A MASTER'S DEGREE COURSE OF STUDY DATABASE

by

GEORGE RICHARD HUGHES

B.A., Kansas University, 1973

M.A., University of Regina, 1974

A MASTER'S REPORT

submitted in partial fulfillment of the

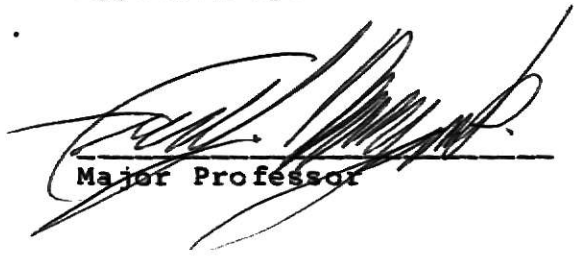
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1979

Approved by:



Major Professor

TABLE OF CONTENTS

List of Figures iv

Part I. Introduction 1

1.1 History 1

1.2 Preliminary Guidelines 2

1.3 Controlling Purpose 4

Part II. Specification Requirements 5

2.1 Listing of requirements 5

2.2 Relationships Between System Components 9

Part III. Description of Functions 11

3.1 User Operations 11

3.2 The Database 12

3.3 Program GPA1 19

3.4 Program CHECK 21

3.5 Program Check2 23

3.6 Program CLASSES 24

3.7 Text Files 25

Part IV. The Interface 26

Part V. Summary 30

5.1 Analysis 30

5.2 Concluding Remarks 31

Bibliography 33

Appendix A - User's Guide 34

Appendix B - Supervisory Manual 36

Appendix C - Functional Specifications . . . 44

List of Figures

FIGURE

2.1 System Component Relations 10

3.1 Database Definition of COURSE 14

B.1 Procedure for Gaining Access to COURSE 38

C.1 Source Code for Program GPA1 51

C.2 Source Code for Program CHECK 54

C.3 Source Code for Program CHECK2 58

C.4 Directory for Text Files 63

CHAPTER I

INTRODUCTION

1.1 History

There has been an ongoing effort in this University to automate certain student records. The college of Home Economics has developed a program which will verify what Bachelor's degree course requirements are being met by their major students. This program helps faculty members advise students in planning course loads for future semesters. This program was implemented using PL/1, with no provisions made for any sort of database management system (DBMS). [4]

Although several Home Economics departments at other institutions have expressed interest in duplicating this implementation, it suffers from being used in the batch mode only. Additionally, this program does not lend itself easily to modification, and given frequent curricula changes at both the departmental and College levels, this is a serious complaint.

Voelz and Garland [8] designed a prototype system using a DBMS that would contain University-wide academic records. This implementation utilized the Integrated Data Base Management System (IDMS) [2] , and was an adequate starting point. The commitment to a DBMS was important because the data can be described, stored, and manipulated independently of different users. Voelz and Garland's

design proved to be too large, expensive, and wide ranging for any Computer Science Department applications.

In 1976, Long [4] constructed an IDMS database that extended upon the concepts of Voelz and Garland's scheme. Long was commissioned to make an economical system which would act as a counselling aid for the Computer Science faculty. Other requirements to be met by this database included Bachelor of Science requirements, major course requirements in Computer Science, some personal information on the student, and courses completed and/or currently enrolled in. This program also used the batch mode on an IBM 370 mainframe.

1.2 Preliminary Guidelines

The evolution of computer programs that automate both academic record keeping and student advising procedures continues. During discussions last Fall, the need was expressed for a prototype system which would go beyond the efforts mentioned above. The major contraints upon this system were:

1. the system must be available to interactive display terminal users
2. the DBMS must provide independence and security
3. the system would advise and help Master's students in Computer Science
4. the prototype must be cost effective

These constraints necessitated using the Department's Interdata 8/32 minicomputer, and a conversational DBMS system INFO32 [3] , as the hardware and software support, respectively, for this project. The choice of the 8/32 reflected the cost constraint, as the overall database would be cheaper to operate on this machine compared with an implementation on the Kansas State Computing Center's Itel AS-5. The smaller machine had the added benefits of terminal display mode, and convenient access for Computer Science students.

Other reasons for choosing the Interdata mini were that the Department would like to encourage the use of the 8/32 by out of town users via telephone links. Furthurmore, the Department has a continuing research interest in database applications for minicomputers over distributed networks.

INFO32 is a software product of Henco, Inc. [3]. Primarily a business oriented system, INFO32 uses English keywords as language command instructions for data entry and update, query formulation, and report writing. INFO32 was selected as the DBMS because it works well in a terminal display environment, was available on the 8/32, and lent itself easily to a frontend interface [5] . This interface will provide much of the data security and independence, as it masks the user from the actual INFO32 database. It will be discussed at a later time in this report.

1.3 Controlling Purpose

The controlling purpose of this report is to show the design and implementation of this Master's degree database prototype project. We have already looked at the background of previous attempts in this area, and the renewed interest in providing new informational services to prospective and enrolled graduate students across the state.

In Section II, we will describe the specific requirements that this project will try to fulfill. In Section III, the database design will be discussed, and the assertions that can be made about the various data items. Other program parts within INFO32 that manipulate this database will also be described in this section.

We will outline the general data independence qualities that the backend interface between the user and the database will support. A summary of this report will be presented, as will a look at future directions for this type of work. Finally, the appendices document the user and supervisory manuals needed to modify or keep this implementation functioning.

CHAPTER II

SPECIFICATION REQUIREMENTS

2.1 Listing of Requirements

The impact of the initial constraints was felt mostly in the selection of physical devices and software packages. These constraints did not really effect the design or logical organization of the fledgling project. Section II will be an amplification of the overall purposes of the project, and a complete statement of specifications that will be realized in this implementation.

This project will be comprised of up-to-date, but un-official academic and personal facts about Master's students in Computer Science. Several different purposes are suggested by the specifications that follow. The first purpose is informational. Students can choose to view reports about admission requirements, equivalent experience in this discipline needed at the outset of graduate work, or other matters relating to the Master's program in Computer Science at Kansas State University.

Secondly, this implementation will securely store courses completed or enrolled in for individual students. The credit hours and grade received for each course, and the semester when the class was taken can also be entered into

the database. The student may, on demand, request that his grade point average be calculated, or have his list of courses evaluated for degree requirement satisfaction.

The third purpose of this project is to duplicate certain Departmental personal files. Faculty members can be spared the tedium of shuffling through file folders when checking on the progress of a graduate student.

These three purposes helped to shape the following required specifications:

1. Interactive capability for Manhattan, and out lying users
    - 1.a. access informational text files
    - 1.b. one academic record stored in the database per user
        - 1.b.1. user can create this record
        - 1.b.2. user can update items in this record
        - 1.b.3. user can read through parts of this created record
2. Database Administrator
    - 2.a. purges inactive records
    - 2.b. supervises the maintenance of the system
    - 2.c. modifies the system due to necessary changes
3. Interface that will monitor communication between the user and the INFO32 database
    - 3.a. controls user creation, updating, and reading

3.b. controls user access to informational text files

3.c. provides some type checking on data

3.d. provides for data independence and security

4. Individual Student records

4.a. name, address, etc.

