ABSTRACT

An Implementation and Evaluation of a Distributed Information System Based on the $OSCA^{TM}$ Architecture Guidelines

by
Babita Masand

The concepts in this thesis discuss and evaluate the need for *systems integration.* As a conceptual architecture, the $OSCA^{TM}$ architecture is investigated. A prototype of the CS Department distributed information system is built following the guidelines and standards of this conceptual architecture. This prototype is implemented in 'C'. RPC (remote procedure call) is used for the communication channel to implement the distributed environment. Conclusions and results that were achieved by implementing this prototype are presented. The main outcome of this thesis is the introduction of the *infrastructure* to the GenSIF framework.

AN IMPLEMENTATION AND EVALUATION OF A DISTRIBUTED INFORMATION SYSTEM BASED ON THE $OSCA^{TM}$ ARCHITECTURE GUIDELINES

by
Babita Masand

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
Department of Computer and Information Science
May 1992

APPROVAL PAGE

An Implementation and Evaluation of a Distributed Information System Based on the $OSCA^{TM}$ Architecture Guidelines

by

Babita Masand

Dr. Wilhelm Rossak
Assistant Professor of Computer and Information Science, NJIT

Dr. Peter Ng
Chairperson for Computer and Information Science, NJIT

BIOGRAPHICAL SKETCH

Author: Babita Masand

Degree: Master of Science in Computer and Information Science

Date: May, 1992

Undergraduate and Graduate Education:

- Master of Science in Computer and Information Science, New Jersey Institute of Technology, Newark, NJ, 1992
- Master of Science in Physics (Electronics), University of Bombay, India, 1987
- Bachelor of Science in Physics, University of Bombay, India, 1985

Major: Computer and Information Science

This thesis is dedicated to my parents.

ACKNOWLEDGEMENT

The author wishes to express her sincere gratitude and thanks to her advisor, Dr. Wilhelm Rossak, for his guidance, friendship, and moral support throughout this work, without which this thesis would not be a success.

Special thanks to Mr. John A. Mills of Bellcore who guided us in our study of OSCA.

The author appreciates and thanks Ms. Felicia Cheng for extended help with the lab facilities.

And finally, a thank you to Ravi, Jatin, and Rajesh for their help.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION / OVERVIEW OF GENSIF

1.1 Need for Systems Integration

The available current methodologies and development strategies are unable to deal with very large distributed systems [2]. These systems are built by different contractors and at different points in time, they are relatively independent projects, but finally they all have to work together in an integrated manner. The factors that necessitate the need for systems integration are:

- More than a single client is involved
- More than a single producer is involved
- More than a single project is involved

Each of the projects in a system are developed independently. These projects together form a larger system. Hence to guarantee the integration optimization and consistent user semantics of the final product, we need a pre-planned and organized integration process. This task includes the integration of these projects into a larger system framework like GenSIF (Generic Systems Integration Framework) [6] and to use integration architectures like Bellcore's $OSCA^{TM}$ [1].

1.2 A Generic Systems Integration Framework (GenSIF)

GenSIF serves as a generalized blue print for the description and handling of large and complex systems [7]. It deals with integration architectures and addresses

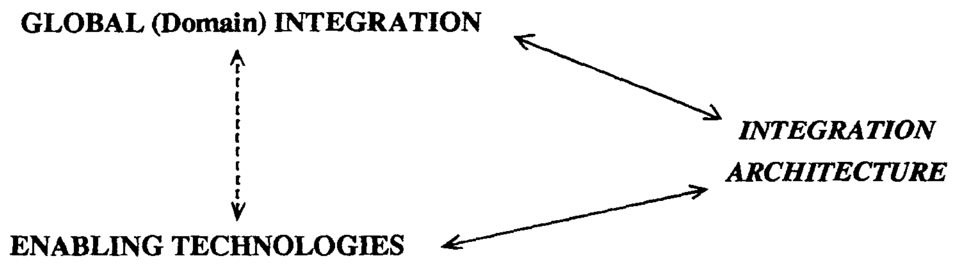


Figure 1.1: Components of the GenSIF framework

standards, guidelines and domain knowledge on a conceptual level. The necessary environmental infrastructure like networking, data storage, operating systems, hardware, etc. are supported on the tool level. From the point of view of the integration architecture, domain knowledge delivers the generalized specification for functionality and data handling capabilities, while the infrastructural level is used as a set of technologies, which enable an optimal implementation of the architecture and the embedded individual applications.

This framework for systems integration [6], includes three components [2]: (Ref. Fig. 1.1)

- *Global domain integration*:

  Specifies the conceptual basis for the integration architecture. One aspect of global integration is to deal with the concepts and semantics of an application domain and with the mapping of these concepts into the installed applications. Domain analysis [8] not only provides the basis for systems integration, but is also the main input to decide the design of the integration architecture.

- *Derivation of an integration architecture*:

  The integration architecture is the core of GenSIF. An integration architecture is a conceptual model that bridges the gap between the results of domain analysis and the tool level. It is also an infrastructure that provides the necessary utilities and components to implement an application system by following the rules of the conceptual model. An integration architecture must fit the needs of the application domain, like a given hardware - architecture must fit the needs of the typical working environment it is serving.

- *Assessment and usage of enabling technologies*:

  Enabling technologies comprise all the tools and products that are required by the infrastructure of an integration architecture to develop and implement the applications which will fill the abstract architecture with functionality and data. It should also provide suggestions for restrictions and standards in this area. Therefore, the choice of enabling technologies is a strategic decision which goes beyond the scope of a single application. This concept coordinates the concepts of domain analysis, global integration and already existing technologies and filters it through the integration architectures. It thus allows a controlled process of reasoning, evaluation and adaptation as is necessary for decisions with long term effects.

1.3 The Conceptual Architecture Model

The conceptual architecture model [2] describes the guidelines and standards of the architecture. It links the model of the application domain with the implementation oriented concepts of development environments and enabling technologies [6]. These guidelines usually include:

- *Standards and guidelines of building blocks*:

  The building block is the elementary concept of integration architectures. Building blocks are the components which provide different functionality in an application domain. A building block acts like a black box, offering services via a predefined interface but hiding implementation details. Each building block emulates the same type of interface and uses the same method of communication and data access as defined in the integration architecture standards. All the building blocks together form the integrated system.

- *A general strategy for system decomposition*:

  One project may deliver one building block or a set of building blocks. This depends on the decomposition strategy of the integration architecture. The architecture only restricts the possible layout of the final product in its appearance on system level. A good example for decomposition rules on a conceptual level would be to separate functionality of the user interface, the data storage units and of additional functional units into different building blocks regardless of project boundaries and development methodologies [1].

- *Communication model*:

  To form a fully functional system, the building blocks must be connected to each other. Standards have to be set for one building block to send and receive data or control to another building block. This communication standard is only of concern at the system level of the application domain but does not interfere with the internal characteristics of the building blocks. Two main strategies are seen on this level: (1) The free communication of standardized messages between building blocks. (2) A centralized approach based on a structure similar to bulletin boards and databases [9].

- *A model for handling of data storage/access*:

  By standardizing the way data is modeled, accessed and stored, an integration architecture can open the set of global data items to all building blocks. In a traditional environment these data items would be guarded and hidden by a given application system. Combining the concepts of decomposition rules, building blocks, standardized communication, data handling can become just another service offered by the building blocks.

A good example for the conceptual part of an integration architecture is $OSCA^{TM}$ [1]. Refer Chapter 2 for further details.

1.4 The Technical Infrastructure

The technical infrastructure [2] of an integration architecture is the basis to finish projects which follow the rules of the conceptual model. The infrastructure provides the necessary standardized services which are the essential ingredients of the architecture. The type of infrastructure used depends to a large extent on the type of architecture the conceptual model describes. The two most important elements of an infrastructure are:

- *A software bus (a channel) for communication*:

  A software bus facilitates the communication between building blocks. It works like a bus or a channel in a hardware architecture. A software bus gives the system engineer a chance to hide a set of distributed hardware/software platforms underneath the interface of the bus. By providing logical addresses, a building block can communicate with another block via the use of the bus without having to know where the other building block is installed. The concept of this software bus was developed in the prototype implemented.

- *Data storage facilities with standardized interface*:

  In case of a distributed database, the data stored has to provide distributed storage facility and an interface to the software bus. Hence storage elements can be added and deleted as necessary. The challenge here is to provide a standardized and generalized interface for these data storage elements and to handle the problems of data conversion between different products, platforms and concepts. Such a solution of a relatively independent data storage building blocks, accessible over the software bus and offering a distributed but still integrated data environment, is described in [4]. The distribution is hidden underneath the software bus. Every building block can send a message to ask for a service without knowing the physical locations. For the user and any building block, the database appears essentially like a centralized system. Such an implementation of independent data storage building blocks has been described in chapter 4 and chapter 5.

The components of the infrastructure should be as general as possible to support a wide variety of conceptual architectures. They should be able to run on different hardware and software platforms, must provide a standardized interface and must be easily updated and adapted while still maintaining upwards compatibility. Most components of such an infrastructure are not readily available. Thus the infrastructure must be developed and maintained as part of a separate project at a meta level above the application projects.

1.5 A Meta Model for System Development

As discussed earlier, we assume that the development environment is diversified and complex. (Sub-)systems are developed in independent projects and are a part of a larger development effort within the application domain [2]. To be able to achieve integrated development in such an environment, a meta level above the level of single projects can be introduced [10].

This meta level is oriented towards long term goals and control of short term projects, in order to assure integration of these projects and their results into the global system framework. Using the components of the integration framework as discussed, a meta level of control for system development can be specified, which provides the system engineer with the basis to coordinate and plan projects in the application domain. For further details see [10].

CHAPTER 2
OVERVIEW OF THE CONCEPTUAL INTEGRATION ARCHITECTURE

2.1 Introduction

As described in chapter 1, a conceptual integration architecture specifies the basic strategy used to design the system from relatively independent components. For the prototype developed here, we used $OSCA^{TM}$[1] as the basis for our variation of such a conceptual guideline.

The $OSCA^{TM}$ Architecture [1], is a system design framework intended to give Bellcore client companies (BCC) the flexibility to combine software products in ways which best satisfy their business needs and to provide access to corporate systems by all authorized users.

The $OSCA^{TM}$ Architecture is designed to allow any heterogeneous software products, when they are designed within the $OSCA^{TM}$ architectural framework, to operate as a system of systems in a loosely coupled, distributed configuration so as to achieve end-to-end automation, corporate data access and rapid development of advanced technologies.

2.2 Concepts of the $OSCA^{TM}$ Architecture.

$OSCA^{TM}$ [1] is an architecture to enable and enhance *interoperability* among software systems. *Interoperability* is the ability of building blocks (i.e., the deliverable components of an OSCA-based system) to communicate with each other and the ability to communicate with any building block irrespective of the internal imple-

[1]OSCA is a trademark of Bellcore - Bell Communications Research

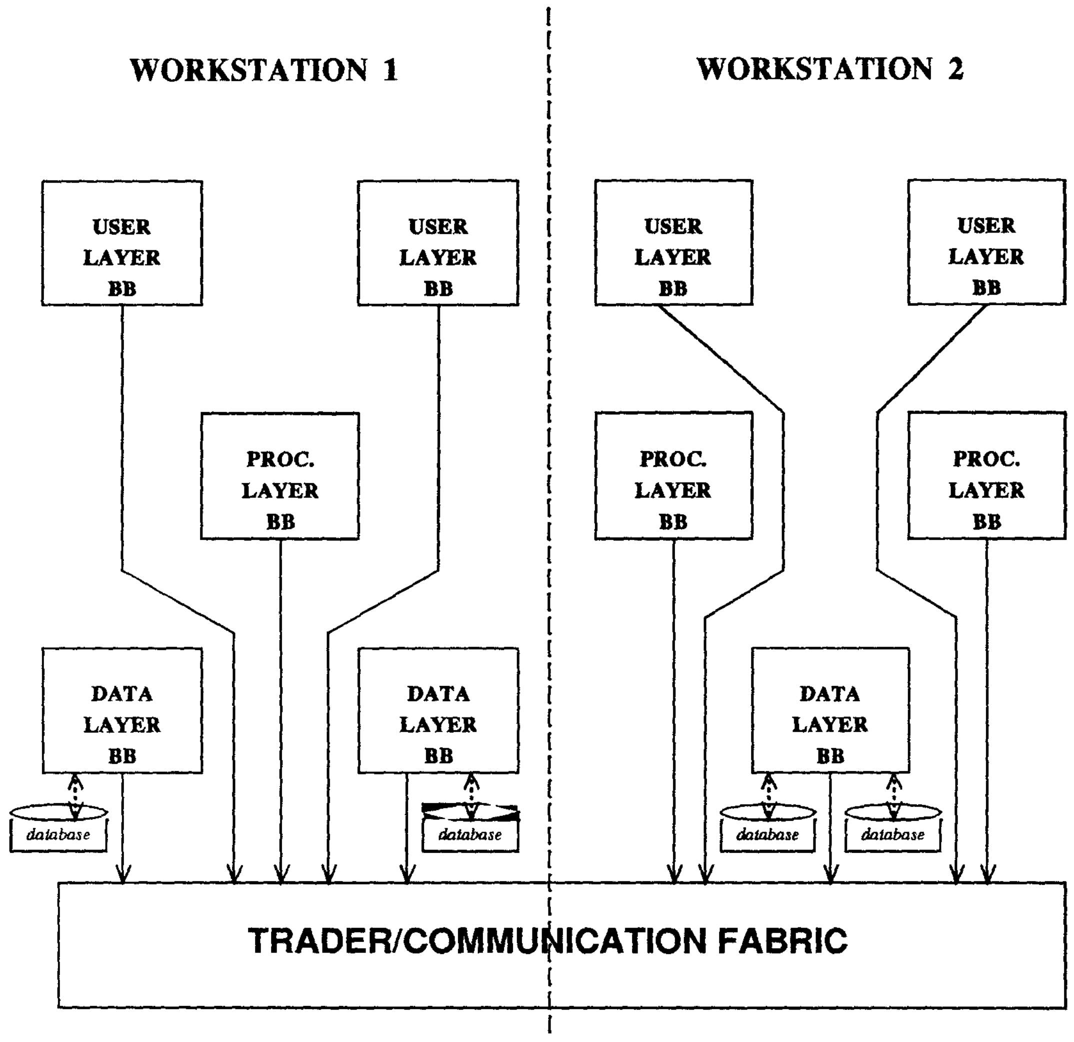


Figure 2.1: General $OSCA^{TM}$ view of a system as derived from[1]

mentations and the environments on which the building blocks reside. The general layout of the $OSCA^{TM}$ architecture is shown in Figure 2.1.

$OSCA^{TM}$ requires that application functionality be separated (grouped) into "layers" or domains: a corporate data layer, a processing layer and a user layer. A layer is the union of all functionality defined as either corporate data functionality, processing functionality or user functionality.

The corporate data layer stewards corporate data and provides functionality to support all create, retrieve, update and delete operations of corporate data in a semantically consistent manner. The corporate data layer also supports redundancy management and generalized query.

The user layer provides functionality to support human users and supports business-specific functions that assume the existence of human users.

The processing layer provides functionality for business processes, otherwise not supported in the other layers, such as complex mathematical algorithms and non-interactive process control.

The software that implements the functionality in these layers is partitioned into "building blocks", and these building blocks must adhere to specific interoperability principles, such as no simultaneous releases among building blocks, resource independence among building blocks, no accessibility assumptions among building blocks, logical building block addressing, and the presence of a secure environment. In addition, interfaces between building blocks must meet certain criteria, such as the use of industry and international standards, restricted set of syntax encodings, isolation from building block internals etc., and having met them are termed *contracts.* (*Contract* - is a deal/an interface between any two building blocks. It is implemented by the building block.)

Based on the $OSCA^{TM}$ architecture, our conceptual architecture can be described on several levels of abstraction as given in the following sections.

2.3 A Conceptual Architecture of Building Blocks

At the highest level of abstraction, the application is made from software on one or more of three logical layers. (Ref. Fig. 2.2) As shown in this figure, level 1 is the user layer, level 2 is the processing layer and level 3 is the data layer. A layer is the union of all functionality defined as either data function, processing function or user function. Each layer is made up of one or more well defined deliverable functional units called building blocks. As seen in the Figure 2.2, any building

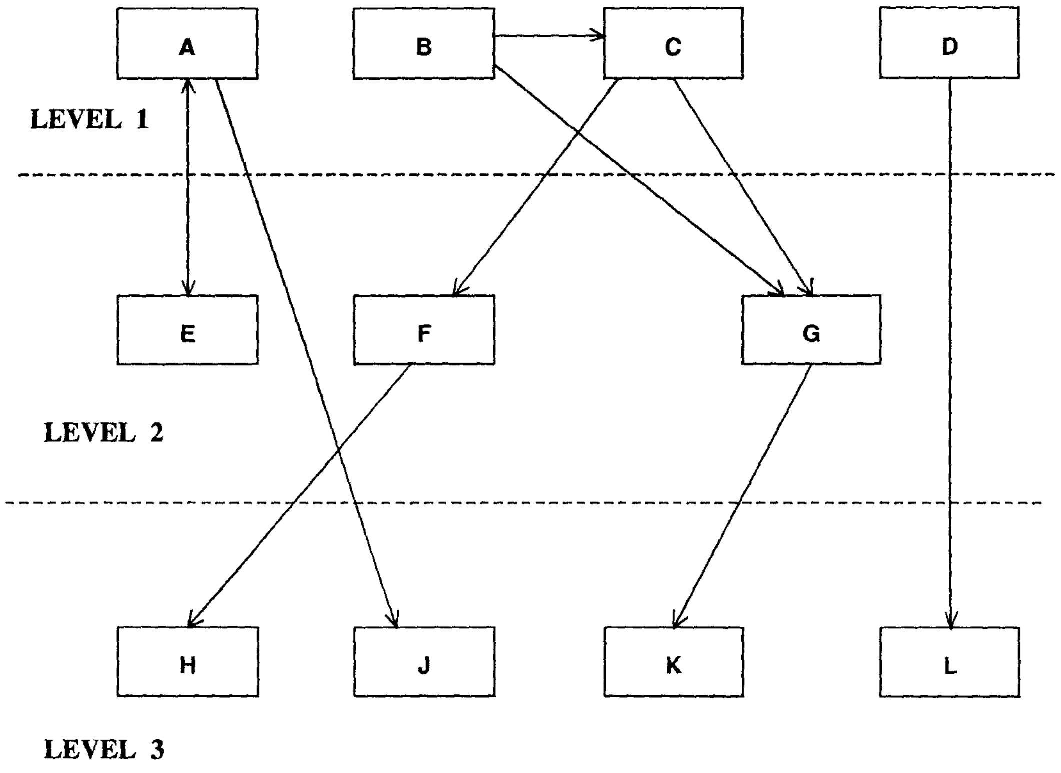


Figure 2.2: A Conceptual Architecture of Building Blocks

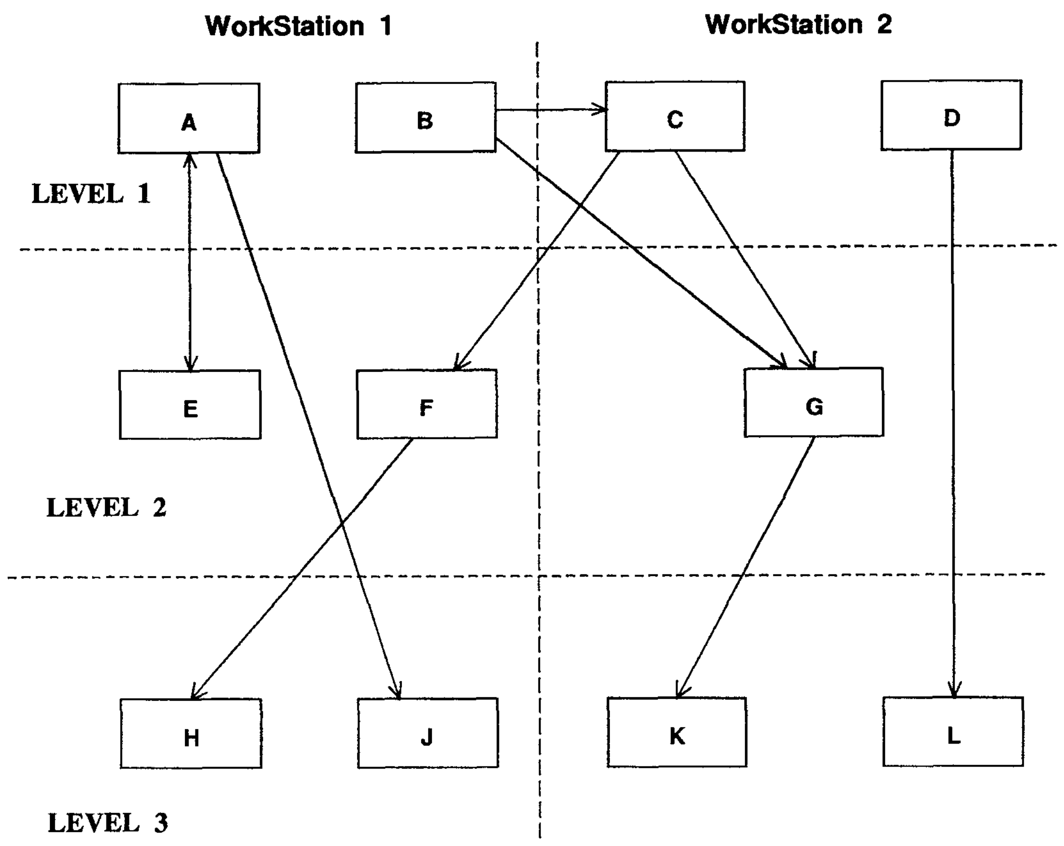


Figure 2.3: The Conceptual Architecture: Workstation View

block can communicate with any other building block that provides a function that it requires. Ex., a user layer building block **C** communicates with a processing layer building block **F**, that fetches and manipulates data from a data layer building block **H** for the end user. User layer building blocks can enquire directly to the data layer building blocks. Building blocks of any layer can communicate with building blocks in the same layer.

2.4 The Conceptual Architecture in a Distributed Environment

A distributed view of the building blocks on two workstations is shown in Figure 2.3. These building blocks exist logically on the same levels as shown in the previous figure (Figure 2.2). However, they are placed physically on two different workstations. As shown in the Figure 2.3, when a building block **B** on workstation 1 requires to communicate with the building block **C** on workstation 2, they can not do so directly as they no longer exists on the same physical environment. Hence the need for a trader/communication fabric as shown in the Figure 2.4.

2.5 The Conceptual Architecture with Explicit Communication Channel

As seen in figure 2.4, the building block **B** will now communicate with building block **C** through the trader, i.e., a software communication fabric. (In terms of $OSCA^{TM}$, this communication between the building blocks should be defined as a *contract*. A *contract* is invoked by providing well defined input which will result in a well defined set of actions. Actions include returning outputs to the invoker of the contract and invoking contracts of other building blocks.) In general, every communication between building blocks residing on different platforms is now done indirectly using the trader.

2.6 The Conceptual Architecture with Fully Transparent Communication

The next step is to utilize the trader to handle all communications in the system, irrespective of the location of the building blocks which want to communicate (Ref. Figure 2.5). (This is done in order to achieve the *interoperability* goals of the $OSCA^{TM}$ architecture.)

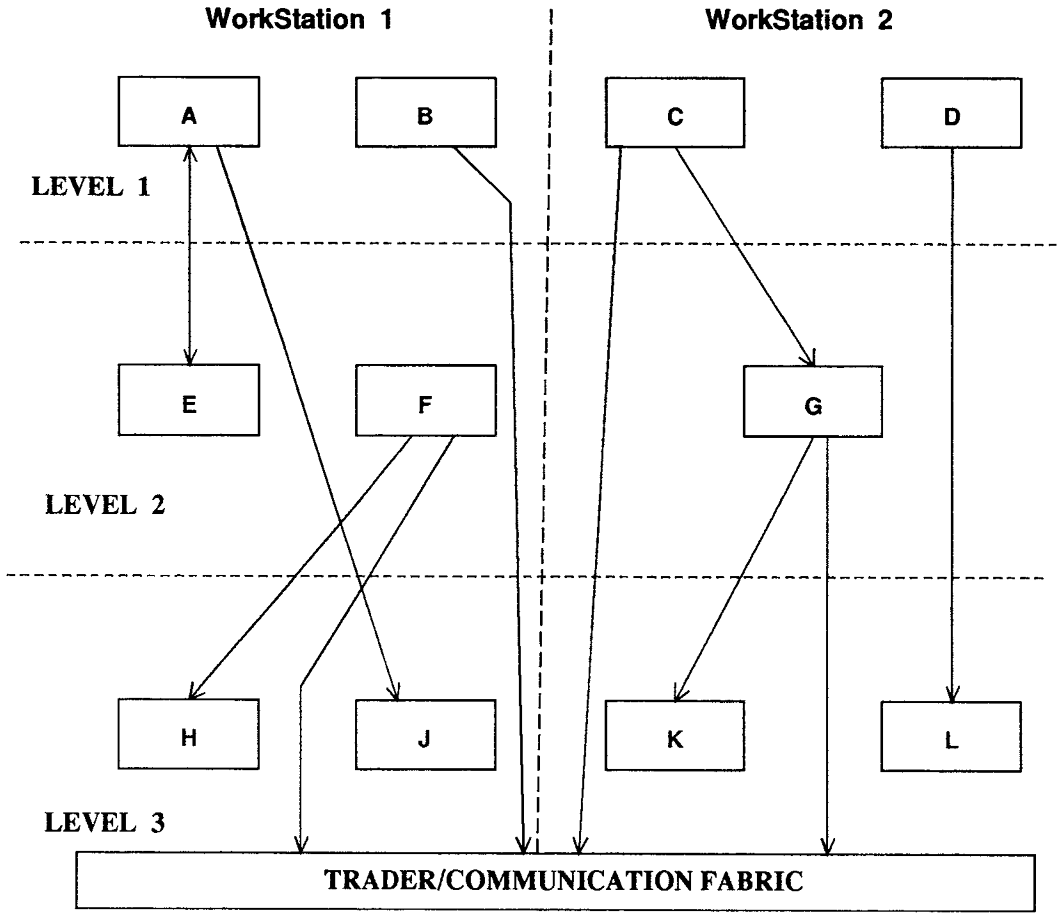


Figure 2.4: The Conceptual Architecture with Explicit Communication Channel

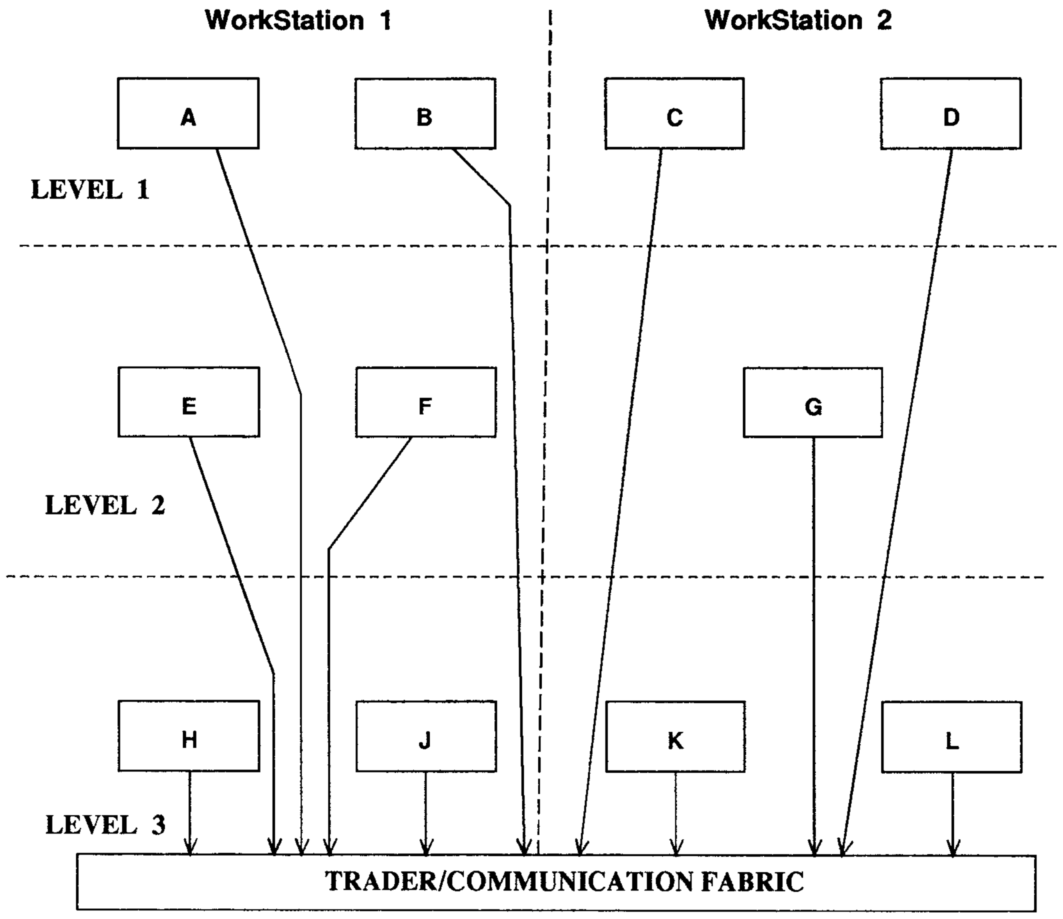


Figure 2.5: The Conceptual Architecture with Fully Transparent Communication

For example, the building blocks **C** and **G** placed on workstation 2, no longer directly communicate with each other, instead they now communicate through the common trader. This allows the trader to guarantee transparent addressing or even better to provide a set of services that is matched internally to messages and addresses.

2.7 The Trader

The **Trader** provides software services and infrastructure for inter-building block communication, building block availability and performance management. In a distributed environment such as promoted in this conceptual architecture, no single building block will provide all of the services required by a user. In such an environment, multiple building blocks operate together in a cooperative effort and communicate their results to each other. The Trader is the "glue" that ties together building blocks and allows this cooperative effort in providing the services to the users. The trader knows the physical addresses of the building blocks. When a particular service of a building block is requested by another building block, the trader maps the requested service to the address of the building block that provides it.

CHAPTER 3
THE CS DEPARTMENT DIS APPLICATION

3.1 Overview of the System

The prototype is built as an example for systems integration, based on the conceptual architecture described in chapter 2. This prototype implements a CS Department Distributed Information System (DIS). In this application, the Computer Science (CS) Department is organized by the faculty, courses, students and registration. Each faculty member teaches one or more of the courses offered by the CS Department. Each student is registered for one or more courses offered by the CS Department. The number of credits offered by a particular course does not vary, i.e., all students who pass the same course receive the same amount of credit. Thus this database keeps track of the following entities:

- faculty - courses taught by the faculty along with the section numbers
- courses - course information, section number, semester and year offered, room number for a particular section of the course and the time when it is offered
- students - their major, college degree, courses registered for and the grades obtained for the courses taken

This application is written in 'C' using flat files for data storage. Since it is a DIS, the executables and *databases* can exist on one or more workstations. This prototype provides the following facilities:

- A simple menu driven user interface providing the following options: (Ref. Appendix Section 7.1)

1. ADD/DELETE faculty records:

   Adds new records or deletes user specified records to and from the faculty database.

2. ADD/DELETE course records

   Adds new records or deletes user specified records to and from the course database.

3. ADD/DELETE student records

   Adds new records or deletes user specified records to and from the student database.

4. ADD/DELETE register records

   Adds new records or deletes user specified records to and from the register database.

- The following output screens:

  1. Listing of courses registered by students
  2. Listing of students registered for the courses offered by the CS Department
  3. Listing of faculty members of the CS Department
  4. Listing of courses taught by a faculty
  5. Listing of courses offered by the Department
  6. Viewing specific records based on specific fields:

     (a) For faculty: based on first name, last name, office location, phone number and social security number

     (b) For course: based on course name and course number

(c) For student: based on first name, last name, phone number and social security number

(d) For registered courses: based on grade in a particular course, course number and student social security number

3.2 System Description

This software has two main facilities. One is to update the CS Department DIS and the other is to handle queries. The functions ADD and DELETE provide the required updation of the DIS. The software provides answers to user interactive queries (ex. statistical, listings, etc.).

3.2.1 The ER Diagram

The ER diagram is as shown in Figure 3.1.

The Student entity has attributes stud_ssn, stud_firstName, stud_middleInitial, stud_lastName, stud_address, stud_city, stud_state, stud_zip, stud_phone, stud_major, stud_college and stud_gpa. The *stud_ssn* is the primary key of this entity.

The course entity has attributes course_number, course_section, course_semester, course_year, course_name, course_inst, course_room, course_bldg, course_day, course_time and course_credit. The *course_number, course_section, course_semester, course_year* is the primary key of this relation. One or more sections of a course are offered in a semester in a particular year.

The faculty entity has attributes ssn, firstName, middleInitial, lastName, phone, location. The *ssn* is the primary key of this relation.

The registered relationship exists between the entities student and course. This is a many to many relationship, since one student can register for many courses and one course can be taken by many students. In addition, this relationship stores the grade attribute which describes the grade earned by a student in a particular course.

The teaches relationship exists between the faculty and course entities. This is a one to many relationship from faculty to the course entity, since one faculty member can teach many courses, but a particular course can be taught by only a single faculty member.

A List of Entities / Relations and Attributes

1. **STUDENT:**

   (a) *stud_ssn:* Is the student's social security number

   (b) *stud_firstName:* Is the student's first name

   (c) *stud_middleInitial:* Is the student's middle initial

   (d) *stud_lastName:* Is the student's last name

   (e) *stud_address:* Is the student's residential street address

   (f) *stud_city:* Is the city in the student's address

   (g) *stud_state:* Is the state in the student's address

   (h) *stud_zip:* Is the zip code in the student's address

   (i) *stud_phone:* Is the student's home phone number

   (j) *stud_major:* Is the student's major

   (k) *stud_college:* Is the student's college

   (l) *stud_gpa:* Is the student's gpa

2. **COURSE:**

   (a) *course_number:* Is the number of a course

   (b) *course_section:* Is the section of a course

   (c) *course_semester:* Is the semester the course is offered

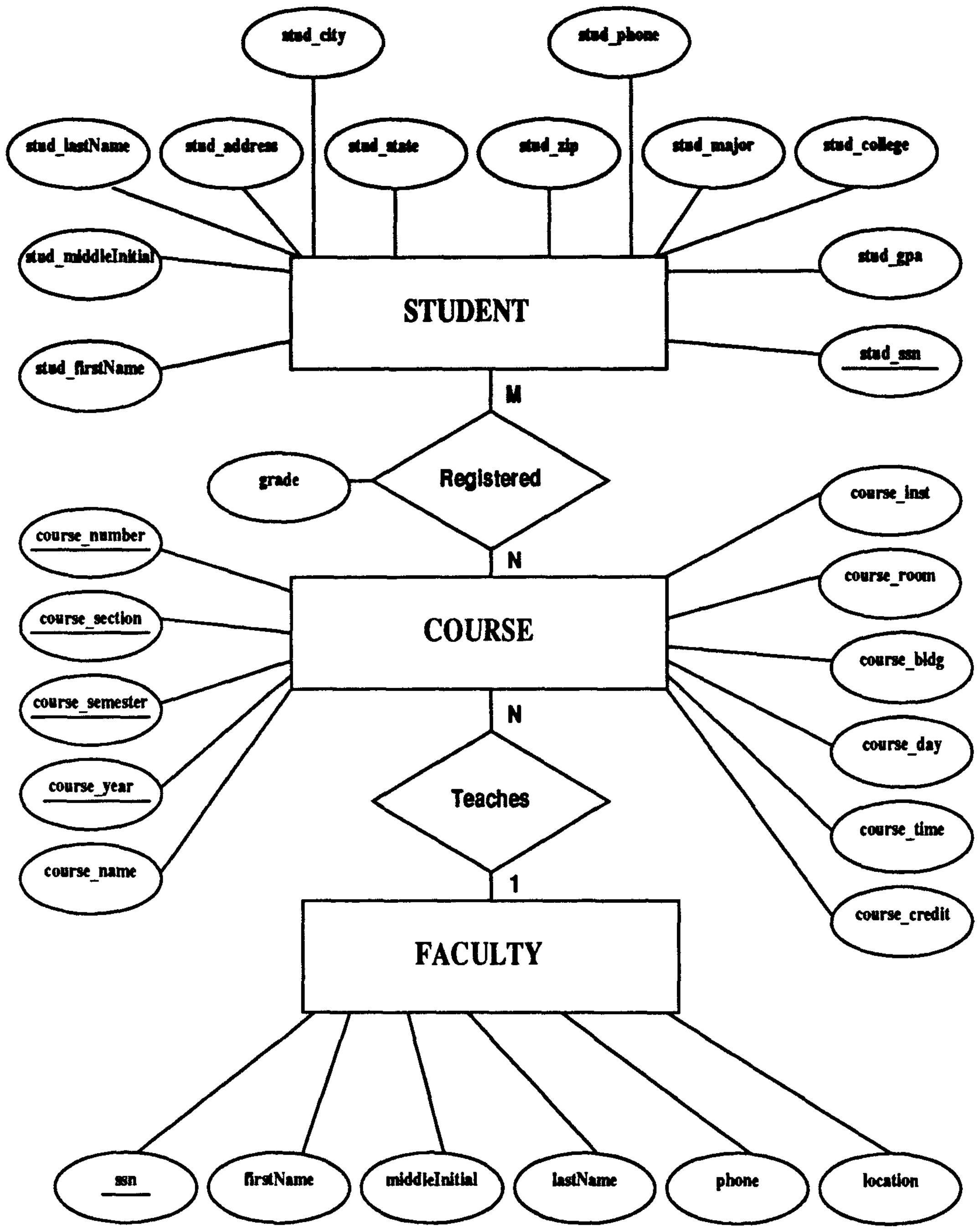


Figure 3.1: ER Diagram for CS Department DIS

(d) *course_year:* Is the year the course is offered

(e) *course_name:* Is the name of the course

(f) *course_inst:* Is the social security number of the instructor teaching the course

(g) *course_room:* Is the room number in which the course is taught

(h) *course_bldg:* Is the building in which the course is taught

(i) *course_day:* Is the day when the course is taught

(j) *course_time:* Is the time when the course is taught

(k) *course_credit:* Is the number of credits for the course

3. **FACULTY:**

    (a) *ssn:* Is the faculty's social security number

    (b) *firstName:* Is the faculty's first name

    (c) *middleInitial:* Is the faculty's middle initial

    (d) *lastName:* Is the faculty's last name

    (e) *phone:* Is the faculty's phone number

    (f) *location:* Is the faculty's location

4. **Registered Relationship:**

    It provides information about the courses registered by a student and the corresponding grade.

    (a) *grade:* Is the grade earned by a student in a particular course

5. **Teaches Relationship:**

    It provides information about the courses taught by a faculty member.

3.2.2 User Characteristics, Assumptions and Dependencies

This application assumes two main end users, a professor i.e., a faculty member and the department secretary. A simple, user friendly environment is provided to help the users use the product efficiently. Different views are provided for the two end users as follows:
An instructor would be interested in the following views of the database:
(1) List of all students in a particular class conducted by him/her.
(2) List of all the PhD students.
(3) List of all students in his/her class based on a specific grade.

The Department secretary would be interested in the following view of the database:
(1) Total credits taught by an instructor.
(2) Total credits taken by a student.
(3) List of all the courses taken by a student.

As of now, a common user interface is designed for both the users. Hence, a faculty member or a department secretary has access to all the above mentioned functionalities.

3.2.3 Product Functions

1. Add a record

   A new record can be added to the faculty, course, student and register databases.

2. Delete a record

   A specific record can be deleted from the faculty, course, student and register databases.

3. List all the records in a specific database

   List of all the courses, faculty, students and courses registered by students.

4. Answer user interactive queries i.e., viewing specific records based on specific fields

   (a) For faculty: based on first name, last name, office location, phone number and social security number

   (b) For course: based on course name and course number

   (c) For student: based on first name, last name, phone number and social security number

   (d) For registered courses: based on grade in a particular course, course number and student social security number

3.2.4 Functional Requirements in Detail

Functional Requirement I

Introduction: This function involves the updation of the CS Department DIS.

$$\text{updation} \rightarrow \text{ADD, DELETE}$$

Inputs:

1. The data to be processed - (source - user)

   To add a new record to the faculty database, the user provides the social security number, first name, middle initial, last name, phone number and office location of the new faculty member.

   To delete a specific faculty member from the faculty database, the user provides the faculty member's social security number.

   To add a new student to the student database, the user provides the social security number, first name, middle initial, last name, phone number, address, major, college and gpa of the student.

To delete a specific student from the student database, the user provides the student's social security number.

To add a new course to the course database, the user provides course number, course section, course semester, course year, course name, course instructor's social security number, course room number, course bldg, course day, course time and course credit.

To delete a specific course from the course database, the user provides the course number.

To register a student to a course, the user provides student's social security number, course number, course section, course semester and year.

To delete a student from the register database, the user provides the student's social security number.

2. The operation to be performed on the input data (the operation is selected by the user from the provided menu)

Processing:

1. **Add:** This process adds the data provided by the user to the CS Department DIS. If the input data is not valid i.e., if the data already exists, the software generates an error message:

   "DUPLICATE RECORD"

2. **Delete:** This process deletes a particular record in the CS Department DIS. The inputs required are social security number or course number corresponding to the record to be deleted. If the input is invalid, the software generates an error message:

   "RECORD DOES NOT EXIST"

3. **Outputs:** A suitable feedback is generated informing the user that the operation is successful.

Functional Requirement II

Introduction: This function "Query" provides answers to user queries.
Input: The query (selected by the user from the provided menu).
Processing:

1. **Statistical Query:** This process provides statistics requested by the user.

   - total number of credits taught by a particular faculty
   - total number of credits taken by a student

2. **Listings:** This process provides the listings of data stored in the DIS.

   - List of all the faculty members in the Department
   - List of all the courses offered by the Department
   - List of all the students registered for the above courses
   - List of all the students in the Department

3. **User Interactive Query:** This process provides answers to any predefined query selected by the user. The system prompts the user to enter information required to process a particular query.

   - To get the listing of courses taught by a particular faculty, the system prompts the user to enter the faculty's last name

- To get the listing of a particular faculty member by the first name, last name, office location, phone number or social security number the system prompts the user to enter first name, last name, office location, phone number or social security number accordingly
- To get the listing for a particular course by the course name or course number, the system prompts the user to enter the course name or course number accordingly
- To get the listing of a particular student by first name, last name, phone number or social security number the system prompts the user to enter the first name, last name, phone number or social security number accordingly
- To get the listing of registered courses by grade in a particular course, course number or student social security number, the system prompts the user to enter the grade in the particular course, course number or student social security number accordingly

Output: The appropriate answer to the requested query is provided.

3.2.5 Level of Maintainability

- The application is written in a modular form, it can be adapted to changing user needs, i.e., anything can be easily added, modified or deleted from the system.
- Since the application is a DIS, all the functionality blocks can be physically placed on any (of the two) workstations. They can be easily moved from one workstation to another. All these movements will be transparent to the end users.

