OBJECT-ORIENTED GRAPHIC USER INTERFACE FOR TRANSPORTATION SCHEDULING PROBLEM SPECIFICATION

By

FELIPE GUACACHE

Information Science Engineer

Central Technological University

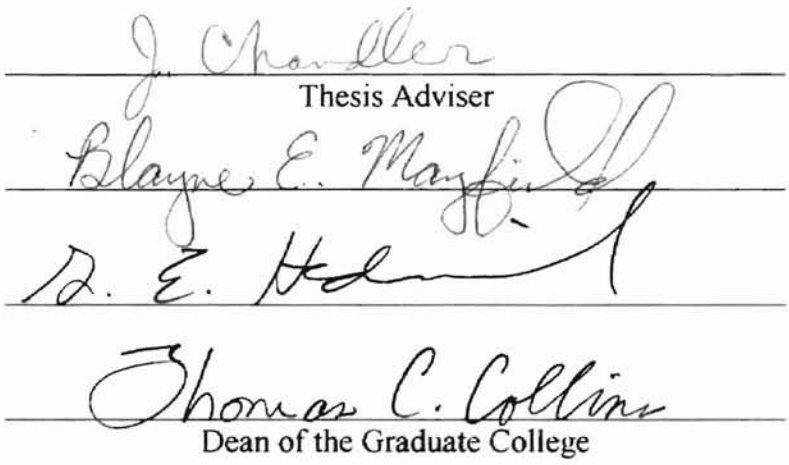
Guacara, Venezuela

1991

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1996

OBJECT-ORIENTED GRAPHIC USER INTERFACE FOR TRANSPORTATION SCHEDULING PROBLEM SPECIFICATION

Thesis Approved:



Thesis Adviser

Dean of the Graduate College

ACKNOWLEDGMENTS

I want to thank my advisor Dr. John P. Chandler for all the help he has given me during my time at Oklahoma State, more than just my Graduate Coordinator and advisor he has been a good friend, this ultimate step would not have been possible without him. My thanks go also to Dr. Blayne Mayfield, for all the help as Head of the Department of Computer Science, God knows how many times I have bothered him, and for the invaluable pieces of advice in Object-Oriented Programming. I extend my thanks to Dr. George Hedrick for accepting to be in my committee with such a short notice, I really appreciate it.

Thanks also to my teachers at the English Language Institute, Kay Keys, Jeanne Horton, Dianne Graalman, Louise Nelson, and the late Mary O'Carroll. Their English classes and valuable advice helped me very much.

I would like to express my appreciation to the people at the Office of International Programs, Dr. Arthur Klatt, Tim Huff, Karen Viljoen, Christie Millis, Karen Sebring, Karen Broam, Wannapha Byrne, Tina Henry, John Witt, Adele Tongco, Monica Shinault, Mahzan Yusof, and Ivanette Blanco for their continuous support and a well done job.

I would like to thank also Mrs. Deanna and John Homer. They have always been like my parents and made me feel at home all the time. Thanks go also to my family back home, Dad, Mom, Angel, Ana, Leo, Neil, Titi, Guma, Yani, Vero, Emmanuel and Angito for their support and their prayers.

Finally, I would like to thank the Department of Computer Science at Oklahoma State for the past two years and a half, my time here has been a lot of fun.

TABLE OF CONTENTS

1. INTRODUCTION.

This work is part of a larger and more ambitious project taking place in Oklahoma State University and the Air Force[Smit93]. The ultimate goal of that project is to develop a tool capable of automatically generating software to solve transportation scheduling problems.

To do so, the data have to be obtained from the user and stored in a convenient way. Next the problem has to be classified since there are many variations of the transportation scheduling problem, and each variation needs a different approach in order to be solved. Finally, once the classification is made, the data are input to a program that will generate the software needed to solve that particular problem.

The goal of this work is not to offer solutions to the transportation scheduling problem but to provide a Graphic User Interface (GUI) to get the data from the user and store the information in a convenient and efficient way.

It might occur that the people in charge of loading the data into the computer are not familiar with the terminology used in scheduling, and will need a graphic user interface that asks them questions of everyday-life kind. Also it will not be relevant for the system what kind of data are being loaded or what kind of transportation scheduling the problem might be classified as. Hence, the targets of this GUI are users that are neither expert programmers nor transportation scheduling specialists; therefore the GUI must be both user-friendly and easy to use.

The goal of this work is:

- To provide a GUI specifically for transportation scheduling problem specification.
- The GUI will "know" about transportation scheduling problems. In this sense it will be "intelligent", since it will have knowledge incorporated making it able to ask the appropriate questions to get the information needed. For instance if the system is loaded with information on two different types of conveyances such as airplanes and trucks , the system will be, in the future, able to ask questions based on the fact that a given conveyance is an airplane or a truck.
- The graphic user interface will be object-oriented, so that it will be extensible, modular, and easy to maintain.
- The graphic user interface should be user-friendly and very easy to use. Dialog boxes, check boxes, text boxes, radio buttons, mouse handling, icons, and help windows will provide the user with all the information he/she will need to use it.
- It will integrate geographical information data, transportation scheduling data and constraint data. Geographical data such as distances between cities, transportation scheduling data such as the location of a vehicle, and constraints such as availability of a certain type of conveyances in a given location will be available in the system.

An example of a company that picks up and delivers cargoes of different sizes and characteristics using two types of conveyances (airplanes and trucks) in twenty cities of the United States will be used to develop the GUI. Cargoes need to be picked up in some locations and dropped off in some others, each one has an earliest arrival time and a latest arrival time, airplanes have a capacity and maximum range and so do trucks, and there will be restrictions such as there are no airplanes in location "x", or no truck heavier than "p" pounds can go to location "y"; information like this is what the system is going to store.

II. TRANSPORTATION SCHEDULING PROBLEM

Let us consider a company that picks up and delivers packages. It has to have one or several vehicles or conveyances. It has shipments that have to be picked up in a location and delivered in another location. It has one or more places in which to store the vehicles and the shipments. It has restrictions on delivery time, vehicle sizes and capacities, the kind of shipments that can be transported using the vehicles (e.g. some vehicles might not be able to transport food or other kind of perishable shipments), and so on.

This company has to figure out an effective way of deciding which vehicle has to pick up which shipments in which locations at what time so it can be profitable. It has to schedule every route and the tasks (pickups and drop-offs) to be done as efficiently as possible so it can respond very quickly to changing situations (e.g. truck breakdowns).

The transportation scheduling problem is a non-trivial one. Its computational complexity is NP-hard, meaning that there is no known algorithm able to solve it in polynomial time based on the size of the input; thus a huge amount of computation is needed in order to solve large problems.

There are many flavors of the transportation scheduling problem, some more complex than others. The data needed to solve the different kinds of problems may vary from one problem to the other, making it very difficult to have an algorithm able to solve efficiently all of the different flavors. Therefore we need to classify transportation scheduling problems in order to solve them.

II.1. Classification of transportation scheduling problems.

The ultimate goal of any transportation scheduling is to produce a schedule as output, i.e., a list of activities (pickups and drop-offs) to be accomplished by each driver in the given locations with the restrictions of time and following a specified route.

The goal of solving scheduling problems is to produce a schedule and a route, which are not very different for different kinds of problems, so the problems must be classified based on their characteristics. Considering two problems of the same kind, the fact that there are time restrictions makes them totally different. For example the problem of delivering newspapers and the problem of delivering food by a catering service are similar in the sense that the goods to be delivered are in a depot and must be taken to the customers.

However, in the case of the newspapers, as long as they are all delivered within a short period of time (say two or three hours) the time considerations may be ignored and the problem is purely one of routing. The caterer, on the other hand, should deliver the food at a certain time; and he/she better do this if he/she wants to keep the customer.

II.2. Characteristics of Transportation Scheduling Problems

Scheduling problems can be represented as graphs where the nodes are locations and the arcs represent the fact that it is possible to go from the location at one end of the arc to a location at the other end. It might be the case that it is possible to go from location "a" to location "b" but not from location "b" to location "a", in this case the arc between location (node) "a" and location (node) "b" is said to be directed, if the direction of the arc is irrelevant, it is said to be undirected.

The U.S. Department of Transportation uses the following list of characteristics of transportation scheduling problems to classify them (see [Bodi81]) :

Size of the vehicle fleet available: the fleet is the set of all the vehicles the company has to carry out the activities that are to be scheduled.

- One Vehicle: only one vehicle is used to carry out the activities.
- More than one vehicle: two or more vehicles are used to carry out the activities.

Type of fleet available

- Homogeneous: the same kind of vehicles, planes, trains, or boats are used.
- Heterogeneous: different kind of vehicles, planes, trains, or boats are used. A vehicle would be considered different from another one of the same kind if it has a special feature. For example two airplanes of the same capacities would be considered different from each other if one of them is equipped to transport refrigerated goods.

Nature of demands: the demand is the amount of goods that are to be delivered at a given location.

- Deterministic: we know ahead of time the amount of goods to be delivered at each location.
- Stochastic: we do not know ahead of time the amount of goods to be delivered to at least one location; there are probabilities associated with the amount of goods to be delivered. For example, based on a study of the behavior of the demand in a given location one could

know the probability of the demand being in different ranges of amount of goods.

Location of demands: sometimes the problem specification requires that goods be picked up at one place (say a depot) and delivered at another place (say a customer or another depot). In that case, we can consider the source and destination as nodes of the graph and the demand is in a node of the graph. However, there are other cases where the drop-offs are not in a final destination, but in the route used to get to the final destination. For instance, a mail truck departs from the post office, delivers mail in its route and comes back to its original location. In this case the demand is not in the nodes of the graph but in the arcs.

- At nodes (not necessarily all): generally the location of the demand will be at the nodes of the graph.
- At arcs (not necessarily all): when the nodes have a special meaning, like locations at which the company has offices for instance, the demands might be located in some of the arcs of the graph. In this case the graph might be redrawn so the demands are located at the nodes.
- Mixed: some demands are located in the arcs and some are located in the nodes.

Underlying network: as in graph theory, the arcs can be directed or undirected, depending on whether or not it is important in which direction the arcs are traversed. For example, if two sites (nodes) are connected through a highway, the arc (highway) can be traversed in both directions and the direction is not relevant anymore. However, if there is a place "x" from where one can fly to place "y"

but to go back one has to go first to place "z" before going back to place "x", for the arc between "x" and "y" the direction is important because it is from "x" to "y" only.

- Undirected: the direction of the arcs in the graph is irrelevant.
- Directed: the direction of the arcs in the graph is important.
- Mixed: some arcs are directed and some are not.

Vehicle capacity constraints: the vehicles themselves have a limited capacity beyond which is not possible to load them. Besides, it might occur that there could be regulations on the weight of the vehicles that circulate in a highway or in a city or in a state

- Imposed all the time: either the capacity of the vehicle is fixed or there are regulations that limit it during all of the route.
- Imposed not all the time: the capacity of the vehicle is constrained by external agents in some parts of the route whereas in others it might be augmented.
- Not imposed: the capacity of the vehicle can be augmented as much as needed; for example cars might be attached to a train.

Costs: the costs can be classified as fixed or variable.

- Variable or routine costs: the costs that increase with the distance that has to be covered in order to carry out the activities, such as gasoline, oil, tires, and maintenance expenses.
- Fixed operating or vehicle acquisition costs (capital costs): the cost of the vehicles themselves, for example.

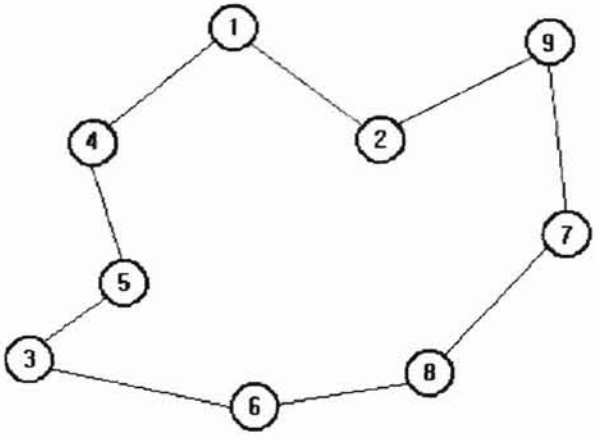
Operations: the type of operations can be either pickups or drop-offs or both.

- Pickups only
- Drop-offs only
- Mixed

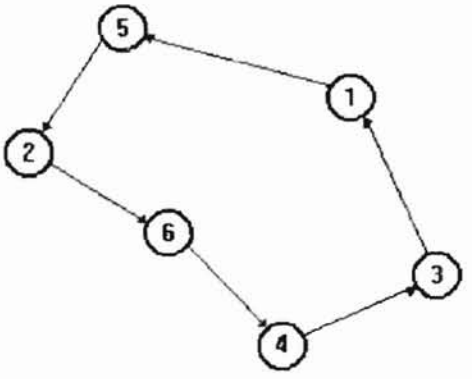
Objective: the objective of the schedule varies from one case to the other

- Minimize routing costs incurred
- Minimize sum of fixed and variable costs
- Minimize number of vehicles required

Combining all the characteristics above, one can have a large number of problem settings. A well known scheduling problem is the so called *traveling salesman problem* (TSP), which requires the determination of a minimal cost route for a salesman that needs to visit several locations; i.e., a cycle that passes through each node exactly once. In this case there is only one vehicle involved, the arcs may be directed or undirected depending on the specification of the problem, and the objective is to minimize the sum of fixed and variable costs. If the costs are symmetric; i.e., they do not depend on the direction of the travel, the problem is said to be symmetric; otherwise they are said to be asymmetric.



Symmetric TSP



Asymmetric TSP

The traveling salesman problem is known to be NP-hard, so we can say intuitively that the complexity of our example of the company that picks up and delivers packages is at least as complex as that one when we have the case of only one vehicle to carry out the activities. In other words our example is of NP-hard complexity.

II.3 Approaches to solving Transportation Scheduling Problems

As said before, the scheduling problem can be represented as graphs where the nodes are locations and the arcs represent the fact that it is possible to go from the location at one end of the arc to the location at the other end. If it is possible to go in one direction the arcs are said to be directed, otherwise they are said to be undirected.

After having the graph drawn in a convenient way, one can apply several approaches to solving it. For example, if the graph is a tree, depth-first search, breadth-first search, best-first search, beam search, hill climbing, or any other tree traversal strategy can be used [Wins92]. If the graph is not a tree, one can either sort the graph based on a certain attribute (topological sort [Wins92] might be a choice) and apply any of the approaches above or explore all the possible combinations until the optimal schedule is found.

The latter approach (exploring all the possibilities) might not be the best one since the amount of computation increases exponentially as the size of the input increases, for all known algorithms that are guaranteed to achieve absolute optimality.

The complexity of the problem makes it necessary that the data needed to solve it be stored in a convenient and efficient way to minimize the computational burden of dealing with the data when the problem is being solved.

III. SPATIAL AND GEOGRAPHICAL DATABASES.

The classical definition of a database is a set of interrelated data files, each one consisting of data records (tuples) that allows a number of operations to manipulate the data files and the tuples of a given data file, while minimizing data redundacy, avoiding data inconsistency and preserving data integrity.[Date91]

Operations such as the creation of new data files, the insertion, modification, retrieval, and deletion of data tuples and the removal of existing data files are the most important ones, but there might be others, like the joining of two or more different files and the generation of reports that, if not essential, are very useful and convenient. If the database is a relational one, the data files are called tables (even the relations are considered to be tables), and the tuples are called rows.[Date91]

Implicit in the previous definition is the existence of named (in the sense of distinguishable) logical units and relationships between those logical units. The universe that is to be represented by means of the database has to be broken down into those logical units in the process of designing and constructing the database.

The universe represented by a spatial database cannot be broken down into logical units, so it must be somehow abstracted, generalized or approximated in order that the database be created. Data modeling is very important for geographical databases and defines the

way the final user will perceive the universe represented. The success of a geographical database depends on its capability of representing accurately the universe; to do so, it requires not only efficient access to a database but also accurate data models.

Spatial and geographical data differ from other kinds of data in that they are associated with a topological object (point, line, or polygon) that has a position somewhere in space. The difference between spatial and geographical data is that spatial data are referred to an arbitrary point in space which is the origin of the coordinate system, whereas geographical data are referred to a point located on the earth. For example a Computer Aided Design (CAD) system uses spatial data rather than geographic data when used to design a tridimensional figure.

From the above it follows that if a data model consists of logical units of data and such logical units are abstractions or approximations of geographical reality, two data models referring to the same universe might represent it in different ways. Geographical data describe geographical reality in terms of: the position of geographical objects in a specified coordinate system, their associated attributes which are unrelated to position and their spatial or topological interrelations which describe how the objects are linked together in the system.

The geographical reality refers to empirically verifiable facts about the real world; however, those facts may not be certain due to subjectivity or errors. Very often the

relationship between reality and database is complicated by the existence of a map or other kind of representation of reality with its own data model. Information is lost between reality and the map or representation, and between the representation and the database. This makes it even more difficult to present the final user with an accurate representation of reality.

III.1. Geographical Reality.

If geographical data are represented as points in a two-dimensional space, the tuple $T = <x,y,z_1,z_2,...,z_n>$ giving the values of n spatial variables at location (x,y) may be used as the fundamental element of geographical information.

Since x and y are continuous, the number of tuples is infinite,so the process of modeling becomes the process of representing an infinite set of tuples by means of a finite set of tuples big enough to give the user an accurate view of the reality and small enough to fit the constraints of digital storage. x and y are also continuous in maps and other kinds of respresentation; the problem is how to represent the variability of z_1 through z_n. It is desirable to map as many variables as possible into a single mapped variable, by means of a function $f(z_1, z_2,...,z_n)$.

In general, the similarity between the variables in the tuples $T_1 = <x_1,y_1,z_{11},z_{12},...,z_{1n}>$ and $T_2 = <x_2,y_2,z_{21},z_{22},...,z_{2n}>$ increases as the locations of T_1 and T_2 become closer to each

other. This is called *spatial autocorrelation* and it can be exploited to discretize the spatial variation. Two general discretization strategies are used. The first one is called the *sampling* strategy and it assumes that (x_1,y_1) and (x_2,y_2) must be more than a certain minimal distance apart before the associated tuples become noticeably different. The second one is called the *piecewise* strategy and it assumes that the universe can be subdivided into homogeneous, interconnected regions, where a function describes the variation within each region.

Many data models are based on discrete objects located in the plane. In this case the spatial variation is represented by a set of tuples $<i,a_1,a_2,...,a_m>$ where i is an object and a_1 through a_m are the object's attributes. Location is described by a set of tuples $<x,y,o_1,...,o_i,...>$, where o_i is a binary variable indicating whether or not object i is present at location (x,y). A variation of this approach will be used to handle the geographic data in the graphic user interface.

There are other ways of representing geographical data. [Good92] offers a fairly thorough discussion and more references for readers interested in further information.

III.2. Geographical Data Modeling

Two spatial data models are used most widely for geographical data modeling. They are the vector and the tesseral (also known as tesselation, grid, matrix, and raster) models. For raster data structures, the fundamental unit of representation is usually the square

pixel, whereas for vector data structures it has been defined in several ways, a link being the one suggested [Raper92].

The expressive power of the representation is determined in a high degree by the information content of the data model; however, spatial data is very dependent on the data model chosen, perhaps more than any other kind of data. The two fundamental units (pixels and links) cannot be considered to be identical because they behave in different ways under transformation. There is a simmetry between these two representations but they are qualitatively different.

[Raper92] introduces the following classification for data structures used in geographical data modeling:

Vector data structures.

- Unstructured (spaghetti): each geographical feature is represented as a string of x,y coordinates.
- Link and node (spaghetti and meatball): intersecting lines in a link and node structure must share a node, and all closed areas are identified.
- Hierarchical: incorporates explicit relationships between polygons and their bounding lines and between lines and their constituent points in a hierarchical system.

- Topological directional: stores start nodes, end nodes, and direction of digitizing; the relationships between lines and polygons is determined by processing.
- Topological complex: the basic atomic unit is the common boundary between polygons; all adjacent polygons are stored for each boundary.
- Object-based: topologically complex units that can represent complex polygons made of other polygons, or support overlapping polygons.

Tesseral (raster) data structures.

- Regular unstructured: a matrix of unindexed cell values; the most popular shape for cells is the square.
- Simple structured: an improved version of the unstructured tesseral that is more efficient in the use of data storage, by run length encoding, for example.
- Nested structured: solves the storage problem by recursively subdividing a grid. The quadtree and the R-trees fall in this category.
- Irregular: the space is subdivided into irregular subspaces. The most widely used is the Triangular Irregular Network (TIN) [Jone94].

Hybrid data structures.

- Vaster: Peuquet (see reference in [Raper92]) developed the vaster data structure to make use of the advantages of both the vector and the tesseral systems. It is little used in practice.

Nonspatial data structures for spatial data.

All spatial and nonspatial data must be stored in a computer system in order to be used by the spatial database system. Five main database designs are the most widely used [Raper92].

- Flat files: the records and fields have very simple table structure. Used in very simple GIS.
- Hierarchical: each record can have links to lower level records but only one link to a higher-order record. Good for one-to-many relationships. It introduces a lack of flexibility to GIS.
- Network: similar to the hierarchical but it allows many-to-many relationships. Efficient, but its complex structure restricts its use in creating spatial databases.
- Relational: data are organized into a series of tables, with the rows of the table corresponding to records and the columns corresponding to fields. They are easy to use and set up. However, some manipulations are rather slow.

- Object Oriented: the ability to incorporate simple and complex geographical features (points, lines, polygons, sets of polygons, and rasters), and support different versions of the same feature makes object oriented databases very suitable to support spatial data.

III.3. Geographical Information Systems (GIS).

Even though this work does not pursue the building of a full GIS, it does use spatial data. Spatial data are very difficult to handle because of their special characteristics discussed above, and GIS technology offers efficient ways of approaching the different issues involved.

III.3.1. Some definitions of GIS:

- Geographical Information Systems are defined as computer systems for "capturing, storing, checking, manipulating, analyzing, and displaying data which are spatially referenced to the Earth" [(DoE 1987)] [Raper92].
- "GIS may be defined as a system capable of efficient input, storage, representation, and retrieval of spatial indexed data" [Rifai93].
- "Loosely defined, a GIS is a mixture of computer hardware and software used to collect, manage and analyze data found on maps, including both maps and associated attributes" [Sonne87].

- "Geographic information systems must be regarded as a special form of database management system which facilitates operations on spatial data" [Masse91].

In a GIS, four main components can be identified: software, hardware, data model (spatial and nonspatial) and the organizational context. In the last few years the hardware technology has become faster and cheaper. A lot of GIS software has been developed. Regarding the data model, even though some authors maintain it has been the ugly duckling since much of the emphasis has been placed on software and hardware, several data models have been made available for GIS. The success of the GIS depends on whether or not it is placed in an appropriate organizational context and properly integrated into the whole work process.

To represent the multiple variables present in each geographical location, GIS place the information in layers. Each layer contains information about a different feature, for example population, forests, roads, rivers, etc.

The fact that many GIS have been developed with different system designs makes it difficult for the user to distinguish between systems and compare their functionality and performance.

III.3.2. GIS relevancy criteria.

In [Masse91], the following GIS relevancy criteria are proposed. Masser and Blakemore state that any spatial analysis technique which is meant to be GIS relevant should include:

1. An ability to handle large N (10,000 or more spatial objects) without any fundamental difficulty.
2. It should be explicitly geographical and either use or incorporate some of the unique features that characterize spatial data.
3 There should be no barriers to eventual portability of the technology.
4. It should be capable of being interfaced or tighly coupled with GIS.
5. It should address issues and areas of applied importance.
6. There should be strong focus on generic topics.
7. The technology should be well founded and able to sustain critical scrutiny by experts from related disciplines
8. It should be usable by the end users that characterize GIS, most of whom will neither be trained spatial scientists nor spatial statisticians nor academic researchers.

In spite of the vast opportunities offered by GIS, there are major shortcomings to their use. The software and hardware needed to implement them are generally very expensive , and the database development itself is a long and expensive process.

III.4. Topological Integrated Geographic Encoding and Referencing (TIGER) System.

The TIGER system [Masse91] is an example of a geographical database. The information contained in it is used by several GIS applications. It was developed jointly by the United States Geological Survey (USGS) and the Census Bureau. It contains digital geographical data that includes roads, railways, and rivers, the associated geographical areas for collecting census data, political areas (cities and townships), and other features that allow basic analysis of changes between the 1980 and 1990 census. It is being used by the Census Bureau as a means for collecting data and in creating census products for sale to users. It contains a total of 19,000 megabytes of information. Each record contains the following data:

1. The name and type of the feature; that is whether the feature is a road or street, a waterway, a railroad, a political boundary, etc.
2. The coordinate values defining intersection points along the feature, together with other geometric characteristics of the feature; for example, the curve vectors defining the shape of the feature;
3. The range of addresses located between intersection points for those records representing streets or roads, in addition to the post office name and ZIP code of each address range;
4. The codes of the geographic areas applicable to each segment of the feature, based on the geometric relationship to the feature of the boundaries in the file for each geographic entity; and

5. Other special situations associated with the record; for example major employment centers or residential structures located along the feature.

IV. OBJECT-ORIENTED PROGRAMMING (OOP).

Object oriented programming is a new programming paradigm that represents real-world physical objects through logical objects that behave in a way that resembles the physical objects.

In the same sense as variables have a type in traditional programming languages, logical objects are grouped into *classes* in object-oriented programming. A *class* is an abstraction of the characteristics of a collection of similar objects. Classes have data components and function components, and they are referred to as *data members* and *member functions*. Member functions define the behavior of the objects of a given class.

Every logical object belongs to a class. The process of creating a new object of a given class is called *instantiation* and objects are said to be *instances* of that given class. Classes can be nested within other classes to any level of depth, in this case the child (inner) class *inherits* the characteristics of the parent (outer) class. *Inheritance* in object-oriented programming is supposed to be analogous to biological inheritance.

Object-oriented programming allows the programmer to "hide" details the final user does not need to know, providing interfaces to request the details he/she needs to know. This capability is called *encapsulation*.

Grady Booch gives the following definition for object-oriented programming: "Object-Oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships" [Booch94].

Logical objects communicate with each other by means of messages. Objects send messages to one another requesting services, and the messages trigger the execution of the appropriate *methods* (functions) in the recipients of the messages. This process is called *message passing*.

Another feature of OOP is the ability of the objects to behave in different ways according to the message passed, and the ability of designing operators to carry out object manipulation. This festure is called *polymorphism* and it can be either *overriding* or *overloading*. In the following sections this concepts will be made clearer

C++ was chosen as the programming language in which to develop the graphic user interface because it is a popular programming language that seems to have become the most widely used OOP language. All the examples in the following sections are presented in C++.

IV.1. Classes.

As said before, every object belongs to a class which contains all the data members and members functions of the objects that are instances of the class. The following example illustrates the definition of the class vector.

```
class vector {
        private:
                float magnitude, angle;
                char name;
        public:
                vector(float m, float a, char* n);
                ~vector(void);
                void setMag(float);
                void setAngle(float);
                float getMag(void);
                float getAngle(void);
                void printName(void);
        };
```

In this example *magnitude*, *angle*, and *name* are the data members of the class vector and *vector*, *~vector*, *setMag*, *setAngle*, *getMag*, *getAngle*, and *printName* are the interfaces for the member functions. The keywords *private* and *public* determine which members are accessible and which ones are not. Private members can be accessed only by objects that are instances of the class, whereas public members are accessible for any object. There is yet another keyword (*protected*), the use of which is explained in the section about inheritance. This is the way encapsulation is implemented in C++

There are two member functions that deserve special attention: *vector* and *~vector*. (Notice that the name "vector" is the same as the name of the class.) Vector and ~vector are called the *constructor* and the *destructor* of the class.

A constructor is a special member function that gets executed automatically every time a new instance of the class is created. If no constructor is defined for a class, the compiler will automatically create one constructor that takes no arguments and is called the default constructor. It is recommended that a constructor be defined in the case that, as in our example, there are data members that need to allocate memory dynamically.

A destructor is a special member function that gets executed automatically every time an object is destroyed. If no destructor is defined, the compiler will automatically create one. Destructors never take arguments. Just as before, when there are data members that need to deallocate memory dynamically allocated, the designer should define a destructor. Both constructors and destructors do not have a return type.

Destructors get executed automatically at the end of a program block for all the objects that are local to that segment of code. In some situations, destructors have to be invoked explicitly by the programmer.

```
// This is a constructor
vector::vector(float m, float a, char* n) {
        magnitude = m;
        angle = a;
        name = new char[strlen(n) + 1];
        strcpy(name, n);
        }
// This is a destructor
vector::~vector(void) {
        delete[] name;
        }
...
vector v1(25, 60.0, "My Vector");
//...
```

The first segment of code above creates an instance of the class vector, assigns values to the data members *magnitude* and *angle*, allocates a chunk of memory for the data member *name*, and copies the contents of the location pointed to by parameter *n* into the new location.

The next segment of code deallocates the chunk of memory previously assigned to name, and after that destroys the instance for which the destructor was called.

The last portion of code creates a vector instance called *v1*, which has a magnitude of 25 units, an angle of 60.0 units, and a stored name "My Vector".

The scope resolution operator ("::") tells the compiler that the functions are member of the class vector The operators *new* and *delete* work in a way similar to the library functions *malloc()* and *free()* but they are specially designed to handle objects.

IV.2. Inheritance.

One of the major advantages to inheritance in OOP is that it reduces code redundancy, which in turn might lead to data inconsistencies and data integrity violations. Children classes, often called *subclasses* or *derived classes*, inherit data members and member functions from the parent(s) class(es), also called *superclasses* or *base classes*. A class whose derived classes are in turn classes themselves is called a *metaclass*.

Inheritance presents a problem since, as we stated before, the private members of a class can only be accessed by instances of that class. That means that a derived class cannot access the private members of its base class. The keyword *protected* allows data members to be accessed by the instances of the class and its derived classes.

The base class of a class can be in turn be public, private or protected. To determine the visibility of the members of the base class the following rules apply:

- The public, protected, and private members of a public base class are inherited as public, protected, and private respectively.
- The public, protected, and private members of a protected base class are inherited as protected, protected, and private respectively.
- The public, protected, and private members of a private base class are inherited as private, private, and private respectively.

Example of inheritance.

```
class vector {
        protected:
                float magnitude, angle;
        public:
                vector(float m, float a, char* n);
                void setMag(float);
                void setAngle(float);
                float getMag(void);
                float getAngle(void);
        };
class namedVector: private vector {
        private:
                char name;
        public:
                vector::getMag(); //overrides private
                vector::getAngle(); //overrides private
                namedVector(float m, float a, char* n);
                ~namedVector(void);
```

```
    void printName(void);
};
```

In the example above, *vector* is the base class and *namedVector* is the derived class. The data members *magnitude* and *angle* are accessible for instances of class *vector* as well as for instances of class *namedVector*. Since in class *namedVector* the base class is declared to be private, all the members of *vector* are inherited as private by *namedVector*. However, the privacy is overriden for the member functions *vector::getmag()* and *vector::getAngle()* by redeclaring them in the public section of *namedVector*.

Inheritance may be used in the following cases [Budd91]:

- Specialization: the most common use of inheritance is for specialization. If an is-a relationship holds, then specialization is appropriate. Sometimes it is not easy to decide whether a relationship is is-a or has-a.
- Specification: the subclasses are not a refinement of an existent class; instead they are the completion of an incomplete, abstract class specification.
- Construction: a class inherits almost all of its functionality from a superclass, changing minor details.
- Generalization: the oposite of specialization. Used when building on existing classes that cannot or should not be changed. Its use should be avoided.
- Extension: adds totally new abilities. Overrides methods from the parent class.
- Limitation: similar to specification, but the behavior of the subclass is smaller than that of the superclass.

- Variance: classes with similar implementation where there is not a clear hierarchical relationship. Specification might be a better solution.
- Combination: also known as multiple inheritance. The base class inherits behavior from several parents.

IV.3. Operator Overloading.

Operators such as +, - ,* , /, %, ++, -- <<, >>, =, [], (), etc., can be overloaded and given new meaning when acting on instances of the class for which they were overloaded.

The following example shows how to overload some operators:

```
class timerClass {
        private:
                int hours, minutes;
        public:
                timerClass (int h=0, int m=0);
                void tick(int m);
                void tick(int h, int m);
                void print (ostream& out);
                timerClass operator ++ () {
                        tick(1);
                        return *this;
                        }
                timerClass operator + (int m) {
                        timerClass temp;
                        temp.tick(m);
                        return temp;
                        }
                timerClass operator + (timerClass& t) {
                        timerClass temp = t;
                        temp.tick(hours, minutes);
                        return temp;
                        }
                friend timerClass operator + (int m, timerClass t);
                friend ostream& operator << (ostream& out, timerClass& t);
        };
timerClass operator + (int m, timerClass& t) {
        return t + m;
        }

ostream& operator << (ostream& out, timerClass& t) {
        t.print(out); return out;
        }
```

There are some other OOP features in this example besides operator overloading. The first one is default parameters. In the *timerClass* constructor, a default value of zero is assigned to the formal parameters *h* and *m*. Also, the reference operator & is used to pass formal parameters by reference to the member functions; this avoids the requirement that the constructor be called every time an object is passed to a function.

In addition, the self-reference variable *this* is used. *This* is a pointer to a class that exists in every instance of the class, so it can refer to itself when necessary. And finally, friend functions are used. A friend function is a function that is granted access to the private members of a class. Friend functions should be used sparsely since they constitute a violation of the encapsulation.

The member function *operator ++()* overloads the ++ operator, so if a *timerClass* instance *timer* exists, *++timer* increments the timer by one minute. It also returns a copy of itself, so it can be used in conjunction with other operators.

