PC Front End and Network Back End for the Composite Information System Tool Kit

by

Wilberto Martinez

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Electrical Science and Engineering

at the Massachusetts Institute of Technology

May 1989

Author____
Department of Electrical Engineering and Computer Science
May 16,1989

Certified by____
Professor Stuart Madnick
Thesis Supervisor

Accepted by____
Leonard A. Gould
Chairman, Department Committee on Undergraduate Theses

PC Front End and Network Back End for the Composite Information System Tool Kit

by

Wilberto Martinez

Submitted to the

Department of Electrical Engineering and Computer Science

May 16, 1989

In Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Electrical Science and Engineering.

ABSTRACT

A menu driven user interface was implemented to facilitate usage of the Composite Information System/ Tool Kit (CIS/TK) currently being developed at the Sloan School Management. CIS/TK capability inquiries can be handled efficiently and quickly with minimal knowledge of the underlying system. The user is able to build, save, and retrieve queries to request information from CIS/TK. Graphic and plotting capabilities have been enhanced and facilities for transporting data to other graphics packages have been provided.

Thesis Supervisor: Stuart Madnick
Title: Professor of Management Science

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Background

The Composite Information System/Tool Kit (CIS/TK) is a research prototype being developed at the Massachusetts Institute of Technology to address the difficulties in combining information from various sources. CIS/TK was initially developed as an element to help the understanding of the issues surrounding the integration of information systems.

Since CIS/TK was intended to serve as a research tool, the user interface that the system possesses does not cater to a wide range of users. This thesis presents the changes that have been implemented to the CIS/TK interface in order to serve a wider range of computer literacy range of CIS/TK system users.

1.2 Goals of Thesis

One of the primary goals of the CIS/TK system is to ease the retrieval of data from disparate databases. Some of the problems that novice users of the CIS/TK system face are: lack of user friendliness, the need to know system semantics in order to retrieve information, and the lack of graphics and plotting facilities to visualize query results. The goals of this thesis are to produce an interface that: 1) facilitates the retrieval of data from CIS/TK, 2) allows the usage of graphics packages on information retrieved from CIS/TK, and 3) implements a multi-user coordinated system that allows multiple users to share one CIS/TK application.

1.3 Approach and Presentation

Chapter 2 describes the environment in which the user interface is built and how the interface interacts with CIS/TK. It presents the implementation of the multi-user CIS/TK, and discusses the Unix mechanisms used in the communication between CIS/TK and the user interface.

Chapter 3 provides a detailed technical analysis of the implementation of the user interface. It presents the problems faced in the implementation, and describes the solutions developed to address those problems. Query generation, information retrieval and graphics packages are the major topics discussed in the chapter. Sample sessions have been provided to present the operation of the enhancements made to the user interface

The conclusions in Chapter 4 suggest enhancements for future versions of the user interface. It discusses some of the major problems that were faced, and suggests possible new ways to confront them.

2. Architecture

The purpose of this section is to describe the relationship between the user interface, CIS/TK and the user environment. The section titled Description of Environment presents the framework of the architecture. The section Multi-user CIS/TK presents the implementation of the multi-user CIS/TK.

2.1 Description of Environment

One of the goals in the implementation of the user interface was its immunity from the recurring changes to CIS/TK. The user interface has been implemented in the AT&T 3B2 Unix environment. It has been developed as a module separate from CIS/TK, and separate from the user environment. Figure 1 presents the architecture of the user interface module, and its inter-connections to the CIS/TK system and to the user environment. The left section of Figure 1 represents the user environment. The current interface supports the AT&T 3B2 and the IBM PC user environments. The middle section represents the user interface that has been developed. The rightmost section describes the CIS/TK environment. CIS/TK has been developed on the IBUKI LISP environment, which in turn is supported by the AT&T 3B2 Unix environment.

A close analysis of the figure shows that the interface has been developed as a module that does not reside on the user environment or on the CIS/TK environment. The only dependance of the interface on CIS/TK occurs in the formatting of the query inquiries to the system, and in the way in which CIS/TK returns its output.

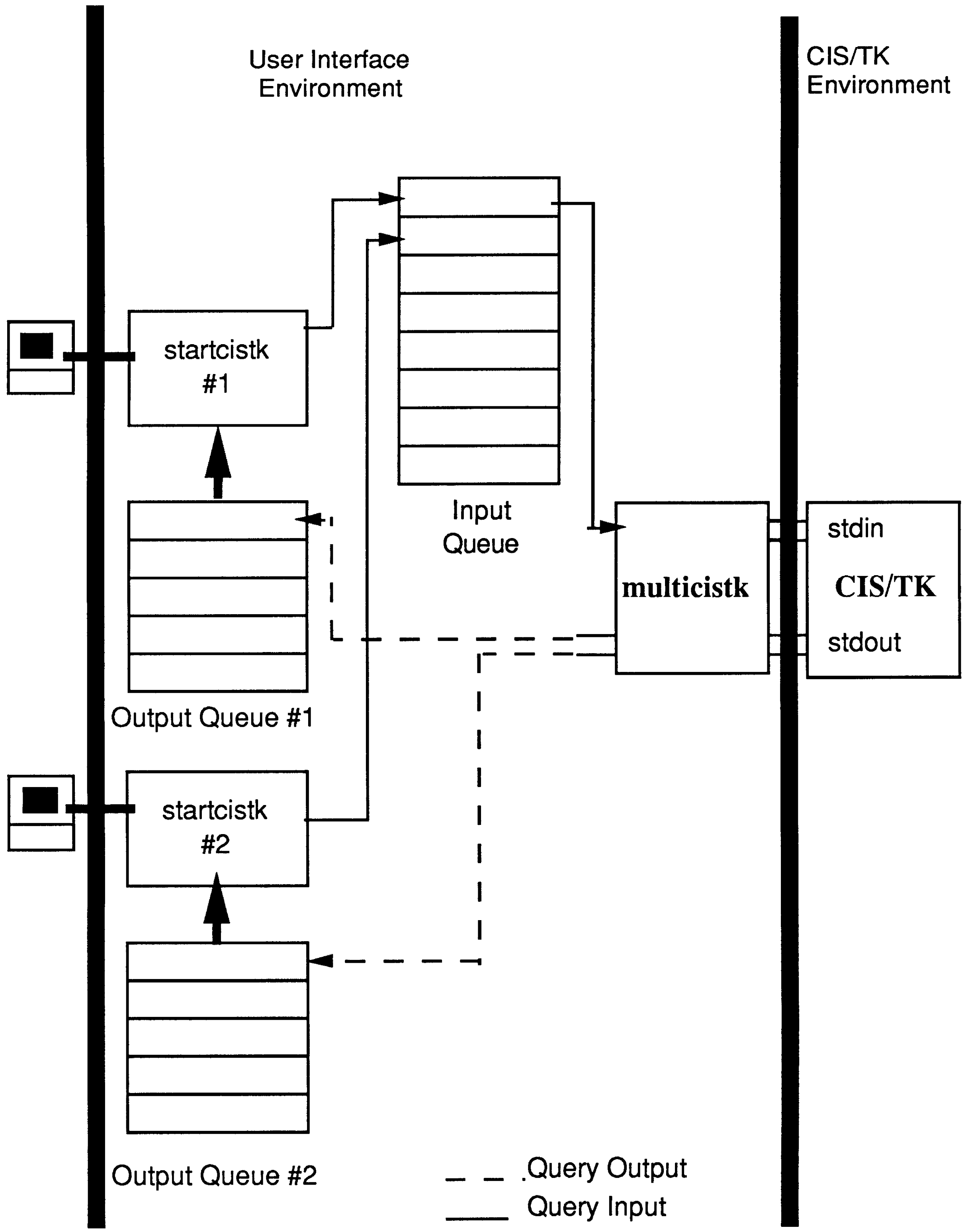

Figure 1. System Architecture

The majority of the user interface has been developed using the C language. Some functions have been written in LISP in order to request and retrieve information from CIS/TK. The communication between the different modules has been achieved by some of the facilities provided by the AT&T 3B2 Unix environment. The next section describes how the user interface communicates with the User environment and with CIS/TK.

2.2 Multi-User CIS/TK

The purpose of the multi-user CIS/TK environment is to allow many users to share the same CIS/TK application. The time it takes to load CIS/TK, and the limitation to the number of CIS/TK applications that can be running concurrently, make it desirable to implement a multi-user CIS/TK. In the multi-cistk environment, requests from different terminals are processed in a first-in, first-out manner by a shared CIS/TK.

The Multi-User CIS/TK system implemented can be divided into three components: 1) the user interface, named startcistk, 2) the CIS/TK application that is shared, and 3) multi-cistk, the multi-user interface to CIS/TK. The communication between the startcistk modules and the multi-cistk interface is achieved via a message queue. Multi-cistk and CIS/TK communicate through pipes. The multi-user CIS/TK system works the following way:

1) a startcistk module reads a request typed at the terminal by the user. The request is sent to multi-cistk via the Input message queue.
2) Multi-cistk gets the message from the queue and sends it to CIS/TK to be processed. Multi-cistk then waits for CIS/TK to send its response.
3) CIS/TK gets the request, processes it and sends the response multi-cistk.
4) Multi-cistk sends the response to the appropriate startcistk module output message queue.

5) Startcistk reads the output from the Output message queue and displays it at the terminal.

Figure 1 in section 2.1 illustrates the flow of the requests in the system. Note that the output from CIS/TK to the startcistk modules is directed to separate output queues. All the startcistk interfaces share an input message queue, but each interface owns a separate output queue. This feature is necessary to avoid having one startcistk module affect the performance of others.

A sample scenario of such a case would be as follows: user 1 logs into CIS/TK from his 1200 baud modem at home, while user 2 logs into the system through a computer that is connected directly to CIS/TK, at approximately 4800 baud. User 2 can retrieve information from CIS/TK four times as fast as user 1. However, the output queue sends the information in a FIFO manner. If user 2 were to generate a query immediately after user 1 had generated a query, the following events would take place:

-- CIS/TK would receive and process the request from User 1. CIS/TK would send its response to Multi-cistk, which in turn would place the response in the output queue.

-- CIS/TK would perform the task described above for the query from user 2. The response would be placed in the Output queue in a FIFO manner. The data from user 2 would be placed after the data for user 1.

-- User 2 would have to wait until User 1 retrieved all his data before he could have access to the information he requested. This scheme is not favorable to User 2 because he can retrieve information faster than User 1.

The implementation of separate output queues allows each user to retrieve his information as soon as it is available from CIS/TK. The rate of data retrieval is limited by the speed at which the user's terminal can retrieve the data, instead of the rate at which other users can retrieve their data from the output queue.

The next section discusses the implementation of the messaging mechanism, and communication between CIS/TK and multi-cistk.

2.2.1 Messages and Pipes

Messages and pipes allow the inter-process communication in Unix. A process in Unix can be identified as a program that is being run by a user. Processes can be classified as foreground processes - the user is interacting with the program- or as background processes - the user is not interacting with the program. The three components of the Multi-user cistk system can be classified as processes.

The message queues that communicate between the user interface and the multi-cistk interface can be identified by the *qnumber* and the *mqueue* number. Qnumber is a variable in a User's Unix path that can be set to different integer values. Mqueue is the identification number that Unix assigns to an output queue for the purpose of sending and receiving messages. The qnumber is assigned by the user while mqueue is a number that the Unix system generates for a specific qnumber.