4.b. city where enrolled

4.c. means of financial support (only important if the student is a Computer Science Department employee)

4.d. courses taken or currently enrolled in

4.d.1. course number

4.d.2. letter grade

4.d.3. credit hours

4.d.4. semester of enrollment

5. Queries allowed on the database record

5.a. grade point average calculation

5.b. student's position with regards to the 30 graduate credit hour minimum

5.c. 'C' and 'D' credit hour deficiency

5.d. curriculum checks for required courses

5.d.1. core curriculum classes

5.d.2. upper level class

5.d.3. graduate seminar

6. Other Information

6.a. Department admission requirements

6.b. Department MS requirements

6.c. Bachelor of Science equivalent experience one should have, or undergraduate classes to be made up upon acceptance into this Master's program

6.d. current and following semester schedules

6.e. Kansas State Computer Science faculty biographical sketches

6.f. courses acceptable for MS degree credit

6.f.1. pre-requisites

6.f.2. content of courses

6.f.3. credit hours for the course

7. No concurrent use of the database, at least for the time being

2.2 Relationships Between System Components

At this point, it would be best to explain the interrelationships between all the different parts of the system that will implement these specifications.

The MTM operating system in the Interdata 8/32 oversees and supervises any communication between the component parts. Its task is to provide the environment by which these parts can utilize the minicomputer's resources.

The DBMS, INFO32, controls all messages regarding the database itself. The INFO32 system only allows qualified

users making legal queries to manipulate any part of the database.

This underscores the fact that the Fortran interface's [6] main job is the interception and interpretation of communications between a terminal user, INFO32, and ultimately, the INFO32 database COURSE. The interface lets the user choose from a limited set of options. The messages outlining these options are sent to the user, the user chooses a particular task, and the interface executes the necessary steps to carry out that task.

The interface, in this way, receives and gives messages to both the DBMS and the student user. The text files are chosen individually by the student, the interface learns of the choice, and then extracts for display the correct file.

Finally, the student user can only send messages directly to the interface, while receiving information from the DBMS programs, the text files, and the monitoring interface. The supervisory user can use the database by circumventing the interface, and dealing directly with INFO32. On the systems level, the supervisor can maintain and create any text files.

A graphical representation of the system components can be seen in Figure 2.1.

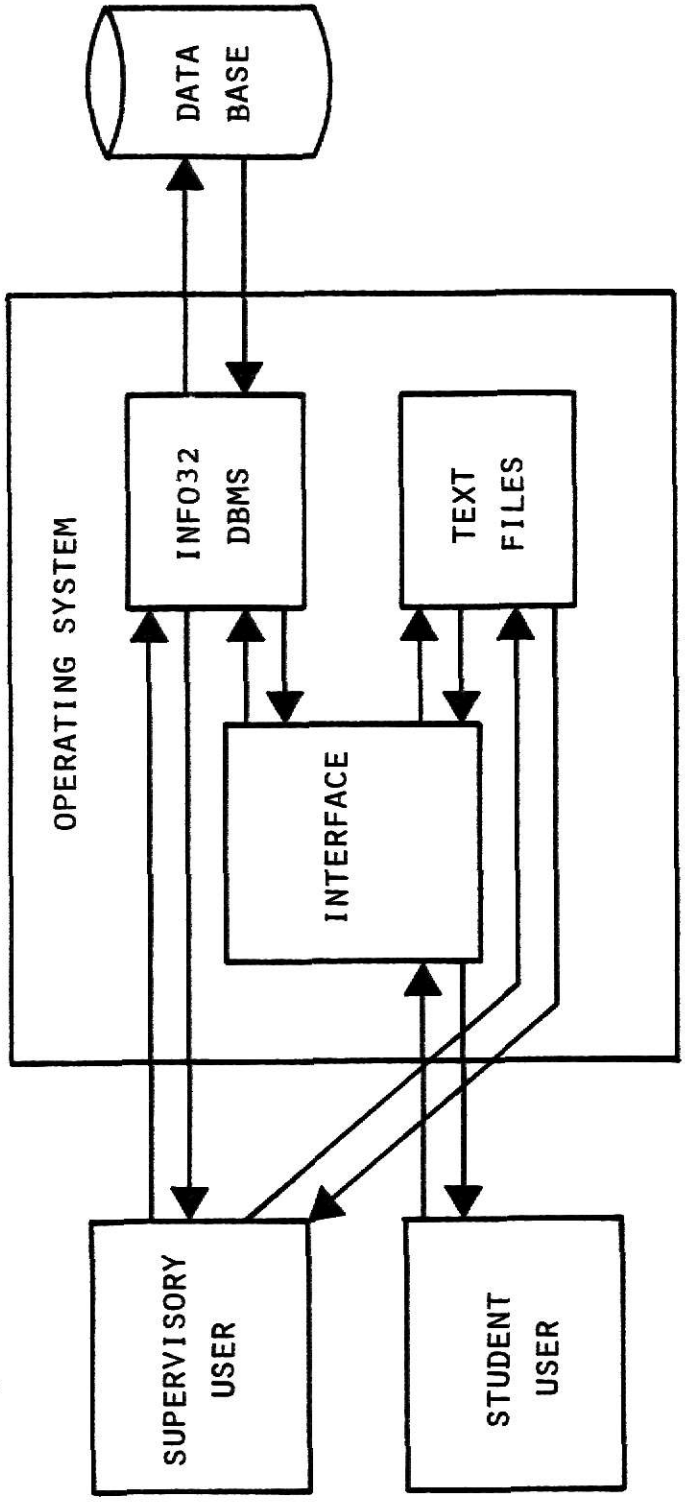

DATA BASE
OPERATING SYSTEM
INFO32 DBMS
TEXT FILES
INTERFACE
SUPERVISORY USER
STUDENT USER

FIGURE 2.1

CHAPTER III

Description of Functions

The next step is to put flesh on the skeleton of existing requirement specifications and constraints. This chapter contains a description of the logical organization of various program segments. The tasks involved here are either the database itself, or command stream programs within the INFO32 file directory which can act upon the currently selected student record.

The latter type of program includes GPA1, CHECK, CHECK2, and CLASSES. The more detailed functional specifications on these tasks will appear in the Appendix.

3.1 User Operations

There are four primary operations that a user, with the interface's help, can make on the database COURSE. A new student can enter the suggested personal information into his own record. Secondly, a continuing student can make updates on selected database items, by either changing the original entries, or entering information that is new, as would be the case after a completed semester.

At any point in the interactive session, the user can choose to view his database record in its entirety.

Finally, the user can request operations that query his record for curriculum checks and grade point average. These user operations suggest the following definitions for the database.

3.2 The Database

The database's chief feature is the reproduction and automation of a graduate student's Departmental file. To this end, many of the data items are fairly straight forward, such as first, last, and middle names; and street, city, state, and zip code for composing addresses. Although ZIP is not yet used for any calculations, it was thought that an integer and not a character representation would be advisable. We don't want to confuse the Postal Service with any 'ABCDE' zip codes.

The database now consists of 82 separate data items or data fields, which amount to a total of 373 bytes. As can be seen from Figure 3.1, more than two thirds (or 56) of these data fields consist of the record keeping apparatus for storing 14 different courses, grades, credit hours, and semester of enrollment.

The association of each CLASS-xx, GRADE-xx, HOURS-xx, and ENROL-xx is made by the last two character places on these data item names. For example, CLASS-09 is associated with GRADE-09, and with HOURS-09, and so on. The choice of

14 different courses was an arbitrary decision based on the fact that most graduate classes are three credit hour courses. Since most graduate candidates do not greatly exceed the 30 credit hour requirement , 14 classes should be sufficient for this amount of course work.

