1981

Draw-a-map: An Interactive graphical system using a relational data base

Ethel Comte
DRAW-A-MAP:
An Interactive Graphical System Using
a Relational Data Base

A thesis submitted in partial fulfillment of the
Master of Science in Computer Science

Ethel C. Comte
May, 1981
This is to certify that _________________ has submitted a thesis entitled:


to the faculty of the School of Computer Science and Technology in partial fulfillment of the requirements for the degree of Master of Science.

Approval:  

Name Illegible  
(thesis advisor)  

29 June 81  
(date)  

Name Illegible  
(committee member)  

29 June 81  
(date)  

Name Illegible  
(committee member)  

29 June 81  
(date)
Table of Contents

Introduction .................................................. 1
1. Review of the Literature .............................. 3
   1.1. Early Work in Data Structures Used in Graphics  ................. 3
   1.2. Cartographic Systems .............................. 6
   1.3. Use of Relational Data Bases for Graphic Data ....................... 8
2. Data Base Considerations .............................. 12
   2.1. Data Base Design and the Structure of Graphical Images ........... 12
   2.2. Selecting the Data Base Model .......................... 16
   2.3. Data Base Design - DRAW-A-MAP .......................... 15
3. Graphic Considerations ................................. 25
   3.1. Representation of Graphic Data .......................... 25
   3.2. Data Structures for Graphic Data .......................... 26
   4.1. Data Base Interface .................................. 30
   4.2. Graphical Device Interface ............................ 31
   4.3. Structure of DRAW-A-MAP ................................ 31
      4.3.1. Controlling Program ................................ 31
      4.3.2. Draw ........................................... 32
      4.3.3. Edit ........................................... 33
      4.3.4. Inquire ........................................ 34
      4.3.5. Graphic Utilities ................................ 35
5. Enhancements ............................................. 35
6. Conclusion ............................................... 40
Appendix A: Structural Model .............................. 42
Appendix B: Data Base Design .............................. 43
Appendix C: Source Code .................................... 46
   C.1 Data Structure ...................................... 46
   C.2 mrscmd, mrscmd1, mrscmd2 ................................ 49
   C.3. Readmap, DeleteMap, Savemap ........................... 50
   C.4 Iomod ............................................... 65
   C.5 Main ................................................ 76
   C.5 Draw ................................................ 73
   C.7 Inquire ............................................. 103
   C.8 Getslope, yintercept .................................. 108
   C.9 Bound_box ........................................... 110
   C.14 Introl, intro2, draw1, draw2 ......................... 125
   cted Bibliography ...................................... 131
Introduction

Recent developments in hardware technology have decreased the cost of storage and manipulation of data. As the cost/benefit ratio has improved, more work has been done in the areas of computer graphics and data base technology. This thesis demonstrates how these two areas can be made to complement to one another by using a data base to store data from a system which manipulates graphical and non-graphical data.

In brief, the focus of this thesis is an interactive graphics system, DRAW-A-MAP, which permits a user at a graphics terminal to draw a map on the screen, enter appropriate information about the map and its component parts, and store the map in a data base. At a later point in time, a user may request that the map be extracted from the data base and may make inquiries about the characteristics of the areas shown on the map.

Section 1 of this thesis discusses previous work in storing graphical data and explores the change from the storage of graphical data in data structures to the storage of such data in data bases. This discussion also features previous projects relating to map generation and the storage of data associated with the map.
Section 2 focuses on the data base aspect of the project. It discusses the demands imposed by computer graphics on data base technology, and the data base model that appropriate to computer graphics. It includes the features and conceptual schema of the data base for the DRAW-A-MAP system.

The graphical aspects of DRAW-A-MAP are considered in Section 3. This section involves a discussion of two techniques used to represent graphical data, their advantages and disadvantages, and the qualities a data structure must possess to represent graphical information. A discussion of the actual data structure used in DRAW-A-MAP follows.

The actual implementation of the DRAW-A-MAP system is the subject of Section 4. It includes the structural model of the system and discusses the user, the graphical and the data base interface.

The next section, Section 5, discusses enhancements that would make the system more "user friendly", more useful, and more efficient. It also describes how DRAW-A-MAP might be enhanced to become a production cartographic tool.

Section 6 concludes with an overview of the thesis.

The appendices follow: Appendix A contains the structural model of the system, Appendix B, a model of the data base design, and Appendix C, source code.
1. **Review of the literature**

Robin Williams [1] in a review of data structures used for computer graphics, states that the data structures for computer graphics must possess the following qualities:

1. The data must be ordered, particularly in the case of line drawings where a coordinate's position in the data structure implies a graphical meaning, i.e., where and when the line will be drawn.
2. The data must be structured to allow the necessary searching and accessing of related data. For example, when a line on the screen is identified, the representation of the line in the data structure must be located with relative ease.
3. The data structure must be flexible enough to allow the updating procedure to be performed quickly.
4. The data structure must be able to handle the large amounts of data generated by computer graphics.

The following articles trace the efforts to achieve the above qualities in data structures.

1.1. **Early Work in Data Structures Used in Graphics**

In the first significant work done in computer graphics, Sutherland [2] in Sketchpad used a ring structure to contain graphical information. In his classic thesis, he demonstrated how screen images could be drawn and stored.

---


interactively. He introduced the concept of instances, i.e., one or more occurrences of a subpicture on a screen. This concept of subpictures is similar to the concept of graphical building blocks. Using a ring type data structure to support these instances, he related the concept of hierarchies in data structures to the graphical image itself.

Data structure flexibility is a very important aspect of handling graphic data efficiently. This flexibility was a result of the development of languages which had capabilities to manipulate and build list structures and to express data relations inherent in the data itself. The following discussion of L6, DSPL, APL and LEAP represent early attempts at creating this flexibility.

Early work involved the development of languages to support list type data structures. L6 [3] and DSPL [4] were low level languages developed to support data structures of the list type. These languages contained blocks (a sequential number of machine words) and fields (a sequential number of bits) and pointers (a word address). The user was


able to create and name blocks and fields and could inter-
connect them with pointers to form structures. DSPL con-
tained a paging facility to handle the large amounts of data
with relative ease.

Higher level languages were also developed to support
flexible data structures. Dodd [5] added six statements to
PL/I creating APL (Associated Programming Language). The
basic elements of APL were entities which were described by
attributes. Related entities were then grouped into sets.
References to entities were made through the entities them-
selves or through a set which owned the entity.

LEAP, a language developed to express data relations
in associative structures, was developed in 1968 by Feldman
and Rovner [6]. Leap itself was an extension of Algol with
statements added to access the data by hash coding tech-
niques. In LEAP, data associations were of the form

\[ \text{Attribute of Object is Value} \]
\[ \text{or} \quad A(0) = V \]

---[5] G.G.Dodd, "APL- a Language for Associative Data Han-
Data Structures for Computer Graphics Systems," ACM Compu-
ing Surveys, 3 (March 1971), p. 10.

Robin Williams, "A Survey of Data Structures for Computer
10-11.
an example of which is father (jack) = jim.

Data in Leap is stored on an A page, an O page, and a V page. Items (jack and jim) are treated as integers. Hash coding techniques were used to store the triple on a page of each type (A, O, V). The effect of this technique was that storage and retrieval of data was fast and did not require extensive pointer chasing.

The above languages and data structure techniques represent a few of many that were explored during the late sixties and early seventies. The late seventies saw an emergence of graphical systems built on techniques developed during the early periods. These systems handled graphical information drawn from traditional files.

\[1.2. \text{Cartographic Systems}\]

In 1977, an extensive cartographic system, the Crime Analysis and Research Package (CARP), was developed at the Geographic Information Systems Laboratory at the State University of Buffalo.\[7\] This system drew upon two files containing census and crime data. A third file contained such spatial information as geometric indices of census block groups, census tract boundaries, and centroids of statistical units. These three files, which were organized in

indexed sequential and random access fashion, were used to generate maps on a graphic device. The generated maps revealed graphically how census data and crime data were related.

Williams [8] reports a similar project developed at the IBM Research Laboratory in San Jose which was used to allocate manpower in the San Jose police department so that each beat had an equal work load. A zone map which divided the area into small zones was used with data collected on housing, schools, traffic and police activities. The system generated a revised zone map which allocated police resources more equally.

Another project, the Domestic Information Display System [9], was built to generate choropleth maps which depicted socioeconomic characteristics of geometrically defined areas for the federal government. In this system the geometric entities (census tracts) were kept by their raster images. These rasters were colored depending on what characteristics that geometric area had. To improve response


time, data for this system was extracted from large data files resident on a main frame and transmitted to a minicomputer on which the graphics system ran.

1.3. Use of Relational Data Bases for Graphic Data

The year 1973 marked the appearance of graphical data stored in a data base. Mcintosh [10] (in a project very similar to the DRAW-A-MAP system) used a relational data base to store a map generated by users on a graphical device. He stored the map in terms of nodes used to draw the map but did not provide the facility for nongraphical data to be stored with the map.

In another project Bonfatti and Tiberic [11] stored a map in a relational data base but its representation was quite different from that of Mcintosh and DRAW-A-MAP. They used a bit map technique to store maps and nongraphical attributes in a data base. They subdivided the territory being studied into geographical cells. Attribute values were subdivided into classes and when a cell contained an attribute of a certain value, its bit was turned on. Therefore, for each class of attribute values there existed a bit map that indicated which geographical cells would be described

by that attribute. Map generation consisted of cell by cell logical operations performed on the bit maps. Data from which the bit maps were generated was stored in a relational data base.

Additional work was done by Uno [12] in 1979 enhancing our notion of the data structure features desirable in graphical data base systems. He developed A-IDAS (Advanced Integrated Design Activity Support System), in which graphic data, geometric data and engineering data were stored in a relational data base. The building block approach was used in that he divided every geometric element into four geometric classes, i.e., vertex, edge, face and body. Each class consisted of several geometric application dependent categories or "legions", i.e., a point, an arc, a circle etc. which were stored in a relation. Another relation stored linking information which linked edges, vertices and faces. Various combinations of linkages and legions were used to generate a screen image of the item so described. The linkage relation also contained parameters which described line style, color, and other graphic symbols to be drawn with the object. Drawing routines were used to draw screen images directly from the data base. This system also provided a device independent graphic management facility thus making the graphic device independent of the data base.

Weller and Williams [13] described a picture building system which allowed a user to interactively create and modify images. The system contained a graphics software package and a relational data base. The data base (XRM) was self-describing, i.e., it provided semantic (how the data is to be used) as well as the traditional syntactic (how the data is stored and accessed) support. This feature allowed the data base to be read by an interpreter which passed commands directly to the display file that generated the pictures on the screen. The data base/interpreter process provided for the operations of rotating, scaling, translating, coloring, setting intensity and line style, manipulating text, and creating transformations from the real world data space into the screen data space, to be done purely from data stored in the data base and not as traditionally done under program control. It also provided an efficient mechanism for correlation, i.e., the operation of identifying an item by pointing to its representation on the display terminal. This operation was performed using a correlation table that contained all the items on the screen with their locations in the data base.

In this thesis, DRAW-A-MAP defines a map as a collection of subareas and attributes specified by the user. Each

subarea is defined by the vertices that determine its graphical image, its name, and the attribute values which describe the characteristics of the subarea. The data for each subarea is stored in a relational data base and may be retrieved on demand. For efficiency in drawing and manipulating the map, the data base also stores, for each subarea, a representative point in the subarea and the coordinates of its surrounding box. For each vertex of the subarea, the slope and y intercept of the line departing from that vertex is stored as well as the representation of that point in user-defined world coordinates.

Building on the past and present techniques of storing graphical data, the next section will look at what must be considered in determining what storage techniques to use.
2. **Data Base Considerations**

2.1. **Data Base Design and the Structure of Graphical Images**

In designing a data base management system to contain graphic data, the characteristics of graphic processes must be considered. By incorporating features in the data base design which support graphic processes, the resultant data base will enhance the execution of those graphic processes. The example of the screen image of a building will illustrate these processes as follows:

1. Graphics elements are defined as the basic building blocks of screen images. As such, they are primitives. An example might be a window in a building.

   ![Window](image)

2. Primitives may be combined with other images to form entities (more complex graphical images). As such, primitives themselves remain distinctly defined but are part of an entity when the entity definition contains a 'call' to them. An example of an entity would be windows with shutters.

   ![Entity](image)

The entity would contain two references to the primitive describing windows as well as two references to a primitive describing shutters.
3. Graphical images are built by the process of grouping entities in a hierarchical manner. In the building example, a window with shutters might be part of the front of a house.

![Image of a house with windows and shutters]

4. Graphic entities and primitives frequently have, associated with them, nongraphical data as well, i.e. the name of the manufacturer of the windows, their model number, their price and dimensions.

5. Graphical entities are subject to change in an interactive environment. In the building example, the dimensions of the windows or the window style might change completely, requiring that the primitive be changed.

A tabular implementation of the primitive/entity structure would be as shown in Figure 1. X and Y are relative coordinates that define the shape of the window in the coordinate space near the origin. The mode provides the name of the graphic operation to be performed in connection with its associated coordinate, i.e., "move" means to make the coordinate the "current position" on the screen, while "draw" also means to make the coordinate the "current position" but in doing so, draw a line from the previous "current position" to the new "current position".
A composite screen image is generated through a table which contains a reference to primitive structures along with information regarding their placement on the screen. For example a table that would generate an image of two shutters beside two windows might appear as shown in Figure 2. This table is an entity which describes the instance of an image composed of two windows and two shutters. Scaling and translating factors control the placement of the windows and shutters on the screen, while the color and linestyle define graphical characteristics the lines are to have. The column “structure” actually contains references to primitive tables which give precise direction on how the respective images are to be drawn.

The facility for storing nongraphical information is shown in Figure 3. The table, window-shutter-info, contains
Table Window_with_shutters

<table>
<thead>
<tr>
<th>Structure</th>
<th>Scalex</th>
<th>Scaley</th>
<th>Translatex</th>
<th>Translatey</th>
<th>Color</th>
<th>Linestyle</th>
</tr>
</thead>
<tbody>
<tr>
<td>shutter</td>
<td>7</td>
<td>7</td>
<td>100</td>
<td>50</td>
<td>Green</td>
<td>Thick</td>
</tr>
<tr>
<td>window</td>
<td>10</td>
<td>10</td>
<td>150</td>
<td>50</td>
<td>Red</td>
<td>Thin</td>
</tr>
<tr>
<td>window</td>
<td>10</td>
<td>10</td>
<td>200</td>
<td>50</td>
<td>Red</td>
<td>Thin</td>
</tr>
<tr>
<td>shutter</td>
<td>7</td>
<td>7</td>
<td>250</td>
<td>50</td>
<td>Green</td>
<td>Thick</td>
</tr>
</tbody>
</table>

Figure 2

such nongraphical information as cost, manufacturer and dimensions. Note that the structure column contains a reference to another table, i.e., window. This is a reference to the window table which dictates the actual image generation of the window.

In selecting a data base management system to support the above structure, several additional requirements need to

Table Window-shutter-info

<table>
<thead>
<tr>
<th>Structure</th>
<th>Cost</th>
<th>Manufacturer</th>
<th>Height</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>window</td>
<td>30.00</td>
<td>Anderson</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>shutter</td>
<td>12.50</td>
<td>Ace</td>
<td>40</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 3
be considered. The first two imply the relaxation of certain assumptions at the primitive level, while the third applies to levels other than the primitive level. These are as follows:

1. In defining the primitives, the ordering of coordinates has specific meaning and, therefore, must be preserved. This, of course, may be avoided by having every line defined by both its beginning and ending coordinates, however this approach would require more storage and effort and is not necessary when ordering is preserved.

2. In defining primitives, duplicate records must be permitted because the same coordinates may be "visited" twice in an image.

3. Attributes of the data base must be permitted to contain the names of relations. As such, a hierarchy may be formed which permits entities to contain calls to primitives as well as calls to other entities.

2.2. Selecting the Data Base Model

A relational data base system seems to be the data base model most appropriate to computer graphics for the following reasons:

1. It has a simple structure, devoid of the fan and fan set structures (parent-child ordering constructs) found in hierarchical and network models. The absence of complex pointer schemes make the operations of data extraction and data integration relatively efficient. Consequently, the language
features, operators, and operands for accessing the data are simple, i.e. only one operator is needed for each of the basic functions: retrieve, insert, delete, and update.