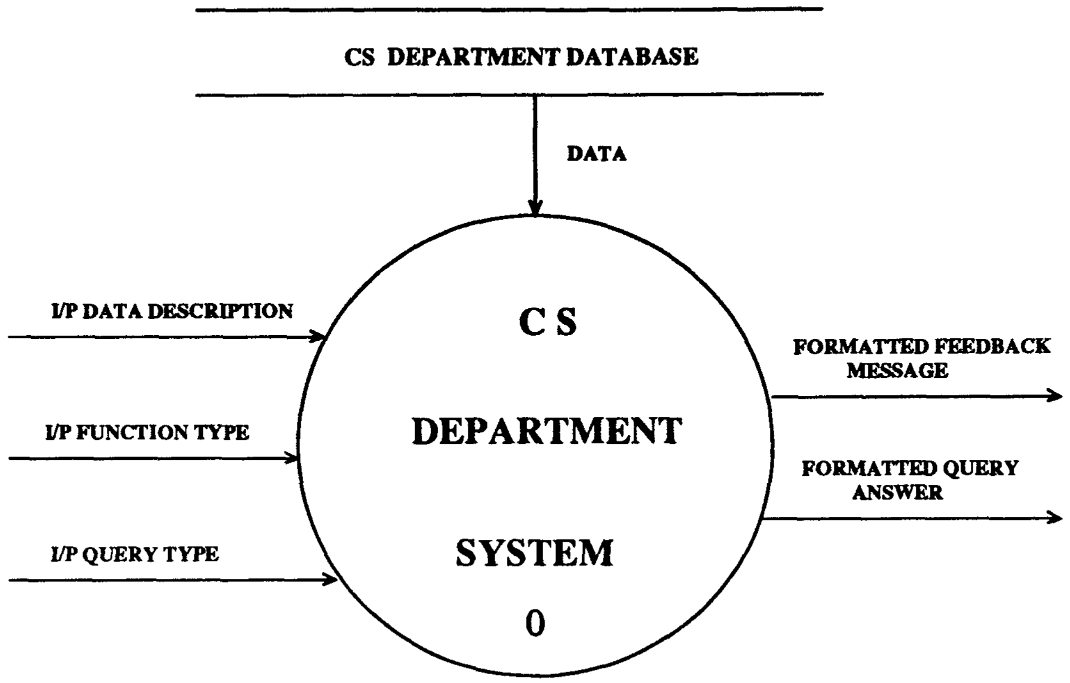


Figure 3.2: Level 0: CS Department DIS

3.2.6 Bubble Charts

– Level 0

The level 0 bubble chart is as shown in Figure 3.2. It consists of the following inputs:

- I/P Data Description: This is the input data provided by the user to add, delete or view records in the faculty, course, student and register databases (as described earlier)
- I/P Function Type: This is selected by the user from the menu to add or delete a particular record in the faculty, course, student and register databases (as described earlier)
- I/P Query Type: This is selected by the user from the available queries listed in the menu (as described earlier)

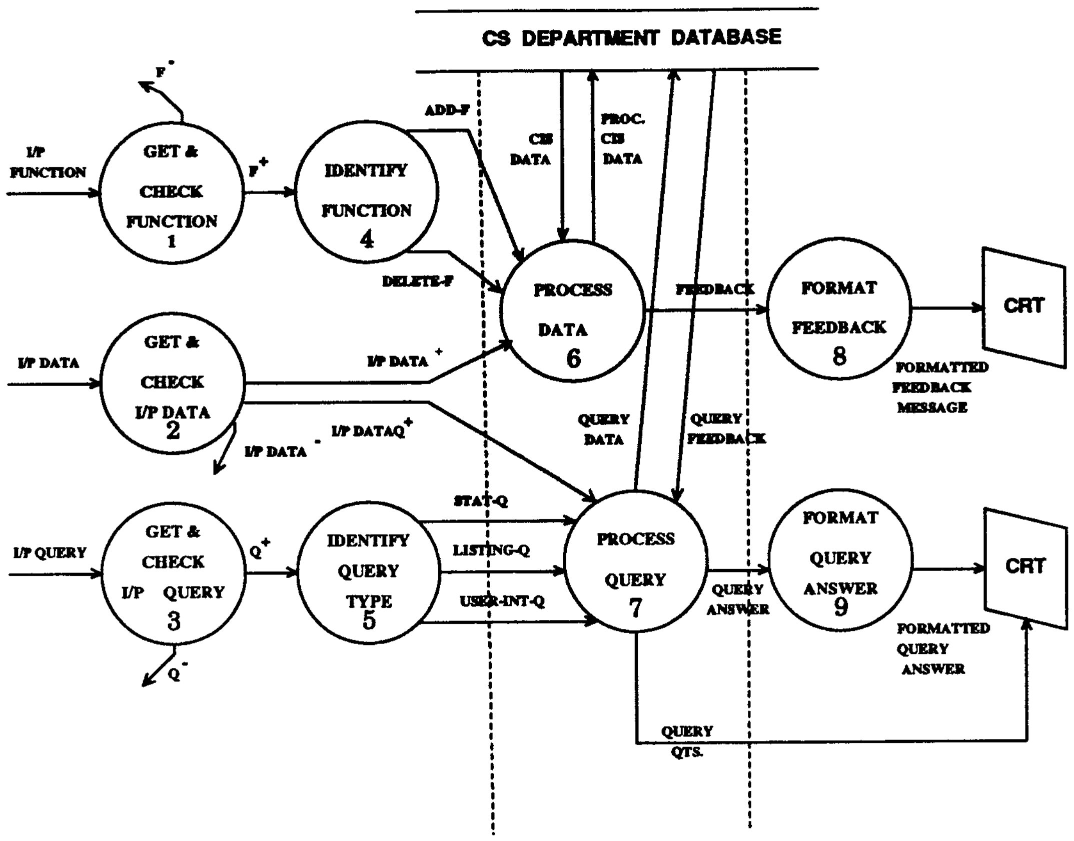


Figure 3.3: Level 1: CS Department DIS

The level 0 bubble chart consists of the following outputs:

- Formatted Feedback Message: This displays a message for the user on the CRT depending on the results of the add or delete operations on the databases
- Formatted Query Answer: Appropriate answer is displayed for the query selected by the user

– Level 1

The level 1 bubble chart is as shown in Figure 3.3. The data flow for this bubble chart is as follows:

The user selects add or delete function for adding or deleting records, in the faculty, course, student or register databases. This is shown by the input *I/P FUNCTION* in the bubble chart. The next bubble identifies the function selected as ADD-F or DELETE-F. ADD-F implies add faculty or add course or add student or add register. Similarly DELETE-F implies the delete faculty, delete course, delete student or delete register functions. The user input shown in the bubble chart as *I/P DATA*, is checked for validity and the verified input data goes to the PROCESS DATA bubble, along with the selected functionality. In case of addition of a new record, the input is checked with the already existing data in the database to avoid duplication. If it does not exist already, then it is added to the database and a message is displayed on the CRT, stating that the new record was added. In case, the record already exists, a message is displayed on the CRT stating that it is a duplicate record. In case of deletion of a specific record, the input is checked for a match with the already existing records in the database. If a match is found, then this record is displayed on the CRT. On the user's approval, the record is deleted and a message is displayed on the CRT, stating that the required record was deleted.

The user selects an *I/P QUERY* from the menu. This query is then checked and identified as STAT-Q (Statistical Query), LISTING-Q (listing query), USER-INT-Q (user interactive query). The selected query along with the input data is then processed. The required query data is obtained from the database and a formatted query answer is displayed on the CRT. In case of user interactive query, a proper query question is posed, after the user enters the data required to process the query, the required query answer is presented to the user.

Abbreviations:

1. F^+: function(Add,Delete)

2. F^-: Any function other than Add, Delete

3. PROC. CIS DATA: Process CS Data

4. Q^+: Query for statistics, listing or interactive
5. Q^-: Invalid query
6. ADD-F: function to Add data
7. DELETE-F: function to Delete data
8. STAT-Q: Statistical query
9. LISTING-Q: Listing query
10. USER-INT-Q: User interactive query
11. I/P DATAQ^+: User selected question

Bubble Chart Description: Level 1

1. GET & CHECK FUNCTION

Input: I/P FUNCTION

Output: F^+

This process gets the function that the user selects from the menu and checks if it is a valid function (F^+), i.e., one of ADD or DELETE. If the function is not valid (F^-), then an error message is generated.

2. GET & CHECK I/P DATA

Input: I/P DATA

Outputs: I/P $DATA^+$, I/P $DATAQ^+$

This process gets the data from the user and checks if the data is valid ($I/PDATA^+$). If the data is not valid ($I/PDATA^-$), then an error message is generated.

3. GET & CHECK I/P QUERY

Input: I/P QUERY

Output: Q^+

This process gets the input query from the user and checks if it is a valid query (Q^+). The input query can be a query for statistics, listings or user interactive query. If the query is not valid (Q^-), then an error message is generated.

4. IDENTIFY FUNCTION

Input: F^+ : function(Add,Delete)

Output: ADD-F, DELETE-F

This process identifies the input function (F^+) type as Add or Delete.

5. IDENTIFY QUERY TYPE

Input: Q^+ : Query

Outputs: STAT-Q, LISTING-Q, USER-INT-Q

This process identifies the query (Q^+) type as STAT-Q, LISTING-Q or USER-INT-Q.

6. PROCESS DATA

Inputs: ADD-F, I/P $DATA^+$, DELETE-F

Output: FEEDBACK : Message indicating that the required updation was successful.

Pseudocode

```
IF input = ADD-F THEN
   DO Add Data
IF input = DELETE-F THEN
   DO Delete Data
```

7. PROCESS QUERY

Inputs: STAT-Q, LISTING-Q, USER-INT-Q

Outputs: QUERY ANSWER : Message - query answer (statistics answer, listings, user interactive answer)

QUERY QTS : Message - query questions (only for user interactive queries)

Pseudocode

```
IF input = STAT-Q THEN
   DO Statistical Query
IF input = LISTING-Q THEN
   DO Listing Query
IF input = USER-INT-Q THEN DO BEGIN
   DO Process User interactive query
   DISPLAY query qts
END
```

8. FORMAT FEEDBACK

Input: FEEDBACK

Output: FORMATTED FEEDBACK MESSAGE

This process formats the feedback message generated by "PROCESS DATA", informing the user that the operation has been performed.

9. FORMAT QUERY ANSWER

Inputs: QUERY ANSWER

Output: FORMATTED QUERY ANSWER

Pseudocode

```
IF QUERY ANSWER = ANSWER TO STATISTICS THEN
    DO Format Answer to Statistics
IF QUERY ANSWER = ANSWER TO LISTING QUERY THEN
    DO Format Answer to Listing Query
IF QUERY ANSWER = ANSWER TO USER INTERACTIVE QUERY THEN
    DO Format Answer to User Interactive Query
```

– Level 2 for PROCESS DATA (No. 6)

The level 2 bubble chart for PROCESS DATA is as shown in Figure 3.4.

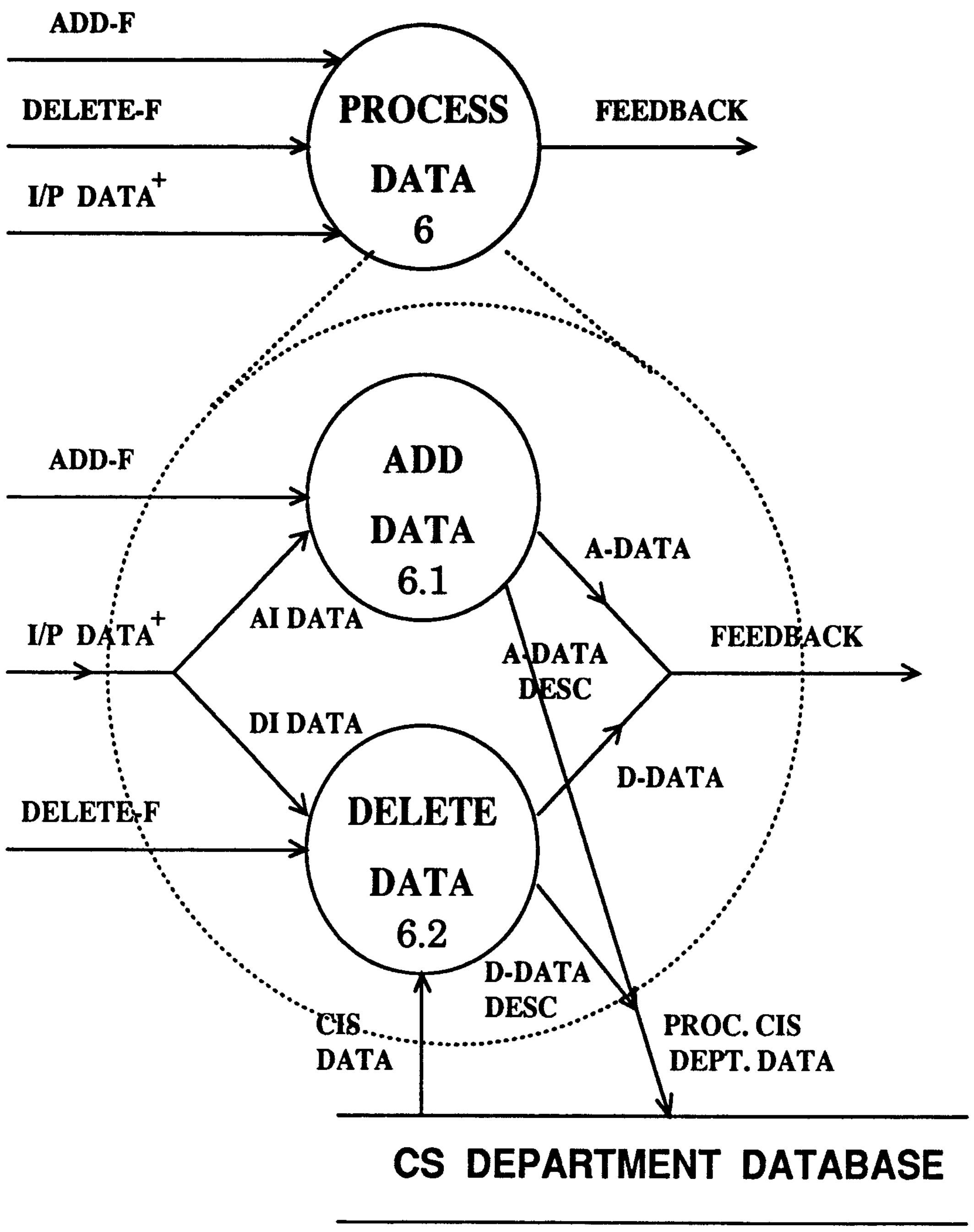


Figure 3.4: Level 2: PROCESS DATA

If the user selects the ADD-F function, i.e., add faculty, or add course or add student or add register, then the I/P DATA i.e., the AI DATA along with the selected functionality is passed to the ADD DATA bubbles for faculty or course or student or register. The new record is then added to the appropriate database, and a message is displayed stating that the record is added. If the user selects the DELETE-F function, i.e., delete faculty, or delete course or delete student or delete register, then the I/P DATA i.e., the DI DATA along with the selected functionality is passed to the DELETE DATA bubbles for faculty or course or student or register. The specified record is first displayed on the screen and on the user's approval is deleted from the appropriate database, and a message is displayed indicating that the record is deleted.

Abbreviations for PROCESS DATA:

1. A-DATA DESC: Added data description
2. A-DATA: Added data feedback
3. AI DATA: Add input data
4. D-DATA DESC: Deleted data description
5. D-DATA: Deleted data feedback
6. DI DATA: Delete input data

Bubble chart description: PROCESS DATA

6.1. ADD DATA

Inputs: ADD-F, AI DATA

Output: A-DATA: Message indicating that the required addition was successful.

Pseudocode for ADD DATA

```
DO Add Data
```

6.2. DELETE DATA

Inputs: DELETE-F, DI DATA

Output: D-DATA: Message indicating that the required deletion was successful.

Pseudocode for DELETE DATA

```
DO Delete Data
```

– Level 2 for PROCESS QUERY(No. 7)

The level 2 bubble chart for PROCESS QUERY is as shown in Figure 3.5.

If the user input is STAT-Q, then the PROCESS STATISTICS bubble processes the statistical query and displays the required statistical answers. If the user input is LISTING-Q, then the PROCESS LISTING QUERY bubble processes the listing query and displays the required listing answers. If the user input is USER-INT-Q, then a proper query question is posed to the user. The user enters the input data as described earlier. Then the PROCESS USER INTERACTIVE QUERY processes these and displays the user interactive answer to the query.

Abbreviations for PROCESS QUERY:

1. A-LIST: Answer to list queries
2. A-U-INT: Answer to user interactive questions
3. LIST-Q: Selected list
4. S-DESC.: Statistical description
5. U-INT. QTS: User interactive questions

Bubble chart description: PROCESS QUERY

7.1. PROCESS STATISTICS

Input: STAT-Q

Output: STATISTICS ANSWER

Pseudocode for PROCESS STATISTICS

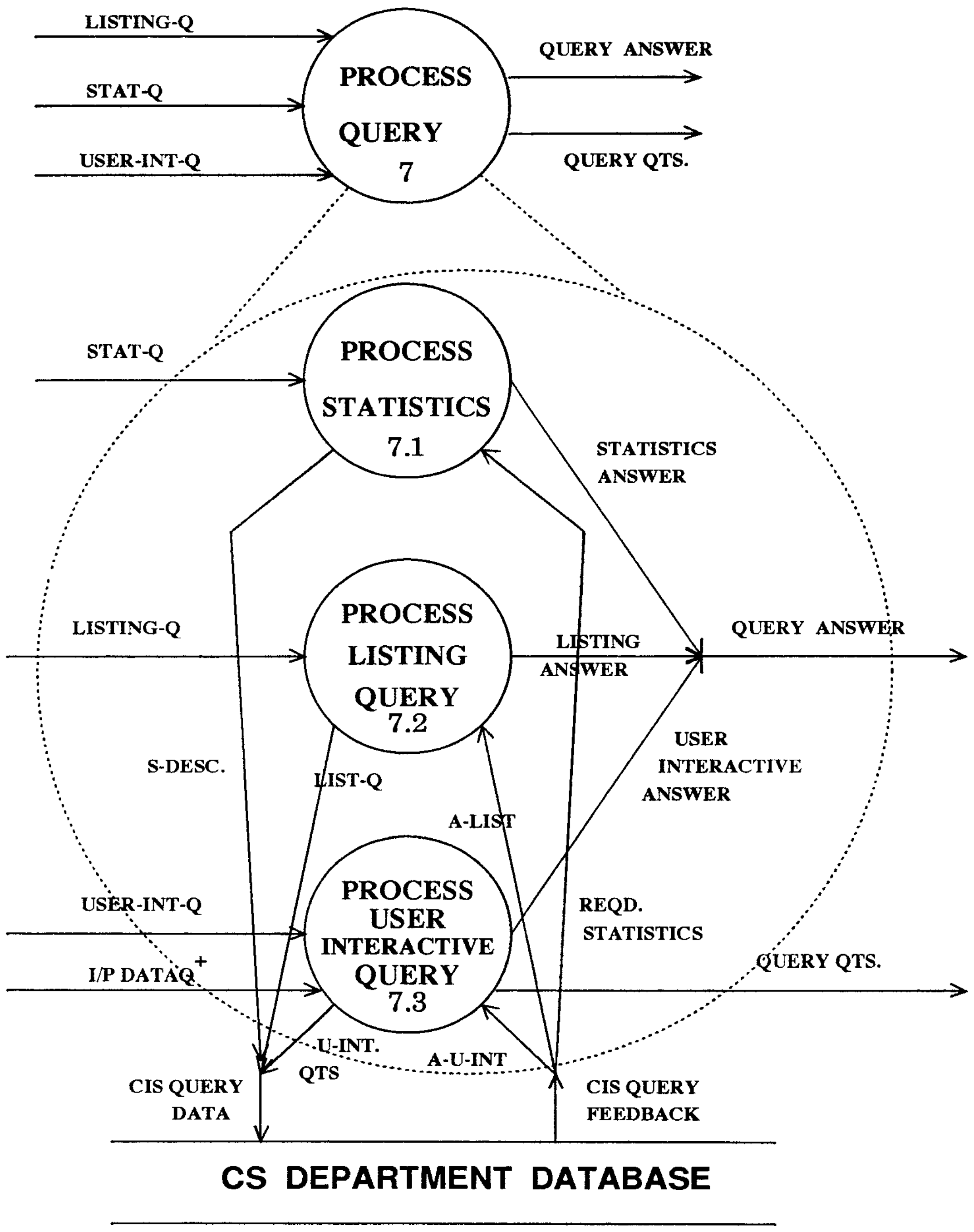


Figure 3.5: Level 2: PROCESS QUERY

```
DO Process Statistics
```

7.2. PROCESS LISTING QUERY

Input: LISTING-Q

Output: LISTING ANSWER

Pseudocode for PROCESS LISTING QUERY

```
DO Process Listing Query
```

7.3. PROCESS USER INTERACTIVE QUERY

Inputs: USER INT-Q, I/P DATAQ^+

Outputs: QUERY QTS, USER INTERACTIVE ANSWER

Pseudocode for PROCESS USER INTERACTIVE QUERY

```
DO Process User Interactive Query
PRINT QUERY QTS
GET User Input
PRINT User Interactive Answer
```

– **Level 2 for FORMAT QUERY ANSWER(No. 9)**

The level 2 bubble chart for FORMAT QUERY ANSWER is as shown in Figure 3.6.

The query answer is the data retrieved from the databases for the required query, selected by the user from the menu. If it is a statistical query, this forms ANSWER TO STATISTICS which is fed to the FORMAT ANSWER TO STATISTICS bubble which formats the data and presents it to the user as FORMATTED ANSWER TO STATISTICS. If it is a listing query, this forms ANSWER TO LISTING QUERY which is fed to the FORMAT ANSWER TO LISTING QUERY bubble which formats the data and presents it to the user as FORMATTED ANSWER TO LISTING QUERY. If it is a user interactive query, this forms ANSWER TO USER-INT QUERY which is fed to the FORMAT ANSWER TO USER-INT QUERY bubble which formats the data and presents it to the user as FORMATTED ANSWER TO USER-INT QUERY.

Bubble chart description: FORMAT QUERY ANSWER

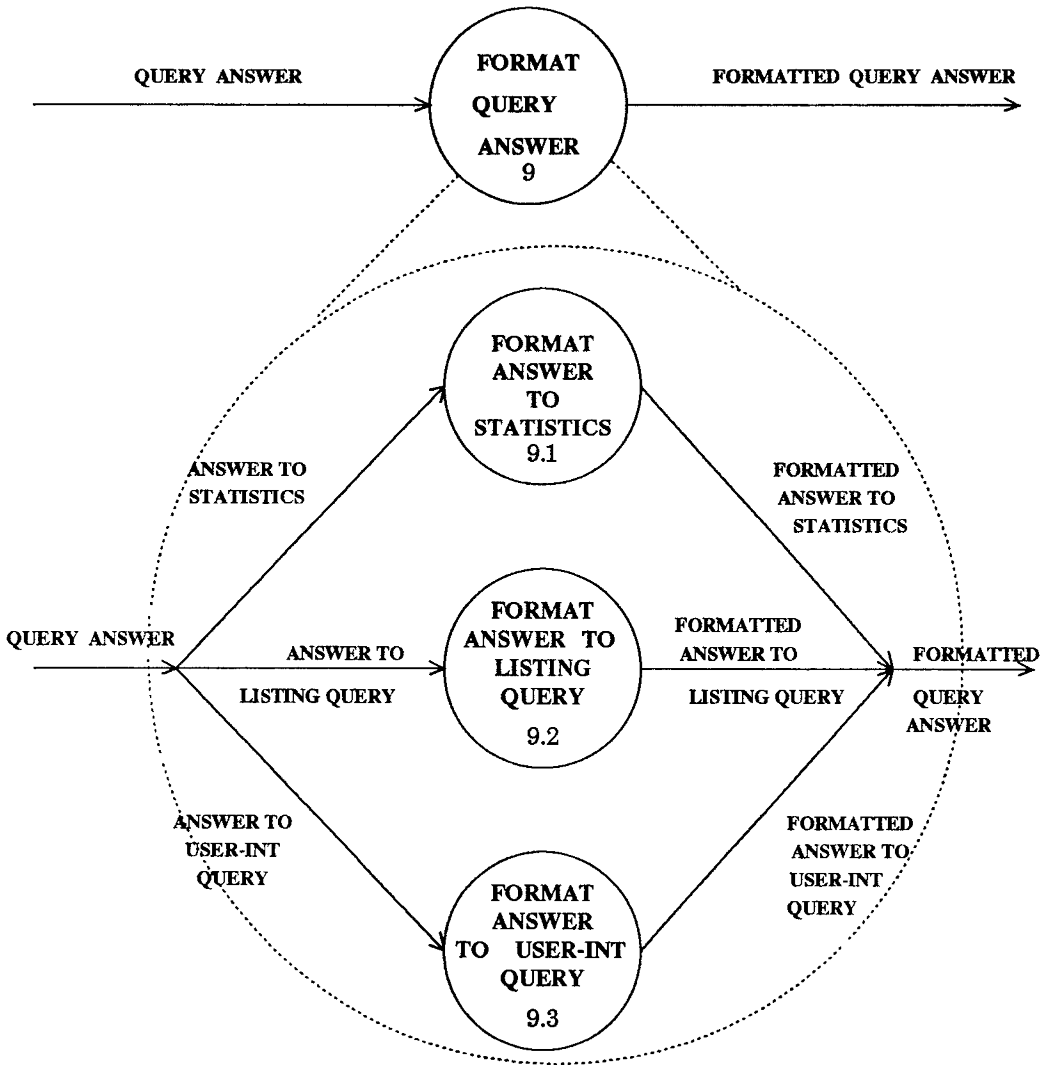

QUERY ANSWER
FORMAT QUERY ANSWER 9
FORMATTED QUERY ANSWER
FORMAT ANSWER TO STATISTICS 9.1
ANSWER TO STATISTICS
FORMATTED ANSWER TO STATISTICS
QUERY ANSWER
ANSWER TO LISTING QUERY
FORMAT ANSWER TO LISTING QUERY 9.2
FORMATTED ANSWER TO LISTING QUERY
FORMATTED QUERY ANSWER
ANSWER TO USER-INT QUERY
FORMAT ANSWER TO USER-INT QUERY 9.3
FORMATTED ANSWER TO USER-INT QUERY

Figure 3.6: Level 2: FORMAT QUERY ANSWER

9.1. FORMAT ANSWER TO STATISTICS

Input: ANSWER TO STATISTICS

Output: FORMATTED ANSWER TO STATISTICS

Pseudocode for FORMAT ANSWER TO STATISTICS

```
DO Format Answer to Statistics
```

9.2. FORMAT ANSWER TO LISTING QUERY

Input: ANSWER TO LISTING QUERY

Output: FORMATTED ANSWER TO LISTING QUERY

Pseudocode for FORMAT ANSWER TO LISTING QUERY

```
DO Format Answer to Listing Query
```

9.3. FORMAT ANSWER TO USER-INT QUERY

Inputs: ANSWER TO USER-INT QUERY

Outputs: FORMATTED ANSWER TO USER-INT QUERY

Pseudocode for FORMAT ANSWER TO USER-INT QUERY

```
DO Format User Interactive Query
```

3.2.7 Structure Charts

– Level 0

The level 0 structure chart is as shown in Figure 3.7. In the CS Department DIS :
(1) The user selects one of the menu options from the main menu as shown in the Appendix Section 7.1 i.e., user selects one of the four databases. Faculty, Course, Student, Register.
(2) Then the user selects either the update options i.e.F-ADD or F-DELETE or the query options QS or QP or QOTHERS from the submenus of the selected database.
(3) The user enters the required input data for the particular selection i.e. I/P DATA for F-ADD or F-DELETE or QOTHERS. Here "F" stands for Faculty or Course or Student or Register.

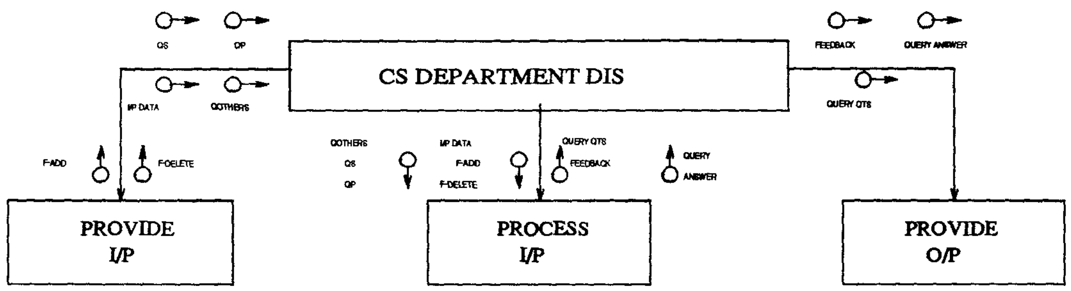


Figure 3.7: Level 0: STRUCTURE CHART

(4) All the input data along with the chosen functionality are then processed in PROCESS I/P and the corresponding outputs are then obtained i.e. Feedback, Query Qts., Query Answer.

(5) These outputs are then presented in the required format by PROVIDE O/P.

Abbreviations

1. F-ADD: Function to add data
2. F-DELETE: Function to delete data
3. QS: Query for statistics
4. QP: Listing query
5. QOTHERS: Other user interactive queries

Pseudocode for CS Department DIS

```
Do {
   DO Provide I/P
   DO Process I/P
   DO Provide O/P
   } WHILE I/P not equal to exit
```

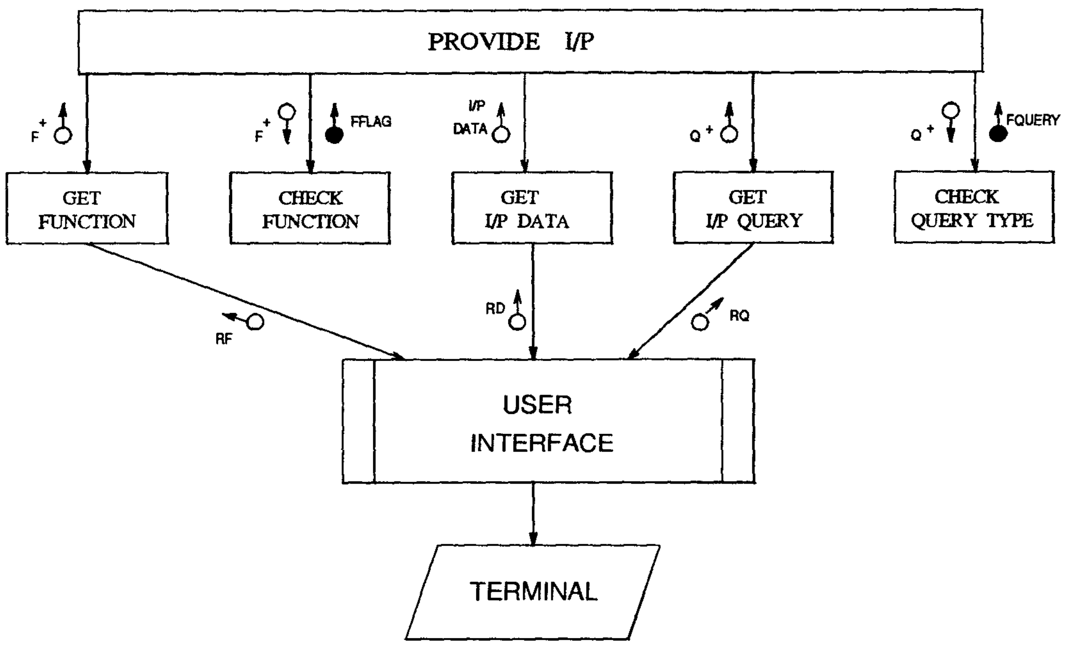


Figure 3.8: Level 1: PROVIDE I/P

– Level 1: for PROVIDE I/P

The level 1 structure chart for PROVIDE I/P is as shown in Figure 3.8. From the user interface, through the main menu (Ref. Appendix Section 7.1), the user selects the required database to be updated or queried. If this database is to be updated, the functions add-faculty, add-course, add-student, add-register or delete-faculty, delete-course, delete-student, delete-register can be selected. This function is then checked and the required input is then provided by the user. The input data provided by the user is as described above.

If the database selected is to be queried, then the required query option is selected from the menu. This selected query is then checked for the query type i.e., if it is a statistical query, a listing query or a user interactive query.

Abbreviations

1. RF: Raw Function

2. F^+: Function(Add,Delete)
3. FFLAG: Flag for function type
4. RD: Raw input data
5. RQ: Raw input query
6. Q^+: Query(Statistics,Listing,User interactive)
7. FQUERY: Flag for query type

Pseudocode for PROVIDE I/P

```
IF input = F+ THEN BEGIN
   DO Get Function
   DO Check Function
   IF FFLAG THEN
      DO Get I/P Data
END


IF input = Q+ THEN BEGIN
   DO Get I/P Query
   DO Check Query Type
END
```

Pseudocode for Get Function (Ref. Appendix Section 7.1)

```
DO {
    IF Input = FACULTY  THEN
       Display FACULTY  MENU
    IF Input = COURSE   THEN
       Display COURSE   MENU
    IF Input = STUDENT  THEN
       Display STUDENT  MENU
    IF Input = REGISTER THEN
       Display REGISTER MENU
   } WHILE Input not equal to EXIT.
```

Pseudocode for Check Function (Ref. Appendix Section 7.1)

```
DO {
    IF Input = ADD    THEN
       DO ADD FUNCTION
    IF Input = DELETE THEN
       DO DELETE FUNCTION
    IF Input = LIST   THEN
       DO LIST   FUNCTION
    IF Input = VIEW   THEN
       DO VIEW   FUNCTION
   } WHILE Input not equal to EXIT.
```

Pseudocode for Get I/P DATA

```
IF FFLAG = ADD THEN
   DO Get Input Data For Add Function
IF FFLAG = DELETE THEN
   DO Get Input Data For Delete Function
```

Pseudocode for Get I/P QUERY

```
IF RQ = faculty query THEN
   DO GET I/P Query for Faculty
IF RQ = course query THEN
```

```
   DO GET I/P Query for Course
IF RQ = student query THEN
   DO GET I/P Query for Student
IF RQ = register query THEN
   DO GET I/P Query for Register
```

Pseudocode for Check Query Type

```
IF Q+ = Statistical Query Type THEN
   Set FQUERY = STATISTICAL
IF Q+ = Listing Query Type THEN
   Set FQUERY = LISTING
IF Q+ = User Interactive Query Type THEN
   Set FQUERY = USER INTERACTIVE
```

– Level 1: for PROCESS I/P

The level 1 structure chart for PROCESS I/P is as shown in Figure 3.9. If the user selects an option to update a particular database, then he/she is required to provide the required input. This input is then processed, i.e., depending on the kind of updation, a new record is being added, or an existing record is deleted from the database. A message stating the output of this procedure is sent back as the output i.e., "feedback". If the user selects an option for querying a particular database, then depending on the type of the query, i.e., statistical, listing or user-interactive, the query is processed. If it is a statistical query, then the required statistics are calculated and the output is presented as AS. If it is a listing query, the required list of records is obtained from the database and the output is presented as AP. If it is a user interactive query, the user is asked to enter specific information according to which the required records are then retrieved and sent back as AQ.

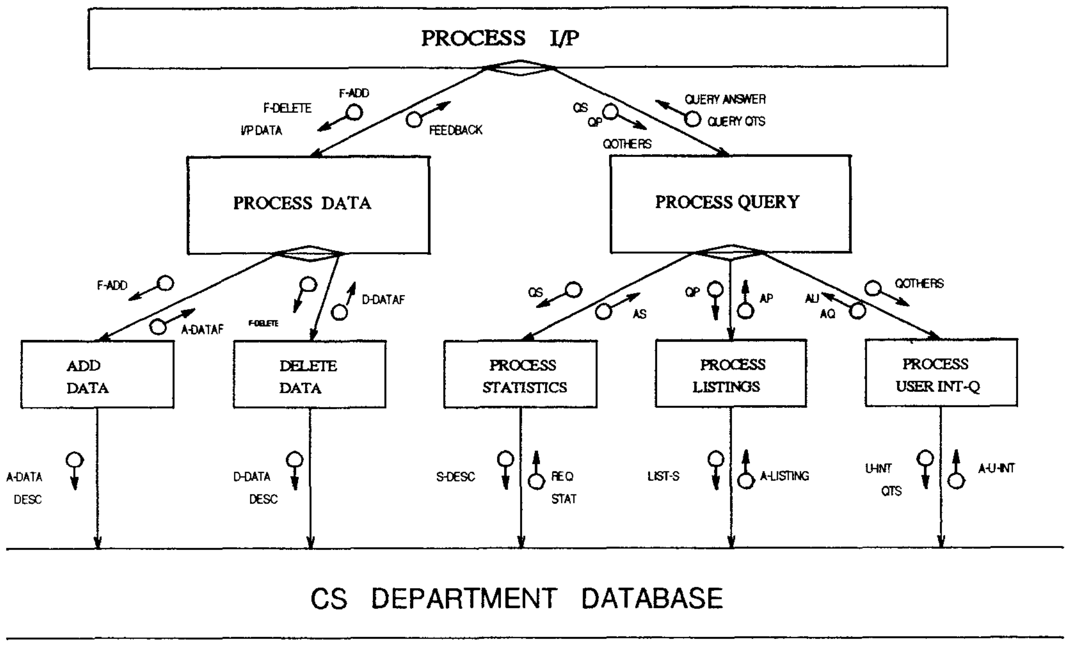


Figure 3.9: Level 1: PROCESS I/P

Abbreviations

1. A-DATAF: Added data feedback
2. D-DATAF: Deleted data feedback
3. AS: Answer to statistics
4. AP: Answer to listing query
5. AU: Answer to user interactive query
6. AQ: Query question

Pseudocode for PROCESS I/P

```
IF input = F-ADD or F-DELETE THEN
   DO Process Data
IF input = QS or QP or QOTHERS THEN
   DO Process Query
```

Pseudocode for PROCESS DATA

```
IF input = F-ADD THEN
   DO Add Data
IF input = F-DELETE THEN
   DO Delete Data
```

Pseudocode for PROCESS QUERY

```
IF input = QS THEN
   DO Process Statistics
IF input = QP THEN
   DO Process List
IF input = QOTHERS THEN
   DO Process USER-INT-Q
```

Pseudocode for ADD DATA

```
IF Input Data = New Faculty Record THEN
   DO FACULTY Add Data
IF Input Data = New Course Record THEN
   DO COURSE Add Data
IF Input Data = New Student Record THEN
   DO STUDENT Add Data
IF Input Data = New Register Record THEN
   DO REGISTER Add Data
```

Pseudocode for DELETE DATA

```
IF Input Data = Faculty Social Security Number THEN
   DO FACULTY Delete Data
```

```
IF Input Data = Course Number THEN
   DO COURSE Delete Data
IF Input Data = Student Social Security Number THEN
   DO STUDENT Delete Data
IF Input Data = Student Social Security Number THEN
   DO REGISTER Delete Data
```

Pseudocode for PROCESS STATISTICS

```
IF Input = Faculty Statistics THEN
   DO Process FACULTY Statistics
IF Input = Course  Statistics THEN
   DO Process COURSE Statistics
IF Input = Student Statistics THEN
   DO Process STUDENT Statistics
IF Input = Register Statistics THEN
   DO Process REGISTER Statistics
```

Pseudocode for PROCESS LISTINGS

```
IF Input = Faculty Listings THEN
   DO Process FACULTY Listings
IF Input = Course  Listings THEN
   DO Process COURSE Listings
IF Input = Student Listings THEN
   DO Process STUDENT Listings
IF Input = Register Listings THEN
   DO Process REGISTER Listings
```

Pseudocode for PROCESS USER INT. QUERY

```
IF Input = Faculty User Int. Query THEN
   DO Process FACULTY User Int. Query
IF Input = Course  User Int. Query THEN
   DO Process COURSE User Int. Query
IF Input = Student User Int. Query THEN
   DO Process STUDENT User Int. Query
IF Input = Register User Int. Query THEN
   DO Process REGISTER User Int. Query
```

– Level 1: for PROVIDE O/P

The level 1 structure chart for PROVIDE O/P is as shown in Figure 3.10. The feedback from the PROCESS DATA is formatted in the procedure FORMAT FEEDBACK and displayed on the user CRT. The query answers from the PROCESS QUERY are formatted accordingly for the three types of queries, i.e., statistical query, listing query, user interactive query, and presented to the user in the required format i.e., as FAS or FAP or FAQ.

Abbreviations

1. FAS: Formatted answer to statistics
2. FAP: Formatted list
3. FAU: Formatted answer to user interactive query
4. FAQ: Formatted question

Pseudocode for PROVIDE O/P

```
IF input = FEEDBACK THEN
   DO Format Feedback
```

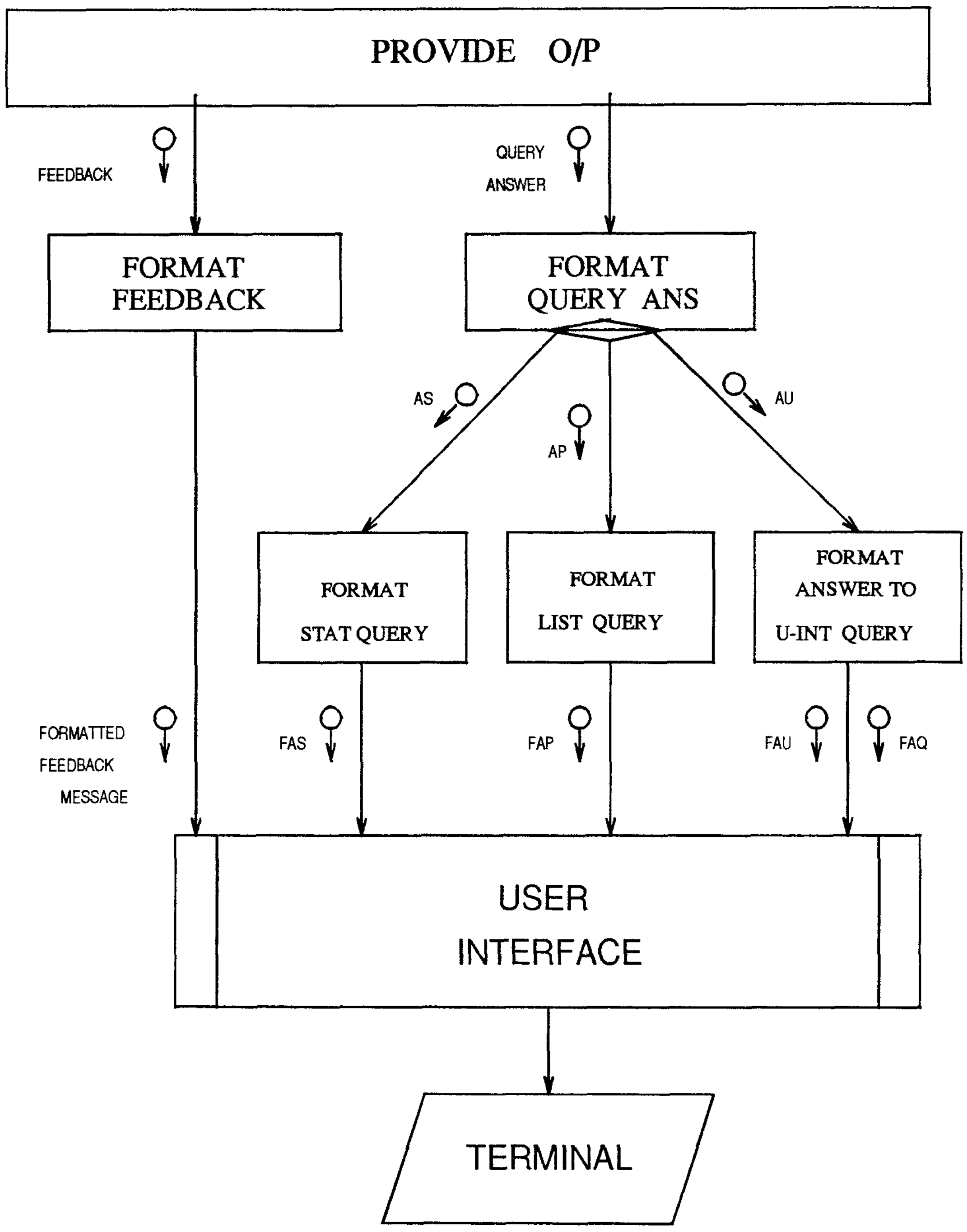


Figure 3.10: Level 1: PROVIDE O/P

```
IF input = QUERY ANSWER THEN
   DO Format Query Ans
```

Pseudocode for **FORMAT QUERY ANS**

```
IF input = AS THEN
   DO Format Stat Query
IF input = AP THEN
   DO Format List Query
IF input = AS THEN
   DO FORMAT Answer to U-INT QUERY
```

Pseudocode for **FORMAT FEEDBACK**

```
IF Feedback = ADD FEEDBACK THEN
   DO Format ADD FEEDBACK
IF Feedback = DELETE FEEDBACK THEN
   DO Format DELETE FEEDBACK
```

Pseudocode for **FORMAT STAT QUERY**

```
IF Input = Statistical Query Answer For Faculty THEN
   DO Format Statistical Query Answer For Faculty
IF Input = Statistical Query Answer For Course THEN
   DO Format Statistical Query Answer For Course
IF Input = Statistical Query Answer For Student THEN
   DO Format Statistical Query Answer For Student
IF Input = Statistical Query Answer For Register THEN
   DO Format Statistical Query Answer For Register
```

Pseudocode for FORMAT LIST QUERY

```
IF Input = List  Answer For Faculty THEN
   DO Format Listing For Faculty
IF Input = List  Answer For Course THEN
   DO Format Listing For Course
IF Input = List  Answer For Student THEN
   DO Format Listing For Student
IF Input = List  Answer For Register THEN
   DO Format Listing For Register
```

Pseudocode for FORMAT ANSWER TO U-INT QUERY

```
IF Input = Answer to U-INT Query for faculty THEN
   DO Format Answer to U-INT Query For Faculty
IF Input = Answer to U-INT Query for course THEN
   DO Format Answer to U-INT Query For Course
IF Input = Answer to U-INT Query for student THEN
   DO Format Answer to U-INT Query For Student
IF Input = Answer to U-INT Query for register THEN
   DO Format Answer to U-INT Query For Register
```

3.3 The Technical Environment

(1) At least two Sun Workstations are required to run this application, as it is a DIS. This application is designed in such a way that it can run on a network of SUN workstations. Currently only two SUN workstations are being used. Actually, right now it runs on a SPARC station SLC and a SUN workstation. Due to the difference in the machine architectures, if the software has to be moved around, the application has to be recompiled.

