].hrdraw();mr.help_menu2();break;
//      case(3): mr.help_menu2();t[1].edraw();mr.help_menu2();break;
//      case(4): mr.help_menu2();t[1].rotdraw();mr.help_menu2();break;
//      case(5): mr.help_menu2();t[1].erdraw();mr.help_menu2();break;
//      case(6): mr.help_menu2();t[1].paste();mr.help_menu2();break;

         }break;
          }
            }

  switch (shlr)
    {

   case(1): mr.updn_menu5();moveto(10, 15);
            setcolor(GREEN);outtext("HELP");
            break;

   case(2): mr.updn_menu5();moveto(105, 15);
            setcolor(GREEN);outtext("CIRKL");
            break;

   case(3): mr.updn_menu5();moveto(200, 15);
            setcolor(GREEN);outtext("RECT");
            break;

   case(4): mr.updn_menu5();moveto(295, 15);
            setcolor(GREEN);outtext("TRIA");
            break;

   case(5): mr.updn_menu5();moveto(390, 15);
            setcolor(GREEN);outtext("POLY");
            break;

   case(6): mr.updn_menu5();moveto(485, 15);
            setcolor(GREEN);outtext("LINE");
            break;

   case(7): mr.updn_menu5();moveto(580, 15);
            setcolor(GREEN);outtext("TEXT");
            break;
          }
```

```
 if (shlr == 1)

 switch(shud)
 {

case(1): mr.updn_menu4();moveto(5, 43);
         setcolor(GREEN);outtext("Left");
         break;


case(2): mr.updn_menu4();moveto(5, 64);
         setcolor(GREEN);outtext("Right");
         break;


case(3): mr.updn_menu4();moveto(5, 85);
         setcolor(GREEN);outtext("Up");
         break;


case(4): mr.updn_menu4();moveto(5, 106);
         setcolor(GREEN);outtext("Down");
         break;


case(5): mr.updn_menu4();moveto(5, 127);
         setcolor(GREEN);outtext("Erase");
         break;


case(6): mr.updn_menu4();moveto(5, 148);
         setcolor(GREEN);outtext("Kill");
         break;


case(7): mr.updn_menu4();moveto(5, 169);
         setcolor(GREEN);outtext("Nlarge");
         break;


case(8): mr.updn_menu4();moveto(5, 190);
         setcolor(GREEN);outtext("roTate");
         break;


case(9): mr.updn_menu4();moveto(5, 211);
         setcolor(GREEN);outtext("VertMov");
         break;


case(10):mr.updn_menu4();moveto(5, 232);
         setcolor(GREEN);outtext("HortMov");
         break;
     }
```

```
   else
  if (shlr == 5)

   switch(shud)
     {
case(1):  mr.updn_menu1();moveto(5, 43);
          setcolor(GREEN);outtext("DRAW");
          break;


case(2): mr.updn_menu1();moveto(5, 64);
         setcolor(GREEN);outtext("MOVE");
         break;


case(3): mr.updn_menu1();moveto(5, 85);
         setcolor(GREEN);outtext("NLARGE");
         break;


case(4): mr.updn_menu1();moveto(5, 106);
         setcolor(GREEN);outtext("ROTAT");
         break;


case(5): mr.updn_menu1();moveto(5, 127);
         setcolor(GREEN);outtext("ERASE");
         break;


case(6): mr.updn_menu1();moveto(5, 148);
         setcolor(GREEN);outtext("PASTE");
         break;


case(7): mr.updn_menu1();moveto(5, 169);
         setcolor(GREEN);outtext("CHN_LNCLR");
         break;


case(8): mr.updn_menu1();moveto(5, 190);
         setcolor(GREEN);outtext("CHN_FLCLR");
         break;


case(9): mr.updn_menu1();moveto(5, 211);
         setcolor(GREEN);outtext("FLPTRN");
         break;


case(10):mr.updn_menu1();moveto(5, 232);
         setcolor(GREEN);outtext("LNSTYL");
         break;
           }
```

```
        else
        if (shlr ==6)
          switch(shud)
           {
case(1): mr.updn_menu2();moveto(5, 43);
         setcolor(GREEN);outtext("DRAW");
         break;
case(2): mr.updn_menu2();moveto(5, 64);
         setcolor(GREEN);outtext("MOVE");
         break;
case(3): mr.updn_menu2();moveto(5, 85);
         setcolor(GREEN);outtext("NLARGE");
         break;
case(4): mr.updn_menu2();moveto(5, 106);
         setcolor(GREEN);outtext("ROTAT");
         break;
case(5): mr.updn_menu2();moveto(5, 127);
         setcolor(GREEN);outtext("ERASE");
         break;
case(6): mr.updn_menu2();moveto(5, 148);
         setcolor(GREEN);outtext("PASTE");
         break;
case(7): mr.updn_menu2();moveto(5, 169);
         setcolor(GREEN);outtext("CHN_LNCLR");
         break;
           }
```

```
        else
           switch(shud)
             {
    case(1): mr.updn_menu3();moveto(5, 43);
             setcolor(GREEN);outtext("DRAW");
             break;
    case(2): mr.updn_menu3();moveto(5, 64);
             setcolor(GREEN);outtext("MOVE");
             break;
    case(3): mr.updn_menu3();moveto(5, 85);
             setcolor(GREEN);outtext("NLARGE");
             break;
    case(4): mr.updn_menu3();moveto(5, 106);
             setcolor(GREEN);outtext("ROTAT");
             break;
    case(5): mr.updn_menu3();moveto(5, 127);
             setcolor(GREEN);outtext("ERASE");
             break;
    case(6): mr.updn_menu3();moveto(5, 148);
             setcolor(GREEN);outtext("PASTE");
             break;
    case(7): mr.updn_menu3();moveto(5, 169);
             setcolor(GREEN);outtext("CHN_LNCLR");
             break;
    case(8): mr.updn_menu3();moveto(5, 190);
             setcolor(GREEN);outtext("CHN_FLCLR");
             break;
             }
            }
  closegraph() ;
}
```