The messages that communicate the different processes are composed of three fields: the message destination, the message text, and the message originator id. All the messages that are originated at the startcistk modules and are intended for the Multi-cistk application have a destination of 1. If the messages are originated by CIS/TK as a response to a request, the message destination will contain the process id of the startcistk module that originated the message. Figure 2 illustrates the message mechanism used to interchange information. The messaging mechanism discussed has been implemented to interchange the information between the user interface and the multi-cistk interface. However, multi-cistk and CIS/TK interchange information via pipes.

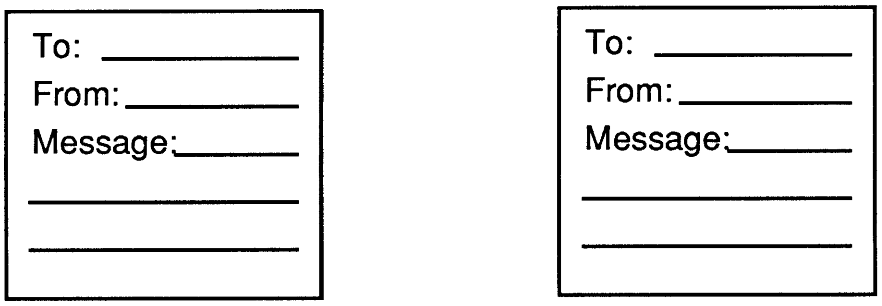

Figure 2. Message Mechanism

A pipe allows the output of one program to be the input of another. Figure 1 shows the pipes between CIS/TK and multi-cistk. The input pipe from multi-cistk to CIS/TK forces CIS/TK to read its input from the multi-cistk interface, instead of from the terminal. The output pipe from CIS/TK to multi-cistk places the information that results from the inquiries to CIS/TK in the multi-cistk instead of the terminal. Multi-cistk then routes the message to the appropriate Output queue.

The piping and message mechanisms allow the Multi-user CIS/TK interface to be established. The next section describes how to invoke the user interface, and how to initialize the multi-user CIS/TK system.

2.2.2 Invoking Startcistk and the Multi-User Interface

There are several ways in which the multi-user interface, and the user interface can be initialized. Table 1 presents the different ways in which the multi-user interface can be invoked, and presents the differences amongst them.

Instruction	Result
multicistk	The multi-user interface is started. The pipes between multicistk and cis/tk are established. Multicistk will display the messages passed between the user interfaces and cis/tk on the terminal. This prevents the use of the terminal for other tasks until multicistk is stopped.
multicistk > filename &	This call invokes multi-user in the background. The messages passed between the user interface and cistk are placed in the file 'filename'. Since multicistk is running in the background, the user can use the terminal for other tasks.
startcistk	The user interface is invoked. It automatically checks if a multicistk process is running. If multicistk does not exist, startcistk invokes multicistk in the background, and prompts the user for input.

Table 1. Initialization of Multi-cistk.

Once multicistk is running, many users can share the application. The problem faced at this time is that multicistk will run until the user that initialized it logs off the system. Currently, it takes approximately two minutes to load CIS/TK. Thus, it would be advantageous to have multi-cistk running in the background at all times. This would avoid the time that the users have to wait until CIS/TK is loaded.

3. Implementation

This section describes the issues that were faced in the implementation, and presents the solutions and trade-offs that were made in order to solve those problems.

3.1 User Interface

The user interface implemented is intended for users whose whose knowledge of CIS/TK is not extensive. Its purpose is to avoid having the user worry about system details. Interaction is provided by means of a menu driven system.

The initial screen is shown in figure 3.

```
***********************************************************************
****************************  CIS/TK MENU  ****************************
***********************************************************************

PLEASE SELECT THE FUNCTION THAT YOU WANT TO PERFORM
        1) Display entities that are available
        2) Display attributes of an entity
        3) Construct query
        4) Retrieve query
        5) Exit

***********************************************************************
```

Figure 3. Initial Screen for User Interface

CIS/TK currently can retrieve information from four entities: AM/FINANCIAL_INFO, AM/COMPANY_DESC, AM/SIC and AM/INDUSTRY. Each one of the entities possesses a set of attributes. The initial screen provides the user with four options. Option 1 allows the user to view the entities that are available in the system. Option 2 allows the user to view the attributes of the entities in the system. The first two options essentially provide the user with the information categories in CIS/TK.

The other two options allow the user to retrieve information that satisfies a certain criterion. Option 3 enables the user to build his questions (construct queries). The user is able to select and place constraints on a set of entities and attributes. Option 4 allows the user to retrieve any queries that have been generated previously. After the query is fetched, it can either be displayed at the terminal, or sent to CIS/TK to generate a response. Section 3.1.1 presents the steps followed in fetching the entities and attributes from CIS/TK.

3.1.1 Fetching CIS/TK Entities and Attributes

The user interface needs to provide the user with the system's entities and attributes. CIS/TK currently has some facilities that can be manipulated to retrieve the entities and the attributes available in the system. Table 2 below shows the commands used in the interface to retrieve the entities and the attributes from CIS/TK.

```
(get-object  'entity  'subordinates)
-->> returns the entities that are available in the system

(length (get-object 'entity 'subordinates))
-->> returns the number of entities in CIS/TK

(get-object (nth number (get-object 'entity 'subordinates)) 'attributes)
-->> returns attributes of the entity indicated by number. That is, if
number is set to one, the attributes of the first entity would be returned.
```

Table 2. CIS/TK Commands Used to Retrieve Entities and Attributes

The user interface implemented interchanges information with CIS/TK through files. All the requests to CIS/TK are placed in files. The requests usually ask CIS/TK for a set of information, and instruct CIS/TK to place the information on a specific file that the user interface will access. Figure 4 depicts the steps followed in process of retrieving the system entities from CIS/TK. The name of the function performed in each box has been highlighted.

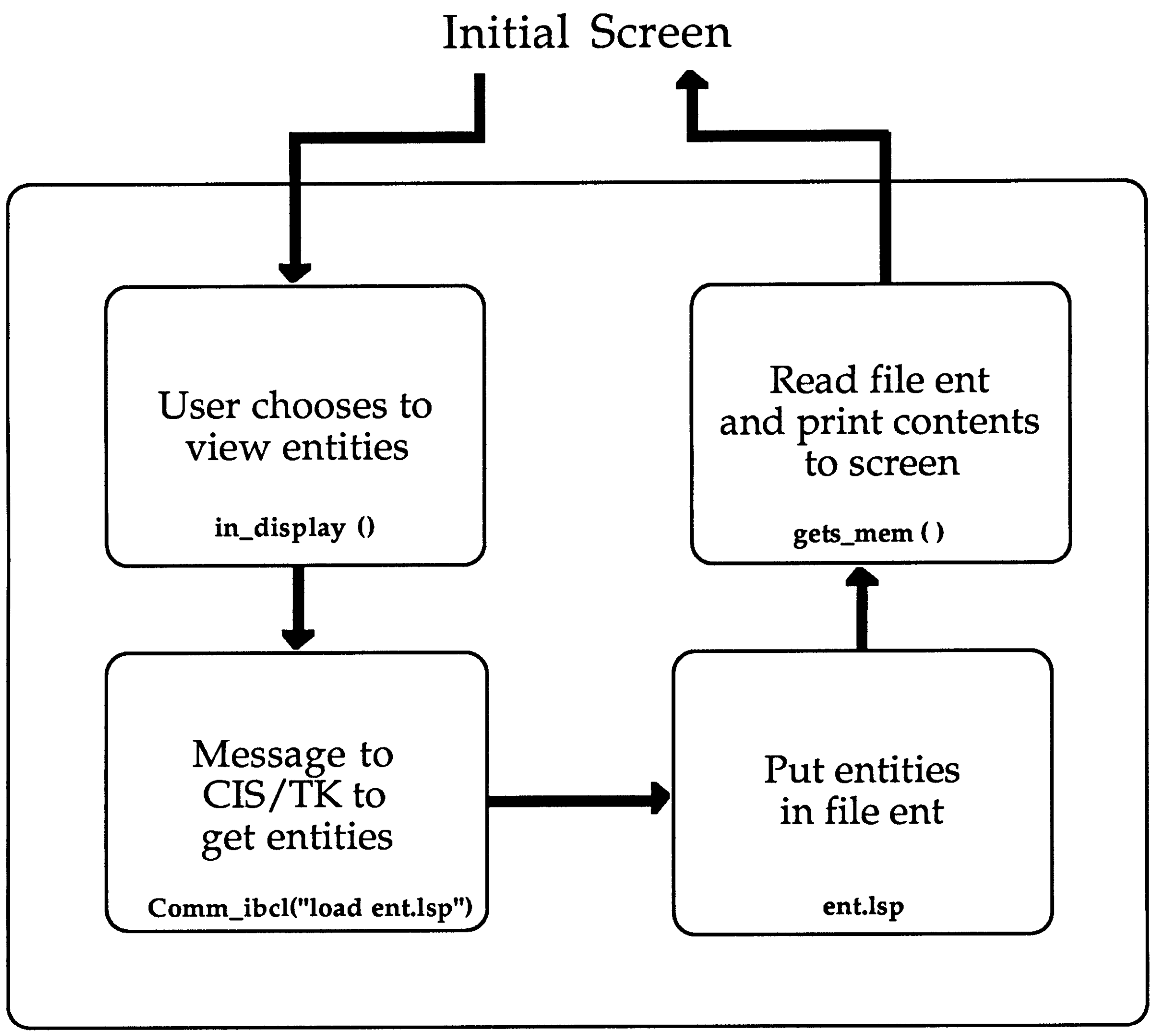
Initial Screen
User chooses to view entities
in_display ()
Message to CIS/TK to get entities
Comm_ibcl("load ent.lsp")
Put entities in file ent
ent.lsp
Read file ent and print contents to screen
gets_mem ( )

Figure 4. Flow Chart of Function to Retrieve Entities.

The function in_display () puts up the initial screen. It reads the command that the user wants to perform, and calls the appropriate function. Comm_ibcl() serves as the communication link between the user interface and CIS/TK. Comm_ibcl() takes as an argument the command that CIS/TK is going to perform. The command is sent to

CIS/TK via multi-cistk. The file ent.lsp instructs CIS/TK to retrieve the entities of the system and place them in the file ent. Att.lsp tells CIS/TK to retrieve the attributes of a given entity and put them in the file att.

The scheme provided in Table 3 has been implemented in the files att and ent, which provide the entities and attributes to the user interface:

Entities File -- ent	
first line	number of entities
2nd line	1st entity
nth line	nth - 1 entity

Attributes File -- att	
first line	entity name
2nd line	number of attributes
3rd line	1st attribute
nth line	nth -2 attribute

Table 3. Scheme for Fetching Entities and Attributes

The format for the two files is almost identical. In fact it only differs because the first member of the attribute file specifies the entity whose attributes are being retrieved. Gets_mem(), which is the function that retrieves and displays the information in the files ent and att, takes advantage of the similarities. It takes as arguments a file name, and a letter. The letter indicates whether the file is the entities file - 'e'- or the attributes file - 'a'. If the letter is 'e', gets_mem first retrieves the number of entities in the file, and then proceeds to retrieve the entities and to display them on the terminal.