(2) Availability of mouse, terminal, keyboard and ethernet is assumed. The user interface which is presently available does not need the mouse. Ethernet connects the different workstations.

(3) Version of SunOS used is Release 4.1.

(4) This application is written in standard 'C'. Due to the difference in the machine architecture, as stated above, the programs have to be recompiled if the software has to be moved around.

(5) The "databases" are actually flat files which store the records. The fields in these records are delimited by a single space.

(6) Version of Sun's ONC RPC is Release 4.0. TCP is used to create a connection. TCP is a connection oreinted transport protocol. TCP lies on top of the ethernet layer. By TCP we mean Internet Transmission Control Protocol.

(7) Interface Requirements: The user interface that has been implemented is very basic. It is implemented in 'C'.

CHAPTER 4
MAPPING THE APPLICATION TO THE CONCEPTUAL ARCHITECTURE

According to the conceptual architecture, the implementation of the prototype is divided into different building blocks. Building blocks are the components which provide the different functionalities in an application domain. The building blocks act like black-boxes, offering services via a predefined interface, but hiding implementation details.

In this application, a building block is realized as a subroutine, a main program, or a system of programs written in C. If a building block consists of a system of programs, then these programs directly call each other. All these programs together form one building block since they contribute towards the same functionality. Decomposition rules used here on a conceptual level separate functionality of the user-interface, the data storage units, and of additional functional elements into different building blocks [2]. These building blocks are placed conceptually in three different layers. These layers as seen in Figure 4.1 are described below:

- Level 1 is the User Layer

  It provides a menu based interface to select the functions: add or delete and allows the user to view different records (Ref. Chapter 3).

- Level 2 is the Processing Layer

  It consists of building blocks that actually process the data from the data layer described below, using combinations of building blocks that exits in the data layer.

- Level 3 is the Data Layer

  It consists of building blocks that perform the basic functions add, delete, list

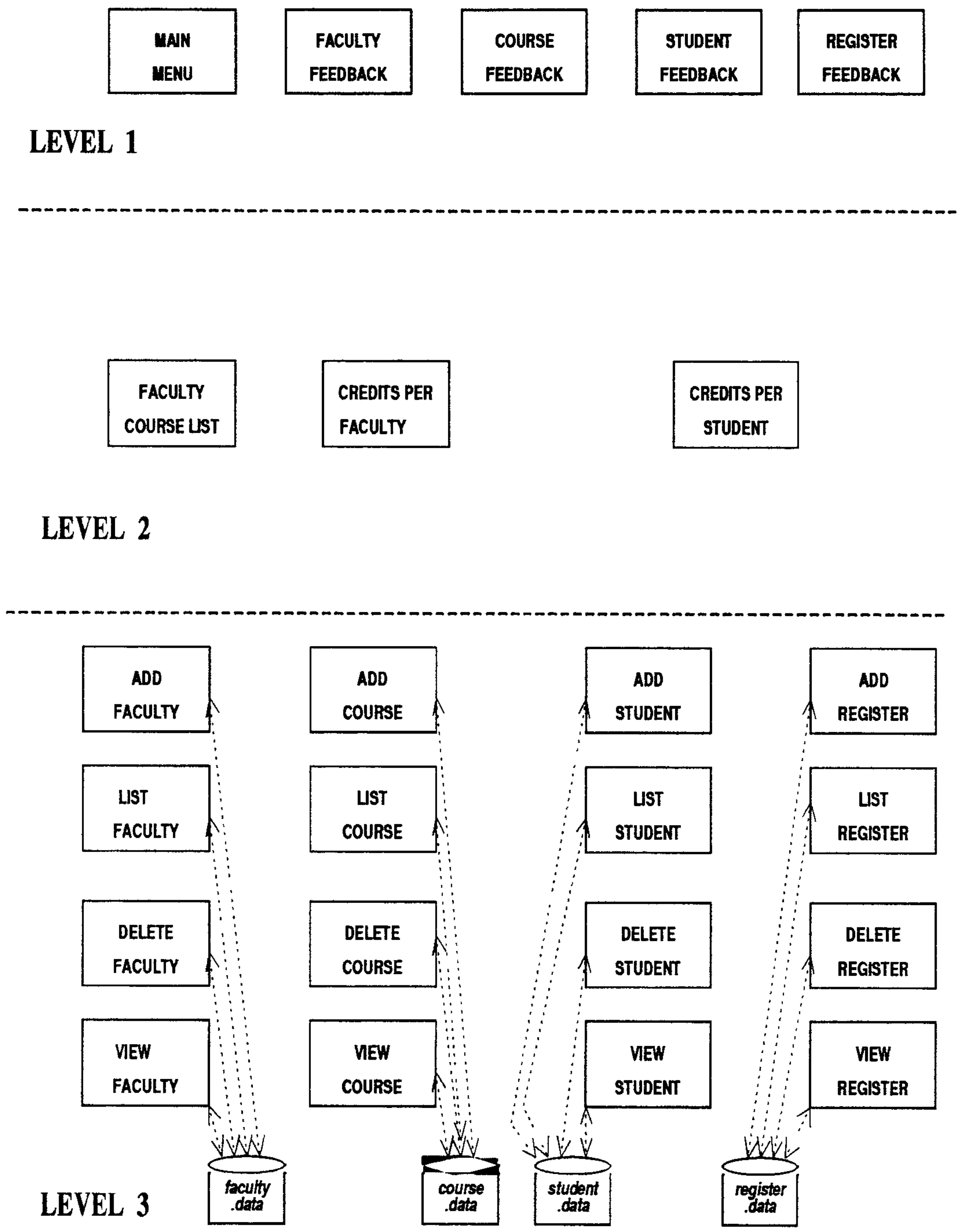


Figure 4.1: Building Blocks of the CS Department DIS

and view a specific record (Ref. Chapter 3). These building blocks actually handle the data in the databases.

The building blocks do not contain application functionality from more than one layer and the interfaces between the functions in one building block and another building block are well defined, so that the functions of one layer are decoupled from the functions of another layer. This grouping of the building blocks according to their functionality into the different layers corresponds to the requirements of the conceptual architecture described in Chapter 2.

These building blocks are capable of communicating with each other and the user is able to communicate with every block via the user interface irrespective of the internal implementation and environments on which the building blocks reside (This is equivalent to *interoperability* as described in the $OSCA^{TM}$ manual [1]).

4.1 Level 1 Building Blocks

The level I consists of the following building blocks: (Ref. Fig. 4.1)

1. Main Menu
2. Faculty Feedback
3. Course Feedback
4. Student Feedback
5. Register Feedback

Main Menu

Components of the main menu building block are as shown in Figures 4.2 and 4.3. The main menu building block comprises of all C programs as shown. All these C programs call each other directly and they together work towards the

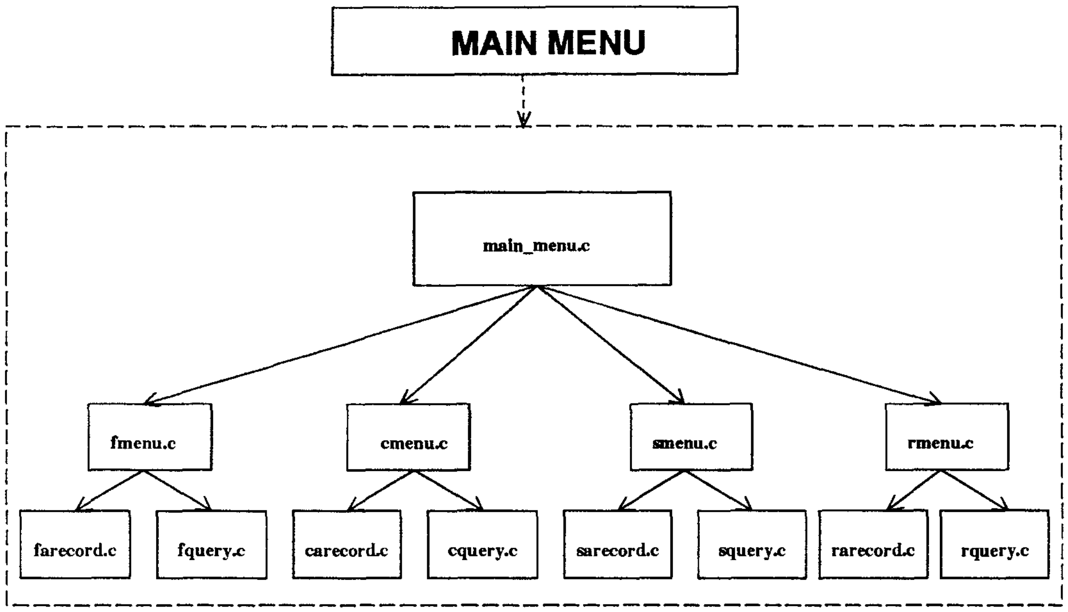
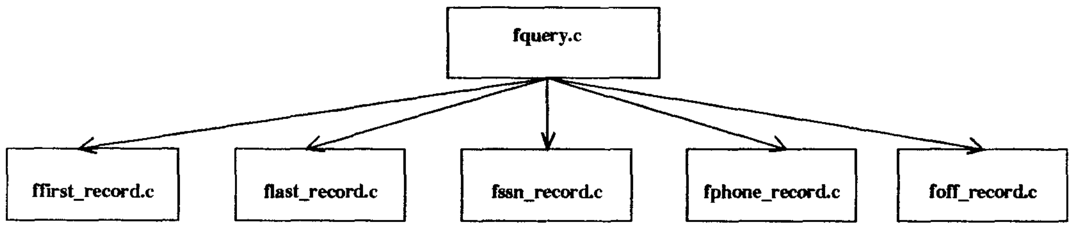


Figure 4.2: Components of the Main Menu Building Block: Part A

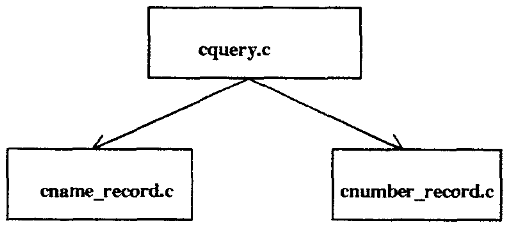
formation of the user interface input functionality. The main menu building block corresponds to the structure chart for *PROVIDE I/P* as seen in Figure 3.8. This provides a simple user interface for updating or querying the faculty, course, student, register databases.

Feedback

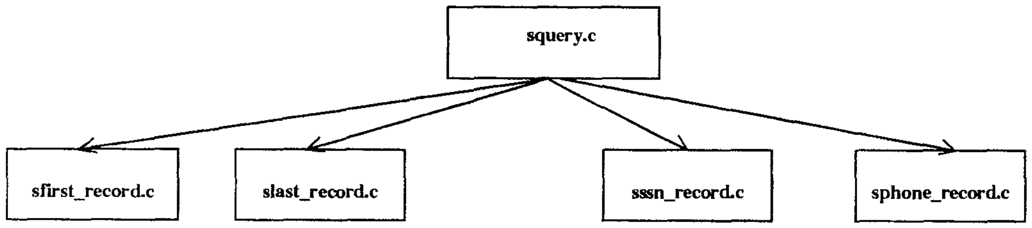
Components of the feedback building blocks are as shown in Figure 4.4. Each feedback building block comprises of a single C program as shown in the Figure 4.4. Each of these C programs work towards the formation of the user output functionality for faculty, course, student and register respectively. The feeback building block corresponds to the structure chart for *PROVIDE O/P* as seen in Figure 3.10. The feedback implies faculty, course, student, register feedbacks. It displays the results of the queries as requested by the user.



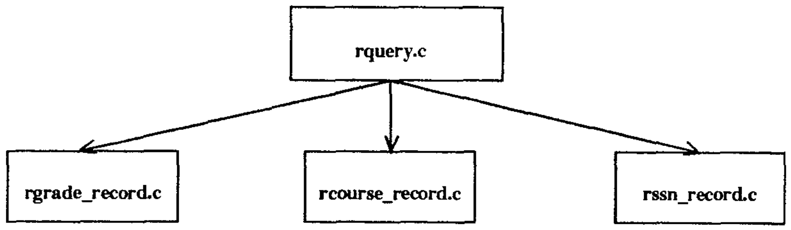
COMPONENTS OF fquery.c



COMPONENTS OF cquery.c



COMPONENTS OF squery.c



COMPONENTS OF rquery.c

Figure 4.3: Components of the Main Menu Building Block: Part B

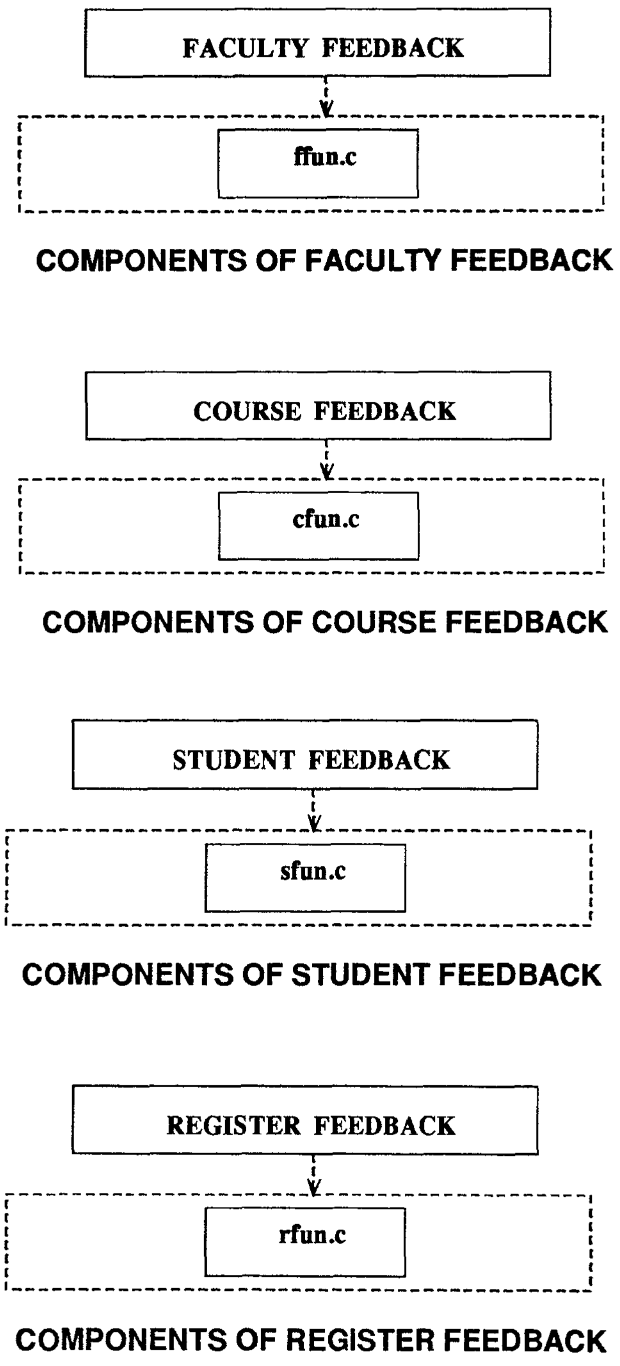

FACULTY FEEDBACK
ffun.c
COMPONENTS OF FACULTY FEEDBACK
COURSE FEEDBACK
cfun.c
COMPONENTS OF COURSE FEEDBACK
STUDENT FEEDBACK
sfun.c
COMPONENTS OF STUDENT FEEDBACK
REGISTER FEEDBACK
rfun.c
COMPONENTS OF REGISTER FEEDBACK

Figure 4.4: Components of Feedback Building Blocks

4.2 Level 2 Building Blocks

The level 2 consists of the following building blocks:(Ref. Fig. 4.1)

1. Faculty Course List
2. Credits Per Faculty
3. Credits Per Student

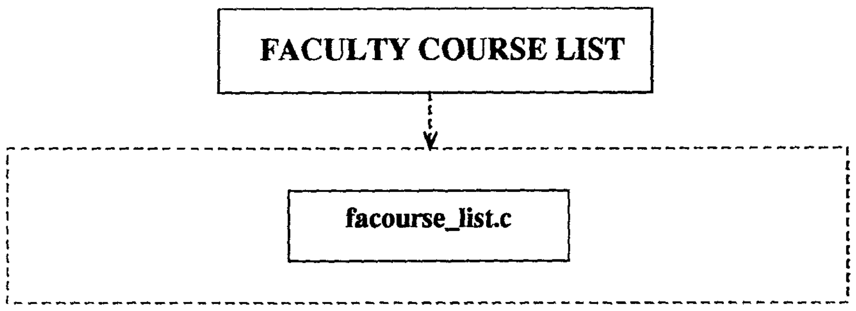
These building blocks fall in level 2 as they all provide functionality to answer complex queries and control process flow. They contain functionality that is not present in the data layer or the user layer. Hence, they do not steward data or support human users directly. These building blocks contain certain messages to access the building blocks in the data layer. They also

Faculty Course List

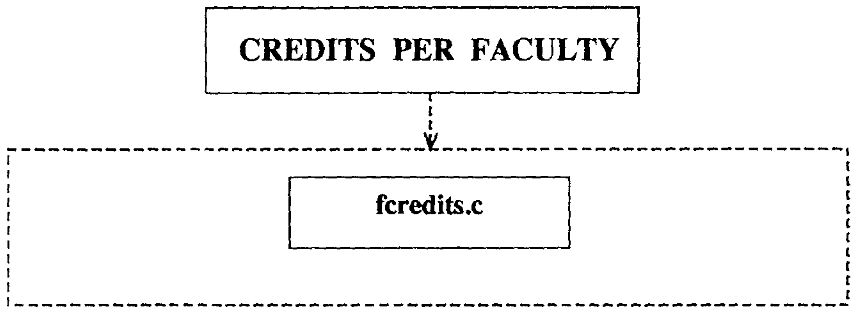
Provides a list of courses taught by a particular faculty member using the last name of the faculty as the search key (Ref. Fig. 4.5). Components of the faculty course list building block are as shown in Figure 4.5. The faculty course list building block comprises of the C program *facourse_list.c.* This C program sends a message to the trader to access the view faculty building block. The data is then sent back to this program. Another message containing this data is sent to the trader to access the list course building block. The required list is then displayed. The faculty course list building block corresponds to the PROCESS USER INT-Q block in the structure chart for *PROCESS I/P* as seen in Figure 3.9.

Credits per Faculty

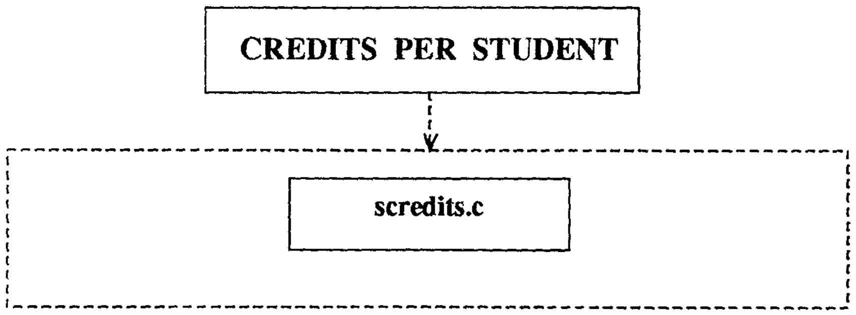
Provides the total number of credits taught by a particular faculty member using the last name of the faculty as the search key (Ref. Fig. 4.5). Components of the credits per faculty building block are as shown in Figure 4.5. The credits per faculty building block comprises of the C program *fcredits.c.* This works towards the formation of the credits per faculty functionality. This C program sends a



COMPONENTS OF FACULTY COURSE LIST BUILDING BLOCK



COMPONENTS OF CREDITS PER FACULTY BUILDING BLOCK



COMPONENTS OF CREDITS PER STUDENT BUILDING BLOCK

Figure 4.5: Level 2 Building Blocks

message to the trader to access the faculty course list building block. The sequence of faculty course building block operations are as explained above. The credits per faculty building block corresponds to the PROCESS STATISTICS block in the structure chart for *PROCESS I/P* as seen in Figure 3.9.

Credits per Student

Provides the total number of credits earned by a student using the social security number of the student as the search key (Ref. Fig. 4.5). Components of the credits per student building block are as shown in Figure 4.5. The credits per student building block comprises of the C program *scredits.c.* This works towards the formation of the credits per student functionality. This C program sends a message to the trader to access the view student building block. The data is then sent back to this program. Another message containing this data is sent to the trader to access the list register building block. The required data is then computed and displayed. The credits per student building block corresponds to the PROCESS STATISTICS block in the structure chart for *PROCESS I/P* as seen in Figure 3.9. This layer contains a variety of service processing layer building blocks. For example, the faculty course list building block, is a basic service processing layer building block. The basic service processing layer provides answers to high level queries, using the data retrieved by a data layer building block and then manipulating the data according to the query requirement. The credits per faculty is an advanced service processing layer building block. An advanced service processing layer building block provides answers to high level queries by using the output data of a basic service processing layer building block and then processes this data further to suit the required queries requirements.

4.3 Level 3 Building Blocks

The level 3 consists of the following building blocks: (Ref. Figures 4.1, 4.6, 4.7, 4.8, 4.9).

1. Add Faculty
2. List Faculty
3. Delete Faculty
4. View Faculty
5. Add Course
6. List Course
7. Delete Course
8. View Course
9. Add Student
10. List Student
11. Delete Student
12. View Student
13. Add Register
14. List Register
15. Delete Register
16. View Register

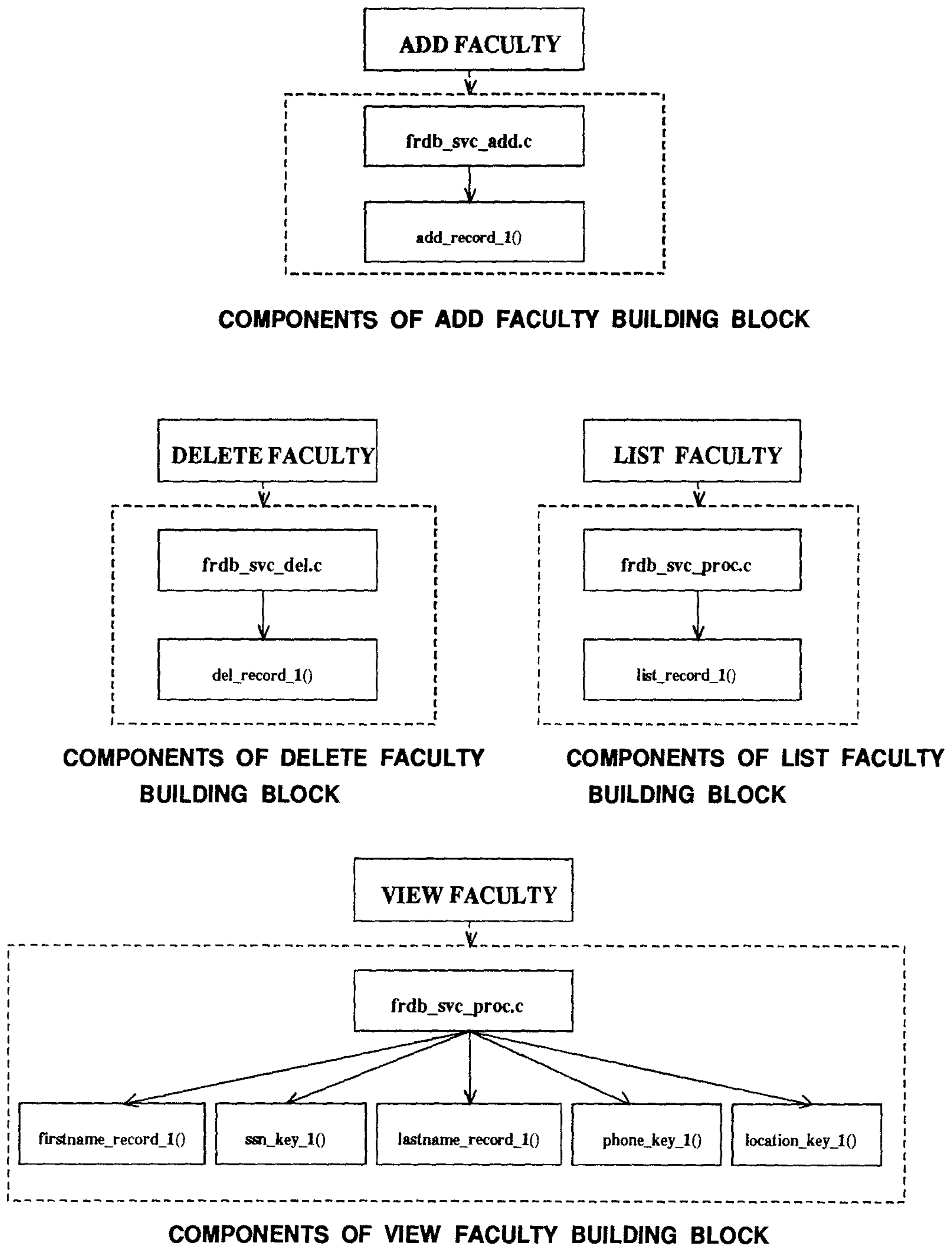
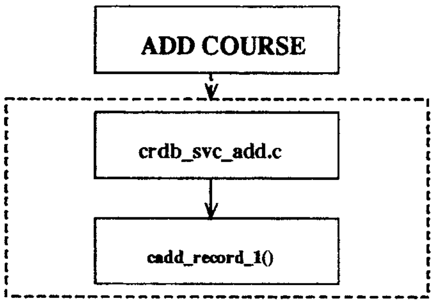
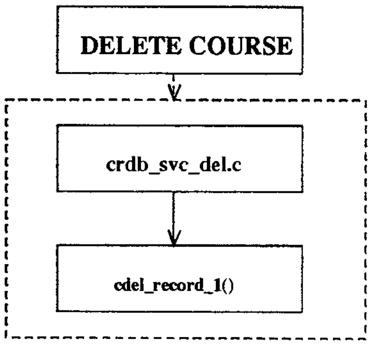


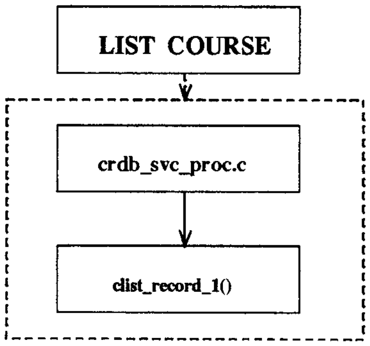
Figure 4.6: Faculty Building Blocks



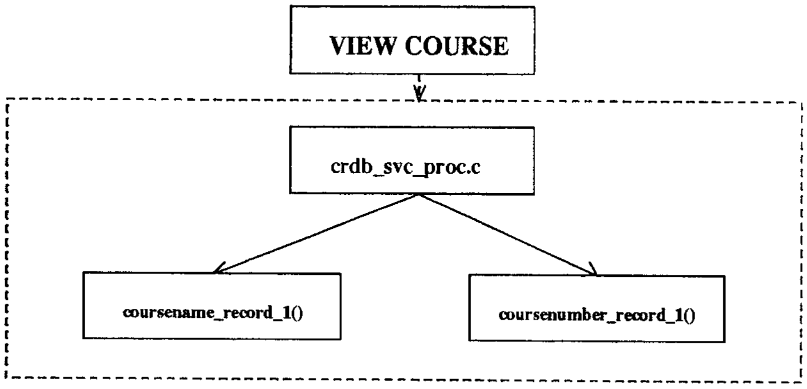
COMPONENTS OF ADD COURSE BUILDING BLOCK



COMPONENTS OF DELETE COURSE BUILDING BLOCK



COMPONENTS OF LIST COURSE BUILDING BLOCK



COMPONENTS OF VIEW COURSE BUILDING BLOCK

Figure 4.7: Course Building Blocks

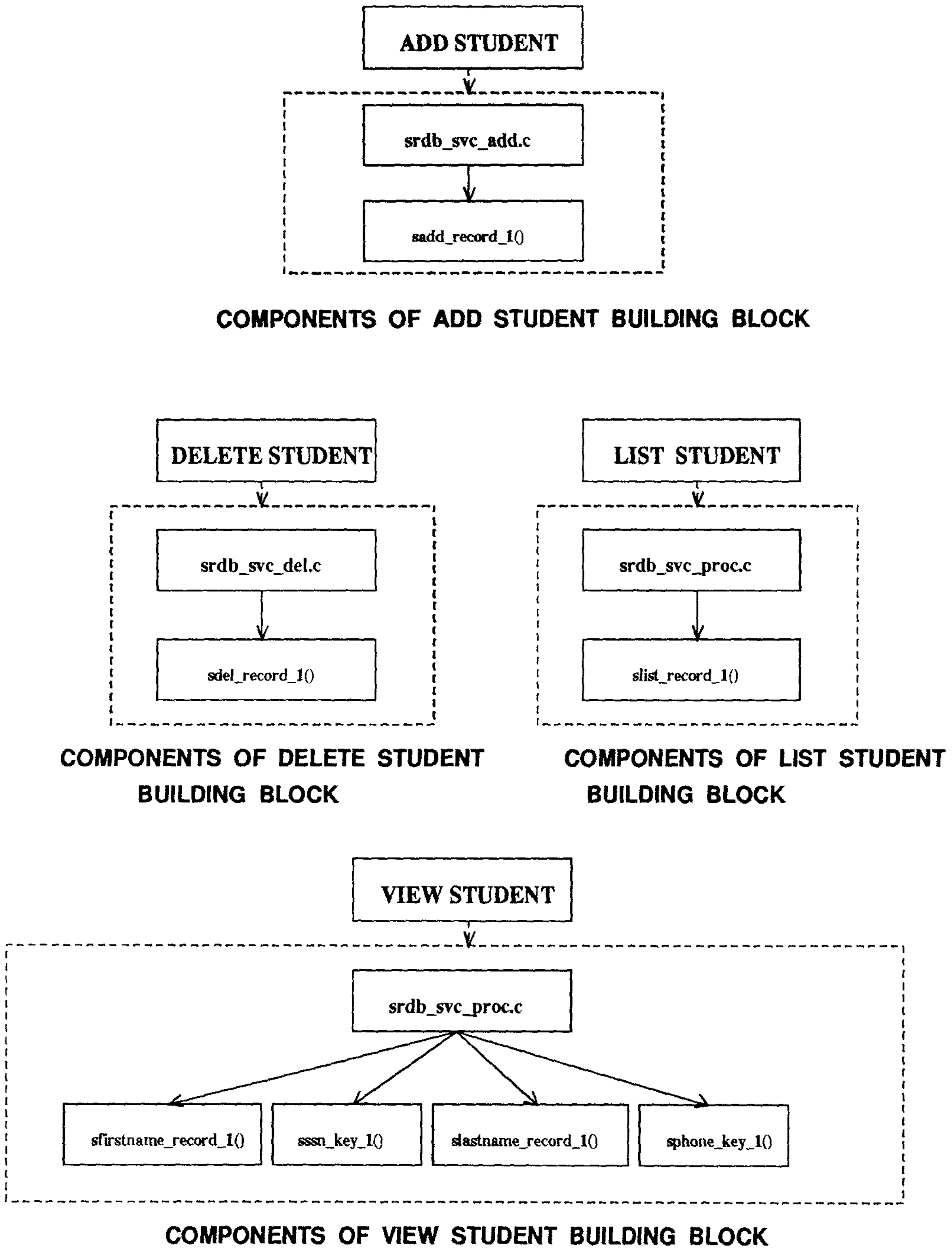


Figure 4.8: Student Building Blocks

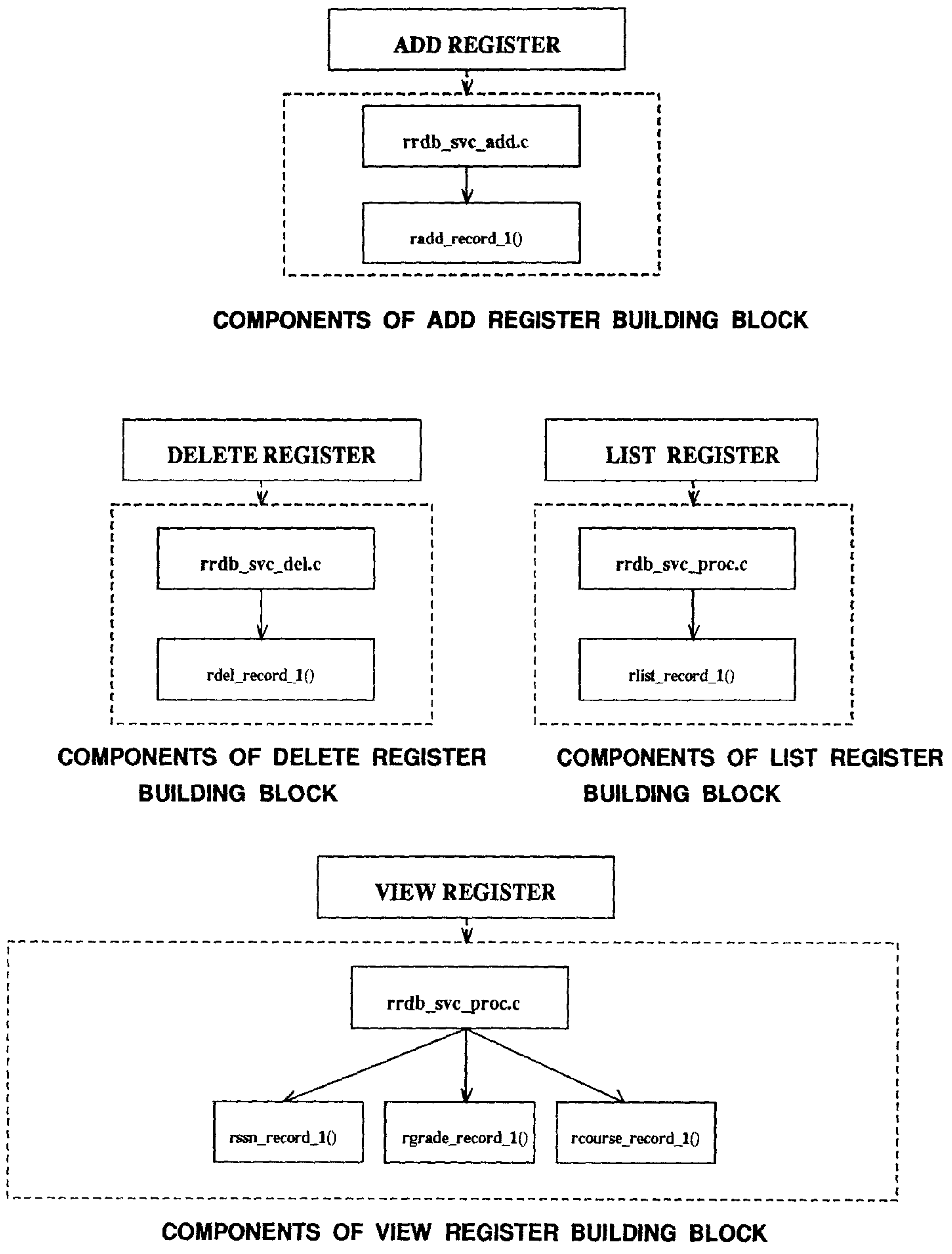


Figure 4.9: Register Building Blocks

ADD Building Blocks

Adds a new record to the specified database. The ADD faculty building block comprises of a C program for adding a new record in faculty database (Ref. Fig. 4.6). The ADD course building block comprises of a C program for adding a new record in course database (Ref. Fig. 4.7). The ADD student building block comprises of a C program for adding a new record in student database (Ref. Fig. 4.8). The ADD register building block comprises of a C program for adding a new record in register database (Ref. Fig. 4.9). The ADD building blocks correspond to the ADD DATA block in the structure chart for *PROCESS I/P* as seen in Figure 3.9.

LIST Building Blocks

Lists all records in the specified database. The LIST faculty building block comprises of a C program for listing particular records from the faculty database (Ref. Fig. 4.6). The LIST course building block comprises of a C program for listing particular records from the course database (Ref. Fig. 4.7). The LIST student building block comprises of a C program for listing particular records from the student database (Ref. Fig. 4.8). The LIST register building block comprises of a C program for listing particular records from the register database (Ref. Fig. 4.9). The LIST building blocks correspond to the PROCESS LISTINGS block in the structure chart for *PROCESS I/P* as seen in Figure 3.9.

DELETE Building Blocks

Deletes a record from the specified database. In case of faculty, student, and register databases the social security number determines the record to be deleted. In case of course database the course number determines the record to be deleted. The DELETE faculty building block comprises of a C program for deleting a specific record from faculty database (Ref. Fig. 4.6). The DELETE course building block comprises of a C program for deleting a specific record from course database (Ref. Fig. 4.7). The DELETE student building block comprises of a C program for deleting a specific record from student database (Ref. Fig. 4.8). The DELETE register building block comprises of a C program for deleting a specific record from register database (Ref. Fig. 4.9). The DELETE building blocks correspond to the DELETE DATA block in the structure chart for *PROCESS I/P* as seen in Figure 3.9.

View Faculty

Displays the required record/s from the faculty database based on either of the following attributes: First Name, Last Name, Social Security Number, Phone, Office Number. Components of the view faculty building block are as shown in Figure 4.6. The view faculty building block comprises of the C program *frdb_svc_proc.c*, which in turn comprises of the subroutines as shown in the figure. These work towards the formation of the faculty view functionality. The view faculty building block corresponds to the PROCESS USER INT-Q block in the structure chart for *PROCESS I/P* as seen in Figure 3.9.

View Student

Displays the required record/s from the student database based on either of the following attributes: First Name, Last Name, Social Security Number, Phone. Components of the view student building block are as shown in Figure 4.8. The view student building block comprises of the C program *srdb_svc_proc.c*, which in turn comprises of the subroutines as shown in the figure. These work towards the formation of the student view functionality. The view student building block corresponds to the PROCESS USER INT-Q block in the structure chart for *PROCESS I/P* as seen in Figure 3.9.

View Course

Displays the required record/s from the course database based on either of the following attributes: Course Number, Course Name. Components of the view course building block are as shown in Figure 4.7. The view course building block comprises of the C program *crdb_svc_proc.c*, which in turn comprises of the subroutines as shown in the figure. These work towards the formation of the course view functionality. The view course building block corresponds to the PROCESS USER INT-Q block in the structure chart for *PROCESS I/P* as seen in Figure 3.9.

View Register

Displays the required record/s from the register database based on either of the following attributes: Grade in a particular course, Course Number, Student Social Security Number. Components of the view register building block are as shown in Figure 4.9. The view register building block comprises of the C program *rrdb_svc_proc.c*, which in turn comprises of the subroutines as shown in the figure. These work towards the formation of the register view functionality. The view register building block corresponds to the PROCESS USER INT-Q block in the structure chart for *PROCESS I/P* as seen in Figure 3.9.

As seen in figures 4.6, 4.7, 4.8 & 4.9, the components of VIEW and LIST building blocks consist of the same C file. This C file consists of subroutines as seen in these figures. These subroutines are physically placed in the same C file but operate as complete independent blocks. They have no kind of interaction or direct communication between each other. If for any reason, this arrangement appears to violate the conceptual integration architecture used, the subroutines can be placed in different files without affecting any kind of functionality of the software.

4.4 Communication between Building Blocks on a Conceptual Level

On an abstract level, the application is seen as a unified whole. The fact that the building blocks are distributed on two or more workstations is ignored as is the existence of a trader which handles communication. (Ref. Fig. 4.10 & Sections 2.1, 2.2 & 2.3)

For example,

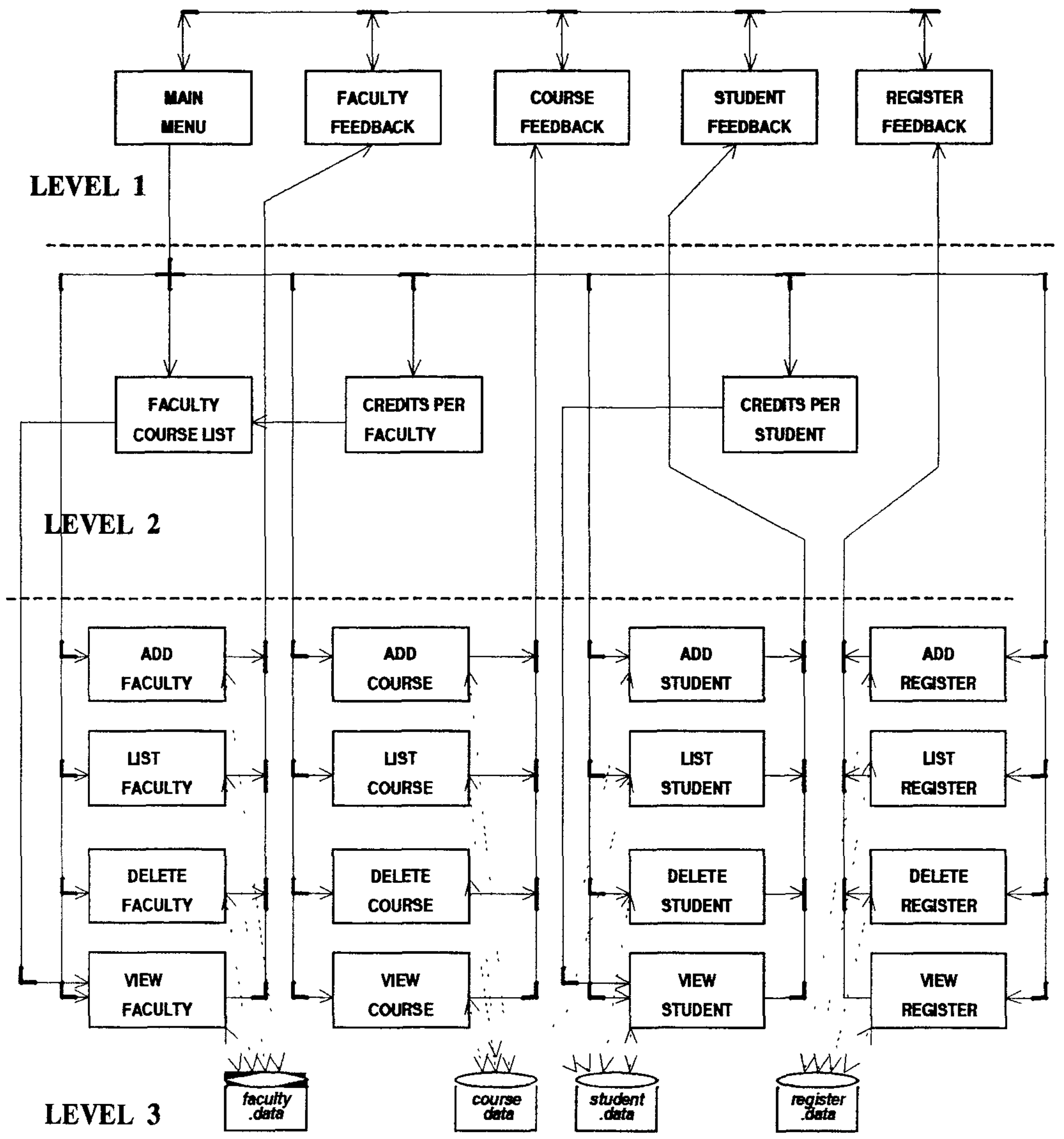


Figure 4.10: Communication between Building Blocks on a Conceptual Level

1. The main menu building block in Level 1 is seen to communicate with all the other building blocks in Level 1, 2, 3.

2. The faculty course list building block in Level 2 communicates with the view faculty building block in Level 3, which in turn communicates with list course building block in Level 3.