```
//PRIMITIVE FILE

#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
#include <string.h>
#include <dos.h>
#include "SHAPE FILE"
#include "MENU CREATION FILE"
#include "MENU ERASURE FILE"


mini_menu mm;                   //Objects of
mini_menuer mr;                 //Menu Files


          //CLASS TEXTS


class texts : public shape
  {
    private:int x,y,X,Y,z,mx,my,dx,dy,f,d,c,bc,bc1,hz,vt;
            char str[50],s[50];

       int  font     ;
       int  direction;
       int  size;
       int  color,bcolor;
       int  horzjustify;
       int  vertjustify;
       int  multx, divx;
       int  multy, divy;

    public:
      texts()
        {
          str[0] = '\0';
          X = 20; Y =450;
          font = DEFAULT_FONT;
          direction = HORIZ_DIR;
          size = 0;
          color = RED;bcolor = BLACK;
          horzjustify = LEFT_TEXT;
          vertjustify = TOP_TEXT;
          multx = 8; divx = 7;
          multy = 1;divy = 1;
        }

   void draw();          //This function writes the text.
   void drawmenu();      //This displays the secondary menu
   void drawmenu1();     //This is called by the above method.
```

```
void set(int x,int y,int f,int d,int z,int c,
         int hz,int vt,int mx,int my,int dx,int dy,char s[]);

void set1(int x,int y,int f,int d,int z,int c1,
     int hz,int vt,int mx,int my,int dx,int dy,char s[]);
 void drawtext1();                    // These  methods
 void drawtext();                     // perform
 void settext(char str1[]);           // the
 void setposition(int x, int y);      // obvious
 void setfont(int f);                 // functions
 void setdirection(int d);            //
 void setsize(int s);                 //
 void setcolor(int c);                //
 void setHJust(int hj);               //
 void setVJust(int vj);               //
 void setHorzSize(int mx, int dx);    //
 void setVertSize(int my, int dy);    //
 void setbk_grnd_color(int bc);       //
 void drawtext2();
 };

       void texts::set(int x,int y,int f,int d,int z,int c,
                 int hz,int vt,int mx,int my,int dx,int dy,char
       { shape::set6(x,y,f,d,z,c,hz,vt,mx,my,dx,dy,s); }

       void texts::set1(int x,int y,int f,int d,int z,int c,
                 int hz,int vt,int mx,int my,int dx,int dy,char
       { shape::set61(x,y,f,d,z,c,hz,vt,mx,my,dx,dy,s); }

     void texts::drawtext1()
     {
      shape::drawtext3();
      }

     void texts::drawtext()
     {
     texts::draw();
     texts::set(x,y,f,d,z,c,hz,vt,mx,my,dx,dy,s);
     texts::drawtext1();
      }

    void texts::settext(char str1[])
        { strcpy(str, str1); }

   void texts::setposition(int x, int y)
       { X = x; Y = y; }

   void texts::setfont(int f)
       { font = f; }

   void texts::setdirection(int d)
       { direction = d; }

   void texts::setsize(int s)
       { size = s; }
```

```
void texts::setcolor(int c)
    { color = c; }
void texts::setHJust(int hj)
    { horzjustify = hj; }

void texts::setVJust(int vj)
    { vertjustify = vj; }

void texts::setHorzSize(int mx, int dx)
    { multx = mx; divx = dx; }

void texts::setVertSize(int my, int dy)
    { multy = my; divy = dy; }

void texts::setbk_grnd_color(int bc)
{  bcolor = bc; }

void texts::drawtext2()
{
settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
setcolor(BLACK);
setusercharsize(8,7,1,1);
outtext(str);
}

void texts::drawmenu()
{
setcolor(WHITE);
setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
rectangle(0, 35, 78, 450);
setfillstyle(SOLID_FILL, LIGHTGRAY);
floodfill(10, 45, WHITE);
setcolor(WHITE);
for(long int li=35;li<=450;li+=21)
{
moveto(0, li);
lineto(78, li);
moveto(5, li+8);
 }
texts::drawmenu1();
}

void texts::drawmenu1()
{
 for(long int li=35;li<=450;li+=21)
  {
  moveto(5,li+8);
switch(li)
   {
      case(35):    setcolor(RED);outtext("LOCATION"); break;
      case(56):    setcolor(RED); outtext("STRING"); break;
      default  : break;
       }
  }
}
```

```
void texts::draw()
{
 outtextxy(20,450,"LOCATION");
 x = shape::getval();
 moveto(20,450);
 y = shape::getval();
 outtextxy(20,400,"STRING");
 gets(s);
}
                                    // CLASS LINES
class lines : public texts
  {
  private , : int X1,Y1,X2,Y2;
              int x1,y1,x2,y2,llc,llc1, a;
  public:

    void set(int x1,int y1,int x2,int y2,int llc)
      { shape::set5(x1,y1,x2,y2,llc); }
    void set1(int x1,int y1,int x2,int y2,int llc1)
      { shape::set51(x1,y1,x2,y2,llc1); }

   void odraw()                     // This is the basic function
   {                                // for drawing
    lines::set(x1,y1,x2,y2,llc);
    shape::draw1();
    line(X1,Y1,X2,Y2);
   }

   void eradraw()                   //This is the basic function
   {                                // for erasing
    llc1 = 0;
    lines::set1(x1,y1,x2,y2,llc1);
    shape::draw1();
    line(X1,Y1,X2,Y2);
   }

   void drawmenu();
   void drawmenu1();

   void draw()
   {
    moveto(20,450);
    texts::drawtext1();
    X1 = shape::getval();
    Y1 = shape::getval();
    moveto(20,450);
    texts::drawtext1();
    X2 = shape::getval();
    Y2 = shape::getval();
    moveto(20,450);
    texts::drawtext1();
    llc = shape::getcolor();
    lines::odraw();
   }
};
```