Approximately 45% of the database storage requirements are needed because INFO32 does not have any dynamic allocation feature, and so these 56 static fields must always be in place. An array implementation with looping would have definitely shortened the command stream programs, which for the most part, manipulate these fields of classroom achievement.

To complete the discussion of the 14 repetitive class instances, data items CLASS-01 through CLASS-14 were chosen to be of type integer, with a range of 1 to 999999. The reason for integer typing for this item was that logical comparisons like 'greater than' or 'equal to' were needed in determining if degree course requirements had been met.

Database Definition of COURSE

ITEM-NAME	TYPE	LENGTH	POSITION
LAST	Character	15	1 to 15
FIRST	Character	10	16 to 25
MIDDLE	Character	10	26 to 35
SSN	Character	9	36 to 44
STREET	Character	20	45 to 64
CITY	Character	15	65 to 79
STATE	Character	2	80 to 81
ZIP	Integer	5	82 to 86
TELEPHONE	Character	12	87 to 98
WHERE-ENROLLED	Character	12	99 to 110
FULL-TIME	Character	1	111 to 111
START-OF-PROGRAM	Date	8	112 to 119
MASTERS-EXAM	Character	1	120 to 120
DATE-MS-EXAM	Date	8	121 to 128
ORALS-PASSED	Character	1	129 to 129
DATE-OF-ORALS	Date	8	130 to 137
DEGREE-SOUGHT	Character	3	138 to 140
SUPPORT-CLASS	Character	5	141 to 145
SUPPORT-START	Date	8	146 to 153
SUPPORT-SOURCE	Character	10	154 to 163
SUPPORT-TENTH	Integer	3	164 to 166
ADVISOR	Character	15	167 to 181
MAJOR-PROF	Character	15	182 to 196
GPA	Numeric	4,2	197 to 200
TOTAL-HOURS	Integer	2	201 to 202
TOTAL-CREDITS	Integer	3	203 to 205
CLASS-01	Integer	6	206 to 211
GRADE-01	Character	1	212 to 212
HOURS-01	Integer	1	213 to 213
ENROL-01	Character	4	214 to 217
. . . . .			
CLASS-14	Integer	6	362 to 367
GRADE-14	Character	1	368 to 368
HOURS-14	Integer	1	369 to 369
ENROL-14	Character	4	370 to 373

Figure 3.1

Six integer digits were used, as a course like C. S. 720 must be designated as its full course number-- 286720-- because 720 is not a unique identifier. Any other department might have a 720 course number, so consequently all six digits are present.

Items GRADE-01 to GRADE-14 were set to be one byte character variables, so that the usual alphabetic marks could be recorded. This GRADE-xx series is not without its problems, as any other letter can inadvertantly be entered. In the cases of 'I' for incomplete, or 'W' for withdraw, this incorrect data entry seems at least plausible.

Since the DBMS and interface do not check for this type of error in data validity, the command stream programs generally treat courses with grades of anything but 'A', 'B', 'C', or 'D' as courses that have yet to be completed. Maybe by being informed that a course he is sure he has completed has not been taken, a student will be moved to recheck the grade entry for this course.

HOURS-01 to HOURS-14 are all one digit integer variables that contain the credit hours for that particular course. It would have been a small matter to check these values for accuracy, as most classes are worth three credit hours. But since there is a curriculum change being considered whereby emphasis courses like C. S. 700 are expanded to four credit hours, this testing was not done.

ENROL-01 through ENROL-14 identify the semester in which a class was taken. A four character shorthand code is the

suggested valid entry, as Fall 1978 becomes 'FA78'. Similarly, 'SP' for Spring and 'SU' for Summer are the other acceptable combinations for the first two data places. The last two places are just the last two digits for that year. No verification is done with this field, as entries like 'WI' for Winter, or several Intersession terms that defy classification are certainly possible for the database user.

The personal information like name and address is not too controversial. Telephone, the items that comprise the address, and last name are updatable, as people move, and in a sometimes related action, change their last name.

Having the SSN, or social security number, as a length nine character string item in this database could be the source for data being compromised, and could even cause legal headaches. The initial design of this database used the SSN as the key by which a user would identify his existing record. The fallacy in using SSN as the key to selecting this database record is that (especially at a school like Kansas State where social security numbers are also student identification numbers) a social security number can often become public knowledge.

An unscrupulous person could then jeopardize a database record containing the SSN that he had somehow obtained. The response to this serious problem was to have the interface generate and display a five character hash key based on an individual's social security number upon record creation.

All subsequent referals to this record would only be allowed by the entering of this hash key at the beginning of the conversation. There is great room for improvement in the area of protecting this data.

The WHERE-ENROLLED variable was included so that Manhattan, Leavenworth, Wichita, or Kansas City students could be differientiated. Full time status is a simple one character 'Y' or 'N' entry, with presumably more Department attention being given to full time students.

In the case of FULL-TIME, as well as the other one character items MASTERS-EXAM and ORALS-PASSED, the 'Y' or 'N' type checking is done at data entry by the interface. An 'N' response to these last two items will cause the interface to branch around DATE-MS-EXAM and DATE-OF-ORALS, respectively. To wit: if one hasn't passed the orals exam, there isn't much point in asking when that exam took place.

INFO32 does make provisions for date types. All dates are checked for validity by this system, with the format being MM/DD/YY, or 6/27/79. This will be the only practical type checking for dates because of the problem encountered when this prototype reaches the twenty-first century. The answer would be to utilize INFO32's 10 character date types. As for now, the saving of two bytes on each date is justification enough for this definition.

The DATE-MS-EXAM and the START-OF-PROGRAM dates can be used for gauging the degree progress of a student whose graduate Computer Science career has been done exclusively

at Kansas State. A transfer student could distort the credibility of these two items by entering accurate dates of his accomplishments at other schools.

The DEGREE-SOUGHT item could be helpful in segregating Master's students, and PhD hopefuls. The advisor and MAJOR-PROF categories could possibly be redundant. But in the case of a student having a different advisor and major professor, this difference will be noted.

The SUPPORT- series of data items are primarily intended for use by the Department head. The SUPPORT-SOURCE, SUPPORT- CLASS, and SUPPORT-TENTH items can be used for bookkeeping, accounting summaries per source of funding, or even performance evaluation of SUPPORT-CLASS types like 'G.T.A' or 'G.R.A'.

Finally, no type checking is done on TOTAL-HOURS, GPA, or TOTAL-CREDITS, other than the usual integer versus character checking INFO32 always does.

The GPA1 program will figure a new value for TOTAL-HOURS, TOTAL-CREDITS, and GPA, and return the new value to the student's tuple. A final note on GPA and real numbers: INFO32 must count the decimal point as a digit place. In order to get the familar grade point form of x.xx, GPA had to be defined as a number of length 4, with two decimal digits.

In summation, the COURSE database is a static entity containing 82 different data items. It contains no links, as would be the case in a network model, and consists of one

schema and one subschema. This database is one big collection of these different data items and relationships. An individual's record represents the academic and personal facts of one student, keyed upon by social security number, and protected by the hash key password.

This design decision of making one database with many records was necessary because INFO32 doesn't provide for making duplicate database copies off of a model database definition for every different user. INFO32 does not allow any concurrent operations, so the database can now be accessed by only one user at a time.

3.3 Program GPA1

Program GPA1 is a command stream program embedded in the INFO32 DBMS. It is a sequential program using IF statements in processing the selected database record for grade point average, total grade credits, and total credit hours. Additionally, the credit hours of grades 'C' and 'D', and the relationship of total credits of that user to the 30 graduate credit hour requirement will be computed. It is the job of the interface to select just that record of the database that is relevant to the current user of GPA1's functions.