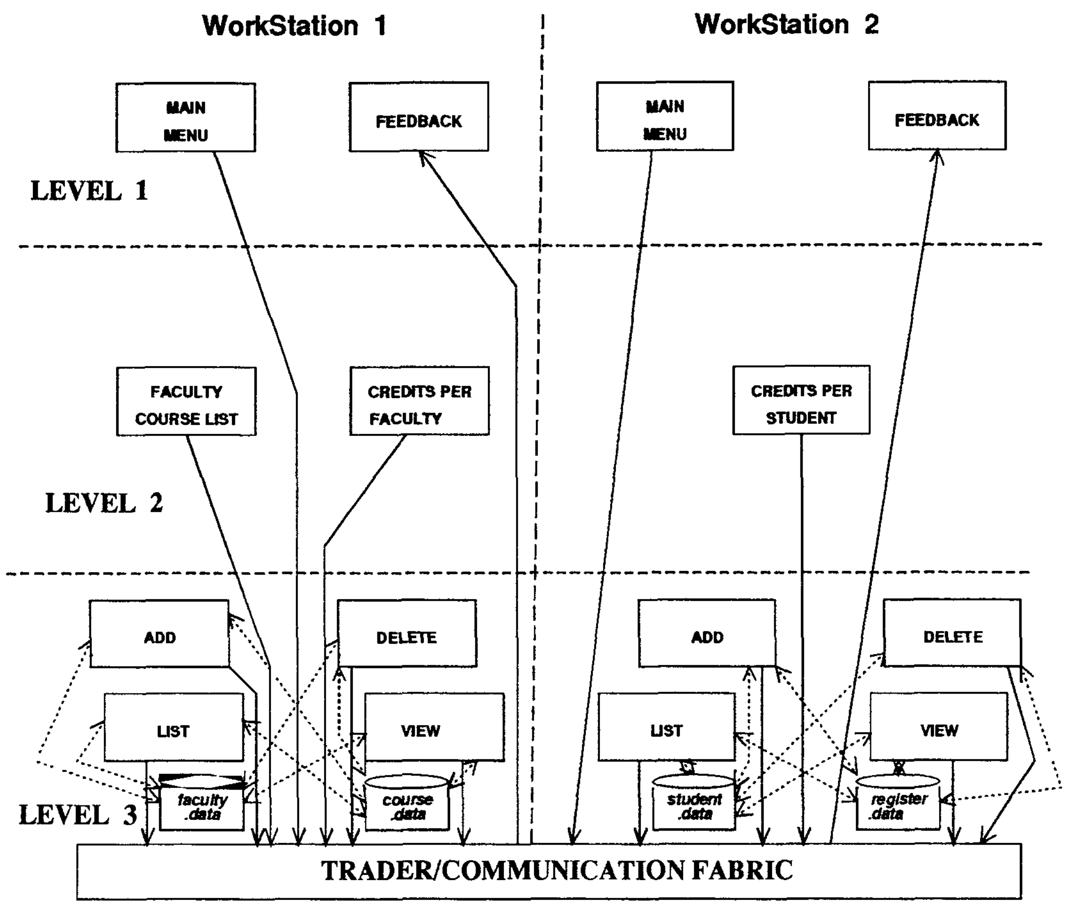


Figure 4.11: Transparent Communication between Building Blocks using a trader

4.5 Communication Using A Trader

The prototype built according to the concepts of communication (with a trader) as outlined in Sections 2.6 & 2.7, is as shown in Fig. 4.11. As seen in the figure all the building blocks communicate with each other through the trader which is the software communication fabric. The building blocks do not communicate directly with each other as seen in the conceptual level description. Also, the building blocks do not exist on the same workstation. They are distributed on two workstations (Ref. Fig. 4.11).

4.5.1 Actual Operation Description

Consider a user working on workstation 1. This user selects an option from the main menu building block which exists on workstation 1. A message stating the function selected by the user is transmitted to the trader. The trader knows where the building block with the required functionality exists (i.e. on which workstation the required building block physically exists). The message is then passed to that particular building block requesting the service required. The service is then performed by the building block and the result is then passed back to the trader. The trader passes the result to the feedback building block, which in turn displays it in the required format on the screen of workstation 1.

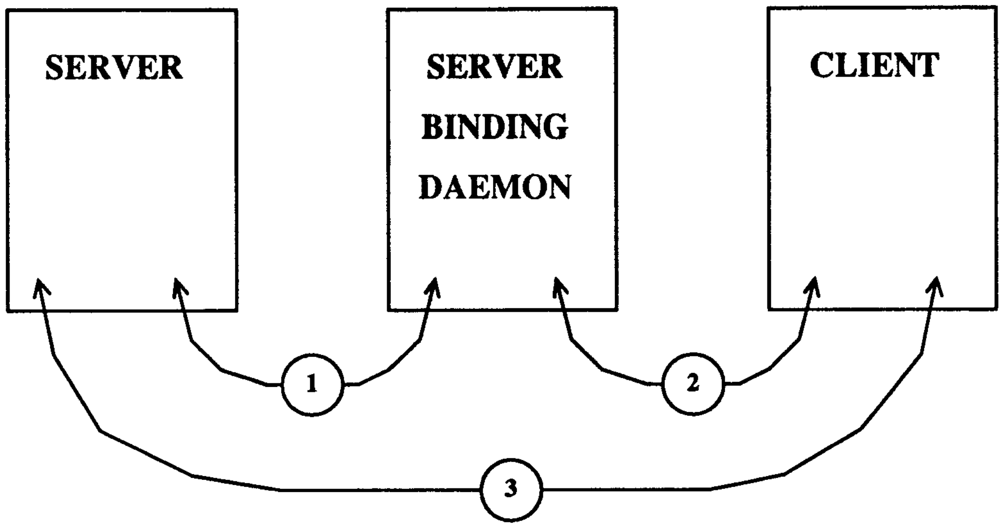
CHAPTER 5
AN IMPLEMENTATION OF THE TRADER USING RPC

5.1 Basics of Remote Procedure Calling

RPC [3], is a sophisticated programming tool that lets you write powerful distributed applications in a client server model. In the client-server communication model, servers are the distributors of the network's services. To adequately perform their jobs, servers must be registered and listening at addresses known to client computers on the network. Or, when initiated, they must register themselves with the appropriate mapping daemon (whose address is known to client machines). These registration and addressing conventions provide a mechanism for clients to establish communications with a particular service. An RPC system makes the procedures for finding and opening connections with a server systematic and flexible.
Three steps are involved in initiating an RPC : (Ref. Fig. 5.1)

1. When servers first start up, they register themselves with their local portmap, using a unique RPC server program number that specifies the version number of the application they support. A port is a logical network communication channel. The portmapper is responsible for mapping services to the ports. The portmapper on every host is well known, it is always assigned to port number 111. This allows direct access by both the client and server to the machine's portmap via the function *portmap()* [11]. By servers registering themselves, we mean that they make themselves accessible to host on the network [11].

2. A client can ask portmaps on the network to locate a particular program number and version of the service. Requests can be made singly or in broadcast



1. Server registers services.
2. Client queries binding daemon to find address of a service.
   Daemon provides server address or issues an error message.
3. Client sends request for a specific procedure.
   Server replies or issues error message.

Figure 5.1: The three steps behind remote procedure calling

fashion.

3. Once the client machine finds the address at which the requested server is listening, the client sends a request for a specific procedure number. If it is available, the server uses the accompanying arguments to call the appropriate service procedure and returns to the client a reply containing the results.
   A request is the information created by the client process RPC code and transmitted to the server according to the RPC protocol, to attempt to initiate the execution of a procedure within the server application. Necessary procedure arguments are included [11].
   A reply is the information assembled by the server RPC code and transmitted back to the client, according to the RPC protocol, to return the results of the remote execution. The information in a reply connotes either a success procedure call or a failure at the server [11].
   This represents one RPC cycle.
   A daemon is a program designed to run continuously in the background, lying dormant until some condition is met [11]. Most servers are daemons.
   Binding is the act of associating a server with a socket. When ONC RPC server transport handles are created, they are bound to a certain network port address [11].

If the server can't provide the requested service or a nonfatal error occurs, the server sends a reply to the client indicating the type of error encountered. If a client makes several requests of one server, the reply to the second request in the sequence may be postponed until a certain number of requests have been processed.

A remote procedure call is similar to a local procedure call, i.e., execution within the calling procedure is postponed until the called procedure executes. Variables are passed to the called procedure and results are passed back to the calling procedure, which resumes execution. In case of local procedure calls, this activity

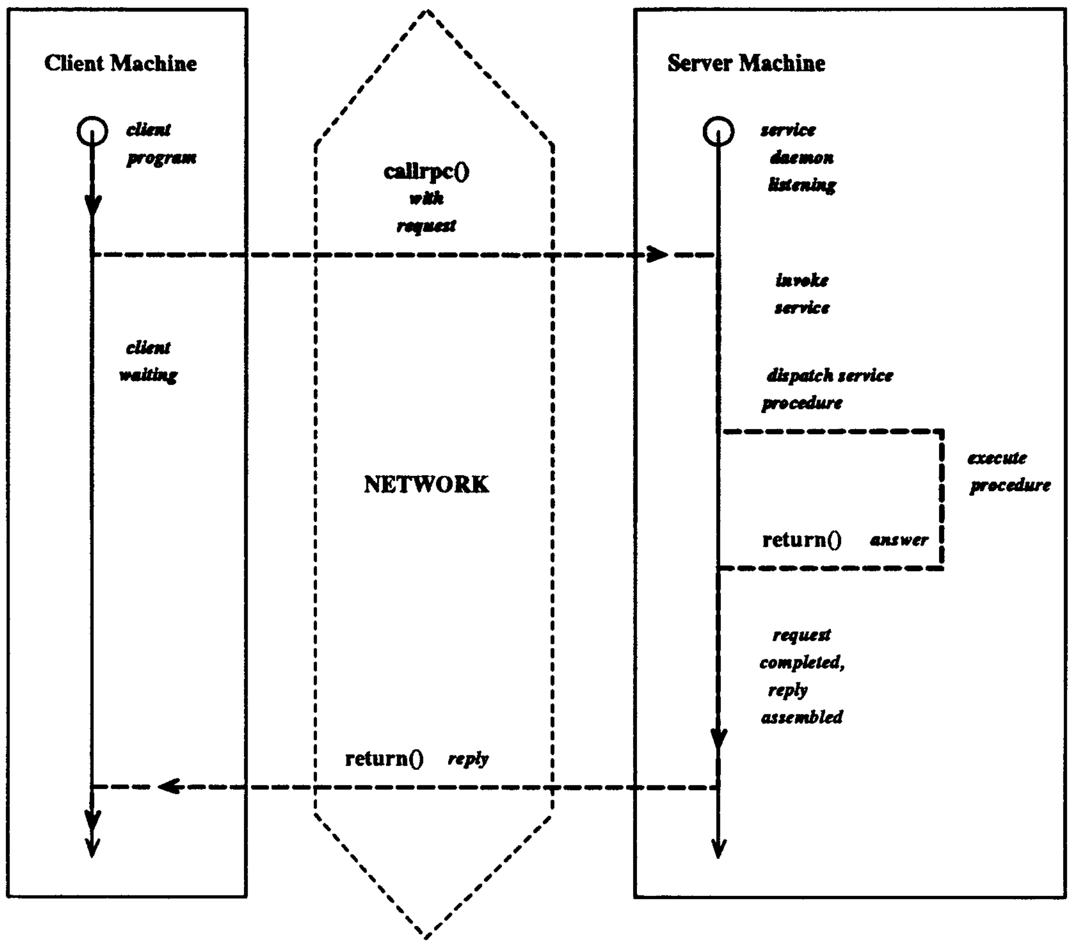


Figure 5.2: RPC Communication

takes place within one process and address space, whereas in the RPC model, called procedures are executed in different processes and either in the same address space or different address spaces (Ref.Fig. 5.2).

As seen in Figure 5.2 [11]:

- The client sends out a request over the network. The service daemon is constantly listening for a request. When a request is received, it invokes the service. The appropriate procedure is dispatched. The request is executed and the reply is returned over the network to the client.

- The client program is inactive between the time of the request and when it

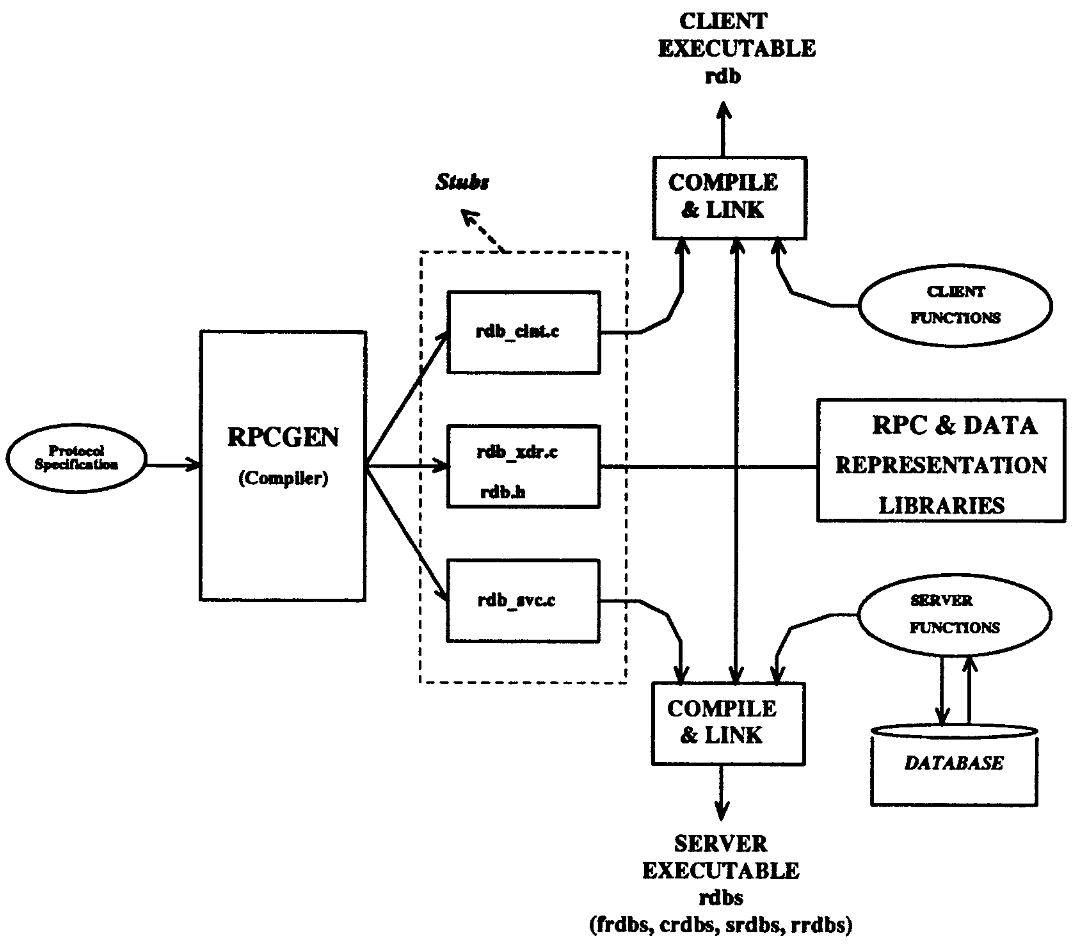


Figure 5.3: Application development with RPCGEN compiler

receives a reply.

- The client and server machines may be the same.

5.2 Development of an Application using RPCGEN

The application development with ONC (Open Network Computing) RPCGEN compiler [3] is as shown in Figure 5.3. The RPCGEN protocol compiler automatically handles the creation of stubs and filters, as seen in Figure 5.3, even producing a skeleton of client and server source files. These client and server source files are referred to as client and server stubs. RPCGEN handles the intricacies of processing within the stubs it generates for linking the client and server. These stubs take care of low level communications and the format in which the data is represented. To accommodate a variety of client operating systems, compilers, and architectures, server data is transmitted in a universally accepted format. For ONC RPC, that format is eXternal Data Representation (XDR). The RPCGEN compiler enables programmers to specify the service procedures and data structures to be exchanged while leaving the translation processing to the stubs. The RPC & Data Representation library contains filters for translating built-in C types as well as more complex types, such as strings and variable length vectors. By putting filters together, the RPCGEN compiler generates filters, capable of translating any kind of data structure. The Client functions are programs written in 'C' to call the server functions. The Server functions are programs written in 'C' to provide the various service functionalities offered by the application. The database as seen in the Figure 5.3, exists on the server side only. This is because the server provides the required functionality to update or query the data in the database. Hence it should have direct access to the data. On the other hand, the client just requests the services from the server. Therefore the database need not exist on the client side. A "cc" compiler is used to compile and link the client functions along with the client stubs and XDR files to create the client executables and it is used to compile and link the server functions along with the server stubs and XDR files to create the server executables.

Four steps are involved in developing the client and server executables for an RPC application (Ref. Fig. 5.3).

1. Protocol Specification

2. Protocol Compilation (using RPCGEN compiler)

Results of Protocol Compilation:

- Client Stubs
- Server Stubs
- xdr files
- Header Files

3. Writing Client and Server Functions

4. Compilation and linking of library routines and stubs and functions generated during 2 and 3

Protocol Specification:

Here the client-server interface is defined and the language with which the client and server will communicate is specified. To establish a framework for application development, service procedure names, parameters and return argument types, and the types of data passed between client and server are defined. Data typing must conform to any limitations of the protocol compiler. The RPC Language (RPCL) read by ONC's RPCGEN protocol compiler is C-like.

Protocols have version numbers and program numbers. The version numbers FRDBVERS make it possible for multiple software generations to co-exist on the network.

Protocol Compilation (RPCGEN):

A protocol compiler is a tool that generates structured C source code, including client and server stubs, common definitions (header file(s)) and data translation filters. It consists of C like definitions of the network application, to allow the client and server applications to carry on remote procedure calls at the highest possible abstraction. RPCGEN produces a client and server stub, the necessary XDR filters and a header file to be included by client and server applications and stubs. It also generates a server dispatch function to take care of validating requests and invoking the appropriate service procedures [5]. A dispatch routine receives the request, attempts to validate and provide the service through a local procedure call on the client and sends the reply to the client [11].

Client and Server Stubs:

A *client stub* is the source code containing all the necessary functions to allow the client application to make remote procedure calls using a local procedure call model. A protocol compiler typically generates this file which gets linked with ant XDR filters and client applications [11].

A *server stub* is the source code containing all the necessary functions to allow the server applications to satisfy remote procedure requests using local procedure calls. A protocol compiler typically generates this file which gets linked with XDR filters and server application [11].

With ONC RPC, client and server parameters are passed by address. Because the service procedures are invoked by the dispatcher, they must work with pointers. To permit their encoding, the dispatcher must have a static address at which to store the results of service procedures. Clients must pass addresses to their stubs for the same reason. The stub generated at the client enables the client code to make local calls using the same names mentioned in the service procedures.

Writing Client and Server Procedures: (Ref. Figures 5.5 & 5.6 for examples)

Client Procedures are programs written in 'C' that requests remote procedure calls to be executed by a server application. Server Procedures are isolated functions performed by the server. This is a program written in 'C' to handle and reply to requests made by the client processes.

Compilation and linking of library routines and stubs generated during compilation:

Here the client and server are compiled. This is done by linking the client functions and the server functions with their respective stubs and XDR filters generated by RPCGEN. Since the XDR and RPC library functions are included in libc.a no extra linking libraries are necessary.

Processing of Client & Server Stubs

The client stub makes a service request by calling the routine *callrpc* in the client stub. The client provides the following eight pieces of information to *callrpc*:

- name of the server system
- program number
- program version
- procedure number
- an argument for the procedure call
- the address of the XDR routine that describes it
- the address of the return variable
- an XDR routine describing the return information

The server stub uses the routine *svctcp_create* to open a socket. The server then registers its functions via the routine register_rpc. The server provides the following information to the routine register_rpc:

- program number
- program version
- procedure number
- the address of the routine that processes the request
- the address of an XDR routine that receives any past argument
- the XDR routine that sends any return data

In this way, each service routine takes a single argument that is the result of the XDR input and returns the pointer to the data to be sent via XDR output. After the server has registered all its procedures, it calls the routine svc_run which loops, waiting for and processing request. This routine never returns a value.

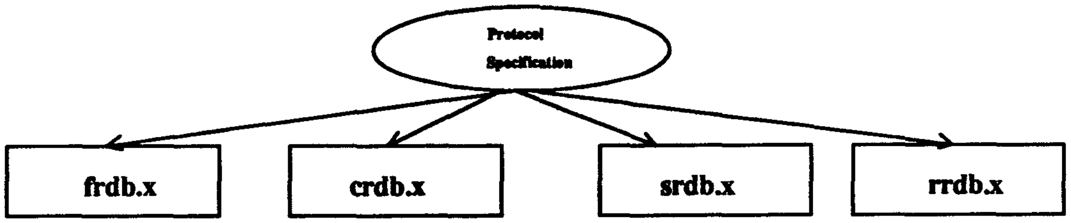


Figure 5.4: Protocol Specification Files

5.3 Our DIS Application As Seen From RPC

Six steps (as described in the previous section) involved in developing the client and server executables for our DIS application using RPC are as follows:

Protocol Definitions:

RPCL protocol definition files for remote database interface servers in this application are included in the Appendix Section 7.2 with a .x extension. These are frdb.x, crdb.x, srdb.x and rrdb.x for the faculty, course, student and register databases respectively (Ref Appendix Section 7.2). They consists of the formal grammar required to define the nature of the client/server interface, including remote procedure declarations, input and output typing, and program, version, and procedure detailing.

For example, frdb.x allows a client user to add, delete, view, and list a faculty record from a database stored on the server. Records in frdb.x are stored as white-space delimited ASCII characters, name(first, middle initial, last), phone number and location. It contains six service procedures for searching through the records by first name, last name, social security number, phone number or location. The service procedures ending with _KEY require a string argument and return a structure record. (In RPCL, a string is defined as a sequence of characters ending with a null character.) The service procedures ending with _RECORD require a record argument and return a structure record. The ADD_RECORD service procedure

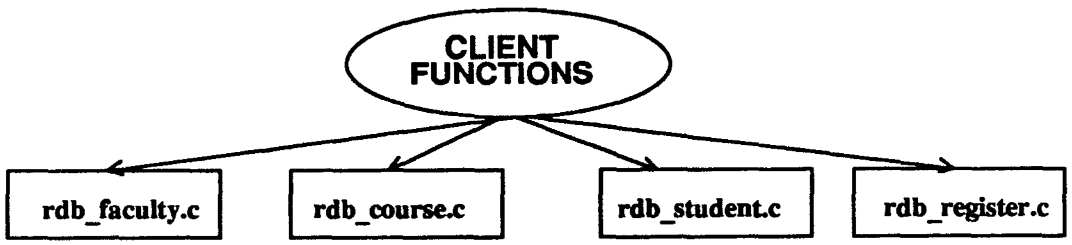


Figure 5.5: Client Functions

adds records to the database, returning a status integer. An additional procedure, number 0, (also called the null procedure) will be inserted by RPCGEN. It is used for "pinging" a server to verify that it is listening for requests. The #define DATABASE preprocessor definition is passed into the header file. The #define MAX_STR also appears there as an integer preprocessor constant, and a structure record will show up as a typedef structure definition in the header.

RPCGEN Protocol Compiler:

The Sun ONC RPC protocol compiler used here, uses C-like specification(RPCL language) for RPC applications and network data to generate code to handle low-level RPC mechanisms. The above mentioned .x files are compiled with this compiler to generate the following header files, data translation filters, and client and server stubs.

Header Files:

frdb.h, crdb.h, srdb.h, and rrdb.h (Ref Appendix Section 7.2).

Client Stubs:

frdb_clnt.c, crdb_clnt.c, srdb_clnt.c and rrdb_clnt.c (Ref Appendix Section 7.2).

Server Stubs:

frdb_svc.c, crdb_svc.c, srdb_svc.c and rrdb_svc.c (Ref Appendix Section 7.2).

Writing Client and Server Functions

Client Functions:

They consist of C code through which the server procedures can be requested.

The calling functions for faculty, course, student and register, are specified in rdb_faculty.c, rdb_course.c, rdb_student.c, and rdb_register.c files (Ref Appendix Section 7.2).

Server Functions:

These service functions are the procedures specified in protocol specification files. They consist of C code through which the replies to requests made by client processes are handled. The called functions for faculty are specified in frdb_svc_proc.c, frdb_svc_add.c, and frdb_svc_del.c (Ref Appendix Section 7.2). The called functions for course are specified in crdb_svc_proc.c, crdb_svc_add.c, and crdb_svc_del.c (Ref Appendix Section 7.2). The called functions for student are specified in srdb_svc_proc.c, srdb_svc_add.c, and srdb_svc_del.c (Ref Appendix Section 7.2). The called functions for register are specified in rrdb_svc_proc.c, rrdb_svc_add.c, and rrdb_svc_del.c (Ref Appendix Section 7.2).

Compilation and linking of library routines and stubs generated during compilation: In this application, the previously described client routines are compiled with the client stubs and XDR files. The server routines are compiled with their respective server stubs and XDR files. The client executable thus obtained is *rdb*. The server executables obtained are *frdbs, crdbs, srdbs & rrdbs.*

In this application (Ref. Fig. 5.7), the user enters a service key and value which is passed to the trader, which in turn attempts to place the appropriate remote calls. Before proceeding the client calls the server machine's portmap daemon to see if the necessary service program is registered. This sequence follows a well defined RPC pattern. If the program is registered, the ONC RPC library function clnt_create() returns a pointer to a CLIENT structure containing vital server communication information (the port address and an associated socket). A connection is created using the transports for transmission control protocol (TCP). As can be seen from the Figure 5.3, the client and server sides are created by linking their functions with the respective stubs and XDR filters generated by RPCGEN.

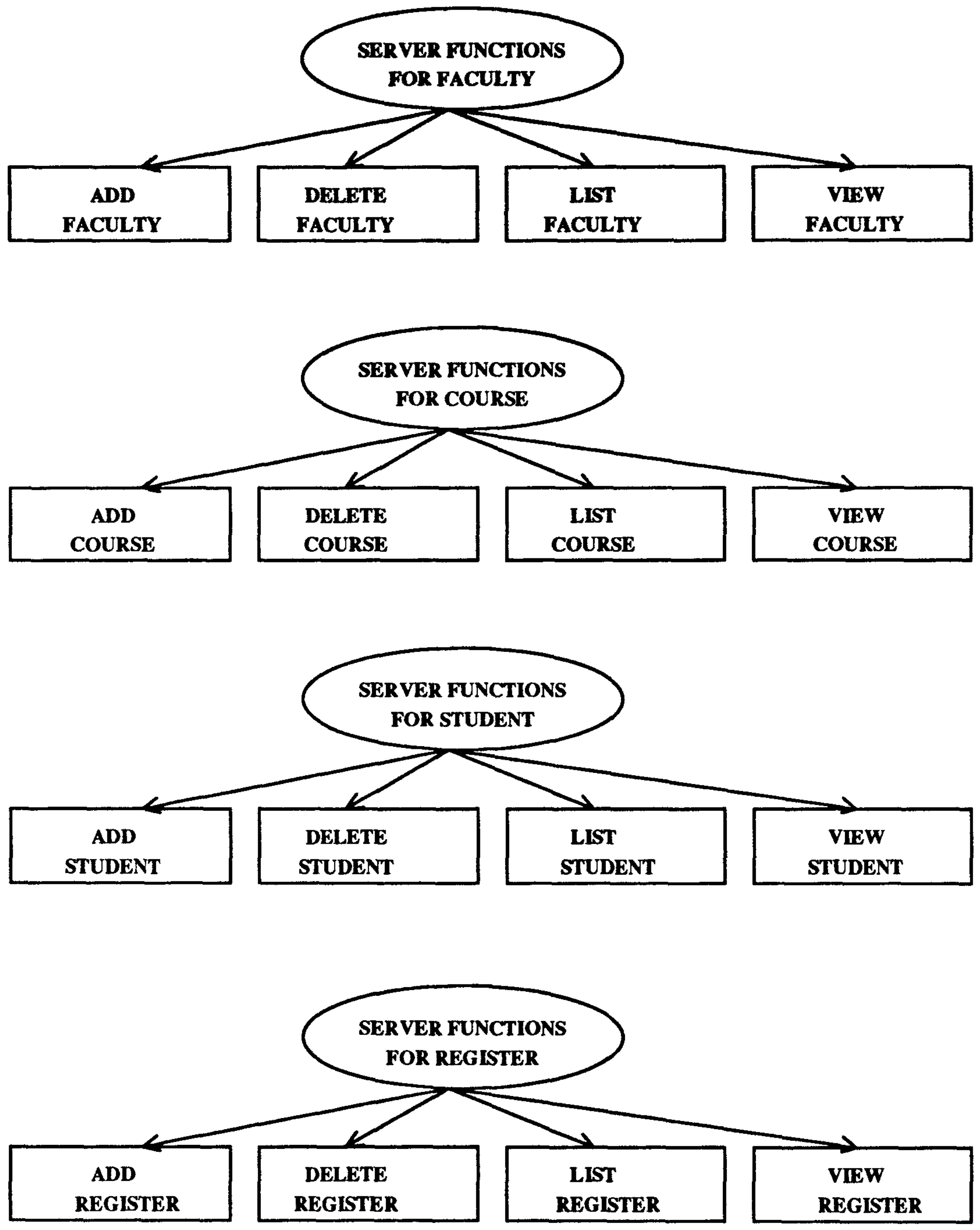


Figure 5.6: Server Functions

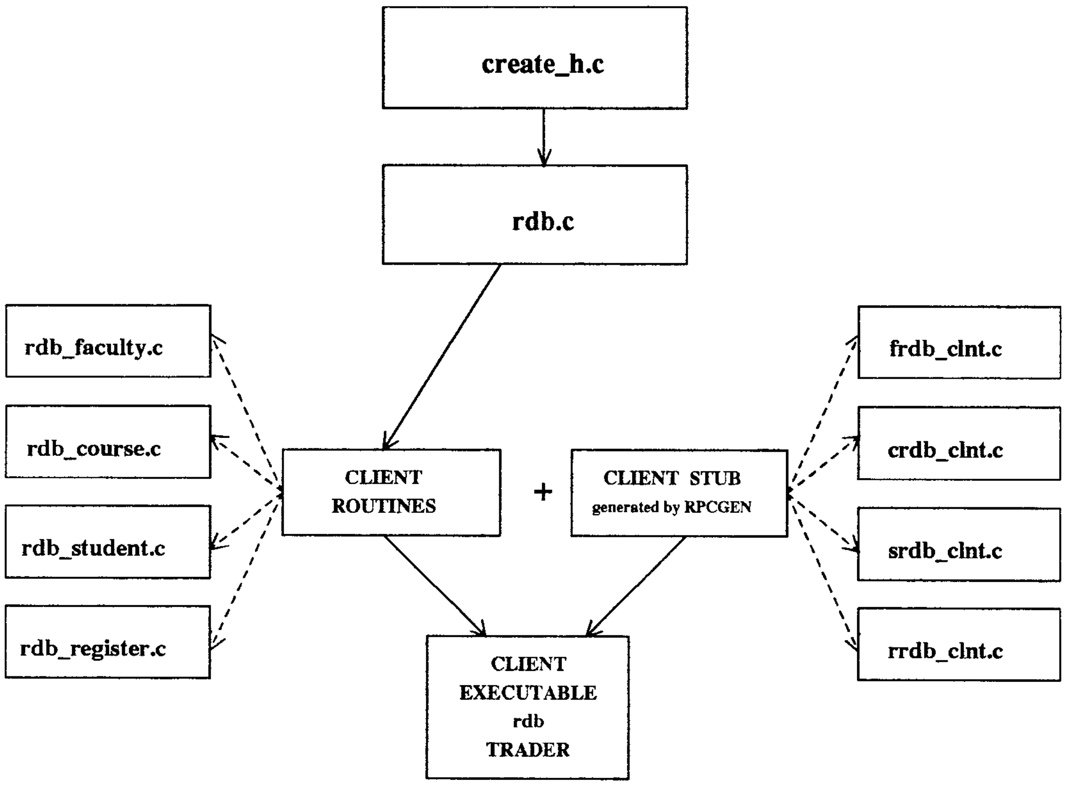


Figure 5.7: Production Process of the Trader using RPC

5.4 Production Process of the Trader using RPC

The trader provides the infrastructure required for this DIS. In this application, we have a number of servers which provide all the services requested by a user. The trader is basically the "glue" that ties together the client-server and the distributed system concept. In this way, it ties together the client routines with the server routines thus providing services to the users on a totally abstract level. In this way, the user does not need to know the underlying communication i.e., the RPC details.

The trader consists of a communication network that provides the basic connectivity and services that are required to run the application. As seen in Figure 5.7, the *create_h.c* file creates the client handles for the registered servers (Ref. Appendix Section 7.2). The *rdb.c* file accepts the messages from the user layer building blocks. These messages are passed to the *client routines.* The client routines as shown in Figure 5.7, consist of routines written in 'C', to access the server functions. The *rdb_faculty.c* contains routines to access the different functions for faculty database. The *rdb_course.c* contains routines to access the different functions for course database. The *rdb_student.c* contains routines to access the different functions for student database. The *rdb_register.c* contains routines to access the different functions for register database. (Ref. Appendix Section 7.2).
The client stubs as seen in the figure 5.7, are generated by the RPCGEN compiler as explained above. The *frdb_clnt.c* is the client stub for faculty, *crdb_clnt.c* is the client stub for course, *srdb_clnt.c* is the client stub for student and *rrdb_clnt.c* is the client stub for register. (Ref. Appendix Section 7.2). The contents of the client stubs are as described in the previous sections.
The dashed arrows imply that in our application we have four different client routines for each of the databases i.e., faculty, course, student and register, and four client stubs for each of these client routines. These are written in this fashion

to make the application modular. The solid arrows imply that all the connected files are compiled together to create the trader. The files create_h.c, rdb.c, *client routines* and *client stubs* (described earlier) are compiled to create the client executables i.e., in this case, the *trader*. This compilation is done before run time i.e., the application can run only after the trader executable is created.

CHAPTER 6

RESULTS & CONCLUSIONS

6.1 Evaluation of the DIS with regard to the Conceptual Architecture

The Conceptual Architecture applied in building this DIS is derived from the $OSCA^{TM}$ architecture. The basic ideas and concepts as understood from [1] have been used for the implementation and derivation of this prototype. As stated in the earlier chapters, this application is divided into building blocks which try to adhere to the *interoperability* principles as described in [1]. The following characteristics have been achieved for the building blocks in this prototype [1] (This is our own verification and has not been formally authorized by Bellcore):

1. *Release Independence*:

   All the building blocks are releasable, installable and upgradable without requiring the simultaneous release of any other building blocks. Hence any building block can now be replaced by an updated version without unduly affecting the rest of the software.

   New building blocks added to the system must be compatible with older building blocks. To obtain a new functionality, it may be required that new versions of several building blocks be installed over a period of time before the new functionality is available. It will not be necessary for all the installations to take place simultaneously.

   Also, if the code for a particular building block is changed, it will not affect the functionality of the other building blocks. Every building block providing a certain functionality has a version number which is defined in the protocol specification file (Ref. Chapter 5). This is in adherence to the RPCGEN (Ref.

Chapter 5) requirements. Now, this changed building block can be given a new version number and be recompiled with the rest of the system to bring the changed functionality into existence.

2. *Physical Data Base Independence*:

   Only the stewarding building blocks like the add building blocks or the delete building blocks are required to run on the same hardware as where the respective stewarded database physically exists. All the other building blocks can be physically placed on any of the workstations.

3. *No Accessibility Assumptions Between Building Blocks*:

   Some building blocks are required by other building blocks. This depends on the functionality and the user needs. However, if for whatever reason a building block becomes non available, then the building block reports its inability to perform the requested function.

4. *Logical Building Block Addressing*:

   No building block knows the physical address of any other building block. The addressing is invariant over location and environment. Building blocks identify other building blocks and their contracts by their logical addresses. The building blocks communicate with each other through the common trader. The trader (Ref. Chapter 5) keeps track of all the physical locations of the various building blocks. The communication software fabric (i.e., the trader) performs the logical to network address mapping. If for whatever reason we want to move a building block residing on one workstation to another workstation, then, the new position of this block will have to be specified in the *create_h.c* file (Ref Appendix Section 7.2) and a recompilation of the trader for this change to be done. The moved building block may need recompilation because of operating system version problems.

5. *Execution in only one Recoverable Environment:*

   A recoverable environment is one physical machine or multiple machines which are made to appear as if they are one environment from the stand point of the application. This application is made to run on two different workstations without the user being aware of this fact. Every building block in the processing and data layer exists only on one recoverable environment. If these building blocks have to be moved around, then they have to be moved as a whole unit along with the database for which these blocks provide the various functionalities.

6. *Interactions among building blocks are defined by* **contracts**:

   A building block communicates with any other building block only via the trader with the help of messages. This is done by using a specific format for the messages. There is no direct communication among building blocks. Hence it does not matter where a building block physically exists. The building blocks can be easily moved from one workstation to the other. This message passing is close to a *contract*, even though it is restricted to a syntactical level.

7. *Secure Environment:*

   Every building block has only one entry point through which only the trader can get access. In case of the user layer building blocks, the entry points are those provided for human access only.

According to the conceptual architecture guidelines [1], these building blocks are divided into three layers depending on their functionality. The *Data Layer* building blocks in this prototype exhibit the following characteristics:

1. *Provide means so that the data is updatable and readable.*

   The data layer building block contains only the update and read functionalities. The functionality enumerated by these building blocks eliminates processing and user functionality. These building blocks provide query and updation functions. These are the only blocks that can access the database directly. This protects against unknown sources damaging the data.

2. *Allow query of all the data i.e., provide list, user-interactive queries and alternate views.*

   Inquiry access to all the data is provided to the end users and to building blocks, with a query facility which interacts with the end user and formulates a well formed query. This gives the end users an easy and uniform access to the data in the databases.

3. *The distributed computing environment hosting the data layer building blocks is transparent to the user of the data.*

   A user on any workstation can access any data layer building block irrespective of whether the block is present on the particular workstation being used.

4. *Manage data redundancy.*

   A database in this application exists on any of the workstations. Hence, the same data is not present on both the workstations. Hence we do not have to deal with data redundancy.

The *User Layer* building blocks in this prototype exhibit the following characteristics:

1. *Provide access to the human user to all the other building blocks in this application through the trader.*

   This is provided by the primitive user menu (Ref. Appendix Section 7.1).

2. *Provide data entry and deletion facilities.*

   Data Entry is provided by the primitive user interface implemented (Ref. Appendix Section 7.1).

3. *Provide display processing, menu processing, list formatting, interactive text and network accessibility.*

   Display processing refers to actions like "refresh screen". This function can be performed without consulting any other building block. Hence it is performed locally by the user layer building block.

   Menu Processing refers to providing service for constructing and presenting menus. The user layer building block accepts the end user's choice and then initiates subsequent processing (Ex. Presentation of the appropriate menu screen).

   List Formatting refers to formatting the display i.e., screen design for the gathered, analyzed and summarized requested data.

   Interactive text refers to screens which are interactive and user driven. For example, for viewing a particular record, the system asks the user to enter data for a particular field like "Enter social security number". The user enters this social security number which is then processed accordingly.

   Network accessibility refers to accessing other building blocks distributed across the network, achieved through the software communication fabric i.e., the trader. Through this trader, any building block has access to the functionalities provided by any other building block in this distributed environment.

4. *Exist on all the end user's machines.*

   The user layer building blocks exist on both the workstations. This is necessary for the users on both the machines to be able to use the primitive menu interface.

The *Processing Layer* building blocks in this prototype exhibit the following characteristics:

1. *Provide functionality to crunch numbers, to answer complex queries, and control process flow.*

The processing layer building blocks provide answers to statistical queries. For example, it calculates the total number of credits taught by a particular faculty or the total number of credits taken by a student. It also answers complex queries like listing the courses taught by a particular faculty member (Ref. Chapter 4).

2. *Provide value-added services to the data that an end user requires.*

   When certain data is needed, the processing layer building block issues a request to the appropriate data layer building block, responsible for accessing the data and returns the desired result.

3. *Contain any functionality that is not expressly enumerated by the data layer or the user layer.*

   The processing layer building blocks contain functions that can not be construed as communications software fabric, data layer or user layer functionality. The processing layer therefore does not steward data, nor supports human users directly.

4. *Contain building blocks that only process services.*

   Contain certain contracts to access the data stewarded by a data layer building block to support service processing. (Ref. Chapter 4).

5. *Contain a variety of service processing layer building blocks.*

   Ex. of a basic service processing layer building block is faculty course list and advanced service processing layer building block is credits per faculty (Ref. Chapter 4).

The following characteristics of the building blocks according to the conceptual architecture [1] could not be implemented:

1. The building blocks do not have their own local data. According to [1], this data can be maintained only by the building block that owns it and no

other building block can access this data. In this application there was no need for this kind of local data, hence it was not required to implement this characteristic.

2. A secure environment could not be provided as required by [1]. According to [1], the identity of the invoking user must be maintained and passed to any other building blocks. This feature and no other security features were implemented in this prototype.

3. The integrity of the data stored in the databases is not checked in this prototype. No constraints are implemented on the data read in from the user.

6.2 Realization of the need for an Infrastructure

A conceptual architecture does not talk explicitly about the need for an infrastructure, i.e., the need for a software communication channel or a trader as a minimun. However, for a distributed application as implemented by this prototype, we conclude that a good infrastructure is essential. While implementing this application, most of the time was spent on designing and developing this infrastructure i.e., the communication channel, than the actual application. The communication channel developed for this application is the **trader**. The **trader** follows the following standards and guidelines (also essential to implement *contracts* [1]):

1. *Use of standards*:

   The trader in this case, uses the SUN's ONC RPC standards of communication as described in Chapter 5. The RPC views the application as a client server model. Hence, an interface is written for the client modules so that they emulate the software communication channel i.e., the trader. The trader is not application independent as it requires the data storage details. Hence the need for a better trader is realized by this application.

2. *Restricted set of syntax encodings*:

   The trader uses a restricted set of syntax encodings supporting the communication fabric services provided by RPC.

3. *Release Independence*:

   When the trader is changed, some changes will have to be made to the building blocks. This makes it possible to release different versions of building blocks which invoke the trader.

4. *Equality of invocation*:

   A *contract* invoked by building block **A** functions in exactly the same way as the same *contract* invoked by building block **B**.

5. *Logical addressing*:

   Contracts between building blocks are identified and invoked in a way that is independent of the physical location of the building block providing the *contract*. Each *contract* that a building block provides is uniquely identifiable and accessible with respect to all other *contracts*.

6. *Error messages*:

   Provides error messages when a particular building block is unaccessible.

6.3 Advantages of the Conceptual Architecture

There are a number of advantages of applying the described conceptual integration architecture to the DIS.

1. *Separation into different layers*:

   The separation into user, processing, data layers provides a manageable means to upgrade this DIS. For example, when new database management techniques are developed to manage the data in this kind of Distributed Environment,

the application software could be easily upgraded to take advantage of these techniques without unduly impacting other software.
Similarly, as user interface techniques are advanced, the application software could be upgraded without unnecessarily impacting other software. The application provides a simple primitive menu driven user interface. A X-window interface could be designed for this application. The existing user layer building blocks could be replaced by this new interface, without actually making any changes in the building blocks in the other layers.