The member functions *operator+(int m)* and *operator+(timerClass& t)* overloads the + operator to allow it to add a *timerClass* instance and an integer value or another *timerClass* instance. They also return a copy of the result.

The friend function *operator+ (int m, timerClass& t)* allows to add and integer to a *timerClass* instance. It is different from the one above because it allows operations such as *5 + timer;* whereas the latter allows the ones such as *timer + 8;*.

The friend function operator << (ostream& out, timerClass& t) overloads the output operator << and returns a reference to the output stream, so it can be used to output several objects at the time.

IV.4. Abstract Classes.

An abstract class is one that is never instantiated because it is too general or is incompletely defined. When subclassing for specification, the parent class describes very vaguely the behavior of its children. In C++, abstract classes are implemented by means of pure virtual functions.

Abstract classes are useful because they allow the implementation of polymorphic objects, i.e., objects whose behavior depends on the type of the data contained in them, rather than on the type of the object itself. In C++, a pointer to a class can hold not only addresses of instances of that class but also addresses of instances of derived classes; a pointer to an abstract class is used to hold a pointer to derived class instances. This allows the execution of the different definitions for the pure virtual function in the derived classes.

IV.5. Multiple Inheritance.

Multiple inheritance is sometimes the only way to get around complicated relationships. The following example of multiple inheritance is from the iostream class library.

```
class istream: virtual public ios {...};
class ostream: virtual public ios {...};
class iostream: public istream, public ostream {...};
```

This is a special case; notice that the derived class, *iostream*, has two base classes, *istream* and *ostream*, which in turn share a common base class, *ios*. This presents a problem since we have to make sure that *iostream* only inherits one copy of the members of ios. The keyword *virtual* before the base class *ios* , in the class declaration of *istream* and *ostream*, does precisely that. It guarrantees that in *iostream* there is going to be only one copy of the members of *ios*.

IV.6. Persistent Objects.

A persistent object is one that exists beyond the execution of the piece of code that created it. To be able to do so, the object has to be stored in some kind of electronic or magnetic medium until it is needed again.

There are several problems to be addressed with persistent objects. For example, when an object is created, it gets the addresses of its member functions. If the object is stored for a while and then reloaded into memory, there is no guarantee that the addresses of its member functions are going to be the same ones.

Persistent objects must be able to be converted back and forth from their physical representation into a symbolic representation. The creation of a physical representation from a symbolic one is called *activation*. The creation of a symbolic representation from a physical one is called *passivation*. By means of passivation and activation, objects can be shared between different programs and perhaps different computer environments.

Persistent objects are the components of an object-oriented database.

V. THE GRAPHIC USER INTERFACE.

In this work, a GUI specifically for transportation scheduling specification has been developed in motif/C++ in the X windows environment. The motif toolkit was chosen because it has definitions for many objects that are useful for GUI's and it is available in the Computer Science Department computer. C++ was chosen because it is a popular object oriented language. Object-orientation has the following advantages: code easy to maintain, code extensibility and code reusability.

V.1. Geographical Data.

Geographical data are stored in databases in layers, each layer having information about different kinds of geographical data. They are very varied and complex, comprising information about natural resources, population, urban development, industrial development, waste disposal, weather and more. Only a fraction of all the information available in geographical databases is relevant for transportation scheduling problems, but it is still a large amount of data.

Some of the relevant data are:

Cities: the access to both geographical and transportation data will be made through an index based on the city where the data belongs. What is basically needed is the location of the city; however, additional information such as the state in which the city is located and the population are provided.

Paths: we call paths the connections between cities; they might be roads, highways, or airways. They serve to determine whether it is possible to go from one city to another and the distance between those cities.

Weather information: this is a variable and complex kind of data that is included only to show the many possiblities that exist in representing geographical data. The weather information will also be associated with a given city to make it easier.

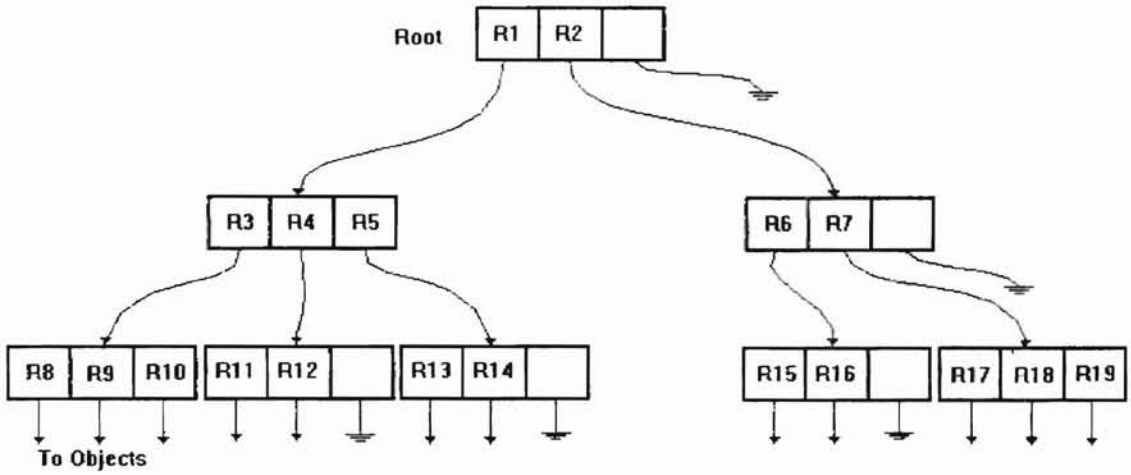
V.1.1. The R*-tree: a data structure for spatial data.

The geographical data are handled by means of an R*-tree[Beck90]. An R*-tree is a data structure for spatial data that has many desirable properties that allow it to give excellent performance handling very large spatial databases. The R*-tree is a variation of the R-tree introduced by Guttman (see [Gutt84]). The R-tree is a height-balanced tree similar to a B-tree with index records in its leaf nodes pointing to spatial objects.

As the B-trees, R-trees remain balanced while maintaining the flexibility of dinamically adjustable windows that deal with "dead space" on the database, like the quadtrees do. A second important characteristic of R-trees is the fact that, at the leaf level they store full and non-atomic spatial objects. This last feature provides a natural high level object oriented search [Falo87].

The leaf nodes of the R-tree contain entries of the form (object-pointer, rectangle) where object-pointer points to a spatial object and rectangle is the minimum bounding rectangle that contains the spatial object. Non-leaf nodes of the R-tree contain entries of the form (child-pointer, rectangle) where child-pointer points to a child node and rectangle is the

minimum bounding rectangle that contains all the rectangles which are entries in the child node.



An R-tree structure with M = 3

If M is the maximum number of entries that will fit in a node, let m <= M/2 be the minimum number of entries in a node. The R-tree satisfies:

(1) Every leaf node contains between m and M index records, inclusive, unless it is the root.

(2) Every non-leaf node has between m and M children, inclusive, unless it is the root.

(3) The root node has at least two children unless it is a leaf.

(4) All leaves are at the same level.

The desirable characteristics of the R*-tree mentioned above are the following (see [Beck90]):

(1) The area covered by a directory rectangle is minimized.

(2) The overlap between directory rectangles is minimized.

(3) The margin (perimeter) of directory rectangles is minimized.

(4) Storage utilization is optimized.

The data needed to specify transportation scheduling problems can be classified as geographical data and transportation scheduling data. The application should integrate them so they can be used later to produce a schedule for the problem being specified.

V.2. Transportation scheduling data.

Four types of transportation scheduling data are considered: conveyances, cargoes (shipments), sites and additional constraints.

Conveyances: they serve to move cargoes from one site to another. We will consider the following characteristics of conveyances:

Type: vehicle, truck, airplane, train, etc

Capacity: the amount of cargo carried, in pounds, kilograms or tons.

Volume: the volume of cargo carried, in cubic feet or cubic meters.

Speed: average speed of the conveyance.

Autonomy: maximum distance a conveyance can go without maintenance or refueling.

Allowable cargoes: some cargoes have special characteristics and need special accommodations in the conveyances. For example transporting troops is different from transporting packages; or some cargoes might need a freezer.

Cargoes/Shipments: refers to the items that are being carried in the conveyances.

Type: cargo, shipment, personnel.

Weight: total weight in pounds, kilograms, or tons.

Volume: total volume in cubic feet or cubic meters.

Destination: the final destination site of the cargo.

Location: the site at which the cargo was located originally.

Earliest arrival: the earliest time a cargo can get to a final destination.

Latest arrival: the latest time a cargo can get to its final destination.

Special handling: some cargoes such as frozen goods, perishable goods, and personnel require special handling.

Sites: the sites refer to the locations from/to which cargoes are picked up/delivered.

Type: airports, depots, train stations.

Location: it refers to the place in the country/state/city where the site is located.

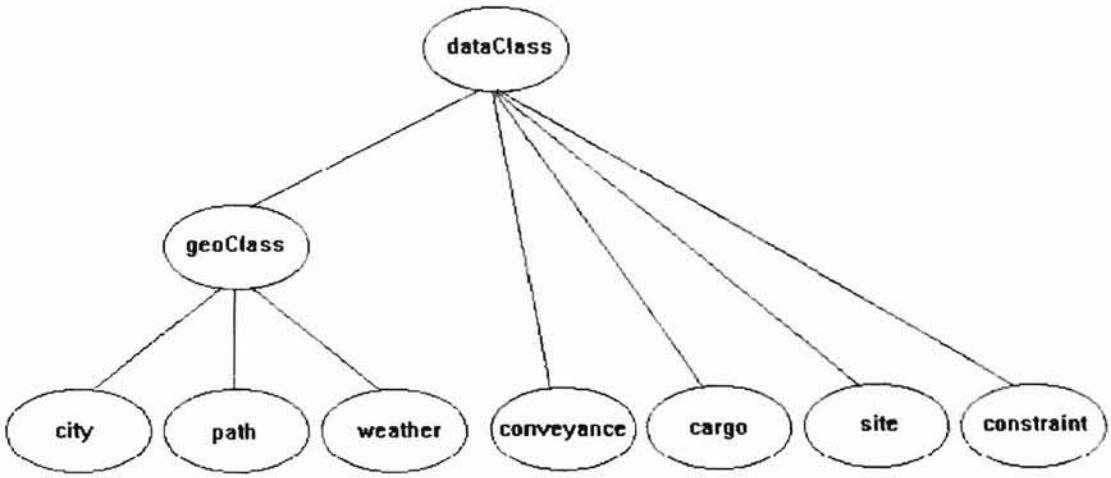
Constraints: even though the conveyances, sites and cargoes have self-imposed constraints, the user might want to include additional constraints that apply on the conveyances, cargoes, and/or sites that affect the scheduling. They comprise time, size, and weight constraints. Cost considerations are not included.

Time: any constraint that affects the duration of the activities to be scheduled.

Size: constraints on the volume, length, or height.

Weight: constraints on the weight of cargoes.

The following class hierarchy allows us to integrate the geographical and transportation scheduling data.



The Class geoClass is used as a superclass of all the objects that store geographical data the conveyance, cargo, site and constraint classes are used to store transportation scheduling data. The dataClass class is implemented as an abstract class to allow more flexibility in handling the derived classes by means of virtual functions.

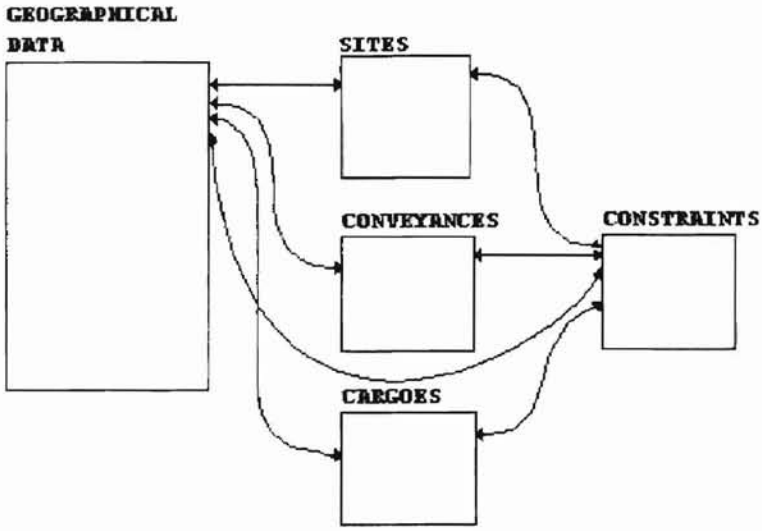
V.3. Queries that the application will allow:

The application will allow the user query the occurrences of the different objects specified in different combinations, such as:

- How many conveyances of the type "x" are there in location "y"?
- Where is vehicle "x"?
- Total weight of cargoes at location "x".
- Total weight of cargoes at location "x" with destination location "y".
- Where is cargo "x"?
- Constraints on cargo "x".
- Where is site "x"?
- Constraints on site "x".

V.4. Application data files.

The application will handle files for the geographical data, sites, conveyances, cargoes, and constraints. The figure below shows the way files are related to each other.



V.5. Evaluating the Application.

To evaluate the application, we will consider separately the evaluation of the GUI itself and the evaluation of the data handling. The evaluation of a GUI involves subjectivity since the parameters used cannot be measured quantitatively. The following parameters (see [Coll95] pag. 34-48) can be used as indicators of how good a GUI is.

Readability: since the system is aimed to serving users with certain special characteristics, the language used must be understandable for such users.

High degree of Interaction: the system should be user friendly and allow the users to correct their own mistakes in inputting the data.

Power of Expression: the system must do something that is relevant for its user and do it in an efficient way. In this case, the system must make it easier for the user to provide the data for the scheduling algorithm.

Modern GUI Characteristics.

- Icons represent data objects. Users express object interactions and commands by moving icons and dropping them on other icons.
- Icons can be asked to show their properties, or their contents, in windows.
- Windows contents are scrollable, and have menus and control icons.
- Commands are expressed almost entirely by pointing and clicking with the mouse.
- The environment is represented graphically, using images of familiar objects.

Object orientation: even though it is not necessary that a GUI be object oriented, object orientation adds desirable properties to it, such as code portability, reusability, and extensibility. In addition, encapsulation, inheritance, and message passing allow us to eliminate data redundancy and to produce robust code. The use of object orientation produces bigger and slower executable code but the cost paid is worth what we get in exchange.

To evaluate the data handling we will refer to the comparisons made in the past by other authors between the R*-tree and other spatial data structures. In [Beck90], the authors compared the R*-tree to the R-tree (linear split), the R-tree (quadratic split), the R-tree (Greene variation), and the GRID file for both rectangle and point access methods, concluding that the R*-tree gives the best performance among those data structures. In [Hoel92] the authors found R*-trees and R+-trees to have similar performance except for queries that required that the space be divided in disjoint subspaces, where the R+-tree performed better.

Since this is the first attempt to build a GUI specifically for transportation scheduling, it is not possible to compare the application as a whole with an existing application. However, the correctness of the data can be verified in the following way: create a matrix with all the data that is going to be loaded into the system, load the data into the system, perform queries and compare against the matrix. This will not prove the application to be correct, but it will provide evidence of the correctness of it.

VI. HOW THIS WORK IS RELATED TO OTHER RESEARCH AREAS.

The application of geographical information data to transportation scheduling problems is relatively new. GIS (Geographical Information Systems) have been in use for some time and have proved to be powerful tools for the compilation, management, and display of data associated with geographic space. However, application of GIS to transportation (GIS-T) has required the extension of basic functionality to include new features needed. For example, many state departments of natural resources have been using GIS applications for some time, but in 1991, even though many transportation agencies expressed interest in GIS-T technology, only one agency had a comprehensive, agency-wide application in place (see [Vond93])

D. R. Smith and E. A. Parra (see [Smit93]) the developers of KIDS (Kestrel Interactive Development System) have implemented a new way of approaching transportation scheduling, based on the application of correctness-preserving transformations to abstract specifications of scheduling problems. This approach has proved to yield very fast and accurate schedulers, and is in use by the U.S. Air Force. The GUI to be developed as this thesis will serve as an intelligent front-end for KIDS, in research currently under way both here at OSU and at the Air Force's Rome Laboratory.

The application of Object-Oriented Programming (OOP) to Graphic User Interfaces (GUI) is a modern technique that is being used more widely every day. The Open Software Foundation (OSF) has developed recommendations for a standard toolkit to develop GUIs called Motif. Motif applications are able to run in many platforms with no change or few changes in the code.

We were not able to find a GUI specifically for transportation scheduling either in bibliography or in the Internet.

VII. CONCLUSIONS AND RECOMMENDATIONS.

The implementation of the GUI has been affected by a number of problems related to the fact that Motif is not fully Object-Oriented but an Object-Oriented-like toolkit for X-windows implemented in a non Object-Oriented programming language (C language). In addition, the programs I developed and adopted can be still improved a great deal to guarantee a better performance of the Graphic User Interface. I will try to point out here some of the aspects that can be improved.

VII.1. The R-tree.*

In [Beck90] the authors recommend some values for the parameters M (maximum number of entries per node), k (number of groups used in the sorting of entries), and p (p best rated entries for reinsertion). I implemented the R*-tree keeping those values as pound-defined values (with the #define compiler directive) in a header file. It is the user's responsibility to choose appropriate values for those parameters.

My implementation of the nodes uses void pointers to point to the children of a given node since they might point to other nodes or to the objects the R*-tree stores. This causes a big problem since the destructor will not know what kind of objects the void pointers are pointing to. My solution to this was to leave to the programmer using the R*-tree the responsibility of coding the destructor for the R*-tree, since the programmer knows what kind of objects are being stored in the R*-tree and, therefore, knows how to deallocate the memory they occupy.

The spatial indexing by definition is not unique; many areas of geographical databases overlap with each other. The approach used to input data through the R*-tree is

somewhat naive and will not avoid, for example, that two different cities be inserted in the same geographical coordinates.

VII.2. Motif Toolkit.

Motif objects respond to events (keyboard input, mouse movement, mouse button clicks, etc.) by calling callback functions. This functions have the form:
void function(Widget w, XtPointer client_data, XtPointer call_data), where w is the object that called the function, client_data is a pointer to a structure containing information about the event that originated the callback, and call_data is a pointer to the parameters needed by the callback function. This puts enormous limitations to the capacity to communicate with the application and seems to encourage the use of global variables, which are used very frequently in the examples provided in [Hell94a] and [Hell94b]. The only way to get around this problem was to create a big and ugly data structure with all the information needed in the different callback functions and pass a reference to it to the callbacks that need it.

In regard to the lack of Object-Orientation of Motif, another problem I detected is related with the application mainloop. In a very simplified way, what an X-window application does is keep running in an infinite loop waiting for external events to happen; when the program terminates it goes out of the main loop by means of an call to the *exit()* function, which does not call the destructors of the objects created in the *main()* function. So when the program exits, the programmer is responsible for destroying the objects created in *main()*.

VII.3. The Data Objects.

The set of data objects I provide are very simple and they do not intend to provide for all the possibilities of data element combinations needed to model a transportation scheduling problems. They are only used to give and idea of what kind of data can be handled by the application.

VII.4. The File System.

The file system is not fully integrated with the application and most of the file errors will cause the program to exit instead of handling them. Additionally, the file system does not provide much for system crashes and data recovery; a hash table can be rebuilt from an existing file but it will not guarantee data integrity if the system crashed in the middle of a multiple file update, in which some files were updated and some were not.

I used hashtables with unique keys for the indexing of the files. However, in some cases there might exist the need for duplication in the keys. In addition, only keys of type *unsigned long* (or other subtype of unsigned long) are allowed by the system.

Some of the problems here depicted can be solved by using features of newer C++ compilers such as the use of templates and try-and-catch blocks, which are not supported in the C++ compiler available in the Computer Science Department. However, nothing can be done about Motif not being Object-Oriented.

I have developed a robust set of programs that is suitable for being integrated into a Graphic User Interface for Transportation Scheduling Problem Specification. They present some shortcomings that I tried to point out here, but they are the basis on top of which a more sophisticated system can be built. An example GUI and the makefile to

compile it are provided in the appendices to show how the set of programs interact with each other.

REFERENCES

[Batt92] Batty, Peter: "Exploiting Relational Database Technology in a GIS." Computers & Geosciences. Vol 18 No. 4 pp. 453-462. 1992.

[Beck90] Beckman, Norbert, H.P. Kriegel, R. Schneider, B. Seeger: "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles". Proceedings of ACM SIGMOD Conference. 1990.

[Bodi81] Bodin, Laurence, B. Golden, A. Assad, and M. Ball: "The State of the Art in the Routing and Scheduling of Vehicles and Crews". U.S. Department of Transportation. Urban Mass Transportation Administration. Washington D.C. 1981.

[Booc94] Booch, Grady: "Object-Oriented Analysis and Design." Benjamin/Cummings Publishing Company. 1994.

[Budd91] Budd, Timothy: "Object-Oriented Programming." Addison -Wesley Publishing Company. 1991.

[Burro92] Burrough, P.A.: "Are GIS Data Structures Too Simple Minded?". Computers & Geosciences. Vol 18 No. 4 pp. 395-400. 1992

[Coll95] Collins, Dave: "Designing Object-Oriented User Interfaces". Benjamin/Cummings Publishing Company , Inc. 1995.

[Date91] Date, C. J.: "An Introduction to Database Systems". Addison-Wesley Publishing Company. 5th Edition, 1991.

[Dobs93] Dobson, Jerome: "Commentary: A conceptual framework for Integrating Remote Sensing, GIS, and Geography." Photogrammetric Engineering and Remote Sensing. Vol 59, No. 10, October 1993. pp. 1491-97

[Falo87] Faloutsos, Christos, T. Sellis, N. Roussopoulos: "Analysis of Object Oriented Spatial Access Methods." Proceedings of ACM. 1987.

[Fink74] Finkel, R. A. and J. L. Bentley: "Quad Trees a Data Structure for retrieval on Composite Keys" Acta Informatica, Vol 4, pp. 1-9, 1974..

[Foth94] Fotheringham, Stewart and P. Rogerson: "Spatial Anlysis and GIS" Taylor & Francis Ltd. 1994. Edited by Stewart Fotherigham and Peter Rogerson.

[Good92] Goodchild, Michael: "Geographical Data Modeling." Computers & Geosciences. Vol 18 No. 4, pp. 401-408. 1992

[Günt94] Günther, Oliver and J. Lamberts: "Object Oriented Techniques for the Management of Geographic and Environmental Data" The Computer Journal, Vol. 37, No. 1, pp 16-25, 1994

[Gutt84] Guttman, Antonin: "R-trees: A Dynamic Index Structure for Spatial Searching." Proceedings of ACM SIGMOD Conference. 1984.

[Hell94a] Heller, Dan, P.M. Ferguson: "Motif Programmimg Manual." O'Reilly and Associates, Inc. Volume 6A. 1994.

[Hell94b] Heller, Dan, P.M. Ferguson: "Motif Reference Manual." O'Reilly and Associates, Inc. Volumen 6B. 1994.

[Hoel92] Hoel, Erick and H. Samet: "A Qualitative Comparison Study of Data Structures for Large Line Segment Databases." Proceedings of ACM SIGMOD Conference. 1992.

[Jone94] Jones, Christopher, D. Kidner, and M. Ware: "The Implicit Triangulated Irregular Network and Multiscale Spatial Databases" The Computer Journal, Vol. 37, No. 1, pp. 37-57, 1994.

[Ladd94] Ladd, Scott: "C++ Components and Algorithms" M & T Books. 2nd. Edition 1994.

[Lock93] Lockfeld, Frank: "GIS Links County to Other Transportation Planners." Public Works. pp. 43-44 May 1993.

[Masse91] Masser, Ian and Michael Blakemore: "Handling Geographical Information." Longman Scientific & Technical. 1991.

[Mayf95] Mayfield, Blayne. Notes for Object-Oriented Programming and C++ course. (All the OOP examples were taken from the notes). Computer Science Department, Oklahoma State University, 1995.

[Nye92a] Nye, Adrian: "Xlib Programming Manual" O'Reilly and Associates, Inc. Volume One. Third Edition. July 1992. Edited by Adrian Nye.

[Nye92a] Nye, Adrian: "Xlib Reference Manual" O'Reilly and Associates, Inc. Volume Two. Third Edition. July 1992. Edited by Adrian Nye.

[O'Re91] O'Reilly and Associates. Staff of: "Xtoolkit Intrinsics Reference Manual" O'Reilly and Associates, Inc. Volume 5. Second Edition. June 1991.

[Raper92] Raper, Jonathan, D. Maguire: "Design Models and Functionality in GIS." Computers & Geosciences Vol 18 No. 4 pp. 387-394. 1992

[Rifai93] Rifai, H.S., L.A. Hendrick, K. Kilborn, and P.B. Bedient: "A Geographic Information System (GIS) User Interface for Delineating Wellhead Protection Areas." Ground Water. Vol 31 No. 3 pp. 480-488. 1993.

[Sell87] Sellis, Timos, N. Roussopoulos and C. Faloutsos: "The R+-tree: a Dynamic Index for Multi-Dimensional Objects." Proceedings of the of the 13th VLDB Conference, Brighton, 1987. Pages 507-518.

[Smit93] Smith, Douglas R. and E. A. Parra: "Transformational Approach to Transportation Scheduling." Proceedings of the 8th Knowledge-Based Software Engineeing Conference. 1993.

[Stev94] Stevens, Al: "C++ Database Development" MIS Press 2nd. Edition 1994.

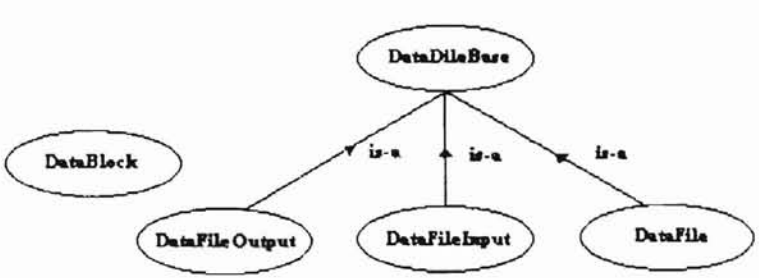
[Vond93] Vonderohe, A.P., L. Travis, R.L. Smith, and V. Tsai: "Adaptation of Geographic Information Systems for Transportation". National Cooperative Research Program (NCHRP) Report 359. Transportation Research Board. National Research Council. National Academy Press. Washington D.C. 1993.

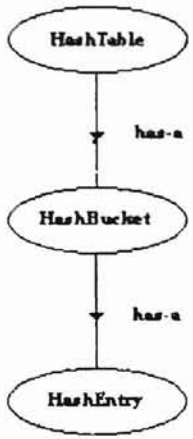
[Wins92] Winston, Patrick H.: "Artificial Intelligence". Addison -Wesley Publishing Company. 3rd. Edition. 1992.

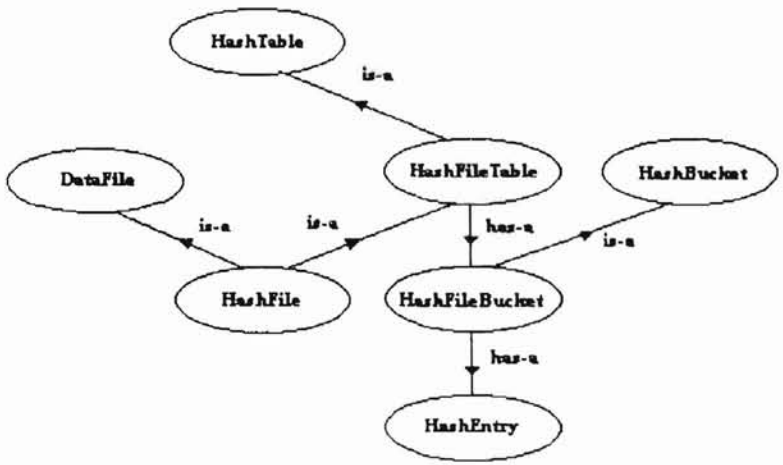
APPENDIX A

CLASS HIERARCHIES

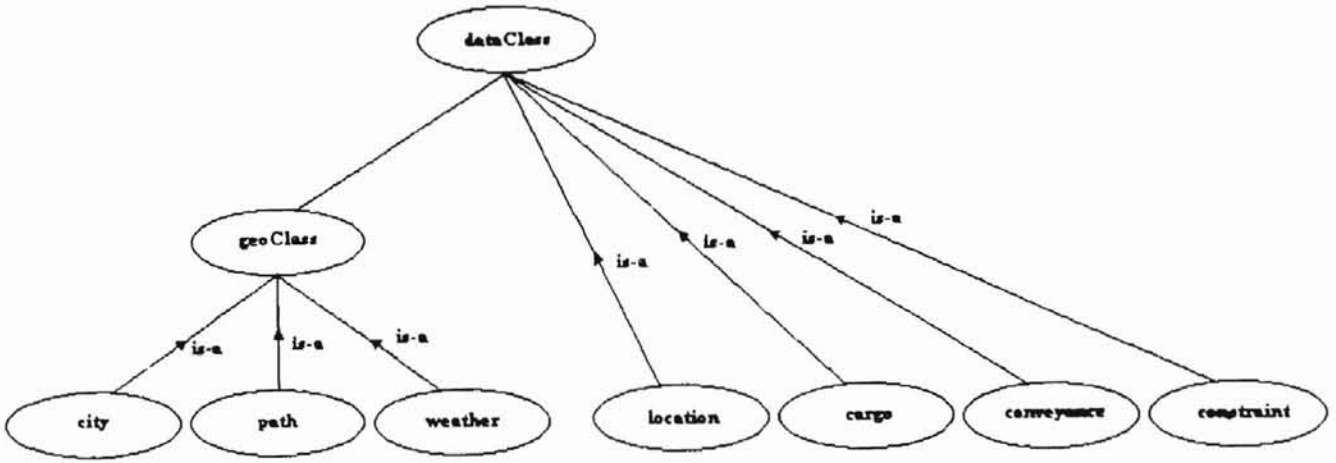
CLASS HIERARCHIES

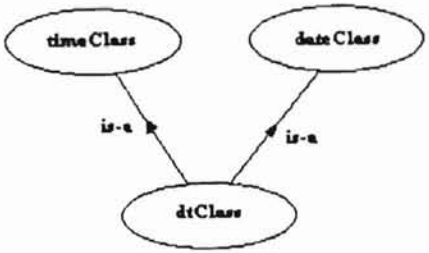


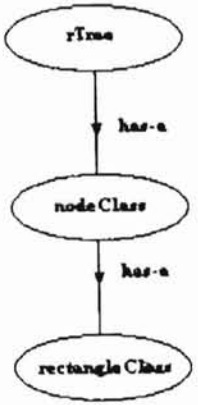




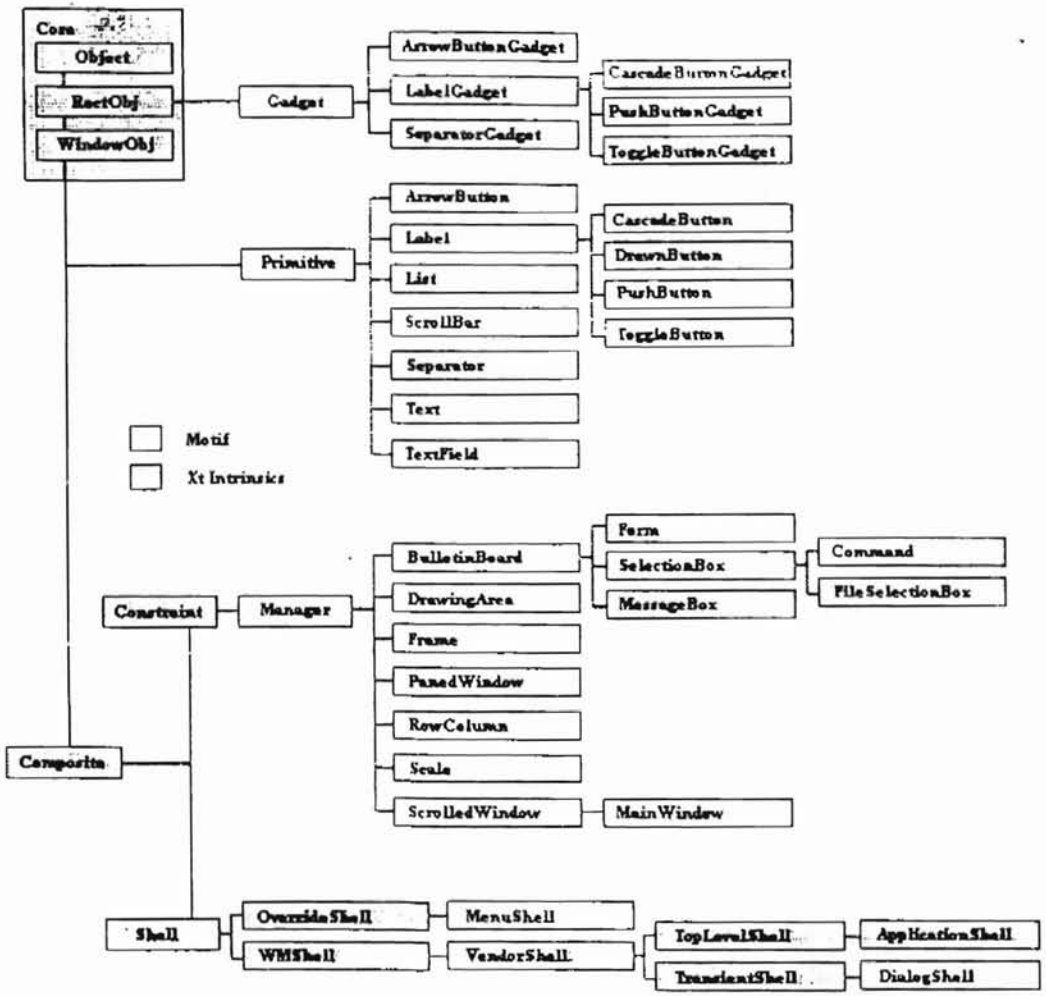
CLASS HIERARCHIES (CONT.)