2. The relational model provides for "symmetric exploitation: the ability to access a relation by specifying known values for any combination of its attributes, seeking the (unknown) values for its other attributes. Symmetric exploitation is possible because all information is presented in the same way".[14]

3. Relational data bases permit different logical views of the data which, in a complex system, would be useful to users with diverse interests.

4. For the nonsophisticated user, the relational model provides an easily understood data image, a table -- a structure with which most users are already familiar.

5. Relational data bases can support the graphics processes stated above. By permitting attributes whose domains contain relation names, hierarchies may be built.

6. Relational data bases offer the flexibility required by computer graphics. The following operations may be done with ease:

a. New relations may be added.
b. Additions may be made to existing relations.
c. Tuples within relations may be deleted.
d. Relations may be deleted.

The foregoing has specified the requirements that graphics places on data bases and has shown the advantages of a relational type data base. It must be noted that using a relational data base for the reasons given above and as suggested by Williams et. al. /** implies the relaxation of the assumptions of non-ordering and non-duplicity of tuples. Consequently, the operations of select, join and project are no longer available to the user. In DRAW-A-MAP these assumptions are relaxed only at the primitive level. At this level, relations are always used as a whole, i.e., each tuple is only meaningful as part of the entire relation. Likewise, the domains of the relation are meaningless standing alone. Therefore, at the primitive level the operations of select, join and project are not required and the appropriate assumptions may be relaxed. The next section describes how the DRAW-A-MAP data base is designed.

2.3. Data Base Design - DRAW-A-MAP

In the DRAW-A-MAP system, the graphical image of a map is drawn on a graphical device. This map, actually a polygon subdivided into subpolygons, is stored in the data base. In the creation of the map the user must indicate to the system what non-graphical types of information (attributes) will be

stored for each subarea of the map. Examples of attributes might be population, average income, median age, housing stock, crime activity indicators, etc. The map is then drawn on the screen, appropriate information pertaining to the specified attribute is entered and the map is stored in the data base.

The particular data base design used in the DRAW-A-MAP system is shown in Appendix B and Figures 4 thru 8.

The relation, "mpdir", shown in Figure 4, contains an entry for each map in the data base. This relation provides the name of the map (MPNAME), the author (AUTHOR), the name of the relation containing the attribute table (ATTRTBL), the number of attributes (NOATTR), and the number of subareas in the map (NOSUBA). It should be noted that two of the attributes contain relation names, i.e., the name of the map is used as a "call" to the relation containing the subarea data and the attribute table is a "call" to the

---

Relation mpdir

<table>
<thead>
<tr>
<th>MPNAME</th>
<th>AUTHOR</th>
<th>ATTRTBL</th>
<th>NOATTR</th>
<th>NOSUBA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monroe</td>
<td>Smith</td>
<td>Mon.atbl</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Ontario</td>
<td>Jones</td>
<td>Ont.atbl</td>
<td>4</td>
<td>12</td>
</tr>
</tbody>
</table>

---

Figure 4
relation containing the attribute table.

The attribute table relation shown in Figure 5, contains the names of the user-defined attributes (ANAME) that will be stored for each subarea of the map. It also contains an attribute (ATYPE) which indicates the type of the attribute ('n' for numeric or 'c' for character). In the case of a numeric attribute low (ALOW) and high (AHIGH) values are stored to form a range of valid values.

The relation called by the map name, shown in Figure 6, contains information about the subareas of the map. This relation represents a graphic entity, as discussed above. As such it contains graphic information about the entity and references to relations containing primitives (SCOORD and WCOORD). The graphic information includes X and Y, coordinates of a point representing the subarea (used for filling the subarea); COLOR, the color of the subarea; BBXMIN,BBYMIN,BBXMAX,BBYMAX, minimum and maximum x and y

<table>
<thead>
<tr>
<th>ANAME</th>
<th>ATYPE</th>
<th>ALOW</th>
<th>AHIGH</th>
</tr>
</thead>
<tbody>
<tr>
<td>population</td>
<td>n</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>area</td>
<td>n</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>income</td>
<td>n</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>party</td>
<td>c</td>
<td>(null)</td>
<td>(null)</td>
</tr>
</tbody>
</table>

Relation Mon.atbl

---

Figure 5
Relation Monroe

<table>
<thead>
<tr>
<th>SURNAME</th>
<th>SCOORD</th>
<th>WCOORD</th>
<th>NNODES</th>
<th>X</th>
<th>Y</th>
<th>COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rochester</td>
<td>Monro.s.2</td>
<td>Monro.w.2</td>
<td>4</td>
<td>230</td>
<td>365</td>
<td>green</td>
</tr>
<tr>
<td>Henrietta</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brighton</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Relation Monroe -- continued)

<table>
<thead>
<tr>
<th>BBXMIN</th>
<th>EBYMIN</th>
<th>BBXMAX</th>
<th>EBYMAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>220</td>
<td>315</td>
<td>280</td>
<td>335</td>
</tr>
</tbody>
</table>

(Relation Monroe -- continued)

<table>
<thead>
<tr>
<th>POPULATION</th>
<th>AREA</th>
<th>INCOME</th>
<th>PARTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>10</td>
<td>20</td>
<td>dem</td>
</tr>
</tbody>
</table>

Figure 6

values which define the bounding box that surrounds the subarea; and NNODES, the number of vertices in the subarea. The subarea relation contains a reference (SCOORD) to the relation containing screen coordinates which define the subarea on the screen, as well as a reference (WCOORD) to the relation containing world coordinates. This relation also contains the attribute values for the attributes specified by the user (in this example AREA, INCOME and PARTY).
The relation containing screen coordinates (called from SCOORD), shown in Figure 7, also contains information about the line departing from this node to the next node (in the next tuple) in the relation. Because in DRAW-A-MAP this structure represents a closed polygon, no graphical operations need to be defined, i.e., the first node is assumed to have to operation of "move" while the remaining nodes have the operation of "draw". The slope and y intercept of each line are kept in the data base to avoid the necessity of continually recalculating them since they are very frequently used in polygon editing, unioning, and intersecting functions. Because storage in MRS is limited to integers, the slope is actually stored by its value multiplied by 10, although its true value is shown here. For convenience infinity is defined as 9999.

A second primitive defining the subarea is kept in the relation referenced from WCOORD and shown in Figure 8. This

<table>
<thead>
<tr>
<th>SCRX</th>
<th>SCRY</th>
<th>SCRSLOPE</th>
<th>SCRYINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>220</td>
<td>350</td>
<td>.9</td>
<td>136</td>
</tr>
<tr>
<td>256</td>
<td>395</td>
<td>-.6</td>
<td>545</td>
</tr>
<tr>
<td>230</td>
<td>370</td>
<td>999.9</td>
<td>9999</td>
</tr>
<tr>
<td>280</td>
<td>315</td>
<td>-.5</td>
<td>473</td>
</tr>
</tbody>
</table>

Figure 7
Relation Monro.w.2

<table>
<thead>
<tr>
<th>WORLDX</th>
<th>WORLDY</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>26</td>
<td>37</td>
</tr>
<tr>
<td>30</td>
<td>34</td>
</tr>
<tr>
<td>70</td>
<td>23</td>
</tr>
</tbody>
</table>

Figure 8

This relation contains the coordinates (WORLDX and WORLDY) which define the subarea in a user-defined (world) coordinate system. Such a system might be latitude and longitude readings, mileage markers, etc. This is useful because it preserves the integrity of a map when different graphical devices are used.

The following provides a review the features of this data base design which makes it useful for graphics.

1. A building block structure is used. The screen and world coordinate relations are primitives. These are referenced from an entity type structure which contains other information about the graphic image of this entity, as well as nongraphical information about the subarea.

2. The maintenance of a world coordinate system makes the image transportable to other graphical devices.
3. It is flexible. More maps may easily be added to the directory (relation mpdir) and subareas may be redefined by replacing these relations by new relations.

Retrieval of the map is an operation which may be done with ease and will be thoroughly explained in a later section. Basically it consists of one reference to the relation mpdir, one reference to obtain the attribute table and the subarea relation, and a reference to obtain the screen and world coordinate relations for each subarea. In DRAW-A-MAP, maps cannot be subdivided for retrieval from the data base, therefore all the references to the data base involve obtaining whole relations, a time saving mechanism.

The next section will present the graphical aspects of the DRAW-A-MAP system.
3. Graphic Considerations

3.1. Representation of Graphic Data

Two techniques appear to provide methods for storing graphical images in a file or a data base; the polygon technique used by McIntosh [15] and the bit map technique used by Bonfatti and Tiberio. [16] (The reader is referred to the discussion of these techniques in Section 1.) The method selected for an application depends upon the nature of the application. The needs of the application in terms of storage, data entry, manipulation of data, and the purpose of the system must be considered.

Storage requirements for the bit map technique depend upon the number of attribute classes that are maintained while those for the polygon technique depend upon the amount of fragmentation of the edges of the polygon. Data entry using the bit map technique is very tedious as data must be entered for each attribute class within each geographic cell of the map.

Manipulation of the map itself must be considered. The bit map representation allows cell by cell logical operations for determining unions and intersections of the map, a very simple process. On the other hand, the polygon technique involves fairly complex algorithms which are quite

[16] Bonfatti and Tiberio, pp. 71-78.
time consuming.

Finally, and most important, the purpose of the map must be considered. The bit map technique best represents nonaggregated collections of data and therefore would be quite useful for analyzing such data. The polygon technique is more appropriate for aggregated data, as typically found in sources like the census.

The polygon technique has been chosen to represent the data in the DRAW-A-MAP system because of the requirements of the system. In terms of storage, maps drawn with DRAW-A-MAP are not necessarily fragmented and the polygon technique is supported. Data entry is by subarea and under this technique is a simple process. Since the major focus of maps drawn by DRAW-A-MAP is to illustrate aggregated data, a static process, as opposed to analyzing data graphically, graphical manipulation operations are infrequent and the polygon technique is the one of choice.

3.2. Data Structures for Graphic Data

The data structure used in DRAW-A-MAP is designed to serve the dual purposes of efficient interface with the database and ease in the execution of the algorithms necessary to perform polygon manipulating functions. The data structure of the DRAW-A-MAP system is shown in Figure 9 (its source code can be found in Appendix C.1). Note that the value of the attribute is shown in Figure 9 shortened to AV.
The arrows in the diagram represent pointers to structures.

The data structure consists of a singly linked list construct within a hierarchical structure. A singly linked list construct is used since concatenation of singly linked lists is easy, a process useful in combining attributes in

---

Organization of Structures in a DRAW-A-MAP
Figure 9
the case of map unions and in compiling a list of all the edges of a map for the process of subarea manipulation. The hierarchical aspect of the structure (maps contain a list of subareas and an attribute table, and subareas contain a list of attributes and a list of nodes, i.e., vertices) eliminates the need for a doubly linked list as the first node of a list may always be accessed directly from the level above.

Through pointers in the map structure, the program may access the attribute table (mattrtbl), the subarea list (msp), or the edge list (mpe) for the whole map. The map structure contains a pointer to the name of the map (map-name), a pointer to the author's name (author), a pointer to the name of the attribute table (mattrnm), the number of attributes for this map (mnoattr) and the number of subareas and nodes in this map (mtotsuba and mtotnode).

The attribute list contains information from the attribute relation (ANAME, ATYPE, ALOW, AHIGH) described in Section 2.3, as well as a pointer (atlnext) to the next entry in the list.

Each subarea structure contains the information described in Section 2.3 regarding subareas as well as pointers to the node list (spn), the edge list for that subarea (spe), the attribute list for that subarea (sattri) and the next subarea (snext).
The node structure contains the screen coordinates (NX and NY), slope (NSLOPE), and y intercept (NYINT) defined in Section 2.3. Since the world coordinates (WORLDEX and WORLDY) are a function of the screen coordinates and since they are not used in the manipulation of data, they are also maintained in the node structure as nwx and nwy. Nnext is a pointer to the next node.

The attribute structure contains the actual values for the attributes named in the attribute table. Each subarea has its own list of attribute structures that contain the values which describe the subarea. The attribute structure contains a pointer to the attribute value (attribute) and a pointer to the next attribute structure (anext). As such this list and the attribute table list must be maintained in like order.

The major advantages to this structure is its extreme flexibility in data manipulation. Edges and attribute lists may be combined or split without the processes of allocating and freeing memory. The next section will describe the actual implementation of DRAW-A-MAP.
4. Implementation of DRAW-A-MAP

The implementation of the DRAW-A-MAP system is modular in nature, allowing changes to be made without worry of unwanted side effects. The modules which communicate with the graphical device and the data base reside on separate files for easy adaptation to new data bases or graphical devices.

4.1. Data Base Interface

The data base, MRS, and DRAW-A-MAP communicate through a set of modules designed for that purpose. Requests generated by these modules are printed on a file named frequest. Control is given to an executable file named mrscmds (shown in Appendix C.2) which calls MRS and directs it to read the file, frequest, for input. MRS then executes the request, places the results on the file, fmapres, and returns control to the DRAW-A-MAP system. Several versions of this file are shown. Which version is used depends upon how many (0,1, or 2) files have been requested.

This process is controlled by the following modules;

1) Readmap (Appendix C.3) receives the name of the map, searches for the map in the data base and reads it into the DRAW-A-MAP data structure.

2) Savemap (Appendix C.3) receives a pointer to the map and stores the map in the data base.

3) Deletemap (Appendix C.3) receives the name of a map and deletes it from the data base.
by limiting the purpose of these modules to that of communication with the data base, the system can easily be modified to operate with other data base management systems.

4.2. Graphical Device Interface

The DRAW-A-MAP system is implemented using the Tektronics graphical terminal.

A set of modules (shown in Appendix C.4) contains the code necessary to communicate with the graphics device. This set contains modules to initialize the screen, to clear the screen, to clip a line against window boundaries using the Cohen-Sutherland clipping algorithm and draw it, to position the cursor on the screen, to read the cursor position, to read a character string from the keyboard, and to ring the bell.

4.3. Structure of DRAW-A-MAP

The structural model of the system is shown in Figure 10 and Appendix A. It consists of three primary modules; DRAW, EDIT and INQUIRE, which are controlled by a driver module (MAIN) and which communicate with the data base and graphical device via the modules described above.

4.3.1. Controlling Program

The driver program, MAIN (shown in Appendix C.5), controls the user's navigation through the system. Depending
4.3.2. Draw

The module DRAW (shown in Appendix C.6) allows the user to draw a map on the screen and enter it in the data base. The user is first prompted for the name of the map, followed by the attributes the map will have. To enhance drawing...
accuracy the user may request that the boundary of a previously stored map be drawn on the screen. The user can then use these lines to draw the new map. In the drawing process itself, the user sets the cursor at a position where he wishes to have a node, and enters it by typing a period ('.'). A bell will ring indicating to the user that the system has received the node. To prevent splintering and overlaying of subareas along boundaries, a mechanism is in place to assure that nodes within a preset radius are assumed to be the same node. The system collects nodes until the user has closed the polygon by reentering the first node of the polygon. After the boundary polygon is entered and drawn, the polygon for each subarea is drawn. After entering each subarea, the user is prompted for the name of the subarea and the attribute values pertaining to that subarea.

4.2.2. Edit

The module, EDIT, while not currently implemented, would allow the user to edit a map that he has just drawn or that has been previously stored in the data base. Editing operations would be of two major types: those affecting the graphical image and those affecting the attributes. Graphical editing would allow the user to manipulate the lines that appear on the screen. Thus he would be able to

1) Add a subarea by extending the boundary of the map.
2) Combine two subareas by deleting a line.
3) Split a subarea by adding a line to the map.

Care must be taken to assure that the necessary operations on attributes must be performed in order to maintain the validity of the attribute values associated with the subareas.

Also very useful for inquiry purposes would be a graphical operation which would allow the user to overlay one map with another, creating a union of the two maps. As a result of this operation, a user could inquire about the joint occurrence of attributes maintained on separate maps that describe the same geographical area.

A facility should be provided for users to edit attributes. Characteristics of subareas may change over time requiring the changing of attribute values therefore, the user may also wish to expand the capability of the map by deleting no longer useful attributes and adding new ones. Operations to handle these procedures would require changes in the attribute table as well as changes to the attribute values associated with each subarea.

4.3.4. Inquire

The module, INQUIRE, (Appendix C.7) is designed to allow the user to inquire about the data the map represents. The user has three ways to use this module.

1) He can request a tabular display of the data
associated with the map. This approach would provide data in tabular form for all subareas of the map but would not show the map on the screen.