```
void lines::drawmenu()
{
 setcolor(WHITE);
 setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
 rectangle(0, 35, 78, 450);
 setfillstyle(SOLID_FILL, LIGHTGRAY);
 floodfill(10, 45, WHITE);
 setcolor(WHITE);
 for(long int li=35;li<=450;li+=21)
  {
   moveto(0, li);
   lineto(78, li);
   moveto(5, li+8);
  }
 lines::drawmenu1();
}


void lines::drawmenu1()
{
 for(long int li=35;li<=450;li+=21)
 {
  moveto(5,li+8);
  setcolor(RED);
  switch(li)
    {
 case(35):outtext("1ST PT"); break;
     case(56): outtext("2ND PT"); break;
     case(77): outtext("LNCLR"); break;
     default  : break;
     }
   }
  }


class circles : public texts , mini_menuer    //CLASS CIRCLES
  {
  private :int Xco,Yco,W,x1,y1,x2,y2,a,x,y,n,vu,hl,vd,p1 ;
           int cLc,cFc,cLc1,cFc1;
           double a2;
  public:

        void set(int x, int y, int w, int cLc, int cFc)
          { shape::set1( x, y, w, cLc, cFc); }
        void set1(int x,int y,int w,int cLc1,int cFc)
          { shape::set11(x,y,w,cLc1,cFc); }

        void odraw()                          //The basic draw method
        {
          circles::set(Xco, Yco, W, cLc , cFc);
          shape::draw();
          circle(Xco, Yco, W);
          floodfill(Xco, Yco, linecolor);
           }
```

```
    void draw()                          //Draws the circle
      {
      moveto (20, 450);
  texts::drawtext1();
      mini_menuer::help_menu2();
      Xco = shape::getval();
      mini_menuer::help_menu2();
      Yco = shape::getval();
      mini_menuer::help_menu2();
      texts::drawtext2();
      mr.help_menu2();
      moveto(20,450);
  texts::drawtext1();
      W = shape::getval();
      mini_menuer::help_menu2();
      moveto(20,450);
  texts::drawtext1();
      cLc = shape::getcolor();
      moveto(20,450);
  texts::drawtext1();
      cFc = shape::getcolor();
      circles::odraw();
       }

void  draw_menu();          //Displays  the  secondary menu
void drawmenu1();           //This is called by the above


void chng_linecolor()       //Changes the outline color
{
cLc = shape::getcolor();
circles::odraw();
}

void chng_fillcolor()       //Changes the fill color
{
cFc = shape::getcolor();
circles::odraw();
}

void eradraw()              //Basic erasing function
{
cLc1 = 0;cFc1 = 0;
circles::set1(Xco, Yco, W, cLc1, cFc1);
shape::draw();
circle(Xco, Yco, W);
floodfill(Xco,Yco,linecolor);
 }
```

```
void erdraw()                    //Erases the circle
 {
  moveto(20,450);
  p1 = getch();
   if ( p1 == 'y')
        {
        circles::eradraw();
        }
        else
        {
      shape::draw();
      moveto (20, 450);
 texts::drawtext1();
      Xco = shape::getval();
      Yco = shape::getval();
      texts::drawtext2();
      mr.help_menu2();
      moveto(20,450);
 texts::drawtext1();
      W = shape::getval();
      circles::eradraw();
       }
       }


    void rotdraw()               //Rotates the circle
       {
      moveto(20,450);
  texts::drawtext1();
      x2 = shape::getval();
      y2 = shape::getval();
      moveto(20, 450);
  texts::drawtext1();
      a  = shape::getval();
      circles::eradraw();
      a2 = double(a) / double(57);
      x1 = Xco; y1 = Yco;
      Xco = x1 * cos(a2) - y1 * sin(a2) -
               x2 * cos(a2) + y2 * sin(a2) + x2;
      Yco = x1 * sin(a2) + y1 * cos(a2) -
               x2 * sin(a2) - y2 * cos(a2) + y2;
      circles::odraw();
      }


    void edraw()                 //Enlarges the circle
       {
      moveto(20,450);
 texts::drawtext1();
      n = shape::getval();
      circles::eradraw();
      W+=n;
      circles::odraw();
      }
```

```
        void hrdraw()              //Moves the circle
         {
         moveto(20,450);
         texts::drawtext1() ;
         x1 = shape::getval();
         y1 = shape::getval();
         circles::eradraw();
         Xco=x1;
         Yco=y1;
         circles::odraw();

         }


        void paste()               //Pastes the circle
           {

         moveto(20,450);
      texts::drawtext1() ;
         x1 = shape::getval();
         y1 = shape::getval();
         Xco=x1;
         Yco=y1;
         circles::odraw();

         }

};


void circles::draw_menu()
 {
 setcolor(WHITE);
 setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
 rectangle(0, 35, 78, 450);
 setfillstyle(SOLID_FILL, LIGHTGRAY);
 floodfill(10, 45, WHITE);
 setcolor(WHITE);
 for(long int li=35;li<=450;li+=21)
 {
 moveto(0, li);
 lineto(78, li);
 moveto(5, li+8);
  }
 circles::drawmenu1();
 }
```

```
void circles::drawmenu1()
  {
  for(long int li=35;li<=450;li+=21)
   {
   moveto(5,li+8);
   setcolor(RED);
   switch(li)
        {
       case(35):outtext("LOCATION"); break;
       case(56): outtext("RADIUS"); break;
       case(77): outtext("LINECLR"); break;
       case(98): outtext("FILLCLR"); break;
       default  : break;
         }
    }
  }
```

NOTE:

The definitions and methods for the primitives RECT and TRIA are the same.Hence they have not been shown here to conserve space.Similarly the methods for the primitive LINES are again same as CIRCLES and hence their representation has been avoided here.

//CLASS POLYGON