2. *Relocation*:
   The whole application is implemented in a distributed networking environment. The building blocks in our application can be statically allocated in any environment, i.e., the building blocks can be physically moved around from one workstation to another. After the building blocks have been placed in their new environment, the trader is informed of the new locations of the building blocks. The whole application is then recompiled. This recompilation is necessary because different workstations can have different versions of the operating system or different architectures. After this, the background service processes are started up on their respective workstations. Now, the application can be run. The whole process of moving the blocks around does not consume much time. Also, none of the actual code is affected by this relocation.

6.4 Guidelines for better System Implementation

If a similar application has to be developed, then it would be very helpful to have the following facilities readily available and to solve the related problems:

1. *A good communication channel*:

   This is very essential in designing such a distributed application. The trader designed and implemented in this prototype is very primitive and has many restrictions. It is specifically designed for this particular prototype as information about the handled data types has to be included in the trader (Ref. Chapter 5 on RPC). The trader, hence is not very general and can not be used for any other application. If new building blocks are added to this application, then the trader has to be updated to include the data handling information of these new building blocks. Also, the availability of new functionalities provided by these building blocks has to be coded into the trader.

2. *A database*:

   The data in this prototype is stored in files. It would be good to have a database to store this data. The database would then facilitate complex SQL queries. The data retrieval and update would become much more simple and efficient. Integrity constraints can also be enforced on the data to be stored, keeping the database consistent. As it exists today, this application does not enforce any entity or referential or key integrity constraints. A database system would automatically enforce these constraints. On the other hand, the presently available database systems have a few problems:

   (a) The distributed database systems are not very compatible with each other.

   (b) Not many are easily available

   (c) The available systems are not very mature.

   (d) Each of these distributed sytems have their own way of communication in a distributed environment. Each of these systems can operate using only the available internal communication. For integrating these individual systems together, the need arises for a common communication platform, which is not available as of today.

3. *An integrable user interface*:

   The prototype provides a simple menu driven user interface. A X/Motif interface was developed but could not be integrated with the application due to integration problems between X/Motif and RPC. Integrating RPC and windowing systems has its challenges.[11] The X protocol supports its own messaging system on top of sockets. It was designed with graphics and windowing in mind, and does not represent a general purpose, high-level, remote execution environment like RPC. In order to mix RPC and X, you have to reach down to low-level RPC and IPC programming. There are several strategies available for using RPC under X. Some of them are as stated below:

   (a) Placing RPCs into the callback routines to be performed synchronously in response to some user action.

   (b) Using a timer to do the same as (1).

   (c) Placing an RPC in a callback, returning immediately to look for the return FRPC with a timer.

   (d) Performing asynchronous servicing of RPCs using event or socket-watching functions built into X toolkits.

   These mentioned solutions could not be implemented due to time constraints. For further details to implement the above mentioned strategies, refer Chapter 10 of [11].

   From the above three points it can be concluded that the basic need to implement an application using the standards of a conceptual integration architecture is a good and complete infrastructure. A good infrastructure consists of a stable and general communication fabric, a database that will work in coherence with this fabric and also a good user interface that will be compatible with both the software fabric and the database.

4. *Configuration Management*:

   In a production environment to run this kind of DIS application, we would need a configuration file to start up the background processes, i.e., the server processes (frdbs, crdbs, srdbs and rrdbs). This can be implemented by using simple shell scripts. Also, we could keep changing the physical positions of the building blocks. This can be done by moving a building block from one workstation to another. However this can not be done at run-time. After moving the building block, the trader has to be informed of its new location and then it has to be recompiled. Note that the data layer building blocks for addition and deletion of records, have to physically exist on the machine where the particular databases exist.

6.5 Conclusions

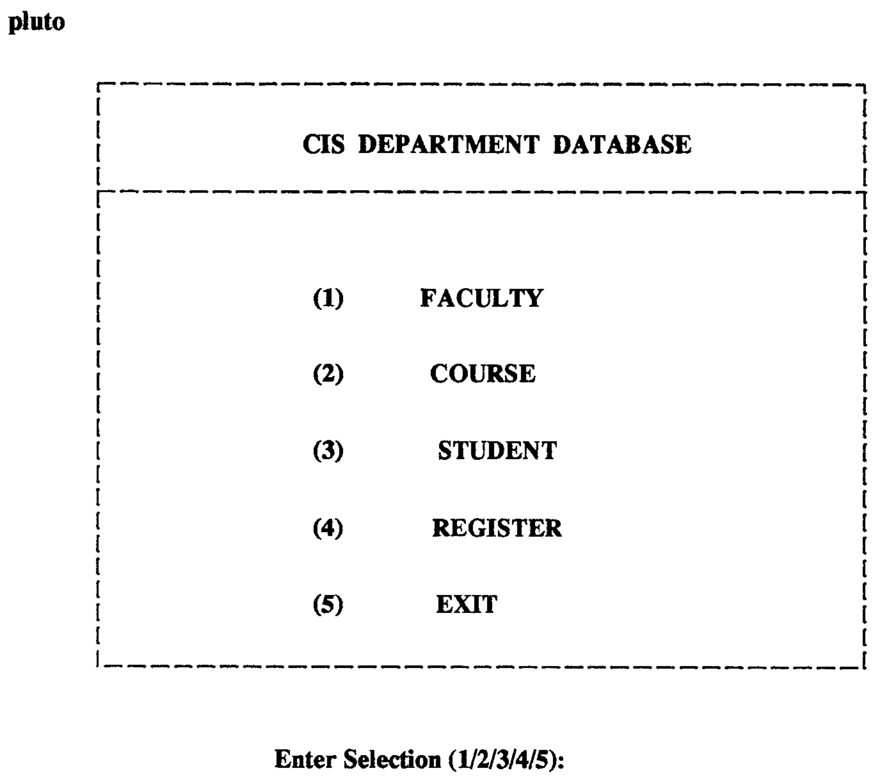
The use of a conceptual integration architecture increased the modularity of the whole application. As building blocks can be easily moved, it makes this system more portable and adaptable. Additional functionality can be easily introduced in the system in the form of additional building blocks without unduly impacting the rest of the system and it is relatively easy to change the current configuration.

However, implementing this prototype also brought forth the need for a complete infrastructure to support the implementation of a conceptual architecture. As a result of this, it was concluded that the *infrastructure* is a vital part of any systems integration architecture.

CHAPTER 7

APPENDIX

7.1 The Primitive User Interface

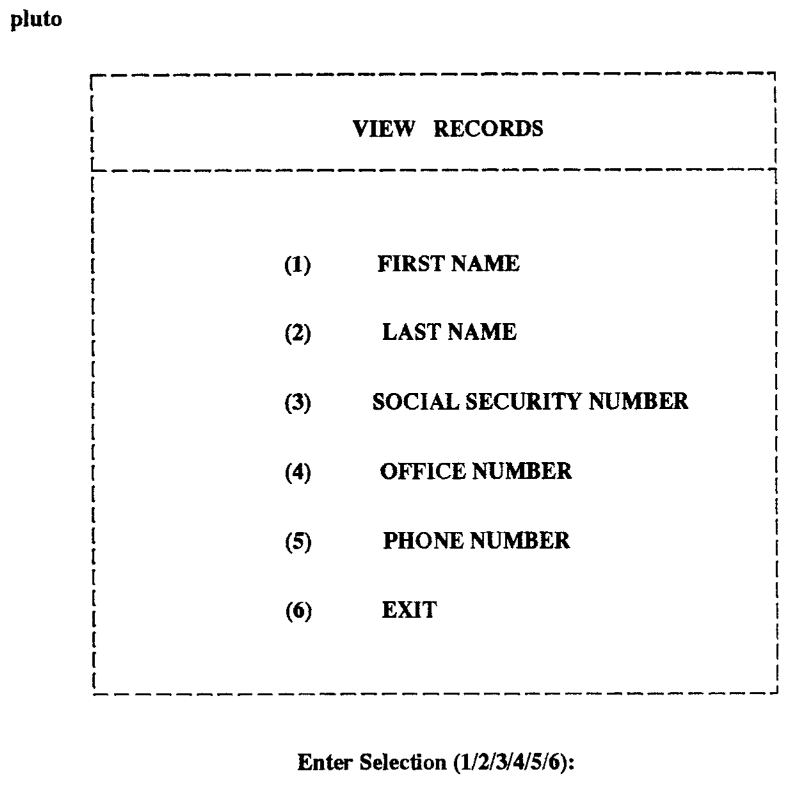


```
pluto

 ----------------------------------------
|                                        |
|          CIS DEPARTMENT DATABASE       |
|                                        |
 ----------------------------------------
|                                        |
|                                        |
|                                        |
|          (1)      FACULTY              |
|                                        |
|          (2)      COURSE               |
|                                        |
|          (3)      STUDENT              |
|                                        |
|          (4)      REGISTER             |
|                                        |
|          (5)      EXIT                 |
|                                        |
 ----------------------------------------

          Enter Selection (1/2/3/4/5):
```

Figure 7.1: Main Menu

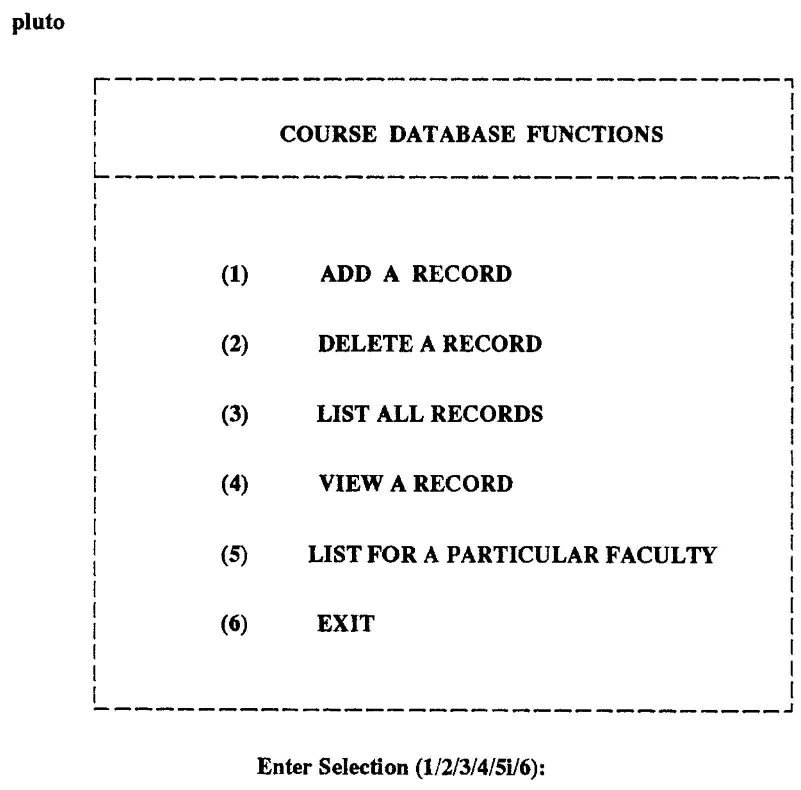
```
pluto

    ┌-----------------------------------------┐
    |                                         |
    |        FACULTY DATABASE FUNCTIONS       |
    |                                         |
    ┌-----------------------------------------┐
    |                                         |
    |                                         |
    |         (1)      ADD A RECORD           |
    |                                         |
    |         (2)      DELETE A RECORD        |
    |                                         |
    |         (3)      LIST ALL RECORDS       |
    |                                         |
    |         (4)      VIEW A RECORD          |
    |                                         |
    |         (5)      EXIT                   |
    |                                         |
    |_________________________________________|

              Enter Selection (1/2/3/4/5):
```

Figure 7.2: Faculty Menu



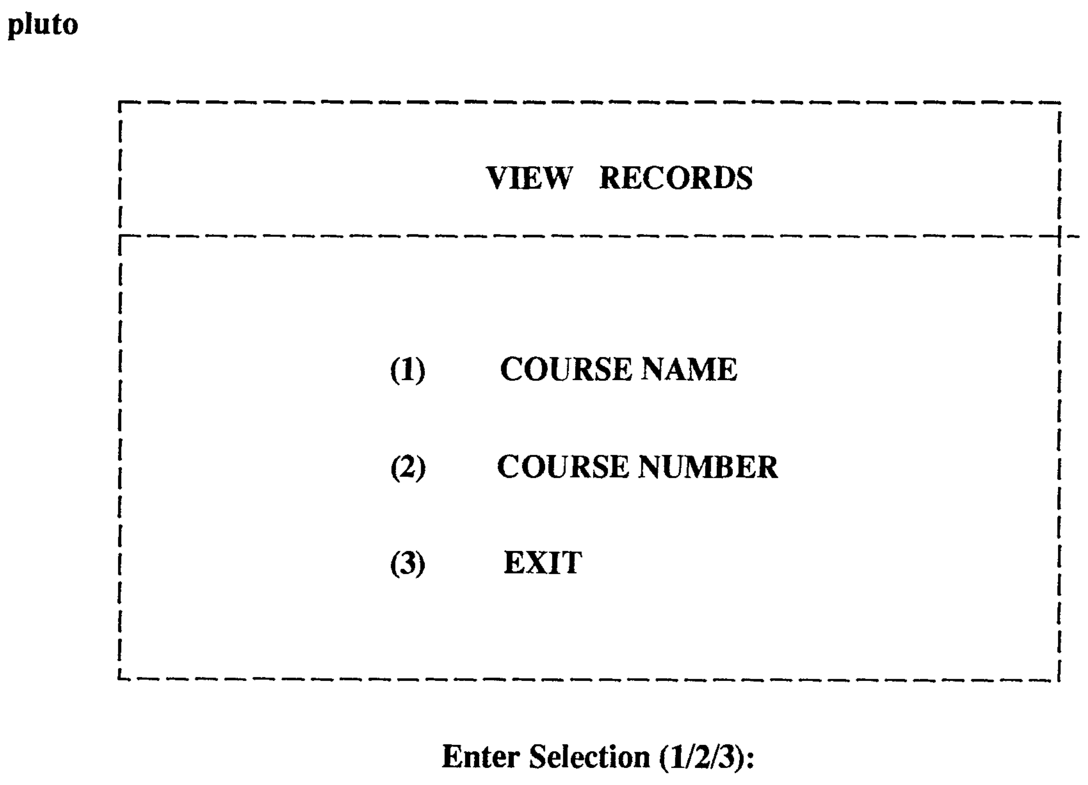
```
pluto

              VIEW   RECORDS

      (1)      FIRST NAME

      (2)      LAST NAME

      (3)      SOCIAL SECURITY NUMBER

      (4)      OFFICE NUMBER

      (5)      PHONE NUMBER

      (6)      EXIT

      Enter Selection (1/2/3/4/5/6):
```

Figure 7.3: Faculty View Menu



```
pluto

 ----------------------------------------
|                                        |
|       COURSE  DATABASE  FUNCTIONS      |
|                                        |
 ----------------------------------------
|                                        |
|                                        |
|     (1)      ADD  A  RECORD            |
|                                        |
|     (2)      DELETE A RECORD           |
|                                        |
|     (3)      LIST ALL RECORDS          |
|                                        |
|     (4)      VIEW A RECORD             |
|                                        |
|     (5)     LIST FOR A PARTICULAR FACULTY |
|                                        |
|     (6)      EXIT                      |
|                                        |
 ----------------------------------------

        Enter Selection (1/2/3/4/5l/6):
```

Figure 7.4: Course Menu

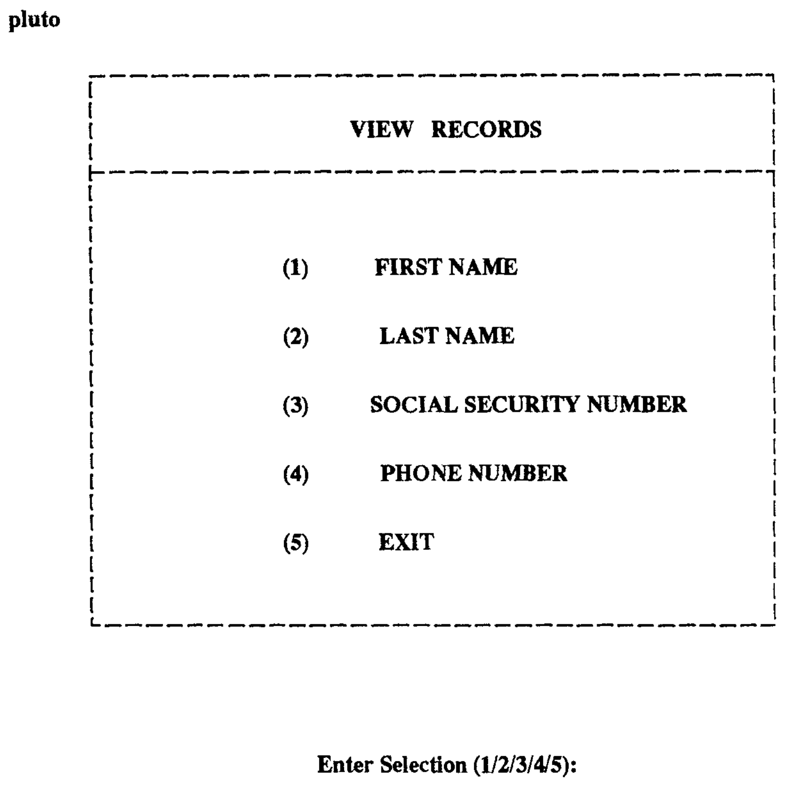


```
pluto

                VIEW   RECORDS

        (1)     COURSE NAME

        (2)     COURSE NUMBER

        (3)     EXIT

            Enter Selection (1/2/3):
```

Figure 7.5: Course View Menu

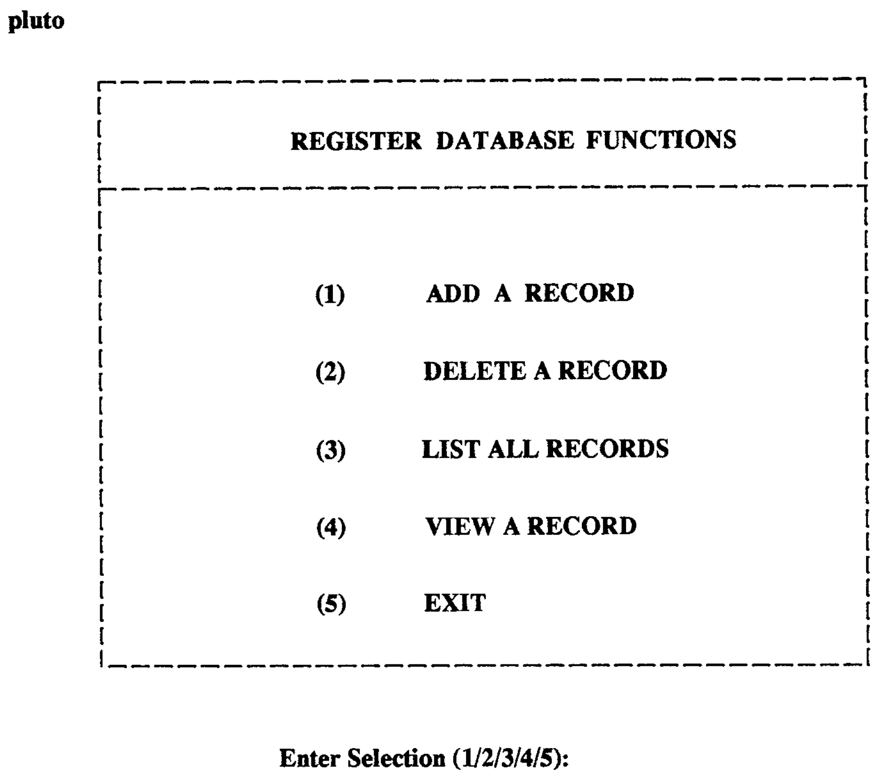
```
pluto

      ---------------------------------------
      |                                     |
      |     STUDENT DATABASE FUNCTIONS      |
      |                                     |
      ---------------------------------------
      |                                     |
      |                                     |
      |        (1)     ADD A RECORD         |
      |                                     |
      |        (2)     DELETE A RECORD      |
      |                                     |
      |        (3)     LIST ALL RECORDS     |
      |                                     |
      |        (4)     VIEW A RECORD        |
      |                                     |
      |        (5)     EXIT                 |
      |                                     |
      ---------------------------------------

           Enter Selection (1/2/3/4/5):
```

Figure 7.6: Student Menu



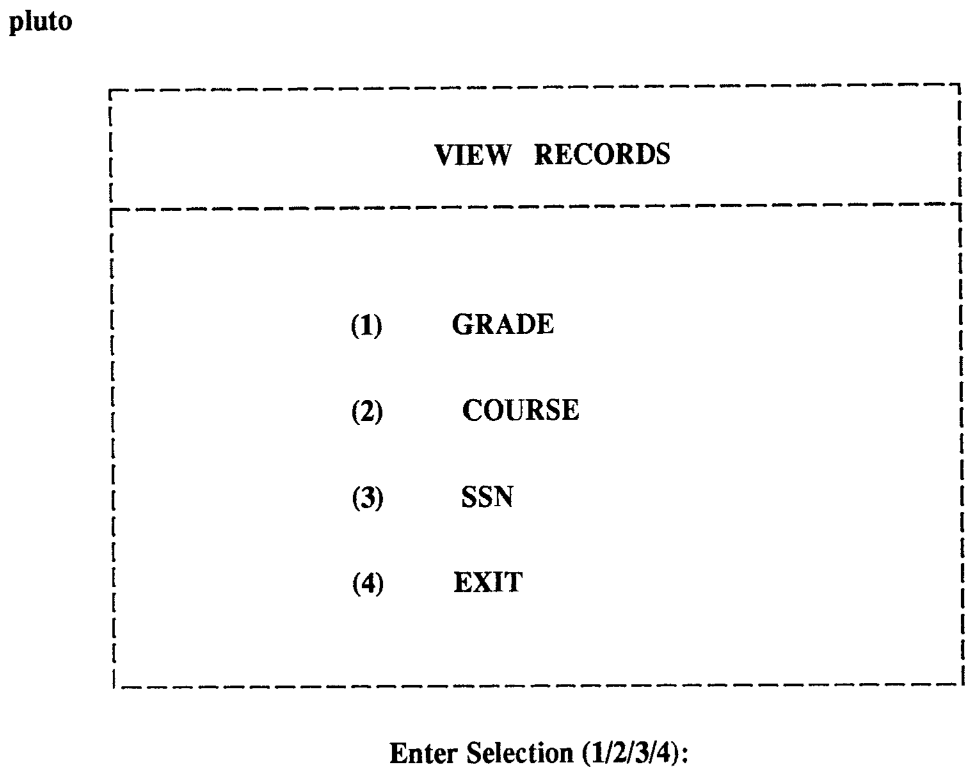
```
pluto

 ----------------------------------------
|                                        |
|             VIEW   RECORDS             |
|                                        |
|----------------------------------------|
|                                        |
|                                        |
|        (1)      FIRST NAME             |
|                                        |
|        (2)      LAST NAME              |
|                                        |
|        (3)      SOCIAL SECURITY NUMBER |
|                                        |
|        (4)      PHONE NUMBER           |
|                                        |
|        (5)      EXIT                   |
|                                        |
 ----------------------------------------


             Enter Selection (1/2/3/4/5):
```

Figure 7.7: Student View Menu



```
pluto

 ------------------------------------------
|                                          |
|      REGISTER DATABASE FUNCTIONS         |
|                                          |
 ------------------------------------------
|                                          |
|                                          |
|       (1)      ADD A RECORD              |
|                                          |
|       (2)      DELETE A RECORD           |
|                                          |
|       (3)      LIST ALL RECORDS          |
|                                          |
|       (4)      VIEW A RECORD             |
|                                          |
|       (5)      EXIT                      |
|                                          |
 ------------------------------------------

        Enter Selection (1/2/3/4/5):
```

Figure 7.8: Register Menu



```
pluto

            VIEW   RECORDS

      (1)      GRADE

      (2)      COURSE

      (3)      SSN

      (4)      EXIT

      Enter Selection (1/2/3/4):
```

Figure 7.9: Register View Menu

7.2 Source Code for BUILDING BLOCKS

7.2.1 Makefile

```
CC= cc

LD= cc

CFLAGS=-g -I.$(CCDEBUG)

LDFLAGS= $(LDDEBUG)



FACDIR = ../faculty/

COURSEDIR = ../course/

STUDDIR = ../student/

REGDIR = ../register/


.o:.c


RDBOBJS = main_menu.o fmenu.o fquery.o farecord.o ffirst_record.o \
  flast_record.o fssn_record.o foff_record.o fphone_record.o \
  cmenu.o cquery.o carecord.o cname_record.o cnumber_record.o \
  smenu.o squery.o sarecord.o sfirst_record.o slast_record.o \
  sssn_record.o sphone_record.o rmenu.o rquery.o rarecord.o \
  rssn_record.o rcourse_record.o rgrade_record.o create_h.o \
    rdb1.o rdb_faculty.o rdb_course.o rdb_student.o rdb_register.o \
  frdb_clnt.o crdb_clnt.o srdb_clnt.o rrdb_clnt.o frdb_xdr.o \
  crdb_xdr.o srdb_xdr.o rrdb_xdr.o ffun.o cfun.o sfun.o rfun.o


FRDBSOBJS= ${FACDIR}frdb_svc_add.o ${FACDIR}frdb_svc_proc.o \
   ${FACDIR}frdb_svc_del.o frdb_svc.o frdb_xdr.o


FDUMMYSOBJS= frdb_svc_add.o frdb_xdr.o frdb_svc_proc.o \
             frdb_svc_del.o frdb_svc.o


CRDBSOBJS= ${COURSEDIR}crdb_svc_proc.o ${COURSEDIR}crdb_svc_add.o \
   ${COURSEDIR}crdb_svc_del.o ${COURSEDIR}facourse_list.o \
   crdb_svc.o crdb_xdr.o

CDUMMYSOBJS= crdb_svc_proc.o crdb_svc_add.o crdb_svc_del.o \
             facourse_list.o crdb_svc.o crdb_xdr.o

SRDBSOBJS= ${STUDDIR}srdb_svc_proc.o ${STUDDIR}srdb_svc_add.o \
   ${STUDDIR}srdb_svc_del.o srdb_svc.o srdb_xdr.o

SDUMMYSOBJS= srdb_svc_proc.o srdb_svc_add.o srdb_svc_del.o\
             srdb_svc.o srdb_xdr.o

RRDBSOBJS= ${REGDIR}rrdb_svc_proc.o ${REGDIR}rrdb_svc_add.o \
   ${REGDIR}rrdb_svc_del.o srdb_svc.o srdb_xdr.o

RDUMMYSOBJS= rrdb_svc_proc.o rrdb_svc_add.o rrdb_svc_del.o\
             srdb_svc.o srdb_xdr.o

# These object files depend on frdb.h:
FXDOBJS= ffun.o rdb.o frdb_svc.o frdb_xdr.o frdb_svc_proc.o \
 frdb_svc_add.o frdb_svc_del.o crdb_svc_proc.c

# These object files depend on crdb.h:
CXDOBJS= cfun.o rdb.o crdb_svc.o crdb_xdr.o crdb_svc_proc.o \
 crdb_svc_add.o crdb_svc_del.o frdb_svc_proc.o

# These object files depend on srdb.h:
SXDOBJS= sfun.o rdb.o srdb_svc.o srdb_xdr.o srdb_svc_proc.o\
         srdb_svc_add.o srdb_svc_del.o rrdb_svc_proc.o
```

```
# These object files depend on rrdb.h:
RXDOBJS= rfun.o rdb.o rrdb_svc.o rrdb_xdr.o rrdb_svc_proc.o\
         rrdb_svc_add.o rrdb_svc_del.o sdb_svc_proc.o

# These source files are produced by rpcgen:
FGENR= frdb.h frdb_clnt.c frdb_svc.c frdb_xdr.c

CGENR= crdb.h crdb_clnt.c crdb_svc.c crdb_xdr.c

SGENR= srdb.h srdb_clnt.c srdb_svc.c srdb_xdr.c

RGENR= rrdb.h rrdb_clnt.c rrdb_svc.c rrdb_xdr.c

all1: rdb frdbs crdbs

all2: rdb srdbs rrdbs

# rdb client
rdb: $(RDBOBJS)
$(LD) -o rdb $(CFLAGS) $(RDBOBJS)

# rdb server for faculty.
frdbs: $(FRDBSOBJS)
$(LD) -o frdbs $(CFLAGS) $(FDUMMYSOBJS)

# rdb server for course.
crdbs: $(CRDBSOBJS)
$(LD) -o crdbs $(CFLAGS) $(CDUMMYSOBJS)

# rdb server for student.
```

```
srdbs: $(SRDBSOBJS)
$(CC) -o srdbs $(CFLAGS) $(SDUMMYSOBJS)



# rdb server for register.
rrdbs: $(RRDBSOBJS)
$(CC) -o rrdbs $(CFLAGS) $(RDUMMYSOBJS)

$(XOBJS): frdb.h crdb.h srdb.h rrdb.h

$(FGENR): frdb.x
rpcgen frdb.x
$(CGENR): crdb.x
rpcgen crdb.x
$(SGENR): srdb.x
rpcgen srdb.x
$(RGENR): rrdb.x
rpcgen rrdb.x

clean1:
rm -f $(RDBOBJS) $(FRDBSOBJS) $(CRDBSOBJS)

clean2:
rm -f $(RDBOBJS) $(SRDBSOBJS) $(RRDBSOBJS)
```