If the argument is the letter 'a', gets_mem() first retrieves the name of the entity whose and the number of attributes possessed by the entity. Then gets_mem () proceeds to fetch the attributes of the entity, and to display them on the terminal. If the number of elements to be retrieved by gets_mem is greater than twenty, gets_mem automatically divides the elements into two columns in order to fit as many elements as possible on a single screen.

The next section presents a sample session depicting how to retrieve the entities and attributes that are available in the system.

3.1.2 Sample Session: Retrieving Entities and Attributes

The first function that is performed in this sample session is the retrieval of the entities from the system.

```
***********************************************************************
****************************** CIS/TK MENU ****************************
***********************************************************************

PLEASE SELECT THE FUNCTION THAT YOU WANT TO PERFORM
        1) Display entities that are available
        2) Display attributes of an entity
        3) Construct query
        4) Retrieve query
        5) Exit

CHOICE: 1

*****************************   ENTITIES   ****************************
        1) AM/FINANCIAL_INFO
        2) AM/COMPANY_DESC
        3) AM/INDUSTRY
        4) AM/SIC
***********************************************************************
```

The next option shown is the retrieval of the attributes of the entity AM/COMPANY_DESC. Please note that the first two options return to the initial screen after providing the user with the data requested.

```
**********************************************************************
***************************  CIS/TK MENU  ****************************
**********************************************************************

PLEASE SELECT THE FUNCTION THAT YOU WANT TO PERFORM
        1) Display entities that are available
        2) Display attributes of an entity
        3) Construct query
        4) Retrieve query
        5) Exit

CHOICE: 2


PLEASE SELECT THE ENTITY:

*****************************  ENTITIES  *****************************

        1) AM/FINANCIAL_INFO
        2) AM/COMPANY_DESC
        3) AM/INDUSTRY
        4) AM/SIC

**********************************************************************

CHOICE: 2

****************************  ATTRIBUTES  ****************************

For the entity selected the attributes are:
        1) ACOMPANY_CODE
        2) ACOMPANY_NAME
        3) AINDUSTRY
        4) AINCORPORATED
        5) APRIMARY_CODE

*********************************************************************
```

3.2 Query Generation

The user interface implemented allows the user to specify a query to retrieve information from the system. This section describes how the user is helped in the generation of queries, and describes the mechanism used to save queries that CIS/TK users have generated.

3.2.1 Generating a Query

The generation of a query is a three step process:

-- The user is presented with the entities in the system. He is first asked to select the entities on which the constraints should be placed.

-- The user is presented with the attributes of each entity. Then he is asked to select the attributes to place in the query.

-- The user is then asked to place constraints on each particular attribute.

The function format_attributes provides the user with the format that each attribute constraint requires. The function looks up the attribute format in the file sample_formats. The file sample_formats contains the sample formats for all the attributes in the system. This file must be updated if new attributes are added to the system.

At this point the user is asked if he would like to save his query. The query saving mechanism that has been implemented is described in the next section. Should the user decide not to save the query, the query will be placed in the file temp. The reason for having the queries in files is that it simplifies the process of loading queries into CIS/TK. Figure 5 presents a sample query which obtains financial information for the Honda Motor Company.

```
(AQP  '(JOIN (AM/COMPANY_DESC
                  (((AM/COMPANY_DESC ACOMPANY_CODE)
                        (AM/COMPANY_DESC ACOMPANY_NAME))
                        (AND (= (AM/COMPANY_DESC ACOMPANY_NAME)
                                "HONDA MOTOR CO LTD"))))
             (AM/FINANCIAL_INFO
                  (((AM/FINANCIAL_INFO AYEAR)
                        (AM/FINANCIAL_INFO ANET_INCOME)
                        (AM/FINANCIAL_INFO AREVENUE))))))
```

Figure 5. Sample Query

Queries to CIS/TK follow a specific format. If a query contains more than one entity, a join must be performed. The function format_query retrieves the constraints selected by the user and organizes them in a format that is acceptable by CIS/TK. After the formatting is done, the user can choose to load his query into the system, and retrieve the information requested.

3.2.2 Saving/ Retrieving Queries

In case the user desires to save his query, a scheme that is represented by Figure 6 is followed. The queries are first organized by the user-name of the person that builds them. The function qgetname() has been developed to retrieve the user-name of the person that is building the query. Qgetname uses the Unix facility getenv(), to fetch the environment variable LOGNAME. The function nsearch() is called to check if the user is already in the User Table. Nsearch() takes as arguments a name, and a character. The name indicates the username to find in the table. The character indicates the disposition of the entry if the name is found in the table. If the character is 'k', the entry is kept. If the character is 'd', the first character in the name is changed to a '!', which indicates that the name has been deleted from the table. Nsearch() either returns: -1 if the user is not on the table, or the index of the location of the user in the table.

A user can be added to the User Table by calling the function ninsert(). Ninsert uses a hash function to calculate the index in the table of the user. The index is calculated by taking the product of the ascii value of the characters in the username, dividing this number by the size of the table, and taking the remainder. If a user is already at this location, ninsert() keeps on increasing index by one until an empty spot is found on the table. If the table is full, ninsert() returns -1.

The current size of the User table is 25. The variable NUMBER_USERS in the file global.h determines the size of the table. The file global.h contains the variables that must be modified if the user interface is transferred to a new location.

Each user in the User Table owns an individual query list. The query list contains the names of the queries that a user has created. The function add_query() adds a query name to the end of the existing query list for a user. The names of the queries are placed in the query list in their order of creation. For example the user wmartinez in Figure 6 has created the queries honda_financial and alumni_industries.

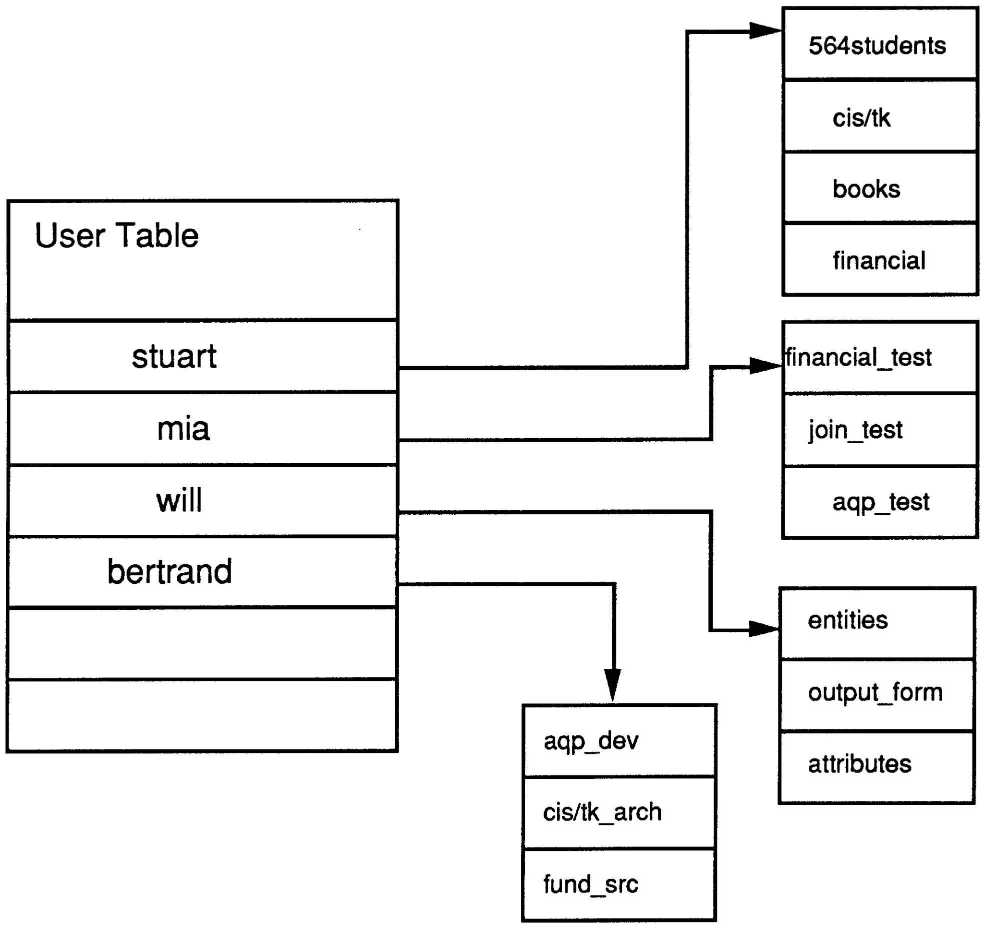

Figure 6. Query Saving Mechanism

The query names are preserved in the query list, but the actual queries are saved to files. The naming scheme used to save the queries is as follows: the queries will be saved as queryname.username. The queryname is the name assigned by the user, the username represents the logname of the person that is creating the queries. The naming scheme avoids the possible conflicts that could arise if two users name use the same name for their queries.

Saving the queries in files has two advantages: queries are readily available for loading into CIS/TK by a simple command e.g.. (load "filename"), and saving the queries in files allows the user to easily transfer queries to other members of his group that have access to CIS/TK.

The mechanism implemented to save queries, requires a process to intialize the User table and the query tables. Startup() is called every time a user interface is loaded. The names of the users that have created queries are kept in the file USERDATA while the name of the queries that have been originated are preserved in the file QUERYDATA. Queries in the QUERYDATA file are indexed by the name of the username that created them. Startup() reads the contents of the USERDATA file, and puts the names in the User Table. Startup() then fetches the contents of the file QUERYDATA and places them in the appropriate query table.

When a user interface is shutdown, a similar house-keeping process must be invoked. Shutdown() saves the names in the user table to the USERDATA file, and the queries within the query table are to the QUERYDATA file. The functions startup() and shutdown() are included in Appendix A.