The GPA1 section will only keep a running total of total credit hours and total grade credits for these courses at least at the 600 level in Computer Science. The C. S. 891

intensive course for non-major graduate students is the lone exception to this rule.

These two running totals will only be made on a course whose GRADE-xx is explicitly an 'A', 'B', 'C', 'D', or 'F'. In the last case, hours of 'F' must be added to total credit hours temporarily for grade point calculations, but left out of total credit hours when that variable is subtracted from the magic number of 30. If 'F' credit hours were not subtracted at this point, the user with any 'F' marks would get a distorted picture of how close to graduation he might be.

Thus the GPA data is limited to completed graduate courses in Computer Science at Kansas State. This differs from the official admissions and record computation by only graduate courses completed in other departments. This might seem to be a trivial matter, since at most six credit hours can be taken outside of the Computer Science curriculum.

I did not see any easy way in INFO32 to account for these possible credits from other departments. INFO32 has no character string functions with which acceptable graduate courses could be excised for GPA1's attention. So I choose to eliminate from GPA1's consideration any classes which did not conform to the domain of Computer Science graduate courses, because of the awkward nature of type integer for CLASS-xx that would adversely affect performance, and because of the absence of character string manipulation in INFO32 that could have been helpful.

Also unresolved is the problem of accepting for credit courses completed at other institutions. The Kansas State Graduate Office is, of course, the final arbitrar in these matters, so perhaps database users of this type should be instructed to write that office. As it stands now, there is no allowance for accurately interpretting course numbers from other schools that may gain entry in a student's tuple, although these transfer credits will not be included in GPA1's processing.

On the subject of lower grades, a successful Master's candidate must have at least three times the number of credit hours of 'A' or 'B' hours, as he has in credit hours of 'C' or 'D'. GPA1 will make known this deficiency due to too many 'C' or 'D' hours. A warning message will be displayed by GPA1 in the event that a student's grades are lacking in this way.

Other display messages given by this program are the grade point average itself, and how many credits remain for graduation for that person. GPA, TOTAL-CREDITS, and TOTAL-HOURS are all returned back in updated form to the student's record in COURSE. A more detailed presentation, and a source code listing of each of these programs can be found in the Appendix.

3.4 Program CHECK

The CHECK command stream program has responsibility in

testing a user's list of classes for completion of the major emphasis courses in Computer Science. The four emphasis courses are C.S. 640, Introduction to Software Engineering; C.S. 700, Translator Design I; C.S. 720, Operating Systems II; and C.S. 760 or C.S. 761, Database Management Systems.

This program was divided with the other curriculum checking program CHECK2, because with only IF statements at my disposal, the program was getting rather long. To provide some degree of modularization, and improve response time, CHECK2 will test for completion of the graduate seminar, as well as the upper level graduate class requirements.

CHECK can take advantage of the fact that courses will be entered by the interface beginning with CLASS-01, and will proceed in order to CLASS-14. This allows for a nesting arrangement of IF statements which test for any numerical value while walking through the sequence of CLASS-xx items.

In other words, if three classes have been entered, the IF test for CLASS-04 will fail, control will 'drop out', and no other CLASS-xx items will be evaluated. In very unscientific testing, it was observed that this nesting approach was found to cut response time for this program by about half, as compared to evaluating all 14 CLASS-xx fields whether they need to be tested or not.

After the CHECK process determines that a CLASS-xx item is active, it next examines the GRADE-xx value in the same

level of association, and determines if a passing mark has been recorded. No assumptions about completion of a class are made about either classes currently enrolled in, or incomplete classes.

Once there is an active class with a passing grade, it is then a matter of testing that class for each of the required emphasis courses. If there is a match under these circumstances, a counter is incremented in the appropriate fashion for that course. The logical details of this operation are spelled out in the Appendix.

After evaluating as many classes as are active, CHECK will output a message to the student describing exactly what emphasis classes he has completed, and what emphasis classes remain to be finished.

3.5 Program CHECK2

Program CHECK2 is similar in form to CHECK. A nesting strategy is again employed that only tests those CLASS-xx items that contain values. A passing grade in an active course is also demanded before any further tests are made. The courses of interest to this program are the graduate seminar, C.S. 897, and a dozen or so courses that have C.S. graduate courses as pre-requisites.

The courses in this latter group that do satisfy the upper level requirement are C.S. 725, Computer Networks; C.S. 750, Advanced Computer Architecture Experiments; C.S.

765, Systems Analysis for Business; and C.S. 785, Numerical Solution of Partial Differiential Equations. Any course numbered between C.S. 800 and C.S. 960, with the exception of C.S. 891, C.S. 898, and C.S. 899, will also complete this requirement.

CHECK2 will report to the user's terminal whether he has completed or not completed both of these requirements. A nice feature that wasn't included in the database, would be to have additional one character fields in COURSE for seminar-passed and upper-level-passed. The CHECK program could then have returned a 'Y' or 'N' value back to the student's record. This feature could help faculty members in making quick graduation checks.

3.6 Program CLASSES

CLASSES is a program used exclusively by the interface. Its job is to determine how many CLASS-xx items are being used by the currently selected record. This information is returned to the calling interface, which then knows what was the last CLASS-xx field in which an entry was made. The interface can then entertain any user requests for updating additional courses, and enter these additional updates in the proper sequential order.

3.7 Text Files

Temporary files were created under the general EDIT facilities of the 8/32 system on the same user disk as the COURSE database and the interface. The temporary files were then transferred with the SCRIPT option to permanent text files. The text files had the limitation of having line lengths of 60 characters, with 20 lines per file. The purpose of this was to make many small text files, each of which would fill one display terminal when called for.

Text files include a description of and pre-requisites needed for each graduate course in Computer Science, Departmental admission requirements, Master's degree requirements in Computer Science, provisional undergraduate courses that one should be exposed to before beginning a Master's program, classes offered for the present and following semester, and short sketches showing the research interests of the Computer Science faculty.

The text files provide the user with helpful information for getting acquainted with what the Master's program has to offer, getting enrolled at Kansas State, or learning all the milestones that graduation demands. Certainly a more imaginative paging system, in conjunction with the interface, would provide the student user with more beneficial information in an easier to use format.

Chapter IV

The Interface

The interface is a Fortran program that monitors the messages and sits between the user, and the COURSE database, the INFO32 command stream programs, and the text files. This interface was developed by Master, and for details see Reference[6]. My purpose in discussing her work is to show how the interface came into being, what the interface gives to the system in terms of data independence, and a brief description of what the interface is.

In preliminary discussions of requirements for this system, one of the foremost requirements was that there would be two entirely different modes of operation. The user mode is composed of various individuals who would choose to exercise some or all of the options available to them. A user would only have access to his own record in the database.

Counterpoised to the various students whose choice of options and frequency of use is random, is the supervisory mode. The frequency of use for this mode may have a random quality to it, as professors would be allowed to browse through any records. Additionally, the Department head would be in charge of purging out-of-date student files, entering financial support data for graduate student employees, and making any modifications or changes to this

system.

The primary difference between these two modes is the extent of knowledge about the database layout. One user can see only a small portion, while the supervisor has access to all of the database and its supporting programs. This interface helps the DBMS in insulating the user's view from the physical representation of data.

The advantages in providing this data independence is that the database administrator has more flexibility in rearranging the database. The user is now unaware of the physical organization and logical relationships within the data. This is justifiable on the grounds that the organizational relationships would be of no proper use to him.