7.2.2 User Layer Building Blocks

```
/*main_menu.c : Main Menu for CIS  Database Services */

#include<stdio.h>
#include<ctype.h>

char bufdata[256];
int  key;
int  DEBUG = 0;

extern void fmenu();
extern void cmenu();
extern void smenu();
extern void rmenu();
extern void create_handle();

main()
{
  int i;
  int j;

 create_handle();
 while(j != 5)
 {
  system("clear");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  system("hostname");



  printf("            -----------------------------------
---------------------\n");
  printf("      |          CIS       DEPARTMENT
 DATABASE  |\n");
  printf("       ----------------------------------
----------------------\n");
  printf("      |
            |\n");
  printf("      |            (1)  FACULTY
            |\n");
  printf("      |
            |\n");
  printf("      |            (2)  COURSE
            |\n");
  printf("      |
            |\n");
  printf("      |            (3)  STUDENT
            |\n");
  printf("      |
             |\n");
  printf("      |            (4)  REGISTER
             |\n");
  printf("      |
                |\n");
  printf("      |            (5)  EXIT
                        |\n");
  printf("       --------------------------------
-------------------------\n");


  printf("\n");
  printf("\n");
  printf("\n");
  printf("                              Enter
 Selection (1/2/3/4/5): ");
  scanf("%d",&i);


  switch(i)  {


  case 1:  system("clear");
               j = 1;
               fmenu();
break;


  case 2:  system("clear");
               j= 2;
               cmenu();
break;


  case 3: system("clear");
               j = 3;
               smenu();
break;


  case 4:  system("clear");
```

```
                j = 4;
                rmenu();
break;

  case 5:  system("clear");
                j = 5;
printf("Exiting.....\n");
             break;

  default :  system("clear");
     printf("invalid input \n");
     break;

}/*end of while*/
} /* end of case - switch */
} /* end of main */
```

```
/*fmenu.c : Menu for Faculty Database Services */

#include<stdio.h>
#include<ctype.h>

extern void enter_record();

fmenu()
{
 extern int key;
 int i;
 int j;

while(j != 5)
{
  system("clear");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  system("hostname");


  printf("           ----------------------------------
----------------------\n");
```

```
  printf("      |                FACULTY   DATABASE
   FUNCTIONS           |\n");
  printf("       ----------------------------------
----------------------\n");
  printf("      |
                       |\n");
  printf("      |            (1)  ADD    A RECORD
                       |\n");
  printf("      |
                       |\n");
/******
  printf("      |            (2)  MODIFY A RECORD
                     |\n");
  printf("      |
                      |\n");
****/
  printf("      |            (2)  DELETE A RECORD
                      |\n");
  printf("      |
                      |\n");
  printf("      |            (3)  LIST   ALL RECORDS
                      |\n");
  printf("      |
                      |\n");
  printf("      |            (4)  VIEW   A RECORD
                        |\n");
  printf("      |
                     |\n");
  printf("      |            (5)  EXIT
                       |\n");
  printf("       -----------------------------------
---------------------\n");


  printf("\n");
  printf("\n");
  printf("\n");
  printf("                           Enter
 Selection (1/2/3/4/5): ");
  scanf("%d",&i);

  switch(i)  {


  case 1:  system("clear");
                j = 1;
                key = 7;
                fenter_record();
break;
/********
  case 2:  system("clear");
                j = 2;
printf("Modifying a record.....\n");
update_record(cl);
break;
**********/
  case 2:  system("clear");
                j = 2;
                key = 8;
fssn_record();
break;
```

```
  case 3: system("clear");
                j = 3;
printf("Listing all records.....\n");
                key = 6;
trader();
break;

  case 4:  system("clear");
                j = 4;
            fquery_record();
break;

  case 5:  system("clear");
                j = 5;
             break;

  default :  system("clear");
     printf("invalid input \n");
     break;

} /* end of case - switch */
}/*end of while*/
} /* end of main */
```

```
/*farecord.c : Data Entry Screen for

               Faculty Database Services */


#include<stdio.h>

#include<ctype.h>

#include<rpc/rpc.h>

#include<frdb.h>



fenter_record()

{

char ch;

record  *pr;

int l =0;

int flag;

extern char bufdata[256];


pr   = (record *) malloc(sizeof(record));

pr->ssn   = (char *)malloc(MAX_STR);

pr->firstName     = (char *)malloc(MAX_STR);

pr->middleInitial = (char *)malloc(MAX_STR);

pr->lastName      = (char *)malloc(MAX_STR);

pr->phone         = (char *)malloc(MAX_STR);

pr->location   = (char *)malloc(MAX_STR);


printf("Enter ssn number :");

scanf("%s",pr->ssn);

getchar();

        sprintf(&bufdata[l],"%s ",pr->ssn);
```

```
 /***** A blank space is placed after each field
- to separate the fields, hence "%s " ********/
 /***** To move the position in the buffer to place
        text pertaining to the next field ********/
        l = strlen(bufdata);
if (0)
 printf("Data Retrieved: %s\n",bufdata);

printf("Enter first name :");
scanf("%s",pr->firstName);
getchar();
        /*sprintf(&bufdata[l],"%s ",pr->firstName);*/
strcat(bufdata , pr->firstName);
strcat(bufdata , " ");
        l = strlen(bufdata);
if (0)
 printf("Data Retrieved: %s\n",bufdata);

printf("Enter middle initial :");
scanf("%s",pr->middleInitial);
getchar();
/******  sprintf(&bufdata[l],"%s ",pr->middleInitial);*******/
strcat(bufdata , pr->middleInitial);
strcat(bufdata , " ");
        l = strlen(bufdata);
if (0)
 printf("Data Retrieved: %s\n",bufdata);

printf("Enter last name :");
scanf("%s",pr->lastName);
```

```
getchar();
        /****sprintf(&bufdata[l],"%s ",pr->lastName);****/
strcat(bufdata , pr->lastName);
strcat(bufdata , " ");
        l = strlen(bufdata);
if (0)
 printf("Data Retrieved: %s\n",bufdata);



printf("Enter phone number :");
scanf("%s",pr->phone);
getchar();
        /***sprintf(&bufdata[l],"%s ",pr->phone);***/
strcat(bufdata , pr->phone);
strcat(bufdata , " ");
if (0)
 printf("Data Retrieved: %s\n",bufdata);

printf("Enter Office No. :");
scanf("%s",pr->location);
getchar();
/***        sprintf(&bufdata[l],"%s ",pr->location);****/
strcat(bufdata , pr->location);
        l = strlen(bufdata);

if (0)
 printf("Data Retrieved: %s\n",bufdata);
 trader();
}
```

```
/*fquery.c : Menu for Faculty Database Services */


#include<stdio.h>

#include<ctype.h>


fquery_record()

{

 int i;

 int j;

 extern int key;


 while(j != 6)

 {

  system("clear");

  printf("\n");

  printf("\n");

  printf("\n");

  printf("\n");

  printf("\n");

  printf("\n");

  printf("\n");

  printf("\n");

  printf("\n");

  system("hostname");



  printf("            ----------------------------

----------------------------\n");

  printf("      |                VIEW

 RECORDS                    |\n");
```

```
  printf("      -----------------------------
---------------------------\n");
  printf("      |
                   |\n");
  printf("      |          (1)  FIRST NAME
                   |\n");
  printf("      |
                     |\n");
  printf("      |          (2)  LAST NAME
                    |\n");
  printf("      |
                    |\n");
  printf("      |          (3)  SOCIAL
SECURITY NUMBER               |\n");
  printf("      |
                       |\n");
  printf("      |          (4)  OFFICE NUMBER
                   |\n");
  printf("      |
                    |\n");
  printf("      |          (5)  PHONE NUMBER
                    |\n");
  printf("      |
                    |\n");
  printf("      |          (6)  EXIT
                     |\n");
  printf("      ------------------------------
--------------------------\n");


  printf("\n");
  printf("\n");
  printf("\n");
  printf("                                  Enter
 Selection (1/2/3/4/5/6): ");
  scanf("%d",&i);

  switch(i)  {

  case 1:  system("clear");
                j = 1;
                key = 1;
                ffirst_record();
break;

  case 2:  system("clear");
                j = 2;
                key = 2;
flast_record();
break;

  case 3:  system("clear");
                j = 3;
                key = 5;
fssn_record();
break;

  case 4: system("clear");
                j = 4;
                key = 4;
foff_record();
break;


  case 5:  system("clear");
               j = 5;
               key = 3;
           fphone_record();
break;


  case 6:  system("clear");
               j = 6;
            break;


  default :  system("clear");
     printf("invalid input \n");
     break;


} /* end of case - switch */
}/*end of while*/
} /* end of main */
```

```
/*fssn_record.c : View Screen for

                  Faculty Database Services */


#include<stdio.h>

#include<ctype.h>

#include<rpc/rpc.h>

#include<frdb.h>


void fssn_record()

{

char ch;

record  *pr;

int l=0;

extern char bufdata[256];

pr = (record *) malloc(sizeof(record));

pr->ssn = (char *)malloc(MAX_STR);


printf("Enter ssn number :");

scanf("%s",pr->ssn);

        sprintf(&bufdata[l],"%s",pr->ssn);


if (0)

 printf("Data Retrieved: %s\n",bufdata);

 trader();

}
```

```
/*ffirst_record.c : View by firstname  Screen
                    for Faculty Database Services */

#include<stdio.h>
#include<ctype.h>
#include<rpc/rpc.h>
#include<frdb.h>

ffirst_record()
{
char ch;
record  *pr;
int l = 0;
extern char bufdata[256];
pr = (record *) malloc(sizeof(record));
pr->firstName = (char *)malloc(MAX_STR);

printf("Enter first name :");
scanf("%s",pr->firstName);
        sprintf(&bufdata[l],"%s",pr->firstName);

if (0)
 printf("Data Retrieved: %s\n",bufdata);
 trader();
}
```

```
/*flast_record.c : View by lastname Screen

                   for Faculty Database Services */


#include<stdio.h>

#include<ctype.h>

#include<rpc/rpc.h>

#include<frdb.h>


void flast_record()

{

char ch;

record  *pr;

int l = 0;

extern char bufdata[256];

pr = (record *) malloc(sizeof(record));

pr->lastName = (char *)malloc(MAX_STR);


printf("Enter last name :");

scanf("%s",pr->lastName);

        sprintf(&bufdata[l],"%s",pr->lastName);


if (0)

 printf("Data Retrieved: %s\n",bufdata);

 trader();

}
```

```
/*fphone_record.c : View  Screen for
                    Faculty Database Services */

#include<stdio.h>
#include<ctype.h>
#include<rpc/rpc.h>
#include<frdb.h>

fphone_record()
{
char ch;
record  *pr;
int l =0;
extern char bufdata[256];
pr = (record *) malloc(sizeof(record));
pr->phone = (char *)malloc(MAX_STR);

printf("Enter phone number :");
scanf("%s",pr->phone);
        sprintf(&bufdata[l],"%s",pr->phone);

if (0)
 printf("Data Retrieved: %s\n",bufdata);
 trader();
}
```

```
/*foof_record.c : View Screen for
                  Faculty Database Services */

#include<stdio.h>
#include<ctype.h>
#include<rpc/rpc.h>
#include<frdb.h>

void foff_record()
{
char ch;
record  *pr;
int l = 0;
extern char bufdata[256];
pr = (record *) malloc(sizeof(record));
pr->location = (char *)malloc(MAX_STR);

printf("Enter location :");
scanf("%s",pr->location);
        sprintf(&bufdata[l],"%s",pr->location);

if (0)
 printf("Data Retrieved: %s\n",bufdata);
 trader();
}
```

```
/***************** GENERAL NOTATIONS  **********************

where ever following character(s) are used in

variable/procedure name


      C,_c     :  used for COURSE DATA BASE


      F,_f     :  used for FACULTY DATA BASE


      S,_s     :  used for STUDENT DATA BASE


      R,_r     :  used for REGISTER DATA BASE


*************************************************************/
#include <stdio.h>
#include <ctype.h>
#include <rpc/rpc.h>
#include "frdb.h"
void copy_frecord();
void print_flist();
void print_frecord1();
void print_frecord();
int print_fdrecord();
int insert_frecord();
char c;

/***********************************************************
Functions copy, print list/record, insert for
*** Faculty Database ****
```

```
*************************************************************/
/*
  insert_frecord : insert record "rec" into the linked list
                   whose header is given by "head"
*/
insert_frecord(head,rec)
record *head,*rec;
{
record *temp;
temp = (record *)malloc(sizeof(record));
if(temp == NULL)
{
printf("insufficient memory : insert_record \n");
return(0);
}
temp->ssn = (char *)malloc(MAX_STR);
temp->firstName = (char *)malloc(MAX_STR);
temp->middleInitial = (char *)malloc(MAX_STR);
temp->lastName = (char *)malloc(MAX_STR);
temp->location = (char *)malloc(MAX_STR);
temp->phone = (char *)malloc(12*sizeof(char));
copy_frecord(temp,rec);
temp->next_record = head->next_record;
head->next_record = temp;
return(1);
}
/*
   copy_frecord : copy record "src" into "dest"
*/
void copy_frecord(dest,src)
```

```
record *src,*dest;
{
strcpy(dest->ssn,src->ssn);
strcpy(dest->firstName,src->firstName);
strcpy(dest->middleInitial,src->middleInitial);
strcpy(dest->lastName,src->lastName);
strcpy(dest->location,src->location);
strcpy(dest->phone,src->phone);
}
/*
   print_flist : print list starting at "first"
*/
void print_flist(first)
record *first;
{
record *temp;
        system("clear");
temp = first;
printf("ssn \t first \tmiddle \tlast
                \t phone \t office no. \n");
        printf("\n");
        printf("\n");
        printf("\n");
while(temp)
{
print_frecord1(temp);
temp = temp->next_record;
}
c = getchar();
printf("Press Any Key To Continue....\n");
c = getchar();
}
/*
   print_frecord1 : print individul items of record "rec"
*/
void print_frecord1(rec)
record *rec;
{
printf("%s  %s  %s  %s  %s  %s \n",rec->ssn,
              rec->firstName, rec->middleInitial,
              rec->lastName,rec->phone,rec->location);
}


/*
   print_frecord : print individul items of record "rec"
*/
void print_frecord(rec)
record *rec;
{
        system("clear");
printf("SSN            %s \n",rec->ssn);
printf("First Name     %s \n",rec->firstName);
printf("Middle Initial %s \n",rec->middleInitial);
printf("Last Name      %s \n",rec->lastName);
printf("Phone          %s \n",rec->phone);
printf("Office         %s \n",rec->location);
c = getchar();
printf("Press Any Key To Continue....\n");
c = getchar();
}
```

```
/*
   print_frecord : print individul items of record "rec"
*/
print_fdrecord(rec)
record *rec;
{
 char ch='n';

   print_frecord(rec);
   printf("\n");
   printf("\n");
   printf("\n");
   c = getchar();
   printf("Delete This Record (Y/N) ? :");
   scanf("%c",&ch);
   if ((ch == 'Y') || (ch == 'y'))
      return(1);
   else
      return(0);
}


/*
   freeF_record : free memory allocated to faculty record
*/
freeF_record(ptr)
record *ptr;
{
if(ptr == NULL)
return(0);
free(ptr->ssn);
free(ptr->firstName);
free(ptr->middleInitial);
free(ptr->lastName);
free(ptr->location);
free(ptr->phone);
if(ptr->next_record)
free(ptr->next_record);
return(1);
}
/*
   freeF_list : free memory allocated to faculty linked list
including "head" pointer
*/
freeF_list(head_ptr)
record  *head_ptr;
{
if(head_ptr->next_record)
freeF_recursive(head_ptr->next_record);
free(head_ptr->next_record); /* head pointer not contains
any memory allocation for elements
of structure */
free(head_ptr);
}
/*
   freeF_recursive : recursive routine to free memory of each
member of linked list of Faculty
*/
freeF_recursive(ptr)
record *ptr;
```

```
{
record *prev_ptr;
if(ptr->next_record != NULL)
freeF_recursive(ptr->next_record);
freeF_record(ptr);
}
```

```
/*cmenu.c : Menu for Course Database Services */

#include<stdio.h>
#include<ctype.h>

extern void center_record();

cmenu()
{
 extern int key;
 int i;
 int j = 0;

 while( j != 6 )
 {
  system("clear");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  system("hostname");


  printf("            ---------------------------------
----------------------\n");
  printf("      |                 COURSE   DATABASE
  FUNCTIONS            |\n");
  printf("       -----------------------------------
---------------------\n");
  printf("      |
                    |\n");
  printf("      |            (1)  ADD     A RECORD
                    |\n");
  printf("      |
                    |\n");
/******
  printf("      |            (2)  MODIFY A RECORD
                     |\n");
  printf("      |
                     |\n");
****/
  printf("      |            (2)  DELETE A RECORD
                     |\n");
  printf("      |
                      |\n");
  printf("      |            (3)  LIST    ALL RECORDS
                    |\n");
  printf("      |
                    |\n");
  printf("      |            (4)  VIEW    A RECORD
                     |\n");
  printf("      |
                      |\n");
  printf("      |            (5)  LIST FOR A PARTICULAR
 FACULTY           |\n");
  printf("      |
                       |\n");
  printf("      |           (6)  EXIT
                       |\n");
  printf("       ----------------------------------
----------------------\n");

  printf("\n");
  printf("\n");
  printf("\n");
  printf("                               Enter
 Selection (1/2/3/4/5/6): ");
  scanf("%d",&i);
  getchar();

  switch(i)  {

  case 1:  system("clear");
                j = 1;
                key = 13;
                center_record();
break;
/********
  case 2:  system("clear");
                j = 2;
printf("Modifying a record.....\n");
update_record(cl);
break;
**********/
```

```
  case 2:  system("clear");
                j = 2;
                key = 14;
cnumber_record();
break;

  case 3: system("clear");
                j = 3;
printf("Listing all records.....\n");
                key = 12;
trader();
break;

  case 4:  system("clear");
                j = 4;
            cquery_record();
break;



  case 5:  system("clear");
                j = 5;
                key = 9;
                flast_record();
             break;

  case 6:  system("clear");
                j = 6;
             break;

  default :  system("clear");
```

```
                printf("invalid input\n");
     break;


} /* end of case - switch */
  }/*end of while */
} /* end of main */
```

```
/*carecord.c : Data entry Screen for
 Course Database Services */

#include<stdio.h>
#include<ctype.h>
#include<rpc/rpc.h>
#include<crdb.h>

center_record()
{
char ch,c;
record1  *pr1;
int l =0;
int flag;
int i,j;
extern char bufdata[256];

pr1  = (record1 *) malloc(sizeof(record1));
pr1->course_number  = (char *)malloc(MAX_STR);
pr1->course_section  = (char *)malloc(MAX_STR);
pr1->course_semester  = (char *)malloc(MAX_STR);
pr1->course_name  = (char *)malloc(MAX_STR);
pr1->course_inst  = (char *)malloc(MAX_STR);
pr1->course_room  = (char *)malloc(MAX_STR);
pr1->course_bldg  = (char *)malloc(MAX_STR);
pr1->course_day  = (char *)malloc(MAX_STR);
pr1->course_time  = (char *)malloc(MAX_STR);
pr1->course_year  = (char *)malloc(MAX_STR);
pr1->course_credit  = (char *)malloc(MAX_STR);
```

```
for(i = 0; i < 256; i++)
bufdata[i] = '\0';
printf("Enter course number :");
scanf("%s",pr1->course_number);
/*   sprintf(&bufdata[l],"%s ",pr1->course_number); */
strcat(bufdata,pr1->course_number);
strcat(bufdata , " ");
 /***** A blank space is placed after each field -
 to separate the fields, hence "%s " ********/
 /***** To move the position in the buffer to place
        text pertaining to the next field ********/
        l = strlen(bufdata);

        getchar();
printf("Enter course section  :");
scanf("%s",pr1->course_section);
/*sprintf(&bufdata[l],"%s ",pr1->course_section);*/
strcat(bufdata , pr1->course_section);
strcat(bufdata , " ");
        l = strlen(bufdata);

        getchar();
printf("Enter course semester  :");
scanf("%s",pr1->course_semester);
/*sprintf(&bufdata[l],"%s ",pr1->course_semester);*/
strcat(bufdata , pr1->course_semester);
strcat(bufdata , " ");
        l = strlen(bufdata);

        getchar();
```

```
printf("Enter course name  :");
/** c = getchar();
j = 0;
while (c != '\n')
{
pr1->course_name[j] = c;
c = getchar();
++j;
}
******/
        scanf("%s",pr1->course_name);
/*sprintf(&bufdata[l],"%s ",pr1->course_name);*/
strcat(bufdata , pr1->course_name);
strcat(bufdata , " ");
        l = strlen(bufdata);

printf("Enter course instructors ssn  :");
scanf("%s",pr1->course_inst);
/*sprintf(&bufdata[l],"%s ",pr1->course_inst);*/
strcat(bufdata , pr1->course_inst);
strcat(bufdata , " ");
        l = strlen(bufdata);

        getchar();
printf("Enter room no. :");
scanf("%s",pr1->course_room);
/*sprintf(&bufdata[l],"%s ",pr1->course_room);*/
strcat(bufdata , pr1->course_room);
strcat(bufdata , " ");
        l = strlen(bufdata);
        getchar();
printf("Enter bldg.   :");
scanf("%s",pr1->course_bldg);
/*sprintf(&bufdata[l],"%s ",pr1->course_bldg);*/
strcat(bufdata , pr1->course_bldg);
strcat(bufdata , " ");
        l = strlen(bufdata);

        getchar();
printf("Enter day   :");
scanf("%s",pr1->course_day);
/*sprintf(&bufdata[l],"%s ",pr1->course_day);*/
strcat(bufdata , pr1->course_day);
strcat(bufdata , " ");
        l = strlen(bufdata);

        getchar();
printf("Enter course time  :");
scanf("%s",pr1->course_time);
/*sprintf(&bufdata[l],"%s ",pr1->course_time);*/
strcat(bufdata , pr1->course_time);
strcat(bufdata , " ");
        l = strlen(bufdata);

        getchar();
printf("Enter course year  :");
scanf("%s",pr1->course_year);
/*sprintf(&bufdata[l],"%s ",pr1->course_year);*/
strcat(bufdata , pr1->course_year);
strcat(bufdata , " ");
        l = strlen(bufdata);


        getchar();
printf("Enter credits   :");
scanf("%s",pr1->course_credit);
/*sprintf(&bufdata[l],"%s ",pr1->course_credit);*/
strcat(bufdata , pr1->course_credit);
        l = strlen(bufdata);


 getchar();
if (0)
 printf("Data Retrieved: %s\n",bufdata);
 trader();
}
```

```
/*cquery.c : Menu for Course Database Services */

#include<stdio.h>
#include<ctype.h>

extern void enter_record();
cquery_record()
{
 int i;
 int j;
 extern int key;

  while( j != 3 )
  {
  system("clear");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  system("hostname");


  printf("            ----------------------------
----------------------------\n");
  printf("      |                VIEW
 RECORDS                       |\n");
  printf("       ------------------------------
-------------------------\n");
  printf("      |
                       |\n");
  printf("      |           (1)  COURSE NAME
                        |\n");
  printf("      |
                      |\n");
  printf("      |           (2)  COURSE NUMBER
                      |\n");
  printf("      |
                        |\n");
  printf("      |           (3)  EXIT
                      |\n");
  printf("       ------------------------------
-------------------------\n");

  printf("\n");
  printf("\n");
  printf("\n");
  printf("                                   Enter
 Selection (1/2/3): ");
  scanf("%d",&i);
  getchar();

  switch(i)  {

  case 1:  system("clear");
               j = 1;
```

```
                key = 10;

                cname_record();

break;


  case 2:  system("clear");

                j = 2;

                key = 11;

cnumber_record();

break;


  case 3:  system("clear");

                j = 3;

             break;


  default :  system("clear");

     printf("invalid input \n");

     break;


} /* end of case - switch */

}/*end of while */

} /* end of main */
```

```
/*cnumber_record.c : View by Course Number
                     for Course Database Services */

#include<stdio.h>
#include<ctype.h>
#include<rpc/rpc.h>
#include<crdb.h>

cnumber_record()
{
char ch;
record1  *pr;
int l = 0;
extern char bufdata[256];
pr = (record1 *) malloc(sizeof(record1));
pr->course_number = (char *)malloc(MAX_STR);

printf("Enter Course number :");
scanf("%s",pr->course_number);
        sprintf(&bufdata[l],"%s",pr->course_number);

if (0)
 printf("Data Retrieved: %s\n",bufdata);
 trader();
}
```

```
/*cname_record.c : View by Course Name
                   for Course Database Services */

#include<stdio.h>
#include<ctype.h>
#include<rpc/rpc.h>
#include<crdb.h>

cname_record()
{
char ch,c;
record1  *pr1;
int l = 0,j;
extern char bufdata[256];
extern int key;
pr1 = (record1 *) malloc(sizeof(record1));
pr1->course_name = (char *)malloc(MAX_STR);

for (j=0;j<256;j++) bufdata[j] = '\0';
printf("Enter Course name :");
/****/ c = getchar();
j = 0;
while (c != '\n')
{
pr1->course_name[j] = c;
++j;
c = getchar();
}
/*****/
/*         scanf("%s",pr1->course_name); */
        sprintf(bufdata,"%s",pr1->course_name);
 key = 10;
if (0)
 printf("Data Retrieved: %s\n",bufdata);
 trader();
}
```

```
/**************************************************
where ever following character(s) are
used in variable/procedure name

      C,_c     :  used for COURSE DATA BASE


      F,_f     :  used for FACULTY DATA BASE


      S,_s     :  used for STUDENT DATA BASE


      R,_r     :  used for REGISTER DATA BASE

***************************************************/
#include <stdio.h>
#include <ctype.h>
#include <rpc/rpc.h>
#include "crdb.h"

void copy_crecord();
void print_clist();
void print_crecord1();
void print_crecord();
int     print_cdrecord();
int insert_crecord();

char c;

/************************************************
Functions copy, print list/record, insert
```

```
** for Course Database **
*************************************************/


/*
   insert_crecord : insert record "rec" into the
                    linked list whose header
        is given by "head" data
                    type : (record1 *)
*/
insert_crecord(head,rec)
record1 *head,*rec;
{
record1 *temp;
temp = (record1 *)malloc(sizeof(record1));
if(temp == NULL)
{
printf("insufficient memory : insert_record \n");
return(0);
}
temp->course_number = (char *)malloc(MAX_STR);
temp->course_section = (char *)malloc(MAX_STR);
temp->course_semester = (char *)malloc(MAX_STR);
temp->course_name = (char *)malloc(MAX_STR);
temp->course_inst = (char *)malloc(MAX_STR);
temp->course_room = (char *)malloc(MAX_STR);
temp->course_bldg = (char *)malloc(MAX_STR);
temp->course_day = (char *)malloc(MAX_STR);
temp->course_time = (char *)malloc(MAX_STR);
temp->course_year = (char *)malloc(5*sizeof(char));
temp->course_credit = (char *)malloc(MAX_STR);
copy_crecord(temp,rec);
temp->next_course = head->next_course;
head->next_course = temp;
return(1);
}
/*
   copy_crecord : copy record "src" into "dest"
*/

void copy_crecord(dest,src)
record1 *src,*dest;
{
strcpy(dest->course_number,src->course_number);
strcpy(dest->course_section,src->course_section);
strcpy(dest->course_semester,src->course_semester);
strcpy(dest->course_name,src->course_name);
strcpy(dest->course_inst,src->course_inst);
strcpy(dest->course_room,src->course_room);
strcpy(dest->course_bldg,src->course_bldg);
strcpy(dest->course_day,src->course_day);
strcpy(dest->course_time,src->course_time);
strcpy(dest->course_year,src->course_year);
strcpy(dest->course_credit,src->course_credit);
}


/*
   print_clist : print list starting at "first"
*/
void print_clist(first)
record1 *first;
```

```
{
record1 *temp;
       system("clear");
       printf("Number\tSection\tSemester\tName\t
       Inst Ssn\tRoom\tBldg\tday\tTime\tYear\tCredit\n");
       temp = first;
       while(temp)
{
print_crecord1(temp);
temp = temp->next_course;
}
c = getchar();
printf("Press Any Key To Continue...\n");
c = getchar();
}
/*
   print_crecord1 : print individul items of record "rec"
*/
void print_crecord1(rec)
record1 *rec;
{
printf("%s  %s  %s  %s  %s  %s  %s  %s  %s  %s  %s\n",
                rec->course_number, rec->course_section,
                rec->course_semester,rec->course_name,
                rec->course_inst,rec->course_room,
                rec->course_bldg,rec->course_day,
rec->course_time,rec->course_year,
rec->course_credit);
}
```

```
/*
   print_crecord : print individul items of record "rec"
*/
void print_crecord(rec)
record1 *rec;
{
        system("clear");
printf("Course Number    %s\n", rec->course_number);
printf("Course section   %s\n", rec->course_section);
printf("Course semester  %s\n", rec->course_semester);
printf("Course name      %s\n", rec->course_name);
printf("Course inst. ssn %s\n", rec->course_inst);
printf("Course room      %s\n", rec->course_room);
printf("Course bldg      %s\n", rec->course_bldg);
printf("Course day       %s\n", rec->course_day);
printf("Course time      %s\n", rec->course_time);
printf("Course year      %s\n", rec->course_year);
printf("Course credit    %s\n", rec->course_credit);
        c = getchar();
        printf("Press Any Key To Continue...\n");
        c = getchar();
}


/*
   print_cdrecord : print individul items of record "rec"
*/
int print_cdrecord(rec)
record1 *rec;
{
 char ch='n';
```

```
   print_crecord(rec);
   printf("\n");
   printf("\n");
   printf("\n");
   c = getchar();
   printf("Delete This Record (Y/N) ? :");
   scanf("%c",&ch);
   if(ch == 'Y' || ch == 'y')
      return(1);
   else
      return(0);
}

/*******
freeC_record : free memory allocated to student record
*/
freeC_record(ptr)
record1 *ptr;
{
        if(ptr == NULL)
                return(0);
        free(ptr->course_number);
        free(ptr->course_section);
        free(ptr->course_semester);
        free(ptr->course_name);
        free(ptr->course_inst);
        free(ptr->course_room);
        free(ptr->course_bldg);
        free(ptr->course_day);
        free(ptr->course_time);

        free(ptr->course_year);

        free(ptr->course_credit);

        if(ptr->next_course)

                free(ptr->next_course);

        return(1);

}


/*

   freeC_list : free memory allocated to course linked list

                including "head" pointer

*/

freeC_list(head_ptr)

record1 *head_ptr;

{

/*        if(head_ptr->next_course)

                freeF_recursive(head_ptr->next_course);

        free(head_ptr->next_course);

* head pointer not contains

 any memory allocation for elements

 of structure *

        free(head_ptr);

*/

}

/*

   freeC_recursive : recursive routine to free memory of each

        member of linked list of Course

*/

freeC_recursive(ptr)

record1 *ptr;
```

```
{
record1 *prev_ptr;
        if(ptr->next_course != NULL)
               freeC_recursive(ptr->next_course);
        freeC_record(ptr);
}
```

```
/*smenu.c : Menu for Student Database Services */

#include<stdio.h>
#include<ctype.h>

extern void senter_record();

smenu()
{
 extern int key;
 int i;
 int j;

while( j != 5)
{
  system("clear");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  system("hostname");


  printf("            -----------------------------------
---------------------\n");
  printf("       |                STUDENT   DATABASE
  FUNCTIONS          |\n");
  printf("       ------------------------------------
--------------------\n");
  printf("       |
                     |\n");
  printf("       |           (1)  ADD    A RECORD
                     |\n");
  printf("       |
                    |\n");
/******
  printf("       |           (2)  MODIFY A RECORD
                     |\n");
  printf("       |
                      |\n");
****/
  printf("       |           (2)  DELETE A RECORD
                     |\n");
  printf("       |
                     |\n");
  printf("       |           (3)  LIST   ALL RECORDS
                    |\n");
  printf("       |
                     |\n");
  printf("       |           (4)  VIEW   A RECORD
                      |\n");
  printf("       |
                        |\n");
  printf("       |           (5)  EXIT
                      |\n");
  printf("       ------------------------------------
--------------------\n");


  printf("\n");
  printf("\n");
  printf("\n");
  printf("                           Enter
 Selection (1/2/3/4/5): ");
  scanf("%d",&i);


  switch(i)  {


  case 1:  system("clear");
                j = 1;
                key = 20;
                senter_record();
break;
/********
  case 2:  system("clear");
                j = 2;
printf("Modifying a record.....\n");
update_record(cl);
break;
**********/
  case 2:  system("clear");
                j = 2;
                key = 21;
sssn_record();
break;
```

```
  case 3: system("clear");
                j = 3;
printf("Listing all records.....\n");
                key = 19;
trader();
break;

  case 4:  system("clear");
                j = 4;
            squery_record();
break;

  case 5:  system("clear");
                j = 5;
             break;

  default :  system("clear");
     break;

} /* end of case - switch */
} /* end of while */
} /* end of main */
```

```
/*sarecord.c : Data Entry Screen for Student Database Services */

#include<stdio.h>
#include<ctype.h>
#include<rpc/rpc.h>
#include<srdb.h>

senter_record()
{
char ch;
record2  *pr2;
int l =0;
int i;
int flag;
extern char bufdata[256];

pr2  = (record2 *) malloc(sizeof(record2));
pr2->stud_ssn  = (char *)malloc(MAX_STR);
pr2->stud_firstName  = (char *)malloc(MAX_STR);
pr2->stud_middleInitial  = (char *)malloc(MAX_STR);
pr2->stud_lastName  = (char *)malloc(MAX_STR);
pr2->stud_address  = (char *)malloc(MAX_STR);
pr2->stud_city  = (char *)malloc(MAX_STR);
pr2->stud_state  = (char *)malloc(MAX_STR);
pr2->stud_zip  = (char *)malloc(MAX_STR);
pr2->stud_phone          = (char *)malloc(MAX_STR);
pr2->stud_major          = (char *)malloc(MAX_STR);
pr2->stud_college  = (char *)malloc(MAX_STR);
pr2->stud_gpa  = (char *)malloc(MAX_STR);
```

```
        for(i = 0; i <256;i++)
           bufdata[i] = '\0';
printf("Enter ssn number :");
scanf("%s",pr2->stud_ssn);
        sprintf(&bufdata[l],"%s ",pr2->stud_ssn);
 /***** A blank space is placed after each field -
to separate the fields, hence "%s " ********/
 /***** To move the position in the buffer to place
        text pertaining to the next field ********/
        l = strlen(bufdata);

        getchar();
printf("Enter first name :");
scanf("%s",pr2->stud_firstName);
strcat(bufdata , pr2->stud_firstName);
strcat(bufdata , " ");
        l = strlen(bufdata);

        getchar();
printf("Enter middle initial :");
scanf("%s",pr2->stud_middleInitial);
strcat(bufdata , pr2->stud_middleInitial);
strcat(bufdata , " ");
        l = strlen(bufdata);

        getchar();
printf("Enter last name :");
scanf("%s",pr2->stud_lastName);
strcat(bufdata , pr2->stud_lastName);
strcat(bufdata , " ");
        l = strlen(bufdata);


        getchar();
printf("Enter street address:");
scanf("%s",pr2->stud_address);
strcat(bufdata , pr2->stud_address);
strcat(bufdata , " ");
        l = strlen(bufdata);



        getchar();
printf("Enter city :");
scanf("%s",pr2->stud_city);
strcat(bufdata , pr2->stud_city);
strcat(bufdata , " ");
        l = strlen(bufdata);



        getchar();
printf("Enter state :");
scanf("%s",pr2->stud_state);
strcat(bufdata , pr2->stud_state);
strcat(bufdata , " ");
        l = strlen(bufdata);



        getchar();
printf("Enter zip :");
scanf("%s",pr2->stud_zip);
```

```
strcat(bufdata , pr2->stud_zip);
strcat(bufdata , " ");
        l = strlen(bufdata);

        getchar();
printf("Enter phone number :");
scanf("%s",pr2->stud_phone);
strcat(bufdata , pr2->stud_phone);
strcat(bufdata , " ");
        l = strlen(bufdata);

        getchar();
printf("Enter major:");
scanf("%s",pr2->stud_major);
strcat(bufdata , pr2->stud_major);
strcat(bufdata , " ");
        l = strlen(bufdata);

        getchar();
printf("Enter college:");
scanf("%s",pr2->stud_college);
strcat(bufdata , pr2->stud_college);
strcat(bufdata , " ");
        l = strlen(bufdata);

        getchar();
printf("Enter gpa:");
scanf("%s",pr2->stud_gpa);
strcat(bufdata , pr2->stud_gpa);
        l = strlen(bufdata);
```

```
 getchar();
if (0)
 printf("Data Retrieved: %s\n",bufdata);
 getchar();
 trader();
}
```

```
/*squery.c : Menu for Faculty Database Services */

#include<stdio.h>
#include<ctype.h>

squery_record()
{
 int i;
 int j;
 extern int key;
while( j !=5 )
 {
  system("clear");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  system("hostname");


  printf("            --------------------------
-------------------------------\n");
  printf("      |                  VIEW
  RECORDS                   |\n");
 printf("       ---------------------------
---------------------------\n");
 printf("      |
                           |\n");
 printf("      |           (1)  FIRST NAME
                            |\n");
 printf("      |
                           |\n");
 printf("      |           (2)  LAST NAME
                           |\n");
 printf("      |
                           |\n");
 printf("      |           (3)  SOCIAL SECURITY
NUMBER                 |\n");
 printf("      |
                        |\n");
 printf("      |           (4)  PHONE NUMBER
                        |\n");
 printf("      |
                          |\n");
 printf("      |           (5)  EXIT
                             |\n");
 printf("       -------------------------------
-----------------------\n");

 printf("\n");
 printf("\n");
 printf("\n");
 printf("                                   Enter
Selection (1/2/3/4/5): ");
```

```
  scanf("%d",&i);

  switch(i)  {

  case 1:  system("clear");
                j = 1;
                key = 15;
                sfirst_record();
break;

  case 2:  system("clear");
                j = 2;
                key = 16;
slast_record();
break;

  case 3:  system("clear");
                j = 3;
                key = 18;
sssn_record();
break;

  case 4:  system("clear");
                j = 4;
                key = 17;
            sphone_record();
break;

  case 5:  system("clear");
                j = 5;
                break;

     default :  system("clear");
                   j = 1;
        printf("invalid input \n");
        break;

   } /* end of case - switch */
   } /* end of while */
   } /* end of main */
```

```
/*sssn_record.c : View Screen for
                  Student Database Services */

#include<stdio.h>
#include<ctype.h>
#include<rpc/rpc.h>
#include<srdb.h>

void sssn_record()
{
char ch;
record2  *pr2;
int l=0;
extern char bufdata[256];
pr2 = (record2 *) malloc(sizeof(record2));
pr2->stud_ssn = (char *)malloc(MAX_STR);

printf("Enter ssn number :");
scanf("%s",pr2->stud_ssn);
        sprintf(&bufdata[l],"%s",pr2->stud_ssn);

if (0)
 printf("Data Retrieved: %s\n",bufdata);
 trader();
}
```

```
/*sfirst_record.c : View by Firstname
 Screen for STUDENT Database Services */

#include<stdio.h>
#include<ctype.h>
#include<rpc/rpc.h>
#include<srdb.h>

sfirst_record()
{
char ch;
record2  *pr2;
int l = 0;
extern char bufdata[256];
pr2 = (record2 *) malloc(sizeof(record2));
pr2->stud_firstName = (char *)malloc(MAX_STR);

printf("Enter first name :");
scanf("%s",pr2->stud_firstName);
        sprintf(&bufdata[l],"%s",pr2->stud_firstName);

if (0)
 printf("Data Retrieved: %s\n",bufdata);
 trader();
}
```

```
/*slast_record.c : View by lastname

 Screen for Student Database Services */


#include<stdio.h>

#include<ctype.h>

#include<rpc/rpc.h>

#include<srdb.h>


void slast_record()

{

char ch;

record2  *pr2;

int l = 0;

extern char bufdata[256];

pr2 = (record2 *) malloc(sizeof(record2));

pr2->stud_lastName = (char *)malloc(MAX_STR);


printf("Enter last name :");

scanf("%s",pr2->stud_lastName);

        sprintf(&bufdata[l],"%s",pr2->stud_lastName);


if (0)

 printf("Data Retrieved: %s\n",bufdata);

 trader();

}
```

```
/*sphone_record.c : View  Screen for
                    Student Database Services */

#include<stdio.h>
#include<ctype.h>
#include<rpc/rpc.h>
#include<srdb.h>

sphone_record()
{
char ch;
record2  *pr2;
int l =0;
extern char bufdata[256];
pr2 = (record2 *) malloc(sizeof(record2));
pr2->stud_phone = (char *)malloc(MAX_STR);

printf("Enter phone number :");
scanf("%s",pr2->stud_phone);
        sprintf(&bufdata[l],"%s",pr2->stud_phone);

if (0)
 printf("Data Retrieved: %s\n",bufdata);
 trader();
}
```

```
/*************** GENERAL NOTATIONS  **********************

where ever following character(s) are

used in variable/procedure name


      C,_c     :  used for COURSE DATA BASE


      F,_f     :  used for FACULTY DATA BASE


      S,_s     :  used for STUDENT DATA BASE


      R,_r    :  used for REGISTER DATA BASE
***********************************************************/
#include <stdio.h>

#include <ctype.h>

#include <rpc/rpc.h>

#include "srdb.h"


void copy_srecord();

void print_slist();

void print_srecord( );

void print_srecord1( );

int print_sdrecord( );

int insert_srecord( );


char c;


/***************************************************

Functions copy, print list/record, insert for
```

```
** Student Database ****
***************************************************/
/*
  insert_srecord : insert record "rec" into the linked
                   list whose header is given by "head"
*/
insert_srecord(head,rec)
record2 *head,*rec;
{
record2 *temp;
temp = (record2 *)malloc(sizeof(record2));
if(temp == NULL)
{
printf("insufficient memory : insert_record \n");
return(0);
}
temp->stud_ssn = (char *)malloc(MAX_STR);
temp->stud_firstName = (char *)malloc(MAX_STR);
temp->stud_middleInitial = (char *)malloc(MAX_STR);
temp->stud_lastName = (char *)malloc(MAX_STR);
temp->stud_address = (char *)malloc(MAX_STR);
temp->stud_city = (char *)malloc(MAX_STR);
temp->stud_state = (char *)malloc(MAX_STR);
temp->stud_zip = (char *)malloc(MAX_STR);
temp->stud_phone = (char *)malloc(12*sizeof(char));
temp->stud_major = (char *)malloc(MAX_STR);
temp->stud_college = (char *)malloc(MAX_STR);
temp->stud_gpa = (char *)malloc(MAX_STR);
copy_srecord(temp,rec);
temp->next_record = head->next_record;
```

```
head->next_record = temp;
return(1);
}
/*
   copy_srecord : copy record "src" into "dest"
*/
void copy_srecord(dest,src)
record2 *src,*dest;
{
strcpy(dest->stud_ssn,src->stud_ssn);
strcpy(dest->stud_firstName,src->stud_firstName);
strcpy(dest->stud_middleInitial,src->stud_middleInitial);
strcpy(dest->stud_lastName,src->stud_lastName);
strcpy(dest->stud_address,src->stud_address);
strcpy(dest->stud_city,src->stud_city);
strcpy(dest->stud_state,src->stud_state);
strcpy(dest->stud_zip,src->stud_zip);
strcpy(dest->stud_phone,src->stud_phone);
strcpy(dest->stud_major,src->stud_major);
strcpy(dest->stud_college,src->stud_college);
strcpy(dest->stud_gpa,src->stud_gpa);
}
/*
   print_slist : print list starting at "first"
*/
void print_slist(first)
record2 *first;
{
record2 *temp;
temp = first;
```

```
        system("clear");
printf("ssn\tfirst\tmiddle\tlast\taddress\t
        city\tstate\tzip\tphone\tmajor\tcollege\tgpa\n");
while(temp)
{
print_srecord1(temp);
temp = temp->next_record;
}
c = getchar();
printf("Press Any Key to continue.....\n");
c = getchar();
}
/*
   print_srecord1 : print individul items of record "rec"
*/
void print_srecord1(rec)
record2 *rec;
{
printf("%s  %s  %s  %s  %s  %s  %s  %s  %s  %s  %s  %s  \n",
rec->stud_ssn,rec->stud_firstName,rec->stud_middleInitial,
rec->stud_lastName,rec->stud_address,rec->stud_city,
rec->stud_state,rec->stud_zip,rec->stud_phone,
rec->stud_major,rec->stud_college,rec->stud_gpa);
}



/*
   print_srecord : print individul items of record "rec"
*/
void print_srecord(rec)
```

```
record2 *rec;
{
        system("clear");
printf("SSN            %s \n",rec->stud_ssn);
printf("First Name     %s \n",rec->stud_firstName);
printf("Middle Initial %s \n",rec->stud_middleInitial);
printf("Last Name      %s \n",rec->stud_lastName);
printf("Address        %s \n",rec->stud_address);
printf("City           %s \n",rec->stud_city);
printf("State          %s \n",rec->stud_state);
printf("Zip            %s \n",rec->stud_zip);
printf("Phone          %s \n",rec->stud_phone);
printf("Major          %s \n",rec->stud_major);
printf("College        %s \n",rec->stud_college);
printf("GPA            %s \n",rec->stud_gpa);
        c = getchar();
        printf("Press Any Key to continue.....\n");
        c = getchar();
}



/*
   print_sdrecord : print individul items of record "rec"
*/
print_sdrecord(rec)
record2 *rec;
{
  char ch='n';


   print_srecord(rec);
   printf("\n");
   printf("\n");
   printf("\n");
   c = getchar();
   printf("Delete This Record (Y/N) ? :");
   scanf("%c",&ch);
   if(ch == 'Y' || ch == 'y')
      return(1);
   else
      return(0);
}

/*
   freeS_record : free memory allocated to student record
*/
freeS_record(ptr)
record2 *ptr;
{
if(ptr == NULL)
return(0);
free(ptr->stud_ssn);
free(ptr->stud_firstName);
free(ptr->stud_middleInitial);
free(ptr->stud_lastName);
free(ptr->stud_address);
free(ptr->stud_city);
free(ptr->stud_state);
free(ptr->stud_zip);
free(ptr->stud_phone);
free(ptr->stud_major);
free(ptr->stud_college);

free(ptr->stud_gpa);

if(ptr->next_record)

free(ptr->next_record);

return(1);

}

/*

   freeS_list : free memory allocated to student linked list

including "head" pointer

*/

freeS_list(head_ptr)

record2  *head_ptr;

{

if(head_ptr->next_record)

freeF_recursive(head_ptr->next_record);

free(head_ptr->next_record);

/* head pointer not contains

any memory allocation for elements

of structure */

free(head_ptr);

}

/*

   freeS_recursive : recursive routine

        to free memory of each

member of linked list of Student

*/

freeS_recursive(ptr)

record2 *ptr;

{

record2 *prev_ptr;
```

```
if(ptr->next_record != NULL)

freeS_recursive(ptr->next_record);

freeS_record(ptr);

}
```

```
/*rmenu.c : Menu for Register Database Services */

#include<stdio.h>
#include<ctype.h>

extern void renter_record();

rmenu()
{
 extern int key;
 int i;
 int j;
while( j != 5)
{
  system("clear");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  system("hostname");


  printf("            ------------------------------------
--------------------\n");
```

```
  printf("     |              REGISTER  DATABASE
  FUNCTIONS        |\n");
  printf("       ------------------------------------
--------------------\n");
  printf("     |
                  |\n");
  printf("     |           (1)  ADD    A RECORD
                  |\n");
  printf("     |
                   |\n");
/******
  printf("     |           (2)  MODIFY A RECORD
                   |\n");
  printf("     |
                     |\n");
****/
  printf("     |           (2)  DELETE A RECORD
                       |\n");
  printf("     |
                      |\n");
  printf("     |           (3)  LIST   ALL RECORDS
                    |\n");
  printf("     |
                        |\n");
  printf("     |           (4)  VIEW   A RECORD
                      |\n");
  printf("     |
                       |\n");
  printf("     |           (5)  EXIT
                           |\n");
  printf("       --------------------------------
----------------------\n");

  printf("\n");
  printf("\n");
  printf("\n");
  printf("                           Enter
 Selection (1/2/3/4/5): ");
  scanf("%d",&i);

  switch(i)  {

  case 1:  system("clear");
              j = 1;
              key = 25;
              renter_record();
break;
/********
  case 2:  system("clear");
              j = 2;
printf("Modifying a record.....\n");
update_record(cl);
break;
**********/
  case 2:  system("clear");
              j = 2;
              key = 26;
rssn_record();
break;
```

```
  case 3: system("clear");
              j = 3;
printf("Listing all records.....\n");
              key = 24;
trader();
break;

  case 4:  system("clear");
              j = 4;
            rquery_record();
break;

  case 5:  system("clear");
              j = 5;
             break;

  default :  system("clear");
     printf("invalid input \n");
     break;

} /* end of case - switch */
} /* end of while*/
} /* end of main */
```

```
/*rarecord.c : Data entry Screen
               for Register Database Services */

#include<stdio.h>
#include<ctype.h>
#include<rpc/rpc.h>
#include<rrdb.h>

renter_record()
{
char ch;
record3  *pr3;
int l =0;
int flag;
int i;
extern char bufdata[256];

pr3  = (record3 *) malloc(sizeof(record3));
pr3->stud_ssn        = (char *)malloc(MAX_STR);
pr3->course_number  = (char *)malloc(MAX_STR);
pr3->course_section  = (char *)malloc(MAX_STR);
pr3->course_semester  = (char *)malloc(MAX_STR);
pr3->course_year  = (char *)malloc(MAX_STR);
pr3->grade       = (char *)malloc(MAX_STR);

for(i = 0; i < 256; i++)
bufdata[i] = '\0';
printf("Enter Student Ssn :");
scanf("%s",pr3->stud_ssn);
```

```
/*        sprintf(&bufdata[l],"%s ",pr3->stud_ssn); */
strcat(bufdata,pr3->stud_ssn);
strcat(bufdata , " ");


        getchar();
printf("Enter course number :");
scanf("%s",pr3->course_number);
/*        sprintf(&bufdata[l],"%s ",pr3->course_number); */
strcat(bufdata,pr3->course_number);
strcat(bufdata , " ");
/***** A blank space is placed after each field -
 to separate the fields,
        hence "%s " ********/
 /***** To move the position in the buffer to place
        text pertaining to the next field ********/
        l = strlen(bufdata);


        getchar();
printf("Enter course section  :");
scanf("%s",pr3->course_section);
        /*sprintf(&bufdata[l],"%s ",pr3->course_section);*/
strcat(bufdata , pr3->course_section);
strcat(bufdata , " ");
        l = strlen(bufdata);


        getchar();
printf("Enter course semester  :");
scanf("%s",pr3->course_semester);
        /*sprintf(&bufdata[l],"%s ",pr3->course_semester);*/
strcat(bufdata , pr3->course_semester);
strcat(bufdata , " ");
        l = strlen(bufdata);


        getchar();
printf("Enter course year  :");
scanf("%s",pr3->course_year);
        /*sprintf(&bufdata[l],"%s ",pr3->course_year);*/
strcat(bufdata , pr3->course_year);
strcat(bufdata , " ");
        l = strlen(bufdata);


        getchar();
printf("Enter grade   :");
scanf("%s",pr3->grade);
        /*sprintf(&bufdata[l],"%s ",pr3->grade);*/
strcat(bufdata , pr3->grade);
        l = strlen(bufdata);


 getchar();
if (0)
 printf("Data Retrieved: %s\n",bufdata);
 trader();
}
```

```
/*rquery.c : Menu for Register Database Services */

#include<stdio.h>
#include<ctype.h>

rquery_record()
{
 int i;
 int j;
 extern int key;

while( j != 4)
{

  system("clear");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n");
  system("hostname");


  printf("            ---------------------------
-----------------------------\n");
```

```
  printf("      |                 VIEW
  RECORDS                      |\n");
  printf("       ----------------------------
----------------------------\n");
  printf("      |
                               |\n");
  printf("      |            (1)  GRADE
                               |\n");
  printf("      |
                               |\n");
  printf("      |            (2)  COURSE
                               |\n");
  printf("      |
                                 |\n");
  printf("      |            (3)  SSN
                               |\n");
  printf("      |
                                |\n");
  printf("      |            (4)  EXIT
                                 |\n");
  printf("       ----------------------------
----------------------------\n");

  printf("\n");
  printf("\n");
  printf("\n");
  printf("                            Enter
 Selection (1/2/3/4): ");
  scanf("%d",&i);
  switch(i)  {

  case 1:  system("clear");
                j = 1;
                key = 22;
                rgrade_record();
break;

  case 2:  system("clear");
                j = 2;
                key = 27;
rcourse_record();
break;

  case 3:  system("clear");
                j = 3;
                key = 23;
rssn_record();
break;

  case 4:  system("clear");
                j = 4;
             break;

  default :  system("clear");
     printf("invalid input \n");
     break;

} /* end of case - switch */
} /* end of while */
} /* end of main */
```

```
/*rgrade_record.c : View by grade Screen
                    for Register Database Services */

#include<stdio.h>
#include<ctype.h>
#include<rpc/rpc.h>
#include<rrdb.h>

rgrade_record()
{
char ch;
record3  *pr3;
int l =0;
int i =0;
int flag;
extern char bufdata[256];
extern int key;

pr3  = (record3 *) malloc(sizeof(record3));
pr3->course_number  = (char *)malloc(MAX_STR);
pr3->course_section  = (char *)malloc(MAX_STR);
pr3->course_semester  = (char *)malloc(MAX_STR);
pr3->course_year  = (char *)malloc(MAX_STR);
pr3->grade        = (char *)malloc(MAX_STR);

        for(i = 0; i < 256; i++)
            bufdata[i] = '\0';
        key = 22;
printf("Enter course number  :");
```

```
scanf("%s",pr3->course_number);
        /*sprintf(&bufdata[l],"%s ",pr3->course_number);*/
strcat(bufdata , pr3->course_number);
strcat(bufdata , " ");
        l = strlen(bufdata);
        printf("l = %d\n",l);


        getchar();
printf("Enter course section  :");
scanf("%s",pr3->course_section);
        /*sprintf(&bufdata[l],"%s ",pr3->course_section);*/
strcat(bufdata , pr3->course_section);
strcat(bufdata , " ");
        l = strlen(bufdata);
        printf("l = %d\n",l);


        getchar();
printf("Enter course semester  :");
scanf("%s",pr3->course_semester);
        /*sprintf(&bufdata[l],"%s ",pr3->course_semester);*/
strcat(bufdata , pr3->course_semester);
strcat(bufdata , " ");
        l = strlen(bufdata);
        printf("l = %d\n",l);


        getchar();
printf("Enter course year  :");
scanf("%s",pr3->course_year);
        /*sprintf(&bufdata[l],"%s ",pr3->course_year);*/
strcat(bufdata , pr3->course_year);
strcat(bufdata , " ");
        l = strlen(bufdata);
        printf("l = %d\n",l);


        getchar();
printf("Enter grade   :");
scanf("%s",pr3->grade);
        /*sprintf(&bufdata[l],"%s ",pr3->grade);*/
strcat(bufdata , pr3->grade);
strcat(bufdata , "\n");
        l = strlen(bufdata);
        printf("l = %d\n",l);


 getchar();
if (0)
 printf("Data Retrieved: %s\n",bufdata);
 trader();
}
```

```
/*rssn_record.c : View Screen for
                  Register Database Services */

#include<stdio.h>
#include<ctype.h>
#include<rpc/rpc.h>
#include<rrdb.h>

void rssn_record()
{
char ch;
record3  *pr3;
int l=0;
extern char bufdata[256];
pr3 = (record3 *) malloc(sizeof(record3));
pr3->stud_ssn = (char *)malloc(MAX_STR);

printf("Enter ssn number :");
scanf("%s",pr3->stud_ssn);
        sprintf(&bufdata[l],"%s",pr3->stud_ssn);

if (0)
 printf("Data Retrieved: %s\n",bufdata);
 trader();
}
```

```
/*********** GENERAL NOTATIONS  **********************

where ever following character(s) are

used in variable/procedure name


      C,_c     :  used for COURSE DATA BASE


      F,_f     :  used for FACULTY DATA BASE


      S,_s     :  used for STUDENT DATA BASE


      R,_r     :  used for REGISTER DATA BASE

*****************************************************/
#include <stdio.h>
#include <ctype.h>
#include <rpc/rpc.h>
#include "rrdb.h"

void copy_rrecord();
void print_rlist();
void print_rrecord1();
void print_rrecord();
int print_rdrecord( );
int insert_rrecord( );

char c;

/**********************************************************
```

```
Functions copy, print list/record, insert
  **for Register Database **
*************************************************************/


/*
   insert_rrecord : insert record "rec" into the
                    linked list whose header
        is given by "head" data
                    type : (record1 *)
*/
insert_rrecord(head,rec)
record3 *head,*rec;
{
record3 *temp;
temp = (record3 *)malloc(sizeof(record3));
if(temp == NULL)
{
printf("insufficient memory : insert_record \n");
return(0);
}
temp->stud_ssn       = (char *)malloc(MAX_STR);
temp->course_number  = (char *)malloc(MAX_STR);
temp->course_section = (char *)malloc(MAX_STR);
temp->course_semester= (char *)malloc(MAX_STR);
temp->course_year    = (char *)malloc(MAX_STR);
temp->grade          = (char *)malloc(MAX_STR);
copy_rrecord(temp,rec);
temp->next_record = head->next_record;
head->next_record = temp;
return(1);
```

```
}
/*
   copy_rrecord : copy record "src" into "dest"
*/
void copy_rrecord(dest,src)
record3 *src,*dest;
{
strcpy(dest->stud_ssn,src->stud_ssn);
strcpy(dest->course_number,src->course_number);
strcpy(dest->course_section,src->course_section);
strcpy(dest->course_semester,src->course_semester);
strcpy(dest->course_year,src->course_year);
strcpy(dest->grade,src->grade);
}
/*
   print_rlist : print list starting at "first"
*/
void print_rlist(first)
record3 *first;
{
record3 *temp;
temp = first;
        system("clear");
printf("Student SSN\tCourse Number\tCourse Section\t
                Course Semester\tCourse Year\tCourse Grade\n");
while(temp)
{
print_rrecord1(temp);
temp = temp->next_record;
}
```

```
c = getchar();
printf("Press Any Key To Continue.....\n");
c = getchar();
}
/*
   print_rrecord1 : print individul items of record "rec"
*/
void print_rrecord1(rec)
record3 *rec;
{
printf("%s  %s  %s  %s  %s  %s  \n",rec->stud_ssn,
                rec->course_number, rec->course_section,
                rec->course_semester,
rec->course_year, rec->grade);
}


/*
   print_rrecord : print individul items of record "rec"
*/
void print_rrecord(rec)
record3 *rec;
{
        system("clear");
printf("Student SSN     %s\n", rec->stud_ssn);
        printf("Course Number   %s\n", rec->course_number);
        printf("Course Section  %s\n", rec->course_section);
        printf("Course Semester %s\n", rec->course_semester);
        printf("Course Year     %s\n", rec->course_year);
        printf("Course Grade    %s\n", rec->grade);
        c = getchar();
        printf("Press Any Key To Continue.....\n");
        c = getchar();
}


/*
   print_rdrecord : print individul items of record "rec"
*/
print_rdrecord(rec)
record3 *rec;
{
 char ch='n';


   print_rrecord(rec);
   printf("\n");
   printf("\n");
   printf("\n");
   c = getchar();
   printf("Delete This Record (Y/N) ? :");
   scanf("%c",&ch);
   if(ch == 'Y' || ch == 'y')
      return(1);
   else
      return(0);
}
```