3.2.3 Sample Session: Building and Saving Queries

The session below provides a sample session in which the user builds, and then saves a query. In the sample provided below, the user selects to save the query as honda_revenue. After putting the query name in the appropriate file, the system confirms that the query has been saved.

```
PLEASE SELECT THE FUNCTION THAT YOU WANT TO PERFORM
      1) Display entities that are available
      2) Display attributes of an entity
      3) Construct query
      4) Retrieve query
      5) Exit

CHOICE:     3

PLEASE SELECT AN ENTITY:
      1) AM/COMPANY_DESC
      2) AM/FINANCIAL_INFO
      3) AM/INDUSTRY
      4) AM/SIC

CHOICE(type 'd' when done): 2


FOR  ENTITY AM/COMPANY_DESC, PLEASE SELECT AN ATTRIBUTE:
      1) ACOMPANY_CODE
      2) ACOMPANY_NAME
      3) AINDUSTRY
      4) AINCORPORATED
      5) APRIMARY_CODE

CHOICE (type 'd' when done): 1
ATTRIBUTE: ACOMPANY_NAME
FORMAT: COMPANY NAME (in CAPS)
CHOICE: HONDA.

FOR  ENTITY AM/COMPANY_DESC, PLEASE SELECT AN ATTRIBUTE:
      1) ACOMPANY_CODE
      2) ACOMPANY_NAME
      3) AINDUSTRY
      4) AINCORPORATED
      5) APRIMARY_CODE

CHOICE (type 'd' when done): d
```

```
PLEASE SELECT AN ENTITY:
      1) AM/COMPANY_DESC
      2) AM/FINANCIAL_INFO
      3) AM/INDUSTRY
      4) AM/SIC

CHOICE(type 'd' when done): 1

FOR  ENTITY FINANCIAL_INFO, PLEASE SELECT AN ATTRIBUTE:
      1) ANET_INCOME
      2) ADATE
      3) ACURRENCY_FROM
      4) ACURRENCY_TO
      5) AREVENUE

CHOICE(type 'd' when done):   2
ATTRIBUTE: ADATE
FORMAT: DAY MONTH(FIRST 3 CHRACTERS) YEAR(LAST TWO DIGITS) -- no space
between day month and year.
CHOICE: 31DEC80


FOR  ENTITY FINANCIAL_INFO, PLEASE SELECT AN ATTRIBUTE:
      1) ANET_INCOME
      2) ADATE
      3) ACURRENCY_FROM
      4) ACURRENCY_TO
      5) AREVENUE

CHOICE(type 'd' when done):   5
ATTRIBUTE: AREVENUE
FORMAT: INTEGER
CHOICE: d


PLEASE SELECT AN ENTITY:
      1) AM/COMPANY_DESC
      2) AM/FINANCIAL_INFO
      3) AM/INDUSTRY
      4) AM/SIC

CHOICE(type 'd' when done): d

PLEASE SELECT THE FUNCTION THAT YOU WANT TO PERFORM
      1) Revise query
      2) Save query
      3) Display query output.

CHOICE:      2

PLEASE ENTER THE NAME OF THE QUERY:honda_revenue

FOR USER wmartinez QUERY honda_revenue HAS BEEN SAVED.
```

The next sample session demonstrates the retrieval of a saved query. For demonstration purposes, the queries named honda_financial and alumni_industries have been pre-defined for the user WMARTINEZ.

```
PLEASE SELECT THE FUNCTION THAT YOU WANT TO PERFORM
      1) Display entities that are available
      2) Display attributes of an entity
      3) Construct query
      4) Retrieve query
      5) Exit

CHOICE:      4

THE QUERIES THAT CAN BE RETRIEVED ARE:
      1) honda_financial
      2) alumni_industries

CHOICE:      1

THE QUERY RETRIEVED WAS honda_financial.
```

3.3 Information Retrieval

CIS/TK generates the solution to queries in the form of LISP frames. In order to understand the mechanisms that have been implemented in the Information Retrieval module, it is necessary to have some knowledge about the nature of the frames. This section provides a brief description of their functionality, and presents the methodology used to fetch the information that they contain.

3.3.1 CIS/TK Generated Objects

In order to generate the solution to a query that it has received, CIS/TK must locate the data that satisfies the constraints placed by the query. Every time an instance that meets the requirements is found, CIS/TK creates one or more frames. The number of frames created for each instance equals the number of entities in the query.

A frame can be described as a list of elements that characterizes an object. CIS/TK puts in the frames the different values of the attributes that describe the query received. Figure 7 provides a sample frame that was generated as a result of sending the query created in section 3.1 to CIS/TK. Since the query places constraints on two entities, the number of frames created for each instance will be equal to two. One frame will be created for the AM/FINANCIAL_INFO entity, and another frame will be created for the AM/COMPANY_DESC. The frames can be traced to each other through the value of the attribute RELATIONS within the frames.

```
(AM/FINANCIAL_INFO8
    (SUPERIORS (MULTIPLE-VALUE-F T) (VALUE AM/FINANCIAL_INFO))
    (INSTANCE-OF (MULTIPLE-VALUE-F T) (VALUE AM/FINANCIAL_INFO))
    (RELATIONS (VALUE AM/COMPANY_DESC7))
    (AYEAR (VALUE "31-12-85"))
    (ANET_INCOME (VALUE "146502.0"))
    (AREVENUE (VALUE "2909574.0")))
```

Figure 7. Sample frame generated by CIS/TK

The name of frames are of the form entityXX, where XX is number that is unused by other frames. The number is generated by the LISP function Gensym. The name of the frame provided in Figure 7 is AM/FINANCIAL_INFO8. This classifies the frame as satisfying the constraints placed by the query on the AM/FINANCIAL_INFO entity . The attribute RELATIONS points out that AM/COMPANY_DESC7 complements the values that have been placed in AM/FINANCIAL_INFO8.

The values within the frame are indexed by means of keys. Each key represents an attribute of the query. It is possible to access the value of the attribute by retrieving the list that is indexed by the key. The next section describes the mechanism employed by the user interface to retrieve the values within the frames.

3.3.2 Fetching Information Created by a Query

LISP provides some facilities that facilitate the process of fetching information from frames. The functions that have been used are provided in Table 4.

`(get-object entity 'instances)`	return a list of the instances for the entity
`(assoc key frame)`	returns the list in frame that is indexed by key
`(nthcdr number frame)`	returns a list of the frame members starting at the location indicated by number.

Table 4. Functions for Fetching Information From Frames

The steps followed in the process of fetching the information created by a query is depicted in Figure 8, which is provided in the next page. Start_query retrieves the number of instances that exist for the entities present in the query and creates the file, current_ent. Current_ent contains the entities that are present in the query, and the attributes and number of instances for each entity.

After the query has been loaded into the system, the actual information retrieval takes place. The function read_frames is invoked to retrieve the information from the instances that have been created. Read_frames works the following way:

-- read the entities that are present in the query from the file current_ent.

-- for each entity: retrieve the current list of the instances. Call the function *nthcdr,* provided in Table 4, with number set equal to the number of instances for the entity before the query was loaded. This number was recorded in the file current_ent.

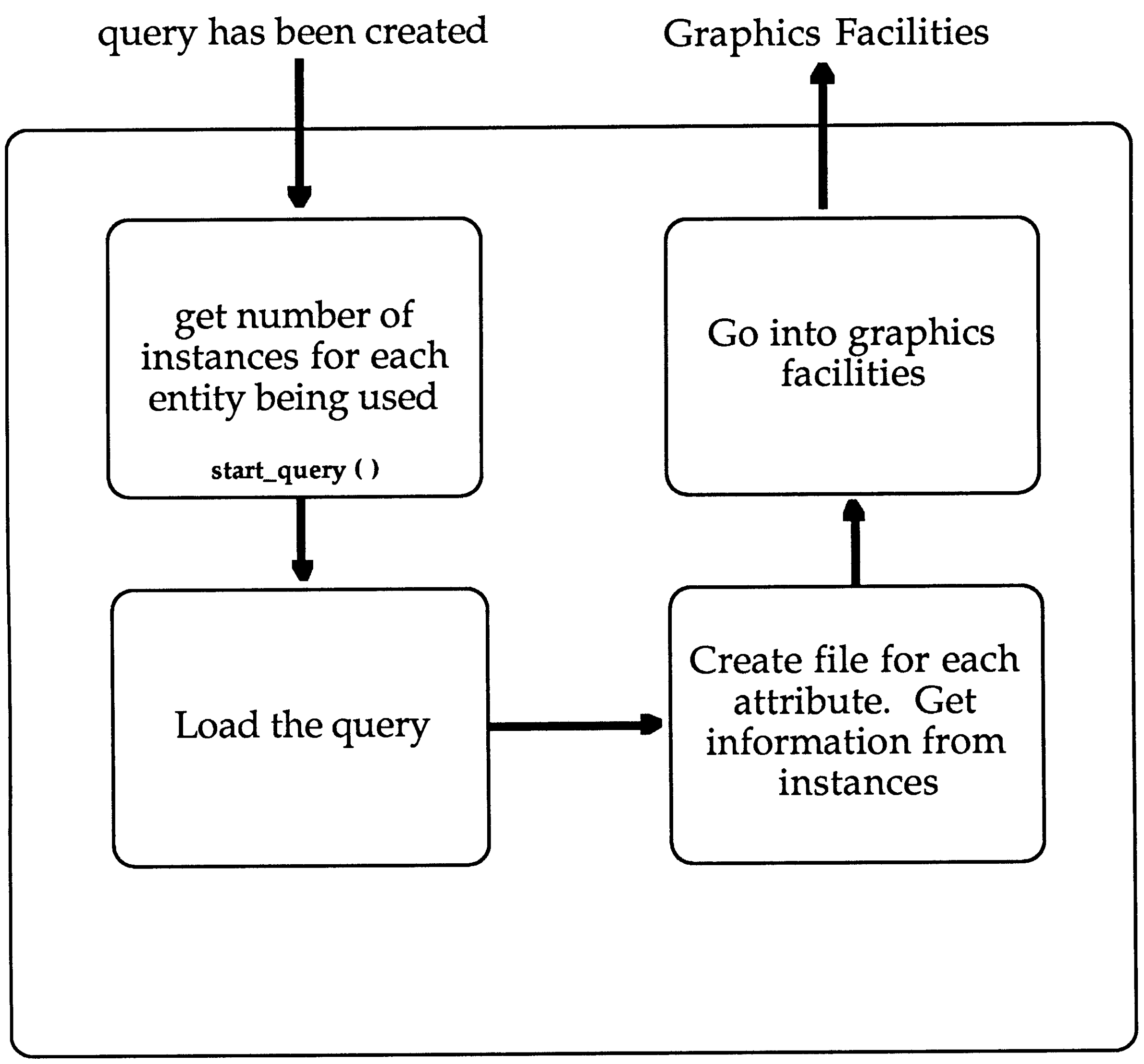
query has been created
Graphics Facilities
get number of instances for each entity being used
start_query ( )
Go into graphics facilities
Load the query
Create file for each attribute. Get information from instances

Figure 8. Flowchart of information retrieval module.

-- for each attribute in that entity: go through the new instances list and using the function *assoc* retrieve the lists indexed by the attribute. From these new lists retrieve the attribute values and write them to a file. The name of the file will be the same as that of the attribute.

The functions *fetch* and *fetch-attribute-range* provided in Table 5 accomplish the task described.

```
(defun fetch (key a-list)
  (cadadr (assoc key (cdr (get a-list 'frame)))))

(defun fetch-attribute-range (key entity start)
  (with-open-file (will_stream key :direction :output)
                  (let ((newlist
                         (nthcdr start (get-object entity 'instances))))
                    (dolist (instance newlist)
                            (print (fetch key instance) will_stream)))))
```

Table 5. Functions Used to Retrieve Information From Frames

Fetch takes an attribute and a frame as arguments, and returns the value of the attribute within that frame. Fetch-attribute-range takes as arguments an attribute, an entity and a number. The function retrieves the value of the attribute in the instances of the entity that start after 'number', and puts them in a file of the same name as the attribute provided. For example, if 'number ' provided equals twenty and the number of instances is twenty-five, fetch-attribute-range will retrieve the values of the attribute in instances twenty-one through twenty-five.

After the information is placed in the files, the output facilities can be invoked on the information retrieved. The next section describes the output facilities that are available in the user interface that has been implemented.

3.4 Output Facilities

This section describes the graphics packages that the user interface provides to CIS/TK. These packages aid the system user visualize the result of the query that has been sent to CIS/TK. Section 3.4.3 discusses the problems faced in finding graphics packages suitable to the CIS/TK environment.