2) He can request to see all the data for a particular subarea. The map would be selected by the user and displayed on the screen. The user would then select a subarea by placing the cursor on it and the information pertaining to that subarea would be displayed.

3) He can request information directly from the data base. The map selected by the user would be displayed. The user would type a request of the form appropriate to the data base management system used, and the results of the inquiry would be displayed on the screen. (not currently implemented)

4.3.5. Graphic Utilities

A set of graphic utility modules are found in UTIL. These modules, described below, are called from the DRAW, EDIT or INQUIRE modules and operate on graphic images on the screen.

The function, getslope, receives the endpoints of a line and returns its slope. The function, yintercept, calculates the y intercept of the line it receives and returns this value. These functions are shown in Appendix C.8.

The function, bound_box (Appendix C.9), determines whether two boxes defined by two sets of endpoints overlap. This function calls another function, bb (Appendix C.10), which takes a point, x and y, and a box, defined by two points, and returns an indicator based on where the point lies in relation to the box. The reader is referred to the
illustration in the comments section of both functions for more information about what indicators are returned.

The function, intercept (Appendix C.11), determines if and how two lines intercept each other. Intercept checks to assure that the bounding boxes of the lines for intersect before proceeding to find the intercept. For additional information on how the intercept function returns the information it has found, the reader is invited to read the comments at the beginning of the function. In the case of a single intersection not at an endpoint, intercept calls fintercept (Appendix C.12), a recursive procedure which examines smaller and smaller sections of the lines for intersection.

The function for determining whether a point is in a polygon is called inpoly and is found in Appendix C.13. It receives a point and returns a pointer to the subarea of the map containing that point. Inpoly first assures that the point lies within the bounding box surrounding the polygon by using the function bb. It then checks all edges for intersection with the horizontal line which contains the point and is bound by the minimum and maximum x values of the surrounding box. After all intersections are calculated, whether the point lies inside or outside the polygon can be determined depending upon where it lies in relation to the intersection points.
The above describes DRAW-A-MAP, as it currently exists, a basic interactive graphical system for drawing maps. The next section will discuss how this system might be enhanced.
5. Enhancements

DRAW-A-MAP could be considerably expanded to create a production quality cartographic tool, to increase its efficiency in interfacing with the data base and to increase its "friendliness" to users.

As a production cartographic tool, a system such as DRAW-A-MAP, could contain:

1. Facilities to maintain line data, such as a river or a highway, point data, such as an historic site, and polygon within a polygon data, such as a lake within a subarea.

2. Facilities to develop specific graphic symbols with which to label the map.

3. A facility for including text data in the map to label the map, its subareas and its points of interest.

4. Facilities for filling subareas with a pattern expressing a meaning rather than a color.

5. Facilities to define free form lines, as would be found in a line defining a winding river or a complex polygon.

DRAW-A-MAP could be made more "user friendly". It could allow the user to enter maps traced from a real map on a bit pad. It could then allow the user to specify how large the map could be on the screen. Other screen operations, such as scroll, expand and contract, rotate, move, center and adjust size, could be added which would allow the user to operate on his drawing.
Drawing a map would be much easier for the user if he could erase lines and if he were allowed to draw the map first, using lines that could cross each other and then enter the subarea data. These features would require several complex polygon manipulation functions but would be useful.

Increased error prevention procedures could be implemented. Currently the system permits a map to contain areas that are undefined. It is also possible to have subareas defined outside of the boundary of the map as well as subareas which overlay or surround each other.

For more efficient operation the interface with the database could be improved. The user and system could be able to detect and process error conditions returned from the database. It could be possible to selectively retrieve or update maps that exist in the database. It would also be useful to have a display/interpreter (as described by Weller and Williams) [17] that reads directly from the database and generates the map image on the graphical device.

The above enhancements, if implemented, would improve the quality and usefulness of the DRAW-A-MAP system.

6. Conclusion

The DRAW-A-MAP system demonstrates how both graphical and nongraphical data may be stored in a relational database. It provides an interactive system for drawing a map and suggests the many routes one might take in its enhancement.

Section 1 traced the development of techniques for storing graphical data. It began with the period during which languages were developed to handle list structures and data relations. It followed with descriptions of computer cartography applications and projects in which graphical data was stored in data bases.

Section 2 provided a discussion of the processes inherent in graphics and their ramifications for data base design. It specified what assumptions must be relaxed for a data management system to support graphical processes. It explained why the relational data base model is appropriate for computer graphics. The DRAW-A-MAP data base design was presented and explained.

The graphic considerations of DRAW-A-MAP were presented in Section 3. Both the bit map technique and the polygon technique (used in DRAW-A-MAP) for storing graphical data were explained. Their respective uses were discussed. The subject of data structures in computer graphics was presented. Those features a data structure must possess to
support computer graphics were discussed. The data structure used in DRAW-A-MAP was presented and explained.

The actual implementation of DRAW-A-MAP was the topic of Section 4. It was shown how DRAW-A-MAP was made portable between data base systems and graphical devices by the use of separate modules to execute the interface processes. The three primary modules of the system; DRAW, EDIT and INQUIRE were examined. The user's view and an internal view of the system were provided.

Section 5 concluded the description of DRAW-A-MAP with a discussion of the features one might add to convert the system to a production quality cartographic tool and how one might make the system more "user friendly".
Appendix A

Structural Model

* Called by DRAW, EDIT, INQUIRE as required
### Appendix B

**Data Base Design**

#### Relation `mpdir`

<table>
<thead>
<tr>
<th>MPNAME</th>
<th>AUTHOR</th>
<th>ATTRTBL</th>
<th>NOATTR</th>
<th>NOSUBA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monroe</td>
<td>Smith</td>
<td>Mon.atbl</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Ontario</td>
<td>Jones</td>
<td>Ont.atbl</td>
<td>4</td>
<td>12</td>
</tr>
</tbody>
</table>

#### Relation `Mon.atbl`

<table>
<thead>
<tr>
<th>ANAME</th>
<th>ATYPE</th>
<th>ALLOW</th>
<th>AHIGH</th>
</tr>
</thead>
<tbody>
<tr>
<td>population</td>
<td>n</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>area</td>
<td>n</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>income</td>
<td>n</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>party</td>
<td>c</td>
<td>(null)</td>
<td>(null)</td>
</tr>
</tbody>
</table>
Relation Monroe

<table>
<thead>
<tr>
<th>SUBNAME</th>
<th>SCOORD</th>
<th>WCOORD</th>
<th>NNODES</th>
<th>X</th>
<th>Y</th>
<th>COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rochester</td>
<td>Monrc.s.2</td>
<td>Monrc.w.2</td>
<td>4</td>
<td>230</td>
<td>365</td>
<td>green</td>
</tr>
<tr>
<td>Henrietta</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brighton</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Relation Monroe -- continued)

<table>
<thead>
<tr>
<th>BBXMIN</th>
<th>BBYMIN</th>
<th>BBXMAX</th>
<th>BBYMAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>220</td>
<td>315</td>
<td>280</td>
<td>335</td>
</tr>
</tbody>
</table>

(Relation Monroe -- continued)

<table>
<thead>
<tr>
<th>POPULATION</th>
<th>AREA</th>
<th>INCOME</th>
<th>PARTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>10</td>
<td>20</td>
<td>dem</td>
</tr>
</tbody>
</table>

---
### Relation Monro.s.2

<table>
<thead>
<tr>
<th>SCRX</th>
<th>SCRY</th>
<th>SCRSLOPE</th>
<th>SCRYINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>220</td>
<td>250</td>
<td>.9</td>
<td>136</td>
</tr>
<tr>
<td>256</td>
<td>335</td>
<td>-.6</td>
<td>545</td>
</tr>
<tr>
<td>280</td>
<td>370</td>
<td>999.5</td>
<td>9999</td>
</tr>
<tr>
<td>290</td>
<td>315</td>
<td>-.5</td>
<td>478</td>
</tr>
</tbody>
</table>

### Relation Monro.w.2

<table>
<thead>
<tr>
<th>WORLDX</th>
<th>WORLDY</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>26</td>
<td>37</td>
</tr>
<tr>
<td>30</td>
<td>34</td>
</tr>
<tr>
<td>30</td>
<td>23</td>
</tr>
</tbody>
</table>
#define TIME 10

struct map {
    char * mapname; /* pointer to array containing map name */
    int mnoattr; /* no. of attributes */
    char * mattrml; /* name of attribute table */
    struct attrl * mattrbl; /* pointer to the attribute table */
    struct subarea * msp; /* pointer to area containing subarea info */
    int mtotsuba; /* no. of subareas in the map */
    struct edge * mpe; /* pointer to beginning of edge array */
    char * author; /* name of person who entered map */
    int mtotnode; /* no. of nodes in this map */
};

struct subarea {
    char * subarname; /* pointer to array containing subarea name */
    int stotnode; /* no. of node in subarea */
    int sx, sy; /* coordinate point in subarea */
    char * scolor; /* pointer to color */
    char * scoord; /* name of relation containing screen coordinateetes */
    char * wcoord; /* name of relation containing world coordinates */
    int sbbminx, sbbminy, sbbmaxx, sbbmaxy; /* bounding box description */
    struct node * snpn; /* pointer to beginning of node array */
    struct attrl * sattri; /* pointer to attribute values */
    struct subarea * snext; /* pointer to next subarea */
    struct edge * sedge; /* pointer to list of edges */
};

struct node {
    int nx; /* x coordinate */
    int ny; /* y coordinate */
    long nyint; /* y intercept of line */
    float nslope; /* slope of line from this node to the next */
    int nwx; /* world x coordinate */
    int nwy; /* world y coordinate */
    struct node * nnext; /* pointer to next node */
};

struct edge {
    int eberx; /* beginning x coor. */
};
int ebegy; /* beginning y coord. */
int eendy; /* ending x coord. */
int eendx; /* ending y coord. */
long eyint; /* y intercept of edge */
float eslope; /* slope of this edge */
struct edge *enext; /* pointer to next edge */
};

struct attbl {
  char * aname; /* pointer to attribute name */
  char atype; /* indicates type of attribute, numeric or char */
  int alow; /* lower limit of numeric variable */
  int ahig; /* upper limit of numeric variable */
  struct attbl * alnext; /* pointer to next attrib in table */
};

struct attri {
  char * attribute; /* value of attribute */
  struct attri * anext; /* pointer to next attribute value */
};

struct map *mp, *mpx; /* pointers to map structures */
struct subarea *sp, *spx; /* pointers to subarea structures */
struct node *np, *npx; /* pointers to node structures */
struct edge *ep, *epx; /* pointers to edge structures */
struct attbl *atbp, *atbpx; /* pointers to attribute table */
struct attri *atp, *apx; /* pointers to attributes */

char * fname; /* function name */
int i,j; /* general purpose subscripts */
int xinter1,yinter1,xinter2,yinter2; /* points of intersection */
xinter3,yinter3,xinter4,yinter4;
int temp; /* variable for saving values */
int xa1,ya1,xa2,ya2; /* points describing line or box a */
int xb1,yb1,xb2,yb2; /* points describing line or box b */
float m_a,m_b; /* slope of line a or b */
long b_a,b_b; /* y intercept of line a or b */
int lb,ub; /* upper and lower bounds */
char instring[17]; /* string to receive input from screen */
char svsstring[17]; /* string to save input from screen */
char outstring[20];

int curx,cury; /* position of the cursor */
indent1 = " "; /* indentation */
indent2 = " "; /* indentation */
int scan; /* results of scanf */
int c; /* general character pointer */
float wxxm,wxxa,wwym,wwya; /* relationships between screen and world */
int worldx1,wordyr,worlyb,wordyt; /* coordinates describing world */
int windx1 = 1;
int windx2 = 1023;
int windyb = 115;
int windyt = ???; /* coordinates describing window on screen in which map will be drawn */
struct map *curmap[10]; /* array of pointers to maps in pgm. */
int nummap = 0;  /* subscript for curmap array */
char line[200];  /* array for characters received in input */

  e1 = "scan error ";
e2 = "error in allocation of map area ";
e3 = "error in allocation of subarea area ";
e4 = "error in allocation of node area ";
e5 = "error in allocation of edge area ";
e6 = "error in allocation of attribute area ";
e7 = "error in allocation of attribute table name ";
e8 = "error in allocation of attribute table area ";
e9 = "error in allocation of mapname ";
e10 = "error in allocation of author name ";
e11 = "error in allocation of screen coordinate relation name ";
e12 = "error in allocation of world coordinate relation name ";
e13 = "error in allocation of subarea name ";
e14 = "error in allocation of color area ";
e15 = "error in allocation of attribute name ";
e16 = "error in allocation of attribute value area ";
C.2 Mrscmd, mrscmd1, mrscmd2

File - Mrscmd

cp frequest ./mapdb/frequest
cd mapdb
mrs <frequest
mrs <stop
rm frequest
cp fmapres ../fmapres

File - Mrscmd1

cp frequest frenx
cp frequest ./mapdb/frequest
cd mapdb
mrs <frequest
mrs <stop
rm frequest

File Mrscmd2

cp frequest ./mapdb/frequest
cd mapdb
mrs <frequest
mrs <stop
rm frequest
cp fmapres ../fmapres
cp fmap2res ../fmap2res
#define SQUOTE \047
#define TIME 5
#include <stdio.h>
#include "mpnamex.h"
char *charp;
into = into 'fmapres' ";
into2 = into 'fmap2res' ";

int num1,num2,num3; /* used only in scanf stmts */
int num4,num5,num6,num7;
long num9;
long num10;
struct edge *node_edge(); /* func. that creates an edge list
   for whole map */
char *encl();
int nattr; /* counter for attributes */
int nsuba; /* counter for subareas */
int nnodes; /* counter for nodes */
int x,y; /* temp. variables for x and y */
char name [50]; /* array to hold names */
char name2 [50]; /* second array to hold names */
char cname [12]; /* array for map name */
char scname [12]; /* array for screen coord names */
char wname [12]; /* array for world coord relation name */
FILE *fopen(), *rp, *rp2, *wp; /* pointers to files */

/*_readmap__________________________________________*/
struct map *readmap (nnun) /* returns a pointer to a map, S0F or
  0 if error */
/* This function will read a map with up to five nongraphical
attributes from a data base. It interfaces with MRS in the
following manner.
  1. This program will print MRS commands on a file called
     FREQUST and close the file.
  2. It will give control to a file called mrs cmd which
     contains the following:
     cp frequent ../frequest (moves the request file
     to the mapdb directory)
     cd mapdb (changes directories)
     mrs <request; (calls MRS, using request)
     from <stop; (instructs MRS to stop)
  3. MRS executes the commands it has received and
     returns the requested information on the file fmapres.
  4. File, fmapres, is moved to the program's directory.
  5. The program reads the file fmapres for its data.
  */
char *mpnm;

/* name of map to be read */
{
    fnname = 'readmap';
    printf("Reading %s from the data base. \n\n\nSTANDBY", mpnm);

    /* This section requests information from the map directory. */
    /* open request file */
    wp = fopen("request", "w");
    /* allocate a structure for the map */
    if (mp = malloc(sizeof(struct map))) == NULL
    {
        /* allocation error */
        printf("%s %s
", fnname, e2);
        delay(TIME);
        return(0);
    }
    /* save a pointer to the map */
    curmap[nummap++] = mp;
    /* allocate room for the map name */
    if ((mp->mapname = malloc(strlen(mpnm) + 1)) == NULL)
    {
        /* allocation error */
        printf("%s %s
", fnname, e9);
        delay(TIME);
        return(0);
    }
    /* copy the map name in */
    strcpy(mp->mapname, mpnm);
    /* request information from map directory */
    fprintf(wp, "%s select author, attrbi, nattri, nosub from ,pinto;
    fprintf(wp, "mpdir where mpname = "%s\n", mpnm);
    fclose(wp);
    system("mrscmd");

    /* The following section reads the information returned by MRS. */
    /* open the PMAPRES file */
    rp = fopen("mapres", "r");
    /* gets headers */
    fgets (line, 200, rp);
    fgets (line, 200, rp);
    fgets (line, 200, rp);
    /* read information from the map directory */
    if (!scan = fscanf(rp, "%s %s %d %d, name2, name, %num1, %num2)) == EOF)
    {
        /* scan error */
        printf("%s %s
", fnname, e1);
        delay(TIME);
        return(scan);
    }
    fclose(rp);
    /* gets the number of attributes */
    mp->nnoatrr = num1;
/* gets the number of subareas */
mp->mtotsuba = num2;
/* allocate room for the name of the attribute table */
if (!mp->mattrnl = malloc(strlen(name) + 1)) == 0)
{
    /* allocation error */
    printf("%s %s\n", fname, e7);
    delay(TIME);
    return(0);
}
/* copy it in */
strcpy(mp->mattrnl, name);
/* allocate room for name of author */
if (!mp->author = malloc(strlen(name2) + 1)) == 0)
{
    /* allocation error */
    printf("%s %s\n", fname, e7);
    delay(TIME);
    return(0);
}
/* copy it in */
strcpy(mp->author, name2);