The interface also furnishes security, an issue not unlike that of data independence. Early on it became apparent that INFO32 could not duplicate a database definition, so that each new user could not be given their own small database. Given that there would be just one large database with many records, security for COURSE had to be improved. [7]

The social security number is still the unique identifier for each record. As explained earlier, a hash key password is generated by the interface upon creation of a database record, and displayed only once to the person. This student must repeat this hash key in every access operation that follows. This method is similar to

protection offered accounts in electronic fund transactions at some banks. [1]

At the macro level, the INFO32 system allows for a four character password that can be placed on data items within a database. The entirety of COURSE is protected by a password at the zero control level, with all data items being hidden at this level. Perhaps this is adequate data security for a prototype course of study database.

Master's interface [6] supplies added data independence and security. Its primary mode of operation is the interception of commands from the user's console, the interpretation of this command, and the submission of a message to the INFO32 DBMS based on this command. Any error messages like 'invalid data' originating in INFO32 would be passed directly to the user's terminal. Because of the interpretting functions, this interface could be thought of as a small, but specialized compiler.

A normal interaction would start by the student being signed on to the account number containing this system on the Interdata mini. The user would then be confronted with a display at his terminal which would list the options open to him. The interface would prompt for an answer to these choices, and would next execute the option of the student's reply. This approach to monitoring interactions and providing options is known as menu selection.

The student can usually complete the phase he is in by responding with a 'STOP'. This will return control to the

original menu, with a new choice of options now possible being presented.

The interface shields the user not only from how the database and text files were organized, but also from any of the conversational instructions in the INFO32 repertoire. In this manner, the interface fulfills the original purposes of this project. The student can only use database and the the informational files, while the supervisor has control over how this system is physically and logically put together.

Chapter V

Summary

5.1 Analysis

My overall impression of this project is a favorable one. I was exposed to all elements of a database, INFO32. Because of the necessity for the interface, many of the English language commands that make INFO32 attractive to novice users in a business environment were not fully utilized. There was also an 8/32 system problem that prevented the use of the report generation feature.

My complaints concerning INFO32 centered around its inflexibility. The static allocation of data items limited much of the database design. The impact of static allocation was felt in the programming of the command stream programs. These programs had no looping action, or subroutine calls available to them.

The burden of programming these programs fell on IF statements. This was tedious work, and in terms of measuring program complexity, nested IF statements rank as a big contributor. I didn't like the editing features of INFO32 when changing parts of the command stream programs.

The logical operator CN, meaning contains, would have been useful in several set operations. From my experience, the CN operator could only be used with one constant, thus

reducing this operator to that of equality.

My last complaint involving INF032 concerned the DEFINE command, which creats a new database. Everything is fine as long as you do not miss a keystroke. Instead of correcting the mistake, one must define the database anew.

This project provided me with good experience in working in all phases of an operation. I was exposed to eliciting requirements, designing and implementing these requirements, interacting and working with another team member, and of course, documenting all stages of this process.

This prototype database could undoubtedly be expanded. Future directions that might prove to be fruitful are the development of a monitor that would allow concurrent users of this database, the development a Bachelor's degree course of study, and the addition of the Computer Science Master's applicant's files as a subschema to COURSE.

As noted earlier, the text files could be expanded so that they supply new or more comprehensive information. Improvement of this prototype could also be done in the areas of data security, type checking, report generation, and the resolving of the course numbers for transfer credits from other schools.

5.2 Concluding Remarks

The purpose of this report has been to describe the process by which a database is developed. This process

started as an idea of extending upon previous attempts at automating or producing 'paperless' student file records at Kansas State. Chapter One amounted to citing these recent prototype systems, and expressing the present need for enhancing these attempts.

Chapter Two was a listing of exactly what the system was going to accomplish given the original idea. The specification requirements for this prototype were then explicitly stated.

Chapters Three and Four explained how these specification requirements were going to be actualized. The rationale behind the definition of the most important items in the database was presented, as were descriptions of what tasks the various command stream programs in INFO32 were to perform.

Chapter Four showed the perspective of the backend interface between the student user and the database. The interface's responsibilities and functions were also explained at this time.

This report was an effort to describe the design and implementation of a database as it was shaped from the first estimates to its final form. As such, it is hoped that this documentation will provide the starting point for future improvements to this project.

BIBLIOGRAPHY

1. Bender, Mark G., EFTS Electric Funds Transfer Systems, Kennikat Press: New York, 1975.

2. Data Manipulation Language Programmer's Reference Guide, Release 3.1, Cullinane Corporation, Boston, Mass., April 1975.

3. INFO32 User's Manual
Henco, Inc., Wellesley, Mass., 1977.

4. Long, Harvey A., Implementation of a Prototype Data Base for Advising Computer Science Students, a Master's Report, Department of Computer Science, Kansas State University, Manhattan, Kansas.

5. Maryanski, Fred, Northsworthy, K. A., and Northsworth, K. E., System Architecture For Distributed Databases,
Technical Report TR CS 78-15, Computer Science Department, Kansas State University, Manhattan, Kansas, 1978.

6. Master, Hilla, Master's Report,
Department of Computer Science, Kansas University, Lawrence, Kansas. (in preparation)

7. Tsichritzis, D. C., and Lochovsky, F. H.,
Data Base Management Systems,
Academic Press: New York, 1977.

8. Voelz, James H., and Garland, Robert L.,
Prototype of a Database Management System For Academic records Utilizing the Integrated Database Management System, a Master's Report, Department of Computer Science, Kansas State University, Manhattan, Kansas, 1975.

APPENDIX A

USER'S GUIDE

To use the Master's degree database prototype, one needs to be signed on to the account number which contains this system, which for the moment is account number 13. Then in response to a prompt, the user inputs 'INFO', and the backend interface will then assume control.

For interactive users who do not wish to travel to Fairchild Hall on the K-State campus, this system can be called up by dialing 913-537-8456.

The input command steps to gain access to the database can be seen in the following example.

```
*SIGNON USERNAME,13,PASSWORD
*INFO
```

Username and password are supplied by the user

APPENDIX B

SUPERVISORY MANUAL

Using the COURSE Database

The supervisor first needs to make an 'end run' around the interface's control over the database and INFO32 programs. To do this, the supervisor needs to sign on to this account in the normal manner. The next prompt requires an input for the system 2 disk, followed by the entry of 'INFO'.

You must then break out of what would normally be an 'INFO32' entry, and pause that task. You then want to get on the user 6 disk and continue your task. You now need to type 'INFTST' (which is Master's renaming of INFO32) in response to the 'greater than' or '>' prompt.

For this interchange listing, look at Figure B.1 on the following page.

Procedure for Gaining Access to COURSE

```
*V SYS2
*INFO
  - INFO32/R01
  -ENTER USER NAME>-> hit break key
*PA
*TASK PAUSED
*V USR6
*CO
>INFTST
```

Figure B.1

You will now be able to make use of any of the regular INFO32 instructions. You will also be able to access any of the database records, provided that you know the current password that protects COURSE.

To end the session, you need to type in 'QUIT' when you see an 'ENTER COMMAND>->' prompting. This will take you back to the 'ENTER USER NAME>->' level, which is a comparable level to the 'INFTST' entry. Now type in 'STOP', and you can sign off.

For example--

```
ENTER COMMAND>->QUIT
ENTER USERNAME>->STOP
*SIGNOFF
```

Password Changing

Passwords on an INFO32 database can be any combination of letters or characters up to four characters in length. Protection keys give security at the zero or control level, as well as levels one, two, and three. COURSE now has a password on it at the control level, which has precedence over the other three levels.