CLASS HIERARCHY OF THE MOTIF WIDGET SET.



APPENDIX B

PROGRAM LISTINGS

```
// Program Name:  TSPS.H
// An Object Oriented Graphic User Interface for Transportation Scheduling Problem Specification
// Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Last Modified: 09/14/95

#include <Xm/CascadeBG.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/Form.h>
#include <Xm/FileSB.h>
#include <Xm/Frame.h>
#include <Xm/MessageB.h>
#include <Xm/PanedW.h>
#include <Xm/PushBG.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/ScrolledW.h>
#include <Xm/SeparatoG.h>
#include <Xm/Text.h>
#include <Xm/TextF.h>
#include <Xm/ToggleB.h>
#include <Xm/ToggleBG.h>
#include <stdio.h>
#include <stdlib.h>
#include "file.h"
#include "hashfile.h"
#include "rpt.h"
#include "str.h"

/* type definition for bitmap information */
typedef struct {
  stringClass filename;
  Pixmap pixmap;
  unsigned int width, height;
  int x_hot, y_hot;
  } bitmapInfo;

/* type definition for application information */
typedef struct {
  bitmapInfo bitmap;
  stringClass mainFilename,
        baseFilename;
  XGCValues gcv;
  GC gc;
  Widget tl,
      mb,
      da,
      dl,
      rc,
            tx;
  HashFile *cityhf;
  long cityCount;
  int option, exitCode;
  stringClass dialogMessage, windowMessage;
  Signature signature;
  rectangleClass rectangle;
  String cityInputTran;
  String cityEditTran;
```

```
  String cityNone;
  rTree *rtree;
  } appInfo;

// structure used to write city objects to disk
struct cityWriteStru {
  HashFile *file;
  rTree *rtree;
  long cityCode;
  rectangleClass rectangle;
  WidgetList textFields;
  int numOfTF;
  };

/* type definition for menu items */
typedef struct _menu_item {
   char       *label;        /* the label for the item */
   WidgetClass *widgetClass;  /* pushbutton, label, separator... */
   char       mnemonic;      /* mnemonic; NULL if none */
   char       *accelerator;  /* accelerator; NULL if none */
   char       *accel_text;   /* to be converted to compound string */
   void       (*callback)(Widget, XtPointer, XtPointer);
                        /* routine to call; NULL if none */
   XtPointer    callback_data; /* client_data for callback() */
   struct _menu_item *subitems; /* pullright menu items, if not NULL */
} menuItems;

  /* Pulldown menu items */
menuItems geoSubItems0[] = {
  { "City", &xmPushButtonGadgetClass, 'C', NULL, NULL,
     NULL, (XtPointer) 0, NULL},
  { "Path", &xmPushButtonGadgetClass, 'P', NULL, NULL,
     NULL, (XtPointer) 1, NULL},
  { "Weather Info", &xmPushButtonGadgetClass, 'W', NULL, NULL,
     NULL, (XtPointer) 2, NULL},
  NULL,
  };

menuItems geoSubItems1[] = {
  { "City", &xmPushButtonGadgetClass, 'C', NULL, NULL,
     NULL, (XtPointer) 3, NULL},
  { "Path", &xmPushButtonGadgetClass, 'P', NULL, NULL,
     NULL, (XtPointer) 4, NULL},
  { "Weather Info", &xmPushButtonGadgetClass, 'W', NULL, NULL,
     NULL, (XtPointer) 5, NULL},
  NULL,
  };

menuItems geoSubItems2[] = {
  { "City", &xmPushButtonGadgetClass, 'C', NULL, NULL,
     NULL, (XtPointer) 6, NULL},
  { "Path", &xmPushButtonGadgetClass, 'P', NULL, NULL,
     NULL, (XtPointer) 7, NULL},
  { "Weather Info", &xmPushButtonGadgetClass, 'W', NULL, NULL,
     NULL, (XtPointer) 8, NULL},
  NULL,
  };

menuItems transpSubItems0[] = {
  { "Locations", &xmPushButtonGadgetClass, 'L', NULL, NULL,
```

```
    NULL, NULL, NULL},
  { "Conveyances", &xmPushButtonGadgetClass, 'C', NULL, NULL,
    NULL, NULL, NULL},
  { "Cargoes", &xmPushButtonGadgetClass, 'a', NULL, NULL,
    NULL, NULL, NULL},
  { "Constraints", &xmPushButtonGadgetClass, 't', NULL, NULL,
    NULL, NULL, NULL},
  NULL,
  };

menuItems transpSubItems1[] = {
  { "Locations", &xmPushButtonGadgetClass, 'L', NULL, NULL,
    NULL, NULL, NULL},
  { "Convevances", &xmPushButtonGadgetClass, 'C', NULL, NULL,
    NULL, NULL, NULL},
  { "Cargoes", &xmPushButtonGadgetClass, 'a', NULL, NULL,
    NULL, NULL, NULL},
  { "Constraints", &xmPushButtonGadgetClass, 't', NULL, NULL,
    NULL, NULL, NULL},
  NULL,
  };

menuItems transpSubItems2[] = {
  { "Locations", &xmPushButtonGadgetClass, 'L', NULL, NULL,
    NULL, NULL, NULL},
  { "Conveyances", &xmPushButtonGadgetClass, 'C', NULL, NULL,
    NULL, NULL, NULL},
  { "Cargoes", &xmPushButtonGadgetClass, 'a', NULL, NULL,
    NULL, NULL, NULL},
  { "Constraints", &xmPushButtonGadgetClass, 't', NULL, NULL,
    NULL, NULL, NULL},
  NULL,
  };

menuItems querySubItems0[] = {
  { "City", &xmPushButtonGadgetClass, 'C', NULL, NULL,
    NULL, NULL, NULL},
  { "Path", &xmPushButtonGadgetClass, 'P', NULL, NULL,
    NULL, NULL, NULL},
  { "Weather Info", &xmPushButtonGadgetClass, 'W', NULL, NULL,
    NULL, NULL, NULL}
  NULL,
  };

menuItems querySubItems1[] = {
  { "Locations", &xmPushButtonGadgetClass, 'L', NULL, NULL,
    NULL, NULL, NULL},
  { "Convevances", &xmPushButtonGadgetClass, 'C', NULL, NULL,
    NULL, NULL, NULL},
  { "Cargoes", &xmPushButtonGadgetClass, 'a', NULL, NULL,
    NULL, NULL, NULL},
  { "Constraints", &xmPushButtonGadgetClass, 't', NULL, NULL,
    NULL, NULL, NULL},
  NULL,
  };

menuItems fileItems[] = {
  { "New", &xmPushButtonGadgetClass, 'N', NULL, NULL,
        NULL, NULL, NULL},
  { "Open...", &xmPushButtonGadgetClass, 'O', NULL, NULL,
```

```
        NULL, NULL, NULL},
  { "Save", &xmPushButtonGadgetClass, 'S', NULL, NULL,
        NULL, NULL, NULL},
  { "Save As", &xmPushButtonGadgetClass, 'A', NULL, NULL,
        NULL, NULL, NULL},
  { "", &xmSeparatorGadgetClass, NULL, NULL, NULL,
        NULL, NULL, NULL},
  { "Exit", &xmPushButtonGadgetClass, 'x', "Ctrl<Key>C", "Ctrl+C",
        NULL, NULL, NULL},
  NULL,
  };

menuItems editItems[] = {
  { "Copy", &xmPushButtonGadgetClass, 'C', NULL, NULL,
    NULL, NULL, NULL},
  { "Paste", &xmPushButtonGadgetClass, 'P', NULL, NULL,
    NULL, NULL, NULL},
  NULL,
  };

menuItems geoItems[] = {
  { "Add", &xmCascadeButtonGadgetClass, 'A', NULL, NULL,
    NULL, NULL, geoSubItems0},
  { "Delete", &xmCascadeButtonGadgetClass, 'D', NULL, NULL,
    NULL, NULL, geoSubItems1},
  { "Edit", &xmCascadeButtonGadgetClass, 'E', NULL, NULL,
    NULL, NULL, geoSubItems2},
  NULL,
  };

menuItems transpItems[] = {
  { "Add", &xmCascadeButtonGadgetClass, 'A', NULL, NULL,
    NULL, NULL, transpSubItems0},
  { "Delete", &xmCascadeButtonGadgetClass, 'D', NULL, NULL,
    NULL, NULL, transpSubItems1},
  { "Edit", &xmCascadeButtonGadgetClass, 'E', NULL, NULL,
    NULL, NULL, transpSubItems2},
  NULL,
  };

menuItems queryItems[] = {
  { "Geographical Data", &xmCascadeButtonGadgetClass, 'G', NULL, NULL,
    NULL, NULL, querySubItems0},
  { "Transportation Data", &xmCascadeButtonGadgetClass, 'T', NULL, NULL,
    NULL, NULL, querySubItems1},
  NULL,
  };

menuItems helpItems[] = {
  { "Index", &xmPushButtonGadgetClass, 'I', NULL, NULL,
    NULL, NULL, NULL},
  { "Help on Help", &xmPushButtonGadgetClass, 'H', NULL, NULL,
    NULL, NULL, NULL},
  { "Geo. Data", &xmPushButtonGadgetClass, 'G', NULL, NULL,
    NULL, NULL, NULL}
  { "Transp. Data", &xmPushButtonGadgetClass, 'T', NULL, NULL,
    NULL, NULL, NULL},
  NULL,
  };
```

```
char *buttonlabels[] = { "One", "Two", "Three", "Four", "Five",
                "Six", "Seven", "Eight", "Nine", "Ten"};

char *citylabels[] = { "City:", "State:", "Population:"};

char *pathlabels[] = { "Path Name", "Path Type:",
                "From City:", "To City:", "Distance:" };
```

```
// Program Name:  TSPS.C
// An Object Oriented Graphic User Interface for Transportation Scheduling Problem Specification
// Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Last Modified: 09/14/95

// Include header file
#include "tsps.h"

void fileCB(Widget, XtPointer, XtPointer);
void loadFile(Widget, XtPointer, XtPointer);
void saveFile(Widget, XtPointer, XtPointer);
void redraw(Widget, XtPointer, XtPointer);
void quit(Widget, XtPointer, XtPointer);
void geoDataIO(Widget, XtPointer, XtPointer);
void grabRectangle(Widget, XEvent *, String *, int *);
void cityWrite(Widget, XtPointer, XtPointer);
void cityInput(appInfo*);

Widget BuildPulldownMenu(Widget, char*, char, menuItems*);

void main(int argc, char *argv[])
{
  Widget topLevel,mainWindow,drawingArea,menuBar,
      button,rowColumn,scrolledWindow, widget, frame,
          msglabel, text, dialog, cityDialog;
  XtAppContext app;
  bitmapInfo binfo;
  XmString str;
  int i;
  appInfo appinfo;
  XtActionsRec cityActions;

  // translations for city input
  appinfo.cityInputTran=
     "<Btn1Down>:   grabRectangle(down) ManagerGadgetArm()  \n\
     <Btn1Up>:    grabRectangle(up)  ManagerGadgetActivate()  \n\
     <Btn1Motion>: grabRectangle(motion) ManagerGadgetButtonMotion()";
  // translations for city edit
  appinfo.cityEditTran=
     "<Btn1Down>:   grabPoint(down) ManagerGadgetArm()  \n\
     <Btn1Up>: grabPoint(up) ManagerGadgetButtonMotion()";
  // used to deactivate city translations
  appinfo.cityNone =
     "<Btn1Down>:   ManagerGadgetArm()  \n\
     <Btn1Up>:     ManagerGadgetActivate()  \n\
     <Btn1Motion>: ManagerGadgetButtonMotion()";
  // Add signature
  appinfo.signature = 0x44440000;
  // Client data for callback for file menu
  fileItems[1].callback = (XtCallbackProc) fileCB;
  fileItems[1].callback_data = (XtPointer) &appinfo;

  fileItems[2].callback = (XtCallbackProc) saveFile;
  fileItems[2].callback_data = (XtPointer) &appinfo;

  fileItems[5].callback = (XtCallbackProc) quit;
  fileItems[5].callback_data = (XtPointer) &appinfo;
  // Geo Data callback
```

```
geoSubItems0[0].callback = (XtCallbackProc) geoDataIO;
geoSubItems0[1].callback = (XtCallbackProc) geoDataIO;
geoSubItems0[2].callback = (XtCallbackProc) geoDataIO;
geoSubItems1[0].callback = (XtCallbackProc) geoDataIO;
geoSubItems1[1].callback = (XtCallbackProc) geoDataIO;
geoSubItems1[2].callback = (XtCallbackProc) geoDataIO;
geoSubItems2[0].callback = (XtCallbackProc) geoDataIO;
geoSubItems2[1].callback = (XtCallbackProc) geoDataIO;
geoSubItems2[2].callback = (XtCallbackProc) geoDataIO;

// Create the top level widget
topLevel = XtVaAppInitialize (&app, "tsps", NULL, 0,
   (Cardinal *)&argc, argv, NULL, NULL);

// Add actions
cityActions.string = "grabRectangle";
cityActions.proc = (XtActionProc) grabRectangle;
XtAppAddActions (app, &cityActions, 1);


// Use a form as main window
mainWindow = XtVaCreateManagedWidget ("mainWindow",
   xmFormWidgetClass, topLevel, NULL);

// Create the menu system
menuBar = XmVaCreateSimpleMenuBar (mainWindow, "menuBar",
                                   XmNtopAttachment, XmATTACH_FORM,
                XmNleftAttachment, XmATTACH_FORM,
                XmNrightAttachment, XmATTACH_FORM,
                NULL);

BuildPulldownMenu(menuBar, "File", 'F', fileItems);
BuildPulldownMenu(menuBar, "Edit", 'E', editItems);
BuildPulldownMenu(menuBar, "Geo. Data", 'G', geoItems);
BuildPulldownMenu(menuBar, "Transp. Data", 'T', transpItems);
BuildPulldownMenu(menuBar, "Query", 'Q', queryItems);
widget = BuildPulldownMenu(menuBar, "Help", 'H', helpItems);

// Tell the menuBar which button is the help menu
XtVaSetValues (menuBar, XmNmenuHelpWidget, widget, NULL);

XtManageChild(menuBar);

// Create a frame for the tool box
frame = XtVaCreateManagedWidget ("frame",
   xmFrameWidgetClass, mainWindow,
   XmNshadowType, XmSHADOW_IN,
   XmNtopAttachment,  XmATTACH_WIDGET,
   XmNleftAttachment,  XmATTACH_FORM,
   XmNbottomAttachment,  XmATTACH_FORM,
   XmNtopWidget, menuBar,
   NULL);

// Create a 2-column row-column to hold the toolbox
rowColumn = XtVaCreateWidget ("rowColumn",
   xmRowColumnWidgetClass, frame,
   XmNnumColumns,     2,
   XmNpacking,        XmPACK_COLUMN,
   NULL);
```

```
// Create push buttons for the toolbox.
for (i = 0; i < XtNumber(buttonlabels); i++) {
   str = XmStringCreateSimple(buttonlabels[i]);
   // Create a text pushbutton
   button = XtVaCreateManagedWidget (buttonlabels[i],
      xmPushButtonWidgetClass, rowColumn,
      XmNlabelString, str,
      NULL);
   XmStringFree(str);
   // Add callback here
};

XtManageChild (rowColumn);

// Create a scrolled window to hold the drawing area
scrolledWindow = XtVaCreateManagedWidget ("scrolledWindow",
    xmScrolledWindowWidgetClass, mainWindow,
    XmNwidth,              600,
    XmNheight,             400,
    XmNscrollingPolicy,      XmAUTOMATIC,
    XmNscrollBarDisplayPolicy, XmAS_NEEDED,
    XmNbottomAttachment,     XmATTACH_FORM,
    XmNbottomOffset,         40,
    XmNtopAttachment,        XmATTACH_WIDGET,
    XmNleftAttachment,       XmATTACH_WIDGET,
    XmNleftWidget,           frame,
    XmNtopWidget,            menuBar
    XmNrightAttachment,      XmATTACH_FORM,
    NULL);

// Create the drawing area
drawingArea = XtVaCreateManagedWidget ("drawingArea",
    xmDrawingAreaWidgetClass, scrolledWindow,
    XmNunitType,    Xm1000TH_INCHES,
    XmNwidth,     8000, // 8 inches
    XmNheight,    6000, // 6 inches
    XmNresizePolicy, XmNONE,  // remain this a fixed size
    NULL);

// Make drawing area the work window of the scrolled window
XtVaSetValues(scrolledWindow, XmNworkWindow, drawingArea, NULL);

// Create a label for the message textfield
msglabel = XtVaCreateManagedWidget("Messages:", xmLabelGadgetClass,
          mainWindow,
          XmNbottomAttachment,     XmATTACH_FORM,
          XmNbottomOffset,         15,
    XmNleftAttachment,       XmATTACH_WIDGET,
    XmNleftWidget,           frame,
          NULL);

// Create a text field widget to display messages
text = XtVaCreateManagedWidget("text", xmTextFieldWidgetClass,
          mainWindow,
    XmNcolumns,                80,
    XmNmaxLength,              256,
    XmNeditable,               False,
    XmNbottomAttachment,     XmATTACH_FORM,
    XmNleftAttachment,       XmATTACH_WIDGET,
    XmNleftWidget,           msglabel,
```

```
            XmNbottomOffset,        5,
            NULL);


  // Realize the top level widget (and its managed children
  XtRealizeWidget (topLevel);

  //Create a graphics context for the drawing area.
   appinfo.gcv.foreground = BlackPixelOfScreen(XtScreen(drawingArea));
   appinfo.gcv.background = WhitePixelOfScreen(XtScreen(drawingArea));
   appinfo.gcv.function = GXcopyInverted;
   appinfo.gc = XCreateGC(XtDisplay (drawingArea), XtWindow(drawingArea),
            GCForeground|GCFunction, &appinfo.gcv );

  // Put the application info in the user data of the drawing area
  XtVaSetValues(drawingArea,
                XmNuserData, &appinfo,
            XmNunitType, XmPIXELS,
                    NULL);

  // Put information in the appInfo structure
  appinfo.tl = topLevel;
  appinfo.mb = menuBar;
  appinfo.da = drawingArea;
  appinfo.rc = rowColumn;
  appinfo.tx = text;

  // The main loop
  XtAppMainLoop (app);
}

// taken from Motif Programming manual by Dan Heller & Paula Ferguson
// O'Reilly & Associates, Inc. 1994 pp 547-548
Widget
BuildPulldownMenu(Widget parent, char *menu_title, char menu_mnemonic,
            menuItems *items)
{
   Widget PullDown, cascade, widget;
   int i, n=0;
   XmString str;

   PullDown = XmCreatePulldownMenu (parent, "_pulldown", NULL, 0);

   str = XmStringCreateSimple (menu_title);
   cascade = XtVaCreateManagedWidget (menu_title,
      xmCascadeButtonGadgetClass, parent,
      XmNsubMenuId,   PullDown,
      XmNlabelString, str,
      XmNmnemonic,    menu_mnemonic,
      NULL);
   XmStringFree (str);

   /* Now add the menu items */
   for (i = 0; items[i].label != NULL; i++) {
      /* If subitems exist, create the pull-right menu by calling this
       * function recursively.  Since the function returns a cascade
       * button, the widget returned is used..
       */
      if (items[i].subitems)
         widget = BuildPulldownMenu (PullDown, items[i].label,
```

```
            items[i].mnemonic, items[i].subitems);
        else
            widget = XtVaCreateManagedWidget (items[i].label,
                *items[i].widgetClass, PullDown,NULL);
        /* Whether the item is a real item or a cascade button with a
         * menu, it can still have a mnemonic.
         */
        if (items[i].mnemonic)
            XtVaSetValues (widget, XmNmnemonic, items[i].mnemonic, NULL);
        /* any item can have an accelerator, except cascade menus. But,
         * we don't worry about that; we know better in our declarations.
         */
        if (items[i].accelerator) {
            str = XmStringCreateSimple (items[i].accel_text);
            XtVaSetValues (widget,
                XmNaccelerator, items[i].accelerator,
                XmNacceleratorText, str,
                NULL);
            XmStringFree (str);
        }
        if (items[i].callback)
            XtAddCallback(widget,
                (items[i].widgetClass == &xmToggleButtonWidgetClass ||
                items[i].widgetClass == &xmToggleButtonGadgetClass) ?
                    XmNvalueChangedCallback : /* ToggleButton class */
                    XmNactivateCallback,      /* PushButton class */
                (XtCallbackProc)items[i].callback, items[i].callback_data);
    }
    return cascade;
}

void
fileCB(Widget widget, XtPointer client_data, XtPointer call_data)
{
    static Widget dialog;
    appInfo *ai = (appInfo *) client_data;
    int n=0;
    XmString dirMask;

    if (!dialog) {
        XmString title = XmStringCreateSimple("File Selection");
        dirMask = XmStringCreateSimple("*.mn");
        dialog = XmCreateFileSelectionDialog (ai->tl, "dialog", NULL, 0);
        XtVaSetValues(dialog, XmNdialogTitle, title, NULL);
        XmFileSelectionDoSearch(dialog, dirMask);
        XtAddCallback (dialog, XmNokCallback, loadFile, client_data);
        XtAddCallback (dialog, XmNcancelCallback,
                (XtCallbackProc)XtUnmanageChild, NULL);
        XmStringFree(dirMask);
        XmStringFree(title);
        }
    XtManageChild (dialog);
    XtPopup (XtParent (dialog), XtGrabNone);
}

void loadFile(Widget widget, XtPointer client_data, XtPointer call_data) {
    char *file = NULL, *buf, bfname[128], bmname[128];
    XmFileSelectionBoxCallbackStruct *cbs =
        (XmFileSelectionBoxCallbackStruct *) call_data;
    appInfo *ai = (appInfo *) client_data;
```

```
XtVaSetValues(ai->tx,
        XmNvalue, "Loading Information...",
        NULL);
XtUnmanageChild(widget);
if (!XmStringGetLtoR (cbs->value, XmSTRING_DEFAULT_CHARSET, &file)) {
// display an error internal error reading compund string
// clear application info
  XtVaSetValues(ai->tx,
                XmNvalue, "Internal error, please try again.",
                NULL);
  return;
  }
ai->mainFilename = (stringClass) file;
DataFileInput input(ai->mainFilename);
XtFree (file); // free allocated data from XmStringGetLtoR()

DataBlock db = input.Read();
if (!db.IsNull() && db.GetSig() == ai->signature) {
  buf = (char *)(void *) db;
  istrstream stream(buf, db.GetSize());
  stream >> bfname >> bmname >> ai->cityCount;
  ai->baseFilename = (stringClass) bfname;
  ai->bitmap.filename = (stringClass) bmname;
  // Load pixmap.
  if (XReadBitmapFile(XtDisplay(ai->da), XtWindow (ai->da),
      ai->bitmap.filename, &ai->bitmap.width,
      &ai->bitmap.height, &ai->bitmap.pixmap,
      &ai->bitmap.x_hot, &ai->bitmap.y_hot) != BitmapSuccess) {
    // display error message
    // clear application info
    XtVaSetValues(ai->tx, XmNvalue, "Error loading bitmap.", NULL);
    return;
    }

  // Make drawing area same size as pixmap
  XtVaSetValues(ai->da, XmNwidth, ai->bitmap.width,
                        XmNheight, ai->bitmap.height, NULL);

  // Add Callback to redraw the pixmap on exposure events
  XtAddCallback(ai->da, XmNexposeCallback, redraw, &ai->bitmap);

  // Different treatment depending on whether the file exists
  if (fileExists(ai->baseFilename+"city"+".HFD"))
    // if it exists open it
    ai->cityhf = new HashFile(ai->baseFilename+"city");
  else
    //create it
    ai->cityhf = new HashFile(13, ai->baseFilename+"city");
  // Load the rtree
  ai->rtree = new rTree;
  ai->cityhf->Rewind();
  DataBlock rdb =  ai->cityhf->Read();
  while (!rdb.IsNull()) {
    city city1 = city(rdb);
    long *ptr = new long;
    *ptr = city1.getKey();
    (*ai->rtree) << entryClass(city1.getRectangle(), ptr);
    rdb = ai->cityhf->Read();
    }
```

```
    }
  else {
    XtVaSetValues(ai->tx, XmNvalue, "Error in Input File.", NULL);
    return;
    }
  XtVaSetValues(ai->tx, XmNvalue, "", NULL);
  }

// this callback function redraws the drawing area on exposure events
void redraw(Widget drawing_a, XtPointer client_data, XtPointer call_data) {
  XmDrawingAreaCallbackStruct *cbs =
    (XmDrawingAreaCallbackStruct *) call_data;
  appInfo *ai;
  bitmapInfo *ptr = (bitmapInfo *) client_data;

  XtVaGetValues(drawing_a, XmNuserData, &ai, NULL);

  XCopyArea (cbs->event->xexpose.display, ptr->pixmap, cbs->window, ai->gc,
    0, 0, ptr->width, ptr->height, 0, 0);
}

// destroys dinamically allocated objects and exits
void quit(Widget widget, XtPointer client_data, XtPointer call_data) {
  appInfo *ai = (appInfo *) client_data;

  if (ai->cityhf != NULL)
    delete ai->cityhf;
  exit(0);
  }

// grabs a rectangle from the pixmap to use as an index in the R*-tree
void grabRectangle(Widget  da, XEvent *event,
                   String *args, int *num_args) {
  static Position x, y, x1, x2, y1, y2;
  XButtonEvent *bevent = (XButtonEvent *) event;
  appInfo *ai;
  GC gc;

  // Get the application information from user data
  XtVaGetValues(da, XmNuserData, &ai, NULL);
  // if event = button1 down
  if (!strcmp (args[0], "down")) {
    x = x1 = x2 = bevent->x;
    y = y1 = y2 = bevent->y;
    }

  if (!strcmp (args[0], "motion")) {
    XCopyArea (XtDisplay (ai->da), ai->bitmap.pixmap, XtWindow(ai->da),
                ai->gc, 0, 0, ai->bitmap.width, ai->bitmap.height, 0, 0);

    ai->gcv.function = GXcopy;
    gc = XCreateGC(XtDisplay (ai->da), XtWindow(ai->da),
          GCFunction, &ai->gcv );
    x1 = bevent->x < x?bevent->x:x;
    y1 = bevent->y < y?bevent->y:y;
    x2 = bevent->x >= x?bevent->x:x;
    y2 = bevent->y >= y?bevent->y:y;
    XDrawRectangle (bevent->display, bevent->window, gc,
                x1, y1, x2-x1, y2-y1);
    }
```

```
    if (!strcmp (args[0], "up")) {
      ai->rectangle = rectangleClass(x1, y1, x2, y2);
      XCopyArea (XtDisplay (ai->da), ai->bitmap.pixmap, XtWindow(ai->da),
                        ai->gc, 0, 0, ai->bitmap.width, ai->bitmap.height, 0, 0);
      cityInput(ai);
      }
}

// handles the options of the Geo. Data menu option
void geoDataIO(Widget widget, XtPointer client_data, XtPointer call_data) {
   int option = (int) client_data;
   Widget form = widget, scrolw, drawa, *wptr;
   Cardinal ccount, i;
   appInfo *ai;

   // get the drawing area through the widget hierarchy
   while (!XmIsForm(form)) form = XtParent(form);
   XtVaGetValues(form, XmNnumChildren, &ccount,
              XmNchildren, &wptr, NULL);
   for(i=0; i<ccount; i++) {
     if (XmIsScrolledWindow(wptr[i])) {
       scrolw = wptr[i];
       break;
       }
     }
   XtVaGetValues(scrolw, XmNworkWindow, &drawa, NULL);
   // get the application info from the drawing area
   XtVaGetValues(drawa, XmNuserData, &ai, NULL);
   switch(option) {
      case 0:  XtVaSetValues(drawa,
              XmNtranslations,
                        XtParseTranslationTable (ai->cityInputTran),
                        NULL);
      }
//   XtOverrideTranslations(ai->da,
//           XtParseTranslationTable (ai->cityTranslations));
   }

// handles the input of city info
void cityInput(appInfo* ai) {
  Widget dialog, rowcol, form, button, textw;
  XmString str;
  int i;
  cityWriteStru *crs;

  // overrides the translations
  XtOverrideTranslations(ai->da,
           XtParseTranslationTable(ai->cityNone));

  // initialize structure to pass to the write callback
  crs = new cityWriteStru;
  crs->file = ai->cityhf;
  crs->rtree = ai->rtree;
  crs->cityCode = ai->cityCount++;
  crs->rectangle = ai->rectangle;
  crs->textFields = new Widget[XtNumber (citylabels)];
  crs->numOfTF =  XtNumber (citylabels);
  // create a dialog shell
  dialog = XtVaCreateWidget("City Information", xmDialogShellWidgetClass,
```

```
        ai->da, NULL);

// create the dialog shell's children
rowcol = XtVaCreateWidget("rowCol", xmRowColumnWidgetClass , dialog,
    NULL);

for (i = 0; i < XtNumber (citylabels); i++) {
    form = XtVaCreateWidget ("form", xmFormWidgetClass, rowcol,
        XmNfractionBase,        10,
        NULL);
    XtVaCreateManagedWidget (citylabels[i],
        xmLabelGadgetClass, form,
        XmNtopAttachment,   XmATTACH_FORM,
        XmNbottomAttachment, XmATTACH_FORM,
        XmNleftAttachment,  XmATTACH_FORM,
        XmNrightAttachment, XmATTACH_POSITION,
        XmNrightPosition,   3,
        XmNalignment,       XmALIGNMENT_END,
        NULL);
    textw = XtVaCreateManagedWidget ("textw",
        xmTextFieldWidgetClass, form,
        XmNtraversalOn,     True,
        XmNrightAttachment, XmATTACH_FORM,
        XmNleftAttachment,  XmATTACH_POSITION,
        XmNleftPosition,    4,
        NULL);
    crs->textFields[i] = textw;
    XtManageChild( form);
        }

XtVaCreateManagedWidget("separator", xmSeparatorGadgetClass, rowcol,
        NULL);

form = XtVaCreateWidget("form", xmFormWidgetClass, rowcol,
        XmNfractionBase,    5,
        NULL);

/* Create okbutton */
str = XmStringCreateSimple(" OK ");
button = XtVaCreateManagedWidget("okbutton", xmPushButtonGadgetClass,
    form,
    XmNlabelString,             str,
    XmNtopAttachment,           XmATTACH_FORM,
    XmNbottomAttachment,        XmATTACH_FORM,
    XmNrightAttachment,         XmATTACH_POSITION,
    XmNleftAttachment,          XmATTACH_POSITION,
    XmNleftPosition,            1,
    XmNrightPosition,           2,
    XmNshowAsDefault,           True,
    XmNdefaultButtonShadowThickness, 1,
    NULL);
XmStringFree(str);

XtAddCallback(button, XmNactivateCallback, (XtCallbackProc) cityWrite,
        crs);

/* create cancelbutton */
str = XmStringCreateSimple("Cancel");
button = XtVaCreateManagedWidget("cancelbutton", xmPushButtonGadgetClass,
        form,
```

```
    XmNlabelString,      str,
    XmNtopAttachment,    XmATTACH_FORM,
    XmNbottomAttachment, XmATTACH_FORM,
    XmNrightAttachment,  XmATTACH_POSITION,
    XmNleftAttachment,   XmATTACH_POSITION,
    XmNleftPosition,     3,
    XmNrightPosition,    4,
    XmNshowAsDefault,    False,
    XmNdefaultButtonShadowThickness, 1,
    NULL);
  XmStringFree(str);

  XtManageChild (form);
  XtManageChild (rowcol);
  XtManageChild (dialog);

  XtPopup(dialog, XtGrabNone);
  }

// handles the writing of city objects to disk
void cityWrite(Widget widget, XtPointer client_data, XtPointer call_data) {
  cityWriteStru *crs = (cityWriteStru *) client_data;
  int ok = True;
  char *strData1, *strData2, *strData3, *ptr;
  Widget shell;
  long value;

  strData1 = XmTextFieldGetString(crs->textFields[0]);
  // process first string to fill blanks with '_'
  ptr = strData1;
  while(*ptr) {
    if (*ptr == ' ') *ptr = '_';
    ptr++;
    }

  strData2 = XmTextFieldGetString(crs->textFields[1]);
  // process second string to fill blanks with '_'
  ptr = strData2;
  while(*ptr) {
    if (*ptr == ' ') *ptr = '_';
    ptr++;
    }

  strData3 = XmTextFieldGetString(crs->textFields[2]);
  // converts third string into a long value:
  char *strError = NULL;
  value = strtol(strData3, &strError, 10);
  if (strError[0] != '\x0')
    ok = False;
  if (ok) {
    city city1(crs->cityCode, strData1, strData2, value);
    city1.setRectangle(crs->rectangle);
    city1.display();
    cout << endl;
    DataBlock db = city1.passivate();
    crs->file->Write(city1.getKey(), db);
    long *ptr = new long;
    *ptr = city1.getKey();
    (*crs->rtree) << entryClass(city1.getRectangle(), ptr);
    shell = XtParent(widget);
```

```
    while (!XmIsDialogShell(shell)) shell = XtParent(shell);
    XtDestroyWidget(shell);
    }
  // deallocate memory used
  delete[] crs->textFields;
  delete crs;
  XtFree(strData1);
  XtFree(strData2);
  XtFree(strData3);
  }

void saveFile(Widget widget, XtPointer client_data, XtPointer call_data) {
  appInfo *ai = (appInfo*) client_data;
  char buf[512];
  DataFileOutput output(ai->mainFilename);
  ostrstream stream(buf, 512);

  memset(buf, 0, 512);
  stream << ai->baseFilename << " "
      << ai->bitmap.filename << " "
      << ai->cityCount;
  DataBlock db(ai->signature, strlen(buf), (void *) buf);
  output.Write(db);
  }
```

```
//          Interface file for the R*-tree
//          Felipe Guacache
//          Oklahoma State University
//          Computer Science Department
//          Last Modified: 07/31/95
#ifndef RPT_H
#define RPT_H

#ifndef __IOSTREAM_H
  #include <iostream.h>
#endif

#ifndef __MATH_H
  #include <math.h>
#endif

#ifndef __STDLIB_H
  #include <stdlib.h>
#endif

#define Mvalue 5 //Maximum number of entries per node
#define mvalue 2 //Minimum number of entries per node
#define kvalue 3 //k = M-2m+2
#define pvalue 2 //40% of M
#define true   1
#define false  0
#define on     1
#define off    0
#define verticalAxis 0
#define horizontalAxis 1
#define lowerRectangleValue 0
#define upperRectangleValue 1

class pointClass {
  private:
    float x,y;
  public:
    pointClass(void);
    pointClass(float, float);
    float getX(void);
    void setX(float);
    float getY(void);
    void setY(float);
    float distance(pointClass&);
  };

class rectangleClass {
  private:
    int x1,y1,x2,y2;
  public:
    rectangleClass(void);
    rectangleClass(int, int, int, int);
    rectangleClass(int, int);
    rectangleClass& operator= (rectangleClass&);
    operator==(rectangleClass&);
    operator!=(rectangleClass&);
    void setCoordinates(int, int, int, int);
    void setX1(int);
    getX1(void);
    void setY1(int);
```

```
    getY1(void);
    void setX2(int);
    getX2(void);
    void setY2(int);
    getY2(void);
    long getArea(void);
    getMargin(void);
    bad(void);
    friend ostream& operator<<(ostream&,rectangleClass&);
    pointClass center(void);
    int contains(rectangleClass&);
  };

class entryClass: public rectangleClass {
  private:
    void *pointer;
  public:
    entryClass(void);
    entryClass(int,int,int,int,void *);
    entryClass(rectangleClass&, void *);
    entryClass& operator=(entryClass&);
    void setPointer(void *);
    void *getPointer(void);
  };

class rTree; //incomplete definition

class nodeClass {
  private:
    entryClass children[Mvalue];
    nodeClass *parent;
    int level, leaf, numOfChildren;
    static int overflow;
    static rTree *tree;
  public:
    nodeClass(void);
    void setLevel(int);
    getLevel(void);
    void arrangeLevels(void);
    void adjust(void);
    void reAdjust(void);
    void setParent(nodeClass *);
    nodeClass& getParent(void);
    void setTree(rTree *);
    entryClass& operator[](int);
    void operator=(nodeClass&);
    rectangleClass bound(void);
    rectangleClass bound(entryClass&);
    long overlap(int);
    entryClass minOverlap(rectangleClass&);
    entryClass minAreaElgmnt(rectangleClass&);
    entryClass minArea(void);
    void setLeaf(int);
    isLeaf(void);
    void setNumOfChildren(int);
    getNumOfChildren(void);
    nodeClass *chooseSubTree(int, entryClass&);
    void insert(int, entryClass&);
    void *remove(rectangleClass&);
    void *remove(nodeClass *);
```

```
    friend ostream& operator<<(ostream&,nodeClass&);
    void operator<<(entryClass&);
    void overflowTreatment(entryClass&);
    void reinsert(entryClass&);
    nodeClass *split(entryClass&);
    void print(ostream&, int);
    nodeClass *searchNode(rectangleClass&);
    entryClass searchEntry(rectangleClass&);
    };

class rTree {
  private:
    int leafLevel;
    nodeClass *root;
    long numOfLeaves;
  public:
    rTree(void);
    void setLeafLevel(int);
    getLeafLevel(void);
    getNumOfLeaves(void);
    void setRoot(nodeClass *);
    nodeClass *getRoot(void);
    friend ostream& operator<<(ostream&, rTree&);
    void operator<<(entryClass&);
    void *remove(rectangleClass&);
    };
#endif
```

```
//          This is the implementation file for an object-oriented R*-Tree
//          Felipe Guacache
//          Oklahoma State University
//          Computer Sciences Department
//          Last modified: 07//95
//          For more information on R*-Trees read Beckmann's The R*-Tree: An
//          Efficient and Robust Access Method for Points and Rectangles.
//          ACM 1990.

#ifndef RPT_H
  #include "rpt.h"
#endif

//this class variable used to control the overflow
//treatment of nodes
int nodeClass::overflow = 0;

//this class variable used to point to the rtree
//remember to set this pointer to the address of the rTree
rTree * nodeClass::tree = NULL;

//pointClass methods
//default constructor
pointClass::pointClass(void) {

  x = 0.0;
  y = 0.0;
  }

//conversion constructor
pointClass::pointClass(float px, float py) {

  x = px;
  y = py;
  }

//gets the value of x
inline float pointClass::getX(void) {

  return x;
  }

//sets the value of x
inline void pointClass::setX(float px) {

  x = px;
  }

//gets the value of y
inline float pointClass::getY(void) {

  return y;
  }

//sets the value of y
inline void pointClass::setY(float py) {

  y = py;
  }
```

```
//returns the distance between two points
inline float pointClass::distance(pointClass& p) {

  return sqrt(pow((x-p.getX()),2) + pow((y-p.getY()),2));
  };

//rectangleClass methods
//default constructor
rectangleClass::rectangleClass(void) {

  x1=0;
  y1=0;
  x2=0;
  y2=0;
  }

//conversion constructor for rectangle access
rectangleClass::rectangleClass(int w, int x, int y, int z) {

  x1=w;
  y1=x;
  x2=y;
  y2=z;
  }

//conversion constructor for point access
rectangleClass::rectangleClass(int x, int y) {

  x1=x;
  y1=y;
  x2=x;
  y2=y;
  }

//sets the value of x1
void rectangleClass::setX1(int x) {

  x1=x;
  }

//returns the value of x1
rectangleClass::getX1(void) {

  return x1;
  }

//sets the value of y1
void rectangleClass::setY1(int y) {

  y1=y;
  }

//returns the value of y1
rectangleClass::getY1(void) {

  return y1;
  }

//sets the value of x2
void rectangleClass::setX2(int x) {
```

```
  x2=x;
  }

//returns the value of x2
rectangleClass::getX2(void) {

  return x2;
  }

//gets the value of y2
void rectangleClass::setY2(int y) {

  y2=y;
  }

//returns the value of y2
rectangleClass::getY2(void) {

  return y2;
  }

//assignment operator
rectangleClass& rectangleClass::operator=(rectangleClass& r) {

  x1 = r.getX1();
  y1 = r.getY1();
  x2 = r.getX2();
  y2 = r.getY2();
  return *this;
  }

//comparison operator
rectangleClass::operator==(rectangleClass& r) {

  if (x1 == r.getX1() && x2 == r.getX2() &&
     y1 == r.getY1() && y2 == r.getY2())
     return true;
  else
     return false;
  }

//comparison operator
rectangleClass::operator!=(rectangleClass& r) {

  return !(*this == r);
  }

//sets the value of the coordinates of a rectangle
void rectangleClass::setCoordinates(int w, int x, int y, int z) {

  x1=w;
  y1=x;
  x2=y;
  y2=z;
  }

//returns the area of a rectangle
inline long rectangleClass::getArea(void) {
```

```
  return (long) (x2-x1)*(y2-y1);
  }

//gets the margin (perimeter) of a rectangle
inline rectangleClass::getMargin(void) {

  return x2-x1+y2-y1;
  }

//returns true if the entry is bad
inline rectangleClass::bad(void) {

  if (x1 < 0 || y1 < 0 || x2 < 0 || y2 < 0 || x1 > x2 || y1 > y2)