3.4.1 Bargraph

The bargraph facility provides a two-dimensional plot of a set of data. Bargraph follows the following convention: the first column of data is plotted in the vertical axis, while the second column of data is displayed in the horizontal axis. The data obtained from CIS/TK by the information retrieval module needs to be modified in order to be used by Bargraph.

The first problem with the data is that values retrieved from LISP frames are enclosed in double quotes. The function parse() removes the double quotes. that surround the data. Another problem faced with the data is that some of the attributes do not have a value for a specific instance that satisfies the query constrains. The parsing module senses the value of the attributes, and removes the instances for which an attribute value equals nil.

Once the modifications describes above have been performed, the data is placed in the file *graph*. Graph serves as the intermediary between the information retrieval module and Bargraph. The system command **'cat graph | bargraph'** is issued to graph the contents of the graph file in bargraph format. A sample session showing the use of bargraph is provided in the next section.

3.4.1.1 Sample Session: Bargraph

This section provides a sample session depicting the use of the bargraph facility. Please note that the user is given the option to chose the data to put in each axis of the plot.

```
PLEASE CHOOSE THE FORMAT OF THE DATA
      1) Reports format
      2) Bar graph format
      3) Exit

CHOICE:    2

THE QUERY CONTAINS DATA FROM THE FOLLOWING ENTITES:
      A)  ACOMPANY NAME
      B) ANET_INCOME
      C) ADATE

PLEASE ENTER THE ORDER OF THE DATA
--> PLEASE ENTER THE HORIZONTAL AXIS: B
--> PLEASE ENTER THE VERTICAL AXIS: C


HONDA Motor Company

3602344.00 +                                         3602344
           |                                         %%%%%%%
           |                                 3498000 %%%%%%%
           |                                 %%%%%%% %%%%%%%
3381529.00 +                                 %%%%%%% %%%%%%%
           |                                 %%%%%%% %%%%%%%
           |                                 %%%%%%% %%%%%%%
           |                                 %%%%%%% %%%%%%% 3240983
3160714.00 +                                 %%%%%%% %%%%%%% %%%%%%%
           |                                 %%%%%%% %%%%%%% %%%%%%%
           |                         3098585 %%%%%%% %%%%%%% %%%%%%%
           |                         %%%%%%% %%%%%%% %%%%%%% %%%%%%%
2939899.00 +                         %%%%%%% %%%%%%% %%%%%%% %%%%%%%
           |         2907473         %%%%%%% %%%%%%% %%%%%%% %%%%%%%
           |         %%%%%%%         %%%%%%% %%%%%%% %%%%%%% %%%%%%%
           |         %%%%%%% 2804809 %%%%%%% %%%%%%% %%%%%%% %%%%%%%
2719084.00 + 2719084 %%%%%%% %%%%%%% %%%%%%% %%%%%%% %%%%%%% %%%%%%%
           | %%%%%%% %%%%%%% %%%%%%% %%%%%%% %%%%%%% %%%%%%% %%%%%%%
           ----------------------------------------------------------------
              1980    1981    1982    1983    1984    1985    1986
```

3.4.2 Report

The report facility provides a tabulated output of the information contained in the instances generated by a query. This facility allows the user to view the values of more than two attributes at the same time. The values of the attributes are separated in columns, with each the name of the attribute serving as the heading for the column.

The report facility allows the user the flexibility of specifying the ordering of the attributes in the report. It also gives the user the capability of saving the report to a file. The next section provides a sample session of the reports facility.

3.4.2.1 Sample Session: Reports

This section presents a sample session in which the reports facility is used. The query displayed is honda_financial. Please note that the user has the ability to write the report to a file for further modification.

```
PLEASE CHOOSE THE FORMAT OF THE DATA
      1) Reports format
      2) Bar graph format
      3) Exit

CHOICE:    1

THE QUERY CONTAINS DATA FROM THE FOLLOWING ENTITES:
      A) ACOMPANY NAME
      B) ANET_INCOME
      C) ADATE

PLEASE ENTER THE ORDER OF THE DATA (DELIMITER IS " "):
A C B

SAVE THE REPORT TO A FILE(Y/N): Y

NAME OF THE FILE: REPORTS.TXT

ACOMPANY_NAME      ADATE   ANET_INCOME
-------------      ----    -----------
Honda Motor Co.    1980      2719084
Honda Motor Co.    1981      2907473
Honda Motor Co.    1982      2804809
Honda Motor Co.    1983      3098585
Honda Motor Co.    1984      3498000
Honda Motor Co.    1985      3602344
Honda Motor Co.    1986      3240983
```

4. Conclusions and Recommendations

The work described in this thesis presents the first user interface developed for CIS/TK. One of the most important problems faced in the development of the interface was the lack of a graphics package based on the AT&T 3B2 environment that would work properly for many terminals. Bargraph was the only facility found that can display its output in a variety of terminals.

The feasibility of providing a file with information from CIS/TK that could readily be loaded into Lotus was examined. However, the file format used by Lotus is very complex and the amount of time left to implement this feature is insufficient. The output graphics facilities of the interface should be enhanced in future versions.

One of the goals of the system built was to have the module reside in the AT&T environment, independent of the CIS/TK and of the user environment. However, building a system that does not utilize some of the enhanced resources offered by the user environment has its setbacks. One possibility in the future is to develop separate interfaces that work for a specific user environment.

The rest of the user interface has been built in a systematic manner. This document provided an explanation of the module implemented. The rest of the interface can be adapted to other user environments with very little change.

Appendix A

This section contains the source code developed for the user interface. This page provides an introduction to the files, and a brief explanation of the task performed by each function.

File	Function	Task Performed
global.h	external declarations	values and locations of external variables for the user interface.
startcistk.c	startcistk	startcistk is invoked to start a user interface.
check_multi		check to see if multicistk is running in the background. If it is not, check_multi invokes it.
startup.c	startup()	initialize the user table. Read the user from USERDATA and the queries from QUERYDATA.
shutdown.c	shutdown()	called to save the values in the User table and the query tables to the files USERDATA and QUERYDATA.
multicistk.c	multicistk()	starts the multi-user cistk interface, and the CIS/TK. Opens the pipes necessary for the multi-user interface.
table_utils.c	add_query()	adds a query to the querytable of a CIS/TK user.
table_utils.c	ninsert()	adds a user to the existing User Table for CIS/TK. Returns -999999 if the user is not found.
table_utils.c	nsearch()	locates the index of a user in the User Table. Returns -999999 if the user is not found.
screens.c	ent_dpy()	gets the entities in CIS/TK.
screens.c	att_dpy()	gets the attributes of an entity.

screens.c	in_display()	puts up initial screen.

File	Function	Task Performed
screens.c	query_display()	puts up the query retrieve/delete menu.
query_utils.c	load_query()	loads a query into CIS/TK.
query_utils.c	retrieve_queries()	displays the names of the queries of a user on the terminal.
query_utils.c	view_query()	displays an actual query on the screen.
query_utils.c	remove_query()	remove a query from the querytable of a user.
graph_utils.c	format_file()	reads two files, and combines them in the format needed by Bargraph.
graph_utils.c	parse()	removes the double quotes that surround the data retrieved from CIS/TK
comm_ibcl.c	strtoint()	converts a string to its integer value.
comm_ibcl.c	gets_mem()	gets the members of the att and ent files and displays them on the terminal.
comm_ibcl.c	comm_ibcl()	sends commands to multi-cistk for CIS/TK to process.
comm_ibcl.c	select_att()	sends multi-cistk a message indicating which entity's attributes to load into the file att
screen_utils.c	qgetname()	gets the user name of the person that is using CIS/TK.
screen_utils.c	make_bold()	The text put up on the screen after this call will be bold and will blink.
screen_utils.c	unmake_bold()	undoes the effects of the call to make_bold().
screen_utils.c	start_screens()	takes a string as an argument. Centers the string and surrounds it with bold and blinking stars to indicate the beginning of a screen.

screen_utils.c	end_screen()	puts up a row of bold and blinking stars on the screen to indicate the end of a screen.