```
class polygon :public texts

  {

  private : int poly[20],j,lc,fc,lc1,fc1,ls;
            double xmax; div_t x;

  public :

      void set( int ls,int w,int j,int lc,int fc)
        {shape::set4(ls,w,j,lc,fc); }

      void set1( int ls,int w,int j,int lc1,int fc1)
        {shape::set41(ls,w,j,lc1,fc1); }

    void chng_linecolor()
    { lc = shape::getcolor(); polygon::odraw();}

    void chng_fillcolor()
    {
    fc = shape::getcolor();
    polygon::odraw();
    }
```

```
    void chng_fillpattern()
    {
    j = shape::getfillpatrn();
    polygon::odraw();
    }

    void chng_linestyle()
    {
    ls = shape::getlinestyle();
    polygon::odraw();
    }

    void odraw()
    {
     polygon::set(ls,w,j,lc,fc);
     shape::draw2();
     fillpoly(xmax+1,poly);
    }

     void eradraw()
     {
     int fc1 = 0,lc1 = 0;
     polygon::set1(ls,w,j,lc1,fc1);
     shape::draw2();
     fillpoly(xmax+1,poly);
     }

  void drawmenu();          //Draws the secondary menu
  void drawmenu1();         //This is called by the above
  void draw();              //draws the polygon
  void hdraw();             //Moves the polygon
  void paste();             //Pastes the polygon
  void rotdraw();           //Rotates the polygon
  void erdraw();            //Erases the polygon
   };

void polygon::draw()        //Draws the polygon
 {
  texts::drawtext1();
  xmax = shape::getval();
  for ( int i = 0;i <= 2 * xmax - 1;i++)
  poly[i] = shape::getval();
  poly[2*xmax] = poly[0];
  poly[2*xmax+1] = poly[1];
  texts::drawtext1();
  lc = shape::getcolor();
  texts::drawtext1();
  ls  = shape::getlinestyle();   // 0 - 4
  texts::drawtext1();
  fc = shape::getcolor();
  texts::drawtext1();
  j = shape::getfillpatrn();
  polygon::odraw();
  }
```

```
void polygon::drawmenu()      //Draws the secondary menu
 {
 setcolor(WHITE);
 setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
 rectangle(0, 35, 78, 450);
 setfillstyle(SOLID_FILL, LIGHTGRAY);
 floodfill(10, 45, WHITE);
 setcolor(WHITE);
 for(long int li=35;li<=450;li+=21)
 {
 moveto(0, li);
 lineto(78, li);
 moveto(5, li+8);
  }
 polygon::drawmenu1();
 }

 void polygon::drawmenu1()
  {
  for(long int li=35;li<=450;li+=21)
   {
   moveto(5,li+8);
   setcolor(RED);
   switch(li)
       {
       case(35):outtext("VERTICES"); break;
       case(56): outtext("PTS"); break;
       case(77): outtext("LINECLR"); break;
       case(98): outtext("LINESTYLE"); break;
       case(119): outtext("FILLCLR"); break;
       case(140): outtext("FILLPTRN"); break;
/*  case(161): outtext(""); break;
       case(182): outtext("roTat"); break;
       case(203): outtext("VertMov"); break;
       case(224): outtext("HortMov"); break;
       case(245): break;    */
       default  : break;
       }
     }
  }

 void polygon::hdraw()          //Moves the polygon
  {
    texts::drawtext1();
    int x = shape::getval();
    int y = shape::getval();
    polygon::eradraw();
    int x1 = x - poly[0];
    int y1 = y - poly[1];
    for(int i = 0;i <= 2*xmax+1;i+=2)
    poly[i] += x1;
    for( i = 1;i <= 2*xmax+1;i+=2)
    poly[i] += y1;
    polygon::odraw();
      }
```

```
void polygon::paste()          //Pastes the polygon
  {
 texts::drawtext1();
     int x = shape::getval();
     int y = shape::getval();
     int x1 = x - poly[0];
     int y1 = y - poly[1];
     for(int i = 0;i <= 2*xmax+1;i+=2)
     poly[i] += x1;
     for( i = 1;i <= 2*xmax+1;i+=2)
     poly[i] += y1;
     polygon::odraw();
  }

void polygon::rotdraw()          //Rotates the polygon
 {
  moveto(20,450);
  texts::drawtext1();
  int x = shape::getval();
  int y = shape::getval();
  moveto(20,450);
  int a = shape::getval();
  polygon::eradraw();
  for(int i = 0;i <= 2*xmax+1;i+=2)
   {
    int x1 = poly[i];int y1 = poly[i+1];
    poly[i]   = shape::calculate1(x1,y1,x,y,a);
    poly[i+1] = shape::calculate2(x1,y1,x,y,a);
   }
   polygon::odraw();
  }

void polygon::erdraw()          //Erases the polygon
 {
   moveto(20,450);
   texts::drawtext1();
   int p3 = getch();
   if (p3 == 'y')
    {
     polygon::eradraw();
    }
   else
    {
     moveto(20, 450);
     texts::drawtext1();
     xmax = shape::getval();
     texts::drawtext1();
     for ( int i = 0;i <= 2 * xmax - 1;i++)
     poly[i] = shape::getval();
     poly[2*xmax] = poly[0];
     poly[2*xmax+1] = poly[1];
     polygon::eradraw();
        }
          }
```

```
//#include "TEXT FILE"