    return true;
  else
    return false;
  }

//output operator for rectangles
ostream& operator<<(ostream& out, rectangleClass& r) {

  out << "(" << r.getX1() << "," << r.getY1() << ","
     << r.getX2() << "," << r.getY2() << ")";
  return out;
  }

//returns the center point of a rectangle
inline pointClass rectangleClass::center(void) {
  pointClass p;

  p.setX(x1 + (float) (x2-x1)/2);
  p.setY(y1 + (float) (y2-y1)/2);
  return p;
  }

//returns true if this rectangle contains rectangle r
rectangleClass::contains(rectangleClass& r) {

  if ( r.getX1() >= x1 && r.getY1() >= y1 &&
          r.getX2() <= x2 && r.getY2() <= y2)
    return true;
  else
    return false;
  }

//entryClass methods
//default constructor
entryClass::entryClass(void) {

  pointer = NULL;
  }

//conversion constructor
entryClass::entryClass(int w, int x, int y, int z, void *obj) {

  setX1(w);
  setY1(x);
  setX2(y);
  setY2(z);
  pointer = obj;
```

```
}

//conversion constructor
entryClass::entryClass(rectangleClass& r, void *obj) {

  setX1(r.getX1());
  setY1(r.getY1());
  setX2(r.getX2());
  setY2(r.getY2());
  pointer = obj;
  }

//assignment operator
entryClass& entryClass::operator=(entryClass& r) {

  setX1(r.getX1());
  setY1(r.getY1());
  setX2(r.getX2());
  setY2(r.getY2());
  pointer = r.getPointer();
  return *this;
  }

//sets the value of the pointer
void entryClass::setPointer(void *ptr) {

  pointer = ptr;
  }

//returns the value of the pointer
void *entryClass::getPointer(void) {

  return pointer;
  }

//nodeClass methods
//default constructor
nodeClass::nodeClass(void) {

  parent = NULL;
  level = 0;
  leaf = 0;
  numOfChildren = 0;
  }

//sets the value of level
void nodeClass::setLevel(int l) {

  level = l;
  }

//returns the value of level
nodeClass::getLevel(void) {

  return level;
  }

//arranges the level of the nodes after a root split
void nodeClass::arrangeLevels(void) {
  int i;
```

```
  nodeClass *n;

  if (parent != NULL)
    level = parent->getLevel()+1;
  if (!isLeaf())
    for (i=0;i<numOfChildren;i++) {
            n = (nodeClass *) children[i].getPointer();
            n->arrangeLevels();
            }
  }

//adjusts the covering rectangles after insertions/splits
void nodeClass::adjust(void) {
  int i;
  nodeClass *temp;

  if (parent != NULL) { //if it is not the root
    temp = parent;
    for(i=0;i<temp->getNumOfChildren();i++)
            if ((*temp)[i].getPointer() == this)  //adjust covering rectangle
              (*temp)[i] = entryClass(this->bound(), this);
    temp->adjust();
    }
  }

//adjusts the nodes rectangles after deletions
void nodeClass::reAdjust(void) {
  int i;
  nodeClass *temp, *newRoot;

  if (parent != NULL) { //if it is not the root
    temp = parent;
    temp->remove(this);
    //reinsert the entries in this node
    for (i=0;i<numOfChildren;i++)
            tree->getRoot()->insert(level, children[i]);
    if (temp->getNumOfChildren() < mvalue) {
            temp->reAdjust();
            if (temp != tree->getRoot())
              delete temp;
            if (temp == tree->getRoot() && temp->getNumOfChildren() == 1
              && !leaf) {
              newRoot = (nodeClass *) (*temp)[0].getPointer();
              delete temp;
              newRoot->setParent(NULL);
              newRoot->setLevel(0);
              newRoot->arrangeLevels();
              tree->setRoot(newRoot);
              tree->setLeafLevel(tree->getLeafLevel()-1);
              }
            }
    else
            temp->adjust();
    }
  }

//sets the parent pointer
void nodeClass::setParent(nodeClass *pr){

  parent = pr;
```

```
  }

//returns the object pointed to by parent
nodeClass& nodeClass::getParent(void) {

  return *parent;
  }

//sets the value of the pointer to rTree
void nodeClass::setTree(rTree *rt) {

  tree = rt;
  }

//returns the i-th entry
entryClass& nodeClass::operator[](int i) {

  return children[i];
  }

//assignment operator
void nodeClass::operator=(nodeClass& n) {
  int i;

  numOfChildren = n.getNumOfChildren();
  for (i=0;i<numOfChildren;i++)
    children[i] = n[i];
  parent = &n.getParent();
  level = n.getLevel();
  leaf = n.isLeaf();
  }

//returns the minimum bound rectangle of a nodeClass instance
rectangleClass nodeClass::bound(void) {
  rectangleClass r;
  int i;

  r = children[0];
  for (i=1;i<numOfChildren;i++) {
    if (children[i].getX1() < r.getX1())
          r.setX1(children[i].getX1());
    if (children[i].getY1() < r.getY1())
          r.setY1(children[i].getY1());
    if (children[i].getX2() > r.getX2())
          r.setX2(children[i].getX2());
    if (children[i].getY2() > r.getY2())
          r.setY2(children[i].getY2());
    }
  return r;
  }

//returns the minimun bound rectangle of a nodeClass instance
//if included a new entry
rectangleClass nodeClass::bound(entryClass& e) {
  rectangleClass r = bound();

  if (e.getX1() < r.getX1())
    r.setX1(e.getX1());
  if (e.getY1() < r.getY1())
    r.setY1(e.getY1());
```

```
  if (e.getX2() > r.getX2())
    r.setX2(e.getX2());
  if (e.getY2() > r.getY2())
    r.setY2(e.getY2());
  return r;
  }

//returns the intersection of two rectangles
rectangleClass intersect(rectangleClass& r1, rectangleClass& r2) {
  rectangleClass r=r1;

  if (r2.getX1() > r.getX1())
    r.setX1(r2.getX1());
  if (r2.getY1() > r.getY1())
    r.setY1(r2.getY1());
  if (r2.getX2() < r.getX2())
    r.setX2(r2.getX2());
  if (r2.getY2() < r.getY2())
    r.setY2(r2.getY2());
  if (r.getX1() > r.getX2() || r.getY1() > r.getY2())
    return rectangleClass(0,0,0,0);
  else
    return r;
  }

//returns the overlap for a given children
long nodeClass::overlap(int j) {
  long area=0;
  int i;

  for (i=0;i<numOfChildren;i++)
    if (i != j)
        area += intersect(children[j],children[i]).getArea();
  return area;
  }

//returns the node with minimum overlap
entryClass nodeClass::minOverlap(rectangleClass& r) {
  int i, choice[Mvalue], count=0;
  long minimum, overlapVal[Mvalue];
  nodeClass node1, node2;

  //a copy of the object in node
  node1 = *this;
  //every rectangle is augmented to contain the new rectangle
  for (i=0;i < numOfChildren;i++) {
    if (r.getX1() < node1.children[i].getX1())
        node1.children[i].setX1(r.getX1());
    if (r.getY1() < node1.children[i].getY1())
        node1.children[i].setY1(r.getY1());
    if (r.getX2() > node1.children[i].getX2())
        node1.children[i].setX2(r.getX2());
    if (r.getY2() > node1.children[i].getY2())
        node1.children[i].setY2(r.getY2());
    }
  for (i=0; i<numOfChildren;i++) //increases in overlap are calculated
    overlapVal[i] = node1.overlap(i) - overlap(i);

  minimum = overlapVal[0];
  choice[0]=0;
```

```
  for (i=0;i < numOfChildren; i++)
    if (overlapVal[i] < minimum) {
            minimum = overlapVal[i];
            count = 0;
            choice[count++] = i;
            }
    else
            if (overlapVal[i] == minimum)
              choice[count++] = i;

  //if there are ties solve with minimum area enlargement
  if (count>1) {
    for(i=0;i<count;i++)
            node2[i]=children[choice[i]];
    node2.setNumOfChildren(count);
    return node2.minAreaElgmnt(r);
    }
  else {
    if (choice[0] > numOfChildren) {
            cerr << "Error in choice.  MinOverlap." << endl;
            exit(1);
            }
    return children[choice[0]];
    }
  }

//returns the node that needs minimum enlargement
entryClass nodeClass::minAreaElgmnt(rectangleClass& r) {
  int i, choice[Mvalue], count=0;
  long minimum, areaElgmntVal[Mvalue];
  nodeClass node1, node2;

  //a copy of the object in node
  node1 = *this;
  //every rectangle is augmented to contain the new rectangle
  for (i=0;i < numOfChildren;i++) {
    if (r.getX1() < node1.children[i].getX1())
            node1.children[i].setX1(r.getX1());
    if (r.getY1() < node1.children[i].getY1())
            node1.children[i].setY1(r.getY1());
    if (r.getX2() > node1.children[i].getX2())
            node1.children[i].setX2(r.getX2());
    if (r.getY2() > node1.children[i].getY2())
            node1.children[i].setY2(r.getY2());
    //increases in area are calculated
    areaElgmntVal[i] = node1.children[i].getArea() - children[i].getArea();
    }
  minimum = areaElgmntVal[0];
  choice[0] = 0;
  for (i=0;i < numOfChildren; i++) {
    if (areaElgmntVal[i] < minimum) {
            minimum = areaElgmntVal[i];
            count = 0;
            choice[count++] = i;
            }
    else
            if (areaElgmntVal[i] == minimum)
              choice[count++] = i;
    }
  //if there are ties resolve with minimum area
```

```
  if (count>1) {
    for(i=0;i<count;i++)
            node2[i]=children[choice[i]];
    node2.setNumOfChildren(count);
    return node2.minArea();
    }
  else {
    if (choice[0]>numOfChildren) {
            cerr << "Error in choice.  minAreaElgmnt." << endl;
            exit(1);
            }
    return children[choice[0]];
    }

  }

//returns the node with minimum area
entryClass nodeClass::minArea(void) {
  int i, choice;
  long minimum;

  minimum = children[0].getArea();
  choice = 0;
  for (i = 1;i < numOfChildren; i++) {
    if (children[i].getArea() < minimum) {
            minimum = children[i].getArea();
            choice = i;
            }
    }
  if (choice>numOfChildren) {
    cerr << "Error in choice.  MinArea." << endl;
    exit(1);
    }
  return children[choice];
  }

//sets the leaf attribute
void nodeClass::setLeaf(int l) {

  leaf = l;
  }


//returns whether the nodeClass is a leaf or not
nodeClass::isLeaf(void) {

  return leaf;
  }

//sets the number of children
void nodeClass::setNumOfChildren(int n) {

  numOfChildren = n;
  }

//returns the number of children of a node
nodeClass::getNumOfChildren(void) {

  return numOfChildren;
  }
```

```
//chooses an appropiate subtree for insertion of nodes
nodeClass *nodeClass::chooseSubTree(int l, entryClass& e) {
  nodeClass *chosen;
  entryClass entry;

  if (level==l)
    return this;
  else {
    if (level == tree->getLeafLevel()-1) //if childpointers point to leaves
            //choose the one with minimum overlap cost
            entry = minOverlap(e);
    else
            //choose the one with minimum area cost
            entry = minAreaElgmnt(e);
    chosen = (nodeClass*)entry.getPointer();

    return chosen->chooseSubTree(l,e);
    }
  }

//handles node insertions
void nodeClass::insert(int l, entryClass& e) {
  nodeClass *node;

  node = chooseSubTree(l,e);
  if(node->getNumOfChildren() < Mvalue)
    *node << e;
  else
    node->overflowTreatment(e);
  }

//deletes an entry from a node given a rectangle
void* nodeClass::remove(rectangleClass& r) {
  int i, todel;
  void *obj;

  for (i=0;i<numOfChildren;i++)
    if (children[i] == r) {
            todel = i;
            obj = children[i].getPointer();
            break;
            }
  if (todel != numOfChildren-1)
    children[todel] = children[numOfChildren-1];
  numOfChildren--;
  return obj;
  }

//deletes an entry from a node given a pointer
void* nodeClass::remove(nodeClass *n) {
  int i, todel;
  void *obj;

  for (i=0;i<numOfChildren;i++)
    if (children[i].getPointer() == n) {
            todel = i;
            obj = children[i].getPointer();
            break;
            }
```

```
  if (todel != numOfChildren-1)
    children[todel] = children[numOfChildren-1];
  numOfChildren--;
  return obj;
  }

//inserts a new entry into a node
void nodeClass::operator<<(entryClass& e) {
  nodeClass* ptr;

  children[numOfChildren] = e;
  numOfChildren++;
  if (!leaf) {
    ptr = (nodeClass*) e.getPointer();
    ptr->setParent(this);
    ptr->setLevel(level+1);
    }
  adjust();
  }


//checks the overflow class variable before splitting nodes
void nodeClass::overflowTreatment(entryClass& e) {
  nodeClass *ptr;

  //if not root level and no previous overflow ocurred at this level
  if ( level!=0 && ((overflow>>level)&1)==0) {
    reinsert(e);
    }
  else {
    ptr = split(e);
    adjust();
    if (ptr != NULL)
          tree->getRoot()->insert(ptr->getLevel()-1,
                                  entryClass(ptr->bound(),ptr));
    }
  }

//reinserts the first pvalues of a node
void nodeClass::reinsert(entryClass& e) {
  int i, j;
  entryClass entries[Mvalue+1], etemp;
  rectangleClass minbound;
  float distances[Mvalue+1], dtemp;
  nodeClass *ptr;

  //update overflow class variable
  overflow = overflow|(1<<level);
  //put all the entries together
  for(i=0;i<Mvalue;i++)
    entries[i] = children[i];
  entries[Mvalue] = e;
  //minimum bound including the new entry
  minbound = this->bound(e);
  //calculates the distances between the centers
  for(i=0; i<Mvalue+1; i++)
    distances[i] = minbound.center().distance(entries[i].center());
  //sorts the first pvalue rectangles by their distances
  for(i=0;i<pvalue;i++)
    for(j=i;j<Mvalue+1;j++)
```

```
            if (distances[j] < distances[i]) {
              dtemp = distances[i];
              distances[i] = distances[j];
              distances[j] = dtemp;
              etemp = entries[i];
              entries[i] = entries[j];
              entries[j] = etemp;
              }
  //nodes readjusted
  for(i=pvalue;i<Mvalue+1;i++) {
    children[i-pvalue] = entries[i];
    if (!leaf) {
            ptr = (nodeClass*)entries[i].getPointer();
            ptr->setParent(this);
            ptr->setLevel(level+1);
            }
    }
  numOfChildren = Mvalue+1-pvalue;
  adjust();

  //first pvalue entries reinserted
  for(i=0;i<pvalue;i++)
//    tree->getRoot()->insert(level,entries[i]);
    parent->insert(level,entries[i]);
  }

//splits a node
nodeClass *nodeClass::split(entryClass& e) {
  int i, j, k, l, count, splitAxis, splitGroup, splitIndex,
    groups[Mvalue+1], indices[Mvalue+1];
  entryClass entries[2][2][Mvalue+1], temp;
  long marginValues[2][2][kvalue],
          overlapValues[2][2][kvalue],
          areaValues[2][2][kvalue],
          sumMarginValues[2],
          mOverlap, mArea;
  nodeClass node1, node2, *newOne, *newRoot, *ptr;

  //distribute the entries into the different groups
  for(i=0;i<2;i++)
    for(j=0;j<2;j++) {
            for(k=0;k<Mvalue;k++)
              entries[i][j][k] = children[k];
            entries[i][j][Mvalue] = e;
            }
  //sorts different groups
  //since Mvalue is not a large number a bubble sort suffices
  for(i=0;i<2;i++)
    for(j=0;j<2;j++)
            for(k=0;k<Mvalue;k++)
              for (l=k;l<Mvalue+1;l++) {
                //vertical axis lower rectangle values
                if(i == verticalAxis && j == lowerRectangleValue &&
                        entries[i][j][l].getY1() < entries[i][j][k].getY1()) {
                        temp = entries[i][j][k];
                        entries[i][j][k] = entries[i][j][l];
                        entries[i][j][l] = temp;
                        }
                //vertical axis upper rectangle values
                if(i == verticalAxis && j == upperRectangleValue &&
```

```
                    entries[i][j][l].getY2() < entries[i][j][k].getY2()) {
                    temp = entries[i][j][k];
                    entries[i][j][k] = entries[i][j][l];
                    entries[i][j][l] = temp;
                    }
                //horizontal axis lower rectangle values
                if(i == horizontalAxis && j == lowerRectangleValue &&
                    entries[i][j][l].getX1() < entries[i][j][k].getX1()) {
                    temp = entries[i][j][k];
                    entries[i][j][k] = entries[i][j][l];
                    entries[i][j][l] = temp;
                    }
                //horizontal axis upper rectangle values
                if(i == horizontalAxis && j == upperRectangleValue &&
                    entries[i][j][l].getX2() < entries[i][j][k].getX2()) {
                    temp = entries[i][j][k];
                    entries[i][j][k] = entries[i][j][l];
                    entries[i][j][l] = temp;
                    }
                }

//  for(i=0;i<2;i++)
//    for(j=0;j<2;j++) {
//          if (i==horizontalAxis)
//            cout << "Horizontal Axis ";
//          else
//            cout << "Vertical Axis ";
//          if (j == lowerRectangleValue)
//            cout << "Lower Rectangle Value.";
//          else
//            cout << "Upper Rectangle Value.";
//          cout << endl;
//          for(k=0;k<Mvalue+1;k++)
//            cout << entries[i][j][k] << endl;
//          }

  //get goodness values for the different groups
  for(i=0;i<2;i++)
    for(j=0;j<2;j++)
          for(k=0;k<kvalue;k++) {
            for(l=0;l<mvalue+k;l++)  //first set of entries
              node1[l] = entries[i][j][l];
            node1.setNumOfChildren(mvalue+k);
            for(l=mvalue+k;l<Mvalue+1;l++) //second set of entries
              node2[l-mvalue-k] = entries[i][j][l];
            node2.setNumOfChildren(Mvalue+1-mvalue-k);
            //goodness values for k-th group
            marginValues[i][j][k] = node1.bound().getMargin() +
                                        node2.bound().getMargin();
            overlapValues[i][j][k] = intersect(node1.bound(),
                                        node2.bound()).getArea();
            areaValues[i][j][k] = node1.bound().getArea() +
                                        node2.bound().getArea();
            }
  //calculate the sum of all margin-values for each axis
  for (j=0;j<2;j++)
    for(k=0;k<kvalue;k++)
          sumMarginValues[j] = marginValues[j][lowerRectangleValue][k] +
                                    marginValues[j][upperRectangleValue][k];
```

```
//choose split axis
if (sumMarginValues[horizontalAxis] < sumMarginValues[verticalAxis])
  splitAxis = horizontalAxis;
else
  splitAxis = verticalAxis;
//choose the best distribution for split within the chosen axis
//split group and split index
mOverlap = overlapValues[splitAxis][0][0];
groups[0] = 0;
indices[0] = 0;
count = 0;
//obtain minimum overlap value
for (i=0;i<2;i++)
  for (j=0; j<kvalue; j++)
          if (overlapValues[splitAxis][i][j] < mOverlap) {
            mOverlap = overlapValues[splitAxis][i][j];
            count = 0;
            groups[count] = i;
            indices[count++] = j;
            }
          else
            if (mOverlap == overlapValues[splitAxis][i][j]) {
              groups[count] = i;
              indices[count++] = j;
              }

splitGroup = groups[0];
splitIndex = indices[0];
//if there are ties resolve with minimum area values
if (count>1) {
  mArea = areaValues[splitAxis][splitGroup][splitIndex];
  for (i=0;i<count;i++)
          if (areaValues[splitAxis][groups[i]][indices[i]] < mArea) {
            splitGroup = groups[i];
            splitIndex = indices[i];
            mArea = areaValues[splitAxis][splitGroup][splitIndex];
            }
  }
//distribute the entries
newOne = new nodeClass;
if (newOne == NULL) {
  cerr << "Memory allocation error for new node.  Method nodeClass.split()." << endl;
  cerr << tree->getNumOfLeaves() << " Leaves." << endl;
  exit(1);
  }
//first group of entries in this node
for (i=0;i<splitIndex+mvalue;i++) {
  children[i] = entries[splitAxis][splitGroup][i];
  if (!leaf) {
          ptr = (nodeClass*)children[i].getPointer();
          ptr->setParent(this);
          }
  }
numOfChildren = splitIndex+mvalue;
//second group of entries in new node
for (i=splitIndex+mvalue; i<Mvalue+1; i++) {
  (*newOne)[i-splitIndex-mvalue] = entries[splitAxis][splitGroup][i];
  if (!leaf) {
          ptr = (nodeClass*)(*newOne)[i-splitIndex-mvalue].getPointer();
          ptr->setParent(newOne);
```

```
            }
    }
  newOne->setNumOfChildren(Mvalue+1-splitIndex-mvalue);
  newOne->setLevel(level);
  newOne->setLeaf(leaf);
  //if the root is splitted
  if (level == 0) {
    newRoot = new nodeClass;
    if (newRoot == NULL) {
            cerr << "Memory allocation error   Method nodeClass.split()." << endl;
            cerr << tree->getNumOfLeaves() << " Leaves." << endl;
            exit(1);
            }
    (*newRoot)[0] = entryClass(bound(),this);
    (*newRoot)[1] = entryClass(newOne->bound(), newOne);
    newRoot->setNumOfChildren(2);
    parent = newRoot;
    newOne->setParent(newRoot);
    newRoot->arrangeLevels();
    //update overflow information
    overflow = 0;
    tree->setRoot(newRoot);
    tree->setLeafLevel(tree->getLeafLevel()+1);
    return NULL;
    }
  else {
    return newOne;
    }
  }

//output operator for nodes
ostream& operator<<(ostream& out, nodeClass& n) {
  int i;
  nodeClass *ptr;

  out << "<" << &n;
  for(i=0;i<n.numOfChildren;i++)
    out << n[i];
  out << n.parent << " " << n.level << " " << n.leaf
     << " " << n.numOfChildren << ">" << endl;
  return out;
  }

//searches for a given rectangle and returns the entry it is in
entryClass nodeClass::searchEntry(rectangleClass& r) {
  nodeClass *ptr;
  entryClass entry;
  int i;

  if (isLeaf()) {
    for (i=0;i<numOfChildren;i++)
            if (children[i]==r)
              return children[i];
    }
  else
    for (i=0;i<numOfChildren;i++)
            if (children[i].contains(r)) {
              ptr = (nodeClass *) children[i].getPointer();
              entry = ptr->searchEntry(r);
              if (entry.getPointer() != NULL)
```

```
            return entry;
          }
  return entryClass();
  }

//searches for a given rectangle and returns a pointer to the node it is in
nodeClass *nodeClass::searchNode(rectangleClass& r) {
  nodeClass *ptr1, *ptr2;
  int i;

  if (isLeaf()) {
    for (i=0;i<numOfChildren;i++)
          if (children[i]==r)
            return this;
    }
  else
    for (i=0;i<numOfChildren;i++)
          if (children[i].contains(r)) {
            ptr1 = (nodeClass *) children[i].getPointer();
            ptr2 = ptr1->searchNode(r);
//          if (ptr2->getNumOfChildren() != 0)
            if (ptr2 != NULL)
              return ptr2;
          }
  return NULL;
  }

//rTree methods
//default constructor
rTree::rTree(void) {

  leafLevel = 0;
  numOfLeaves = 0;
  root = NULL;
  }

//sets the pointer to the root
void rTree::setRoot(nodeClass *ptr) {

  root = ptr;
  }

//returns the pointer to the root
nodeClass *rTree::getRoot(void) {

  return root;
  }

//sets the leaf level
void rTree::setLeafLevel(int l) {

  leafLevel = l;
  }

//returns the leaf level
rTree::getLeafLevel(void) {

  return leafLevel;
  }
```

```
//returns the number of leaves
rTree::getNumOfLeaves(void) {

  return numOfLeaves;
  }

//insertion operator for rTree
void rTree::operator<<(entryClass& e) {

  if (root == NULL) {
    root = new nodeClass;
    if (root == NULL) {
              cerr << "Memory allocation error for root. rTree.operator<<(entryClass)."
                   << endl;
              cerr << numOfLeaves << " Leaves." << endl;
              }
    root->setLeaf(true);
    root->setTree(this);
    *root << e;
    }
  else
    root->insert(leafLevel,e);
  numOfLeaves++;
  }

//deletes nodes from an rTree
void *rTree::remove(rectangleClass& r) {
  nodeClass *ptr, *newRoot;
  void *obj = NULL;

  ptr = root->searchNode(r);
  if (ptr == NULL)
    cout << "Entry not found." << endl;
  else {
    obj = ptr->remove(r);
    if (ptr->getNumOfChildren() < mvalue && ptr != root) {
              ptr->reAdjust();
              //dispose pointer here
              delete ptr;
              }
    else
              ptr->adjust();

    if (root->getNumOfChildren() == 1 && leafLevel ) {
              newRoot = (nodeClass *) (*root)[0].getPointer();
              delete ptr;
              newRoot->setParent(NULL);
              newRoot->setLevel(0);
              root = newRoot;
              leafLevel--;
              }
    numOfLeaves--;
    if (numOfLeaves == 0) {
              delete root;
              root = NULL;
              }
    }
  return obj;
  }
```

```
//prints out all the nodes at a given level
void nodeClass::print(ostream& out, int l) {
  nodeClass *ptr;
  int i;

  if (level == l)
    out << *this;
  else {
    if (level < l) {
              for (i=0; i<numOfChildren; i++) {
                ptr = (nodeClass*) children[i].getPointer();
                ptr->print(out, l);
                }
              }
    }
  }

//output operator for rTree
ostream& operator<<(ostream& out, rTree& r) {
  int i;

  out << "Number of Leaves: " << r.numOfLeaves << endl
     << "Leaflevel  " << r.leafLevel << endl
     << "Root : " << r.root << endl;
  for (i=0;i<=r.leafLevel;i++) {
     r.root->print(out, i);
     out << endl << endl;
     }
  return out;
  }
```

```
// Program Name:  DTCL.H
// An Object Oriented Graphic User Interface for Transportation Scheduling Problem Specification
// Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Last Modified: 08/31/95

#ifndef DTCL_H
#define DTCL_H

#ifndef __IOSTREAM_H
  #include <iostream.h>
#endif

#ifndef __STDIO_H
  #include <stdio.h>
#endif

#ifndef __IOMANIP_H
  #include <iomanip.h>
#endif

class timeClass {
  protected:
    unsigned char hours, minutes, seconds;
  public:
    timeClass(void);
    timeClass(unsigned char, unsigned char, unsigned char);
    timeClass(const char *);
    timeClass(timeClass&);
    timeClass& operator= (timeClass&);
    operator< (timeClass&);
    operator> (timeClass&);
    operator== (timeClass&);
    operator!= (timeClass&);
    friend ostream& operator<< (ostream&, timeClass&);
    };

class dateClass {
  protected:
    unsigned int year;
    unsigned char month, day;
  public:
    dateClass(void);
    dateClass(unsigned int, unsigned char, unsigned char);
    dateClass(const char *);
    dateClass(dateClass&);
    dateClass& operator= (dateClass&);
    operator< (dateClass&);
    operator> (dateClass&);
    operator== (dateClass&);
    operator!= (dateClass&);
    friend ostream& operator<< (ostream&, dateClass&);
    };

class dtClass: public dateClass, public timeClass {
  public:
    dtClass(void);
    dtClass(const char *, const char*);
    dtClass(timeClass&, dateClass&);
```

```
    dtClass(dtClass&);
    dtClass& operator= (dtClass&);
    operator< (dtClass&);
    operator> (dtClass&);
    operator== (dtClass&);
    operator!= (dtClass&);
    friend ostream& operator<< (ostream&, dtClass&);
    };

#endif
```

```
//
// Program Name: DTCL.CPP
// Date and time objects
// Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Stillwater, Oklahoma
// Last updated: 08/24/95
//

#ifndef DTCL_H
  #include "dtcl.h"
#endif

// timeClass member functions
// default constructor
timeClass::timeClass(void) {

  hours = 0;
  minutes = 0;
  seconds = 0;
  }

// conversion constructor
timeClass::timeClass(unsigned char h, unsigned char m, unsigned char s) {

  if (h < 24 && m < 60 && s < 60) {
    hours = h;
    minutes = m;
    seconds = s;
    }
  else {
    hours = 0;
    minutes = 0;
    seconds = 0;
    }
  }

//conversion constructor
timeClass::timeClass(const char *timeStr) {
  unsigned int h, m, s;

  sscanf(timeStr, "%u:%u:%u", &h, &m, &s);
  if (h < 24 && m < 60 && s < 60) {
    hours = h;
    minutes = m;
    seconds = s;
    }
  else {
    hours = 0;
    minutes = 0;
    seconds = 0;
    }
  }

//copy constructor
timeClass::timeClass(timeClass& t1) {

  hours = t1.hours;
  minutes = t1.minutes;
```

```
  seconds = t1.seconds;
  }

//assigment operator
timeClass& timeClass::operator=(timeClass& t1) {

  hours = t1.hours;
  minutes = t1. minutes;
  seconds = t1.seconds;
  return t1;
  }

//less-than operator
timeClass::operator<(timeClass& t1) {

  if (hours != t1.hours)
    return (hours < t1.hours);
  else
    if (minutes != t1.minutes)
          return (minutes < t1.minutes);
  return (seconds < t1.seconds);
  }

//greater-than operator
timeClass::operator>(timeClass& t1) {

  if (hours != t1.hours)
    return (hours > t1.hours);
  else
    if (minutes != t1.minutes)
          return (minutes > t1.minutes);
  return (seconds > t1.seconds);
  }

//equal-to operator
timeClass::operator==(timeClass& t1) {

  if (hours == t1.hours && minutes == t1.minutes && seconds == t1.seconds)
    return 1;
  else
    return 0;
  }

//different-from operator
timeClass::operator!=(timeClass& t1) {

  return !(*this == t1);
  }

//output operator friend function
ostream& operator<< (ostream& out, timeClass& t1) {

  out << setw(2) << setfill('0') << (int)t1.hours << ':'
     << setw(2) << setfill('0') << (int)t1.minutes << ':'
     << setw(2) << setfill('0') << (int)t1.seconds;
  return out;
  }

//dateClass member functions
//default constructor
```

```
dateClass::dateClass(void) {

  year = 1995;
  month = 1;
  day = 1;
  }

//conversion constructor
dateClass::dateClass(unsigned int y, unsigned char m, unsigned char d) {

  // validate month and day
  if (m<13 && d>0 &&
     (((m==1 || m==3 || m==5 || m==7 || m==8 || m==10 || m == 12) && d<32) ||
          ((m==4 || m==6 || m==9 || m==11) && d<31) ||
          (m==2 && y%4 == 0 && d<30) ||
          (m==2 && y%4 && d<29))) {
    year = y;
    month = m;
    day = d;
    }
  else { //if incorrect date assign an arbitrary one
    year = 1995;
    month = 1;
    day = 1;
    }
  }

//conversion constructor
dateClass::dateClass(const char *dateStr) {
  unsigned int m,d,y;

  sscanf(dateStr,"%u/%u/%u", &m, &d, &y);
  //validate month and day
  if (m<13 && d>0 &&
     (((m==1 || m==3 || m==5 || m==7 || m==8 || m==10 || m == 12) && d<32) ||
          ((m==4 || m==6 || m==9 || m==11) && d<31) ||
          (m==2 && y%4 == 0 && d<30) ||
          (m==2 && y%4 && d<29))) {
    year = y;
    month = m;
    day = d;
    }
  else {  //if incorrect assign an arbitrary one
    year = 1995;
    month = 1;
    day = 1;
    }
  }

//copy constructor
dateClass::dateClass(dateClass& d1) {

  year = d1.year;
  month = d1.month;
  day = d1.day;
  }

//assignment operator
dateClass& dateClass::operator=(dateClass& d1) {
```

```
  year = d1.year;
  month = d1.month;
  day = d1.day;
  return d1;
  }

//less-than opeartor
dateClass::operator<(dateClass& d1) {

  if (year != d1.year)
    return (year < d1.year);
  else
    if (month != d1.month)
          return (month < d1.month);
  return (day < d1.day);
  }

//greater-than operator
dateClass::operator>(dateClass& d1) {

  if (year != d1.year)
    return (year > d1.year);
  else
    if (month != d1.month)
          return (month > d1.month);
  return (day > d1.day);
  }

//equal-to operator
dateClass::operator==(dateClass& d1) {

  if (year == d1.year && month == d1.month && day == d1.day)
    return 1;
  else
    return 0;
  }

//different-from operator
dateClass::operator!=(dateClass& d1) {

  return !(*this == d1);
  }

//output operator friend function
ostream& operator<<(ostream& out, dateClass& d1) {

  out << setw(2) << setfill('0') << (int)d1.month << '/'
     << setw(2) << setfill('0') << (int)d1.day << '/'
     << setw(4) << setfill('0') << d1.year;
  return out;
  }

//dtClass member functions
//default constructor
dtClass::dtClass(void):timeClass(),dateClass(){

 //nada
 }

//conversion constructor
```

```
dtClass::dtClass(const char *timeStr, const char *dateStr):
                        timeClass(timeStr),dateClass(dateStr) {

  //nada
  }

//conversion constructor
dtClass::dtClass(timeClass& t1, dateClass& d1) timeClass(t1),
                                                    dateClass(d1) {

  //nada
  }

//copy constructor
dtClass::dtClass(dtClass& dt1) {

  hours = dt1.hours;
  minutes = dt1.minutes;
  seconds = dt1.seconds;
  year = dt1.year;
  month = dt1.month;
  day = dt1.day;
  }

//assignment opeartor
dtClass& dtClass::operator=(dtClass& dt1) {

  hours = dt1.hours;
  minutes = dt1.minutes;
  seconds = dt1.seconds;
  year = dt1.year
  month = dt1.month;
  day = dt1.day;
  return dt1;
  }

//less-than operator
dtClass::operator< (dtClass& dt1) {

  if ((dateClass)*this != (dateClass) dt1)
    return (dateClass) *this < (dateClass)dt1
  else
    return (timeClass) *this < (timeClass)dt1;
  }

//greater-than operator
dtClass::operator> (dtClass& dt1) {

  if ((dateClass)*this != (dateClass) dt1)
    return (dateClass) *this > (dateClass)dt1;
  else
    return (timeClass) *this > (timeClass)dt1;
  }

//equal-to operator
dtClass::operator==(dtClass& dt1) {

  if ((dateClass)*this == (dateClass)dt1 &&
     (timeClass)*this == (timeClass)dt1)
    return 1;
```

```
  else
    return 0;
  }

//different-from operator
dtClass::operator!=(dtClass& dt1) {

  return !(*this == dt1);
  }

//output operator friend function
ostream& operator<<(ostream& out, dtClass& dt1) {

  out << (dateClass)dt1 << ',' << (timeClass)dt1;
  return out;
  }
```

```
// DATACL.H
// This program handles the different kinds of data used for
// transportation scheduling specification.
//
// Felipe Guacache
// Department of Computer Science
// Oklahoma State University
// Stillwater, Oklahoma
// Last modified: 08/26/95
//
#ifndef DATACL_H
#define DATACL_H

#include "rpt.h"
#include "dtel.h"
#include "persist.h"
#include "str.h"
#ifndef __STRING_H
  #include <string.h>
#endif

// Some enumerated types to keep
// track of details

enum pathTypeEnum {
  _highWay,
  _road,
  _airWay,
  _railRoad,
  _seaRoute
  };

enum locationTypeEnum {
  _depot,
  _airport,
  _trainStation,
  _seaPort
  };

enum cargoTypeEnum {
  _miscellaneous,
  _personnel,
  _food,
  _frozenFood,
  _cannedFood,
  _printedMatter,
  _equipment,
  _heavyMachinery,
  _fireArms,
  _waste
  };

enum convTypeEnum {
  _vehicle,
  _truck,
  _plane,
  _train,
  _ship
  };
```

```
enum constTypeEnum {
  _weight,
  _volumen,
  _temperature,
  _speed,
  _noWaste
  };

enum constOnEnum {
  _city,
  _path,
  _conveyance,
  _cargo
  };

// Class definitions
class dataClass {
  protected:
    stringClass className;
  public:
    stringClass getClassName(void){return className;};
    virtual void getInput()=0;
    virtual void display()=0;
    virtual DataBlock passivate()=0;
    virtual unsigned long getKey()=0;
  };

class geoClass: public dataClass {
  protected:
    // used to rebuild spatial index if necessary
    rectangleClass rectangle;
  public:
    void setRectangle(int, int, int, int);
    void setRectangle(int, int);
    void setRectangle(rectangleClass&);
    rectangleClass getRectangle(void) {return rectangle;}
  };

class city: public geoClass {
  private:
    unsigned long cityCode;
    stringClass name, state;
    unsigned long population;
    static const Signature signature;
  public:
    city(void);
    city(DataBlock&);
    city(unsigned long, stringClass&, stringClass&, unsigned long);
    city(city&);
    city& operator= (city&);
    void getInput(void);
    void display(void);
    DataBlock passivate(void);
    unsigned long getKey() {return cityCode;};
    };

class weather: public geoClass {
  private:
    unsigned long cityCode;
    dateClass date;
```

```
    char maxTemp, minTemp,
    unsigned char chanceOfRain, chanceOfSnow,
                  tornadoWng, tornadoWatch,
                  tstormWng, tstormWatch;
    static const Signature signature;
  public:
    weather(void);
    weather(DataBlock&);
    weather(unsigned long, dateClass&, char, char,
            unsigned char, unsigned char,
            unsigned char, unsigned char,
            unsigned char, unsigned char);
    weather(weather&);
    weather& operator=(weather&);
    void getInput(void);
    void display(void);
    DataBlock passivate(void);
    unsigned long getKey() {return cityCode;};