/* This section requests, receives and loads the attribute table 
   and subarea information. */
wp = fopen("frequest", w);
/* issues request for attribute table */
fprintf(wp, "%s %s\n", pinto, mp->mattrnl);
/* requests all subarea information */
fprintf(wp, "%s %s\n", pinto2, mp->mapname);
fclose(wp);
system("mrsccmd2");
rp = fopen("fmapres", r");
/* gets headers */
fgets(line, 200, rp);
fgets(line, 200, rp);
fgets(line, 200, rp);
/* This section loads the attribute table. */
for (nattr = 0; nattr < mp->mnattr; nattr++)
{
    /* allocate attribute table entry */
    if (!atbp = malloc(sizeof (struct attrbl))) == 0)
    /* allocation error */
    {
        printf("%s %s\n", fname, e8);
        delay(TIME);
        return(0);
    }
    if (nattr == 0)
    /* put attribute table pointer in map structure */
    mp->mattrtbl = atbp;
else
    /* tack this attribute table entry on list */
    atbnext->atlnext = atbp;
    /* save pointer for next link*/
    atbpx = atbp;
    /* read an attribute table entry */
    if ((scan = fscanf(rp, "%s %s %d %d", name,
            name2,&num1,&num2)) == 0)
        { /* scan error */
          printf ("%s %s\n",fname,e1);
          delay(TIME);
          return(0);
        }
    atbp->alow = num1;
    atbp->ahigh = num2;
    if (*name2 == 'N')
        atbp->atype = 'N';
    else
        atbp->atype = 'C';
    /* allocate room for name of attribute */
    if ((atbp->aname = malloc(strlen(name) +1)) == 0)
        /* allocation error */
        { printf("%s %s",e15,fname);
          delay(TIME);
          return(0);
        }
    /* copy it in */
    strcpy(atbp->aname,name);
}
fclose(rp);
/* end the attribute table list */
atbp->atlnext = NULL;

/* This section loads all subarea information. */
rp2 = fopen("fmap2res .","r");
fgets (line,200,rp2);
fgets (line,200,rp2);
fgets (line,200,rp2);
/* initialize counter for all the nodes in this map */
mp->mtotnode = 0;
for (nsuba = 1; nsuba <= mp->mtotsuba; nsuba++)
    /* loading subareas with map data */
    { /* must allocate room for the subarea */
      if ((sp = malloc(sizeof (struct subarea))) == 0)
          /* allocation error */
          {

        }
printf("%s %s \n", fnname, e3);
delay(TIME);
return(0);
}

if (nsuba == 1)
    /* load 1st subarea pointer into map structure */
    mp->msp = sp;
else
    /* tack on subarea list */
    spx->snext = sp;

    /* save pointer to subarea */
    spx = sp;
    if ((scan = fscanf(rp2, "%s %s %x %d %x %d %x %d %x %d", name, scname, wcname, &num1, &num2, &num3, cname, &num4, &num5, &num6, &num7))
        == EOF
        || scan == 0)
    /* reading subarea data */
    {
        printf("%s %s\n", fnname, e1);
        delay(TIME);
        return(0);
    }

    sp->sx = num2;
    sp->sy = num3;
    sp->sbminx = num4;
    sp->sbminy = num5;
    sp->sbmaxx = num6;
    sp->sbmaxy = num7;
    sp->stnode = num1;
    /* accumulates total nodes in map */
    mp->mtotnode += sp->stnode;
    if (((sp->subname = malloc(strlen(name) + 1)) == 0)
        /* allocate room for subareaname */
        {
            /* allocation error */
            printf("%s %s\n", fnname, e13);
            delay(TIME);
            return(0);
        }

    /* copy it in */
    strcpy(sp->subname, name);
    if (((sp->scolor = malloc(strlen(cname) + 1)) == 0)
        /* allocate room for color name */
        {
            /* allocation error */
            printf("%s %s\n", fnname, e14);
            delay(TIME);
            return(0);
        }

    /* copy name in */
    strcpy(sp->scolor, cname);
    if (((sp->scoord = malloc(strlen(scname) + 1)) == 0)
/* allocate room for screen coordinate relation name */
{
    /* allocation error */
    printf ("%s %s\n", fnname, e11);
    delay(TIME);
    return(0);
}

/* copy it in */
strcpy(sp->scoord, scname);
if (!(sp->wcoord = malloc(strlen(wcname) + 1)) == 0)
    /* allocate room for world coordinate relation name */
{
    /* allocation error */
    printf ("%s %s\n", fnname, e12);
    delay(TIME);
    return(0);
}

/* copy it in */
strcpy(sp->wcoord, wcname);
if (nsuba == 1)
    /* no attributes for boundary area-- skip it */
{
    sp->sattri = NULL;
    continue;
}
for (atbp=mp->mattrtbl; atbp!=NULL; atbp=atbp->atlnext)
    /* get attribute values */
{
    if ((ap = malloc(sizeof(struct attri))) == 0)
        /* allocate an attribute structure */
        {
            /* allocation error */
            printf ("%s %s\n", e6, fnname);
            delay(TIME);
            return(0);
        }
    /* read attribute value */
    if (atbp->atype == 'C')
        /* this attribute is character */
        {
            if (((scan = fscanf(rp2, "%s", name)) == EOF)
                || scan == 0)
                {
                    /* scan error */
                    printf ("%s %s \n\n", fnname, e1);
                    delay(TIME);
                    return(0);
                }
        }
    else
        /* this attribute is numeric */
        {
            if (((scan = fscanf(rp2, "%d", tmem)) == EOF)
                || scan == 0)
                {
                    /* scan error */
                    printf ("%s %s \n\n", fnname, e1);
                    delay(TIME);
                    return(0);
                }
        }
import sys

if scan == 0:
    /* scan error */
    printf("%s %s \n", fnname, e1);
    delay(TIME);
    return(0);

    itoa(temp, name);

    if (atbp == mp->matr tbl)
        /* enter attribute address in subarea structure */
        sp->sattri = ap;
    else
        /* tack it on the list of other attributes */
        apx->anext = ap;

    if ((ap->attribute = malloc(strlen(name) + 1)) == 0)
        /* allocate room for attribute itself */
        /* allocation error */
        printf("%s %s ,e1?, fnname);
        delay(TIME);
        return(0);

    /* copy it in */
    strcpy(ap->attribute, name);
    /* save pointer to most recently created structure */
    apx = ap;

    /* end loop for attributes of this subarea */
    /* complete the list */
    apx->anext = NULL;
    /* save pointer to most recently created subarea structure */

    /* end loop for subareas */
    /* complete subarea list */
    sp->snext = NULL;
    /* close */MAPRES */
    fclose(rp2);

    /* This section retrieves and loads screen and world coordinates */
    for (sp = mp->mat; sp != NULL; sp = sp->snext)
        /* for each subarea retrieve and load screen and world coordinates */
        /* open REQUEST */
        wp = fopen("frequest", "w");
        /* request screen coordinates */
        /* request world coordinates */
        fprintf(wp, "%s select from %s; \n", pntc, sp->scoord);
        fprintf(wp, "%s select from %s; pinto2, sp->wcoord);
        fclose(wp);
system ("mr$cmd2");
    rp = fopen("fmapres", "r");
    rb2 = fopen ("fmap2res", r");
/* get headers */
    f�ets (line, 200, rp);
    f�ets (line, 200, rp);
    f�ets (line, 200, rp);
    f�ets (line, 200, rp);
    f�ets (line, 200, rp);
for (nnodes = 1; nnodes <= sp->stnode; nnodes++)
/* load screen coordinates in node structure */
{
    /* allocate a node structure */
    if ((np = malloc (sizeof (struct node ))) == 0)
    {
        /* allocation error */
        printf("%s %s\n", fname, e4);
        delay(TIME);
        return(0);
    }
/* read the coordinates */
    if ((scan = fscanf(rp, "%d %d %d %d ",
                        &x, &y, &num10, &num9)) == EOF
        || scan == 0)
    {
        /* scan error */
        printf ("%s %s \n", e1, fname);
        delay(TIME);
        return(0);
    }
    if ((scan = fscanf(rp2,"%d %d %d %d ",
                        &num3, &num2)) == EOF
        || scan == 0)
    {
        /* scan error */
        printf ("%s %s \n", e1, fname);
        delay(TIME);
        return(0);
    }
    if (nnodes == 1)
        /* enter node pointer into subarea structure */
        sp->spn = np;
    else
        /* tack new node structure on the node list */
        np->nnext = np;
        np->nx = x;
        np->ny = y;
        np->nslope = num10 / 10;
        np->nint = num9;
        np->nwx = num3;
        np->nwy = num2;
        np = np; }
/* complete the node list */
np->nnext = NULL;
fclose (rp);
fclose (rp2);
/* collect edges for this subarea */
sp->spe = node_edge (sp);
} /* end for loop for subareas */
/* collect all the edges for the map */
mp->mpE = coll_edge(mp);
printf ("\n%5sMap %s available for use", indent2, mp->mapname);
delay (TIME);
return (mp);
/* __deletemap ----------------------------------------------------------*/

int deletemap (infile) /* function to delete map from the data base */
char *infile; /* pointer to name of map to be deleted */
{
  fpname = "deletemap; ";
  printf ("Deleting %s from the data base, %s\n", infile);
  wp = fopen ("frequest", "w"); /* requesting name of attribute table */
  printf (wp, "%s select attrib from mmdir", pinto);
  printf (wp, " where mname = %s\n", enc1 (infile)); /* requesting names of screen and world coordinate relations */
  printf (wp, "%s select scoor, wcoord from %s; %s\n", pinto2, infile);
  fclose (wp);
  system ("rmsemd2");
  rp = fopen ("fmapres", "r"); /* reads headers */
  fgets (line, 200, rp);
  fgets (line, 200, rp);
  fgets (line, 200, rp);
  wp = fopen ("frequest", "w");
  if (scanf (rp, "%s", name) != 0) /* gets headers */
  {
    /* getting name of attribute table and dropping it */
    printf (wp, "drop table %s; %s\n", name);
  }
  fclose (rp);
  rp = fopen ("fmap2res", "r");
  fgets (line, 200, rp);
  fgets (line, 200, rp);
  fgets (line, 200, rp);
  do /* reads and issues a drop command for attribute table relation, */
    /* screen and world coordinate relations */
    while (scanf (rp, "%s", name) != 0)
    {
      if (scanf (rp, "%s", name) == 0)
        /* no more relations */
        break;
    } /*else*/
    printf (wp, "drop table %s; %s\n", name);
    if (scan != EOF);
  /* deletes subarea relation */
  printf (wp, "drop table %s; %s\n", infile);
  /* deletes tuple identifying the map from the map directory */
  printf (wp, "delete from mmdir where mname = %s\n", enc1 (infile));
  fclose (wp);
system ("mrscmd1");
fclose(RP);
printf("%s Map %s removed from the data base.\n", indent2, instring);
delay (TIME);
/* savemap */
savemap(mp)
/* This function writes the map pointed to by mp to the data base. */
struct map * mp;
{
    int tslope; /* holds slope * 10 */
    char * encl(); /* function the encloses a string in single quotes */
    /* open REQUEST */
    printf ("\n\nSaving map in data base\n");
    printf ("\n\nMap name: %s", mp->mapname);
    printf ("\n\n%" "STANDARD", indent2);
    wp = fopen ("request", "w");
    /* put map entry into map directory */
    fprintf (wp,"insert into mpdir:[%s ," ,encl(mp->mapname));
    fprintf (wp,"%s ," ,encl(mp->author));
    fprintf (wp,"%s ," ,encl(mp->mattrnm),mp->mnoattr,mp->mtotsuba);
    /* creates attribute table */
    fprintf (wp,"create table %s(aname char(16) ," ,mp->mattrnm);
    fprintf (wp," atype char(1) , alow numeric, ahigh numeric)\n");
    for (atbp = mp->mattrtbl; atbp != NULL; atbp = atbp->atlnext)
    /* filling attribute table */
    {
        fprintf (wp,"insert into %s:" ,mp->mattrnm);
        fprintf (wp,"%s ," ,encl(atbp->aname));
        enclc (atbp->atype);
        fprintf (wp,"%s ," ,atbp->astring,atbp->alow,atbp->ahigh);
    }
    /* create subarea relation */
    fprintf (wp,"create %s (subname char(12), scoord char(12) ,", mp->mapname);
    fprintf (wp," wcoord char(12), nnodes numeric, x numeric, y numeric, ");
    fprintf (wp," color char(8), bbxmin numeric, bbymin numeric, ");
    fprintf (wp," bbxmax numeric, bbymax numeric \n");
    for (atbp = mp->mattrtbl; atbp != NULL; atbp = atbp->atlnext)
    /* gets names of attributes */
    {
        fprintf (wp,"%s ," ,atbp->aname);
        if (atbp->atype == "N")
            fprintf (wp," numeric ");
        else
            fprintf (wp," char(16) ");
    }
    fprintf (wp,"\n");
    for (sp=mp->msp; sp!=NULL; sp=sp->snex)
    /* write data for each subarea */
    /* */
fprintf (wp, "insert into %s: [%s, %s], mp->mapname, encl(sp->subarname));
fprintf (wp, %s, %s, encl(sp->scoord));
fprintf (wp, %s, %s, encl(sp->wcoord));
fprintf (wp, "%d, %d, %d, %d", sp->sotnode, sp->sx, sp->sy, encl(sp->scol));
fprintf (wp, %s, %s, encl(sp->wcol));
sp->sbbminx, sp->sbbminy);
sp->sbbmaxx, sp->sbbmaxy);

for (ap=sp->sattri, atbp=mp->mattrtbl; ap!=NULL; ap=ap->anext, atbp=atbp->atlnext)
    /* get each attribute * /
    {
        if (atbp->atype == 'N')
            {temp = atoi (ap->attribute);
             fprintf (wp, "", %s, temp);
            }
        else
            fprintf (wp, %s, encl(ap->attribute));
    } /* end for */
fprintf (wp, ");
/* create screen coordinate relation */
fprintf (wp, "create table %s", sp->scoord);
fprintf (wp, "(" scrx numeric, scry numeric, ");
fprintf (wp, "create table %s", sp->wcoord);
fprintf (wp, "(" worldx numeric, worldly numeric);\n\n"));

for (np= sp->spni; np!=NULL; np=np->nnext)
    /* write out nodes */
    {
        /* to screen relation */
        fprintf (wp, "insert into %s: ", sp->scoord);
tslope = 10 * np->nslope;
 fprintf (wp, ["%d, %d, %d, %d", np->nx, np->ny,
tslope, np->nyint]);
 /* to world relation */
 fprintf (wp, "insert into %s: ", sp->wcoord);
 fprintf (wp, ["%d, %d":, np->nxw, np->nyw]);
}
fclose (wp);
system ("mrscmd1");
pri.tf ("%s saved in the data base ,indents, mp->mapname);
delay (TIME);

/* encl-----------------------------------------------------------*/

char * encl(instring)
/* This function takes a string, encloses it in single quotes and
returns it. */
char * instring;
{
    char encstring[50];
    encstring[0] = SQUOTE;
    for (i=1; *instring != '\0'; instring++)
        /\* copy whole string */
        encstring[i++] = *instring;
    /\* append final quote */
    encstring[i++] = SQUOTE;
    /\* complete the string */
    encstring[1] = '\0';
    return (&encstring[0]);
}
/\* _enc1c__________________________________________________________*/
int enc1c(ch)
\/* This function takes a character, encloses it in single quotes and
    returns it. */
char ch;
{
    outstring[0] = SQUOTE;
    \/* copy the character */
    outstring[1] = ch;
    \/* append final quote */
    outstring[2] = SQUOTE;
    \/* complete the string */
    outstring[3] = '\0';
    return;
}
/\* _itola__________________________________________________________*/
itola(n,s)
\/* This function takes an integer and converts it to a string */
int n; \/* integer to be converted */
char s[]; \/* string to be returned */
{
    int c,i,j;
i=0;
do { \/* generate digits in reverse order */
        s[i++] = n % 10 + '0'; \/* get next digit */
    }
    while (n/=10 > 0); \/* delete it */
    /\* complete the string */
s[1] = '\0';
for (i=0; i = strlen(s)-1; i < j; i++, j--)
    /* reverse string in place */
    { c = s[i];
      s[i] = s[j];
      s[j] = c;
    }
}
#define ALPHAENTER 31 /* ascii char. for alpha mode */
#define GRAPHENTER 29 /* ascii char. for graph mode */
#define GINENTER 26 /* ascii char. for graphic input mode */
#define CHANGE 5 /* flag indicating a change in mode */
#define SAME 6 /* flag indicating no change of mode */
#define ESCAPE 27 /* ascii escape character */
#define ERASESCREEN 12 /* ascii form feed character */
#define PELL 7 /* ascii char to ring bell */
#define SETDARKLINE 29 /* ascii char to set next line as invisible */