//#include <string.h>

class mini_menu
  {

 public:
  void help_menu1();          //Displays the help menu
  void wrttext1();            //This is called by the above
  void help_menu2();          //same as above
  void wrttext2();            //same as above
  void help_menu4();          //same as above
  void wrttext4();            //same as above
  void help_menu5();          //same as above
  void wrttext5();            //same as above
  void submenu();             //Displays the sub menu


  void mainmenu()            //Displays the main menu
        {
        setcolor(GREEN);
        setlinestyle(SOLID_LINE,0,NORM_WIDTH);
        rectangle(0,35,630,450);
        moveto(82,35);
        lineto(82,450);
        setfillstyle(SOLID_FILL,GREEN);
        floodfill(150,150,BLACK);
        setcolor(RED);
        setusercharsize(1,1,1,1);
        }

   void setscreen()                    //Sets the screen
   {
         moveto(580, 15);setcolor(RED);outtext("TEXT");
         moveto(105, 15);outtext("CIRKL");
         moveto(200, 15);outtext("RECT");
         moveto(295, 15);outtext("TRIA");
         moveto(390, 15);outtext("POLY");
         moveto(485, 15);outtext("LINE");
         moveto(10, 15);setcolor(GREEN);outtext("HELP");
         moveto(5, 43);setcolor(GREEN);outtext("Left");
         moveto(5, 64);setcolor(RED);outtext("Right");
         moveto(5, 85);setcolor(RED);outtext("Up");
         moveto(5, 106);setcolor(RED);outtext("Down");
         moveto(5, 127);setcolor(RED);outtext("Erase");
         moveto(5, 148);setcolor(RED);outtext("Kill");
         moveto(5, 169);setcolor(RED);outtext("Nlarge");
         moveto(5, 190);setcolor(RED);outtext("roTate");
         moveto(5, 211);setcolor(RED);outtext("VertMov");
         moveto(5, 232);setcolor(RED);outtext("HortMov");
           }
};
```

```
void mini_menu::help_menu1()

{

setcolor(WHITE);
setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
rectangle(0, 35, 78, 450);
setfillstyle(SOLID_FILL, LIGHTGRAY);
floodfill(10, 45, WHITE);
setcolor(WHITE);

for(long int li=35;li<=450;li+=21)
{
moveto(0, li);
lineto(78, li);
moveto(5, li+8);
 }
mini_menu::wrttext1();
}



void mini_menu::wrttext1()

{

for(long int li=35;li<=450;li+=21)
 {
 moveto(5,li+8);
 setcolor(RED);

switch(li)
      {
      case(35):outtext("Left"); break;
      case(56): outtext("Right"); break;
      case(77): outtext("Up"); break;
      case(98): outtext("Down"); break;
      case(119): outtext("Erase"); break;
      case(140): outtext("Kill"); break;
      case(161): outtext("Nlarge"); break;
      case(182): outtext("roTat"); break;
      case(203): outtext("VertMov"); break;
      case(224): outtext("HortMov"); break;
      case(245): break;
      default  : break;
      }

   }
   }
```

```
void mini_menu::help_menu2()

{

setcolor(WHITE);
setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
rectangle(0, 35, 78, 450);
setfillstyle(SOLID_FILL, LIGHTGRAY);
floodfill(10, 45, WHITE);
setcolor(WHITE);

for(long int li=35;li<=450;li+=21)
{
moveto(0, li);
lineto(78, li);
moveto(5, li+8);
 }
mini_menu::wrttext2();
}



void mini_menu::wrttext2()

{

for(long int li=35;li<=450;li+=21)
 {
 moveto(5,li+8);
 setcolor(RED);

switch(li)
      {
      case(35):outtext("DRAW"); break;
      case(56): outtext("MOVE"); break;
      case(77): outtext("NLARGE"); break;
      case(98): outtext("ROTAT"); break;
      case(119): outtext("ERASE"); break;
      case(140): outtext("PASTE"); break;
      case(161): outtext("CHN_LNCLR"); break;
      case(182): outtext("CHN_FLCLR"); break;
/*  case(203): outtext("VertMov"); break;
      case(224): outtext("HortMov"); break;*/
      default  : break;
      }

   }
   }
```

```
void mini_menu::help_menu4()

{
setcolor(WHITE);
setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
rectangle(0, 35, 78, 450);
setfillstyle(SOLID_FILL, LIGHTGRAY);
floodfill(10, 45, WHITE);
setcolor(WHITE);

for(long int li=35;li<=450;li+=21)
{
moveto(0, li);
lineto(78, li);
moveto(5, li+8);
 }

mini_menu::wrttext4();
}



void mini_menu::wrttext4()
{

for(long int li=35;li<=450;li+=21)
 {
 moveto(5,li+8);
 setcolor(RED);

switch(li)
      {
      case(35):outtext("DRAW"); break;
      case(56): outtext("MOVE"); break;
      case(77): outtext("NLARGE"); break;
      case(98): outtext("ROTAT"); break;
      case(119): outtext("ERASE"); break;
      case(140): outtext("PASTE"); break;
      case(161): outtext("CHN_LNCLR"); break;
      case(182): outtext("CHN_FLCLR"); break;
      case(203): outtext("FLPTRN"); break;
      case(224): outtext("LNSTYL"); break;
      default  : break;
      }

   }
   }
```

```
void mini_menu::help_menu5()
{
setcolor(WHITE);
setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
rectangle(0, 35, 78, 450);
setfillstyle(SOLID_FILL, LIGHTGRAY);
floodfill(10, 45, WHITE);
setcolor(WHITE);

for(long int li=35;li<=450;li+=21)
{
moveto(0, li);
lineto(78, li);
moveto(5, li+8);
 }

mini_menu::wrttext5();
}




void mini_menu::wrttext5()
{

for(long int li=35;li<=450;li+=21)
 {
 moveto(5,li+8);
 setcolor(RED);

switch(li)
      {
      case(35):outtext("DRAW"); break;
      case(56): outtext("MOVE"); break;
      case(77): outtext("NLARGE"); break;
      case(98): outtext("ROTAT"); break;
      case(119): outtext("ERASE"); break;
      case(140): outtext("PASTE"); break;
      case(161): outtext("CHN_LNCLR"); break;
/*  case(182): outtext("CHNG_FLCLR"); break;
      case(203): outtext("CHNG_LNSTYL"); break;
      case(224): outtext("CHNG_FLPTRN"); break;*/
      default  : break;
      }