To change this or any other password, you must first select the database, supply the current password, and use the PROTECT command followed by the new password(s). The passwords should be inputted in order, with the first being the 0 level, the second password protecting items at level 1, etc. Since all items in COURSE are hidden at the control level, all other levels of passwords seem superfluous.

```
ENTER COMMAND>->SEL COURSE
 ENTER PASSWORD>->WWWW
       6 RECORD(S) SELECTED
ENTER COMMAND>->PROTECT YYYY
```

Purging Old Records

There should be a time when some records in COURSE become obsolete. Students could transfer, drop out, or graduate. To remove a record from a database, you must first select or reselect until you are dealing with just that obsolete file. When you are sure that this is the correct student record, you need to enter 'PURGE' in response to the enter command suggestion.

INFO32 will reply with--

```
THIS COMMAND WILL DELETE ALL SELECTED RECORDS.
DO YOU WISH TO CONTINUE  (Y OR N) ?>
```

A 'Y' keystroke will take care of the rest.

Query Examples in INFO32

To set up COURSE as the current database, you have this general format to work with:

```
SELECT database-name [FOR logical expression]
```

which could translate to this--

```
SEL COURSE
```

And the next prompt will demand that you input the correct password that covers this database.

To choose student Smith's record, you could select COURSE, enter the password, and then enter

```
RESELECT FOR LAST EQ SMITH
```

This is equivalent to saying

```
SEL COURSE FOR LAST EQ SMITH
```

At this point, INFO32 will come back with how many records have been selected. There might be several Smith's, so the answer from INFO32 could be

```
4 RECORD(S) SELECTED
```

You should then do a reselect for 'FIRST EQ JIM', and that should get the record you want. To browse through Jim Smith's file, just type in LIST, and all 82 items will be displayed.

You could select by SSN, since it is the unique key to this database. The logical expression would be something like select for 'SSN EQ 515151515'. Other logical operators are NE, LE, GT, CN (contains) and NC (not containing). These last two operators were a disappointment, as they seemed to only work with one operand on each side of the operator, thus reducing its effectiveness.

To get all the students whose grade point averages are below 3.0, and who are employed as GTA's, you would input

```
SEL COURSE FOR GPA LT 3
RESEL FOR SUPPORT-CLASS EQ G.T.A
```

APPENDIX C

FUNCTIONAL SPECIFICATIONS

To change any of the database definitions for any of the items in COURSE, you must first erase COURSE by selecting it, and providing the proper password. This is a serious matter, as it involves wiping out all records contained in the database. This inflexibility in INFO32's database definition features suggests that only non-trivial changes should be considered for COURSE, unless you could save these records and reconstruct them in the new COURSE.

```
ENTER COMMAND>-> SEL COURSE
ENTER PASSWORD>-> password
ENTER COMMAND>-> ERASE COURSE
```

This will completely take out COURSE from the user disk. You can also do a DELETE instruction using COURSE'S internal name, but COURSE will remain on the user's INFO32 directory.

Now that you have destroyed the database, you should now use the DEFINE command to create COURSE with the changes you want. You have to supply the name, size, type, and number of decimal places if any, to every item name.

Protection can either be done at program creation by entering the control level password immediately after the

database name:

DEFINE COURSE WWWW

You can get the same effect by using the PROTECT command, after selecting COURSE.

PROTECT WWWW

All 82 data items were hidden at the 0 control level in the tedious process of using the HIDE command. You have to input

ENTER COMMAND>-> HIDE itemname

and then input a 0 after the prompt for

PROTECTION LEVEL>-> 0

INFO32 Command Stream Programs

Command stream programs in INFO32 are sequential programs which are composed of three parts. These parts are like a pre-processor-- Section One, a processor-- Section Two, and a post-processor-- Section Three.

The interface will select the database record of the terminal user for use by the command stream program, and could be thought of as playing the role of Section ONe. I used Section Two exclusively for these programs, as this was the only section of the three that allowed IF statements.

Program Section Two will contain all initializations necessary, the program steps comprised mainly of IF statements that carry out the functions mentioned in Chapter III, and displays the calculations back to the user. So really, no post-processor is needed for our purposes.

To execute a command stream program, you need to select all or part of a database's records. The 'enter command' prompt should be answered by a 'RUN program name' reply.

To alter any of these programs, you need to again select a database. Next type in

PROGRAM program-name

in response to the request for a command prompt. If the program does not exist, you can enter whatever program lines you want for all three sections. This is the method of creating new programs.

In modifying an existing program, INFO32 will display the entire program line by line, and then give the user a prompt of '>' . The user has the choice of deleting lines by typing in the line number with a return key strike; or modifying existing lines by entering the line number with the changes, plus return; or inserting new lines by proceeding the new lines with unique line numbers that fall at the numerical point in the program where they are needed.

This editing must be done from low line number forward in the program, as there is no provision for backing up the text from where you have editted; except by entering 'PROGRAM-program name', watching the program being displayed, and starting your editing anew.

Program GPA1

Program GPA1 will calculate total grade credits, total credit hours, grade point average, and the total number of 'C' and 'D' credit hours. IF-ENDIF groupings allow GPA1 to check the CLASS-xx item to see if its in the range of 286600 to 286999-- the Computer Science graduate courses. If this course is not 286891, the GRADE-xx item will be examined, and $NUM1 set accordingly.

INFO32 lets the programmer use temporary floating point variables that run by name from $NUM1 to $NUM20. I used $NUM1 as the variable which helped translate letter grades into comparable grade credits. Thus, for an 'A', $NUM1 was assigned a 4, a 'B' would make it a 3, and so on.

$NUM1 was reset to a value of 0 after each CLASS-xx, by saying 'CALC $NUM1 = 0'. This was done so that a 0 value would not have to be assigned to $NUM1 for each 'F', and because I wanted the TOTAL-CREDITS and TOTAL-HOURS incremented only upon a new value that represented a new class with possibly a new grade.

In other words, $NUM1 is set to 0 after each class, and the class must be a Computer Science graduate course with a grade that will change this variable to a value that is greater than zero. In this way, total credits and hours are only increased under these circumstances.

TOTAL-HOURS will be increased by any HOURS-xx which has a GRADE-xx of 'F'. $NUM3 also keeps a running total of the 'F' hours in order that they can be subtracted from

TOTAL-HOURS after the GPA has been figured. $NUM2 keeps a running total on the number of 'C' and 'D' credit hours, so that possible problems in this area can be discovered.

$NUM7 is the variable for total credit hours without the 'C' and 'D' hours. $NUM8 stores a value which is 3 times the 'C' and 'D' hour figure, and is compared to $NUM7 for the C-D deficiency. $NUM10 is a temporary field used in subtracting TOTAL-HOURS from the 30 needed for graduation.

The nesting of IF statements in INFO32 can reach up to a level of 25, and in these programs, a level of 14 is attained. The 'IF CLASS-xx GT 0' statement is used as the basic weapon for nesting. The premise is that all classes will be entered in sequential order from 1 to 14, so those classes with no entries will have a value of 0. As explained earlier, if you do not have any class entries after CLASS-04, you needn't check any other classes whatsoever.

This program wound up with 14 ENDIF'S at the last of Section Two, as they closed up the IF'S checking on CLASS-xx entries. I think you will get the idea of GPA1 by Figure C.1, which lists the initializations, the CLASS-01 segment, and the displays made at the end. Notice that the zero divide is not a possibility in calculating the grade point average.