#define GINMODE -1 /* flag indicating the terminal is in GINMODE */
#define ALPHAMODE -2 /* flag indicating the terminal is in ALPHAMODE */
#define GRAPHMODE -3 /* flag indicating the terminal is in GRAPHMODE */

#include <stdio.h>
#include <mpnamex.h>

int xlow,xhigh,ylow,yhigh; /* used to input numeric values */
static int current_mode = ALPHAMODE; /* current mode */
int low_x_last = 0, /* last low x plotted */
  low_y_last = 0, /* last low y plotted */
  high_x_last = 0, /* last high x plotted */
  high_y_last = 0, /* last high y plotted */
  last_x = 0, /* last x plotted */
  last_y = 0; /* last y plotted */
/*__init__*/
int init()
/* This procedure initializes the screen. */
{
clearscreen();
cursorhome();
anmode();
}

/*__file_to_screen*/
int file_to_screen(char *fn)
int lineno; /* the lineno on which it is to be written */
FILE *fp, *fopen();
int i; /* result of fgets */
char *fgets; /* result of fgets */
/* open file */
fp = fopen(fn, "r");
/* clear the screen */
clearscreen();
for (i=0; i<lineno; i++)
    /* space down to first print line */
    printf ("\n");
while ((fgets = fgets (line, 80, fp)) != NULL)
    /* copy lines out as they are read from the file */
    printf ("%s", indent2, line); 
fclose(fp);
return;
}

/*__placecursor*/
int placecursor(int x, int y)
/* This function places a cursor at the specified x, y coordinate position. */
{
    if (check_mode (GRAPHMODE) == SAME)
        putchar (EETDNRKLINE);
    plot_val (x, y);
    plot_val (x, y);
anmode();

/*__clearscreen*/
int clearscreen()
/* This function clears the terminal. */
{
    putchar (ESCAPE);
    putchar (ERASESCREEN);
    cursorhome();
    sleep (3);
}

/*__readcursor.__________________________________________________________/*/

int readcursor()
/* This function gets the cursor position
 and places it in curx and cury. */
{
    fnname = "readcursor;"
    check_mode (GINMODE);     /* turn on crosshairs */
    c = getchar();            /* read first character */
    /* read high and low order x and y */
    xhigh = getchar();
    xlow = getchar();
    yhigh = getchar();
    ylow = getchar();
    c = getchar();            /* get the 'ret' */
    curx = ((xhigh & 0037) << 5) | (xlow & 0037);
    cury = ((yhigh & 0037) << 5) | (ylow & 0037);
    check_mode (ALPHAMODE);
    return(0);
/* _readstring __________________________________________________________*/

int _readstring (char *string)
/* This function reads a string from the terminal and
places it in the string string */
char *string;
{
    int nc;  /* number of characters read */
    anmode ();
    nc = 16;
    i = 0;
    while ((--nc > 0) && ((c = getchar ()) != '\n'))
        /* read a character and store into string until end
of string is reached or string is full */
        {
            string[i++] = c;
        }
    /* terminate string */
    string[i] = '\0';
    return (i);
}

/* _drawline ___________________________________________________________*/

int _drawline (int x1, int y1, int x2, int y2)  /* draws a line from point 1 to point 2 */
int x1, y1, x2, y2;  /* coordinates */
{
    moveabs (x1, y1);
    drawabs (x2, y2);
}

/* _moveabs ___________________________________________________________*/

int _moveabs (int x, int y)  /* This function moves the cursor to the given point. */
int x, y;  /* The point */
{
    if (check_mode (GRAPHMODE) == SAME)
        putchar (SETDARKLINK);
    plot_val (x, y);
}

/* _drawabs ___________________________________________________________*/

int _drawabs (int x, int y)  /* This function draws a line from the cursor position
to the given point. */
int x, y;  /* the point */
{
    if (check_mode (GRAPHMODE) == CHANGED)
plot_val (last_x, last_y);

plot_val (x, y);

/* _cursorhome_-----------------------------------------------*/

cursorhome()
/* returns cursor to upper left corner of screen */
{
    int x=0, y=779;
    if (check_mode (GRAPHMODE) == SAME)
        putchar (SETDARKLINE);
    plot_val (x, y);
    check_mode (ALPHAMODE);
    return;
}

/* _bell_------------------------------------------------------*/

cbell()
/* sounds the bell */
{
    putchar (BELL);
    return;
}

/* _delay_----------------------------------------------------*/

delay (time)
/* This function causes a delay so the user has a chance to read the
screen. */
int time;
{
    sleep (time);
}

/* _check_mode_-----------------------------------------------*/

static int check_mode (mode)
/* This function checks the mode the terminal is in and resets it
if necessary. */
int mode;  /* Desired mode */
{
    if (current_mode != mode)
    {
        putchar (ESCAPE);
        switch (mode)
        {
        case ALPHAMODE:
            putchar (ALPHAENTER);
            break;
        default:  /* DEBUGGING, print out mode */
            printf ("Mode %2d
", mode);
        }
    }
break;
case GRAPHMODE:
    putchar (GRAPHENTER);
    break;
case GINMODE:
    putchar (GINENTER);
    break:
}
current_mode = mode;
return (CHANGED);
} else
    return(SAME);
static plot_val (x,y)

/* This function sends x and y values to the terminal to move
the cursor to a new position. It will leave a dark line or a visible
line depending on what was set before the routine was called.
Note that the x and y values are broken up into high and low bits
with a code put on each value to tell the terminal which value
is being sent. */

int x,y; /* point to be plotted */
{
  if (x == last_x) /* x value is the same as the last */
  {
    xlow = low_x_last;
    xhigh = high_x_last;
  }
  else
  {
    xlow = (x & 0037) | 0100;
    xhigh = ((x >> 5) & 0037) | 0040;
  }
  if (y == last_y)
  {
    ylow = low_y_last;
    yhigh = high_y_last;
  }
  else
  {
    ylow = (y & 0037) | 0140;
    yhigh = ((y >> 5) & 0037) | 0040;
  }
  /* put out characters */
  putchar (yhigh);
  putchar (ylow);
  putchar (xhigh);
  putchar (xlow);
  /* save current values */
  high_x_last = xhigh;
  low_x_last = xlow;
  high_y_last = yhigh;
  low_y_last = ylow;
  /* save current point */
  last_x = x;
  last_y = y;

  /*__showline__________________________________________________________*/

int showline(x1,y1,x2,y2)

/* This function controls the drawing of a line after having
clipped it. */

int x1,y1,x2,y2; /* endpoints of the line */
{
    if (clip (sx1, sy1, sx2, sy2) == 0)
        /* line is visible */
        drawline (x1, y1, x2, y2);
}

/* _clip_ */

int clip(x1, y1, x2, y2)
    /* This function will clip the lines sent to drawline to restrict
    them to the window in which they are to be written. */
    int *x1, *y1, *x2, *y2;
    /* These are pointers to endpoints
    of a line. */

    int c, x, y;
    int cl, c2;
    cl = code (*x1, *y1);
    c2 = code (*x2, *y2);
    while ((cl > 0) || (c2 > 0))
        /* line is not entirely on screen */
        { if (cl == c2)
            /* line is entirely off screen */
            return (-1);
            c = cl;
        if (c == 0)
            c = c2;
        if (c & 0001)
            /* crosses left edge */
            { x = windx1;
                y = *y1 + (*y2 - *y1) * (windx1 - *x1) / (*x2 - *x1);
            }
        if (c & 0002)
            /* crosses right edge */
            { x = windxr;
                y = *y1 + (*y2 - *y1) * (windxr - *x1) / (*x2 - *x1);
            }
        if (c & 0004)
            /* crosses bottom edge */
            { x = *x1 + (*x2 - *x1) * (windyb - *y1) / (*y2 - *y1);
                y = windyb;
            }
        if (c & 0017)
            /* crosses top edge */
            { x = *x1 + (*x2 - *x1) * (windyt - *y1) / (*y2 - *y1);
                y = windyt;
            }
        if (c == -1)
{  
    *x1 = x;
    *y1 = y;
    c1 = code (*x1, *y1);
}
else
{
    *x2 = x;
    *y2 = y;
    c2 = code (*x2, *y2);
}  
/* end while loop */
return(0);
/*_code______________________________*/

int code (x,y)
/* This function returns an integer c which denotes how the
   point falls in relation to the screen.*/

    1001   1000   1010

    0001   0000   0010

screen

    0101   0100   0110

/*
{
    c =0;
    if (x < windxl)
        c = c+1;
    else if (x > windxr)
        c = c + 2;
    if (y < windyl)
        c = c + 4;
    else if (y > windyt)
        c = c + 8;
    return(c);
}
/*_skiplines________________________________________*/

skiplines(x)
int x;    /* no. of lines to skip */
/* This function skips x number of lines.*/
{
    int i;
    anmode();
    for (i=0; i<x; i++)
        printf ("\n");
}
/*_anmode__________________________________________*/

int anmode()
/* This function puts the terminal in alpha/numeric mode.*/
{
    check_mode (ALPHAMODE);
}
/*_finish__________________________________________*/
int finish ()
/* This function resets the terminal. */
{
    clearsreen ();
cursorhome ();
anmode ();
}
/*_outline---------------------------------------------*/
int outline ()
/* This function outlines the section of the screen on which
the map will be drawn */
{
    check_mode (GRAPHMODE);
drawline (windxl,windyb,windxr,windyb);
drawline (windxr,windyb,windxr,windyt);
drawline (windxr,windyt,windxl,windyt);
drawline (windxl,windyt,windxl,windyb);
}
#include <stdio.h>
#include <Pixmap.h>

main()
{
    int b; /* result of bounding box test */
    int stopflag; /* flag telling us to exit the program */
    init(); draw(), edit(), inquire(), delete(); /* functions */
    stopflag = 0;
    /* initializes the screen */
    init();
    /* puts up the intro screen */
    file_to_screen("intro1", 0);
    /* getting the carriage return (cr) */
    readstring (cstring);
    do
    {
        /* puts up selection screen */
        file_to_screen("intro2", 5);
        readcursor();
        if (((b = bb (curx,cury,0,437,1023,900)) == 0) || (b == 9))
        {
            draw();
        }
        else if (((b = bb (curx,cury,0,356,1023,436)) == 0) || (b == 9))
        {
            delete();
        }
        else if (((b = bb (curx,cury,0,267,1023,355)) == 0) || (b == 9))
        {
            edit();
        }
        else if (((b = bb (curx,cury,0,174,1023,266)) == 0) || (b == 9))
        {
            inquire();
        }
        else
        {
            stopflag = 1;
            clearscreen();
            printf ("\n\n\nDRAW-A-MAP completed", indent2);
            delay(5);
            /* reset terminal */
            finish();
        }
    }
    while (stopflag == 0);
    return;
77

} /*_delete


delete()
{
    fnname = "delete; ";
clearscreen();
printf("DELETE\n");
printf("\nWhat map do you wish to remove from the data base? ");
/* delete the specified map from the data base */
deleteMap(instring);
return;
}
} /*_edit

edit()
{
    fnname = 'edit';
clearscreen();
printf("\n\n%sEdit module entered. ", indent2);
printf("\n%sNot currently implemented.\n", indent2, indent2, indent2);
printf("\n%sReturning to main program--", indent2, indent2, indent2, indent2);
delay(10);"
#define TIME 5
#define US '\137' /* underscore character */
#define MAXNODES 50
#define NEAR 50
#include <stdio.h>
#include <pmname.h>
/* functions used in entering the map */

int getmpname(), draw(), savemap(), gattractbl(), drawmap(),
drawpoly(), getworldc(), getbbox(),
getsubname(), getcolor(), swcoord(),
build(), getattribute(), isnum();

long yintercept();
float getslope();

struct subarea *drawsuba(), *drawprev(), *drawboundary();
char *itoa();

struct edge *node_edge(); *coll_edge();

int startx, starty; /* coord. values used in entering
subarea nodes */
/* temporary storage for all nodes */
int nnodesx [MAXNODES];
int nnodesy [MAXNODES];
int ncount; /* counter for nnodesx and nnodesy */
int nnodes; /* counter for nodes */
ret = ",", ";
per = ",", ";
/* draw */
/* function controlling the drawing of a map */

int draw() /* function controlling the drawing of a map */
{
    fnname = "draw; ";
clearscreen();
    printf("You are now ready to draw the map.\n");
    /* allocate map structure */
    if ((mp = malloc(sizeof(struct map))) == NULL)
        /* allocation error */
    {
        printf("%s\n", e2, fnname);
delay (TIME);
    return(0);
}
/* save map pointer in curmap */
curmap[nummap++] = mp;

/* initialize counters */
mp->mtotsuba = 0;
mp->mctcnode = 0;

/* get the author's name */
if (getauthor(mp) == 0)
    /* error occurred */
    return(0);

/* get the map name */
if (getmpname(mp) == 0)
    /* error occurred */
    return(0);

/* get the attribute table */
if (attrtbl(mp) == 0)
    /* error occurred */
    return(0);

/* draw the map */
if (drawmap(mp) == 0)
    /* error occurred */
    return(0);

clearscreen();
printf ("\n\nDo you wish to save this map?", indent1);
readstr(instring);
if (* instring == 'y')
    savemap(mp);
return(1);

/* _getmpname ________________________________ */

int getmpname(mp)
/* gets the name of the map and puts it in the map structure */

struct map *mp;
{
    fname = 'getmpname';
    printf("\n\nPlease enter the name of the map you are creating.",
        indent2);
    readstr(instring);
    /* allocate area for map name */
    if ((mp->mapname = malloc(strlen(instring) +1)) == 0)
        /* allocation error */
    {
        printf("%s\n",e9,fname);
        delay (TIME);
        return(0);
    }
    /* copies name of map it allocated area */
    strcpy(mp->mapname,instring);
    return(1);
}
/**_getauthor__________________________________________________________*/

getauthor(mp)
/* This function enters the name of the author into the
 map. */

struct map *mp;
{
    printf("\n\n%5sPlease enter your name. ", indent2);
    if (readstring (instring) == 0)
        /* no name entered */
        {
            mp->author = 'none';
            return(1);
        }
    /* allocates area for author name */
    if (mp->author = malloc(strlen(instring) +1)) == 0)
        /* allocation error */
        {
            printf ("%s \s", e10, fname);
            delay (TIME);
            return(0);
        }
    /* copies name of author it allocated area */
    strcpy(mp->author,instring);
    return(1);
}
int gattrtbl(mp)
/* This function prompts for the attributes, their types and
their ranges if numeric. It stores the items in the attribute
table */

struct map *mp;
{
    fnname = gattrtbl; ";
clearscreen();
printf("\n\%Enter the attributes this map will have, each \",indent2);
printf ("\n\%attribute followed by a %. Each map must have at least \",ret);
printf ("\n\%attribute. When the list is complete \"");
printf ("\n\%enter another %?\n\%\",ret,indent2);
/* set attribute count to zero */
mp->mnoattr = 0;
readstring (instring);
while (*instring != "\0")
{
    printf ("\%",indent2);
    /* count the attributes */
    mp->mnoattr++;
    /* allocate attribute table structure */
    if ((atbp = malloc(sizeof (struct attbl))) == 0)
        /* allocation error */
        {
            printf ("\%s",e8,fnname);
delay (TIME);
            return(0);
        }
    if (mp->mnoattr == 1)
        /* enter beginning of attr. table in map */
    mp->mattrtbl = atbp;
    else /* link this one to the previous one */
        atbp->atlnext = atbp;
/* allocate room for storage of attribute name */
    if ((atbp->aname = malloc(strlen(instring) + 1)) == 0)
        /* allocation error */
        {
            printf ("\%s",e15,fnname);
delay (TIME);
            return(0);
        }
    strcpy (atbp->aname,instring);
/* read the next attribute */
readstring(instring);
/* save pointer to current attribute */
atbp = atbp;
/* attribute table list is completed */
atbp->atlnext = NULL;
clearscreen();
printf("Enter the following information for ");
printf(" the specified attributes.