    };

class path: public geoClass {
  private:
    //to ensure that a unique path exist between two cities
    //maybe the two city codes is enough check on that
    unsigned long pathCode;
    stringClass name;
    pathTypeEnum pathType;
    unsigned long fromCity, toCity;
    unsigned int distance;
    static const Signature signature;
  public:
    path(void);
    path(DataBlock&);
    path(unsigned long, stringClass&, pathTypeEnum,
         unsigned long, unsigned long, unsigned int);
    path(path&);
    path& operator=(path&);
    void getInput(void);
    void display(void);
    DataBlock passivate(void);
    unsigned long getKey() {return pathCode;}
    };

class conveyance: public dataClass {
  private:
    unsigned long convCode;
    stringClass description;
    convTypeEnum type;
    //number of seats for personnel
    unsigned int numOfSeats,
    //maximum weight of things transported
    weight,
    //maximum volume of things transported
    volume,
    //average speed
    avrgSpeed,
    //maximum autonomy, miles with no refueling
    autonomy;
    unsigned long locatedAt;
    static const Signature signature;
```

```
  public:
    conveyance(void);
    conveyance(DataBlock&);
    conveyance(unsigned long, stringClass&, convTypeEnum,
                        unsigned int, unsigned int, unsigned int,
                        unsigned int, unsigned int, unsigned long);
    conveyance(conveyance&);
    conveyance& operator= (conveyance&);
    void getInput(void);
    void display(void);
    DataBlock passivate(void);
    unsigned long getKey() {return convCode;};
    };

class cargo: public dataClass {
  private:
    unsigned long cargoCode;
    stringClass description;
    cargoTypeEnum type;
    //if personnel, number of people travelling
    unsigned int numOfPeople,
    weight,
    volume,
    unsigned long origin, destination;
    dtClass accepted, earliestArrival, latestArrival;
    static const Signature signature;
  public:
    cargo(void);
    cargo(DataBlock&);
    cargo(unsigned long, stringClass&, cargoTypeEnum,
              unsigned int, unsigned int, unsigned int,
              unsigned long, unsigned long,
              dtClass, dtClass, dtClass);
    cargo(cargo&);
    cargo& operator= (cargo&);
    void getInput(void);
    void display(void);
    DataBlock passivate(void);
    unsigned long getKey() {return cargoCode;};
    };

class location: public dataClass {
  private:
    unsigned long locationCode, cityCode;
    locationTypeEnum type;
    static const Signature signature;
  public:
    location(void);
    location(DataBlock&);
    location(unsigned long, unsigned long, locationTypeEnum);
    location(location&);
    location& operator= (location&);
    void getInput(void);
    void display(void);
    DataBlock passivate(void);
    unsigned long getKey() {return locationCode;};
    };

class constraint: public dataClass {
  private:
```

```
    constOnEnum target;
    constTypeEnum type;
    unsigned long targetCode;
    int minimum, maximum;
    static const Signature signature;
  public:
    constraint(void);
    constraint(DataBlock&);
    constraint(constOnEnum, constTypeEnum, unsigned long, int, int);
    constraint(constraint&);
    constraint& operator=(constraint&);
    void getInput(void);
    void display(void);
    DataBlock passivate(void);
    unsigned long getKey() {return targetCode;}
    };

#endif
```

```
// DATACL.CPP
// This program handles the different kinds of data used for
// transportation scheduling specification
//
// Felipe Guacache
// Department of Computer Science
// Oklahoma State University
// Stillwater, Oklahoma
// Last modified: 08/26/95
//

#include "datacl.h"

#ifndef __IOMANIP_H
  #include <iomanip.h>
#endif

#ifndef __STRSTREA_H
  #include <strstream.h>
#endif

// character strings with constents of enumerated types
char *pathTypeStr[] = {"Highway",
                       "Road",
                       "Airway",
                       "Railroad",
                       "Sea Route"
                       };

char *locationTypeStr[] = { "Depot",
                            "Airport"
                            "Train Station",
                            "Sea Port"
                            };

char *cargoTypeStr[] = { "Miscellaneous",
                         "Personnel",
                         "Food",
                         "Frozen Food",
                         "Canned Food",
                         "Printed Matter",
                         "Equipment",
                         "Heavy Machinery",
                         "Fire Arms",
                         "Waste"
                         };

char *convTypeStr[] = { "Plane",
                        "Vehicle",
                        "Truck",
                        "Train",
                        "Ship"
                        };

char *constTypeStr[] = { "Weight",
                         "Volume",
                         "Temperature",
                         "Speed",
                         "No Waste Allowed"
                         };
```

```
char *constOnStr[] = { "City",
                       "Path",
                       "Conveyance",
                       "Cargo"
                       };

// define signatures for the different classes
const Signature city::signature =       0x0000000F;
const Signature path::signature =       0x000000F0;
const Signature weather::signature =    0x00000F00;
const Signature conveyance::signature = 0x0000F000;
const Signature cargo::signature =      0x000F0000;
const Signature location::signature =   0x00F00000;
const Signature constraint::signature = 0x0F000000;


// geoClass member functions
// sets the rectangles coordinates based on four points
void geoClass::setRectangle(int x, int y, int z, int w) {

  rectangle = rectangleClass(x, y, z, w);
  }

// sets the rectangle coordinates based on two points
void geoClass::setRectangle(int x, int y) {

  rectangle = rectangleClass(x,y);
  }

// sets the rectangle based on a rectangle
void geoClass::setRectangle(rectangleClass& r) {

  rectangle = r;
  }

//city member functions
//default constructor
city::city(void) {

  className = "City";
  cityCode = 0L;
  population = 0L;
  }

// conversion constructor
city::city(unsigned long cc, stringClass& n, stringClass& s, unsigned long p) {

  className = "CITY";
  cityCode = cc;
  name = n;
  state = s;
  population = p;
  }

// conversion constructor
city::city(DataBlock& db) {
  char *buf, nm[128], st[128];
  int x1, y1, x2, y2;
```

```
  className = "CITY";
  buf = (char *) (void *) db;
  istrstream stream(buf, db.GetSize());
  stream >> x1 >> y1 >> x2 >> y2 >> cityCode >> nm >> st >> population;
  rectangle.setCoordinates(x1, y1, x2, y2);
  name = nm;
  state = st;
  }

// copy constructor
city::city(city& c1) {

  className = "CITY";
  rectangle = c1.rectangle;
  cityCode = c1.cityCode;
  name = c1.name;
  state = c1.state;
  population = c1.population;
  }

// assignment operator
city& city::operator=(city& c1) {

  rectangle = c1.rectangle;
  cityCode = c1.cityCode;
  name = c1.name;
  state = c1.state;
  population = c1.population;
  return c1;
  }

// gets data from standard input
void city::getInput(void) {
  unsigned long code, pop;
  char n[128], s[128];

  cout << endl << "City Code: ";
  cin >> code;
  cout << endl << "Name: ";
  cin >> n;
  cout << endl << "State: ";
  cin >> s;
  cout << endl << "Population:";
  cin >> pop;
  *this = city (code, n, s, pop);
  }

// outputs city info
void city::display(void) {
  cout << "City Code:  " << cityCode << endl
       << "City Name   " << name << endl
       << "Sate:       " << state << endl
       << "Population: " << population;
  }

// writes the city info into a DataBlock
DataBlock city::passivate(void) {
  char buf[512];
  ostrstream stream(buf, 512);
```

```
  memset(buf,0,512);
  stream << rectangle.getX1() << ' ' << rectangle.getY1() << ' '
         << rectangle.getX2() << ' ' << rectangle.getY2() << ' '
         << cityCode << ' '
         << name << ' ' << state << ' '
         << population;
  return DataBlock(signature, strlen(buf), (void *) buf);
  }

// weather member functions
// default constructor
weather::weather(void) {

  className = "WEATHER";
  cityCode = 0L;
  maxTemp = 0;
  minTemp = 0;
  chanceOfRain = 0;
  chanceOfSnow = 0;
  tornadoWng = 0;
  tornadoWatch = 0;
  tstormWng = 0;
  tstormWatch = 0;
  }

//conversion constructor
weather::weather(unsigned long cc, dateClass& d1, char mxt, char mnt,
                 unsigned char cr, unsigned char cs,
                 unsigned char twg, unsigned char twc,
                 unsigned char tswg, unsigned char tswc) {


  className = "WEATHER";
  cityCode = cc;
  date = d1;
  maxTemp = mxt;
  minTemp = mnt;
  chanceOfRain = cr;
  chanceOfSnow = cs;
  tornadoWng = twg;
  tornadoWatch = twc;
  tstormWng = tswg;
  tstormWatch = tswc;
  }

// conversion constructor
weather::weather(DataBlock& db) {
  char *buf;
  char dateStr[12];
  int x1, y1, x2, y2,
     mx, mn, cr, cs,
     twg, twc, tswg, tswc;

  className = "WEATHER";
  buf = (char *) (void *) db;
  istrstream stream(buf, db.GetSize());
  stream >> x1 >> y1 >> x2 >> y2
         >> cityCode >> dateStr >> mx >> mn
         >> cr >> cs
         >> twg >> twc
```

```
            >> tswg >> tswc;
  rectangle.setCoordinates(x1,y1,x2,y2);
  date = dateClass(dateStr);
  maxTemp = mx;
  minTemp = mn;
  chanceOfRain = cr;
  chanceOfSnow = cs;
  tornadoWng = twg;
  tornadoWatch = twc;
  tstormWng = tswg;
  tstormWatch = tswc;
  }

// copy constructor
weather::weather(weather& w1) {

  className = "WEATHER";
  rectangle = w1.rectangle;
  cityCode = w1.cityCode;
  date = w1.date;
  maxTemp = w1.maxTemp;
  minTemp = w1.minTemp;
  chanceOfRain = w1.chanceOfRain;
  chanceOfSnow = w1.chanceOfSnow;
  tornadoWng = w1.tornadoWng;
  tornadoWatch = w1.tornadoWatch;
  tstormWng = w1.tstormWng;
  tstormWatch = w1.tstormWatch;
  }

// assignment operator
weather& weather::operator=(weather& w1) {

  rectangle = w1.rectangle;
  cityCode = w1.cityCode;
  date = w1.date;
  maxTemp = w1.maxTemp;
  minTemp = w1.minTemp;
  chanceOfRain = w1.chanceOfRain;
  chanceOfSnow = w1.chanceOfSnow
  tornadoWng = w1.tornadoWng;
  tornadoWatch = w1.tornadoWatch;
  tstormWng = w1.tstormWng;
  tstormWatch = w1.tstormWatch;
  return w1;
  }

// gets data from standard input
void weather::getInput(void) {
  unsigned long cc;
  char  dateStr[128];
  int mxt, mnt;
  unsigned int cr, cs;
  unsigned char twg, twc, tswg, tswc;

  cout <<endl << "City Code: ";
  cin >> cc;
  cout << endl << "Date: ";
  cin >> dateStr;
  dateClass d1((const char *)dateStr);
```

```
  cout << endl << "Maximum Temperature: ";
  cin >> mxt;
  cout << endl << "Minimum Temperature: "
  cin >> mnt;
  cout << endl << "Chance of Rain (%): ";
  cin >> cr;
  cout << endl << "Chance of Snow (%): ";
  cin >> cs;
  cout << endl << "Tornado Warning (Y/N): ";
  cin >> twg;
  cout << endl << "Tornado Watch (Y/N): ";
  cin >> twc;
  cout << endl << "Thunder Storm Warning (Y/N): ";
  cin >> tswg;
  cout << endl << "Thunder Storm Watch (Y/N): ";
  cin >> tswc;
  *this = weather(cc, d1, mxt, mnt, cr, cs, twg, twc, tswg, tswc);
  }

// outputs weather info
void weather::display(void) {

  cout << "City Code: " << cityCode << endl
       << "Date:      " << date << endl
       << "Maximum Temperature: " << (int)maxTemp << endl
       << "Minimum Temperature: " << (int)minTemp << endl
       << "Chance of Rain: " << (int) chanceOfRain << "%" << endl
       << "Chance of Snow: " << (int) chanceOfSnow << "%" << endl
       << "Tornado Warning: " << (int)tornadoWng << endl
       << "Tornado Watch: " << (int)tornadoWatch << endl
       << "Thunder Storm Warning: " << (int)tstormWng << endl
       << "Thunder Storm Watch: " << (int)tstormWatch << endl;
  }

// writes the weather info into a DataBlock
DataBlock weather::passivate(void) {
  char buf[512];
  ostrstream stream(buf, 512);

  memset(buf,0,512);
  stream << rectangle.getX1() << ' ' << rectangle.getY1() << ' '
         << rectangle.getX2() << ' ' << rectangle.getY2() << ' '
         << cityCode << ' '
         << date << ' '
         << (int)maxTemp << ' ' << (int)minTemp << ' '
         << (int)chanceOfRain << ' ' << (int)chanceOfSnow << ' '
         << (int)tornadoWng << ' ' << (int)tornadoWatch << ' '
         << (int)tstormWng << ' ' << (int)tstormWatch;
  return DataBlock(signature, strlen(buf), (void *) buf);
  }

// path member functions
// default constructor
path::path(void) {

  className = "PATH";
  pathCode = 0L;
  pathType = _highWay;
  fromCity = 0L;
  toCity = 0L;
```

```
  distance = 0;
  }

//conversion constructor
path::path(unsigned long cd, stringClass& nm,pathTypeEnum pt,
           unsigned long fc, unsigned long tc, unsigned int dist) {

  className = "PATH";
  pathCode = cd;
  name = nm;
  pathType = pt;
  fromCity = fc;
  toCity = tc;
  distance = dist;
  }

// conversion constructor
path::path(DataBlock& db) {
  char *buf;
  int x1, y1, x2, y2, pt;
  char nm[128];


  className = "WEATHER";
  buf = (char *) (void *) db;
  istrstream stream(buf, db.GetSize());
  stream >> x1 >> y1 >> x2 >> y2
         >> pathCode >> nm >> pt
         >> fromCity >> toCity >> distance;
  rectangle.setCoordinates(x1,y1,x2,y2);
  name = nm;
  pathType = (pathTypeEnum) pt;
  }

//copy constructor
path::path(path& p1) {

  className = "PATH";
  rectangle = p1.rectangle;
  pathCode = p1.pathCode;
  name = p1.name;
  pathType = p1.pathType;
  fromCity = p1.fromCity;
  toCity = p1.toCity;
  distance = p1.distance;
  }

// assignment operator
path& path::operator=(path& p1) {

  rectangle = p1.rectangle;
  pathCode = p1.pathCode;
  name = p1.name;
  pathType = p1.pathType;
  fromCity = p1.fromCity;
  toCity = p1.toCity;
  distance = p1.distance;
  return p1;
  }
```

```
// gets data from standard input
void path::getInput(void) {
  unsigned long cd, fc, tc;
  char nm[128];
  unsigned int pt, dist;
  pathTypeEnum ptt;

  cout << endl << "Code: ";
  cin >> cd;
  cout << endl << "Name: ";
  cin >> nm;
  cout << endl << "Path Type: ";
  cin >> pt;
  ptt = (pathTypeEnum) pt;
  cout << endl << "From City: ";
  cin >> fc;
  cout << endl << "To City   ";
  cin >> tc;
  cout << endl << "Distance:  ";
  cin >> dist;
  *this = path(cd, nm, ptt, fc, tc, dist);
  }

// outputs path info
void path::display(void) {

  cout << "Code: " << pathCode << endl
       << "Name: " << name << endl
       << "Path Type: " << pathTypeStr[pathType] << endl
       << "From City: " << fromCity << endl
       << "To City:   " << toCity << endl
       << "Distance:  " << distance << endl;
  }

// writes the path info into a DataBlock
DataBlock path::passivate(void) {
  char buf[512];
  ostrstream stream(buf, 512);

  memset(buf,0,512);
  stream << rectangle.getX1() << ' ' << rectangle.getY1() << ' '
         << rectangle.getX2() << ' ' << rectangle.getY2() << ' '
         << pathCode << ' '
         << name << ' ' << pathType << ' '
         << fromCity << ' ' << toCity << ' '
         << distance;
  return DataBlock(signature, strlen(buf), (void *) buf);
  }

// conveyance member functions
// default constructor
conveyance::conveyance(void) {

  className = "CONVEYANCE";
  convCode = 0;
  type = _vehicle;
  numOfSeats = 0;
  weight = 0;
  volume = 0;
  avrgSpeed = 0;
```

```
  autonomy = 0;
  locatedAt = 0;
  }

// conversion constructor
conveyance::conveyance(unsigned long cc, stringClass& desc, convTypeEnum tp,
                       unsigned int ns, unsigned int wt,
                       unsigned int vol, unsigned int avgs,
                       unsigned int aut, unsigned long la) {

  className = "CONVEYANCE";
  convCode = cc;
  description = desc;
  type = tp;
  numOfSeats = ns;
  weight = wt;
  volume = vol;
  avrgSpeed = avgs;
  autonomy = aut;
  locatedAt = la;
  }

// conversion constructor
conveyance::conveyance(DataBlock& db) {
  char *buf;
  int ct;
  char desc[128];

  className = "CONVEYANCE";
  buf = (char *) (void *) db;
  istrstream stream(buf, db.GetSize());
  stream >> convCode >> desc >> ct
         >> numOfSeats >> weight >> volume
         >> avrgSpeed >> autonomy >> locatedAt;
  description = desc;
  type = (convTypeEnum) ct;
  }

// copy constructor
conveyance::conveyance(conveyance& c1) {

  className = "CONVEYANCE";
  convCode = c1.convCode;
  description = c1.description;
  type = c1.type;
  numOfSeats = c1.numOfSeats;
  weight = c1.weight;
  volume = c1.volume;
  avrgSpeed = c1.avrgSpeed;
  autonomy = c1.autonomy;
  locatedAt = c1.locatedAt;
  }

//assignment operator
conveyance& conveyance::operator=(conveyance& c1) {

  convCode = c1.convCode;
  description = c1.description;
  type = c1.type;
  numOfSeats = c1.numOfSeats;
```

```
  weight = c1.weight;
  volume = c1.volume;
  avrgSpeed = c1.avrgSpeed;
  autonomy = c1.autonomy;
  locatedAt = c1.locatedAt;
  return c1;
  }

// gets data from standard input
void conveyance::getInput(void) {
  unsigned long cc, la;
  char desc[128];
  unsigned int tp, nos, wt, vol, avgs, aut;
  convTypeEnum tpp;

  cout << endl << "Conveyance Code: ";
  cin >> cc;
  cout << endl << "Description:     ";
  cin >> desc;
  cout << endl << "Type: ";
  cin >> tp;
  tpp = (convTypeEnum) tp;
  cout << endl << "Number of Seats: ";
  cin >> nos;
  cout << endl << "Maximum Weight: ";
  cin >> wt;
  cout << endl << "Maximum Volume: ";
  cin >> vol;
  cout << endl << "Average Speed:  ";
  cin >> avgs;
  cout << endl << "Autonomy: ";
  cin >> aut;
  cout << endl << "Located at: ";
  cin >> la;
  *this = conveyance(cc, desc, tpp, nos, wt, vol, avgs, aut, la);
  }

// outputs conveyance info
void conveyance::display(void) {

  cout << "Conveyance Code: " << convCode << endl
       << "Description:     " << description << endl
       << "Type: " << convTypeStr[type] << endl
       << "Number of Seats: " << numOfSeats << endl
       << "Maximum Weight: " << weight << endl
       << "Maximum Volume: " << volume << endl
       << "Average Speed:  " << avrgSpeed << endl
       << "Autonomy: " << autonomy << endl
       << "Located at: " << locatedAt << endl;
  }

// writes the conveyance info into a DataBlock
DataBlock conveyance::passivate(void) {
  char buf[512];
  ostrstream stream(buf, 512);

  memset(buf,0,512);
  stream << convCode << ' ' << description << ' '
         << type << ' ' << numOfSeats << ' '
         << weight << ' ' << volume << ' '
```

```
            << avrgSpeed  << ' ' << autonomy << ' '
            << locatedAt;
  return DataBlock(signature, strlen(buf), (void *) buf);
  }

// ****************
// Need to add location to cargo info
// ********************
// cargo member functions
// default constructor
cargo::cargo(void) {

  className = "CARGO";
  cargoCode = 0L;
  type = _miscellaneous;
  numOfPeople = 0;
  weight = 0;
  volume = 0;
  origin = 0L;
  destination = 0L;
  }

// conversion constructor
cargo::cargo(unsigned long cc, stringClass& desc, cargoTypeEnum tp,
             unsigned int np, unsigned int wt, unsigned int vol,
             unsigned long ori, unsigned long dest,
             dtClass ac, dtClass ea, dtClass la) {

  className = "CARGO";
  cargoCode = cc;
  description = desc;
  type = tp;
  numOfPeople = np;
  weight = wt;
  volume = vol;
  origin = ori;
  destination = dest;
  accepted = ac;
  earliestArrival = ea;
  latestArrival = la;
  }

// conversion constructor
cargo::cargo(DataBlock& db) {
  char *buf;
  int ct;
  char desc[128], ac[128], ea[128], la[128];


  className = "CARGO";
  buf = (char *) (void *) db;
  istrstream stream(buf, db.GetSize());
  stream >> cargoCode >> desc >> ct
         >> numOfPeople >> weight >> volume
         >> origin >> destination
         >> ac >> ea >> la;
  description = desc;
  type = (cargoTypeEnum) ct;
  buf = strchr(ac, ',');
  *buf = '\x0';
```

```
  accepted = dtClass(buf+1,ac);
  buf = strchr(ea, ':');
  *buf = '\x0';
  earliestArrival = dtClass(buf+1,ea);
  buf = strchr(la, ':');
  *buf = '\x0';
  latestArrival = dtClass(buf+1,la);
  }

// copy constructor
cargo::cargo(cargo& c1) {

  className = "CARGO";
  cargoCode = c1.cargoCode;
  description = c1.description;
  type = c1.type;
  numOfPeople = c1.numOfPeople;
  weight = c1.weight;
  volume = c1.volume;
  origin = c1.origin;
  destination = c1.destination;
  }

// assignment operator
cargo& cargo::operator=(cargo& c1) {

  cargoCode = c1.cargoCode;
  description = c1.description;
  type = c1.type;
  numOfPeople = c1.numOfPeople;
  weight = c1.weight;
  volume = c1.volume;
  origin = c1.origin;
  destination = c1.destination;
  accepted = c1.accepted;
  earliestArrival = c1.earliestArrival;
  latestArrival = c1.latestArrival;
  return c1;
  }

// gets data from standard input
void cargo::getInput(void) {
  unsigned long cc;
  char desc[128];
  cargoTypeEnum tpp;
  unsigned int tp, np, wt, vol;
  unsigned long ori, dest;
  char  acDate[128], acTime[128],
        eaDate[128], eaTime[128],
        laDate[128], laTime[128];


  cout << endl << "Cargo Code:  ";
  cin >> cc;
  cout << endl << "Description:  ";
  cin >> desc;
  cout << endl << "Type        ";
  cin >> tp;
```

```
  tpp = (cargoTypeEnum) tp;
  cout << endl << "Number of People: ";
  cin >> np;
  cout << endl << "Weight: ";
  cin >> wt;
  cout << endl << "Volume: ";
  cin >> vol;
  cout << endl << "Original Location: ";
  cin >> ori;
  cout << endl << "Destination: ";
  cin >> dest;
  cout << endl << "Accepted Date: ";
  cin >> acDate;
  cout << endl << "Accepted Time: ";
  cin >> acTime;
  cout << endl << "Earliest Arrival Date: ";
  cin >> eaDate;
  cout << endl << "Earliest Arrival Time: ";
  cin >> eaTime;
  cout << endl << "Latest Arrival Date: ";
  cin >> laDate;
  cout << endl << "Latest Arrival Time: ";
  cin >> laTime;
  *this = cargo(cc, desc, tpp, np, wt, vol, ori, dest,
                dtClass((const char *)acTime, (const char *)acDate),
                dtClass((const char *)eaTime, (const char *)eaDate),
                dtClass((const char *)laTime, (const char *)laDate));
  }

// outputs cargo info
void cargo::display(void) {

  cout << "Cargo Code:  " << cargoCode << endl
       << "Description: " << description << endl
       << "Type :       " << cargoTypeStr[type] << endl
       << "Number of People: " << numOfPeople << endl
       << "Weight: " << weight << endl
       << "Volume: " << volume << endl
       << "Original Location: " << origin << endl
       << "Destination: " << destination << endl
       << "Accepted: " << accepted << endl
       << "Earliest Arrival: " << earliestArrival << endl
       << "Latest Arrival: " << latestArrival << endl;
  }

// writes the cargo info into a DataBlock
DataBlock cargo::passivate(void) {
  char buf[512];
  ostrstream stream(buf, 512);

  memset(buf,0,512);
  stream << cargoCode << ' ' << description << ' '
         << type << ' ' << numOfPeople << ' '
         << weight << ' ' << volume << ' '
         << origin << ' ' << destination << ' '
         << accepted << ' ' << earliestArrival << ' '
         << latestArrival;
  return DataBlock(signature, strlen(buf), (void *) buf);
  }
```

```
// location member functions
// default constructor
location::location(void) {

  className = "LOCATION";
  locationCode = 0L;
  cityCode = 0L;
  type = _depot;
  }

// conversion constructor
location::location(unsigned long lc, unsigned long cc, locationTypeEnum tp) {

  className = "LOCATION";
  locationCode = lc;
  cityCode = cc;
  type = tp;
  }

// conversion constructor
location::location(DataBlock& db) {
  char *buf;
  int lt;

  className = "LOCATION";
  buf = (char*) (void*) db;
  istrstream stream(buf, db.GetSize());
  stream >> locationCode >> cityCode >> lt;
  type = (locationTypeEnum) lt;
  }

// copy constructor
location::location(location& ll) {

  className = "LOCATION";
  locationCode = ll.locationCode;
  cityCode = ll.cityCode;
  type = ll.type;
  }

// assignment operator
location& location::operator= (location& ll) {

  locationCode = ll.locationCode;
  cityCode = ll.cityCode;
  type = ll.type;
  return ll;
  }

// gets data from standard input
void location::getInput(void) {
  unsigned long lc, cc;
  unsigned int tp;
  locationTypeEnum tpp;

  cout << endl << "Location Code: ";
  cin >> lc;
  cout << endl << "City Code: ";
  cin >> cc;
  cout << endl << "Location Type: ";
```

```
  cin >> tp;
  tpp = (locationTypeEnum) tp;
  *this = location (lc, cc, tpp);
  }

// outputs location info
void location::display(void) {

  cout << "Location Code: " << locationCode << endl
       << "City Code:     " << cityCode << endl
       << "Location Type: " << locationTypeStr[type] << endl;
  }

// writes the location info into a DataBlock
DataBlock location::passivate(void) {
  char buf[512];
  ostrstream stream(buf, 512);

  memset(buf,0,512);
  stream << locationCode << ' ' << cityCode << ' ' << type;
  return DataBlock(signature, strlen(buf), (void *) buf);
  }

// constraint member functions
// default constructor
constraint::constraint(void) {

  className = "CONSTRAINT";
  target = _city;
  type = _weight;
  targetCode = 0L;
  minimum = 0;
  maximum = 0;
  }

// conversion constructor
constraint::constraint(constOnEnum tg, constTypeEnum tp, unsigned long tc,
                       int mn, int mx) {

  className = "CONSTRAINT"
  target = tg;
  type = tp;
  targetCode = tc;
  minimum = mn;
  maximum = mx;
  }

// conversion constructor
constraint::constraint(DataBlock& db) {
  char *buf;
  int tg, ct;

  className = "CONSTRAINT";
  buf = (char*) (void*) db;
  istrstream stream(buf, db.GetSize());
  stream >> tg >> ct >> targetCode >> minimum >> maximum;
  target = (constOnEnum) tg;
  type = (constTypeEnum) ct;
  }
```

```
// copy constructor
constraint::constraint(constraint& c1) {

  className = "CONSTRAINT";
  target = c1.target;
  type = c1.type;
  targetCode = c1.targetCode;
  minimum = c1.minimum;
  maximum = c1.maximum;
  }

// assignment operator
constraint& constraint::operator=(constraint& c1) {

  target = c1.target;
  type = c1.type;
  targetCode = c1.targetCode;
  minimum = c1.minimum;
  maximum = c1.maximum;
  return c1;
  }

// gets data from standard input
void constraint::getInput(void) {
  int tg, ct, mn, mx;
  unsigned long tc;

  cout << endl << "Target: ";
  cin >> tg;
  cout << endl << "Constraint Type: ";
  cin >> ct;
  cout << endl << "Target Code: ";
  cin >> tc;
  cout << endl << "Minimum: ";
  cin >> mn;
  cout << endl << "Maximum: ";
  cin >> mx;
  *this = constraint((constOnEnum)tg, (constTypeEnum)ct, tc, mn, mx);
  }

// outputs constraint data
void constraint::display(void) {

  cout << "Target: " << constOnStr[target] << endl
       << "Constraint Type: " << constTypeStr[type] << endl
       << "Target Code: " << targetCode << endl
       << "Minimum: " << minimum << endl
       << "Maximum: " << maximum << endl;
  }

// writes the constraint info into a DataBlock
DataBlock constraint::passivate(void) {
  char buf[512];
  ostrstream stream(buf, 512);

  memset(buf,0,512);
  stream << target << ' ' << type << ' ' << targetCode << ' '
         << minimum << ' ' << maximum;
  return DataBlock(signature, strlen(buf), (void *) buf);
  }
```

```
// FILE LIBRARY -- datafile.h (v2.00 4/30/94)
//    Defines a file type for disk-based data files
// Copyright 1992, 1994 Scott Robert Ladd. All rights reserved
// Modified to run in a UNIX system by Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Last Modified: 08/31/95


#ifndef DATAFILE_H
#define DATAFILE_H

#include "persist.h"
#include "str.h"
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <unistd.h>

enum DataFileError
        {
        FE_EOF,
        FE_ALLOC,
        FE_GEN_TEMP,
        FE_INV_MODE,
        FE_NO_OPEN,
        FE_NO_WRT_FHDR,
        FE_NO_RD_FHDR,
        FE_NO_WRT_RHDR,
        FE_NO_RD_RHDR,
        FE_NO_WRT_REC
        FE_NO_RD_REC,
        FE_BAD_SEEK,
        FE_NO_OPEN_TEMP,
        FE_NO_REMOVE,
        FE_NO_RENAME,
        FE_NO_REWIND,
        FE_BAD_SIG
        };

enum DataFileMode
  {
  FM_NEW,
  FM_EXISTING,
  FM_APPEND,
  FM_TEMPORARY
  };

typedef unsigned long Signature;

struct RecordHeader {
  size_t RecSize;
  Signature RecSig;
  };

class DataFileBase {
  public:
        // constructor
```

```
    DataFileBase(const stringClass & name, DataFileMode m = FM_TEMPORARY);

    // destructor
    ~DataFileBase();

        // flush buffers
        // void Commit();

  protected:
    // data members
    DataFileMode Mode;
    stringClass  FileName;
        int    DOSHandle;
    int    DOSMode;
    int    DOSAttr;

        static const int StdModeNew;
        static const int StdModeExists;
        static const int StdModeAppend;
        static const int StdAttr;
        static const int StdAttrNew;
  };


//---------------------------------------------------------
// DataFileInput
//    A file for the sequential input of information
//---------------------------------------------------------

class DataFileInput : virtual public DataFileBase {

    public:
        DataFileInput(const stringClass & name)
                    : DataFileBase(name,FM_EXISTING) { }

        DataBlock Read() const;

        BooleanEnum Rewind();
        BooleanEnum Skip();
    };


//---------------------------------------------------------
// DataFileOutput
//    A file for the sequential output of information
//---------------------------------------------------------

class DataFileOutput : virtual public DataFileBase {
    public:
        DataFileOutput(const stringClass & name, DataFileMode m = FM_NEW)
                        DataFileBase(name,m) { }

        BooleanEnum Write(const DataBlock&);
    };

//---------------------------------------------------------
// DataFilePtr
//    A type used to locate information in a data file
//---------------------------------------------------------
```

```
// type of a data file ptr
typedef long DataFilePtr;

// constants
extern const DataFilePtr DFP_NULL;
extern const Signature SIG_DELETED;

//---------------------------------------------------------------
// DataFile
//    Defines a random access, I/O file
//---------------------------------------------------------------

struct DataFileHdr {
  DataFilePtr FirstEmpty;
  };

struct IORecordHeader : public RecordHeader {
  size_t     Size;
  DataFilePtr NextDeleted;
  };

class DataFile : public DataFileBase {
  public:
    // constructor
    DataFile(const stringClass & name, DataFileMode m = FM_EXISTING);

    // write a record
    DataFilePtr Write(const DataBlock& data);

    // read a record
    DataBlock Read() const;

    // delete a record
    void Delete();

    // remove blanks and wasted space
    void Compact(void (* func)(DataFilePtr ptr, const DataBlock & data) = NULL);

    // move to next record
    BooleanEnum Skip();

    // return to beginning of file
    void Rewind();

    // go to  position in file
    DataFilePtr Seek(DataFilePtr pos) const;

    // get current file position
    DataFilePtr CurrentPtr() const;

  protected:
    DataFileHdr Hdr;
  };

#endif
```

```
// FILE LIBRARY -- datafile.cpp (v2.00 4/30/94)
//    Defines a file type for disk-based data files
// Copyright 1992, 1994 Scott Robert Ladd. All rights reserved
// Modified to run in a UNIX system by Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Last Modified: 08/31/95

#include "datafile.h"

const DataFilePtr DFP_NULL    = -1;
const Signature   SIG_DELETED = 0xFFFFFFFF;

// This function returns the current file pointer of an open file handle

long tell(int handle) {
  long ptr = lseek(handle, 0L, SEEK_CUR);
  if (ptr == -1L) {
     cerr << "Bad file seek <-correct this" << endl;
     exit(1);
     }
  return ptr;
  }

//--------------------------------------------------------------
// DataFileBase
//    The base class for binary data files
//--------------------------------------------------------------

const int DataFileBase::StdModeNew    = O_CREAT | O_TRUNC | O_RDWR;
const int DataFileBase::StdModeExists = O_RDWR;
const int DataFileBase::StdModeAppend = O_APPEND | O_WRONLY;
const int DataFileBase::StdAttr       = 0;
const int DataFileBase::StdAttrNew    = S_IREAD | S_IWRITE;

DataFileBase::DataFileBase(const stringClass & name, DataFileMode m)
              : FileName(name) {

  // generate DosMode string
  Mode = m;

  switch (Mode)
    {
    case FM_NEW:
    case FM_TEMPORARY:
      DOSMode = StdModeNew;
      DOSAttr = StdAttrNew;
      break;

        case FM_EXISTING:
      DOSMode = StdModeExists;
          DOSAttr = StdAttr;
          break;

        case FM_APPEND:
          DOSMode = StdModeAppend;
          DOSAttr = StdAttr;
      break;

        default:
```

```
            //throw DataFileEx(FE_INV_MODE);
            cerr << "Invalid File Mode." << endl;
            exit(1);
          }

  // open file
  DOSHandle = open(FileName,DOSMode,DOSAttr);

  if (DOSHandle == -1) {
          //throw DataFileEx(FE_NO_OPEN);
          cerr << "Unable to Open file " << FileName << endl;
          exit(1);
          }
  }

DataFileBase::~DataFileBase()
  {
  // close up!
  close(DOSHandle);

  // if temp file, delete it!
  if (Mode == FM_TEMPORARY)
          remove(FileName);
  }


//-----------------------------------------------------------------
// DataFileInput
//    A file for the sequential input of information
//-----------------------------------------------------------------

DataBlock DataFileInput::Read() const {

    RecordHeader hdr;
    size_t n;

    // read record header
    n = read(DOSHandle,&hdr,sizeof(RecordHeader));

    if (n == 0) {
            //throw DataFileEx(FE_EOF);
            DataBlock db;
            return db;
            }

    if (n != sizeof(RecordHeader)) {
            //throw DataFileEx(FE_NO_RD_RHDR);
            cerr << "Invalid Record Header." << endl;
            exit(1);
            }

    // allocate buffer to hold data
    DataBlock db(hdr.RecSig,hdr.RecSize);

    if (!db.IsNull()) {
            // read data
            n = read(DOSHandle,(void *)db.hdr.RecSize);

            if (n != hdr.RecSize) {
              //throw DataFileEx(FE_NO_RD_REC);
```

```
                cerr << "Unable to read file record." << endl;
                exit(1);
                }
            }

    return db;
    }

BooleanEnum DataFileInput::Rewind() {

    size_t res = (size_t)lseek(DOSHandle,0,SEEK_SET);

    if (res == 0)
            return BOOL_TRUE;
    else
            return BOOL_FALSE;
    }

BooleanEnum DataFileInput::Skip() {

    RecordHeader hdr;
    size_t res;

    // read record header
    res = read(DOSHandle,&hdr,sizeof(RecordHeader));

    if (res != sizeof(RecordHeader))
            return BOOL_FALSE;

    // skip data
    res = (size_t)lseek(DOSHandle,hdr.RecSize,SEEK_CUR);

    if (res == 0)
            return BOOL_TRUE;
    else
            return BOOL_FALSE;
    }

//--------------------------------------------------------------
// DataFileOutput
//     A file for the sequential output of information
//--------------------------------------------------------------

BooleanEnum DataFileOutput::Write(const DataBlock& db) {


    // create record header
    RecordHeader hdr;
    hdr.RecSig  = db.GetSig();
    hdr.RecSize = db.GetSize();

    // store record header
    size_t n = write(DOSHandle,&hdr,sizeof(RecordHeader));