Source Code for Program GPA1

```
10000 PROGRAM SECTION ONE
20000 PROGRAM SECTION TWO
20010 CALC $NUM1 = 0
20020 CALC $NUM2 = 0
20030 CALC $NUM3 = 0
20040 CALC TOTAL-HOURS = 0
20050 CALC TOTAL-CREDITS = 0
20060 IF CLASS-01 GT 0
20070  IF CLASS-01 GT 286600
20080   IF CLASS-01 LT 286999
20090    IF CLASS-01 NE 286891
20100     IF GRADE-01 EQ A
20110      CALC $NUM1 = 4
20120     ENDIF
20130     IF GRADE-01 EQ B
20140      CALC $NUM1 = 3
20150     ENDIF
20160     IF GRADE-01 EQ C
20170      CALC $NUM1 = 2
20180      CALC $NUM2 = $NUM2 + HOURS-01
20190     ENDIF
20200     IF GRADE-01 EQ D
20210      CALC $NUM1 = 1
20220      CALC $NUM2 = $NUM2 + HOURS-01
20230     ENDIF
20240     IF GRADE-01 EQ F
20250      CALC $NUM3 = $NUM3 + HOURS-01
20260      CALC TOTAL-HOURS = TOTAL-HOURS + HOURS-01
20270     ENDIF
20280     IF $NUM1 GT 0
20290      CALC TOTAL-HOURS = TOTAL-HOURS + HOURS-01
20300      CALC TOTAL-CREDITS = TOTAL-CREDITS + HOURS-01 *
           $NUM1
20310     ENDIF
20320    ENDIF
20330   ENDIF
20340  ENDIF
```

Figure C.1

And this approach continues through the other 13 classes. After all the classes are checked, this program segment will be at the end of the GPA1 to print the information.

```
24530 IF TOTAL-HOURS GT 0
24540  CALC GPA = TOTAL-CREDITS / TOTAL-HOURS
24550 ENDIF
24560 DISPLAY ' YOUR GPA IS ',GPA
24570 CALC TOTAL-HOURS = TOTAL-HOURS - $NUM3
24580 CALC $NUM10 = 30 - TOTAL-HOURS
24590 IF $NUM10 GT 0
24600  DISPLAY 'YOU HAVE ',$NUM10,'GRADUATE HOURS LEFT'
24610 ENDIF
24620 DISPLAY 'YOU HAVE COMPLETED ',TOTAL-HOURS,' HOURS OF'
24630 DISPLAY 'C.S. GRADUATE WORK'
24640 CALC $NUM7 = TOTAL-HOURS - $NUM2
24650 CALC $NUM8 = $NUM2 * 3
24660 IF $NUM8 GT $NUM7
24670  DISPLAY 'TOO MANY LOW GRADES'
24680  DISPLAY 'FOR EVERY HOUR OF C OR D YOU HAVE'
24690  DISPLAY 'YOU MUST HAVE 3 TIMES THAT AMOUNT'
24700  DISPLAY 'IN A OR B HOURS TO GRADUATE!'
24710 ENDIF
30000 PROGRAM SECTION THREE
```

Figure C.1 (continued)

Program CHECK

CHECK will examine a student's list of classes for completion of the major emphasis courses. The nesting arrangement is similar to that of GPA1, as the first CLASS-xx with a value of 0 breaks the chain. CHECK does not worry about whether a class is a Computer Science graduate class, per se. This eliminates the three IF tests that control for the range of these courses in each CLASS-xx in GPA1.

CHECK does look at whether the class had a legitimate grade, and if this was so, $NUM3 is assigned a value of 1. ($NUM3 is again reset to 0 following each CLASS-xx). Next, explicit IF statements check the current CLASS-xx for equality with 286640, 286700, 286720, and 286760 or 286761.

The latter two classes will cause $NUM1 to be increased by 1000. The 286640 class means that one will be added to $NUM1, 286700 will add 10 to $NUM1, and a class of 286720 adds 100 to this variable.

There will be 16 combinations of $NUM1 depending upon what mix of classes a student has completed. After all the classes have been evaluated, there are 16 IF groupings which test for, and then display the appropriate message in regards to the major course emphasis requirement. These combinations are not unlike that of binary 0 to 15, as $NUM1 is compared to a value like 1110. Figure C.2 follows.

Source Code for Program CHECK

```
10000 PROGRAM SECTION ONE
20000 PROGRAM SECTION TWO
20010 CALC $NUM1 = 0
20020 CALC $NUM3 = 0
20030 IF CLASS-01 GT 286600
20040  IF GRADE-01 EQ A
20050   CALC $NUM3 = 1
20060  ENDIF
20070  IF GRADE-01 EQ B
20080   CALC $NUM3 = 1
20090  ENDIF
20100 IF GRADE-01 EQ C
20110   CALC $NUM3 = 1
20120  ENDIF
20130  ENDIF
20140  IF GRADE-01 EQ D
20150   CALC $NUM3 = 1
20160  ENDIF
20170  IF $NUM3 EQ 1
20180   IF CLASS-01 EQ 286640
20190    CALC $NUM1 = $NUM1 + 1
20200   ENDIF
20210   IF CLASS-01 EQ 286700
20220    CALC $NUM1 = $NUM1 + 10
20230   ENDIF
20240   IF CLASS-01 EQ 286720
20250    CALC $NUM1 = $NUM1 + 100
20260   ENDIF
20270   IF CLASS-01 GT 286759
20280    IF CLASS-01 LT 286762
20290     CALC $NUM1 = $NUM1 + 1000
20300    ENDIF
20310   ENDIF
20320  ENDIF
```

Figure C.2

There will be 16 tests made on the variable $NUM1. The displays tell the user what major emphasis classes he has taken, and which ones remain.

```
24360 IF $NUM1 EQ 0
24370  DISPLAY 'YOU HAVE NOT COMPLETED ANY OF THE FOUR'
24380  DISPLAy 'REQUIRED CLASSES'
24390  DISPLAY 'YOU NEED: C.S. 640 INTRO TO SOFTWARE ENGI.'
24400  DISPLAY '           C.S. 700 TRANS. DESIGN I'
24410  DISPLAY '           C.S. 720 OPER. SYSTEMS II'
24420  DISPLAY '           C.S. 761 D.B. MAN. SYSTEMS'
24430 ENDIF
```

There will be 15 other displays similar to this one.

Figure C.2 (continued)

Program CHECK2

Program CHECK2 uses the nesting based on CLASS-xx entries in looking for completion of the upper level class requirement and the graduate seminar. $NUM1 is reset to 0 after every nesting level, as it is the variable that will be changed to 1 if a GRADE-xx is 'A', 'B', 'C', or 'D'.

A grade such as this allows admittance to a series of IF statements which will change $NUM2 to 1 in the event that the class is either 286725, 286750, 286765, or 286785.

Any class between C.S. 800 and C.S. 960 will cause $NUM4 to be bumped by 1, but any class that is also between C.S. 891 and C.S. 899 is an exception to this general rule, and will not meet this requirement. This latter condition will cause $NUM3 to be increased by 1. Only if $NUM3 is less than $NUM4 will the upper level requirement be met by this route.

An instance of C.S. 897, the graduate seminar, will cause $NUM5 to be increased to a value of 1. Displays are made to the user based on the values these temporary variables took on during this program. A $NUM2 value of 1 means that the upper level requirement has definitely been completed, while a value of 0 could still mean that the requirement could have been met by the 800 or 900 level classes. $NUM3 must be less than $NUM4 for this requirement to be met by those classes. A $NUM5 value of 1 means that

the graduate seminar has been finished.