");
atbp = mp->matrtbl;
while (atbp != NULL)
    /* for each attribute entered, get its characteristics */
    {
        /* print attribute name */
        printf("%s", indent2, atbp->aname);
        /* print attribute type */
        printf("%sData type ('n' for numeric or 'c' for character): ", indent1);
        readstring(instring);
        if (*instring == 'n')
            /* this attribute is numeric */
            {
                atbp->atype = 'N';
                /* set range */
                printf("%sEnter lowest integer value permitted: ", indent1);
                readstring(instring);
                atbp->allow = atol(instring);
                printf("%sEnter highest integer value permitted: ", indent1);
                readstring(instring);
                atbp->ahigh = atol(instring);
            }
        else /* this attribute is a character type */
            {
                atbp->atype = 'C';
                atbp->allow = 0;
                atbp->ahigh = 0;
            }
    }
    /* skip 3 lines */
    skiplines (3);
atbp = atbp->atlnext;
}
if (.mp->matrtnm = malloc(10)) == 0)
    {
        printf("%s %s
", e7, fnname);
        delay (TIME);
        return (0);
    }
/* create a name for the attribute table */
build (.mp->matrtnm, 'a', mp->mapname, 0);
return(1);
/* drawmap */

int drawmap(mp)
/* This function controls the actual drawing of the map. */

struct map *mp;

{
    fnname = "drawmap";
clearscreen();
skiplines(5);
printf(",%Do you wish to draw a new map",16,indent2);
/* skip 2 lines */

printf(",%Do you wish to draw a previously drawn map? ");
/* skip 2 lines */

readstring(instring);
if (*instring == 'y')
    /* first subarea of the map will be the boundary of a previously 
       drawn and stored map */

    [ mp->msp = drawprev();
    if (getworldc() == 0)
        /* get world coordinates, if error return 0 */
        return(0);
    /* get new world coordinates */
    windtoworld(mp->msp->spn);
    ]
else
    [ if (getworldc() == 0)
        /* get world coordinates, if error return 0 */
        return(0);
    /* first subarea will represent boundary */
    mp->msp = drawboundary (mp);
    ]
/* prepare to get subareas */
spx = mp->msp;
file_toscreen("draw2",2);
/* read the 'ret' */
readstring (instring);
do
    /* draw and load the subareas */

    [ /* get next subarea */
        spx->snext = drawsuba(mp);
        placecursor (0,00);
        printf("%Do you have more subareas to enter? ",indent2);
        readstring(instring);
        if (*instring == 'y')
            { spx = spx->snext; 
            }
    ]
clearscreen();

while (*instring == 'y');
/* collect the edges of the map */
/* subareas are complete */
mp->mpe = coll_edge (mp);
return(1);
struct subarea * drawprev()
/* draws previous map boundary and returns a pointer to the boundary */

{
    clearscreen();
    printf("Enter the name of the map to be \", indent2);
    printf ("used in drawing this one. ");
    readstring(instring);
    /* get the map here */
    readmap(instring);
    clearscreen();
    outline();
    /* draw the boundary for this map */
    drawpoly(mp->msp->spn);
    /* disconnect rest of previous map */
    mp->msp->spn->snext = NULL;
    for (ncount=0, np=mp->msp->spn; np!=NULL; np=np->nnext, ncount++)
        /* load nodes into array for checking later */
        {
            nnodesx[ncount] = np->nx;
            nnodesy[ncount] = np->ny;
    }
    return(mp->msp);
}
/* drawpoly */

int drawpoly(np)
/* This function draws the polygon represented by the node list. */
struct node *np;

{
    int xfirst,yfirst,xprev,yprev;
    for (i=0; np != NULL; i++, np=np->nnext)
        /* drawing each line represented in the node structure */
        {
            if (i == 0)
                {
                    xfirst = np->nx;
                    yfirst = np->ny;
                }
            else
                showline(xprev,yprev,np->nx,np->ny);
            xprev = np->nx;
            yprev = np->ny;
        }
    drawline(xprev,yprev,xfirst,yfirst);
    return (i);
}
/* __getworldc_-------------------------------------------------------------*/

int getworldc()
/* This function gets the ranges of the world coordinates 
and controls the screen to world transformation. */
{
    fnname = getworldc; ";
clearscreen();
outline(); /* These coordinates are used to establish the relationship 
to the world coordinate system. */
placecursor(0,0);
printf ("%sPlease enter the world coordinates ",indent2);
printf (" of the lower left \n");
printf (" of the lower left \n");
printf (" of the upper right \n");
printf (" of the upper right \n");
/* read these coordinates */
do
    if ((scan = scanf ("%d,%d", &worldx1, &worldy1)) == EOF)
    {
        /* scan error */
        printf ("%s %s", e1, fnname);
delay (TIME);
return(0);
    }
while (scan == 0);
printf ("%sPlease enter the world coordinates ",indent2);
printf (" of the lower left \n");
printf (" of the lower left \n");
printf (" of the upper right \n");
printf (" of the upper right \n");
do
    if ((scan = scanf ("%d,%d", &worldx2, &worldy2)) == EOF)
    {
        /* scan error */
        printf ("%s %s", e1, fnname);
delay (TIME);
return(0);
    }
while (scan == 0);
/* get the 'ret' */
readstring far instring;
/* set screen to world relationships */
setwrel1();
return(1);
}/* __setwrel1_-------------------------------------------------------------*/

int setwrel1()
/* This function sets the relationship between the screen 
coordinates and the given world coordinates. */
{ 
    /* holds differences between world and window values */
    float worlddiff, winndiff;
    worlddiff = worldxr - worldxl;
    winndiff = windxr - windxl;
    wwxm = worlddiff / winndiff;
    wwxr = worldxl - worldx * wwxm;
    worlddiff = worldyr - worldyb;
    winndiff = windyr - windyb;
    wwyf = worlddiff / winndiff;
    wwyb = worldyb - windyb * wwyf;
    return;
}

/*__windowworld_______________________________*/

windowworld(np)
/* This function maps screen coordinates to world coordinates. */
struct node *np;      /* pointer to a node */
{
    np->nx = (wwxm * np->nx) + wwxr + .5;
    np->ny = (wwym * np->ny) + wwyb + .5;
    return;
}
/* __drawboundary__________________________________________ */

struct subarea * drawboundary(mp)
struct map *mp;
/* This function controls the drawing of the boundary by the user. */
{
    char found; /* flag to indicate that a match has been found */
    fname = "drawboundary; ";
    file_to_screen("draw1",0);
    /* read the 'ret' */
    readstring(instring);
    clearscren();
    outline();
    readcursor();
    bell();
    ncount = 0;
    nnodesx[ncount] = startx = curx;
    nnodesy[ncount++] = starty = cury;
    /* counter for nodes */
    nnodes = 0;
    do
    /* This loop controls the entry of nodes of a polygon
and continues until the polygon is closed. For each
point that is entered, a list of previous nodes are checked.
If any node is within NEAR distance the nodes are considered
to be equal. This prevents splintering and overlaying of
adjacent polygons. */
    {
        /* initialize flag */
        found = 'N';
        for (j=0;j < ncount; j++)
        {
            /* search for near nodes */
            if ( (curx > nnodesx[j]) - NEAR ) $$$
                (curx < nnodesx [j] + NEAR) $$$
                (cury > nnodesy [j]) - NEAR ) $$$
                (cury < nnodesy [j] + NEAR)
                /* if a new node or a node within NEAR of it is in
the node list the node is made equal to that node
otherwise it is added to the list */
            {
                curx = nnodesx[j];
                cury = nnodesy[j];
                found = 'Y';
                break;
            } /* end for loop */
            if ( found == 'N' )
                /* node was not found */
                /* add new node to lists */
nnodesx[ncount] = curx;
nodesy[ncount] = cury;
ncount++;
}
/* allocate a node for this point */
if ((np = malloc (sizeof(struct node))) == 0) {
  /* allocation error */
  printf ("%s%s",e4,fname);
delay (TIME);
return(0);
}
np->nx = curx;
np->ny = cury;
/* make transformation to world coordinates */
windtoworld(np);
/* increment node counter */
nnodes++;
if (nnodes == 1) {
  /* get a subarea for it */
  if ((sp = malloc (sizeof(struct subarea))) == 0) {
    /* allocation error */
    printf ("%s%s",e3,fname);
delay (TIME);
return(0);
  }
  /* enter first node addr. in subarea */
  sp->spn = np;
} else
  /* tack node on the end of the node list */
{
  npx->nnext = np;
  /* calculate slope */
  npx->nslope = getslope(npx->nx, npx->ny, curx,cury);
  /* calculate y intercept */
  npx->nyint = yintercept(npx->nx, npx->ny, npx->nslope);
  /* get world coordinates */
  windtoworld(np);
  /* draw the line to it */
  showline (npx->nx,npx->ny,curx,cury);
}
/* get next node */
readcursor();
/* ring the bell */
bell();
/* save last address */
npx = np;
/* do the above until the starting position is reached */
while (((curx > startx - NEAR) && (curx < startx + NEAR))}
& (cury > starty - NEAR) & (cury < starty + NEAR)) == 0); /* complete the list */
np->nnext = NULL; /* complete the polygon */
be() ;
showline(np->nx, np->ny, startx, starty); /* calculate the slope */
np->nslope = getslope(np->nx, np->ny, startx, starty); /* calculate y intercept */
np->nyint = yintercept (np->nx, np->ny, np->nslope); /* get world coordinates */
windtoworld(sp->spn); sp->scolor = "none" ;
sp->snext = NULL ;
sp->subarname = 'bound" ;
sp->stotnode = nnodes ;
getbox(sp) ;
sp->sx = NULL ;
sp->sy = NULL ;
swcoord(mp, sp) ;
sp->spe = node_edge(sp) ;
sp->satrr1 = NULL ; /* accumulate total nodes in map */
mp->mtotnode += nnodes ; /* accumulate total subareas */
mp->mtotsuba ++ ;
clearscreen () ;
return(sp) ;
/* _drawsuba_ -------------------------------------------------------------*/

struct subarea * drawsuba(mp)
struct map *mp;
/* This function controls the drawing of the subareas by the user. */
{
    char found; /* flag to indicate that a node match has been found */
    ffname = "drawsuba; ";
    clearscreen();
    outline();
    for (sp = mp->msp; sp != NULL; sp = sp->snext)
        drawpoly (sp->spn);
    anmode();
    readcursor();
    bell();
    found = 'N';
    for (j=0; j < ncount; j++)
        /* search for near nodes */
        if ((curx > nnodesx[j] - NEAR) &&
            (curx < nnodesx[j] + NEAR) &&
            (cury > nnodesy[j] - NEAR) &&
            (cury < nnodesy[j] + NEAR))
        /* if a new node or a node within NEAR of it is in 
        the node list the node is made to equal that node 
        otherwise it is added to the list */
        {
            curx = nnodesx[j];
            curry = nnodesy[j];
            found = "Y";
            break;
        }
    if (found == 'N')
        /* node was not found add it to the list */
        {
            nnodesx[ncount] = curx;
            nnodesy[ncount] = curry;
            ncount++;
        }
    startx = curx;
    starty = curry;
    n nodes=0;
    do
    /* This loop controls the entry of nodes of a polygon 
    and continues until the polygon is closed. For each 
    point that is entered, a list of previous nodes are checked. 
    If any node is within NEAR distance the nodes are considered 
    to be equal. This prevents splintering and overlaying of 
    adjacent polygons. */
        { 
            found = 'N';
            for (j=0; j < ncount; j++)
/* search for near nodes */
if ((curx > nnodesx[i]) - NEAR) &&
    (curx < nnodesx[i] + NEAR) &&
    (cury > nnodesy[i] - NEAR) &&
    (cury < nnodesy[i] + NEAR))
    /* if a new node or a node within NEAR of it is in
       the node list the node is made to equal that node
       otherwise it is added to the list */
{
    curx = nnodesx[i];
    cury = nnodesy[i];
    found = 'Y';
    break;
}
if (found == 'N')
    /* node was not found add it to the list */
{
    nnodesx [ncount] = curx;
    nnodesy [ncount] = cury;
    ncount++;
}
/* allocate a node structure */
if ((np = malloc (sizeof (struct node))) == NULL)
    /* allocation error */
{
    printf ("%s\n", e4, fname);
    delay (TIME);
    return (0);
}
np->nx = curx;
np->ny = cury;
/* calculate world coordinates */
windowworld(np);
/* increment node counter */
nodes++;
if (nodes == 1)
{
    /* allocate subarea space */
    if ((sp = malloc (sizeof (struct subarea))) == NULL)
        /* allocation error */
    {
        printf ("%s\n", e3, fname);
        delay (TIME);
        return (0);
    }
    /* enter node list in subarea */
    sp->spn = np;
    else
    {
        /* tack node on the end of the node list */
        npx->nnext = np;
    }
/* calculate slope */
np->nslope = getslope(np->nx, np->ny, curx, cury);
/* calculate y intercept */
np->nyint = yintercept(np->nx, np->ny, np->nslope);
/* get world coordinates */
windowworld(no);
/* draw the line to it */
showline(np->nx, np->ny, curx, cury);

} /* get next node */
readcursor();
/* ring the bell */
bell();
/* save last address */
npx = np;

while (((curx >= startx - NEAR) && (curx < startx + NEAR) 
    && (cury >= starty - NEAR) && (cury < starty + NEAR)) == 0); /* do the above until the starting position is reached */
/* close the polygon */
showline(np->nx, np->ny, startx, starty);
bell();
/* get the last slope */
np->nslope = getslope(np->nx, np->ny, startx, starty);
/* get the last y intercept */
np->nyint = yintercept(np->nx, np->ny, np->nslope);
/* indicate the end of the list */
np->nnext = NULL;
/* temporarily end the subarea list */
sp->snext = NULL;
if (getsubname(sp) == 0) /* error occurred */
    return(0);
/* save total number of nodes */
sp->stotnodes = nnodes;
/* get bounding box parameters */
setbox(sp);
/* color the polygon and save the color */
getcode(sp);
/* create names for the screen and world coordinate relations */
scoord(mp, sp);
/* create list of edges for use in manipulating the whole map */
sp->sps = node_end(sp);
/* accumulate total nodes for map */
mp->mtotnodes += nnodes;
/* accumulate subarea total for map */
mp->mtotsuba++;
clrscreen();
/* get the attributes for this subarea and store them */
if (!etattrib(sp, mp) == 0) /* error occurred */
    return(0);
return(sp);
/* getbox */

int getbox(sp)
/* Determines the parameters of the box that surrounds this subarea */

struct subarea * sp;
{
    npx = sp->spn;
    /* set minimum and maximum to be the first node */
    sp->sbbminx = npx->nx;
    sp->sbbminy = npx->ny;
    sp->sbbmaxx = npx->nx;
    sp->sbbmaxy = npx->ny;
    npx = npx->nnext;
    for (; npx != NULL; npx = npx->nnext)
        /* check all other nodes */
        { if (npx->nx < sp->sbbminx)
            sp->sbbminx = npx->nx;
        else if (npx->nx > sp->sbbmaxx)
            sp->sbbmaxx = npx->nx;
        if (npx->ny < sp->sbbminy)
            sp->sbbminy = npx->ny;
        else if (npx->ny > sp->sbbmaxy)
            sp->sbbmaxy = npx->ny;
        }
    return(1);
}

/* getsubname */

int getsubname(sp)
/* Gets the name of the map and puts it in the map structure */

struct subarea * sp;
{
    fnname = getsubname; "
    placecursor (0,98);
    printf("%s Please enter the name of the subarea you are creating. ",
            indent2);
    readstring(instring);
    /* allocate room for the subarea name */
    if ((sp->subname = malloc (strlen(instring) +1)) == 0)
        /* allocation error */
        { printf("%s%s\n",e13,fnname);
            return(0);
        }
    /* transfer name */
    strcpy(sp->subname,instring);
    return(1);