    if (n != sizeof(RecordHeader))
    return BOOL_FALSE;

    // store data
    n = write(DOSHandle,(void *)db,hdr.RecSize);
```

```
    if (n == hdr.RecSize)
            return BOOL_TRUE;
    else
            return BOOL_FALSE;
    }

//---------------------------------------------------------
// DataFile
//    Defines a random access, I/O file
//---------------------------------------------------------

// constructor
DataFile::DataFile(const stringClass & name, DataFileMode m)
                        DataFileBase(name,m) {

   size_t res;

   if ((m == FM_NEW) || (m == FM_TEMPORARY)) {
            Hdr.FirstEmpty = DFP_NULL;

            res = write(DOSHandle,&Hdr,sizeof(DataFileHdr));

            if (res != sizeof(DataFileHdr)) {
               //throw DataFileEx(FE_NO_WRT_FHDR);
               cerr << "Unable to write file header " << endl;
               exit(1);
               }
            }
   else {
            res = read(DOSHandle,&Hdr,sizeof(DataFileHdr));

            if (res != sizeof(DataFileHdr)) {
               //throw DataFileEx(FE_NO_RD_FHDR);
               cerr << "Unable to read file header " << endl;
               exit(1);
               }
            }
   }

// write a record
DataFilePtr DataFile::Write(const DataBlock& db) {

   // search for first open record that can hold this one
   IORecordHeader rechdr;
   DataFilePtr ptr;
   size_t n;
   long pos;

   if (Hdr.FirstEmpty == DFP_NULL) {
            // append new data to end of file
            pos = lseek(DOSHandle,0,SEEK_END);

            if (pos == -1L) {
               //throw DataFileEx(FE_BAD_SEEK);
               cerr << "File seek failed." << endl;
               exit(1);
               }

            ptr = tell(DOSHandle);
```

```
        rechdr.Size = db.GetSize();
        }
else {
        // start with first empty record
        DataFilePtr prev = DFP_NULL;

        ptr = Hdr.FirstEmpty;

        for (;;) {
           // set file pointer
           pos = lseek(DOSHandle,ptr,SEEK_SET);

           if (pos == -1L) {
                   //throw DataFileEx(FE_BAD_SEEK);
                   cerr << "File seek failed." << endl;
                   exit(1);
                   }

           // read record header
           n = read(DOSHandle,&rechdr,sizeof(IORecordHeader));

           if ((n != sizeof(IORecordHeader))
                   || (rechdr.RecSig != SIG_DELETED)) {
                   //throw DataFileEx(FE_NO_RD_RHDR);
                   cerr << "Unable to read record header " << endl;
                   exit(1);
                   }

           // is it big enough?
           if (rechdr.Size >= db.GetSize()) {
                   // go to previous record
                   if (prev == DFP_NULL) {
                      // change entry in header
                      Hdr.FirstEmpty = rechdr.NextDeleted;

                      // go to beginning of file
                      pos = lseek(DOSHandle,0,SEEK_SET);

                      if (pos == -1L) {
                              //throw DataFileEx(FE_BAD_SEEK);
                              cerr << "File seek failed." << endl;
                              exit(1);
                              }

                      // write new header
                      n = write(DOSHandle,&Hdr,sizeof(DataFileHdr));

                      if (n != sizeof(DataFileHdr)) {
                              //throw DataFileEx(FE_NO_WRT_FHDR);
                              cerr << "Unable to write file header " << endl;
                              exit(1);
                              }
                      }
                   else {
                      // adjust chain by putting next in prev
                      pos = lseek(DOSHandle,prev,SEEK_SET);

                      if (pos == -1L) {
                              //throw DataFileEx(FE_BAD_SEEK);
                              cerr << "File seek failed " << endl;
```

```
                exit(1);
                }

        // read previous record's header
        IORecordHeader prevhdr;

        n = read(DOSHandle,&prevhdr,sizeof(IORecordHeader));

        if (n != sizeof(IORecordHeader)) {
                //throw DataFileEx(FE_NO_RD_RHDR);
                cerr << "Unable to read record header." << endl;
                exit(1);
                }

        // change next deleted reference
        prevhdr.NextDeleted = rechdr.NextDeleted;

        // rewrite prev. header
        pos = lseek(DOSHandle,prev,SEEK_SET);

        if (pos == -1L) {
                //throw DataFileEx(FE_BAD_SEEK);
                cerr << "File seek failed." << endl;
                exit(1);
                }

        // read previous record's header
        n = write(DOSHandle,&prevhdr,sizeof(IORecordHeader));

        if (n != sizeof(IORecordHeader)) {
                //throw DataFileEx(FE_NO_WRT_RHDR);
                cerr << "Unable to write record header " << endl;
                exit(1);
                }
        }

      // set pointer to beginning of ptr record
      pos = lseek(DOSHandle,ptr,SEEK_SET);

      if (pos == -1L) {
        //throw DataFileEx(FE_BAD_SEEK);
        cerr << "File seek failed." << endl;
        exit(1);
        }

      break;
      }

// save pointer
prev = ptr;

if (rechdr.NextDeleted == DFP_NULL) {
        // append new data to end of file
        pos = lseek(DOSHandle,0,SEEK_END);

        if (pos == -1L) {
          //throw DataFileEx(FE_BAD_SEEK);
          cerr << "File seek failed." << endl;
          exit(1);
          }
```

```
                    ptr = tell(DOSHandle);

                    rechdr.Size = db.GetSize();

                    break;
                    }
                else
                    ptr  = rechdr.NextDeleted;
                }
            }

    rechdr.RecSig     = db.GetSig();
    rechdr.RecSize    = db.GetSize();
    rechdr.NextDeleted = DFP_NULL;

    // store signature
    n = write(DOSHandle,&rechdr,sizeof(IORecordHeader));

    if (n != sizeof(IORecordHeader)) {
            //throw DataFileEx(FE_NO_WRT_RHDR);
            cerr << "Unable to write record header " << endl;
            exit(1);
            }

    // store data
    n = write(DOSHandle,(void *)db,rechdr.RecSize);

    if (n != rechdr.RecSize) {
            //throw DataFileEx(FE_NO_WRT_REC);
            cerr << "Unable to write file record." << endl;
            exit(1);
            }

    // return location data was written to
    return ptr;
    }

// read a record
DataBlock DataFile::Read() const {

    size_t n;
    long   pos;
    IORecordHeader rechdr;

    while (1) {
            // read record header
            n = read(DOSHandle,&rechdr,sizeof(IORecordHeader));

            if (n == 0) { // eof
              //throw DataFileEx(FE_EOF);
//            cerr << "End of File Encountered " << endl;
//            exit(1);
              DataBlock db;
              return db;
              }

            if (n != sizeof(IORecordHeader)) {
              //throw DataFileEx(FE_NO_RD_RHDR);
              cerr << "Unable to read record header " << endl;
```

```
            exit(1);
            }

         // we've found an in-use record
         if (rechdr.RecSig != SIG_DELETED)
            break;

         // skip to next record
         pos = lseek(DOSHandle,rechdr.Size,SEEK_CUR);

         if (pos == -1L) {
            //throw DataFileEx(FE_BAD_SEEK);
            cerr << "File seek failed." << endl;
            exit(1);
            }
         }

   // allocate buffer to hold data
   DataBlock db(rechdr.RecSig, rechdr.RecSize);

   if (!db.IsNull()) {
         // read data
         n = read(DOSHandle,(void *)db.rechdr.RecSize);

         if (n != rechdr.RecSize) {
            //throw DataFileEx(FE_NO_RD_REC);
            cerr << "Unable to read file record." << endl;
            exit(1);
            }

         // skip over any "waste" characters
         if (rechdr.RecSize < rechdr.Size) {
            pos = lseek(DOSHandle,rechdr.Size - rechdr.RecSize, SEEK_CUR);

            if (pos == -1L) {
                  //throw DataFileEx(FE_BAD_SEEK);
                  cerr << "File seek failed " << endl;
                  exit(1);
                  }
            }
         }


   // outa here
   return db;
   }

// delete a record
void DataFile::Delete(void) {

   size_t n;
   long  pos;
   IORecordHeader rechdr;

   // save this position
   DataFilePtr curptr = tell(DOSHandle);

   // make sure we're not in the header
   if (curptr < sizeof(DataFileHdr)) {
         //throw DataFileEx(FE_BAD_SEEK);
```

```
            cerr << "File seek failed." << endl;
            exit(1);
            }

// read header
n = read(DOSHandle,&rechdr,sizeof(IORecordHeader));

if (n != sizeof(IORecordHeader)) {
            //throw DataFileEx(FE_NO_RD_RHDR);
            cerr << "Unable to read record header " << endl;
            exit(1);
            }

// mark as deleted
if (rechdr.RecSig != SIG_DELETED) {
            // update record header
            rechdr.RecSig      = SIG_DELETED;
            rechdr.NextDeleted = Hdr.FirstEmpty;

            // write record header
            pos = lseek(DOSHandle,curptr,SEEK_SET);

            if (pos == -1L) {
               //throw DataFileEx(FE_BAD_SEEK);
               cerr << "File seek failed." << endl;
               exit(1);
               }

            n = write(DOSHandle,&rechdr,sizeof(IORecordHeader));

            if (n != sizeof(IORecordHeader)) {
               //throw DataFileEx(FE_NO_WRT_RHDR);
               cerr << "Unable to write record header " << endl;
               exit(1);
               }

            // modify header
            Hdr.FirstEmpty = curptr;

            pos = lseek(DOSHandle,0,SEEK_SET);

            if (pos == -1L) {
               //throw DataFileEx(FE_BAD_SEEK);
               cerr << "File seek failed." << endl;
               exit(1);
               }

            n = write(DOSHandle,&Hdr,sizeof(DataFileHdr));

            if (n != sizeof(DataFileHdr)) {
               //throw DataFileEx(FE_NO_WRT_FHDR);
               cerr << "Unable to write file header " << endl;
               exit(1);
               }
            }

// move back to start of deleted record
pos = lseek(DOSHandle,curptr,SEEK_SET);

if (pos == -1L) {
```

```
            //throw DataFileEx(FE_BAD_SEEK);
            cerr << "File seek failed." << endl;
            exit(1);
            }
    }

// remove blanks and wasted space
void DataFile::Compact(void (* func)(DataFilePtr ptr, const DataBlock& db)) {

    // generate temporary file name
    char *tname = "datafile.tmp";

    //sprintf(tname,"%lx.TMP",time(NULL));

    // open temporary file
    int newfile = open(tname, O_CREAT | O_RDWR,
                                    S_IREAD | S_IWRITE);

    if (newfile == -1) {
            //throw DataFileEx(FE_NO_OPEN_TEMP);
            cerr << "Unable to open temporal file." << endl;
            exit(1);
            }

    // read and copy header
    long pos;
    size_t n;

    pos = lseek(DOSHandle,0,SEEK_SET);

    if (pos == -1L) {
            //throw DataFileEx(FE_BAD_SEEK);
            cerr << "File seek failed." << endl;
            exit(1);
            }

    n = read(DOSHandle,&Hdr,sizeof(DataFileHdr));

    if (n != sizeof(DataFileHdr)) {
            //throw DataFileEx(FE_NO_RD_FHDR)
            cerr << "Unable to read file header " << endl;
            exit(1);
            }

    n = write(newfile,&Hdr,sizeof(DataFileHdr));

    if (n != sizeof(DataFileHdr)) {
            //throw DataFileEx(FE_NO_WRT_FHDR);
            cerr << "Unable to write file header." << endl;
            exit(1);
            }

    IORecordHeader rechdr;
    DataFilePtr ptr;
    char* buf

    // read each record
    for (;;) {
            // read record header
            n = read(DOSHandle,&rechdr,sizeof(IORecordHeader));
```

```
if (n == 0) break;

if (n != sizeof(IORecordHeader)) {
   //throw DataFileEx(FE_NO_RD_RHDR);
   cerr << "Unable to read record header." << endl;
   exit(1);
   }

if (rechdr.RecSig == SIG_DELETED) {
   // skip deleted records
   pos = lseek(DOSHandle,rechdr.Size,SEEK_CUR);

   if (pos == -1L) {
            //throw DataFileEx(FE_BAD_SEEK);
            cerr << "File seek failed." << endl;
            exit(1);
            }
   }
else {
   // allocate buffer to hold data
   buf = new char[rechdr.Size];

   if (buf == NULL) {
            //throw DataFileEx(FE_ALLOC);
            cerr << "File memory allocation failure " << endl;
            exit(1);
            }

   // read data
   n = read(DOSHandle,buf,rechdr.Size);

   if (n != rechdr.Size) {
            //throw DataFileEx(FE_NO_RD_REC);
            cerr << "Unable to read file record." << endl;
            exit(1);
            }

   // get output position in new file
   ptr = tell(newfile);

   // write header to new file
   rechdr.Size = rechdr.RecSize;

   n = write(newfile,&rechdr,sizeof(IORecordHeader));

   if (n != sizeof(IORecordHeader)) {
            //throw DataFileEx(FE_NO_WRT_RHDR);
            cerr << "Unable to write record header " << endl
            exit(1);
            }

   // write data to new file
   n = write(newfile,buf,rechdr.Size);

   if (n != rechdr.Size) {
            //throw DataFileEx(FE_NO_WRT_REC);
            cerr << "Unable to write file record." << endl;
            exit(1);
            }
```

```
                // call function
                if (func != NULL) {
                        DataBlock db(rechdr.RecSig,rechdr.RecSize,buf);
                        func(ptr,db);
                        }

                // delete buffer
                delete buf;
                }
            }

    // close files
    close(newfile);
    close(DOSHandle);

    // delete old file
    if (-1 == remove(FileName)) {
            //throw DataFileEx(FE_NO_REMOVE);
            cerr << "Unable to delete file." << endl;
            exit(1);
            }

    // rename new file
    if (-1 == rename(tname,FileName)) {
            //throw DataFileEx(FE_NO_RENAME);
            cerr << "Unable to rename file." << endl;
            exit(1);
            }

    // open newly-compacted file
    DOSHandle = open(FileName,DOSMode);

    if (DOSHandle == -1) {
            //throw DataFileEx(FE_NO_OPEN);
            cerr << "Unable to open file." << endl;
            exit(1);
            }

    // read file header
    n = read(DOSHandle,&Hdr,sizeof(DataFileHdr));

    if (n != sizeof(DataFileHdr)) {
            //throw DataFileEx(FE_NO_RD_FHDR);
            cerr << "Unable to read file header " << endl;
            exit(1);
            }
    }

// go to position in file
DataFilePtr DataFile::Seek(DataFilePtr pos) const {

    // make sure position is outside of header
    if (pos < sizeof(DataFileHdr)) {
            //throw DataFileEx(FE_BAD_SEEK);
            cerr << "File seek failed." << endl;
            exit(1);
            }

    // get current position
```

```
    DataFilePtr ptr = tell(DOSHandle);

    // move to new position
    long n = lseek(DOSHandle,pos,SEEK_SET);

    // check for error
    if (n == -1L) {
            //throw DataFileEx(FE_BAD_SEEK);
            cerr << "File seek failed." << endl;
            exit(1);
            }

    // done
    return ptr;
    }

// move to next record
BooleanEnum DataFile::Skip(void) {

    IORecordHeader rechdr;
    long pos;
    size_t n;

    // read record header
    n = read(DOSHandle,&rechdr,sizeof(IORecordHeader));

    if (n == 0)
            return BOOL_FALSE;

    if (n != sizeof(IORecordHeader)) {
            //throw DataFileEx(FE_NO_RD_RHDR);
            cerr << "Unable to read record header." << endl;
            exit(1);
            }

    // skip data
    pos = lseek(DOSHandle,rechdr.Size,SEEK_CUR);

    if (pos == -1L) {
            //throw DataFileEx(FE_BAD_SEEK);
            cerr << "File seek failed" << endl;
            exit(1);
            }

    return BOOL_TRUE;
    }

// return to beginning of file
void DataFile::Rewind(void) {

    long pos = lseek(DOSHandle,sizeof(DataFileHdr),SEEK_SET);

    if (pos == -1L) {
            //throw DataFileEx(FE_NO_REWIND);
            cerr << "Unable to rewind file." << endl;
            exit(1);
            }
    }

// get current file position
```

```
inline DataFilePtr DataFile::CurrentPtr(void) const {

  return tell(DOSHandle);
  }
```

```
// Program Name: FILE.H
// Two useful functions not available in UNIX
// Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Last Modified: 08/31/95


#ifndef FILE_H

#define FILE_H
#include "str.h"
#include <stdio.h>
#include <unistd.h>

Boolean fileExists(stringClass fname){

  if (access((const char *)fname, F_OK))
    return FALSE;
  else
    return TRUE;
}


stringClass getDirectory(void) {
  char *buf=NULL;

  return (stringClass) getcwd(buf, 512);
  }
#endif
```

```
// HASH TABLE LIBRARY -- hash.h (v2.00 4/23/94)
//    Defines the abstract base class for implementing a
//    hash table
// Copyright 1992,1994 Scott Robert Ladd. All rights reserved
// Modified to run in a UNIX system by Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Last Modified: 08/31/95


#ifndef HASH_H
#define HASH_H

#include "str.h"
#include "simple.h"
#include <stdlib.h>

// A type definition for types returned by HashTable::LookUp
// and HashBucket::FindEntry

typedef struct {
   BooleanEnum found;
   unsigned long data;
   } returnType;

inline size_t Hash(unsigned long x, size_t buckets)
            {
            return size_t(x % (unsigned long)(buckets));
            }

enum HashError
            {
            HE_ALLOC,    // fatal:  memory allocation failure
            HE_ZEROSIZE, // fatal:  created bucket w/ zero buckets
            HE_CORRUPTED, // fatal:  table / list has been corrupted
            HE_BADTYPES,  // fatal:  mismatched data types
            HE_TOOSMALL,  // warning: about very small table sizes
            HE_DUPEKEY,   // warning: about duplicate keys
            HE_NOTFOUND   // warning: key not found
            };


//---------------------------------
// HashEntry
//          Entries of HashBucket
//---------------------------------
struct HashEntry {

   HashEntry();
   HashEntry(const unsigned long& kx, const unsigned long& dx);
   HashEntry(const HashEntry& he);

   HashEntry& operator = (const HashEntry& he);
   int operator == (const HashEntry& he);
   int operator != (const HashEntry& he);

   HashEntry *Prev;
   HashEntry *Next;
   unsigned long Key;
   unsigned long Data;
```

```
};

//-----------------------------------------
// HashBucket
//          a bucket for a HashTable
//-----------------------------------------
class HashBucket {
  public:
          HashBucket();
          HashBucket(const HashBucket& hb);

          ~HashBucket();
          HashBucket & operator = (const HashBucket& hb);

          void AddEntry(const unsigned long& kx, const unsigned long& dx);
          BooleanEnum DelEntry(const unsigned long& kx);
          returnType FindEntry(const unsigned long& kx) const;
          BooleanEnum Traverse(BooleanEnum (*func)(const unsigned long& kx,
                                                   const unsigned long& dx));

  protected:
          HashEntry* First;
          void Copy(const HashBucket& hb);
          void Kill();
  };

//-----------------------------------
// HashTable
//          a hash table
//-----------------------------------
class HashTable {
  public:
          HashTable(size_t buckets);
          HashTable(const HashTable& ht);

    ~HashTable();

          HashTable& operator = (const HashTable& ht);

          void Insert(const unsigned long& kx, const unsigned long& dx);
          BooleanEnum Delete(const unsigned long& kx);
          returnType LookUp(const unsigned long& kx) const;
          BooleanEnum TraversetBooleanEnum (*func)(const unsigned long& kx,
                                                   const unsigned long& dx));

  protected:
          size_t NoOfBuckets;
          HashBucket *Table;

          HashTable(); // only for use by derived classes!
          void Copy(const HashTable& ht);
          void Kill();
          };

#endif
```

```
// HASH TABLE LIBRARY -- hash.cpp (v2.00 4/23/94)
//    Defines the abstract base class for implementing a
//    hash table.
// Copyright 1992,1994 Scott Robert Ladd. All rights reserved
// Modified to run in a UNIX system by Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Last Modified: 08/31/95


#include "hash.h"

//---------------------------------
// HashEntry
//          Entries of HashBucket
//---------------------------------
// default constructor
HashEntry::HashEntry() {

  Prev = NULL;
  Next = NULL;
  Key  = 0L;
  Data = 0L;
  }

// conversion constructor
HashEntry::HashEntry(const unsigned long& kx, const unsigned long& dx)
                    : Key(kx), Data(dx) {

  Prev = NULL;
  Next = NULL;
  }

// copy constructor
HashEntry::HashEntry(const HashEntry& he)
                    : Key(he.Key), Data(he.Data) {

  Prev = NULL;
  Next = NULL;
  }

// assignment operator
HashEntry& HashEntry::operator = (const HashEntry& he) {

  Prev = NULL;
  Next = NULL;
  Key  = he.Key;
  Data = he.Data;
  return *this;
  }

// equal-to comparison operator
inline int HashEntry::operator == (const HashEntry& he) {

  return ((Key == he.Key) & (Data == he.Data));
  }

// different-from comparison operator
inline int HashEntry::operator != (const HashEntry& he) {
```

```
    return ((Key != he.Key) & (Data != he.Data));
    }

//-----------------------------------------
// HashBucket
//          a bucket for a HashTable
//-----------------------------------------
// copies a HashBucket into this one
void HashBucket::Copy(const HashBucket& hb) {

    if (hb.First == NULL)
            First = NULL;
    else {
            First = new HashEntry(*(hb.First));

            HashEntry *work = First;
            HashEntry *src  = hb.First;
            HashEntry *last = NULL;
            for (;;) {
                if (work == NULL) {
                        //throw HashEx(HE_ALLOC);
                        cerr << "Error: memory allocation failure " << endl;
                        exit(1);
                        }

                if (last != NULL) {
                        work->Prev = last;
                        last->Next = work;
                        }

                if (src->Next == NULL)
                        break;

                last = work;
                src  = src->Next;
                work = new HashEntry(*src);
                }
            }
    }

// deletes all th entries of a this HashBucket
void HashBucket::Kill() {

    if (First != NULL) {
            HashEntry *work = First, *next;

            while (work != NULL) {
                next = work->Next;
                delete work;
                work = next;
                }

            First = NULL;
            }
    }

// deafult constructor
HashBucket::HashBucket() {

    First = NULL;
```

```
}

// copy constructor
inline HashBucket::HashBucket(const HashBucket& hb) {

  Copy(hb);
  }

// destructor
inline HashBucket::~HashBucket() {

  Kill();
  }

// assignment operator
inline HashBucket& HashBucket::operator = (const HashBucket& hb) {

  Kill();
  Copy(hb);
  return *this;
  }

// adds a new entry to a HashBucket
void HashBucket::AddEntry(const unsigned long& kx, const unsigned long& dx) {

  HashEntry *entry = new HashEntry(kx,dx);

  if (First == NULL)
        First = entry;
  else {
        HashEntry *work = First;

        while (work != NULL) {
            if (work->Key == kx) {
                  delete entry;
                  //throw HashEx(HE_DUPEKEY);
                  cerr << "Duplicated hash key ignored." << endl;
                  return;
                  }

            if (work->Next == NULL) {

                  work->Next = entry;
                  entry->Prev = work;

                  break;
                  }

            work = work->Next;
            }
        }
  }

// deletes an entry from a HashBucket
BooleanEnum HashBucket::DelEntry(const unsigned long& kx) {

  HashEntry *work = First;

  while (work != NULL) {
        if (work->Key == kx) {
```

```
            if (work->Prev == NULL)
                First = work->Next;
            else
                work->Prev->Next = work->Next;

            if (work->Next != NULL)
                work->Next->Prev = work->Prev;

            delete work;

            return BOOL_TRUE;
            }

        work = work->Next;
        }

    return BOOL_FALSE;
    }

// tries to find an entry in this HashBucket
returnType HashBucket::FindEntry(const unsigned long& kx) const {
    HashEntry *work = First;
    returnType rt;

    while (work != NULL) {
        if (work->Key == kx) {
            rt.found = BOOL_TRUE;
            rt.data = work->Data;
            return rt;
            }

        work = work->Next;
        }

    rt.found = BOOL_FALSE;
    return rt;
    }

// traverses the entries of a HashBuckets and applies func() to them
BooleanEnum HashBucket::Traverse(BooleanEnum (*func)(const unsigned long& kx,
                                                     const unsigned long& dx)) {

    HashEntry *work = First;

    while (work != NULL) {
        if (!func(work->Key,work->Data))
            return BOOL_FALSE;

        work = work->Next;
        }

    return BOOL_TRUE;
    }

//-----------------------------------
// HashTable
//        a hash table
//-----------------------------------
// copies the HashBuckets of a HashTable into this one
void HashTable::Copy(const HashTable& ht) {
```

```
    NoOfBuckets = ht.NoOfBuckets;
    Table = new HashBucket[NoOfBuckets];

    if (Table == NULL) {
            //throw HashEx(HE_ALLOC);
            cerr << "Error: memory allocation failure." << endl;
            exit(1);
            }

    for (size_t i = 0; i < NoOfBuckets; ++i)
            Table[i] = ht.Table[i];
    }

// deletes the HashBuckets of this HashTable
inline void HashTable::Kill() {

    delete [] Table;
    Table = NULL;
    }

// default constructor
HashTable::HashTable() {// only used by derived classes'

    NoOfBuckets = 0;
    Table = NULL;
    }

// conversion constructor
HashTable::HashTable(size_t buckets) {

    // verify number of buckets
    if (buckets < 9) {
            //throw HashEx(HE_TOOSMALL);
            cerr << "Number of HashBuckets is too small" << endl;
            exit(1);
            }

    // store number of buckets
    NoOfBuckets  = buckets;

    // allocate Table
    Table = new HashBucket [NoOfBuckets];

    if (Table == NULL) {
            //throw HashEx(HE_ALLOC);
            cerr << "Error: memory allocation failure " << endl;
            exit(1);
            }
    }


// copy constructor
inline HashTable::HashTable(const HashTable & ht) {

    Copy(ht);
    }


// destructor
```

```
HashTable::~HashTable() {

   Kill();
   }


// assignment operator
inline HashTable & HashTable::operator = (const HashTable & ht) {

   Kill();
   Copy(ht);
   return *this;
   }


// inserts entries in HashTable
void HashTable::Insert(const unsigned long & kx, const unsigned long & dx) {

   Table[Hash(kx,NoOfBuckets)].AddEntry(kx,dx);
   }


// deletes entries from HashTable
BooleanEnum HashTable::Delete(const unsigned long & kx) {

   return Table[Hash(kx,NoOfBuckets)].DelEntry(kx);
   }


// tries to find entries in HashTable
returnType HashTable::LookUp(const unsigned long& kx) const {

   return Table[Hash(kx,NoOfBuckets)].FindEntry(kx);
   }


// traverses all the entries in HashTables and applies func() to them
BooleanEnum HashTable::Traverse(BooleanEnum (*func)(const unsigned long & kx,
                                                    const unsigned long & dx)) {

   for (size_t n = 0; n < NoOfBuckets; ++n) {
            if (BOOL_FALSE == Table[n].Traverse(func))
               return BOOL_FALSE;
            }

   return BOOL_TRUE;
   }


size_t Hash(const char * x, size_t buckets) {
   // assume that character array is null terminated
   unsigned long n = 0;
   const char *ch = x;

   while (*ch) {
            n <<= 1;
            n += size_t(*ch);
            ++ch;
            }
```

```
    return size_t(n % (unsigned long)buckets);
    }

/*void HashEx::Explain(DiagOutput & out)
    {
    switch (Error)
            {
            case HE_ALLOC:
              out.DisplayMsg("memory allocation failure",DIAG_ERROR);
              break;
            case HE_ZEROSIZE:
              out.DisplayMsg("cannot create zero-size hash table",DIAG_ERROR);
              break;
            case HE_CORRUPTED:
              out.DisplayMsg("hash table corrupted",DIAG_ERROR);
              break;
            case HE_BADTYPES:
              out.DisplayMsg("mismatched types",DIAG_ERROR);
              break;
            case HE_TOOSMALL:
              out.DisplayMsg("# of hash buckets is very small",DIAG_WARNING);
              break;
            case HE_DUPEKEY:
              out.DisplayMsg("duplicate hash key ignored",DIAG_WARNING);
              break;
            case HE_NOTFOUND:
              out.DisplayMsg("hash record not found",DIAG_WARNING);
              break;
            default:
              out.DisplayMsg("unexpected hash file exception",DIAG_FATAL);
            }
    ; */
```

```
// FILE LIBRARY – hashfile.h (v2.00 5/1/94)
//    Defines a file type for disk-based data files that are
//    indexed via a hash table.
// Copyright 1992,1994 Scott Robert Ladd. All rights reserved
// Modified to run in a UNIX system by Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Last Modified: 08/31/95


#ifndef HASHFILE_H
#define HASHFILE_H

#include "str.h"
#include "datafile.h"
#include "hash.h"
#include "datacl.h"

enum HashFileError
        {
        HFE_ENT_ALLOC,
        HFE_BKT_ALLOC,
        HFE_INV_HFILE
        };


//--------------------------------------------------------------
// HashFileBucket
//    A bucket for a HashFile
//--------------------------------------------------------------

class HashFileBucket : public HashBucket {
  public
        HashFileBucket();
        HashFileBucket(const HashFileBucket & hb);
        HashFileBucket(const HashBucket & hb);

        void operator = (const HashFileBucket & hb);

        void Write(DataFileOutput& file);
        void Read(DataFileInput& file);
  };

//--------------------------------------------------------------
// HashFileTable
//    The type of HashTable used by HashFile
//--------------------------------------------------------------

class HashFileTable : public HashTable {
  public:
        // constructor
        HashFileTable(size_t buckets, const stringClass & file);
        HashFileTable(const stringClass & file);
        HashFileTable(const HashFileTable & hft);
        HashFileTable(const HashTable & hft);

    ~HashFileTable();

        void operator = (const HashFileTable & hft);
```

```
        void Commit();

    protected:
        stringClass FileName;
    };

//-----------------------------------------------------------
// HashFile
//    A file storing data by key, using a hash table
//-----------------------------------------------------------

class HashFile : private HashFileTable, private DataFile {
    public:
        // existing hash file
        HashFile(const stringClass& basename);

        // new hash file
        HashFile(size_t buckets, const stringClass& basename);

        // create hash index from existing DataFile
        HashFile(size_t buckets,
        const stringClass& basename,
        const stringClass& srcname,
        const stringClass& className);

        // copy constructor
        HashFile(const HashFile & hfile);

        // assignment operator
        HashFile & operator = (const HashFile & hfile);

        // write a record
        void Write(const unsigned long & key, const DataBlock& db);

    // positions file pointer in the first record
    void Rewind(void) {DataFile::Rewind();}

        // read a record
        DataBlock Read(const unsigned long & key);

    // sequentially reads a record
    DataBlock Read(void) { return DataFile::Read();}

        // delete a record
        BooleanEnum Delete(const unsigned long & key);

        // write all data to disk
        void Commit();
    };

#endif
```

```
// FILE LIBRARY -- hashfile.cpp (v2.00 5/1/94)
//     Defines a file type for disk-based data files that are
//     indexed via a hash table
// Copyright 1992,1994 Scott Robert Ladd. All rights reserved
// Modified to run in a UNIX system by Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Last Modified  08/31/95


#include "hashfile.h"

const Signature HFTSig      = 0xFF746668, // 'hft'
const Signature UNLONG_SIGN = 0xF0000001; // unsigned long signature

const stringClass HashFileExt(".HFH");
const stringClass DataFileExt(".HFD");

//---------------------------------------------------------------
// HashFileBucket
//     A bucket for a HashFile
//---------------------------------------------------------------


// default constructor
inline HashFileBucket::HashFileBucket() : HashBucket() { ; }

// copy constructor
inline HashFileBucket::HashFileBucket(const HashFileBucket & hb)
                                      : HashBucket(hb) { ; }

// conversion constructor
inline HashFileBucket::HashFileBucket(const HashBucket& hb)
                                      : HashBucket(hb) { ; }

// assignment operator
inline void HashFileBucket::operator = (const HashFileBucket & hb) {

   HashBucket::operator = (hb);
   }

// Writes the entries of a HashFileBucket to disk
void HashFileBucket::Write(DataFileOutput& file) {
   DataBlock db;
   HashEntry *e = First;
   char buf[32];
   ostrstream stream(buf, 32);

   while (e != NULL) {
            stream << e->Key;
            db = DataBlock(UNLONG_SIGN, sizeof(unsigned long), buf);
            file.Write(db);
            stream.flush();

            stream << e->Data;
            db = DataBlock(UNLONG_SIGN, sizeof(unsigned long), buf);
            file.Write(db);

            e = (HashEntry*)(e->Next);
            }
```

```
    file.Write(NULL_BLOCK);
    }


void HashFileBucket::Read(DataFileInput& file) {
    Kill();

    DataBlock db;
    HashEntry *e;
    HashEntry *prev = NULL;
    char *buf;

    for (;;) {
            db = file.Read();

            if (db.IsNull()) break;

            e = new HashEntry;

            if (e == NULL) {
               //throw HashEx(HE_ALLOC);
               cerr << "Error: memory allocation failure " << endl;
               exit(1);
               }
            buf = (char *) (void *) db;
            istrstream stream(buf, db.GetSize());

            // read key
            stream >> e->Key;
            db = file.Read();
            buf = (char *) (void *) db;

            // read file pointer
            stream >> e->Data;
            e->Prev  = prev;

            if (prev != NULL)
               prev->Next = e;
            else
               First = e;

            prev = e;
            }
    }

//---------------------------------------------------------------
// HashFileTable
//     The type of HashTable used by HashFile
//---------------------------------------------------------------

// conversion constructor
inline HashFileTable::HashFileTable(size_t buckets, const stringClass & file)
                                    HashTable(buckets), FileName(file) { }