File	Function	Task Performed
utils.c	getsf()	gets a line from a file. Ends when a newline, or the prompt from CIS/TK -- cistk> -- is received.
utils.c	qgetnum()	gets the Queue number for a user.
utils.c	qopen()	opens a message queue.
utils.c	qget()	gets a message from the message queue.
utils.c	qput()	puts a message in the message queue for multi-cistk.
utils.c	qremove()	removes a message queue and utility pipes.
att.lsp		instructs CIS/TK to create the file att and place the attributes of an entity in the file.
ent.lsp		directs CIS/TK to retrieve the entities in the system and put them in the file ent
fetch.lsp	fetch()	fetch returns the list in frame that has the indicated attribute as a key
fetch.lsp	fetch-attribute-range()	fetch-attribute-range opens a file. It searches the frames in a specified range for occurrences of the attribute, and puts the values found in the file created.
top-level.lsp		function that changes the prompt in CISTK from '>' to 'cistk'.

```
            /***  GLOBAL.H  FILE ***/
#define NUMBER_USERS 100    /* users in the customer table */
#define MAXPATH 128
#include <stdio.h>

/* location of new prompt for cistk */
char load_prompt[MAXPATH] = "/usr/cistk/wmartine/thesis";

/* loation of cistk demo */
char demo_location[MAXPATH] = "/usr/cistk/demo/v2/fin-demo";

/* directory where files will be created and read */
char myhome[MAXPATH] = "/usr/cistk/wmartine/thesis";
```

```
/***     STARTCISTK    ***/
#include "global.h"
struct customer
{
  char name[10];
  int numqueries;
  struct querytable *querylist;
};

struct customer custbl[NUMBER_USERS];

main()
{
  /* invoke the initializing functions  */
  chdir(myhome);
  startup();

  /* check if multicistk is running in the  background */
  system("check_multi");
  in_display();
  shutdown();
}
```

```
###                  CHECK_MULTI FILE
### checks if multicistk is running in the background
### if multicistk is not running, the program calls it
### if multicistk is running, the program checks to see if
### multicistk is ready for input

BOLD=`tput smso`        # reverse screen
OFF=`tput sgr0`         # normal screen
BLINK=`tput blink`      # blinking
export BOLD OFF BLINK status
ibcl_run=`ps -e | egrep 'multicis' | wc -l`
if [ "$ibcl_run" -eq "0" ]   # check if multicistk is running in background
then
echo "$BLINK" ; echo "$BOLD"
echo
"**************************************************************************"
echo "***************       NOW LOADING CIS/TK
********"
echo "***************       IT WILL TAKE ABOUT 2 MINUTES
********"
echo "***************       PLEASE WAIT UNTIL MENU SCREEN APPEARS
********"
echo
"**************************************************************************"
echo "$OFF" ; echo "                    \c" ; date
multicistk > multi_log &      # invoke multiibcl in background

while true
do
        multi_loaded=`tail multi_log | egrep 'waiting ' | wc -l`
        if [ "$multi_loaded" -eq "0" ]   # check if multicistk is loaded
        then sleep 10
        else exit
        fi
done
else  # multicistk is running in the background
echo "$BLINK" ; echo "$BOLD"
echo
"**************************************************************************"
echo "***************            WELCOME TO CIS/TK
********"
echo "***************       PLEASE WAIT UNTIL MENU SCREEN APPEARS
********"
echo
"**************************************************************************"
echo "$OFF" ; echo "                    \c" ; date
while true
do
        multi_loaded=`tail multi_log | egrep 'waiting ' | wc -l`
        if [ "$multi_loaded" -eq "0" ]   # make sure it is ready for input
        then sleep 5
        else exit
        fi
done
fi
```

```
/***  STARTUP.C ****/
/** function called to initialize the tables that hold the users that
 have created queries, and the names of the queries that they have created
*/
#include "global.h"
struct customer          /* structure that holds the user */
{
  char name[10];
  int numqueries;
  struct querytable *querylist;
};

struct querytable       /* structure for queries */
{
  char qname[12];
  struct querytable *nextquery;
};
extern struct customer custbl[100];
startup()
{
  int i;
  char line[200], name[10],qname[], *getsf();
  FILE *fc, *fi, *fopen(), *test;
  setbuf(stdout,0);
   for(i=0;i<NUMBER_USERS;i++)    /* initialize tables to blanks */
    {
      custbl[i].name[0] = '\0';
      custbl[i].numqueries = 0;
      custbl[i].querylist = NULL;
    }
  fc = fopen("USERDATA","r");  /* open file USERDATA for reading */
  if (fc == NULL)    /* file does not exist */
    {
      printf("Could not open the file USERDATA!!\n\n");
      return;
    }
  while ( fgets(line, 200, fc) != NULL)
    {
      sscanf(line, "%s",name);    /* get name */
      i = ninsert(name);          /* insert it in the table */
    }
  fclose(fc);                     /* close file pointer */
  fi = fopen("QUERYDATA","r");    /* open QUERYDATA pointer */
  if (fi == NULL)                 /* no queries created */
    {
      printf("Could not open the file QUERYDATA!!\n\n");
      return;
    }
  while ( fgets(line, 200, fi) != NULL)
    {
      sscanf(line,"%s %s", name,qname);/* get name of user and queryname */
      add_query(name, qname);               /* add the query */
    }
  fclose(fi);
}
```

```
/*** SHUTDOWN PROGRAM -- ***/
/* called to update the files that contain the users that have created
   queries and the queries that they have created */
#include "global.h"

struct customer
{
  char name[10];
  int numqueries;
  struct querytable *querylist;
};

struct querytable
{
  char qname[12];
  struct querytable *nextquery;
};

extern struct customer custbl[100];
shutdown()
{
  int i;
  struct querytable *ptr;
  FILE *fi, *fc;
  /* open data files for writing */
  fc = fopen("USERDATA","r+");     /* open USERDATA file for writing */
  if (fc == NULL)                  /* if error return */
    {
      return(-1);
    }
  fi = fopen("QUERYDATA","r+");   /* open QUERYDATA for writing */
  if (fi == NULL)            /* if error return */
    {
      fclose(fc);            /* close the opened USERDATA file */
      return(-1);
    }
  /* start to loop through custbl */
  for(i=0;i<NUMBER_USERS;i=i+1)
    if (custbl[i].name[0] != '\0' && custbl[i].name[0] != '!')
      {
        /* write one line from custbl to USERDATA */
        fprintf(fc,"%s\n",custbl[i].name);
        /* write each querytable for a customer to QUERYDATA */
        ptr = custbl[i].querylist;
        while(ptr != NULL)
          {
            fprintf(fi,"%s %s\n",custbl[i].name,ptr->qname);
            ptr = ptr -> nextquery;
          }
      }
  /* close them files */
  fclose(fc);
  fclose(fi);
  return;
}
```

```
/*** Multicistk.c FILE ***/
#include "global.h"

main()
{
  int qnumber, mqueue;           /* queue number and queue identifier*/
  int from_pid;        /* process id of  user that sent command */
  FILE *pipefromibcl, *pipetoibcl; /* file ids for named pipes */
  char line[80];                  /* line returned from CISTK */
  char msg[80];        /* the message typed by the user(from stcistk) */
  char command[25], arg[25];    /*to hold command and first argument */
  char message[50];       /* to hold message for another user */
  extern char load_prompt[MAXPATH];        /* to hold load command */
  extern char demo_location[MAXPATH];     /* to hold demo location */

  char send_demo[MAXPATH], send_prompt[MAXPATH];
  struct user
    {
      char name[10];
      int pid;
    };

  int index;
  int true;
  setbuf(stdout, 0);      /* set stdout buffer size to 0 */
  printf("multicistk: started......\n");
  qnumber=qgetnum();      /* get the queue number */
  printf("multicistk: using queue number %d\n", qnumber);
  qremove(qnumber);       /* remove queue & pipes if they already exist */
  mqueue=qopen(qnumber);   /* open the queue */
  system("/etc/mknod pipetoibcl p");          /* create the named pipes */
  system("/etc/mknod pipefromibcl p");
  /* start cistk in background */
  system("ibcl <pipetoibcl >pipefromibcl &");
pipetoibcl=fopen("pipetoibcl","w");/* open pipe from multicistk to cistk */
  setbuf(pipetoibcl, 0);        /* no buffering in this pipe */
  /* pipe from cistk to multicistk */
  pipefromibcl=fopen("pipefromibcl","r");
  /* read startup messages from CISTK until > prompt received */
  while (strcmp(getst(line,pipefromibcl),">") != 0)
    {
      printf("%s\n", line);
    }

  strcpy(send_demo, " (load \" " );
  strcat(send_demo, demo_location);
  strcat(send_demo, " \") ");

  strcpy(send_demo, "(load \"/usr/cistk/demo/v2/fin-demo\") ");
  /* send message to ibcl directly */
  fprintf(pipetoibcl,"%s\n", send_demo);
  while (strcmp(getst(line,pipefromibcl),">") != 0)
    {
      printf("%s\n", line);
    }
  /* place where reset prompt routine is located */
  strcpy(send_prompt, "(load \" ");
  strcat(send_prompt, load_prompt);
  strcat(send_prompt, " \") ");
  strcpy(send_prompt, " (load \"/usr/cistk/wmartine/thesis/top-level\") ");

  /* send message to ibcl directly */
  fprintf(pipetoibcl,"%s\n", send_prompt);
  /* wait til next prompt */
  while (strcmp(getst(line,pipefromibcl), ">") != 0)
    {
      printf("%s\n", line);
    }

  /* send reset prompt command */
  fprintf(pipetoibcl,"%s\n", "(system:top-level)" );

  /* wait until new prompt */
  while (strcmp(getsf(line, pipefromibcl), "cistk>") !=0)
    {
      printf("%s\n", line);
    }

  true=1;

  /* Loop forever, until shutdown is read from CISTK */
  while(true==1)     /* start of big while loop */
    {
      printf("multicistk: waiting for msg from startcistk\n\n");
      from_pid=qget(mqueue,1,msg) /* read msg from queue from startcistk */
      /* dest=1 gets mesages for multicistk */
      printf("multicistk: msg from startcistk %d is    -- %s --\n",
             from_pid, msg);

      /* send message (received from startcistk) to CISTK */
      fprintf(pipetoibcl,"%s\n",msg);

      do  /* Keep receiving from CISTK until it has sent all its output */
        {  /* The output will be terminated by cistk> or    */
          printf("multicistk:waiting for output from CISTK\n");
          getsf(line,pipefromibcl);      /* read line from CISTK */
          printf("multicistk: msg from CISTK is ..%s\n",line);
          qput(mqueue, from_pid, line);

        }
      while(strcmp(line,"cistk>") !=0 && strcmp(line, " ->") !=0);
      /* if not ? next line */
      }    /* end of big while loop */
  qremove(qnumber);
  return;
}
```

```
/***      ADD_QUERY  ***/
/* function that takes adds a query of name tquery_name
   to the queries that the user tname has. */
#include "global.h"

add_query(tname, tquery_name)
     char tname[10],tquery_name[];
{
  struct customer
    {
      char name[10];
      int numqueries;
      struct querytable *querylist;
    };

  struct querytable
    {
      char qname[12];
      struct querytable *nextquery;
    };

  extern struct customer custbl[100];
  struct querytable *thisquery, *queryptr;
  int index;

  index = nsearch(tname,'k');         /* find user in customer table */
  if (index < 0) return (-999999);

  queryptr = (struct querytable *) malloc( sizeof(struct querytable));
  /* if name is found allocate a querytable */
  strcpy(queryptr->qname, tquery_name);
  queryptr -> nextquery = NULL;       /* structure and fill it in */

  custbl[index].numqueries++;          /* update number if queries */
  /* if first query just add directly */
  if (custbl[index].querylist == NULL)
       custbl[index].querylist = queryptr;  /* to the customer table */

  else
    {
      /* otherwise chain to the end of the querylist */
      thisquery = custbl[index].querylist;
      /* find end of the linked list */
      while (thisquery -> nextquery != NULL)
       {
          thisquery = thisquery -> nextquery;
        }
      thisquery -> nextquery = queryptr;
    }
  return(custbl[index].numqueries);       /* return number of queries */
}
```

```
/******          NINSERT.C                    ******/
/* function used to place a user in the Usertable */

#include "global.h"


ninsert (tname)
     char tname[10];
{
  int index,i, len, product;
  struct customer
    {
      char name[10];
      int numqueries;