/* getcolor---------------------------------------------------------------*/

getcolor(sp)
/* This function prompts for the color of the subarea, enters it in
the subarea structure and colors the subarea. */

struct subarea *sp;
{
    printf("%sPlease enter color.(0=black, 1=blue, 2=green or 4=red) ", indent2);
    readstring(instring);
    /* save color in subarea structure */
    if (*instring == '0')
        sp->sColor = "black";
    else if (*instring == '1')
        sp->sColor = "blue";
    else if (*instring == '2')
        sp->sColor = "green";
    else if (*instring == '4')
        sp->sColor = "red";
    else
        sp->sColor = "white";
    printf("%sPlace cursor in the subarea and enter %s. ", indent2, per);
    readcursor();
    /* save a representative point in the subarea */
    sp->sx = curx;
    sp->sy = cury;
}
```c
/* __swcoord__------------------------------------------------------------*/

int swcoord (mp, sp)
/* Sets up names for world and screen relations. */
struct map *mp;
struct subarea *sp;
{
    static int subno; /* counter of subareas */
    char **sname = swcoord; /*
    /* allocate room for screen and world coordinate names and copy
    the names to those places */
    if ((sp->scoord = malloc (10))) == 0)
    /* allocation error */
    {
        printf("%s\n", e11, sname);
        delay (TIME);
        return(0);
    }
    /* create a name for the screen relation */
    build (sp->scoord, 's', mp->mapname, subno);
    if ((sp->wcoord = malloc(10))) == 0)
    {
        printf("%s\n", 12, sname);
        delay (TIME);
        return(0);
    }
    /* create a name for the world relation */
    build(sp->wcoord, 'w', mp->mapname, subno);
    /* increment for next subarea */
    subno++;
    return(0);
}

/* __build__--------------------------------------------------------------*/
int build(dest, type, mname, subno)
/* This function builds names for the world, screen and attribute relations. */
char dest[]; /* world or screen */
int subno; /* counter for label */
char mname[]; /* name of map */
{
    char ssubno [2]; /* counter in a string */
    j=i=0;
    /* ret first 5 letters from the map name */
    while ((i<5) && (mname[i] != '\0'))
    {
        dest[j++] = mname[i++];
    }
    /* insert a */
    dest[j++] = US;
```
/\ insert a 'w', 's' or 'a' */
dest[j++] = type;
/* insert a */
dest[j++] = US;
/* change subno to alpha */
itoa (subno, ssubno);
for (i=0; ssubno[i] != '\0'; )
{
    /* add subno to name */
    dest[j++] = ssubno[i++];
}
dest[j] = '\0';
return;
}
/** attrib**

```c
int getattrib(sp, mp)
/* Gets the attributes for subarea and stores in sattribute structure */
struct subarea *sp;
struct map *mp;
{
    int anum, first;
    fname = "getattrib: ";
clearscreen();
printf("Enter attribute values followed by a ", indent2, ret);
first = 1;
for (atbp->map->mapattrb; atbp != NULL; atbp = atbp->atbnext)
    /\ get a value for each attribute in the table */
    {
        printf("\\n%\%s", indent2, atbp->aname);
        if (atbp->atype == 'N')
            printf (" (numeric) ");
        else
            printf (" (character) ");
        readstring(instring);
        /* allocate an attribute structure for it */
        if ((ap = malloc(sizeof (struct attri))) == 0)
            /* allocation error */
            {
                printf ("%s\n", e17, fname);
                delay (TIME);
                return(0);
            }
        if (first == 1)
            /* connect this attribute structure to the subarea */
            first = 0;
        sp->sattri = ap;
    }
    else
        /* tack it on to end of attribute list */
        apx->anext = ap;
        /\ allocate room for the attribute value */
        if ((ap->attribute = malloc (strlen (instring) +1)) == 0)
            /* allocation error */
            {
                printf ("%s\n", e6, fname);
                delay (TIME);
                return(0);
            }
        if (atbp->atype == 'N')
            /* the attribute is numeric, you need to check its bounds */
            {
                if (isnum(instring) == 0)
                    /* user did not enter numeric data */
                {
```
printf ("\nExpecting numeric attribute--skipping");
ap->attribute = "0";
apx = ap;
continue;
} /* change it to numeric */
anum = atoi (instring);
if ((num < atbp->alow) || (num > atbp->ahigh))
    /* it is out of range */
    {
        printf ("\nOut of range-- skipping");
ap->attribute = "0";
apx = ap;
continue;
} /* copy attribute into attribute structure */
if ((ap->attribute = malloc (strlen (instring) +1)) == NULL)
    /* allocation error */
    {
        printf ("%s\n",errno,fname);
delay (TIME);
return(0);
    }
strcpy (ap->attribute,instring);
/* save pointer to last attribute */
apx = ap;
} /* attribute list is completed */
ap->anext = NULL;
return(1);
} /* isnum */

int isnum(instring)
/* This function determines if instring contains a number. */
char *instring;
{
    for (i*=instring!='\0'; instring++)
        /* checking each character */
        if (((*instring < '0') || (*instring > '9')))
        /* found non numeric character */
            return(0);
    return(1);
}
/* node_edge */

struct edge * node_edge(sp) /* returns pointer to an edge */
struct subarea * sp; /* sp is pointer to a subarea */
{
    struct edge *sepx; /* pointer to an edge */
    fname = node_edge; /* */
    for (np=sp->spn;np!=NULL;)
    {
        /* allocate edge structure */
        if ((ep = malloc (sizeof (struct edge))) == NULL)
        {
            printf ("%s %s\n",.fname);
            return(0);
        }
        if (np == sp->spn)
            /* save beginning of edge list */
            sepx = ep;
        else
            /* link to the list */
            epx->enext = ep;
        /* save edge pointer for next link */
        epx = ep;
        ep->ebegx = np->nx;
        ep->ebegy = np->ny;
        ep->eslope = np->nslope;
        ep->eyint = np->nyint;
        if (np->nnext == NULL)
        {
            np = np->nnext;
            ep->eendx = np->nx;
            ep->eendy = np->ny;
            ep->enext = NULL;
        }
        else
        {
            np = np->nnext;
            ep->eendx = np->nx;
            ep->eendy = np->ny;
        }
    }
    return(sepx);
}

/* coll_edge */

struct edge * coll_edge(mp) /* collects all edges for a map, */
places them in a list and returns a
pointer to the edge list */
struct map *mp;        /* pointer to a map */
{
    struct edge *sepx, *epxl;     /* pointer to an edge */
    ffname = "coll_edge; 
    i=0;
    clearscreen();
    for (sp= mp->msp; sp!= NULL; sp=sp->snext)
    {
        for (ep= sp->spe; ep!= NULL; ep=ep->enext)
        {
            if ((epx = malloc ( sizeof (struct edge ))) == 0)
            {
                printf ("%s %s\n",e5,ffname);
                return(0);
            }
            if (i++ == 0)
            /* first edge */
                /* save beginning edge list pointer */
                sepx = epx;
            else
                /* link it to the previous one */
                epxl->enext = epx;
                /* save pointer for linking */
                epxl = epx;
                epx->ebeax = ep->ebeax;
                epx->ebeay = ep->ebeay;
                epx->eendx = ep->eendx;
                epx->eendy = ep->eendy;
                epx->eslope = ep->eslope;
                epx->eyint = ep->eyint;
        }
    }
    /* complete the list */
    epx->enext = NULL;
    return(sepx);
}

```c
#define TIME 5
#include <stdio.h>
#include "mpanemx.h"

int inquire() /* This function controls the map inquiry process. */
{  
    char distype; /* display type */
    clearscreens();
    anmode();
    printf ("INQUIRY\n");
    printf ("What type display do you want? (t=table or d= drawing of map) ");
    readstring (instring);
    if ("instring == 't'")
        distype = 't';
    else
        distype = 'm';
    printf ("What map are you requesting? ");
    readstring (instring);
    skiplines (3);
    if ((mpx = ckcurmp (instring)) == 0) /* user is requesting a different map than the one we have in storage. */
        mp = readmap (instring);
    else
        mp = mpx;
    if (distype == 't')
    {
        printmap (mp);
        return;
    }
    /* user wants a drawing */
    do
    {
        puts_screen (mp);
        placecursor (0, 0);
        printf ("Select a subarea for inquiry ");
        printf ("by placing the cursor in that subarea.\n");
        readcursor(); /* identify subarea */
        p = inpoly (curx, cury, mp);
        if (sp == (1)
```
/* error - this is not a subarea */
placecursor(0,120);
printf ("Please position the cursor in a valid subarea\n");
delay (5);
else
{
placecursor(970,760);
printf ("%s Name: %s", indent2,sp->subname);
prtatt (mp,sp);
}
placecursor (0,60);
printf ("\nPress %s to continue",i1ret);
printf ("or terminate inquiry by typing %s ",done);
tmp = readstring (instring);
}while (tmp == 0);

put_on_screen(mp)  
struct map *mp;  /* pointer to a map */
{
clearscreen();
/* draw interior */
for (sp= mp->msp; sp != NULL; sp = sp->snext)
{
    /* outline the subarea */
drawpoly (sp->spn);
}
int printman(mp) /* prints the map as in storage */
struct map *mp: /* pointer to a map struct */
{
    fname = "printman; 
    clearscreen();
    printf("Map name = %s\n", mp->mapname);
    printf("Author = %s\n", mp->author);
    prtattb1(mp);
    printf("No. of subareas = %d\n", mp->mtoptsub);
    printf("No. of nodes in map structure = %d\n", mp->mtotnode);
    printf("Map edges\n");
    prtend(mp->mpei);
    printf(\"\nPress %s to continue.\", iret);
    /* get the ret */
    readstring (instring);
    clearscreen();
    for (sp=mp->msp; sp != NULL; sp=sp->snext)
    {
        prtsub(sp);
        prtatt(mp,sp);
        printf("\nPress %s to continue.\", iret);
        /* get the ret */
        readstring (instring);
        clearscreen();
    }
}
/* __prtattb1_____________________________________________________*/
prtattb1(mp)
struct map *mp;
{
    printf("\nAttribute table-- %s %d attributes\n", mp->mattrnb, mp->mntattr);
    printf("%sNAME TYPE LOW HIGH\n", indent1);
    for (atbp = mp->mattrbl; atbp != NULL; atbp = atbp->atblnext)
    {
        printf("%s%12s%3d%3d%7d\n", indent2, atbp->aname, atbp->atype,
                atbp->alow, atbp->ahigh);
    }
}
/* __prtsub_____________________________________________________*/
prtsub(sp)
struct subarea *sp;
{
    printf("\nSubarea %s %d nodes\n", sp->subarnme, sp->mtotnode);
    printf("%sx%dy%zd color=%5s\n", indent2, sp->sx, sp->sy, sp->scolor);
    printf("%sScreen coord. relation name = %s\n", indent2, sp->scoord);
    printf("%sWorld coord. relation name = %s\n", indent2, sp->wcoord);
    printf("%sBound by\n", indent2);
    printf("%s%zd%zd%zd%zd\n", indent2, sp->sbminx, sp->sbminy, sp->sbmaxx,
                sp->sbmaxy);
    printf("%sX Y SLOPE YINTERCEPT\n", indent2);
    printf("%sX Y\n", indent2);
    printf("WORLD X WORLD Y\n");
}
for (np = sp->spn; np != NULL; np = np->nnext) {
    prtnode(np);
}
printf ("Subarea edges\n");
prtdg(sp->spe);
}/* _prtnode________________________________________________________*/
prtnode(np)
struct node *np;
{
    printf ("%6d%5d%12.2f%11d%12d%10d\n", np->nx,np->ny,np->nslope,
        np->nyint,np->nxp,np->nwy);
}/*_prttatt__________________________________________________________*/
prttatt(mp,sp)
struct subarea *sp;
struct map *mp;
{
    printf ("\nAttributes\n");
    if (sp->stattr == NULL)
        printf ("%sNone\n", indent2);
    else
    {
        for (ap = sp->stattr.atbp = mp->matrtbl; ap != NULL; ap = ap->anext,
            apbp = apbp->atlnext)
            {
                printf ("%s%s (%c) %s\n", indent2, apbp->aname, apbp->atype,
                    ap->attribute);
            }
    }
}/*_prtdg__________________________________________________________*/
prtdg(ep)
struct edge *ep;
{
    printf ("XsBEG X Y END X Y SLOPE YINTERCEPT\n", indent2);
    for (i; ep != NULL; ep = ep->enext)
        {
            printf ("%1d%1d%1d%11.1f%11d\n", ep->begx,ep->begy,
                ep->endx, ep->endy, ep->eslope, ep->eyint);
        }
}/* _ckcurmp_______________________________________________________*/
struct map *ckcurmp (instr)
/* This function checks to see whether the requested map
 is in core. If it is, the function returns a pointer
to it, else it returns a zero. */
char *instr; /* name of map */
{
    for (i=0; i < nummap; i++)
    {
        mpx = curmap [i];
    }
if (strcmp (mpx->mapname, instruc) == 0)
    /* found the map */
    return (mpx);

/* map not found */
return(0);
C.8 Getslope, yintercept

/*__getslope___________________________________________________________*/
float getslope(x1,y1,x2,y2) /* returns the slope of a line */
int x1,y1,x2,y2; /* end points of the line */
{
    float ret; /* value returned */
    float xdiff, ydiff; /* variables containing x and y differences */
    if (x1 == x2)
        ret = 999.9;
    else
        {
            ydiff = y1 - y2;
            xdiff = x1 - x2;
            ret = ydiff / xdiff;
        }
    return (ret);
}
/*__yintercept________________________________________________________*/
long yintercept (x,y,m) /* a function that calculates the y intercept */
    when given an x, y, and the slope of the line */
int x,y;
float m;
{
    float temp; /* temporary variable */
    long ret;
    long ceil,floor;
    if (m == 0.0)
    {
        ret = y;
        return (ret);
    }
    if (m >= 999.0)
    {
        ret = 9999;
        return (ret);
    }
    ceil = 9999 - y;
    floor = y - 9999;
    temp = m * x;
    if (temp < floor) || (temp > ceil)
        return (9999);
    ret = y - temp;
    if (ret > 9999)
```c
#include <stdio.h>
#include "pmnameex.h"

/*__bound_box-----------------------------------------------*/

int bound_box (b1x1,b1y1,b1x2,b1y2,b2x1,b2y1,b2x2,b2y2)
/* This function returns a 1 if the boxes
   described by (b1x1,b1y1,b1x2,b1y2) and
   (b2x1,b2y1,b2x2,b2y2) overlap, 2 if they overlap
   on boundaries only, else it
   returns a 0. */

int b1x1,b1y1,b1x2,b1y2,b2x1,b2y1,b2x2,b2y2; /* box boundaries */
{
  int p1,p2,p3,p4; /* vertices of the box 1 */

  /*
     p1----------p2
     |          |
     |          |
     |          |
     |          |
     p4----------p3
  */

  int xmin,xmax,ymin,ymax; /* min. and max. x and y values */

  if (b1x1 < b1x2)
  {
    xmin = b1x1;
    xmax = b1x2;
  }
  else
  {
    xmin = b1x2;
    xmax = b1x1;
  }

  if (b1y1 < b1y2)
  {
    ymin = b1y1;
    ymax = b1y2;
  }
  else
  {
    ymin = b1y2;
    ymax = b1y1;
  }
```
*/ If any vertex of box 1 lies in box 2, they overlap. */
if (  
    (p1 = bb (xmin,ymin,b2x1,b2y1,b2x2,b2y2)) == 0) ||  
    (p2 = bb (xmax,ymin,b2x1,b2y1,b2x2,b2y2)) == 0) ||  
    (p3 = bb (xmax,ymin,b2x1,b2y1,b2x2,b2y2)) == 0) ||  
    (p4 = bb (xmin,ymin,b2x1,b2y1,b2x2,b2y2)) == 0)  
    return (1);  
if ( p1 == 9) || (p2 == 9) || (p3 == 9) || (p4 == 9)  
    /* point is on the boundary of the box */  
    return(2);  
if ((p1 == 1) || (p1 == 2) || (p1 == 8))  
    &&  
    ((p3 == 4) || (p3 == 5) || (p3 == 6))  
    return(1);  
else  
    return(0);  
}
int bb (x,y,x1,y1,x2,y2)
/* function which tells us if a point is contained in a box */
int x,y; /* coordinates to be tested */
int x1,y1,x2,y2; /* boundaries of the box */
{
    int xmin,xmax,ymin,ymax; /* min. and max. of x1,x2 and y1,y2 */
    if (x1 <= x2)
    {
        xmin = x1;
        xmax = x2;
    }
    else
    {
        xmin = x2;
        xmax = x1;
    }
    if (y1 <= y2)
    {
        ymin = y1;
        ymax = y2;
    }
    else
    {
        ymin = y2;
        ymax = y1;
    }
    /* Note: This function returns zero if the point is in the box,
     a 9 if the point is on the border of the box,
     otherwise it returns an integer describing the relation of the point
     to the box. */

if (((x == xmin) || (x == xmax)) &&
    ((y <= ymax) && (y >= ymin)) ||
    ((y == ymin) && (y == ymax)) &&
    ((x <= xmax) && (x >= xmin)))
    /* boundary condition */
    return(9);

if (x < xmin)
    /* it must be in 1, 7, or 8 */
    {  
      if (y < ymin)
          return(7);
      else if (y > ymax)
          return(1);
      else return(1);
    }
else if (x > xmax)
    /* it must be in 3, 4, or 5 */
    {  
      if (y < ymin)
          return(5);
      else if (y > ymax)
          /* it must be in 2, 6, or 8 */
          return(3);
      else return(4);
    }
else
    {  
      if (y > ymax)
          return(2);
      else if (y < ymin)
          return(6);
      else return(0);
    }
int intercept(xa1, ya1, xa2, ya2, m_a, b_a, xb1, yb1, xb2, yb2, m_b, b_b)
/* This function determines the intercept of 2 lines,
a and b. It returns the following:
0 - If the lines intersect at one point.
xinter1 and yinter1 are set to reflect the intersection.
Josh
Josh

1 - If the lines are identical.
------

2 - If the intersection is a line with left endpoints
identical. Xinter1 and yinter1 are set to reflect the
intersection.
------
1 2 3

3 - If the intersection is a line with right endpoints
identical. Xinter1 and yinter1 are set to reflect the
intersection.
------
3 2 1

4 - If the intersection is a line which is entirely contained
in the first line. Xinter(1,2,3, and 4) and yinter
(1,2,3 and 4) are set as follows.
------
1 2 3 4

5 - If the intersection is a line which is not entirely
contained in the first line. Xinter (1,2,3, and 4) and
yinter(1,2,3, and 4) are set as follows.
------
1 2 3 4

6 - If the intersection is at one of the endpoints only.
Xinter1 and yinter1 are set.