// conversion constructor
HashFileTable::HashFileTable(const stringClass & file)  FileName(file) {
    // open data file
    DataFileInput input(FileName);
```

```
    // read header w/ # of buckets
    DataBlock db = input.Read();

    if ((HFTSig != db.GetSig()) || (sizeof(size_t) != db.GetSize())) {
            //throw HashFileEx(HFE_INV_HFILE);
            cerr << "Error: invalid hash table " << endl;
            exit(1);
            }

    NoOfBuckets = *((size_t *)((void *)db));

    // allocate Table
    Table = new HashFileBucket [NoOfBuckets];

    if (Table == NULL) {
            //throw HashEx(HE_ALLOC);
            cerr << "Error allocating hash file table." << endl;
            exit(1);
            }

    HashFileBucket * tptr = (HashFileBucket *)Table;

    // read buckets
    for (size_t n = 0; n < NoOfBuckets; ++n) {
             tptr->Read(input);
             ++tptr;
             }
    }


// copy constructor
inline HashFileTable::HashFileTable(const HashFileTable & hft)
                                    :HashTable(hft) { }

// conversion constructor
inline HashFileTable::HashFileTable(const HashTable& hft)
                                    :HashTable(hft) { }

// writes the HashFileTable to disk
HashFileTable::~HashFileTable() {

    Commit();
    }

// assignment operator
inline void HashFileTable::operator = (const HashFileTable & hft) {

    HashTable::operator = (hft);
    }

// executes the actual writing of data to disk
void HashFileTable::Commit() {

    if (Table != NULL) {
            // open data file
            DataFileOutput output(FileName);

            // write header w/ # of buckets
            DataBlock hdr(HFTSig,sizeof(size_t),&NoOfBuckets);
```

```
        output.Write(hdr);

        // write buckets
        HashFileBucket * bucket = (HashFileBucket *)Table;

        for (size_t n = 0; n < NoOfBuckets; ++n) {
            bucket->Write(output);
            ++bucket;
            }
        }
}

//---------------------------------------------------------------
// HashFile
//    A file storing data by key, using a hash table
//---------------------------------------------------------------
stringClass MakeFileName(const stringClass & basename, const stringClass & ext);

// conversion constructor
HashFile::HashFile(const stringClass& basename)
        : DataFile(MakeFileName(basename,DataFileExt),FM_EXISTING),
          HashFileTable(MakeFileName(basename,HashFileExt)) {}

// conversion constructor
HashFile::HashFile(size_t buckets, const stringClass & basename)
        : DataFile(MakeFileName(basename,DataFileExt),FM_NEW),
          HashFileTable(buckets,MakeFileName(basename,HashFileExt)) {}

// conversion constructor
HashFile::HashFile(size_t buckets,
                   const stringClass& basename,
                   const stringClass& srename,
                   const stringClass& className)
        DataFile(MakeFileName(basename,DataFileExt),FM_NEW),
        HashFileTable(buckets,MakeFileName(basename,HashFileExt)) {

  DataFile input(srename);

  DataBlock db;
  BooleanEnum reading = BOOL_TRUE;
  dataClass *obj;

  while (reading) {
        db = input.Read();

        if (db.IsNull())
          break;

        if (className == (stringClass) "CITY")
          obj = new city(db);
        else
          if (className == (stringClass) "WEATHER")
                obj = new weather(db);
          else
                if (className == (stringClass) "PATH")
                  obj = new path(db);
                else
                  if (className == (stringClass) "LOCATION")
                        obj = new location(db);
                  else
```

```
                                        if (className == (stringClass) "CONVEYANCE")
                                          obj = new conveyance(db);
                                        else
                                          if (className == (stringClass) "CARGO")
                                                obj = new cargo(db);
                                          else {
                                                cerr << "Wrong className" << endl;
                                                exit(1);
                                                }

                        Write(obj->getKey(), db);
                        }
        Commit();
        }

// write a record
void HashFile::Write(const unsigned long & key, const DataBlock& db) {

        // store data
        DataFilePtr ptr = DataFile::Write(db);

        // add key and pointer to hash table
        HashFileTable::Insert(key,ptr);
        }

// read a record
DataBlock HashFile::Read(const unsigned long & key) {

        // get pointer to record via key in hash table
        returnType rt = HashFileTable::LookUp(key);
        if (rt.found) {
                // set file pointer
                DataFile::Seek(rt.data);

                // read record at pointer
                return DataFile::Read();
                }
        else
          return DataBlock();
        }

// delete a record
BooleanEnum HashFile::Delete(const unsigned long& key) {

        // get pointer to record via key lookup in hash table
        returnType rt;

        rt = HashFileTable::LookUp(key);
        if (!rt.found)
          return BOOL_FALSE;

        // set pointer to record
        DataFile::Seek(rt.data);

        // delete record
        DataFile::Delete();

        // delete key from hash table
        HashFileTable::Delete(key);
```

```
    // done
    return BOOL_TRUE;
    }


void HashFile::Commit() {

    HashFileTable::Commit();
    }

// builds a file name
stringClass MakeFileName(const stringClass & basename, const stringClass & ext) {
    stringClass name;

    if (basename.Length() < 8)
            name = basename;
    else
            name = basename.CutHead(8);

    name += ext;

    return name;
    }
```

```
//              FILE LIBRARY -- persist.h (5-1-94)
//    Defines a class for handling type-independant data.
// Copyright 1992,1994 Scott Robert Ladd  All rights reserved
// Modified to run in a UNIX system by Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Last Modified: 08/31/95


#ifndef PERSIST_H
#define PERSIST_H

#include "simple.h"
#include "stdlib.h"

//-------------------------------------------------------
// Signature
//    The data type used as an identification code for
//    blocks of data. This should be a unique value for
//    each object type.
//-------------------------------------------------------

typedef unsigned long Signature;

extern const Signature USER_SIG_BASE;
extern const Signature SYS_SIG_BASE;
extern const Signature USER_SIG_MAX;

//-----------------------------------------------------------
// DataBlock
//    Containers store these; objects are created from them
//-----------------------------------------------------------

enum DataBlockError {
            DBE_ALLOC,
            DBE_ZEROSZ,
            DBE_BADSIG,
            DBE_NULLBUF
            };

class DataBlock {
  public
            // constructors
            DataBlock();
            DataBlock(Signature sig, size_t sz, const void * data = NULL);
            DataBlock(const DataBlock & db);

            // destructor
            ~DataBlock();

    // assignment
            DataBlock & operator = (const DataBlock & db);

            // interrogation
    Signature GetSig() const;
            size_t GetSize() const;
            operator const void * () const;
            operator void * ();

    // check for NULL block
```

```
        BooleanEnum IsNull() const;

protected:
        Signature BlockSig;
        size_t    BufferSize;
        void *    BufferPtr;
};

extern const DataBlock NULL_BLOCK;

inline DataBlock::DataBlock() {

  BufferSize = 0;
  BufferPtr  = NULL;
  BlockSig   = 0;
  }

inline DataBlock::~DataBlock() {

  if (BufferPtr != NULL)
        delete [] BufferPtr;
  }

inline Signature DataBlock::GetSig() const {

  return BlockSig;
  }

inline size_t DataBlock::GetSize() const {

  return BufferSize;
  }

inline DataBlock::operator const void * () const {

  return BufferPtr;
  }

inline DataBlock::operator void * () {

  return BufferPtr
  }

inline BooleanEnum DataBlock::IsNull() const {

  if (BufferSize == 0)
        return BOOL_TRUE;
  else
        return BOOL_FALSE;
  }

// internal function for reading/writing DataBlocks
inline void operator <<= (DataBlock & dest, const DataBlock & src) {

        dest = src;
        }

#endif
```

```
// FILE LIBRARY -- persist.cpp (5-1-94)
// Defines a class for handling type-independant data.
// Copyright 1992,1994 Scott Robert Ladd  All rights reserved
// Modified to run in a UNIX system by Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Last Modified: 08/31/95


#include "persist.h"
#include "string.h"
#include "iostream.h"

const Signature USER_SIG_BASE = 0x00000000;
const Signature SYS_SIG_BASE  = 0xF0000000;
const Signature USER_SIG_MAX  = SYS_SIG_BASE - 1;

const DataBlock NULL_BLOCK;

//--------------------------------------------------------
// DataBlock
//    A dynamically-allocated piece of memory
//--------------------------------------------------------


DataBlock::DataBlock(Signature sig, size_t sz, const void * data) {

        if (sz == 0) {
          BufferSize = 0;
          BlockSig   = 0;
          BufferPtr  = NULL;
          }
        else {
          BufferSize = sz;
          BlockSig   = sig;
          BufferPtr  = (void *)new char[sz];

          if (BufferPtr == NULL) {
                cerr << "DataBlock memory alloc failed" << endl;
                exit(1);
                }

          if (data != NULL)
                memcpy(BufferPtr,data,sz);
          }
        }

DataBlock::DataBlock(const DataBlock & db) {
  BufferSize = db.BufferSize;
  BlockSig   = db.BlockSig;

  BufferPtr  = (void *)new char[BufferSize];

  if (BufferPtr == NULL) {
        cerr << "DataBlock memory alloc failed" << endl;
        exit(1);
        }

  memcpy(BufferPtr,db.BufferPtr,BufferSize);
  }
```

```
DataBlock & DataBlock::operator = (const DataBlock & db) {
  BlockSig = db.BlockSig;

  if (BufferSize != db.BufferSize) {
          BufferSize = db.BufferSize;
          if (BufferPtr != NULL)
            delete [] BufferPtr;

          BufferPtr = (void *)new char[BufferSize];

          if (BufferPtr == NULL) {
            cout << "DataBlock memory alloc failed" << endl;
            exit(1);
            }

          }

  memcpy(BufferPtr,db.BufferPtr,BufferSize);

  return *this;
  }
```

```
// DYNAMIC STRING CLASS -- str.h
// Provides a general dynamic string class.
// Copyright 1991-1994 Scott Robert Ladd. All Rights Reserved.
// Modified to run in a UNIX system by Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Last Modified: 08/31/95


#ifndef STR_H
#define STR_H

class stringClass;

#include "stddef.h"
#include "iostream.h"
#include "strstream.h"
#include "simple.h"

enum StrCompVal { SC_LESS, SC_EQUAL, SC_GREATER, SC_ERROR };
enum StrCompMode { SM_SENSITIVE, SM_IGNORE };
enum StrError { SE_OKAY, SE_ALLOC, SE_TOO_LONG, SE_INVALID };

class stringClass
  {
  public:
    // constructor
    stringClass();

    stringClass(const stringClass & str);
    stringClass(const char * cstr);
    stringClass(strstream & strm);

    stringClass(size_t count,  char fillCh = '\0');
    stringClass(size_t maxsize, const char * format, ...);

    // destructor
    ~stringClass();

    // version number
    static int Version();

    // value return methods
    size_t Length() const;
    size_t Size() const;

    // create a c-string from String method
    operator const char * () const;

    // assignment method
    stringClass operator = (const stringClass & str);
    stringClass operator = (strstream & strm);

    // concatenation methods
    friend stringClass operator + (const stringClass & str1,
                    const stringClass & str2);

    friend stringClass operator + (const stringClass & str1, char ch);

    void operator += (const stringClass & str);
```

```
    void operator += (char ch);

    // comparison methods
    int operator <  (const stringClass & str) const;
    int operator >  (const stringClass & str) const;
    int operator <= (const stringClass & str) const;
    int operator >= (const stringClass & str) const;
    int operator == (const stringClass & str) const;
    int operator != (const stringClass & str) const;

    StrCompVal Compare(const stringClass & str,
            StrCompMode caseChk = SM_IGNORE) const;

    // substring search methods

    BooleanEnum Find(const stringClass & str,
            size_t & pos,
            StrCompMode caseChk = SM_IGNORE) const;

    // substring deletion method
    void Delete(size_t pos, size_t count);

    // substring insertion methods
    void Insert(size_t pos, char ch);
    void Insert(size_t pos, const stringClass & str);

    // substring retrieval method
    stringClass Cut(size_t start, size_t count) const;
    stringClass CutHead(size_t count) const;
    stringClass CutTail(size_t count) const;

    // character retrieval method
    char operator [] (size_t pos) const;
    char operator () (size_t pos) const

    // case-modification methods
    void ToUpper();
    void ToLower();

    stringClass AsUpper() const;
    stringClass AsLower() const;

    // stream input/output methods
    friend ostream & operator << (ostream & strm,
                    const stringClass & str);

    friend istream & operator >> (istream & strm,
                    stringClass & str);

protected:
    // instance variables
    size_t Siz; // allocated size
    size_t Len; // current length
    char * Txt; // pointer to text

private:
    // class constant
    static size_t AllocIncr;

    // calc alloc size for needed bytes
```

```
    static size_t CaleSiz(size_t needed);
};

// obtain version number
inline int stringClass::Version()
    {
    return 0600; // version 6.00
    }

// value return methods
inline size_t stringClass::Length() const
    {
    return Len;
    }

inline size_t stringClass::Size() const
    {
    return Siz;
    }

// add-assignment operator
inline void stringClass::operator += (const stringClass & str)
    {
    *this = *this + str;
    }

inline void stringClass::operator += (char ch)
    {
    *this = *this + ch;
    }

// create a c-string from String method
inline stringClass::operator const char * () const
    {
    return Txt;
    }

// comparison methods
inline int stringClass::operator < (const stringClass & str) const
    {
    return (Compare(str) == SC_LESS);
    }

inline int stringClass::operator > (const stringClass & str) const
    {
    return (Compare(str) == SC_GREATER);
    }

inline int stringClass::operator <= (const stringClass & str) const
    {
    return (Compare(str) != SC_GREATER);
    }

inline int stringClass::operator >= (const stringClass & str) const
    {
    return (Compare(str) != SC_LESS);
    }

inline int stringClass::operator == (const stringClass & str) const
    {
```

```
    return (Compare(str) == SC_EQUAL);
    }

inline int stringClass::operator != (const stringClass & str) const
    {
    return (Compare(str) != SC_EQUAL);
    }

// character retrieval method
inline char stringClass::operator [] (size_t pos) const
    {
    if (pos >= Len)
        return '\x00';
    else
        return Txt[pos];
    }

inline char stringClass::operator () (size_t pos) const
    {
    if (pos >= Len)
        return '\x00';
    else
        return Txt[pos];
    }

// stream I/O functions
inline ostream & operator << (ostream & strm, const stringClass & str)
    {
    strm << str.Txt;
    return strm;
    }

#endif
```

```
// DYNAMIC STRING CLASS -- str.cpp
// Provides a general dynamic string class.
// Copyright 1991-1994 Scott Robert Ladd. All Rights Reserved.
// Modified to run in a UNIX system by Felipe Guacache
// Oklahoma State University
// Department of Computer Science
// Last Modified: 08/31/95


#include "str.h"
#include "iomanip.h"
#include "string.h"
#include "stdlib.h"
#include "stdio.h"
#include "stdarg.h"
#include "ctype.h"
#include "limits.h"

// Two useful small functions
char* strlwr(char* str) {

  char *ptr=str;
  while(*ptr) *ptr++ = tolower(*ptr);
  return str;
  }

char* strupr(char* str) {

  char *ptr=str;
  while(*ptr) *ptr++ = toupper(*ptr);
  return str;
  }

//---------------------------------------------------------------
// String class
//---------------------------------------------------------------

// class-global constant initialization
size_t stringClass::AllocIncr = 8;

// calculate the allocation size for a string
inline size_t stringClass::CalcSiz(size_t needed)
  {
  size_t x = ((needed + AllocIncr) / AllocIncr) * AllocIncr;
  return x;
  }

// constructor
stringClass::stringClass()
  {
  Len = 0;
  Siz = 0;
  Txt = NULL;
  }

stringClass::stringClass(const stringClass & str)
  {
  Len = str.Len;
  Siz = str.Siz;
```

```
  if (str.Txt == NULL)
    Txt = NULL;
  else
    {
    Txt = new char[Siz];

          if (Txt == NULL) {
            //throw DebugString(*this,SE_ALLOC);
            cerr << "Error allocating memory for string." << endl;
            exit(1);
            }

     memcpy(Txt,str.Txt,Len + 1);
     }
  }

stringClass::stringClass(const char * estr)
  {
  if ((estr == NULL) || (estr[0] == '\x00'))
     {
     Len = 0;
     Siz = 0;
     Txt = NULL;
     }
  else
     {
     Len = strlen(estr);
     Siz = CalcSiz(Len);

     Txt = new char [Siz];

          if (Txt == NULL) {
            //throw DebugString(*this,SE_ALLOC);
            cerr << "Error allocating memory for string." << endl;
            exit(1);
            }

     memcpy(Txt,estr,Len + 1);
     }
  }

stringClass::stringClass(size_t count, char fillCh)
  {
  if (count == 0)
                    {
                    Len = 0;
                    Siz = 0;
                    Txt = NULL;
                    return;
                    }

  Siz = CalcSiz(count);
  Len = count;

  Txt = new char[Siz];

  if (Txt == NULL) {
          //throw DebugString(*this,SE_ALLOC);
          cerr << "Error allocating memory for string." << endl;
          exit(1);
```

```
        }

    memset(Txt,fillCh,count);

    Txt[count] = '\x00';
    }

stringClass::stringClass(strstream & strm)
    {
    // terminate stream with a null
    strm << ends;

    // get number of characters in stream
    Len = strlen(strm.str());

    // calculate size
    Siz = CalcSiz(Len);

    // allocate text buffer
    Txt = new char[Siz];

    if (Txt == NULL) {
            //throw DebugString(*this,SE_ALLOC);
            cerr << "Error allocating memory for string " << endl;
            exit(1);
            }

    // copy characters from stream
    strcpy(Txt,strm.str());
    }

stringClass::stringClass(size_t maxsize, const char * format, ... )
    {
    // allocate temporary buffer
    char * buffer = new char[maxsize];

    if (buffer == NULL) {
            //throw DebugString(*this,SE_ALLOC);
            cerr << "Error allocating memory for string " << endl;
            exit(1);
            }

    // initialize argument list
    va_list args;

    va_start(args,format);

    // format items into buffer based on format
    #if (MSC_VER >= 0x700)
      Len = _vsnprintf(buffer,maxsize,format,args);

      if (Len == size_t(-1))
        Len = maxsize;
    #else
      Len = vsprintf(buffer,format,args);
    #endif

    // end argument list processing
    va_end(args);
```

```
    // calculate required Txt length
    Siz = CalcSiz(Len);

    // allocate Txt
    Txt = new char[Siz];

    if (Txt == NULL) {
            //throw DebugString(*this,SE_ALLOC);
            cerr << "Error allocating memory for string." << endl;
            exit(1);
            }

    // duplicate data from buffer
    strcpy(Txt,buffer);

    // delete buffer
    delete buffer;
    }

// destructor
stringClass::~stringClass()
    {
    if (Txt != NULL)
       delete [] Txt;
    }

// assignment method
stringClass stringClass::operator = (const stringClass & str)
    {
    // copy length and size
    Len = str.Len;
    Siz = str.Siz;

    // delete existing buffer
    if (Txt != NULL)
       delete [] Txt;

    if (Siz == 0)
       Txt = NULL;
    else
       {
       // allocate a new buffer
       Txt = new char[Siz];

            if (Txt == NULL) {
               //throw DebugString(*this,SE_ALLOC);
               cerr << "Error allocating memory for string." << endl;
               exit(1);
               }

       // copy source text to destination
       memcpy(Txt,str.Txt,Len + 1);
       }

    // outa here
    return *this;
    }

stringClass stringClass::operator = (strstream & strm)
    {
```

```
// delete existing text buffer
if (Txt != NULL)
   delete [] Txt;

// terminate stream with a null
strm << ends;

// get number of characters in stream
Len = strlen(strm.str());

// calculate size
Siz = CaleSiz(Len);

// allocate text buffer
Txt = new char[Siz];

if (Txt == NULL) {
          //throw DebugString(*this,SE_ALLOC);
          cerr << "Error allocating memory for string." << endl;
          exit(1);
          }

// copy characters from stream
strcpy(Txt,strm.str());

return *this;
}

// concatenation methods
stringClass operator + (const stringClass & str1,const stringClass & str2)
   {
   stringClass result;

   // calculate total length of string
   unsigned long totalLen = (unsigned long)str1.Len
                                                    + (unsigned long)str2.Len;

   // if adding two 0-length strings, return a 0-length string
   if (totalLen == 0)
      return result;

   // make sure that the combined length isn't too long
   if (totalLen > UINT_MAX) {
          //throw StringEx(SE_TOO_LONG);
          cerr << "Error: string is too long." << endl;
          exit(1);
          }

   // set size and length
   result.Len = 0;
   result.Siz = stringClass::CaleSiz((size_t)totalLen);

   // allocate buffer
   result.Txt = new char[result.Siz];

   if (result.Txt == NULL) {
          //throw StringEx(SE_ALLOC);
          cerr << "Error allocating memory for string " << endl;
          exit(1);
          }
```

```
  // plkace a terminating character
  result.Txt[0] = '\000';

  // copy str1.Txt
  if (str1.Txt != NULL)
     {
     memcpy(result.Txt,str1.Txt,str1.Len);
     result.Len = str1.Len;
     }

  // copy str2.Txt
  if (str2.Txt != NULL)
     {
     memcpy(&result.Txt[result.Len],str2.Txt,str2.Len + 1);
     result.Len += str2.Len;
     }

  return result;
  }

// concatenation methods
stringClass operator + (const stringClass & str, char ch)
  {
  stringClass result;

  if (str.Txt == NULL)
     {
     // if the string is empty, it contains one character
     result.Len = 1;
     result.Siz = stringClass::AllocIner;
     result.Txt = new char [result.Siz];

           if (result.Txt == NULL) {
              //throw StringEx(SE_ALLOC);
              cerr << "Error allocating memory for string." << endl;
              exit(1);
              }

     result.Txt[0] = ch;
     result.Txt[1] = '\000';
     }
   else
     {
     // if string is maximum length, error
           if (str.Len == UINT_MAX) {
              //throw StringEx(SE_TOO_LONG);
              cerr << "Error: string is too long." << endl;
              exit(1);
              }

     // increment length of string
     result.Len = str.Len + 1;

     // if length = size, increase size
     if (result.Len == str.Siz)
        result.Siz = str.Siz + stringClass::AllocIner;
     else
        result.Siz = str.Siz;
```

```
    // allocate buffer
    result.Txt = new char[result.Siz]

          if (result.Txt == NULL) {
            //throw StringEx(SE_ALLOC);
            cerr << "Error allocating memory for string." << endl;
            exit(1);
            }

    // copy orginal string
    memcpy(result.Txt,str.Txt,str.Len);

    // append character
    result.Txt[str.Len]  = ch;
    result.Txt[result.Len] = '\000';
    }

  return result;
  }

StrCompVal stringClass::Compare(const stringClass & str,
                StrCompMode caseChk) const
  {
  // handle special cases where one string is empty
  if (Txt == NULL)
    if (str.Txt == NULL)
      return SC_EQUAL;
    else
      return SC_LESS;

  if (str.Txt == NULL)
    return SC_GREATER;

  // compare the # of characters in the shorter string
  size_t count;

  if (str.Len < Len)
    count = str.Len;
  else
    count = Len;

  // working variables
  char  c1, c2;
  size_t i;

  if (caseChk == SM_IGNORE)
    {
    // case insensitive comparison
    for (i = 0; i < count; ++i)
      {
      c1 = (char)tolower(Txt[i]);
      c2 = (char)tolower(str.Txt[i]);

      // if characters differ
      if (c1 != c2)
        {
        // select appropriate result
        if (c1 < c2)
          return SC_LESS;
        else
```

```
            return SC_GREATER;
          }
        }
      }
    else
      {
      for (i = 0; i < count; ++i)
        {
        c1 = Txt[i];
        c2 = str.Txt[i];

        // if characters differ
        if (c1 != c2)
          {
          // select appropriate result
          if (c1 < c2)
            return SC_LESS;
          else
            return SC_GREATER;
          }
        }
      }

    // at this point, no differences were found
    if (Len == str.Len)
      return SC_EQUAL;
    else
      {
      // is lengths differ, shorter string < longer one
      if (Len < str.Len)
        return SC_LESS;
      else
        return SC_GREATER;
      }
    }

BooleanEnum stringClass::Find(const stringClass & str,
                size_t & pos,
                        StrCompMode caseChk) const {

    // uses the brute force method
    if (Len < str.Len)
      return BOOL_FALSE;

    // duplicate buffers
    char * target = new char[Len + 1];

    if (target == NULL) {
            //throw DebugString(*this,SE_ALLOC);
            cerr << "Error allocating memory for string " << endl;
            exit(1);
            }

    strcpy(target,Txt);

    char * pattern = new char[str.Len + 1];

    if (pattern == NULL) {
            //throw DebugString(*this,SE_ALLOC);
            cerr << "Error allocating memory for string " << endl;
```

```
        exit(1);
        }

strcpy(pattern,str.Txt);

// create return value variable
BooleanEnum result;

// convert to all lowercase if case-insensitive search
if (caseChk == SM_IGNORE) {
   strlwr(target);
   strlwr(pattern);
   }

// calculate last position in *this where str could be
size_t end = Len - str.Len;
size_t p, t;

// start at the beginning of ttarget
pos = 0;

for (;;)
   {
   p = 0;  // beginning of pattern
   t = pos; // beginning of search position in target

   // while characters match
   // and we're not at the end of the strings

   while ((pattern[p] == target[t])
      && (pattern[p] != 0)
      && (target[t]  != 0))
      {
      // move to next character
      ++t;
      ++p;
      }

   // if we've reached the end of pattern
   //    we've found pattern in target

         if (pattern[p] == 0) {
      result = BOOL_TRUE;
      break;
      }

   // if we've reached the end of target
   // or we've searched far enough
   //    pattern has not been found

   if ((target[t] == 0) || (pos >= end))
      {
      result = BOOL_FALSE;
      break;
      }

   // keep looking, starting at the mismatch

   ++pos;
   }
```

```
    // delete resultorary buffers
    delete target;
    delete pattern;

    // outa here
    return result;
    }

// substring deletion method
void stringClass::Delete(size_t pos, size_t count) {

    if (Txt == NULL)
        return;

    size_t newLen, i;

    // error if deleting outside of string
    if ((pos + count - 1) > Len) {
            //throw DebugString(*this,SE_INVALID);
            cerr << "Error: deleting outside string." << endl;
            exit(1);
            }

    // length of new string
    newLen = Len - count;

    if ((Siz - newLen) > AllocIner)
        {
        // allocation size has changed
        // calculate new size

        Siz = CalcSiz(newLen);

        // create new buffer

        char * result = new char[Siz];

            if (result == NULL) {
              //throw DebugString(*this,SE_ALLOC);
              cerr << "Error allocating memory for string." << endl;
              exit(1);
              }

        // copy characters into new buffer
        char * tptr = result;

        for (i = 0; i <= Len; ++i)
            {
            // when count is reached, skip deleted characters
            if (i == pos)
                i += count;

            *tptr = Txt[i];
            ++tptr;
            }

        // delete old buffer
        delete [] Txt;
```

```
    // assign new buffer
    Txt = result;
    }
  else
    {
    // just "slide" characters down
    for (i = pos; i <= pos + count; ++i)
      Txt[i] = Txt[i + count];
    }

  Len = newLen;
  }

// substring insertion methods
void stringClass::Insert(size_t pos, char ch)
  {
  if (pos > Len) {
        //throw DebugString(*this,SE_INVALID);
        cerr << "Error: inserting outside string." << endl;
        exit(1);
        }

  if (Txt == NULL)
    {
    // an empty string == ch
    Len = 1;
    Siz = AllocInc;

    Txt = new char [Siz];

        if (Txt == NULL) {
          //throw DebugString(*this,SE_ALLOC);
          cerr << "Error allocating memory for string " << endl
          exit(1);
          }

    Txt[0] = ch;
    Txt[1] = '\000';
    }
  else
    {
    size_t newLen = Len + 1;
    size_t i;

    if (newLen == Siz)
      {
      // need a larger buffer
      Siz += AllocInc;

      // create resultorary buffer
      char * result = new char[Siz];
      char * tptr = result;

          if (result == NULL) {
                //throw DebugString(*this,SE_ALLOC);
                cerr << "Error allocating memory for string " << endl;
                exit(1);
                }

      // copy in old buffer, inserting ch when needed
```

```
        for (i = 0; i <= Len; ++i)
          {
          if (i == pos)
            {
            *tptr = ch;
            ++tptr;
            }

          *tptr = Txt[i];
          ++tptr;
          }

        // delete old buffer
        delete [] Txt;

        // assign new buffer and length
        Txt = result;
        Len = newLen;
        }
      else
        {
        // slide characters right
        for (i = newLen; i > pos; --i)
          Txt[i] = Txt[i-1];

        // insert character
        Txt[pos] = ch;

        // adjust length
        Len = newLen;
        }
      }
    }

void stringClass::Insert(size_t pos, const stringClass & str)
    {
    if (str.Txt == NULL)
      return;

    if (pos > Len) {
              //throw DebugString(*this,SE_INVALID);
              cerr << "Error: inserting outside string." << endl;
              exit(1);
              }

    if (Txt == NULL)
      {
      // empty string = str
      *this = str;
      }
    else
      {
      // calculate new length
      unsigned long totalLen = str.Len + Len;

            if (totalLen > UINT_MAX) {
              //throw DebugString(*this,SE_TOO_LONG);
              cerr << "Error: string is too long " << endl;
              exit(1);
              }
```

```
    size_t i, j;

    // if new length > current size
    if (totalLen > Siz)
       {
       // allocate new buffer
       Siz = CalcSiz((size_t)totalLen);

       char * result = new char [Siz];
       char * tptr = result;

       // copy buffers from source strings
       for (i = 0; i <= Len; ++i)
          {
          if (i == pos)
             {
             for (j = 0; j < str.Len; ++j)
                {
                *tptr = str.Txt[j];
                ++tptr;
                }
             }

          *tptr = Txt[i];
          ++tptr;
          }

       // delete old buffer
       delete [] Txt;

       // assign new buffer
       Txt = result;
       }
    else
       {
       // slide section old buffer to right
       for (i = Len + str.Len; i > pos + str.Len; --i)
          Txt[i] = Txt[i - str.Len];

       // insert new string
       for (i = 0; i < str.Len; ++i)
          Txt[pos + i] = str.Txt[i];
       }

    Len = (size_t)totalLen;
    }
 }

// substring retrieval method
stringClass stringClass::Cut(size_t start, size_t count) const
   {
   if ((start + count) > Len) {
            //throw DebugString(*this,SE_INVALID);
            cerr << "Error: invalid cut operation." << endl;
            exit(1);
            }

   stringClass result;
```

```
    if ((start < Len) && (count > 0))
        {
        result.Len = count;
        result.Siz = CalcSiz(count);
        result.Txt = new char[result.Siz];

                if (result.Txt == NULL) {
                    //throw DebugString(*this,SE_ALLOC);
                    cerr << "Error allocating memory for string." << endl;
                    exit(1);
                    }

        memcpy(result.Txt,&Txt[start],count);

        result.Txt[count] = '\000';
        }

    return result;
    }

stringClass stringClass::CutHead(size_t count) const
    {
    if (count > Len) {
                //throw DebugString(*this,SE_INVALID);
                cerr << "Error: invalid cut operation." << endl;
                exit(1);
                }

    stringClass result;

    if (count > 0)
        {
        result.Len = count;
        result.Siz = CalcSiz(count);
        result.Txt = new char[result.Siz];

                if (result.Txt == NULL) {
                    //throw DebugString(*this,SE_ALLOC);
                    cerr << "Error allocating memory for string." << endl;
                    exit(1);
                    }

        memcpy(result.Txt,Txt,count);

        result.Txt[count] = '\000';
        }

    return result;
    }

stringClass stringClass::CutTail(size_t count) const
    {
    if (count > Len) {
                //throw DebugString(*this,SE_INVALID);
                cerr << "Error: invalid cut operation." << endl;
                exit(1);
                }

    stringClass result;
```

```
    if (count > 0)
        {
        result.Len = count;
        result.Siz = CalcSiz(count);
        result.Txt = new char[result.Siz];

                if (result.Txt == NULL) {
                    //throw DebugString(*this,SE_ALLOC);
                    cerr << "Error allocating memory for string." << endl;
                    exit(1);
                    }

        memcpy(result.Txt,&Txt[Len - count - 1],count);

        result.Txt[count] = '\000';
        }

    return result;
    }

// case-modification methods
void stringClass::ToUpper()
    {
    if (Txt != NULL)
        strupr(Txt);
    }

void stringClass::ToLower()
    {
    if (Txt != NULL)
        strlwr(Txt);
    }

stringClass stringClass::AsUpper() const
    {
    stringClass result = *this;

    if (result.Txt != NULL)
        strupr(result.Txt);

    return result;
    }

stringClass stringClass::AsLower() const
    {
    stringClass result = *this;

    if (result.Txt != NULL)
        strlwr(result.Txt);

    return result;
    }

// stream I/O functions

istream & operator >> (istream & strm, stringClass & str)
    {
    static const size_t bufsize = 256;
    static char buf[bufsize];
```

```
char endline;

strm.get(buf,bufsize,'\n');
strm.get(endline);

str = buf;

return strm;
}
```