   }
   }
```

```
void mini_menu::submenu()
      {

      int a,b,c,d;
      a = 0;c = 60;
      setcolor(RED);
      setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
      setfillstyle(SOLID_FILL,RED);
      floodfill(5,5,BLACK);

      for(int i=0;i<=6;i++)
      {
      rectangle(a, 0, c, 30);
      moveto(a+10, 15);

      switch(i)
      {
      case 0 :setcolor REEN);outtext("HELP");setcolor(RED);break
      case 1 : setcolor(RED);outtext("CIRKL");break;
      case 2: setcolor(RED);outtext("RECT");break;
      case 3 :setcolor(RED);outtext("TRIA");break;
      case 4 :setcolor(RED);outtext("POLY");break;
      case 5 :setcolor(RED);outtext("LINE");break;
      case 6 :setcolor(RED);outtext("TEXT");break;
      }

      a=a+95;c=a+60;
      }
      }
```

```
   // MENU ERASE FILE

class mini_menuer

{

 public:

  void help_menu1();
  void wrttext1();
  void xerasemenu();
  void lerasemenu();
  void cerasemenu();
  void rerasemenu();
  void terasemenu();
  void perasemenu();
  void gen_erase_menu();

  void openfile()
    {
     fstream file;
     file.open("PERSON.DAT",ios::app | ios::out | ios::in );
     }

    void add_file()
     {
     fstream file;
     file.write( (char*)&c1,sizeof(c1) );
     }

  void closefile()
    {
    fstream file;
    file.close();
    }

  void hor_menu()
   {
    moveto(5,43);
    setcolor(WHITE);outtext("LOCATION");
    }

  void paste_menu()
    {
    moveto(5,43);
    setcolor(WHITE);outtext("LOCATION");
    }

  void eng_menu()
  {
    moveto(5,43);
    setcolor(WHITE);outtext("NLARGMNT");
   }
```

```
void rotat_menu()
{
  moveto(5,43);
  setcolor(WHITE);outtext("W.R.T");
  moveto(5,64);
  setcolor(WHITE);outtext("ANGLE");
 }


void erase_menu()
{ moveto(5,43);setcolor(WHITE);outtext("Y"); }


void chng_lnclr()
{ moveto(5,43);setcolor(WHITE);outtext("CHN_LNCLR"); }


void chng_flclr()
{  moveto(5,43);setcolor(WHITE);outtext("CHN_FLCLR"); }


void chng_flptrn()
{  moveto(5,43); setcolor(WHITE);outtext("FLPTRN"); }

 void updn_menu1()
   {
          moveto(5, 43);setcolor(RED);outtext("DRAW");
          moveto(5, 64);setcolor(RED);outtext("MOVE");
          moveto(5, 85);setcolor(RED);outtext("NLARGE");
          moveto(5, 106);setcolor(RED);outtext("ROTAT");
          moveto(5, 127);setcolor(RED);outtext("ERASE");
          moveto(5, 148);setcolor(RED);outtext("PASTE");
          moveto(5, 169);setcolor(RED);outtext("CHN_LNCLR");
          moveto(5, 190);setcolor(RED);outtext("CHN_FLCLR");
          moveto(5, 211);setcolor(RED);outtext("FLPTRN");
          moveto(5, 232);setcolor(RED);outtext("LNSTYL");
   }
```

REMARK:

Similar looking updn_menus numbering 2,3,4,5 have not been incorporated here as they are very similar to the above one.

```
void help_menu2()
  {
  setcolor(WHITE);
  setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
  rectangle(0, 453, 630, 470);
  setfillstyle(SOLID_FILL, WHITE);
  floodfill(10, 465,WHITE);
  setcolor(WHITE);
  }
};
```

```
void mini_menuer::help_menu1()  //This erases the menu for HELP
{
setcolor(BLACK);
setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
rectangle(0, 35, 78, 450);
setfillstyle(SOLID_FILL, BLACK);
floodfill(10, 45, BLACK);
setcolor(BLACK);
for(long int li=35;li<=450;li+=21)
{
moveto(0, li);
lineto(78, li);
moveto(5, li+8);
 }
setcolor(WHITE);
setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
rectangle(0, 35, 78, 450);
setfillstyle(SOLID_FILL, WHITE);
floodfill(10, 45, BLACK);
  }

void mini_menuer::wrttext1()
{
for(long int li=35;li<=450;li+=21)
   {
   moveto(5,li+8);
   switch(li)
        {
        case(35) :setcolor(BLACK);outtext("Left");break;
        case(56):setcolor(BLACK);outtext("Right");break;
        case(77):setcolor(BLACK);outtext("Up");break;
        case(98):setcolor(BLACK);outtext("Down");break;
        case(119):setcolor(BLACK);outtext("Erase");break;
        case(140):setcolor(BLACK);outtext("Kill");break;
        case(161):setcolor(BLACK);outtext("Nlarge");break;
        case(182):setcolor(BLACK);outtext("roTate");break;
        case(203):setcolor(BLACK);outtext("VertMov");break;
        case(224):setcolor(BLACK);outtext("HortMov");break;
        case(245):setcolor(BLACK);break;
        default  :setcolor(BLACK);break;
        }
   }
   }
```

REMARK:
Similar looking erase menus exist for all other primitives.But these have been erased to conserve space.