7.2.3 Faculty Building Blocks

```
/* frdb_svc_proc.c : Remote database service
                     procedures for Faculty.data*/

#include <stdio.h>
#include <string.h>
#include <rpc/rpc.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#include "frdb.h"

record  *pR1;    /* For Faculty.data */
char ch[1000];
char  buf[MAX_STR];
FILE  *fpp1;    /* For Faculty.data */
int fd1;


/***Func. readRecord for Faculty.data **********/

int readRecord(fpos)
int  fpos;
{
int  i;
char ch;
int ret,flag;

        if (0) printf("In read record......\n");
```

```
if(!pR1)

{

pR1 = (record *)malloc(sizeof(record));

pR1->ssn = (char *)malloc(MAX_STR);

pR1->firstName = (char *)malloc(MAX_STR);

pR1->middleInitial = (char *)malloc(MAX_STR);

pR1->lastName = (char *)malloc(MAX_STR);

pR1->location = (char *)malloc(MAX_STR);

pR1->phone              = (char *)malloc(12 * sizeof(char));

pR1->next_record = NULL;

}


        if (0)

        printf("In read record...memory allocated to pR1...\n");

if(lseek(fd1,fpos,SEEK_SET) == -1)

{

if (0) printf("lseek1 : %x \n",errno);

return(0);

}

i = flag = 0;

ret = read(fd1,&ch,1);

if(ret == 0)

return(0); /* end of file */

while( ch != '\n' && ch != EOF && ret > 0)

{

   buf[i++] = ch;

flag = 1;

ret = read(fd1,&ch,1);

}

buf[i] = '\0';
```

```
if(sscanf(buf,"%s %s %s %s %s %s",pR1->ssn,pR1->firstName,
pR1->middleInitial,pR1->lastName,pR1->phone,
pR1->location) != 6)
return(0);
return((int)tell(fd1));
}

/*** Server procedures for Faculty.data ***********/

record  *firstname_key_1(name)
char **name;
{
return( getF_record(*name,FIRSTNAME_KEY) ? (record *)pR1 :
(record  *)NULL);
}

record *ssn_key_1(ssnumber)
char **ssnumber;
{
        if (0)
        printf("ssn = %s\n",ssnumber);
return( getF_record(*ssnumber,SSN_KEY) ? (record *)pR1 :
(record  *)NULL);
}

record *lastname_key_1(name)
char **name;
{
return( getF_record(*name,LASTNAME_KEY) ? (record *)pR1 :
     (record  *)NULL);
```

```
}

record *phone_key_1(phnumber)
char **phnumber;
{
return( getF_record(*phnumber,PHONE_KEY) ? (record *)pR1 :
(record  *)NULL);
}

record *location_key_1(location)
char **location;
{
return( getF_record(*location,LOCATION_KEY) ? (record *)pR1 :
(record  *)NULL);
}

record *list_record_1(record_number)
int **record_number;
{
int i;
fd1 = open(DATABASE,O_RDONLY);
if(fd1 < 0)
{
if (0) printf("file open error : %s \n",DATABASE);
return((record *)NULL);
}
pR1->file_offset = readRecord(*record_number);
close(fd1);
return((record  *)pR1);
}
```

```
record *firstname_record_1(rec)
record  *rec;
{
int     i;
        fd1 = open(DATABASE,O_RDONLY);
        if(fd1 < 0)
        {
          if (0) printf("file open error: %s \n",DATABASE);
          return((record *)NULL);
        }
/*note rec?*/   i = rec->file_offset;
                if (0) printf("i = %d\n",i);
        while(i = readRecord(i))
        {
            if ((!strcmp(rec->firstName,pR1->firstName)))
                {
                        close(fd1);
                        pR1->file_offset = i;
                        return((record  *)pR1);
                }
        }
        close(fd1);
        pR1->file_offset = i;
        return((record  *)pR1);
}


record *lastname_record_1(rec)
record *rec;
```

```
{
int     i;
        fd1 = open(DATABASE,O_RDONLY);
        if(fd1 < 0)
        {
           if (0) printf("file open error: %s \n",DATABASE);
                return((record *)NULL);
        }
/*note rec?*/   i = rec->file_offset;
                if (0) printf("i = %d\n",i);
        while(i = readRecord(i))
        {
            if ((!strcmp(rec->lastName,pR1->lastName)))
                {
                        close(fd1);
                        pR1->file_offset = i;
                        return((record  *)pR1);
                }
        }
        close(fd1);
        pR1->file_offset = i;
        return((record  *)pR1);
}

/************Func. getF_record for Faculty.data **********/

getF_record(c_string,c_key)
char *c_string;
int c_key;
{
```

```
int i;
int fflag;
fd1 = open(DATABASE,O_RDONLY);
if(fd1 < 0)
{
if (0) printf("file open error : %s \n",DATABASE);
return(0);
}
i = fflag = 0;
while(i = readRecord(i))
{
switch(c_key)
{
case FIRSTNAME_KEY :
if(!strcmp(pR1->firstName,c_string))
fflag = 1;
break;
case LASTNAME_KEY :
if(!strcmp(pR1->lastName,c_string))
fflag = 1;
break;
case PHONE_KEY :
if(!strcmp(pR1->phone,c_string))
fflag = 1;
break;
case LOCATION_KEY :
if(!strcmp(pR1->location,c_string))
fflag = 1;
break;
case SSN_KEY :
```

```
                         if (0) printf("ssn = %s\n",pR1->ssn);
                         if (0) printf("c_string = %s\n",c_string);
  if(!strcmp(pR1->ssn,c_string))
                          {
                            if (0) printf("fflag = %d\n",fflag);
           fflag = 1;
                           }
break;
default :
break;
} /* end of SWITCH */
if(fflag)
break;
} /* end of WHILE */

close(fd1);
pR1->file_offset = i;
return(1);
}
```

```
/*frdb_svc_del.c : Remote database Delete
           service procedures for Faculty.data*/

# include <stdio.h>
# include <string.h>
# include <rpc/rpc.h>
# include <errno.h>
# include <fcntl.h>
# include <unistd.h>
# include "frdb.h"

/***Delete Procedure for Faculty.data *****/

int *del_record_1();

extern record *pR1;
extern int fd1;

int *del_record_1(ssnumber)
char **ssnumber;
{
static  int  status;
int i,cnt;
char  buf[MAX_STR];
record *prev,*head;
record  *current,*temp;
record *pcurent;

fd1 = open(DATABASE,O_RDONLY);
```

```
if(fd1 < 0)
{
status = 0;
if (0) printf("file open error \n");
return((int *) &status); /* file open error */
  }
prev = current = (record *)NULL;
temp = head = (record *)malloc(sizeof(record));
/* first : header */
temp->next_record = NULL;
i = cnt =  0;
while(i = readRecord(i))
{
prev = temp;
temp->next_record = (record *)malloc(sizeof(record));
temp = temp->next_record;
temp->next_record = NULL;
++cnt;
copy_record(temp,pR1);
if (0)
printf("temp : %x : %x : %s\n",temp,temp->next_record,temp->ssn);
if(!strcmp(temp->ssn,*ssnumber))
{
pcurent = prev;
current = temp;
}
}
close(fd1);
if(current == (record *)NULL)
{
```

```
status = 0;
if (0) printf("record not found \n");
}
else
{
if (0)
                 printf("record found \n");
if (0)
                printf("ssn = %s : %d \n",current->ssn,cnt);
pcurent->next_record = current->next_record;
temp = head->next_record;
fd1 = open(DATABASE,O_WRONLY|O_TRUNC);
if(fd1 < 0)
{
status = 0;
if (0) printf("file open error \n");
return((int *) &status);
  }
i = 0;
while(temp != (record *)NULL)
{
++i;
if(i > cnt) break;
sprintf(buf,"%s %s %s %s %s %s\n",temp->ssn,
                temp->firstName, temp->middleInitial,
                temp->lastName, temp->phone,temp->location);
write(fd1,buf,strlen(buf));
if (0) printf("i = %d \n",i);
if (0)
                printf("%s %s %s %s %s %s\n",temp->ssn,
                temp->firstName,temp->middleInitial,
                temp->lastName, temp->phone,temp->location);
temp = temp->next_record;
if (0) printf(" temp = %x \n",temp);
}
  close(fd1);
status = 1;
}
  return ((int *) &status);
}
copy_record(dest,src)
record *dest,*src;
{
dest->ssn           = (char *)malloc(MAX_STR);
dest->firstName     = (char *)malloc(MAX_STR);
dest->middleInitial = (char *)malloc(MAX_STR);
dest->lastName      = (char *)malloc(MAX_STR);
dest->location      = (char *)malloc(MAX_STR);
dest->phone         = (char *)malloc(MAX_STR);

strcpy(dest->ssn,src->ssn);
strcpy(dest->firstName,src->firstName);
strcpy(dest->middleInitial,src->middleInitial);
strcpy(dest->lastName,src->lastName);
strcpy(dest->location,src->location);
strcpy(dest->phone,src->phone);
return(1);
}
```

```
/* frdb_svc_add.c : Remote database

      Add service procedures for Faculty.data*/


# include <stdio.h>

# include <string.h>

# include <rpc/rpc.h>

# include <errno.h>

# include <fcntl.h>

# include <unistd.h>

# include "frdb.h"


/**** Add Procedure for Faculty.data ******/

int *add_record_1();

int readFacultyRecord();

/******

extern record *pR1;

extern int fd1;

******/


record  *pR1;    /* For Faculty.data */

char ch[1000];

FILE  *fpp1;    /* For Faculty.data */

int fd1;



int *add_record_1(r)

record *r;

{

static  int  status;
```

```
char  buf[MAX_STR];
int i;
        if (0) printf("in *add_record.....\n");

fd1 = open(DATABASE,O_RDWR);    /* O_RDONLY */
if(fd1 < 0)
{
status = 0;
if (0)
                printf("file open error \n");
return((int *) &status); /* file open error */
  }

if (0)
printf("in *add_record....opened data file for reading.\n");
i = 0;
while(i = readFacultyRecord(i))
{
if (0)
printf("%s %s %s %s %s %s\n",pR1->ssn,pR1->firstName,
pR1->middleInitial,pR1->lastName,pR1->phone,
        pR1->location);
if(!strcmp(r->firstName,pR1->firstName) &&
(!strcmp(r->middleInitial,pR1->middleInitial)) &&
(!strcmp(r->lastName,pR1->lastName)) &&
(!strcmp(r->phone,pR1->phone)) &&
(!strcmp(r->location,pR1->location)))
{
/* duplicate record */
status = -2;
if (0) printf("duplicate record \n");
  return ((int *) &status);
}
}
/*
close(fd1);
fd1 = open(DATABASE,O_APPEND);
if(fd1 < 0)
{
status = 0;
if (0) printf("file open error \n");
return((int *) &status);  * file open error *
  }
*/
        if (0) printf("Writing in buf......\n");
sprintf(buf,"%s %s %s %s %s %s\n",r->ssn,
                r->firstName,r->middleInitial,
r->lastName,r->phone,r->location);
        if (0) printf("Writing in file......\n");
write(fd1,buf,strlen(buf));
  close(fd1);
status = 1;
        if (0) printf("status = %d\n",status);
  return ((int *) &status);
}

/*************Func. readRecord for Faculty.data *******************/

int readFacultyRecord(fpos)
int  fpos;
```

```
{
int  i;
char ch;
int ret,flag;
char  buf[MAX_STR];
        if (0) printf("In read record......\n");
if(!pR1)
{
pR1 = (record *)malloc(sizeof(record));
pR1->ssn = (char *)malloc(MAX_STR);
pR1->firstName = (char *)malloc(MAX_STR);
pR1->middleInitial = (char *)malloc(MAX_STR);
pR1->lastName = (char *)malloc(MAX_STR);
pR1->phone              = (char *)malloc(12 *
                                          sizeof(char));
pR1->location = (char *)malloc(MAX_STR);
pR1->next_record = NULL;
}

if (0)
 printf("In read record...memory allocated to pR1...\n");
if(lseek(fd1,fpos,SEEK_SET) == -1)
{
if (0) printf("lseek1 : %x \n",errno);
return(0);
}
i = flag = 0;
ret = read(fd1,&ch,1);
        if (0) printf("ret = %d\n",ret);
if(ret == 0)
return(0); /* end of file */
while( ch != '\n' && ch != EOF && ret > 0)
{
   buf[i++] = ch;
flag = 1;
        ret = read(fd1,&ch,1);
}
buf[i] = '\0';
if(sscanf(buf,"%s %s %s %s %s %s",pR1->ssn,
           pR1->firstName, pR1->middleInitial,
           pR1->lastName,pR1->phone, pR1->location) != 6)
return(0);
return((int)tell(fd1));
}
```

7.2.4 Course Building Blocks

```
/* crdb_svc_proc.c : Remote Course database
                     service procedures */

#include <stdio.h>
#include <string.h>
# include <rpc/rpc.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#include "crdb.h"
#include "frdb.h"

record1 *pR2;    /* For Course.data  */
char ch[1000];
char  buf[MAX_STR];
FILE  *fpp2;    /* For Course.data */
int fd2;

/****Func. readCRecord for Course.data ***********/

int readCRecord(fpos)
int  fpos;
{
int  i;
int ret,flag;
char ch;
        if (0) printf("In readcrecord.....\n");
   if (!pR2)
```

```
{
     pR2 = (record1 *) malloc(sizeof(record1));
if(pR2 == NULL)
return(0);
     pR2->course_number   = (char *) malloc(MAX_STR);
     pR2->course_section  = (char *) malloc(MAX_STR);
     pR2->course_semester = (char *) malloc(MAX_STR);
     pR2->course_name     = (char *) malloc(MAX_STR);
     pR2->course_inst     = (char *) malloc(MAX_STR);
     pR2->course_room     = (char *) malloc(MAX_STR);
     pR2->course_bldg     = (char *) malloc(MAX_STR);
     pR2->course_day      = (char *) malloc(MAX_STR);
     pR2->course_time     = (char *) malloc(MAX_STR);
pR2->course_year     = (char *) malloc(5*sizeof(char));
     pR2->course_credit   = (char *) malloc(MAX_STR);
        pR2->next_course = NULL;
   }
i = lseek(fd2,fpos,SEEK_SET);
if(i == -1)
{
if (0) printf("lseek2 : %x \n",errno);
return(0);
}
i = flag = 0;
ret = read(fd2,&ch,1);
if(ret == 0)
return(0); /* eof */
while(ch !=  '\n' && ch != EOF && ret > 0)
{
buf[i++] = ch;
```

```
ret = read(fd2,&ch,1);
flag = 1;
}
buf[i] = '\0';
if(sscanf(buf,"%s %s %s %s %s %s %s %s %s %s %s",
           pR2->course_number,  pR2->course_section,
           pR2->course_semester,pR2->course_name,
           pR2->course_inst,    pR2->course_room,
   pR2->course_bldg,    pR2->course_day,
           pR2->course_time,    pR2->course_year,
           pR2->course_credit) != 11)
return(0);
return ((int)tell(fd2));
}

/********Server Procedures for Course.data ***********/
record1  *coursename_key_1(name)
char **name;
{
if (0) printf("In course_name_key_1\n");
return( getC_record(*name,COURSENAME_KEY) ? (record1 *)pR2 :
(record1  *)NULL);
}

record1 *coursenumber_key_1(coursenumber)
char **coursenumber;
{
if (0) printf("in coursenumber........%s\n",*coursenumber);
return( getC_record(*coursenumber,COURSENUMBER_KEY) ?
                                          (record1 *)pR2 :
```

```
            (record1  *)NULL);
}



record1 *clist_record_1(record_number)
int **record_number;
{
int i;
fd2 = open(DATABASE1,O_RDONLY);
if(fd2 < 0)
{
if (0) printf("file open error : %s \n",DATABASE1);
return((record1 *)NULL);
}
pR2->file_offset = readCRecord(*record_number);
close(fd2);
return((record1  *)pR2);
}



record1 *coursename_record_1(rec)
record1  *rec;
{
int     i;
        fd2 = open(DATABASE1,O_RDONLY);
        if(fd2 < 0)
        {
         if (0) printf("file open error: %s \n",DATABASE1);
            return((record1 *)NULL);
        }
```

```
/*note rec?*/   i = rec->file_offset;
                if (0) printf("i = %d\n",i);
        while(i = readCRecord(i))
        {
        if (0) printf("%s\n",pR2->course_name);
        if ((!strcmp(rec->course_name,pR2->course_name)))
          {
                close(fd2);
                pR2->file_offset = i;
                return((record1  *)pR2);
           }
        }
        close(fd2);
        pR2->file_offset = i;
        return((record1  *)pR2);
}

record1 *coursenumber_record_1(rec)
record1  *rec;
{
int     i;
        fd2 = open(DATABASE1,O_RDONLY);
        if(fd2 < 0)
        {
         if (0) printf("file open error: %s \n",DATABASE1);
            return((record1 *)NULL);
        }
/*note rec?*/   i = rec->file_offset;
                if (0) printf("i = %d\n",i);
        while(i = readCRecord(i))
        {
        if (0) printf("%s\n",pR2->course_number);
        if ((!strcmp(rec->course_number,pR2->course_number)))
          {
                close(fd2);
                pR2->file_offset = i;
                return((record1  *)pR2);
           }
        }
        close(fd2);
        pR2->file_offset = i;
        return((record1  *)pR2);
}
/*******Func. getC_record for Course.data **********/


getC_record(c_string,c_key)
char *c_string;
int c_key;
{
int i;
int fflag;
fd2 = open(DATABASE1,O_RDONLY);
if(fd2 < 0)
{
if (0) printf("file open error : %s \n",DATABASE1);
return(0);
}
i = fflag = 0;
if (0) printf("coursename_key = %d \n",c_key);
while(i = readCRecord(i))
```

```
{
switch(c_key)
{
case COURSENAME_KEY :
if (0) printf("%s  %s\n",pR2->course_name,c_string);
if(!strcmp(pR2->course_name,c_string))
fflag = 1;
break;
case COURSENUMBER_KEY :
if(!strcmp(pR2->course_number,c_string))
fflag = 1;
break;
default :
break;
} /* end of SWITCH */
if(fflag)
break;
} /* end of WHILE */

close(fd2);
pR2->file_offset = i;
return(1);
}
```

```
/* crdb_svc_del.c : Remote Course database
              delete service procedures */

# include <stdio.h>
# include <string.h>
# include <rpc/rpc.h>
# include <errno.h>
# include <fcntl.h>
# include <unistd.h>
# include "crdb.h"

/*** Delete Procedure for Course.data *****/

int *cdel_record_1();

extern record1 *pR2;
extern int fd2;

int *cdel_record_1(coursenumber)
char **coursenumber;
{
static  int  status;
int i,cnt;
char  buf[MAX_STR];
record1 *prev,*head;
record1 *current,*temp;
record1 *pcurent;

fd2 = open(DATABASE1,O_RDONLY);
```

```
if(fd2 < 0)
{
status = 0;
if (0) printf("file open error \n");
return((int *) &status);
                /* file open error */
  }
prev = current = (record1 *)NULL;
temp = head = (record1 *)malloc(sizeof(record1));
        /* first : header */
temp->next_course = NULL;
i = cnt =  0;
while(i = readCRecord(i))
{
prev = temp;
temp->next_course = (record1 *)malloc
                              (sizeof(record1));
temp = temp->next_course;
temp->next_course = NULL;
++cnt;
copy_crecord(temp,pR2);
if (0)
         printf("temp : %x : %x : %s\n",
         temp,temp->next_course,temp->course_number);
if(!strcmp(temp->course_number,*coursenumber))
{
pcurent = prev;
current = temp;
}
}
```

```
close(fd2);
if(current == (record1 *)NULL)
{
status = 0;
if (0) printf("record not found \n");
}
else
{
if (0) printf("record found \n");
if (0)
                printf("course_number = %s : %d \n",
                 current->course_number,cnt);
pcurent->next_course = current->next_course;
temp = head->next_course;
fd2 = open(DATABASE1,O_WRONLY|O_TRUNC);
if(fd2 < 0)
{
status = 0;
if (0) printf("file open error \n");
return((int *) &status);
  }
i = 0;
while(temp != (record1 *)NULL)
{
++i;
if(i > cnt) break;
sprintf(buf,"%s %s %s %s %s %s %s %s
                              %s %s %s\n",
                 temp->course_number,  temp->course_section,
  temp->course_semester,temp->course_name,
                 temp->course_inst,    temp->course_room,
  temp->course_bldg,    temp->course_day,
  temp->course_time,    temp->course_year,
                 temp->course_credit);
write(fd2,buf,strlen(buf));
if (0) printf("i = %d \n",i);
if (0)
                printf("%s %s %s %s %s %s %s %s
                                %s %s %s\n",
                temp->course_number,  temp->course_section,
temp->course_semester,temp->course_name,
                temp->course_inst,    temp->course_room,
temp->course_bldg,    temp->course_day,
                temp->course_time,    temp->course_year,
                temp->course_credit);
temp = temp->next_course;
if (0) printf(" temp = %x \n",temp);
}
  close(fd2);
status = 1;
}
  return ((int *) &status);
}
copy_crecord(dest,src)
record1 *dest,*src;
{
dest->course_number   = (char *)malloc(MAX_STR);
dest->course_section  = (char *)malloc(MAX_STR);
dest->course_semester = (char *)malloc(MAX_STR);
dest->course_name     = (char *)malloc(MAX_STR);
```

```
dest->course_inst      = (char *)malloc(MAX_STR);
dest->course_room      = (char *)malloc(MAX_STR);
dest->course_bldg      = (char *)malloc(MAX_STR);
dest->course_day       = (char *)malloc(MAX_STR);
dest->course_time      = (char *)malloc(MAX_STR);
dest->course_year      = (char *)malloc(MAX_STR);
dest->course_credit    = (char *)malloc(MAX_STR);

strcpy(dest->course_number,src->course_number);
strcpy(dest->course_section,src->course_section);
strcpy(dest->course_semester,src->course_semester);
strcpy(dest->course_name,src->course_name);
strcpy(dest->course_inst,src->course_inst);
strcpy(dest->course_room,src->course_room);
strcpy(dest->course_bldg,src->course_bldg);
strcpy(dest->course_day,src->course_day);
strcpy(dest->course_time,src->course_time);
strcpy(dest->course_year,src->course_year);
strcpy(dest->course_credit,src->course_credit);

return(1);
}
```

```
/* crdb_svc_add.c : Remote Course database
                  Add service procedures */

# include <stdio.h>
# include <string.h>
# include <rpc/rpc.h>
# include <errno.h>
# include <fcntl.h>
# include <unistd.h>
# include "crdb.h"

/***** Add Procedure for Course.data *****/
int *cadd_record_1();

extern record1 *pR2;
extern int fd2;

int *cadd_record_1(r1)
record1 *r1;
{
static  int  status;
char  buf[MAX_STR];
int i;
fd2 = open(DATABASE1,O_RDWR);    /* O_RDONLY */
if(fd2 < 0)
{
status = 0;
if (0) printf("file open error \n");
return((int *) &status);
```

```
                /* file open error */
  }
i = 0;
while(i = readCRecord(i))
{
if (0)
        printf("%s %s %s %s %s %s %s %s %s %s %s\n",
           pR2->course_number,  pR2->course_section,
           pR2->course_semester,pR2->course_name,
           pR2->course_inst,    pR2->course_room,
           pR2->course_bldg,    pR2->course_day,
           pR2->course_time,    pR2->course_year,
           pR2->course_credit);

if(!strcmp(r1->course_number,pR2->course_number) &&
  (!strcmp(r1->course_section,pR2->course_section)) &&
  (!strcmp(r1->course_semester,pR2->course_semester))&&
  (!strcmp(r1->course_year,pR2->course_year)))
{
/* duplicate record */
status = -2;
if (0) printf("duplicate record \n");
  return ((int *) &status);
}
        }
/*
close(fd2);
fd2 = open(DATABASE1,O_APPEND);
if(fd2 < 0)
{
```

```
status = 0;
if (0) printf("file open error \n");
return((int *) &status);  * file open error *
  }
*/
sprintf(buf,"%s %s %s %s %s %s %s %s %s %s %s\n",
                r1->course_number,  r1->course_section,
r1->course_semester,r1->course_name,
                r1->course_inst,    r1->course_room,
                r1->course_bldg,    r1->course_day,
                r1->course_time,    r1->course_year,
                r1->course_credit);
write(fd2,buf,strlen(buf));
  close(fd2);
status = 1;
  return ((int *) &status);
}
```

```
/* facourse_list.c : Remote Course database
                     service procedures */

#include <stdio.h>
#include <string.h>
# include <rpc/rpc.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#include "crdb.h"
#include "frdb.h"

record1 *pR2;    /* For Course.data  */
char ch[1000];
char  buf[MAX_STR];
FILE  *fpp2;    /* For Course.data */
int fd2;

/******Func. readCRecord for Course.data ****/

int readCLRecord(fpos)
int  fpos;
{
int  i;
int ret,flag;
char ch;
        if (0) printf("In readcrecord.....\n");
   if (!pR2)
```

```
{
   pR2 = (record1 *) malloc(sizeof(record1));
if(pR2 == NULL)
return(0);
     pR2->course_number   = (char *) malloc(MAX_STR);
     pR2->course_section  = (char *) malloc(MAX_STR);
     pR2->course_semester = (char *) malloc(MAX_STR);
     pR2->course_name     = (char *) malloc(MAX_STR);
     pR2->course_inst     = (char *) malloc(MAX_STR);
     pR2->course_room     = (char *) malloc(MAX_STR);
     pR2->course_bldg     = (char *) malloc(MAX_STR);
     pR2->course_day      = (char *) malloc(MAX_STR);
     pR2->course_time     = (char *) malloc(MAX_STR);
pR2->course_year     = (char *) malloc(5*sizeof(char));
     pR2->course_credit   = (char *) malloc(MAX_STR);
        pR2->next_course = NULL;
   }
i = lseek(fd2,fpos,SEEK_SET);
if(i == -1)
{
if (0) printf("lseek2 : %x \n",errno);
return(0);
}
i = flag = 0;
ret = read(fd2,&ch,1);
if(ret == 0)
return(0); /* eof */
while(ch !=  '\n' && ch != EOF && ret > 0)
{
buf[i++] = ch;
```

```
ret = read(fd2,&ch,1);
flag = 1;
}
buf[i] = '\0';
if(sscanf(buf,"%s %s %s %s %s %s %s %s %s %s %s",
           pR2->course_number,  pR2->course_section,
           pR2->course_semester,pR2->course_name,
           pR2->course_inst,    pR2->course_room,
           pR2->course_bldg,    pR2->course_day,
           pR2->course_time,    pR2->course_year,
           pR2->course_credit) != 11)
return(0);
return ((int)tell(fd2));
}


/******Server Procedures for Course.data ***********/
record1 *get_course_1(rec)
record1 *rec;
{
int    i;
fd2 = open(DATABASE1,O_RDONLY);
if(fd2 < 0)
{
if (0)
                printf("file open error: %s \n",DATABASE1);
return((record1 *)NULL);
}
i = rec->file_offset;
while(i = readCLRecord(i))
{
```

```
if(!strcmp(rec->course_inst,pR2->course_inst))
{
close(fd2);
pR2->file_offset = i;
return((record1  *)pR2);
}
}
close(fd2);
pR2->file_offset = i;
   return((record1  *)pR2);
}
```

7.2.5 Student Building Blocks

```
/* srdb_svc_proc.c : Remote database service
                     procedures for Student.data*/

#include <stdio.h>
#include <string.h>
# include <rpc/rpc.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#include "srdb.h"

record2 *pR3;    /* For Student.data */
char ch[1000];
char  buf[MAX_STR];
FILE    *fpp3;    /* For Student.data */
int fd3;

/****Func.readSRecord for Student.data ************/

int readSRecord(fpos)
int  fpos;
{
int  i;
char ch;
int ret,flag;

if(!pR3)
{
```

```
pR3 = (record2 *)malloc(sizeof(record2));
pR3->stud_ssn     = (char *)malloc(MAX_STR);
pR3->stud_firstName = (char *)malloc(MAX_STR);
pR3->stud_middleInitial = (char *)malloc(MAX_STR);
pR3->stud_lastName = (char *)malloc(MAX_STR);
pR3->stud_address = (char *)malloc(MAX_STR);
pR3->stud_city   = (char *)malloc(MAX_STR);
pR3->stud_state   = (char *)malloc(MAX_STR);
pR3->stud_zip     = (char *)malloc(MAX_STR);
pR3->stud_phone         = (char *)malloc(12 * sizeof(char));
pR3->stud_major   = (char *)malloc(MAX_STR);
pR3->stud_college = (char *)malloc(MAX_STR);
pR3->stud_gpa     = (char *)malloc(MAX_STR);
pR3->next_record = NULL;
}


if(lseek(fd3,fpos,SEEK_SET) == -1)
{
if (0)
printf("lseek3 : %x \n",errno);
return(0);
}
i = flag = 0;
ret = read(fd3,&ch,1);
if(ret == 0)
return(0); /* end of file */
while( ch != '\n' && ch != EOF && ret > 0)
{
   buf[i++] = ch;
flag = 1;
```

```
ret = read(fd3,&ch,1);
}
buf[i] = '\0';
if(sscanf(buf,"%s %s %s %s %s %s %s
                       %s %s %s %s %s ",
         pR3->stud_ssn,pR3->stud_firstName,
         pR3->stud_middleInitial,pR3->stud_lastName,
         pR3->stud_address,pR3->stud_city,pR3->stud_state,
         pR3->stud_zip,pR3->stud_phone,pR3->stud_major,
         pR3->stud_college,pR3->stud_gpa) != 12)
         return(0);
return((int)tell(fd3));
}


/****Server procedures for Student.data *************/


record2  *sfirstname_key_1(name)
char **name;
{
return( getS_record(*name,SFIRSTNAME_KEY) ? (record2 *)pR3 :
(record2  *)NULL);
}


record2 *sssn_key_1(ssnumber)
char **ssnumber;
{
return( getS_record(*ssnumber,SSSN_KEY) ? (record2 *)pR3 :
(record2  *)NULL);
}
```

```
record2 *slastname_key_1(name)
char **name;
{
return( getS_record(*name,SLASTNAME_KEY) ? (record2 *)pR3 :
(record2  *)NULL);
}


record2 *sphone_key_1(phnumber)
char **phnumber;
{
return( getS_record(*phnumber,SPHONE_KEY) ? (record2 *)pR3 :
(record2  *)NULL);
}


record2 *slist_record_1(record_number)
int **record_number;
{
int i;
fd3 = open(DATABASE2,O_RDONLY);
if(fd3 < 0)
{
if (0)
printf("file open error : %s \n",DATABASE2);
return((record2 *)NULL);
}
pR3->file_offset = readSRecord(*record_number);
close(fd3);
return((record2  *)pR3);
}
```

```
record2 *sfirstname_record_1(rec)
record2  *rec;
{
int     i;
        fd3 = open(DATABASE2,O_RDONLY);
        if(fd3 < 0)
        {
if (0)
              printf("file open error: %s \n",DATABASE2);
              return((record2 *)NULL);
        }
/*note rec?*/   i = rec->file_offset;
if (0)
                printf("i = %d\n",i);
        while(i = readSRecord(i))
        {
            if ((!strcmp(rec->stud_firstName,pR3->stud_firstName)))
                {
                        close(fd3);
                        pR3->file_offset = i;
                        return((record2  *)pR3);
                }
        }
        close(fd3);
        pR3->file_offset = i;
        return((record2 *)pR3);
}


record2 *slastname_record_1(rec)
```

```
record2  *rec;
{
int     i;
        fd3 = open(DATABASE2,O_RDONLY);
        if(fd3 < 0)
        {
if (0)
            printf("file open error: %s \n",DATABASE2);
            return((record2 *)NULL);
        }
/*note rec?*/   i = rec->file_offset;
if (0)
                printf("i = %d\n",i);
        while(i = readSRecord(i))
        {
         if ((!strcmp(rec->stud_lastName,pR3->stud_lastName)))
                {
                        close(fd3);
                        pR3->file_offset = i;
                        return((record2  *)pR3);
                }
        }
        close(fd3);
        pR3->file_offset = i;
        return((record2 *)pR3);
}

/******* getS_record for Student.data **********/
getS_record(c_string,c_key)
char *c_string;
```

```
int c_key;
{
int i;
int fflag;
fd3 = open(DATABASE2,O_RDONLY);
if(fd3 < 0)
{
if (0)
printf("file open error : %s \n",DATABASE2);
return(0);
}
i = fflag = 0;
while(i = readSRecord(i))
{
switch(c_key)
{
case SFIRSTNAME_KEY :
        if(!strcmp(pR3->stud_firstName,c_string))
fflag = 1;
break;
case SLASTNAME_KEY :
if(!strcmp(pR3->stud_lastName,c_string))
fflag = 1;
break;
case SPHONE_KEY :
if(!strcmp(pR3->stud_phone,c_string))
fflag = 1;
break;
case SSSN_KEY :
if(!strcmp(pR3->stud_ssn,c_string))
fflag = 1;
break;
default :
break;
} /* end of SWITCH */
if(fflag)
break;
} /* end of WHILE */

close(fd3);
pR3->file_offset = i;
return(1);
}
```

```
/* srdb_svc_del.c : Remote Student database

                  Delete service procedures */


# include <stdio.h>

# include <string.h>

# include <rpc/rpc.h>

# include <errno.h>

# include <fcntl.h>

# include <unistd.h>

# include "srdb.h"


/**** Delete Procedure for Student.data ****/


int *sdel_record_1();


extern record2 *pR3;

extern int fd3;


int *sdel_record_1(ssnumber)

char **ssnumber;

{

static  int  status;

int i,cnt;

char  buf[MAX_STR];

record2 *prev,*head;

record2 *current,*temp;

record2 *pcurent;


fd3 = open(DATABASE2,O_RDONLY);
```

```
if(fd3 < 0)
{
status = 0;
if (0)
printf("file open error \n");
return((int *) &status);
                  /* file open error */
  }
prev = current = (record2 *)NULL;
temp = head = (record2 *)malloc(sizeof(record2));
                /* first : header */
temp->next_record = NULL;
i = cnt =  0;
while(i = readSRecord(i))
{
prev = temp;
temp->next_record = (record2 *)
                 malloc(sizeof(record2));
temp = temp->next_record;
temp->next_record = NULL;
++cnt;
copy_srecord(temp,pR3);
if (0)
printf("temp : %x : %x : %s\n",temp,
                temp->next_record,temp->stud_ssn);
if(!strcmp(temp->stud_ssn,*ssnumber))
{
pcurent = prev;
current = temp;
}
```

```
}
close(fd3);
if(current == (record2 *)NULL)
{
status = 0;
if (0)
printf("record not found \n");
}
else
{
if (0)
printf("record found \n");
if (0)
printf("ssn = %s : %d \n",current->stud_ssn,cnt);
pcurent->next_record = current->next_record;
temp = head->next_record;
fd3 = open(DATABASE2,O_WRONLY|O_TRUNC);
if(fd3 < 0)
{
status = 0;
if (0)
printf("file open error \n");
return((int *) &status);
  }
i = 0;
while(temp != (record2 *)NULL)
{
++i;
if(i > cnt) break;
sprintf(buf,"%s %s %s %s %s %s %s %s
                    %s %s %s %s\n",
              temp->stud_ssn,temp->stud_firstName,
        temp->stud_middleInitial,temp->stud_lastName,
              temp->stud_address,temp->stud_city,
              temp->stud_state,temp->stud_zip,
              temp->stud_phone,temp->stud_major,
              temp->stud_college,temp->stud_gpa);
write(fd3,buf,strlen(buf));
if (0)
printf("i = %d \n",i);
if (0)
printf("%s %s %s %s %s %s %s %s
               %s %s %s %s\n",
               temp->stud_ssn,temp->stud_firstName,
         temp->stud_middleInitial,temp->stud_lastName,
               temp->stud_address,temp->stud_city,
               temp->stud_state,temp->stud_zip,
               temp->stud_phone,temp->stud_major,
               temp->stud_college,temp->stud_gpa);


        temp = temp->next_record;
if (0)
printf(" temp = %x \n",temp);
}
  close(fd3);
status = 1;
}
  return ((int *) &status);
}
copy_srecord(dest,src)
```

```
record2 *dest,*src;
{
dest->stud_ssn           = (char *)malloc(MAX_STR);
dest->stud_firstName     = (char *)malloc(MAX_STR);
dest->stud_middleInitial = (char *)malloc(MAX_STR);
dest->stud_lastName      = (char *)malloc(MAX_STR);
dest->stud_address       = (char *)malloc(MAX_STR);
dest->stud_city          = (char *)malloc(MAX_STR);
dest->stud_state         = (char *)malloc(MAX_STR);
dest->stud_zip           = (char *)malloc(MAX_STR);
dest->stud_phone         = (char  *)malloc(MAX_STR);
dest->stud_major         = (char *)malloc(MAX_STR);
dest->stud_college       = (char *)malloc(MAX_STR);
dest->stud_gpa           = (char *)malloc(MAX_STR);

strcpy(dest->stud_ssn,src->stud_ssn);
strcpy(dest->stud_firstName,src->stud_firstName);
strcpy(dest->stud_middleInitial,src->stud_middleInitial);
strcpy(dest->stud_lastName,src->stud_lastName);
strcpy(dest->stud_address,src->stud_address);
strcpy(dest->stud_city,src->stud_city);
strcpy(dest->stud_state,src->stud_state);
strcpy(dest->stud_zip,src->stud_zip);
strcpy(dest->stud_phone,src->stud_phone);
strcpy(dest->stud_major,src->stud_major);
strcpy(dest->stud_college,src->stud_college);
strcpy(dest->stud_gpa,src->stud_gpa);
return(1);
}
```

```
/* srdb_svc_add.c : Remote Student database
                   Add service procedures */

# include <stdio.h>
# include <string.h>
# include <rpc/rpc.h>
# include <errno.h>
# include <fcntl.h>
# include <unistd.h>
# include "srdb.h"

/***** Add Procedure for Student.data *********/
int *sadd_record_1();

extern record2 *pR3;
extern int fd3;

int *sadd_record_1(r2)
record2 *r2;
{
static  int  status;
char  buf[MAX_STR];
int i;
fd3 = open(DATABASE2,O_RDWR);    /* O_RDONLY */
if(fd3 < 0)
{
status = 0;
if (0)
printf("file open error \n");
```

```
return((int *) &status);
                /* file open error */
  }
i = 0;
while(i = readSRecord(i))
{
if (0)
printf("%s %s %s %s %s %s %s %s %s %s %s %s \n",
        pR3->stud_ssn,pR3->stud_firstName,
pR3->stud_middleInitial,pR3->stud_lastName,
        pR3->stud_address,pR3->stud_city,pR3->stud_state,
        pR3->stud_zip, pR3->stud_phone,pR3->stud_major,
        pR3->stud_college,pR3->stud_gpa);
if(!strcmp(r2->stud_ssn,pR3->stud_ssn))
{
/* duplicate record */
status = -2;
if (0)
printf("duplicate record \n");
  return ((int *) &status);
}
}
/*
close(fd3);
fd3 = open(DATABASE2,O_APPEND);
if(fd3 < 0)
{
status = 0;
if (0)
printf("file open error \n");
```

```
return((int *) &status);  * file open error *
  }
*/
sprintf(buf,"%s %s %s %s %s %s %s %s %s
                     %s %s %s \n",
        r2->stud_ssn,r2->stud_firstName,
r2->stud_middleInitial,r2->stud_lastName,
        r2->stud_address,r2->stud_city,r2->stud_state,
        r2->stud_zip,r2 ->stud_phone,r2->stud_major,
        r2->stud_college,r2->stud_gpa);
write(fd3,buf,strlen(buf));
  close(fd3);
status = 1;
  return ((int *) &status);
}
```