      struct querytable *querylist;
    };

  extern struct customer custbl[NUMBER_USERS];

  /* COMPUTE HASH FUNCTION */
  len = strlen(tname);
  for(product=1, i=0; i<len-1; i++)
    product *= len *tname[i];
  i=index = product%NUMBER_USERS;

  /* SEARCH FOR EMPTY SPOT OR END OF TABLE */
  while ( i<NUMBER_USERS && custbl[i].name[0]!='\0' )
    { i = i + 1; }

  /* TEST IF EMPTY SPOT FOUND IN TABLE */
  if(i<NUMBER_USERS)
    {
      /* INSERT NAME INTO TABLE */
      strcpy(custbl[i].name,tname);
      custbl[i].numqueries = 0;
      return(i);
    }

  /* OTHERWISE INDICATE NO ROOM FOR INSERT */
  return(-1);
}
```

```
                /***   NSEARCH.C  ***/
/** search for a customer and return his index number in the USERTABLE
    A hash function is used to place the user in the table, and to retrieve
   the user from the table. If tdidp is equal to 'd', user will be deleted
   from the table **/

#include "global.h"

nsearch(tname,tdisp)
     char tname[10],tdisp;
{
  struct customer
    {
      char name[10];
      int numqueries;
      struct querytable *querylist;
    };
  extern struct customer custbl[NUMBER_USERS];

  int index, entries_searched;   /*** index provided by hash function ***/
  int flag=0;                              /*** index used in loop ***/
  entries_searched=0;                      /*** upper limit on loop ***/

  index = (tname[0] * tname[1]) % NUMBER_USERS ; /* compute hash index */

  for(index; entries_searched < NUMBER_USERS; index++, entries_searched++)
    {
      /* check if name are equal */
      if (strcmp(tname, custbl[index].name) == 0)
        {
          if (tdisp == 'd')              /* check disposition */
            custbl[index].name[0] = '!';
          return(index);                 /* return index number */
        }
      if (index == NUMBER_USERS)        /* for wraparound in the table */
        index = 0;           /* go to zero location and keep searching */
      if (custbl[index].name[0] == '\0')
        break;
    }
  if (flag!= 1)     /* customer not found */
    return(-999999);
}
```

```
/***    ATT_DPY()    ***/
#include <stdio.h>
att_dpy()
{
  int att_choice, number_entities;
  char current_entity;
  printf("\n\nPLEASE SELECT THE ENTITY:\n\n");
  number_entities = ent_dpy();

  printf("CHOICE: ");
  while(1)
    {
      scanf("%d", &att_choice);
      if (att_choice > 0 && att_choice <= number_entities)
        break;
      else
        {
          printf("INVALID SELECTION.PLEASE");
          printf(" ENTER A NUMBER FROM 1 TO %d\n", number_entities);
          printf("\nCHOICE: ");
        }
    }
  printf("\n\n");
  /* call function that inquires for attributes */
  select_att(att_choice);
  start_screen("ATTRIBUTES");

  gets_mem("att", 'a');    /* get the attributes from file att */
  printf("\n\n");
  end_screen();
  printf("\n\n");
  return;
}

/***   ENT_DPY()    ***/
ent_dpy()
{  /* load function that requests entities */
  int i, number_entities;
  comm_ibcl( "(load \"ent.lsp\")" );
  printf("\n\n");
  start_screen("ENTITIES");

  /* get entities from file ent */
  number_entities = gets_mem("ent", 'e');
  printf("\n\n");
  end_screen();
  printf("\n\n");
  return(number_entities);
}
```

```
/***   In Display   ***/
in_display()
{
  int in_choice;

  while(1)
    {
      printf("\n\n");
      end_screen();
      start_screen("CIS/TK MENU");
      end_screen();
      printf("\nPLEASE SELECT THE FUNCTION THAT YOU WANT TO PERFORM\n");
      printf("\t1) Display entities that are available\n");
      printf("\t2) Display attributes of an entity\n");
      printf("\t3) Construct query\n");
      printf("\t4) Retrieve query\n");
      printf("\t5) Exit\n");
      printf("\nCHOICE: ");
      scanf("%d", &in_choice);
      if (in_choice > 0 && in_choice < 6)
        break;
      else
        printf("INVALID SELECTION.PLEASE ENTER A NUMBER FROM 1 to 5\n");
    }

  switch(in_choice)
    {
    case 1:
      ent_dpy();
      in_display();
      break;
    case 2: att_dpy();
      in_display();
      break;
    case 3: query_make();
      break;
    case 4: query_display();
      break;
    case 5: printf("Have a nice day\n");
      printf("CIS/TK version 2\n");
      return;
    }
  return;
}
```

```
/***   QUERY DISPLAY   ***/
void query_display()
{
  int in_choice;
  while(1)
    {
      printf("\n\n");
      start_screen("QUERY RETRIEVE/DELETE MENU");
      printf("\nPLEASE SELECT THE FUNCTION THAT YOU WANT TO PERFORM\n");
      printf("\t1) View a query\n");
      printf("\t2) Delete a query\n");
      printf("\t3) Load query\n");
      printf("\t4) Return to Main Screen\n");
      printf("\nCHOICE: ");
      scanf("%d", &in_choice);
      if (in_choice > 0 && in_choice < 5)
        break;
      else
        printf("INVALID SELECTION.PLEASE ENTER A NUMBER FROM 1 to 4\n");
    }

  switch(in_choice)
    {
    case 1: retrieve_queries();
      query_display();
      break;
    case 2:retrieve_queries();
      query_display();
      break;
    case 3:
      load_query();
      in_display();
      break;
    case 4: in_display();
      return;
      break;
    }
  return;
}
```

```
/*** TABLE_UTILS.C FILE ***/
#include <stdio.h>
retrieve_queries()
{
  int i, in_choice;
  char *name, *qgetname(), choice;
  name = qgetname();                /* get username */
  i = nsearch(name, 'k');           /* check if user has queries */
  if (i == -999999)                 /* user does not have queries */
    {
   printf("USER %s HAS NOT CREATED ANY QUERIES\n", name);
   printf("Would you like to access the queries of another user? (y/n): ");
      while(1)
        {
          getchar();               /* to get return */
          choice = getchar();      /* get user's choice */
          if (choice == 'y' || choice == 'n')
            break;                 /* if valid choice, continue */
          else                     /* otherwise, loop until valid choice */
            printf("\nINVALID SELECTION. PLEASE ENTER A 'y' or 'n': ");
        }

      if (choice == 'y')
        {
          while(1)
            {
             printf("\nPlease enter the name of the user, or q to exit: ");
              scanf("%s", name);    /* user can view other users queries */
              if (strcmp(name, "q") == 0)  /* quit option */
                return;
              else
                break;
            }
          i=nsearch(name, 'k');     /* see if other user has queries */
          if (i < 0)                /* if not break */
            {
              printf("User does not exist in the table\n");
              printf("...Returning to the main menu\n\n\n");
              return;
            }
          else                     /* otherwise display that users queries */
            {
              i = display_queries(name);
              while(1)
                {
                  printf("\nCHOICE: ");
                  scanf("%d", &in_choice);
                  if (in_choice > 0 && in_choice <= i)
                    {
                      view_query(in_choice);
                      break;
                    }
                  else
                    {
                     printf("Invalid choice.");
                    printf("Please enter a number between 1 and %d\n", i);
                   }
                 }
               return;
              }
          }
        else
          {
            printf("No queries retrieved. Return to main menu\n\n\n");
            return;
          }
      }
    else
      {
        i = display_queries(name);
        while(1)
          {
            printf("\nCHOICE: ");
            scanf("%d", &in_choice);
            if (in_choice > 0 && in_choice <= i)
              {
                view_query(in_choice);
                break;
              }
            else
              {
                printf("Invalid choice.");
                printf("Please enter a number between 1 and %d\n", i);
              }
          }
      }
    return;
}
```

```
display_queries(name)
     char name[];
{
  int i, choice;
  char header[];
  int index = nsearch(name, 'k');
  struct customer
    {
      char name[10];
      int numqueries;
      struct querytable *querylist;
    };

  struct querytable
    {
      char qname[12];
      struct querytable *nextquery;
    };
  extern struct customer custbl[100];
  struct querytable *thisptr;
  thisptr = custbl[index].querylist;
  if (thisptr != NULL)
    {
      printf("\n\n");
      make_bold();
      printf("*************************  USER %s  ****************", name);
      printf("*************\n");
      unmake_bold();
      i = 1;
      while (thisptr != NULL)
        {
          printf("\t%d) %s\n", i, thisptr->qname);
          i++;
          thisptr = thisptr->nextquery;
        }
    }
  else
    printf("User %s has not created any queries\n\n", name);
  return(i);
}
```

```
#include <stdio.h>
/* function that takes two files as arguments, and produces the
   file file graph, which can readily be used to load into bargraph */
format_file (file1, file2, number)
     char file1[80], file2[80];
     int number;
{
  int i, flag;
  char first_mem[12], second_mem[12], *parse();
  FILE *fp1, *fp2, *new;
  fp1=fopen(file1, "r");     /* open the files for reading */
  fp2=fopen(file2, "r");
  new=fopen("graph", "w");   /* open graph to hold values of both files */
  i=1;
  flag=0;   /* set if any value is NIL */
  while(i <= number)  /* loop until all strings are received */
    {
      fscanf(fp1, "%s", first_mem);  /* get string  from first file */
      fscanf(fp2, "%s", second_mem); /* get second from second file */

      /*if either one is equal to NIL, set flag, do not write values*/
      if (strcmp(first_mem, "NIL")==0 || strcmp(second_mem, "NIL")==0)
        flag=1;
      else
        {
          fprintf(new, "%s\t", parse(first_mem));   /* write the values */
          fprintf(new, "%s\n", parse(second_mem));
        }
      i++;   /* keep looping */
    }
  if(flag)   /* flag is set ?? */
    printf("Some of the values have been ommitted from the graph\n");
}


/* data received from CIS/TK is enclosed in double qoutes. This program
   removes the double quotes. After the data is 'parsed' it can be used
   by the graphics packages */
char *parse(old)
     char old[];
{
  char tmp[];
  int i, j;
  int len=strlen(old);

  for (i=0; i<len; i++)   /* parse out the " and the - fro data */
    {
      if (old[i] == '"' || old[i] == '-')
        old[i] = ' ';
    }

  strcpy(tmp, old + 1);  /* remove unnecesary first space */
  return(tmp);
}
```

```
/*** COMM_IBCL.C FILE ***/
#include <stdio.h>

/*** String to Integer Conversion
 *** developed 3/31/89 wmartinez ***/
strtoint(alpha)
     char alpha[];      /* string to be converted */
{
  int beta;             /* variable that will hold integer value */
  int j;                /* counter */
  beta=0;
  for (j=0; alpha[j] >= '0' && alpha[j] <= '9'; j++)
    beta=10 * beta + (alpha[j] - '0');  /* convert from ascii to integer */
  return(beta);         /* return integer value */

}


/*** function that gets from a file a number of strings. The first
  string in the file specifies the number of strings to retrieve ***/
gets_mem(filename, select)
     char filename[20];        /* name of the file that contains strings */
     char select;              /* entity or attribute file */
{
  char current_mem[80],current_mem2[80] , nb_mem[], entity_sel[];
  int i, number_members;
  FILE *fp;
  fp=fopen(filename, "r");
  printf("\n\n");

  if (select == 'a')
    { /* get the entity selected and the number of attributes*/
      fscanf(fp, "%s", entity_sel);
      fscanf(fp, "%s",nb_mem);
      number_members = strtoint(nb_mem); /* convert to an integer */
      printf("\For the entity selected the attributes are:\n");
    }

  if (select == 'e')
    {
      fscanf(fp, "%s", nb_mem);
      number_members = strtoint(nb_mem); /* convert to an integer */
    }

  if (number_members > 10)
    {
      for (i=0; i< (number_members-2); i+=2)
        {
          fscanf(fp, "%s %s", current_mem, current_mem2);
          if (strlen(current_mem) >10)
            printf("\t%d) %s \t\t%d) %s\n", i+1, current_mem,
                   i+2, current_mem2);