Figure C.3 gives a picture of CHECK2.

Source Code for Program CHECK2

```
10000 PROGRAM SECTION ONE
20000 PROGRAM SECTION TWO
20010 CALC $NUM1 = 0
20020 CALC $NUM2 = 0
20030 CALC $NUM3 = 0
20040 CALC $NUM4 = 0
20050 CALC $NUM5 = 0
20060 IF CLASS-01 GT 0
20070  IF GRADE-01 EQ A
20080   CALC $NUM1 = 1
20090  ENDIF
20100  IF GRADE-01 EQ B
20110   CALC $NUM1 = 1
20120  ENDIF
20130  IF GRADE-01 EQ C
20140   CALC $NUM1 = 1
20150  ENDIF
20160  IF GRADE-01 EQ D
20170   CALC $NUM1 = 1
20180  ENDIF
20190  IF $NUM1 EQ 1
20200   IF CLASS-01 EQ 286725
20210    CALC $NUM2 = 1
20220   ENDIF
20230   IF CLASS-01 EQ 286750
20240    CALC $NUM2 = 1
20250   ENDIF
20260   IF CLASS-01 EQ 286765
20270    CALC $NUM2 = 1
20280   ENDIF
20290   IF CLASS-01 EQ 286785
20300    CALC $NUM2 = 1
20310   ENDIF
20320   IF CLASS-01 EQ 286897
20330    CALC $NUM5 = 1
20340   ENDIF
20350   IF CLASS-01 GE 286800
20360    IF CLASS-01 LE 286960
20370     CALC $NUM4 = $NUM4 + 1
20380    ENDIF
20390    IF CLASS-01 GT 286890
20400     IF CLASS-01 LT 286900
20410      CALC $NUM3 = $NUM3 + 1
20420     ENDIF
20430    ENDIF
20440   ENDIF
20450  ENDIF
20460 ENDIF
```

Figure C.3

After evaluating all 14 classes if necessary, and amounting to almost 600 lines of code, CHECK2 then makes displays according to this segment:

```
25930 IF $NUM2 EQ 1
25940  DISPLAY 'YOU HAVE COMPLETED THE UPPER LEVEL'
25950  DISPLAY 'REQUIREMENT'
25960 ENDIF
25970 IF $NUM2 EQ 0
25980  IF $NUM3 GE $NUM4
25990   DISPLAY 'YOU HAVE YET TO COMPLETE THE'
26000   DISPLAY 'LEVEL CLASS REQUIREMENT'
26010  ENDIF
26020 ENDIF
26030 IF $NUM2 EQ 0
26040  IF $NUM3 LT $NUM4
26050   DISPLAY 'YOU HAVE COMPLETED THE UPPER LEVEL
        REQUIREMENT'
26060  ENDIF
26070 ENDIF
26080 IF $NUM5 EQ 1
26090  DISPLAY 'YOU HAVE COMPLETED THE GRADUATE
       SEMINAR'
26100 ENDIF
26110 IF $NUM5 EQ 0
26120  DISPLAY 'YOU HAVE YET TO COMPLETE THE GRADUATE
SEMINAR'
26130 ENDIF
30000 PROGRAM SECTION THREE
```

Figure C.3 (continued)

Establishing Text Files

The text files were first created by requesting system space with the allocate command. Seven files were established from TEXT01 through TEXT07. An example of this command is:

```
*AL TEXT01.XXX,IN,80
```

Temporary files were needed to contain the textual information with embedded SCRIPT commands. These SCRIPT instructions allowed for the formatting of the words into the size of a display screen-- approximately 20, 60 character length lines. This command creates the temporary script files:

```
*PEDIT ,RUDE01.txt
```

After the temporary files were complete, this file was transferred to the permanent text file via the SCRIPT option

like this

```
*SCRIPT RUDE11,TEXT03.XXX
```

To modify the existing TEXTxx series of files, you need to do a PEDIT instruction on the temporary file--

```
*PEDIT RUDE12
```

And then make the changes that are warranted.
Next, script this modified file back to the permanent file that RUDE12 is associated with.

```
*SCRIPT RUDE12,TEXT04.XXX
```

The current and next semester class offering files will definitely need to be maintained. One suggestion is to script the temporary file for the next semester over to the permanent file of the current semester when a change is needed.

```
*SCRIPT RUDE15,TEXT05.XXX
```

What you need to do next is to establish a script temporary file with the new next semester class listings, and script it to the TEXT06 file. The EXTRACT option is handy for browsing through these permanent files. Any new files should be created in the described manner, and then adjustments made in the interface [6] that will both offer this choice to the user, and display the new text file on demand.

Here is the directory of permanent and script files.

Directory for Text Files

FILE NAME	DESCRIPTION	SCRIPTED TO
RUDE01	Course Numbers CS 640 to CS 662	TEXT01
RUDE02	Course Numbers CS 665 to CS 690	TEXT01
RUDE03	Course Numbers CS 700 to CS 725	TEXT01
RUDE04	Course Numbers CS 730 to CS 761	TEXT01
RUDE05	Course Numbers CS 765 to CS 798	TEXT01
RUDE06	Course Numbers CS 800 to CS 840	TEXT01
RUDE07	Course Numbers CS 870 to CS 899	TEXT01
RUDE08	Course Numbers CS 900 to CS 920	TEXT01
RUDE09	Course Numbers CS 926 to CS 960	TEXT01
RUDE10	Department Admission Requirements	TEXT02
RUDE11	Dept. MS Degree Requirements	TEXT03
RUDE12	BS Equivalent Experience needed	TEXT04
RUDE13	Current Classes Offered	TEXT05
RUDE14	KSU CS Faculty part 1	TEXT07
RUDE15	Next Semester Classes Offered	TEXT06
RUDE16	KSU CS Faculty part 2	TEXT07

Figure C.4

A MASTER'S DEGREE COURSE OF STUDY DATABASE

by

GEORGE RICHARD HUGHES

B.A., Kansas University, 1973

M.A., University of Regina, 1974

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1979

This report will describe the implementation of a database using the conversational data language system INFO32. This database will be comprised of up-to-date, but unofficial, academic and personal facts about master's students in Computer Science.

This on-line implementation is designed to serve several different purposes. The first purpose is in providing information, as students can choose to view reports about various aspects of the Computer Science Master's program at Kansas State.

Secondly, this implementation will securely store courses completed or enrolled in for individual students. The credit hours and the grade received-- if any-- for each course, and the semester when the course was taken can also be entered. The student may on demand, request that this grade point average be calculated, or have his list of courses evaluated for degree requirement satisfaction.

The third purpose of this implementation is to duplicate certain Departmental personal files. Faculty members can be spared the tedium of shuffling through personal files when checking on the progress of a graduate student.

These three purposes imply two different modes of users. The first mode of user will be the graduate student interested in either entering personal data, or reviewing aspects of the Master's program. The individual student has responsibility for entering his personal data. This type of user will be limited by an interface program, written by

Master, which monitors communication between the student user, and just his own records contained in the INFO32 database.

The supervisory mode will be used by the Department head, or his designate. Activities of this mode include the perusal of all student records, the request for statistical summaries of types of support for graduate students, and the purging of old (graduated) student files.

In conclusion, this database will be used interactively both in Manhattan, and from off campus locations. The prototype will be a counselling and informational aid to prospective Master's candidates.