```
                    //This erases the general menu

 void mini_menuer::gen_erase_menu()
 {
 setcolor(WHITE);
 setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
 rectangle(0, 35, 78, 450);
 setfillstyle(SOLID_FILL, LIGHTGRAY);
 floodfill(10, 45, WHITE);
 setcolor(WHITE);

for(long int li=35;li<=450;li+=21)
 {
 moveto(0, li);
 lineto(78, li);
 moveto(5, li+8);
  }

   moveto(5,43);
   setcolor(LIGHTGRAY);outtext("LOCATION");
   moveto(5,43);
   setcolor(LIGHTGRAY);outtext("NLARGMNT");
   moveto(5,43);
   setcolor(LIGHTGRAY);outtext("W.R.T");
   moveto(5,43);
   setcolor(LIGHTGRAY);outtext("CHN_LNCLR");
   moveto(5,43);
   setcolor(LIGHTGRAY);outtext("CHN_FLCLR");
   moveto(5,43);
   setcolor(LIGHTGRAY);outtext("FLPRTN");
   moveto(5,43);
   setcolor(LIGHTGRAY);outtext("FLSTYL");
   moveto(5,64);
   setcolor(LIGHTGRAY);outtext("ANGLE");
   moveto(5,43);
   setcolor(LIGHTGRAY);outtext("Y");

 }
```

```
//SHAPE FILE

   //CLASS SHAPE

class shape
  {

  protected:

      int Xco, Yco,w,n, vm, hm,h,b,c,x,y,X1,Y1,
              X2,Y2,X3,Y3,X4,Y4,X,Y,W;
          int linecolor,linestyle,linewidth,pattern;
          int fillcolor,color,font,direction,
          horzjustify ,vertjustify, multx ,multy,
      size, divx ,divy ;
          char str[50];

  public:

       shape ()
            { linecolor=WHITE;fillcolor=WHITE;}


  //   The  following  SET functions set  the
  //   values  for  the apprioate  variables


      void set1(int x, int y, int w, int cLc, int cFc)
        {Xco = x;Yco = y;W = w;
          linecolor = cLc;fillcolor = cFc;}

      void set11(int x, int y, int w, int cLc1, int cFc1)
        {Xco = x;Yco = y;W = w;
           linecolor = cLc1;fillcolor = cFc1;}

      void set2(int x1,int y1,int x2,int y2,int x3,int y3,
                int tLc,int tFc)
        {
        X1 = x1;Y1 = y1;X2 = x2;Y2 = y2;X3 = x3;Y3 = y3;
        linecolor= tLc;fillcolor=tFc;
        }

      void set21(int x1,int y1,int x2,int y2,int x3,int y3,
                 int tLc1,int tFc1)
        {
        X1 = x1;Y1 = y1;X2 = x2;Y2 = y2;X3 = x3;Y3 = y3;
        linecolor =tLc1 ;fillcolor =tFc1;
        }

      void set3(int x1,int y1,int x2,int y2,int x3,
                        int y3,int x4,int y4,int rLc,int rFc)
        {X1 = x1;Y1 = y1;X2 = x2;Y2 = y2;
        X3 = x3;Y3 = y3;X4 = x4;Y4 = y4;
        linecolor = rLc; fillcolor = rFc;
        }
```

```
void set31(int x1,int y1,int x2,int y2,int x3,
                  int y3,int x4,int y4,int rLc1,int rFc1)
  {X1 = x1;Y1 = y1;X2 = x2;Y2 = y2;
  X3 = x3;Y3 = y3;X4 = x4;Y4 = y4;
  linecolor = rLc1; fillcolor= rFc1;
  }

void set4(int ls,int w,int j,int lc,int fc)
   { linestyle = ls;linewidth = w;pattern = j;linecolor = lc;
     fillcolor = fc; }

void set41(int ls,int w,int j,int lc1,int fc1)
   { linestyle = ls;linewidth = w;pattern = j;linecolor = lc1;
     fillcolor = fc1; }

void set5(int x1,int y1,int x2,int y2,int llc)
    { X1 = x1;Y1 = y1;X2 = x2;Y2 = y2;linecolor = llc;}

void set51(int x1,int y1,int x2,int y2,int llc1)
    { X1 = x1;Y1 = y1;X2 = x2;Y2 = y2;linecolor = llc1;}

void set6(int x,int y,int f,int d,int z,int c,
           int hz,int vt,int mx,int my,int dx,int dy,char s[])
   { X = x;Y = y;font = f;direction = d;
   size = z;color = c;
  horzjustify = hz;vertjustify = vt;
  multx = mx;multy = my;
  divx = dx;divy = dy;strcpy(str,s);
  }

void set61(int x,int y,int f,int d,int z,int c,
           int hz,int vt,int mx,int my,int dx,int dy,char s[])
   { X = x;Y = y;font = f;direction = d;
   size = z;color = BLACK;
  horzjustify = hz;vertjustify = vt;
  multx = mx;multy = my;
  divx = dx;divy = dy;strcpy(str,s);
  }

  void drawtext3()
{
  settextjustify(horzjustify, vertjustify);
  setusercharsize(multx, divx,multy, divy);
   setcolor(color);
}

void draw()
{
  setcolor(linecolor);
  setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
  setfillstyle(SOLID_FILL,fillcolor);
  setusercharsize(1,1,1,1);
  }
```

```
void draw1()
{
setcolor(linecolor);
setlinestyle(SOLID_LINE,0,NORM_WIDTH);
}

void draw2()
 {
 setfillstyle(pattern,fillcolor);
 setlinestyle(linestyle,0,linewidth);
 setcolor(linecolor);
 }

void drawtext1()
 {
 setusercharsize(multx, divx,multy, divy);
 setcolor(color);
 outtext(str);
 }

double calculate1(int x1,int y1,int x2,int y2,int a)
 {
 double k,a2;
 a2 = double(a)/double(57);
 k = x1 * cos(a2) - y1 * sin(a2) -
        x2 * cos(a2) + y2 * sin(a2) + x2;
 return k;
 }

double calculate2(int x1,int y1,int x2,int y2,int a)
 {
 double k,a2;
 a2 = double(a)/double(57);
 k = x1 *sin(a2) + y1*cos(a2) -
          x2*sin(a2) -y2*cos(a2) +y2;
 return k;
 }