7.2.6 Register Building Blocks

```
/* rrdb_svc_proc.c : Remote Register database
                     service procedures */

#include <stdio.h>
#include <string.h>
# include <rpc/rpc.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#include "rrdb.h"

record3 *pR4;    /* For Register.data  */
char c,ch[1000];
char  buf[MAX_STR];
FILE  *fpp4;    /* For Register.data */
int fd4;

/***Func. readRRecord for Register.data *****/

int readRRecord(fpos)
int  fpos;
{
int  i;
int ret,flag;
char ch;
   if (!pR4)
{
if (0)
```

```
            printf("In readRRecord.....\n");
        pR4 = (record3 *) malloc(sizeof(record3));
    if(pR4 == NULL)
    return(0);
        pR4->stud_ssn        = (char *) malloc(MAX_STR);
        pR4->course_number   = (char *) malloc(MAX_STR);
        pR4->course_section  = (char *) malloc(MAX_STR);
        pR4->course_semester = (char *) malloc(MAX_STR);
    pR4->course_year     = (char *) malloc(5*sizeof(char));
        pR4->grade           = (char *) malloc(MAX_STR);
      }
    i = lseek(fd4,fpos,SEEK_SET);
    if(i == -1)
    {
    printf("lseek4 : %x \n",errno);
    c = getchar();
    return(0);
    }
    i = flag = 0;
    ret = read(fd4,&ch,1);
    if(ret == 0)
    return(0); /* eof */
    while(ch !=  '\n' && ch != EOF && ret > 0)
    {
    buf[i++] = ch;
    ret = read(fd4,&ch,1);
    flag = 1;
    }
    buf[i] = '\0';
    if(sscanf(buf,"%s %s %s %s %s %s ",
          pR4->stud_ssn,pR4->course_number,
  pR4->course_section,pR4->course_semester,
          pR4->course_year, pR4->grade) != 6)
return(0);
return ((int)tell(fd4));
}

/******Server Procedures for Register.data ***********/


record3 *grade_record_1(rec)
record3  *rec;
{
int    i;
fd4 = open(DATABASE3,O_RDONLY);
if(fd4 < 0)
{
if (0)
printf("file open error: %s \n",DATABASE3);
return((record3 *)NULL);
}
/*note rec?*/ i = rec->file_offset;
if (0)
                printf("i = %d\n",i);
while(i = readRRecord(i))
{
if (0)
printf("%s  %s  %s  %s  %s  %s \n",
              pR4->stud_ssn,pR4->course_number,
              pR4->course_section,pR4->course_semester,
              pR4->course_year, pR4->grade);
```

```
if ((!strcmp(rec->course_number,pR4->course_number)) &&
   (!strcmp(rec->course_section,pR4->course_section)) &&
   (!strcmp(rec->course_semester,pR4->course_semester)) &&
   (!strcmp(rec->course_year,pR4->course_year)) &&
   (!strcmp(rec->grade,pR4->grade)))
{
close(fd4);
pR4->file_offset = i;
return((record3  *)pR4);
}
}
close(fd4);
pR4->file_offset = i;
   return((record3  *)pR4);
}


record3 *rcourse_record_1(rec)
record3  *rec;
{
int    i;
fd4 = open(DATABASE3,O_RDONLY);
if(fd4 < 0)
{
if (0)
printf("file open error: %s \n",DATABASE3);
return((record3 *)NULL);
}
/*note rec?*/ i = rec->file_offset;
while(i = readRRecord(i))
```

```
{
if ((!strcmp(rec->course_number,pR4->course_number)) &&
  (!strcmp(rec->course_section,pR4->course_section)) &&
  (!strcmp(rec->course_semester,pR4->course_semester)) &&
  (!strcmp(rec->course_year,pR4->course_year)))
{
close(fd4);
pR4->file_offset = i;
return((record3  *)pR4);
}
}
close(fd4);
pR4->file_offset = i;
   return((record3  *)pR4);
}



record3  *rssn_record_1(rec)
record3  *rec;
{
int    i;
fd4 = open(DATABASE3,O_RDONLY);
if(fd4 < 0)
{
if (0)
printf("file open error: %s \n",DATABASE3);
return((record3 *)NULL);
}
/*note rec?*/ i = rec->file_offset;
while(i = readRRecord(i))
```

```
{
if(!strcmp(rec->stud_ssn,pR4->stud_ssn))
{
close(fd4);
pR4->file_offset = i;
return((record3  *)pR4);
}
}
close(fd4);
pR4->file_offset = i;
   return((record3  *)pR4);
}


record3 *rlist_record_1(record_number)
int **record_number;
{
int i;
fd4 = open(DATABASE3,O_RDONLY);
if(fd4 < 0)
{
if (0)
printf("file open error : %s \n",DATABASE3);
return((record3 *)NULL);
}
pR4->file_offset = readRRecord(*record_number);
close(fd4);
return((record3  *)pR4);
}


record3 *rssn_key_1(ssnumber)
```

```
char    **ssnumber;
{
if (0)
        printf("ssn = %s\n",ssnumber);
 return( getR_record(*ssnumber,1) ? (record3 *)pR4 :
                                   (record3  *)NULL);
}
/***************** getR_record for Register.data **********/
getR_record(c_string,c_key)
char *c_string;
int c_key;
{
int i;
int fflag;
fd4 = open(DATABASE3,O_RDONLY);
if(fd4 < 0)
{
if (0)
printf("file open error : %s \n",DATABASE3);
return(0);
}
i = fflag = 0;
while(i = readRRecord(i))
{
switch(c_key)
{
case 1 :
if(!strcmp(pR4->stud_ssn,c_string))
fflag = 1;
break;
/* case COURSE_KEY :
if(!strcmp(pR4->course_number,c_string))
fflag = 1;
break;
*/
default :
break;
} /* end of SWITCH */
if(fflag)
break;
} /* end of WHILE */

close(fd4);
pR4->file_offset = i;
return(1);
}
```

```
/* rrdb_svc_del.c : Remote Register
            database delete service procedures */

# include <stdio.h>
# include <string.h>
# include <rpc/rpc.h>
# include <errno.h>
# include <fcntl.h>
# include <unistd.h>
# include "rrdb.h"

/**Delete Procedure for Student.data *****/

int *rdel_record_1();

extern record3*pR4;
extern int fd4;

int *rdel_record_1(ssnumber)
char **ssnumber;
{
static  int  status;
int i,cnt;
char  buf[MAX_STR];
record3 *prev,*head;
record3 *current,*temp;
record3 *pcurent;

fd4 = open(DATABASE3,O_RDONLY);
```

```
if(fd4 < 0)
{
status = 0;
if (0)
printf("file open error \n");
return((int *) &status);
                /* file open error */
  }
prev = current = (record3*)NULL;
temp = head = (record3*)malloc(sizeof(record3));
             /* first : header */
temp->next_record = NULL;
i = cnt =  0;
while(i = readRRecord(i))
{
prev = temp;
temp->next_record = (record3*)malloc(sizeof(record3));
temp = temp->next_record;
temp->next_record = NULL;
++cnt;
copy_rrecord(temp,pR4);
if (0)
printf("temp : %x : %x : %s\n",temp,temp->next_record,temp->stud_ssn);
if(!strcmp(temp->stud_ssn,*ssnumber))
{
pcurent = prev;
current = temp;
}
}
close(fd4);
```

```
if(current == (record3*)NULL)
{
status = 0;
if (0)
printf("record not found \n");
}
else
{
if (0)
printf("record found \n");
if (0)
printf("ssn = %s : %d \n",current->stud_ssn,cnt);
pcurent->next_record = current->next_record;
temp = head->next_record;
fd4 = open(DATABASE3,O_WRONLY|O_TRUNC);
if(fd4 < 0)
{
status = 0;
if (0)
printf("file open error \n");
return((int *) &status);
  }
i = 0;
while(temp != (record3*)NULL)
{
++i;
if(i > cnt) break;
sprintf(buf,"%s %s %s %s %s %s\n",
                temp->stud_ssn,temp->course_number,
        temp->course_section,temp->course_semester,
                temp->course_year,temp->grade);
write(fd4,buf,strlen(buf));
if (0)
printf("i = %d \n",i);


        temp = temp->next_record;
if (0)
printf(" temp = %x \n",temp);
}
  close(fd4);
status = 1;
}
  return ((int *) &status);
}
copy_rrecord(dest,src)
record3*dest,*src;
{
dest->stud_ssn        = (char *)malloc(MAX_STR);
dest->course_number   = (char *)malloc(MAX_STR);
dest->course_section  = (char *)malloc(MAX_STR);
dest->course_semester = (char *)malloc(MAX_STR);
dest->course_year     = (char *)malloc(MAX_STR);
dest->grade           = (char *)malloc(MAX_STR);

strcpy(dest->stud_ssn,src->stud_ssn);
strcpy(dest->course_number,src->course_number);
strcpy(dest->course_section,src->course_section);
strcpy(dest->course_semester,src->course_semester);
strcpy(dest->course_year,src->course_year);
strcpy(dest->grade,src->grade);
```

```
return(1);
}
```

```
/* rrdb_svc_add.c : Remote Register database
                    Add service procedures */

# include <stdio.h>
# include <string.h>
# include <rpc/rpc.h>
# include <errno.h>
# include <fcntl.h>
# include <unistd.h>
# include "rrdb.h"

/**Add Procedure for Course.data *******/
int *radd_record_1();

extern record3 *pR4;
extern int fd4;

int *radd_record_1(r3)
record3 *r3;
{
static  int  status;
char  buf[MAX_STR];
int i;
fd4 = open(DATABASE3,O_RDWR);
              /* O_RDONLY */
if(fd4 < 0)
{
status = 0;
if (0)
```

```
printf("file open error \n");
return((int *) &status);
                /* file open error */
  }
i = 0;
while(i = readRRecord(i))
{
if (0)
printf("%s %s %s %s %s %s \n",pR4->stud_ssn,
               pR4->course_number,pR4->course_section,
pR4->course_semester,pR4->course_year,
                pR4->grade);
if((!strcmp(r3->stud_ssn,pR4->stud_ssn)) &&
(!strcmp(r3->course_number,pR4->course_number)) &&
(!strcmp(r3->course_section,pR4->course_section)) &&
(!strcmp(r3->course_year,pR4->course_year)))
{
/* duplicate record */
status = -2;
if (0)
printf("duplicate record \n");
  return ((int *) &status);
}
         }
/*
close(fd4);
fd4 = open(DATABASE3,O_APPEND);
if(fd4 < 0)
{
status = 0;
```

```
if (0)
printf("file open error \n");
return((int *) &status);  * file open error *
  }
*/
sprintf(buf,"%s %s %s %s %s %s\n", r3->stud_ssn,
                r3->course_number,r3->course_section,
r3->course_semester,r3->course_year,
                r3->grade);
write(fd4,buf,strlen(buf));
  close(fd4);
status = 1;
  return ((int *) &status);
}
```

7.3 Source Code for the TRADER

```
/* create_h.c : Create Client Handles for server processes */

# include <stdio.h>
# include <ctype.h>
# include <rpc/rpc.h>
# include "frdb.h"
# include "rrdb.h"
# include "crdb.h"
# include "srdb.h"

# define SERVERF "newark"
# define SERVERC "newark"
# define SERVERS "pluto"
# define SERVERR "pluto"

CLIENT *clf;                        /* A client handle */
CLIENT *cls;                        /* A client handle */
CLIENT *clr;                        /* A client handle */
CLIENT *clc;                        /* A client handle */
char c;


void create_handle()
{
 initialize_f(SERVERF);
if (0)
 printf("Client handle created...clf = %d,for  %s \n",clf,SERVERF);
 initialize_c(SERVERC);
if (0)
```

```
 printf("Client handle created...clc = %d,for  %s \n",clc,SERVERC);
 initialize_s(SERVERS);
if (0)
 printf("Client handle created...cls = %d,for  %s \n",cls,SERVERS);
 initialize_r(SERVERR);
if (0)
 printf("Client handle created...clr = %d,for  %s \n",clr,SERVERR);

} /* end of MAIN */

/*
initialize : Create client Handles for the various servers
*/
initialize_f(serv_mc)
char *serv_mc;
{
    extern CLIENT  *clf;
   if (!(clf = clnt_create(serv_mc, FRDBPROG, FRDBVERS, "tcp")))
     {
      clnt_pcreateerror(serv_mc);
                system("clear");
      printf("Client Handle not created; serverf missing. \n");
      c = getchar();
                return;

     }
if (0)
printf("in initf--Client handle created...clf = %d,for  %s \n",clf,serv_mc);
}
```

```
initialize_c(serv_mc)
char *serv_mc;
{
extern    CLIENT  *clc;
   if (!(clc = clnt_create(serv_mc, CRDBPROG, CRDBVERS, "tcp")))
     {
      clnt_pcreateerror(serv_mc);
                system("clear");
      printf("Client Handle not created; serverc missing. \n");
      c = getchar();
                return;
     }
if (0)
printf("in initc--Client handle created...clc = %d,for  %s \n",clc,serv_mc);
}


initialize_s(serv_mc)
char *serv_mc;
{
extern    CLIENT  *cls;
   if (!(cls = clnt_create(serv_mc, SRDBPROG, SRDBVERS, "tcp")))
     {
      clnt_pcreateerror(serv_mc);
                system("clear");
      printf("Client Handle not created; servers missing. \n");
      c = getchar();
                return;
     }
if (0)
printf("in inits--Client handle created...cls = %d,for  %s \n",cls,serv_mc);
}


initialize_r(serv_mc)

char *serv_mc;

{

extern    CLIENT  *clr;

   if (!(clr = clnt_create(serv_mc, RRDBPROG, RRDBVERS, "tcp")))

     {

      clnt_pcreateerror(serv_mc);

                system("clear");

      printf("Client Handle not created; serverr missing. \n");

      c = getchar();

                return;

     }

if (0)

printf("in initr--Client handle created...clr = %d,for  %s \n",clr,serv_mc);

}
```

```
/* rdb1.c : Client application for rdb */

# include <stdio.h>
# include <ctype.h>
# include <rpc/rpc.h>
# include "frdb.h"
# include "rrdb.h"
# include "crdb.h"
# include "srdb.h"

trader ()
{
 extern int key;
 extern CLIENT *clf;
 extern CLIENT *clc;
 extern CLIENT *cls;
 extern CLIENT *clr;
 extern char bufdata[256];
 char *value;

if (0){
     printf("In trader....\n");
     printf("key = %d\n",key);
     printf("In Trader Data = %s\n",bufdata);
}
     value = bufdata;

    switch (key)
    {
```

```
/************Cases for Faculty Database **********/
     case FIRSTNAME_RECORD:
faculty_first_name(value,clf);
     break;

case SSN_KEY :
faculty_ssn(value,clf);
     break;

case LASTNAME_RECORD :
faculty_last_name(value,clf);
     break;

case PHONE_KEY :
faculty_phone(value,clf);
     break;

case LOCATION_KEY :
faculty_location(value,clf);
     break;

case ADD_RECORD :
if (0)
                                printf("In Trader -add_record Data = %s\n",
                                        bufdata);
faculty_add_record(value,clf);
/**faculty_add_record(clf);****/
break;

case DEL_RECORD :
```

```
faculty_del_record(value,clf);
break;

case LIST_RECORD :
faculty_list_record(value,clf);
break;

/************Cases for Course Database **********/

case GET_COURSE:
if (0)
                                printf("value = %s\n",value);
course_get_course(value,clc,clf);
break;

     case COURSENAME_RECORD:
course_name(value,clc);
     break;

case COURSENUMBER_RECORD:
course_number(value,clc);
     break;

case CLIST_RECORD :
course_list_record(value,clc);
     break;

case CADD_RECORD :
if (0)
                                printf("value = %s\n",value);
```

```
course_add_record(value,clc);
     break;


case CDEL_RECORD :
if (0)
                                printf("value = %s\n",value);
course_del_record(value,clc);
     break;


/************Cases for Student Database **********/
     case SFIRSTNAME_RECORD:
student_first_name(value,cls);
break;


case SSSN_KEY :
student_ssn(value,cls);
     break;


case SLASTNAME_RECORD :
student_last_name(value,cls);
     break;


case SPHONE_KEY :
student_phone(value,cls);
     break;


case SADD_RECORD :
student_add_record(value,cls);
     break;
```

```
case SDEL_RECORD :
student_del_record(value,cls);
     break;

case SLIST_RECORD :
student_list_record(value,cls);
     break;

/*********************** Register Database *****************/
case GRADE_RECORD :
register_grade(value,clr);
break;

case RCOURSE_RECORD :
register_course(value,clr);
break;

case RSSN_RECORD :
register_ssn(value,clr);
break;

case RLIST_RECORD :
register_list_record(value,clr);
break;

case RADD_RECORD :
register_add_record(value,clr);
break;

case RDEL_RECORD :
```

```
register_del_record(value,clr);
break;


    default:
    printf("Unknown remote procedure\n");
                                break;
  } /* end of SWITCH */
} /* end of MAIN */
```

```
/******rdb_faculty.c : Client applications for faculty database Services */

# include <stdio.h>

# include <ctype.h>

# include <rpc/rpc.h>

#include "frdb.h"

# include "crdb.h"



static record  *pRF;


char c;


record   *temp;

record  *head;


/******extern  msg_memo();**/

extern  int *add_record_1();

void  faculty_phone();

void  faculty_location();

void  faculty_add_record();

void  faculty_del_record();

void  faculty_list_record();

void  faculty_ssn();

void    initialize1_f();


/********

void  faculty_first_name(value,cl)

char  *value;
```

```
CLIENT *cl;
{
do {
     temp = firstname_key_1(&value, cl);
if(!temp)
                                  {
                                        system("clear");
printf("FIRSTNAME_KEY : null ptr \n");
                                        c = getchar();
                                        return;
                                  }
else
if(temp->file_offset < 1)
                                  {
                                        system("clear");
printf("data not found...\n");
                                        c = getchar();
                                  }
else
                                  {
print_frecord(temp);
                                        c = getchar();
                                  }
        }while(temp->file_offset > 0);
}
******/

void  faculty_first_name(value,cl)
char    *value;
CLIENT  *cl;
```

```
{
 extern CLIENT *clf;
if (0)
                              printf("cl = %x : %x \n",cl,*cl);
                              initialize1_f();
                              sscanf(value,"%s", pRF->firstName);
if (0)
                              printf("value = %s\n",value);
                              head = (record *)malloc(sizeof(record));
                              if(head == NULL)
                              {
                              system("clear");
                              printf("head : insuff. memory \n");
                              c = getchar();
                              return;
                              }
                              head->next_record = NULL;
if (0)
                              printf("cl = %x : %x \n",cl,*cl);
                              c = getchar();
                              pRF->file_offset = 0;
                              do{
                              temp = firstname_record_1(pRF,clf);
                              if(temp == NULL)
                                {
                                 system("clear");
                                 printf("Faculty : temp null ptr \n");
                                 c = getchar();
                                 return;
                                }
                                else
                                if(temp->file_offset > 0)
                                 insert_frecord(head,temp);
                                pRF->file_offset = temp->file_offset;
                                }
                                while(temp->file_offset > 0);
                                 print_flist(head->next_record);
}

void  faculty_last_name(value,cl)
char    *value;
CLIENT  *cl;
{
 extern CLIENT *clf;
if (0)
                                printf("cl = %x : %x \n",cl,*cl);
                                initialize1_f();
                                sscanf(value,"%s", pRF->lastName);
if (0)
                                printf("value = %s\n",value);
                                head = (record *)malloc(sizeof(record));
                                if(head == NULL)
                                {
                                system("clear");
                                printf("head : insuff. memory \n");
                                c = getchar();
                                return;
                                }
                                head->next_record = NULL;
if (0)
```

```
                                printf("cl = %x : %x \n",cl,*cl);
                                c = getchar();
                                pRF->file_offset = 0;
                                do{
                                temp = lastname_record_1(pRF,clf);
                                if(temp == NULL)
                                  {
                                   system("clear");
                                   printf("Faculty : temp null ptr \n");
                                   c = getchar();
                                   return;
                                  }
                                else
                                if(temp->file_offset > 0)
                                 insert_frecord(head,temp);
                                pRF->file_offset = temp->file_offset;
                                }
                                while(temp->file_offset > 0);
                                 print_flist(head->next_record);
}
void faculty_ssn(value,cl)
char *value;
CLIENT *cl;
{
temp = ssn_key_1(&value,cl);
if(!temp)
                                  {
                                        system("clear");
printf("SSN_KEY : null ptr \n");
                                        c = getchar();
```

```
                                            return;
                                    }
else
if(temp->file_offset < 1)
                                    {
                                            system("clear");
printf("data not found...\n");
                                            c = getchar();
                                            return;
                                    }
else
                                    {
print_frecord(temp);
                                            c = getchar();
                                            return;
                                    }
}
void faculty_phone(value,cl)
char *value;
CLIENT *cl;
{
temp = phone_key_1(&value,cl);
if(!temp)
                                    {
                                            system("clear");
printf("PHONE_KEY : null ptr \n");
                                            c = getchar();
                                            return;
                                    }
else if(temp->file_offset < 1)
```

```
                                  {
                                        system("clear");
printf("data not found...\n");
                                        c = getchar();
                                        return;
                                  }
else
                                  {
print_frecord(temp);
                                        c = getchar();
                                        return;
                                  }
}
void faculty_location(value,cl)
char *value;
CLIENT *cl;
{
temp = location_key_1(&value,cl);
if(!temp)
                                  {
                                        system("clear");
printf("LOCATION_KEY : null ptr \n");
                                        c = getchar();
                                        return;
                                  }
else
if(temp->file_offset < 1)
                                  {
                                        system("clear");
printf("data not found...\n");
```

```
                                        c = getchar();
                                        return;
                                   }
else
                                   {
print_frecord(temp);
                                        c = getchar();
                                        return;
                                   }
}

void faculty_add_record(value,cl)
char *value;
CLIENT *cl;
{
int ret;
/*****extern  char    bufdata[2560];***/
char *msgbuf[90];
                                initialize1_f();
if (0)
                                printf("In faculty_add_record....1\n");
if (0)
                                printf("In faculty_add_record -%s\n",value);
                                msgbuf[0] = (char *)malloc(sizeof(char)*30);
                                msgbuf[1] = (char *)malloc(sizeof(char)*30);
                                msgbuf[2] = (char *)malloc(sizeof(char)*30);
if (0)
                                printf("In faculty_add_record....2\n");
sscanf(value,"\n%s %s %s %s %s",
pRF->ssn,pRF->firstName,pRF->middleInitial,
pRF->lastName,pRF->phone,pRF->location);
if (0)
                                printf("In faculty_add_record....3\n");
ret = *add_record_1(pRF,cl);
if (0)
                                printf("In faculty_add_record..ret = %d\n",ret);
if(ret < 0)
                                 {
                                        system("clear");
                                        printf("Duplicate Record \n");
sprintf(msgbuf[0],"%s"," DUPLICATE ");
sprintf(msgbuf[1],"%s"," RECORD ");
                                        c = getchar();
                                        return;
/**msg_memo(2,msgbuf);
                                        ***/
                                  }
else if(!ret)
{
                                        system("clear");
                                        printf("Database file error \n");
sprintf(msgbuf[0],"%s"," DATABASE ");
sprintf(msgbuf[1],"%s"," FILE ");
sprintf(msgbuf[1],"%s"," ERROR ");
c = getchar();
                                        return;
/***msg_memo(3,msgbuf);***/
}
                                        system("clear");
                                        printf("New Record Added \n");
```

```
c = getchar();
sprintf(msgbuf[0],"%s"," RECORD ");
sprintf(msgbuf[1],"%s"," ADDED ");
/***msg_memo(2,msgbuf);***/
}
void faculty_del_record(value,cl)
char *value;
CLIENT *cl;
{
char ch;
/*** do {  ****/
temp = ssn_key_1(&value,cl);
if(temp == NULL)
{/* D1 */
                                system("clear");
printf("DEL_RECORD : temp null ptr \n");
                                c = getchar();
                                return;
}/* D1 */
if(temp->file_offset <= 0)
{    /* D2 */
                                system("clear");
printf("DEL_RECORD : record not found \n");
                                c = getchar();
                                return;
}  /* D2 */
else{
if (print_fdrecord(temp))
if(*del_record_1(&value,cl) <= 0)
                                  {
                                        system("clear");
printf("delete failed \n");
                                        c = getchar();
                                        return;
   }  /* D3 */
                                }
                  /*****       } while(temp->file_offset > 0);*****/
}
void faculty_list_record(value,cl)
char *value;
CLIENT *cl;
{
int rec_number;
head = (record *)malloc(sizeof(record));
if(head == NULL)
{/* L1 */
printf("LIST_RECORD(1) : insuff. memory \n");
                                c = getchar();
                                return;
                                }/*L1*/
                                head->next_record = NULL;
                                rec_number = 0;
                                do{/* L2 */
                                temp = list_record_1(&rec_number,cl);
                                if(temp == NULL)
                                {
                                printf("LIST_RECORD(2) : null ptr \n");
                                c = getchar();
                                return;
                                }
                                rec_number = temp->file_offset;
                                if(!rec_number)
                                        break;
                                insert_frecord(head,temp);
                                }
                                while(rec_number > 0);
                                print_flist(head->next_record);
}

void initialize1_f()
{
if (0)
        printf("in pRF initializef\n");
        pRF = (record  *)malloc(sizeof(record));
        if(pRF == NULL)
        {
                printf("pRF (1) : insufficient memory \n");
                c = getchar();
                return;
        }
        pRF->ssn           = (char *)malloc(MAX_STR);
        pRF->firstName     = (char *)malloc(MAX_STR);
        pRF->middleInitial = (char *)malloc(MAX_STR);
        pRF->lastName      = (char *)malloc(MAX_STR);
        pRF->location      = (char *)malloc(MAX_STR);
        pRF->phone         = (char *)malloc(MAX_STR);
        pRF->next_record   = NULL;
}
```

```
/* rdb_course.c : Client application for Course Database Services */

# include <stdio.h>
# include <ctype.h>
# include <rpc/rpc.h>
# include "crdb.h"
# include "frdb.h"

static record1 *pRC;

void initialize1_c();
record  *temp;
record  *head;
record1  *temp1;
record1  *head1;
record1  *temp22;

char c;

void course_get_course(value,cl,cf)
char *value;
CLIENT *cl,*cf;
{
if (0)
                                printf("value = %s \n",value);
if (0)
                                printf("cf = %d \n",cf);
                        temp = lastname_key_1(&value, cf);
if(temp == NULL)
```

```
{
                                system("clear");
printf("GET_COURSE : temp null ptr \n");
                                c = getchar();
return;
}
                                temp22 = (record1 *) malloc(sizeof(record1));
         temp22->course_number = (char *)malloc(MAX_STR);
         temp22->course_section = (char *)malloc(MAX_STR);
         temp22->course_semester = (char *)malloc(MAX_STR);
         temp22->course_name = (char *)malloc(MAX_STR);
         temp22->course_inst = (char *)malloc(MAX_STR);
         temp22->course_room = (char *)malloc(MAX_STR);
         temp22->course_bldg = (char *)malloc(MAX_STR);
         temp22->course_day = (char *)malloc(MAX_STR);
         temp22->course_time = (char *)malloc(MAX_STR);
         temp22->course_year = (char *)malloc(MAX_STR);
         temp22->course_credit = (char *)malloc(MAX_STR);
         temp22->next_course = NULL;
if (0)
                                printf("firstname ssn = %s\n",temp->ssn);
                        temp22->course_inst = temp->ssn;
if (0)
                                printf("firstname ssn = %s\n",temp22->course_inst);
head1 = (record1 *)malloc(sizeof(record1));
if(head1 == NULL)
{/* 2 */
                                system("clear");
printf("head1 : insuff. memory \n");
                                        c = getchar();
```

```
        return;
}/* 2 */
head1->next_course = NULL;
if (0)
                                printf("cl = %d\n",cl);
                                temp1 = get_course_1(temp22,cl);
if(temp1 == NULL)
{
                                system("clear");
printf("GET_COURSE : temp1 null ptr \n");
                                        c = getchar();
        return;
}
while(temp1->file_offset > 0)
{/* 3 */
insert_crecord(head1,temp1);
                                temp22->file_offset = temp1->file_offset;
                        temp1 = get_course_1(temp22,cl);
if(temp1 == NULL)
{
                                system("clear");
printf("GET_COURSE : temp1 null ptr \n");
                                        c = getchar();
        return;
}
}/* 3 */
print_clist(head1->next_course);
/**** else
                                  {
                                system("clear");
```

```
printf("invalid prof. name \n");
                                        c = getchar();
        return;
                                   }
            ******/
}

void course_name(value,cl)
char *value;
CLIENT *cl;
{
 extern CLIENT *clc;
if (0)
                                printf("cl = %x : %x \n",cl,*cl);
                                initialize1_c();
                                sscanf(value,"%s", pRC->course_name);
if (0)
                                printf("value = %s\n",value);
                                head1 = (record1 *)malloc(sizeof(record1));
                                if(head1 == NULL)
                                {
                                system("clear");
                                printf("head1 : insuff. memory \n");
                                c = getchar();
                                return;
                                }
                                head1->next_course = NULL;
if (0)
                                printf("cl = %x : %x \n",cl,*cl);
                                c = getchar();
                                pRC->file_offset = 0;
                                do{
                                temp1 = coursename_record_1(pRC,clc);
                                if(temp1 == NULL)
                                  {
                                   system("clear");
                                   printf("COURSE : temp1 null ptr \n");
                                   c = getchar();
                                   return;
                                  }
                                else
                                if(temp1->file_offset > 0)
                                 insert_crecord(head1,temp1);
                                pRC->file_offset = temp1->file_offset;
                                }
                                while(temp1->file_offset > 0);
                                 print_clist(head1->next_course);
}

void course_number(value,cl)
char *value;
CLIENT *cl;
{
 extern CLIENT *clc;
if (0)
                                printf("cl = %x : %x \n",cl,*cl);
                                initialize1_c();
                                sscanf(value,"%s", pRC->course_number);
if (0)
                                printf("value = %s\n",value);
                                head1 = (record1 *)malloc(sizeof(record1));
                                if(head1 == NULL)
                                {
                                system("clear");
                                printf("head1 : insuff. memory \n");
                                c = getchar();
                                return;
                                }
                                head1->next_course = NULL;
if (0)

                                printf("cl = %x : %x \n",cl,*cl);
                                c = getchar();
                                pRC->file_offset = 0;
                                do{
                                temp1= coursenumber_record_1(pRC,clc);
                                if(temp1 == NULL)
                                  {
                                   system("clear");
                                   printf("COURSE : temp1 null ptr \n");
                                   c = getchar();
                                   return;
                                  }
                                else
                                if(temp1->file_offset > 0)
                                 insert_crecord(head1,temp1);
                                pRC->file_offset = temp1->file_offset;
                                }
                                while(temp1->file_offset > 0);
                                 print_clist(head1->next_course);
}
```

```
void course_list_record(value,cl)
char *value;
CLIENT *cl;
{
int rec_number;
if (0)
                                printf("%s \n",value);
head1 = (record1 *)malloc(sizeof(record1));
if(head1 == NULL)
{/* L1 */
                                {
                                system("clear");
printf("LIST_RECORD(1) : insuff. memory \n");
                                        c = getchar();
        return;
                                   }
}/* L1 */
head1->next_course = NULL;
rec_number = 0;
do{/* L2 */
temp1 = clist_record_1(&rec_number,cl);
if(temp1 == NULL)
{
                                system("clear");
printf("LIST_RECORD(2) : null ptr \n");
                                        c = getchar();
        return;
                                   }
rec_number = temp1->file_offset;
```

```
if(!rec_number)
break;
insert_crecord(head1,temp1);
}
while(rec_number > 0);
print_clist(head1->next_course);
}
void course_add_record(value,cl)
char *value;
CLIENT *cl;
{
int ret;
if (0)
                                printf("%s \n",value);
                                initialize1_c();
sscanf(value,"%s %s %s %s %s %s %s %s %s %s %s",
pRC->course_number,   pRC->course_section,
                                pRC->course_semester, pRC->course_name,
                                pRC->course_inst,     pRC->course_room,
                                pRC->course_bldg,     pRC->course_day,
                                pRC->course_time,     pRC->course_year,
                                pRC->course_credit);
if (0)
printf("pRC.coursenum = %s \n",pRC->course_number);
ret = *cadd_record_1(pRC,cl);
if(ret < 0)
                                 {
                                system("clear");
printf("Duplicate record \n");
                                        c = getchar();
```

```
        return;
                                   }
else if(!ret)
{
                                System("clear");
printf("Database file error \n");
                                        c = getchar();
        return;
                                   }
else
                                   {
                                System("clear");
                                        printf("record added \n");
                                        c = getchar();
        return;
                                   }
}
void course_del_record(value,cl)
char *value;
CLIENT *cl;
{
char ch;
if (0)
                                printf("%s \n",value);
/*  do{                                      */
                                temp1 = coursenumber_key_1(&value,cl);
                                if(temp1 == NULL)
                                {/* D1 */
                                system("clear");
                                printf("DEL_RECORD : temp null ptr \n");
```

```
                                        c = getchar();
        return;
                                }/* D1 */
                                if(temp1->file_offset <= 0)
                                {/* D2 */
                                system("clear");
                                printf("DEL_RECORD : record not found \n");
                                        c = getchar();
        return;
                                   }
                                else{
                                if (print_cdrecord(temp1))
                                if(*cdel_record_1(&value,cl) <= 0)
                                 {
                                system("clear");
                                        printf("delete failed \n");
                                        c = getchar();
                                        return;
                                }
                             }
/*                          }while(temp1->file_offset > 0);           */
}


void initialize1_c()
{
if (0)
        printf("in pRC initialize1\n");
        pRC = (record1 *)malloc(sizeof(record1));
        if(pRC == NULL)
        {
                printf("pRC (2) : insufficient memory \n");
c = getchar();
                return;
        }
        pRC->course_number = (char *)malloc(MAX_STR);
        pRC->course_section = (char *)malloc(MAX_STR);
        pRC->course_semester = (char *)malloc(MAX_STR);
        pRC->course_name = (char *)malloc(MAX_STR);
        pRC->course_inst = (char *)malloc(MAX_STR);
        pRC->course_room = (char *)malloc(MAX_STR);
        pRC->course_bldg = (char *)malloc(MAX_STR);
        pRC->course_day = (char *)malloc(MAX_STR);
        pRC->course_time = (char *)malloc(MAX_STR);
        pRC->course_year = (char *)malloc(MAX_STR);
        pRC->course_credit = (char *)malloc(MAX_STR);
        pRC->next_course = NULL;
}
```

```
/* rdb_student.c : Client application for Student database Services */

# include <stdio.h>

# include <ctype.h>

# include <rpc/rpc.h>

# include "srdb.h"

static record2 *pRS;

void initialize1_s();

char c;

record2  *temp2;

record2  *head2;

/***********Cases for Student Database **********/

student_first_name(value,cl)

char *value;

CLIENT *cl;

{

                                initialize1_s();

                                sscanf(value,"%s", pRS->stud_firstName);

if (0)

                                printf("value = %s\n",value);

                                head2 = (record2 *)malloc(sizeof(record2));

                                if(head2 == NULL)

                                {

                                system("clear");
```

```
                                        printf("head2 : insuff. memory \n");
                                        c = getchar();
                                        return;
                                        }
                                        head2->next_record = NULL;
if (0)

                                        printf("cl = %x : %x \n",cl,*cl);
                                        c = getchar();
                                        pRS->file_offset = 0;
                                        do{
                                        temp2 = sfirstname_record_1(pRS,cl);
                                        if(temp2 == NULL)
                                          {
                                           system("clear");
                                           printf("Student : temp2 null ptr \n");
                                           c = getchar();
                                           return;
                                          }
                                        else
                                        if(temp2->file_offset > 0)
                                         insert_srecord(head2,temp2);
                                        pRS->file_offset = temp2->file_offset;
                                        }
                                        while(temp2->file_offset > 0);
                                         print_slist(head2->next_record);
}

student_last_name(value,cl)
char *value;
CLIENT *cl;
```

```
{
                                initialize1_s();
                                sscanf(value,"%s", pRS->stud_lastName);
if (0)
                                printf("value = %s\n",value);
                                head2 = (record2 *)malloc(sizeof(record2));
                                if(head2 == NULL)
                                {
                                system("clear");
                                printf("head2 : insuff. memory \n");
                                c = getchar();
                                return;
                                }
                                head2->next_record = NULL;
if (0)
                                printf("cl = %x : %x \n",cl,*cl);
                                c = getchar();
                                pRS->file_offset = 0;
                                do{
                                temp2 = slastname_record_1(pRS,cl);
                                if(temp2 == NULL)
                                  {
                                   system("clear");
                                   printf("Student : temp2 null ptr \n");
                                   c = getchar();
                                   return;
                                  }
                                else
                                if(temp2->file_offset > 0)
                                 insert_srecord(head2,temp2);
                                pRS->file_offset = temp2->file_offset;
                                }
                                while(temp2->file_offset > 0);
                                 print_slist(head2->next_record);


}
student_ssn(value,cl)
char *value;
CLIENT *cl;
{


temp2 = sssn_key_1(&value,cl);
if(!temp2)
                                      {
                                        system("clear");
printf("SSN_KEY : null ptr \n");
                                        c = getchar();
                                        return;
                                       }
else
if(temp2->file_offset < 1)
                                      {
                                        system("clear");
printf("data not found...\n");
                                        c = getchar();
                                        return;
                                       }
else
print_srecord(temp2);
}
```

```
student_phone(value,cl)
char *value;
CLIENT *cl;
{
temp2 = sphone_key_1(&value,cl);
if(!temp2)
                                        {
                                         system("clear");
printf("PHONE_KEY : null ptr \n");
                                         c = getchar();
                                         return;
                                        }
else if(temp2->file_offset < 1)
                                        {
                                         system("clear");
printf("data not found...\n");
                                         c = getchar();
                                         return;
                                        }
else
print_srecord(temp2);
}
student_add_record(value,cl)
char *value;
CLIENT *cl;
{
int ret;
                                 initialize1_s();
sscanf(value,"%s %s %s %s %s %s %s %s %s %s %s %s ",
pRS->stud_ssn,pRS->stud_firstName,
                                pRS->stud_middleInitial,
pRS->stud_lastName,pRS->stud_address,
pRS->stud_city,pRS->stud_state,
pRS->stud_zip,pRS->stud_phone,
pRS->stud_major,pRS->stud_college,
                                pRS->stud_gpa);
ret = *sadd_record_1(pRS,cl);
if(ret < 0)
                                      {
                                        system("clear");
printf("Duplicate record \n");
                                        c = getchar();
                                        return;
                                       }
else if(!ret)
{
                                        system("clear");
printf("Database file error \n");
                                        c = getchar();
                                        return;
                                       }
}
student_del_record(value,cl)
char *value;
CLIENT *cl;
{
char ch;
 /****                                do{*****/
temp2 = sssn_key_1(&value,cl);
if(temp2 == NULL)
```

```
{/* D1 */
                                        system("clear");
printf("DEL_RECORD : temp null ptr \n");
                                c = getchar();
                                return;
}/* D1 */
if(temp2->file_offset <= 0)
{/* D2 */
                                        system("clear");
printf("DEL_RECORD : record not found \n");
                                c = getchar();
return;
}/* D2 */
                                else{
if (print_sdrecord(temp2))
if(*sdel_record_1(&value,cl) <= 0)
                                  {
                                        system("clear");
printf("delete failed \n");
                                        c = getchar();
        return;
   }
                                 }
                   /****             }while(temp2->file_offset > 0);*****/
}
student_list_record(value,cl)
char *value;
CLIENT *cl;
{
int rec_number;
```

```
head2 = (record2 *)malloc(sizeof(record2));
if(head2 == NULL)
{/* L1 */
                                        system("clear");
printf("LIST_RECORD(1) : insuff. memory \n");
                                        c = getchar();
        return;
}/* L1 */
head2->next_record = NULL;
rec_number = 0;
do{/* L2 */
temp2 = slist_record_1(&rec_number,cl);
if(temp2 == NULL)
{
                                        system("clear");
printf("LIST_RECORD(2) : null ptr \n");
                                        c = getchar();
        return;
   }
rec_number = temp2->file_offset;
if(!rec_number)
break;
insert_srecord(head2,temp2);
}
while(rec_number > 0);
print_slist(head2->next_record);
}

/******initialize1_s()**********/
void initialize1_s()
```

```
{

if (0)
       printf("in initialize1_s()\n");
        pRS= (record2  *)malloc(sizeof(record2));
        if(pRS == NULL)
        {
                printf("pRS (1) : insufficient memory \n");
                c = getchar();
                return;
        }
        pRS->stud_ssn           = (char *)malloc(MAX_STR);
        pRS->stud_firstName     = (char *)malloc(MAX_STR);
        pRS->stud_middleInitial = (char *)malloc(MAX_STR);
        pRS->stud_lastName      = (char *)malloc(MAX_STR);
        pRS->stud_address       = (char *)malloc(MAX_STR);
        pRS->stud_city          = (char *)malloc(MAX_STR);
        pRS->stud_state         = (char *)malloc(MAX_STR);
        pRS->stud_zip           = (char *)malloc(MAX_STR);
        pRS->stud_phone         = (char *)malloc(MAX_STR);
        pRS->stud_major         = (char *)malloc(MAX_STR);
        pRS->stud_college       = (char *)malloc(MAX_STR);
        pRS->stud_gpa           = (char *)malloc(MAX_STR);
        pRS->next_record = NULL;
}
```

```
/* rdb_register.c : Client application for register.data*/


# include <stdio.h>

# include <ctype.h>

# include <rpc/rpc.h>

# include "rrdb.h"


static record3 *pRR;



record3  *temp3;

record3  *head3;


/************Cases for Register Database **********/

void register_grade(value,cl)

char *value;

CLIENT *cl;

{


sscanf(value,"%s %s %s %s %s ",

pRR->course_number,pRR->course_section,

                                pRR->course_semester,pRR->course_year,

pRR->grade);

     temp3 = grade_key_1(pRR, cl);

if(temp3 == NULL)

                                {

printf("GRADE_KEY : null ptr \n");

                                return;

                                }

                                if(temp3->file_offset > 0)
```

```
{
head3 = (record3 *)malloc(sizeof(record3));
if(head3 == NULL)
{
printf("head3 : insuff. memory \n");
exit(1);
}
head3->next_record = NULL;
temp3->file_offset = 0;
temp3 = grade_1(temp3,cl);
if(temp3 == NULL)
  {
   printf("GRADE : temp3 null ptr \n");
              return;
   }
 while(temp3->file_offset > 0)
 {
 insert_rrecord(head3,temp3);
 temp3->file_offset ;
 temp3 = grade_1(temp3,cl);
 if(temp3 == NULL)
 {
  printf("GRADE : temp3 null ptr \n");
  break;
 }
 }
 print_rlist(head3->next_record);
 }
 else
 printf("invalid grade \n");
```

```
}
/*********************************************************************/
void register_ssn(value,cl)
char *value;
CLIENT *cl;
{

     temp3 = rssn_key_1(&value, cl);
if(temp3 == NULL)
                                {
printf("RSSN_KEY : null ptr \n");
                                return;
                                }
                                if(temp3->file_offset > 0)
                                {
                                head3 = (record3 *)malloc(sizeof(record3));
                                if(head3 == NULL)
                                {
                                printf("head3 : insuff. memory \n");
                                exit(1);
                                }
                                head3->next_record = NULL;
                                temp3->file_offset = 0;
                                temp3 = rssn_1(temp3,cl);
                                if(temp3 == NULL)
                                  {
                                   printf("RSSN : temp3 null ptr \n");
                                              return;
                                   }
                                 while(temp3->file_offset > 0)
                                {
                                insert_rrecord(head3,temp3);
                                temp3->file_offset ;
                                temp3 = rssn_1(temp3,cl);
                                if(temp3 == NULL)
                                {
                                 printf("RSSN : temp3 null ptr \n");
                                 break;
                                }
                                }
                                print_rlist(head3->next_record);
                                }
                                else
                                printf("invalid ssn \n");
}
/*********************************************************/
void register_list_record(value,cl)
record3 *value;
CLIENT *cl;
{
int rec_number;
head3 = (record3 *)malloc(sizeof(record3));
if(head3 == NULL)
{/* L1 */
printf("LIST_RECORD(1) : insuff. memory \n");
exit(1);
}/* L1 */
head3->next_record = NULL;
rec_number = 0;
do{/* L2 */
```

```
temp3 = rlist_record_1(&rec_number,cl);
if(temp3 == NULL)
{
printf("LIST_RECORD(2) : null ptr \n");
exit(1);
}
rec_number = temp3->file_offset;
if(!rec_number)
break;
                                else
insert_rrecord(head3,temp3);
}
while(rec_number > 0)
print_rlist(head3->next_record);
}
void register_add_record(value,cl)
char *value;
CLIENT *cl;
{
int ret;

sscanf(value,"%s %s %s %s %s %s ",
pRR->stud_ssn,pRR->course_number,
                                pRR->course_section,pRR->course_semester,
pRR->course_year);
{
printf("ADD_RECORD requires a compl. quoted record\n");
exit(1);
}
```

```
ret = *radd_record_1(pRR,cl);
if(ret < 0)
printf("Duplicate record \n");
else if(!ret)
{
printf("Database file error \n");
exit(1);
}
}
/******************************************************************/
void register_del_record(value,cl)
char *value;
CLIENT *cl;
{
char ch;
                                temp3 = rssn_key_1(&value,cl);
                                if(temp3 == NULL)
                                {/* D1 */
                                printf("DEL_RECORD : temp null ptr \n");
                                return;
                                }/* D1 */
                                if(temp3->file_offset <= 0)
                                {/* D2 */
                                printf("DEL_RECORD : record not found \n");
                                return;
                                }/* D2 */
                                printf("\n");
                                print_rrecord(temp3);
                                printf("Delete (y/n) ? ");
                                scanf("%c",&ch);
                                if(ch == 'Y' || ch == 'y')
                                {/* D3 */
                                printf("deleting record ...\n");
                                if(*rdel_record_1(&value,cl) <= 0)
                                        printf("delete failed \n");
                                }/* D3 */
}
```

BIBLIOGRAPHY

[1] "The Bellcore $OSCA^{TM}$ Architecture." *Bellcore - Bell Communications Research*, Technical Advisory, TA-STS-000915, ISSUE 2, July 1990.

[2] Rossak, Wilhelm. "INTEGRATION ARCHITECTURES A Concept and a Tool to Support Integrated Systems Development." *Institute for Integrated Systems Research*, Department of Computer & Information Science, New Jersey Institute of Technology.

[3] Bloomer, John. "Ticket to Ride." *SunWorld*, Nov 1991, pp. 39-55.

[4] Mills, John. "The Operations Systems Computing Architecture." *Proceedings of the First Intl. Conference on Systems Integration*, Morristown, NJ, IEEE Computer Society Press, April 1990.

[5] Mallett, Mark. "A Look at Remote Procedure Calls." *Byte*, May 1991.

[6] Rossak, W., and P. Ng. "Some Thoughts on Systems Integration - A Conceptual Framework." *Journal of Systems Integration*, Vol. 1, No. 1, Kluwer, 1991, pp. 97-114.

[7] Rossak, W., and S. Prasad. "Integration Architectures - A Framework for System Integration Decisions." *Proc. of the IEEE Intl. Conf. on Systems, Man and Cybernetics*, Charlottesvilee VA, October 1991, pp. 545-550.

[8] Prieto-Diaz, R., and G. Arango. "Domain Analysis and Software Systems Modeling." *IEEE Computer Society Press*, Los Alamitos, CA, 1991.

[9] Best, L. "Application Architecture - Modern Large Scale Information Processing." Wiley, NY, 1990.

[10] Rossak, W. "System Development with Integration Architectures." *International Conference on Systems Integration*, Morristown, NJ, June 1992, to appear.

[11] Bloomer, John. "Power Programming with RPC." O'Reilly and Associates, Inc., February 1992.