          else
            printf("\t%d) %s \t\t\t%d) %s\n", i+1, current_mem,
                 i+2, current_mem2);
        }

      if(number_members%2 ==0)
        printf("\t%d) %s \t\t\t%d) %s\n", i+1, current_mem,
               i+2, current_mem2);
      else
        printf("\t%d) %s\n", i+1, current_mem);
    }

  else
    {
      for (i=0; i< number_members; i++)
        {
          fscanf(fp, "%s", current_mem);
          printf("\t%d) %s\n", i+1, current_mem);
        }
    }
  fclose(fp);

  if (select=='e')
    system("rm -r ent");

  if (select=='a')
    system("rm -r att");

  /* return number of entities or attributes that were retrieved */
  return(number_members);

}
```

```
/*** This function serves as the message sender between
  the cis/tk system and the user interface
  Modified version of startcids    ***/

comm_ibcl(line)
     char line[];
{

  int qnumber, mqueue; /* the queue number and queue identifier */
  int pid, multi_pid;  /* process ids of this process and multicistk */
  char msg[80];        /* the message from multicistk */
  char command[25], arg[25]; /* for tasks 10 & 11 */
  FILE *fp;            /* for task 11 */
  fp = NULL;           /* for task 11 */

  setbuf(stdout, 0);   /* set stdout buffer size to 0 */
  qnumber=qgetnum();   /* get the queue number */
  mqueue=qopen(qnumber);   /* open the queue */

  pid=getpid();        /* get process id for this process */

  /* Loop to read CISTK commands and send the to multicistk to process */

  /* invoke a new shell -- task 8 */
  if (strcmp(line,"!")==0)
    {
      system("sh -i");
      printf("\ncistk>"); /*print prompt to the user */
    }

  /* check for quit password -- task 10 */
  command[0]=arg[0]='\0'; /* initialize both the '/0' */
  sscanf(line,"%s %s",command,arg);
  if (strcmp(command,"quit")==0 && strcmp(arg,"foobar")!=0)
    {
      printf("\nsorry your quit password is invalid.\n");
      printf("\ncistk>");
    }

  /* check for redirection of output command -- task 11 */
  if (strcmp(command,">")==0)
    {
      if (fp != NULL) fclose(fp); /* close file */
      /* if file given, open it for writing */
      if (strcmp(arg,"")!=0) fp=fopen(arg,"w");
    }

  qput(mqueue,1,line); /* send command to multicistk (dest=1) */

  /* check for disconnect command -- task 16 */
  if (strcmp(line,"disconnect")==0)
    {
      printf("\nstartcistk: startcistk is disconnected.\n");
      return; /* return to Unix */
    }
  do /* keep receiving from multicistk until "cistk>" is received */
    {
      /* receive a message from multicistk with dest=pid */
      multi_pid=qget(mqueue,pid,msg);
      if (strcmp(msg,"shutdown")==0)
        {
          printf("\nstartcistk: multicistk has been shutdown.\n");
          return; /* return to Unix */
        }
      /* redirect output -- task 11 */
      if (fp != NULL)
        {
          fprintf(fp,"\n%s",msg);
        }
    } while (strcmp(msg,"cistk>")!=0 && strcmp(msg," ->")!=0);
  /* if not cistk> or ->, get next msg */
} /* end of big while loop */



/***   SELECT_ATT   ***/
select_att(selection)
     int selection;          /* entity that user selects */
{
  FILE *fp;
  fp=fopen("att.lsp", "w");
  fprintf(fp, "%s %d %s",
          "(with-open-file (will_stream \"att\"  :direction :output)
               (let* ((entity_sel (nth", selection-1, "(get-object 'entity
            'subordinates))) (atts (get-object entity_sel 'attributes))
(nb-attributes (length atts)))(print entity_sel will_stream)
(print nb-attributes will_stream)(dolist (entity atts)
                                    (print entity will_stream))))" );
  fclose(fp);
  comm_ibcl( "(load \"att.lsp\")" );
}


/* QGETNAME  Function that gets the username to be used from the
   shell variable LOGNAME */
char *qgetname()
{
  char *c;    /* pointer to the first character of the logname */
  char *getenv();
  c=getenv("LOGNAME"); /* the value of the shell variable LOGNAME */
  if (c==NULL)
    {
      printf("qgetnum: the shell variable LOGNAME must be set.\n");
      printf("Program exits.\n");
      exit(0);
    }

  return (c);
}
```

```
/***  SCREEN UTILS  ***/
make_bold()
{
       system(" tput blink ");    /* blinking cursor */
       system(" tput smso  ");    /* reverse screen */
 }


unmake_bold()
{
  system(" tput sgr0 ");          /* turn bold & blink off */
}


/* function that takes a string as an argument and enters it, surrounded
   by starts. It also makes the heades bold and blink */
start_screen(header)
     char header[];
{
  int i;
  int len = strlen(header);

  make_bold();
  for(i=0; i <= (65 - len)/2; i++)
    printf("*");
  printf("  %s  ", header);

  i = (65 - len)/2 + len + 4;
  for(i; i < 69; i++)
    printf("*");
  printf("*\n");
  unmake_bold();
}

end_screen()
{
  int i;
  make_bold();
  for(i = 0; i <69; i++)
    printf("*");
  printf("*\n");
  unmake_bold();
}
```

```
/* GETSF: reads a line from a file */
/*        returns when a newline or the prompt cistk> is read */

#include <stdio.h>

char *getsf(buf,fp)
char *buf;
FILE *fp;
{
  int bytes, fd;
  int i=0;

  fd = fileno(fp);
  while(i<640)
    {
      bytes=read(fd,buf+i,1);
      if (bytes == -1)
        return (NULL);
      if (buf[i] == '\n') break;
      i++;
      if (i==6 && strncmp(buf,"cistk>",6) == 0) break;
      if (i==3 && strncmp(buf," ->", 3) == 0) break;
    }
    buf[i]='\0';
  return(buf);
}
```

```
/* UTILS.C - entered 2/15/89 */

/* message queue utility routines */

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct qmessage    /* what each message looks like */
{
  long mdest;      /* the message destination */
  char mtext[80];  /* the text of this message */
  int mpid;        /* the process id of the sender */
};

/* QGETNUM */
/* This routine qgetnum gets the queue number to be used from the */
/* shell variable QUEUE. */
/* It returns the queue number to the calling program. */

qgetnum()
{
  char *c;    /* pointer to the first character of the queue number */
  char *getenv();
  c=getenv("QUEUE"); /* the value of the shell variable QUEUE */
  if (c==NULL)
    {
      printf("qgetnum: the shell variable QUEUE must be set.\n");
      printf("Program exits.\n");
      exit(0);
    }
  return atoi(c);
}


/* QOPEN */
/* The routine qopen establishes access to the message queue */
/* It returns the queue identifier, mqueue */

qopen(qnumber)
int qnumber;  /* the number for the queue to access */
{
  int msgflg=0001666;
  int mqueue; /* the message queue identifier */
  mqueue= msgget(qnumber,msgflg); /* get the id for this queue */
  if (mqueue ==-1)
    {
      printf("qopen: message queue number %d cannot be opened.\n",
             qnumber);
      printf("Program exits.\n");
      exit(0);
    }
  return mqueue;
}
```

```
/* QGET */

/* The routine qget read a message from the message queue */
/* It returns the process id of the sender */

qget(mqueue,dest,msg)
int mqueue; /* the message queue to read from */
int dest; /* only messages with specified destination will be read */
char msg[80]; /* the message read will be placed here by qget */
{
  int retcode;
  struct qmessage mbuf; /* the queue message buffer */
  retcode=msgrcv(mqueue,&mbuf,sizeof(mbuf),dest,MSG_RWAIT); /* get msg */
  if (retcode == -1)
    {
      printf("qget: messsage was not received\n");
      printf("Program exits.\n");
      exit(0);
    }
  strcpy(msg,mbuf.mtext); /* copy the message text */
  return mbuf.mpid; /* return the process id of the sender */
}

/* QPUT */
/* The routine qput puts a message into the message queue */
qput(mqueue,dest,msg)
int mqueue; /* the message queue to write to */
int dest; /* thes destination of the message */
char msg[80];
{
  int retcode;
  struct qmessage mbuf; /* the queue message buffer */
  mbuf.mpid=getpid(); /* get the process id of this process */
  mbuf.mdest=dest;      /* establish the destination */
  strcpy(mbuf.mtext,msg); /* copy msg text into message buffer */
  retcode=msgsnd(mqueue,&mbuf,sizeof(mbuf),MSG_WWAIT); /* send msg */
  if (retcode == -1)
    {
      printf("qput: message was not send\n");
      printf("Program exits.\n");
      exit(0);
    }
  return;
}
```

```
/* QREMOVE */
/* The routine qremove removes the message queue and the named pipes */
qremove(qnumber)
int qnumber;  /* the number of the queue to remove */
{
  struct msqid_ds buf;
  int retcode, mqueue;
  mqueue=qopen(qnumber);
  retcode=msgctl(mqueue,IPC_RMID,&buf);
  if (retcode==-1)
    {
      printf("qremove: queue was not removed\n");
      printf("Program exits.\n");
      exit(0);
    }

  /* remove the named pipes */
  system("rm -f pipetoibcl");
  system("rm -f pipefromibcl");
  return;
}
```

```
;;;  ATT.LSP
;;;  Returns the attibutes of an entity. By changing the first argument
;;;  to nth we select the specific entity.
(with-open-file (will_stream "att" :direction :output)
                (let* ((entity_sel
                         (nth 1 (get-object 'entity 'subordinates)))
                        (atts (get-object entity_sel 'attributes))
                        (nb-attributes (length atts)))
                  (print entity_sel will_stream)
                  (print nb-attributes will_stream)
                  (dolist (entity atts)
                          (print entity will_stream))))


;;;    ENT.LSP  FILE
;;; file that retrieves the entities that are available in the system
;;; the entities will be placed in the file ent
(with-open-file (will_stream "ent" :direction :output)
                (let* ((entities (get-object 'entity 'subordinates))
                        (nb-entities (length entities)))
                  (print nb-entities will_stream)
                  (dolist (entity entities)
                          (print entity will_stream))))
```

```
;;;         FETCH.LSP FILE
;;; functions to retrive information from frames


;;; fetch returns the list in 'a-list' that has attribute as a key
(defun fetch (attribute a-list)
  (cadadr (assoc attribute (cdr (get a-list 'frame)))))

;;; fetch-attribute range creates a file of name 'attribute'.
;;; The frames after location 'start' in frames' list are searched
;;; for occurences of 'attribute'. The values of 'attribute'
;;; are placed in the file
(defun fetch-attribute-range (attribute entity start)
  (with-open-file (will_stream attribute :direction :output)
                (let ((newlist
                       (nthcdr start (get-object entity 'instances))))
                  (dolist (instance newlist)
                          (print (fetch attribute instance)
                      will_stream)))))
```

```
;;; Top-level.lsp
;;; Function to change the prompt of the IBCL system
;;; After loading this function, the command (system :top-level)
;;; must be invoked to get the new prompt
;;; Developed by Mia Paget
(in-package 'system)
(defun top-level ()
  (let ((+ nil) (++ nil) (+++ nil)
        (- nil)
        (* nil) (** nil) (*** nil)
        (/ nil) (// nil) (/// nil)
        (back-p nil))
    (setq cond::*restart-clusters* nil
          *ihs-top* (ihs-top))
    (setq *lisp-initialized* t)
    (cond:with-simple-restart (cond:abort "Lisp Top Level")
                              (when (probe-file "init.lsp") (load
"init.lsp")))
    (loop
     (setq +++ ++ ++ + + -)
     (when back-p
           (format t "~% Back to Lisp Top Level..."))
     (setq back-p t)
     (reset-stack-limits)
     (when (cond:with-simple-restart
            (cond:abort "Lisp Top Level")
            (format t "~%~acistk>"
                    (if (eq *package* (find-package 'user)) ""
                      (package-name *package*)))
            (setq - (locally (declare (notinline read))
                             (read *standard-input* nil *eof*)))
            (when (eq - *eof*) (bye))
            (let ((values (multiple-value-list
                           (locally (declare (notinline eval))
                                    (eval -)))))
              (setq /// // // / / values *** ** ** * * (car /))
              (fresh-line)
              (dolist (val /)
                      (locally (declare (notinline prinl)) (prinl val))
                      (terpri))
             nil)
            (setq back-p nil))
           (terpri *error-output*)))))
```