APPENDIX C

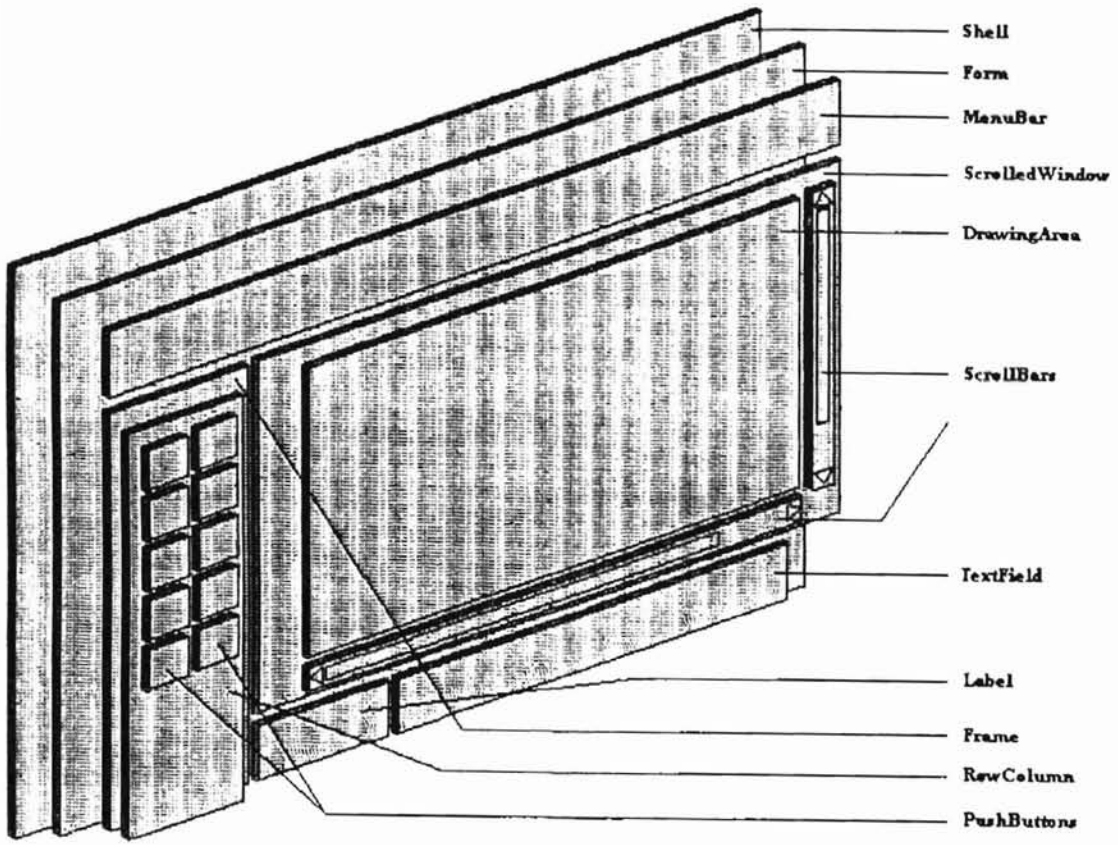
MAKEFILE

```
#
##############################################################################
# Makefile for tsps:
# An Object-Oriented Graphic User Interface for Transportation Scheduling
# Problem Specification
##############################################################################
#

CC = CC

DEBUG = -g

DEBUG1 =        -c $(DEBUG) -DFUNCPROTO -DXTFUNCPROTO

LIBS =          -lm -lXm -lXt -lX11 -lXext -lPW -leh -linet -lnsl -lseq

OBJS =      tsps.o str.o hashfile.o datafile.o hash.o persist.o\
            datael.o dtel.o rpt.o

INCLUDE =       tsps.h


##############################################################################
# Main files
##############################################################################
tsps:                               $(OBJS)
                                    $(CC) $(DEBUG) -o tsps $(OBJS) $(LIBS)

tsps.o                  tsps.C $(INCLUDE)
                                    $(CC) $(DEBUG1) tsps.C

str.o                   str.C str.h
                                    $(CC) $(DEBUG1) str.C

hashfile.o:     hashfile.C hashfile.h
                                    $(CC) $(DEBUG1) hashfile.C

hash.o:                 hash.C hash.h
                                    $(CC) $(DEBUG1) hash.C

datafile.o:     datafile.C datafile.h
                                    $(CC) $(DEBUG1) datafile.C

persist.o:      persist.C persist.h
                                    $(CC) $(DEBUG1) persist.C

datael.o        datael.C datael.h
                                    $(CC) $(DEBUG1) datael.C

dtel.o:                 dtel.C dtel.h
                                    $(CC) $(DEBUG1) dtel.C

rpt.o:                  rpt.C rpt.h
                                    $(CC) $(DEBUG1) rpt.C
```

APPENDIX D

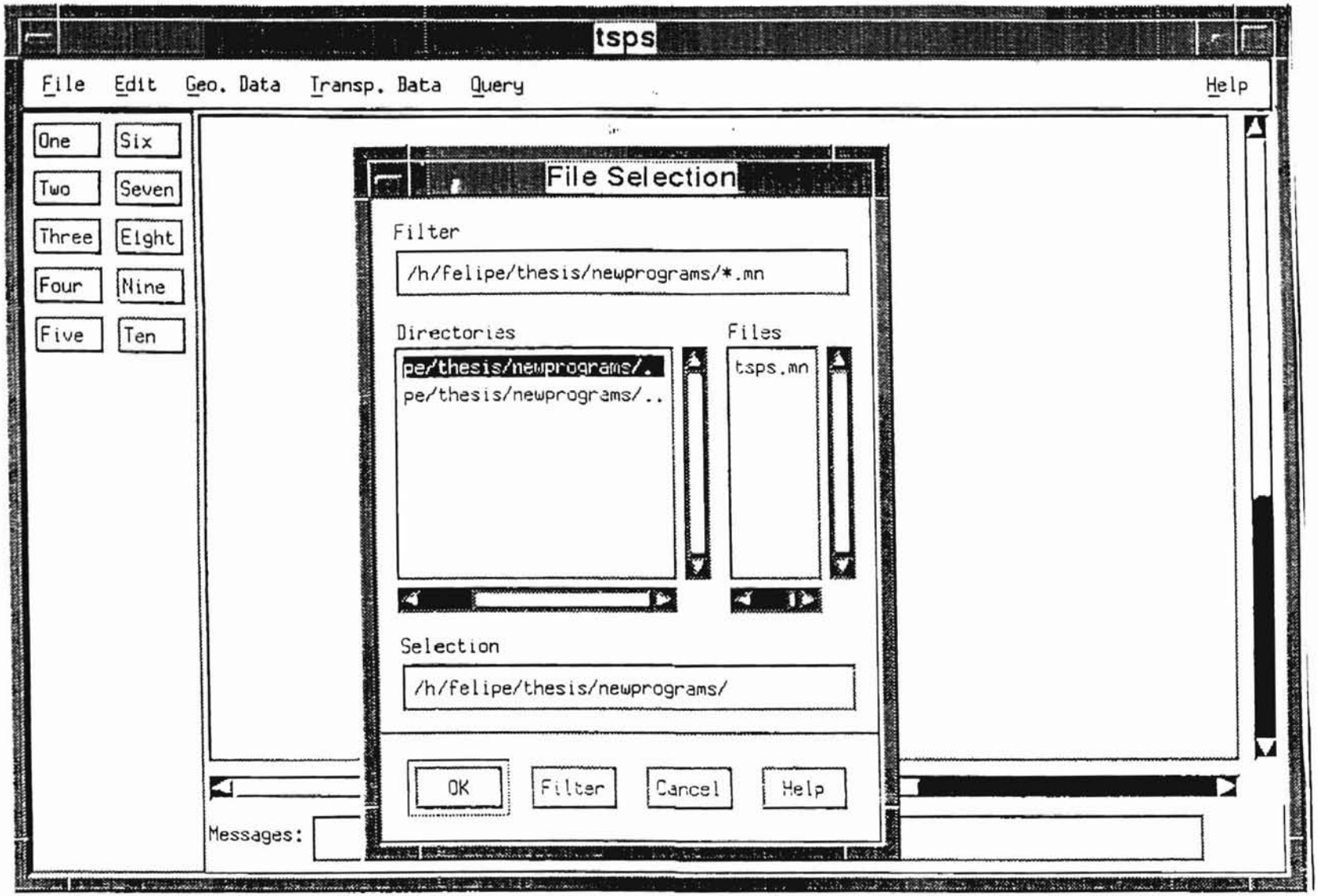
GUI LAYOUT

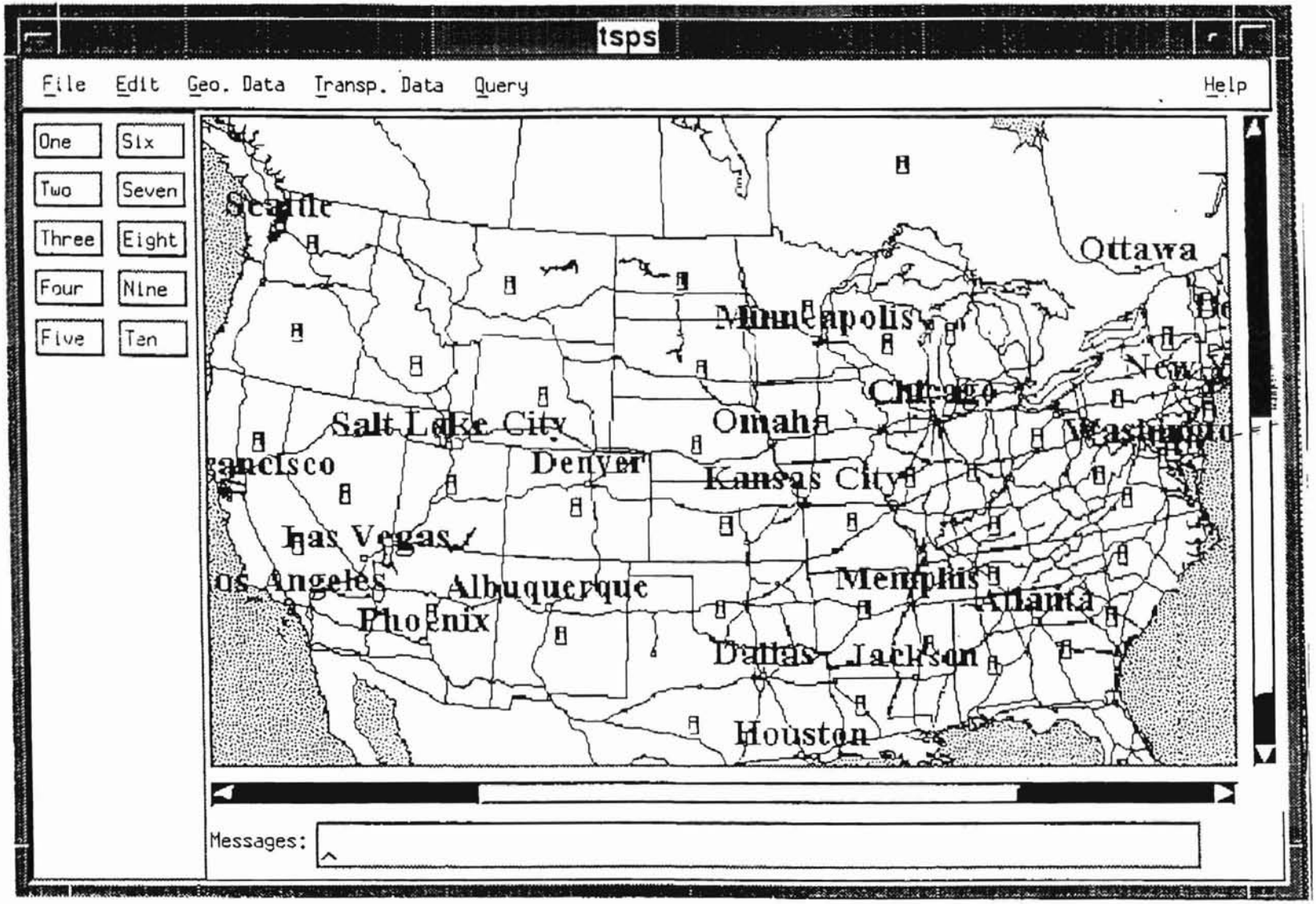
GUI LAYOUT.

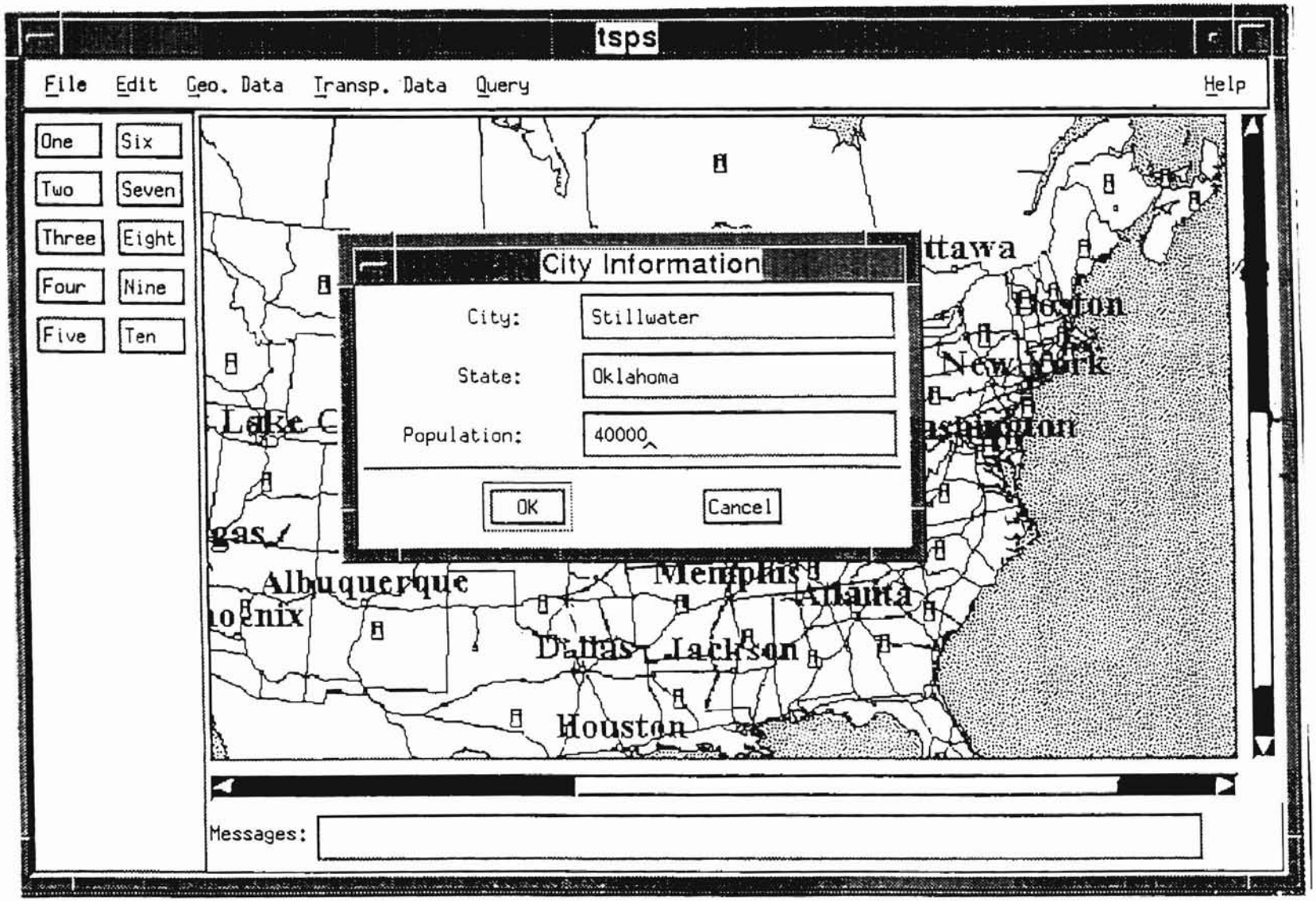

Shell
Form
MenuBar
ScrolledWindow
DrawingArea
ScrollBars
TextField
Label
Frame
RowColumn
PushButtons

APPENDIX E

SCREEN PRINTOUTS


tsps
File
Edit
Geo. Data
Transp. Data
Query
Help
One
Six
Two
Seven
Three
Eight
Four
Nine
Five
Ten
File Selection
Filter
/h/felipe/thesis/newprograms/*.mn
Directories
Files
pe/thesis/newprograms/.
pe/thesis/newprograms/..
tsps.mn
Selection
/h/felipe/thesis/newprograms/
OK
Filter
Cancel
Help
Messages:


tsps
File
Edit
Geo. Data
Transp. Data
Query
Help
One
Six
Two
Seven
Three
Eight
Four
Nine
Five
Ten
Seattle
Ottawa
Minneapolis
Chicago
Salt Lake City
Omaha
Denver
Kansas City
Las Vegas
Albuquerque
Memphis
Atlanta
Phoenix
Dallas
Jackson
Houston
Messages:


tsps
File
Edit
Geo. Data
Transp. Data
Query
Help
One
Two
Three
Four
Five
Six
Seven
Eight
Nine
Ten
City Information
City:
Stillwater
State:
Oklahoma
Population:
40000
OK
Cancel
Albuquerque
Memphis
Atlanta
Dallas
Houston
Boston
Messages:

VITA

Felipe Guacache

Candidate for the Degree of

Master of Science

Thesis: OBJECT-ORIENTED GRAPHIC USER INTERFACE FOR TRANSPORTATION SCHEDULING PROBLEM SPECIFICATION

Major Field: Computer Science

Biographical:

Personal Data: Born in Puerto Cabello, Venezuela, On August 14, 1967, the son of Neri Felipe Guacache and Ramona Leonor de Guacache.

Education: Graduated from Pedro Gual High School, Valencia, Venezuela in July 1984; received an Associate Degree in Information Science and a Degree in Information Science Engineering from Central Technological University, Guacara, Venezuela in September 1989 and September 1991, respectively. Completed the requirements for the Master of Science degree in Computer Science at Oklahoma State University in May 1996.

Experience: Employed by Oklahoma State University, Department of Computer Science as a Graduate Teaching Assistant; from January to May 1995; University of Carabobo,Valencia, Venezuela, part-time professor, from November 1991 to July 1992; Corpoven S.A., Puerto Cabello, Venezuela, Computer Systems Analyst, from July 1991 to October 1992.

Professional Memberships: Student Member of the Association for Computing Machinery.