//This returns the font.
   int getfont();

/* font_names                    <GRAPHICS.H>

Enum: Names for BGI fonts

     Name         3Value3 Meaning

 DEFAULT_FONT     3   0   3 8x8 bit-mapped font
 TRIPLEX_FONT     3   1   3 Stroked triplex font
 SMALL_FONT       3   2   3 Stroked small font
 SANS_SERIF_FONT3     3   3 Stroked sans-serif font
 GOTHIC_FONT      3   4   3 Stroked gothic font

                                                    */
```

```
int getdir();


    //This reads integer values from the key_board.

int getval1()
  {
   int n;
   char *str;
   moveto(20, 450);
   gets(str);
   n = atoi(str);
   return n;
   }

    //Similarly, this reads integer values from the key_board.
    int getval();


    //This function returns color.
   int getcolor();

    //This function returns the fillpattern.
   int getfillpatrn();

    //This function returns the linestyle.
   int getlinestyle();

    //This function returns the text_set_style.
   int getsettext();

   };

        int shape::getfont()
        {
        int a,f;
        a = getch();
        switch(a)
        {                          //  fonts
        case('d'):f = 0;break;     //  "DEFAULT font",
        case('t'):f = 1;break;     //   "TRIPLEX font",
        case('m'):f = 2;break;     //   "SMALL font",
        case('n'):f = 3;break;     //   "SANS SERIF font",
        case('g'):f = 4;break;     //   "GOTHIC font"
        case(13) :f = 0;break;     //  "DEFAULT font",
        }
        getch();
        return f;
        }
```

```
int shape::getdir()

{

int a,d;
a = getch();

switch(a)
{
 case('h'):d = 0;break;          //HORIZONTAL DIRECTION
 case('v'):d = 1;break;          //VERTICAL DIRECTION
 case(13):d = 0;break;          //HORIZONTAL DIRECTION

 }
 getch();
 return d;
 }
```

```
int shape::getval()

 {

 int g,g1,t;
 char A[3];
 int i = 0;
 moveto(20,400);

 while ( (t = getch() ) != 13)
 {
  A[i] = t;
 i++;
 }

  g1 = atoi(A);
  return g1;
 }
```

```
int shape::getcolor()
{
char t;
int p;
t = getch();

switch(t)
{
case('k') :p = 0;break;
case('b') :p = 1;break;
case('g') :p = 2 ;break;
case('c') :p = 3 ;break;
case('r') :p = 4 ;break;
case('m') :p = 5 ;break;
case('a') :p = 7 ;break;
case('n') :p = 20 ;break;
case('y') :p = 56 ;break;
case('u') :p = 57 ;break;
case('e') :p = 58 ;break;
case('l') :p = 59 ;break;
case('d') :p = 60 ;break;
case('i') :p = 61 ;break;
case('w') :p = 62 ;break;
case('h') :p = 63 ;break;
case(13)  :p = 0  ;break;

}
getch();
return p;

}
/* EGA_ COLORS (graphics mode)
\\\\\\\\\\\\\\\\\\\\\\\\\\\\

 Constant       3Value:
MMMMMMMMMMMMMM  XMMMM
 EGA_BLACK        3  0
 EGA_BLUE         3  1
 EGA_GREEN        3  2
 EGA_CYAN         3  3
 EGA_RED          3  4
 EGA_MAGENTA      3  5
 EGA_LIGHTGRAY    3  7
 EGA_BROWN        3 20
 EGA_DARKGRAY     3 56
 EGA_LIGHTBLUE    3 57
 EGA_LIGHTGREEN    3 58
 EGA_LIGHTCYAN     3 59
 EGA_LIGHTRED      3 60
 EGA_LIGHTMAGENTA 3 61
 EGA_YELLOW       3 62
 EGA_WHITE        3 63  */
```

```
int shape::getfillpatrn()
       {
       char t;int p;
       t = getch();
      switch(t)
      {                                    //fill_patterns
      case('e') :p = 0;break;              //EMPTY_FILL
      case('d') :p = 1;break;              //SOLID_FILL
      case('l') :p = 2 ;break;             //LINE_FILL
      case('t') :p = 3 ;break;             //LTSLASH_FILL
      case('s') :p = 4 ;break;             //SLASH
      case('b') :p = 5 ;break;             //BKSLASH_FILL
      case('k') :p = 6 ;break;             //LTBKSLASH_FILL
      case('h') :p = 7 ;break;             //HATCH_FILL
      case('x') :p = 8 ;break;             //XHATCH_FILL
      case('i') :p = 9 ;break;             //INTERLEAVE_FILL
      case('w') :p = 10 ;break;            //WIDE_DOT_FILL
      case('c') :p = 11 ;break;            //CLOSE_DOT_FILL
      case(13)  :p = 0;break;              //EMPTY_FILL
      }
      getch();
      return p;
       }

    int  shape::getlinestyle()
       {
       char t;int p;
       t = getch();
      switch(t)
      {                                    //line styles
      case('s') :p = 0;break;              //SOLID_LINE
      case('d') :p = 1;break;              //DOTTED_LINE
      case('c') :p = 2 ;break;             //CENTER_LINE
      case('h') :p = 3 ;break;             //DASHED_LINE
      case('u') :p = 4 ;break;             //USERBIT_LINE
      case(13)  :p = 0;break;              //SOLID_LINE
      }
      getch();
      return p;
      }

   int shape::getsettext()
      {
      char t;int p;
      t = getch();
      switch('t')
      {                          //text_set_styles
      case('l'):p= 0;break;      //LEFT_JUSTIFY
      case('c'):p=1;break;       //CENTRE_JUSTIFY
      case('r'):p=2;break;       //RIGHT_JUSTUFY
      case(13) :p= 0;break;      //LEFT_JUSTIFY
       }
       getch();
       return p;
       }
```