-1 - No intersection.
-2 - Error.
/*

int x1, y1, x2, y2, x3, y3, x4, y4;
int xmin, xmax, ymin, ymax;

int k = 0; /* count number of intercepts */

for (i = 0; i < 4; i++)

    for (j = i + 1; j < 4; j++)

        intercept(x1, y1, x2, y2, x3, y3, x4, y4, xmin, xmax, ymin, ymax, k);

*/

int x1, y1, x2, y2, x3, y3, x4, y4; /* points */
long b_a, b_b; /* y intercepts */
float m_a, m_b; /* slopes */

{
    int ret; /* value to be returned */
    int std_form(); /* function which puts lines in standard form */
    int bound_box(); /* function which checks to see in two boxes
or lines overlap */
    int fintercept(); /* recursive function which finds actual intersection */
    int lb, ub; /* upper and lower bounds */
    int b; /* holds results of bound_box */

    /* initialize variables that hold intercept points */
    xinter1 = 999;
    xinter2 = 999;
    xinter3 = 999;
    xinter4 = 999;
    yinter1 = 999;
    yinter2 = 999;
    yinter3 = 999;
    yinter4 = 999;

    /* if either line is almost vertical assume it is vertical */
    if (b_a == 999).
        m_a = 999.9;
        m_b = 999.9;
    else
        if (std_form(&x1, &y1, &x2, &y2, &m_a, &b_a, &x3, &y3, &x4, &y4, &m_b, &b_b) != 0)
            return(-2);
        if (.b = bound_box(x1, y1, x2, y2, x3, y3, x4, y4)) == 0
            /* bounding boxes do not overlap */
            return(-1);
        if (.m_a == m_b) &\& .m_a == 999.9)
            /* both lines are vertical */
        {
            yinter1 = y1;
            xinter1 = x1;
            xinter2 = x1;
            xinter3 = x1;
            xinter4 = x1;
            if (y1 == yb1)
                /* lowest points intersect */
                {
                    if (ya2 == yb2)
                        return(1);
                    if (ya2 < yb2)
                        yinter2 = ya2;
                        yinter3 = yb2;
                        return(2);
                }
        }
    else
    { /* lines do not overlap */
        /* find intersection */
        /* find intersection */
        /* find intersection */
    }
    return(-3);
}
{  
yinter2 = yb2;
yinter3 = ya2;
return(2);
}
else if (ya2 == yb2)  
/* upper points intersect */
{
yinter2 = yb1;
yinter3 = yb2;
return(3);
}
else if (ya2 == yb1)  
{
yinter1 = yb1;
return(6);
}
else  
{
yinter2 = yb1;
if (ya2 < yb2)  
{
yinter4 = yb2;
yinter3 = ya2;
return(5);
}
else  
{
yinter4 = ya2;
yinter3 = yb2;
return(4);
}
}
if (m_a == 999.9)  
/* only a is vertical */
{
xinter1 = xa1;
if (((xa1 == xb1) && (ya1 == yb1)))  
{
yinter1 = ya1;
return(6);
}
if (((xa2 == xb1) && (ya2 == yb1)))  
{
yinter1 = ya2;
return(6);
}
if (((xa2 == xb1) && (ya2 == yb1)))  
{
yinter1 = yb1;
return(6);
}
```c
if ((x1 == x2) && (y1 == y2))
{
    yinter1 = y2;
    return(6);
}
else if (m_b == 999.9)
    /* only b is vertical */
{
    xinter1 = x1;
    if ((x2 == x1) && (y2 == y1))
    {
        yinter1 = y1;
        return(6);
    }
    if ((x2 == x1) && (y2 == y2))
    {
        yinter1 = y2;
        return(6);
    }
    if ((x1 == x2) && (y1 == y2))
    {
        yinter1 = y1;
        return(6);
    }
    if ((x1 == x2) && (y1 == y2))
    {
        yinter1 = y2;
        return(6);
    }
    yinter1 = m_a * x1 + b_a;
    return(0);
}
if ((x1 == x2) &&
(y1 == y2))
    /* left endpoints are the same */
{
    if ((x2 == x1) &&
(y2 == y1))
        /* left and right endpoints are the same */
        return (1);
    else if (m_a == m_b)
        /* left endpoints and slopes are the same */
{
    xinter1 = x1;
    yinter1 = y1;
    if (x2 < x1)
    {
        xinter2 = x2;
        yinter2 = y2;
```
xinter3 = xb2;
yinter3 = yb2;
}
else
{
    xinter2 = xb2;
yinter2 = yb2;
xinter3 = xa2;
yinter3 = ya2;
}
return(2);
}
else if (( xa2 == xb2) &&
        (ya2 == yb2) &&
        (m_a == m_b))
/* right endpoints and slopes are the same */
{
    xinter1 = xa1;
yinter1 = ya1;
xinter2 = xb1;
yinter2 = yb1;
xinter3 = xb2;
yinter3 = yb2;
return(3);
}
if ((m_a == m_b) &&
    (b_a == b_b))
/* slopes and y intercepts are identical */
{
    xinter1 = xa1;
yinter1 = ya1;
xinter2 = xb1;
yinter2 = yb1;
    if (xa2 < xb2)
    {
        xinter3 = xa2;
yinter3 = ya2;
xinter4 = xb2;
yinter4 = yb2;
return(5);
    }
else
    {
        xinter3 = xb2;
yinter3 = yb2;
xinter4 = xa2;
yinter4 = ya2;
return(4);
    }
}
if ((x1 == xb1) &&
(y1 == yb1))
/* endpoints are identical */
{
xinter1 = xa1;
yinter1 = ya1;
return(6);
}
else if ((xa2 == xb2) && (ya2 == yb2))
{
xinter1 = xa2;
yinter1 = ya2;
return(6);
}
else if ((xa2 == xb1) && (ya2 == yb1))
{
xinter1 = xa2;
yinter1 = ya2;
return(6);
}
lb = xb1;
if (xb2 < xa2)
   ub = xb2;
else
   ub = xa2;
ret = fintercept (lb,ub,m_a,m_b,b_a,b_b);
return(ret);
/* __fintercept */

int fintercept(int lb, ub, m_a, m_b, b_a, b_b)
/* returns a zero and sets xinter1 and yinter1 when the intercept
is found */

int lb, ub; /* upper and lowers bounds in x direction */
float m_a, m_b; /* slope of line a and b */
long b_a, b_b; /* yintercept of lines a and b */

{
    int lb_y_a, lb_y_b, ub_y_a, ub_y_b; /* y values of lines a and b at
    upper and lower bounds */
    int f; /* result of fintercept */
    int wid, hwid; /* distance between ub and lb */
    int diff_lb, diff_ub; /* distance between y values at upper and
    lower bounds */
    lb_y_a = m_a * lb + b_a;
    lb_y_b = m_b * lb + b_b;
    if (lb_y_a == lb_y_b)
    {
        xinter1 = lb;
        yinter1 = lb_y_a;
        return(0);
    }
    ub_y_a = m_a * ub + b_a;
    ub_y_b = m_b * ub + b_b;
    if (ub_y_a == ub_y_b)
    {
        xinter1 = ub;
        yinter1 = ub_y_a;
        return(0);
    }
    if ( ((lb_y_a > lb_y_b) &&
          (ub_y_a > ub_y_b)) ||
         ((lb_y_a < lb_y_b) &&
          (ub_y_a < ub_y_b)))
        return(-1); /* due to rounding error the intercept was missed */

    diff_lb = lb_y_a - lb_y_b;
    diff_ub = ub_y_a - ub_y_b;
    if (diff_lb < 0) diff_lb = -1; /* take absolute value of diff_lb and diff_ub */
    if (diff_ub < 0) diff_ub = -1;
    wid = ub_lb;
    hwid = wid / 2;
    if (hwid == 0)
        hwid = 1;
    if (diff_lb < diff_ub)
    {
        if ( wid > 2)
        {
            /* further processing here */
        }
    }
}
}`
  ` else if (diff > 2)
  ` else return(f);
  ` else if (diff < 1)
  ` else return(f);
  ` else
else
  ` else return(f);
  ` else return(f);
  ` else
  ` else
  ` else
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f);
  ` else return(f)

```
int switchit(int x1, int y1, int x2, int y2, int m, int b)
    /* switchit(x1, y1, x2, y2, m, b) */
    /* switches these two variables */
    int *var1, *var2;
    /* pointers to variables to be switchitected */
    { int temp;
      temp = *var1;
      *var1 = *var2;
      *var2 = temp;
      return(0);
    }

    /*_switchit___________________________*/

    int fswitchit(float m_a, float m_b)
    /*_fswitchit___________________________*/
int fswitchit(var1, var2)          /* switches these two variables */
float *var1, *var2;               /* pointers to variables to be switchited */
{
    float temp;
    temp = *var1;
    *var1 = *var2;
    *var2 = temp;
    return(0);
}

int lswitchit(var1, var2)         /* switches these two variables */
long *var1, *var2;                /* pointers to variables to be switchited */
{
    long temp;
    temp = *var1;
    *var1 = *var2;
    *var2 = temp;
    return(0);
}
struct subarea * inpoly (x, y, mp)
/* This function returns a 0 if the given point (x, y) is not in
any subarea of the map, otherwise it returns a pointer to
the subarea it is in. */

int x, y; /* points given */
struct map * mp; /* pointer to map */

int nsuba; /* counter for subareas */
sp = mp->msp;
for (sp = mp->msp->snext; sp != NULL; sp = sp->snext)
{
    if (ckedge (x, y, sp) == 0)
        return(sp);
}
return(0);

/* __ckedee */
int ckedee pij(x, y, ep) /* returns 0 if in this subarea else returns -1 */
/* After checking that the point is within the bounding box, this function
checks each edge for intersection with the vertical line that
passes through the point (x, y). In the event of intersections
the x values are saved in the array, savex. This array is then
scanned to determine whether the point is actually inside or
outside the polygon. */

int x, y; /* points given */
struct subarea *sp; /* pointer to a subarea */

int savex[10]; /* area to save x values that cross y */
int cnt; /* counter of points on vertical lines */
int bi; /* holds results of bb test */
static int cnt; /* counter for savex */
long yinterc; /* y intercept of scan line */
/* checking that the point is contained in the bounding box */
cnt = 0;
if ((bb(x, y, sp->sbminx, sp->sbminy, sp->sbmaxx, sp->sbmaxy) > 0)
    & (b < 0))
    return(-1);

yinterc = y;
for (ep = sp->sp; ep != NULL; ep = ep->enext)
{
    temp = intercept (sp->sbminx, y, sp->sbmaxx, y, 0, yinterc,
    ep->ebe, ep->eben, ep->eenx, ep->eendy,
    ep->eslope, ep->eyint);
    if (temp == 0)
    {
        savex[cnt++] = xinter1;
else if (temp == 1)
    /* lines are identical -- the point is in the polygon */
    return(0);
else if ((temp == 2) || (temp == 3))
{
    savex[cnt++] = xinter1;
    savex[cnt++] = xinter2;
}
else if (temp == 4)
{
    savex[cnt++] = xinter2;
    savex[cnt++] = xinter3;
}
else if (temp == 5)
{
    /* error -- bounding box exceeded */
    return(-1);
}
else if (temp == 6)
    /* intersection is at an endpoint of the bounding box line */
{
    savex[cnt++] = xinter1;
}
}

ctr = 0;
for (j=0; j<cnt; j++)
{
    /* count points that are left of x */
    if (x < savex[j])
        .ctr++;
}
if (ctr % 2 == 0)
    /* counter was even, point is not in */
    return(-1);
else
    /* counter, odd, point is in */
    return(0);

/* getslope_-----------------------------------------*/

float getslope(x1,y1,x2,y2) /* returns the slope of a line */
int x1,y1,x2,y2; /* end points of the line */
{
    float ret; /* value returned */
    float xdiff,ydiff; /* variables containing x and y differences */
if (x1 == x2)
  ret = 999.9;
else
{
  ydiff = y1 - y2;
  xdiff = x1 - x2;
  ret = ydiff / xdiff;
}
return (ret);

/* __yintercept_________________________________________________________ */

long yintercept (x,y,m) /* a function that calculates the y intercept
when given an x, y, and the slope of the line */
int x,y;
float m;
{
  float temp;      /* temporary variable */
  long ret;
  long ceil,floor;
  if (m == 0.0)
  {
    ret = y;
    return (ret);
  }
  if (m >= 999.0)
  {
    ret = 9999;
    return (ret);
  }
  ceil = 9999 - y;
  floor = y - 9999;
  temp = m * x;
  if (temp < floor) || (temp > ceil)
  return (9999);
  ret = y - temp;
  if (ret > 999)
    ret = 9999;
  return (ret);
}
C.14 Intro1,intro2,draw1,draw2

File Intro1

DRAW-A-MAP

You have entered the DRAW-A-MAP system. The purpose of this system is to use an interactive graphics terminal to draw a map and then to store the map with its graphical and non-graphical data in a relational data base.

As you use this system you will be prompted to enter data or make decisions. Unless otherwise instructed, the answers to the prompts must be typed on the keyboard followed by a 'ret' (i.e., the return key). Where you are required to select an option from a list, place the cursor, using the wheels with arrows, on the option you want and then type a period ('.').

The cursor in this system is a square. When it is flashing, the system is indicating to you that you must respond.

Now type 'ret' to begin.
Select the operation you wish to perform by placing the cursor on that operation.

1. Draw a map.

2. Delete a map that has been stored in the data base.

3. Edit a map that has been stored in the data base.

4. Make an inquiry of a map.

5. Exit this program.
You are now ready to draw the boundaries of the map. Place the
 cursor at a vertex and enter a `. `. Move the cursor to the next
 vertex and enter `. `. A bell will ring to indicate to you
 that the system has recorded the vertex. Continue to enter vertices
 around the boundary. The boundary will be considered to be closed
 when the first vertex is reentered. At the closure of the boundary,
 two bells will ring.

 Now type `ret` to continue.

You are now ready to draw the subareas of the map. Place the
cursor at a vertex and enter a `. `. Move the cursor to the next
vertex and enter `. `. A bell will ring to indicate to you
that the system has recorded the vertex. Continue to enter vertices
around the subarea. The subarea will be considered to be closed
when the first vertex is reentered. At the closure of the subarea,
two bells will ring.

 Now type `ret` to continue.


Van Dam, A. and Evans, L., "Data Structures Programming System," *Proc. IFIP Cong.*, 1 Amsterdam: North


