An Interactive Information System for Tracking Student Academic Progress and for Projecting Quarterly Course Enrollments

1976

Marilyn Sellers Khorsandi

University of Central Florida

Find similar works at: https://stars.library.ucf.edu/rtd

University of Central Florida Libraries http://library.ucf.edu

Part of the Operational Research Commons

STARS Citation

Khorsandi, Marilyn Sellers, "An Interactive Information System for Tracking Student Academic Progress and for Projecting Quarterly Course Enrollments" (1976). Retrospective Theses and Dissertations. 226.
https://stars.library.ucf.edu/rtd/226

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
AN INTERACTIVE INFORMATION SYSTEM FOR TRACKING STUDENT ACADEMIC PROGRESS AND FOR PROJECTING QUARTERLY COURSE ENROLLMENTS

BY

MARILYN SELLERS KHORSANDI
B.S., Florida Technological University, 1973

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science: Operations Research in the Graduate Studies Program of the College of Engineering of Florida Technological University

Orlando, Florida
1976.
ABSTRACT

An interactive information system has been developed for the College of Engineering to promote faster and more accurate trial advisement interviews between professors and the students they advise. This system also provides departmental administrative personnel with information to guide course scheduling. The system has the capability of providing the following information concerning undergraduate students enrolled in the College of Engineering:

1. A student's academic progress toward graduation.
2. A projection for four quarters of enrollments in engineering core courses.
3. The current status of any course offered as a requirement for obtaining a Bachelor's Degree in one of the Engineering disciplines.
4. The current status of any program leading to a Bachelor's Degree in any of the Engineering disciplines.

The system provides the user with the additional capabilities to:

1. Update the database containing information pertaining to students enrolled in the College of Engineering.
2. Update the database containing information concerning courses offered as a requirement for obtaining a Bachelor's Degree in one of the Engineering disciplines.
3. Update the database containing information concerning the degree programs offered by the College of Engineering. The system is user oriented and operation does not require any knowledge of computer programming. Complete documentation is available to facilitate trouble-free user operation and programming any desireable additional capabilities in the future.
ACKNOWLEDGMENTS

I wish to thank the members of my supervisory committee: Dr. Benjamin W. Lin and Dr. Harold I. Klee for their assistance in the preparation of this thesis.

I am particularly grateful to my committee chairman, Dr. Christian S. Bauer, P.E., whose friendship and inspirational guidance encouraged me throughout my classwork and the preparation of this thesis.

Additional thanks goes to Mr. Tom Peeples of the Florida Technological University Computer Center who answered many questions in connection with this thesis and earlier classwork.

I am especially indebted to Mr. Chris Maukonen who supported me with his knowledge of the IBM 360/65 time-sharing system during my efforts to convert my original batch version to an interactive program.

Finally, I wish to thank my husband, Marshall, for his love, patience, strength, and encouragement during this two year effort to each attain our degrees.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>**I. **INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td><strong>II. MANAGEMENT INFORMATION SYSTEMS RELATED TO STUDENT ADVISEMENT--A LITERATURE SURVEY</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>III. PROGRAMMING LANGUAGE USED--PL/I.</strong></td>
<td>12</td>
</tr>
<tr>
<td><strong>IV. SYSTEMS ANALYSIS</strong></td>
<td>19</td>
</tr>
<tr>
<td>Databases Required</td>
<td>19</td>
</tr>
<tr>
<td>Student database--structure and contents</td>
<td>19</td>
</tr>
<tr>
<td>Courses database--structure and contents</td>
<td>23</td>
</tr>
<tr>
<td>Course requirements database--structure and contents</td>
<td>24</td>
</tr>
<tr>
<td>Programs Required</td>
<td>26</td>
</tr>
<tr>
<td><strong>V. BUILDING THE DATABASES</strong></td>
<td>28</td>
</tr>
<tr>
<td>Building the Courses Database</td>
<td>28</td>
</tr>
<tr>
<td>Building the Courses Requirements Database</td>
<td>28</td>
</tr>
<tr>
<td>Building the Student Database</td>
<td>30</td>
</tr>
<tr>
<td><strong>VI. ADVISE--STUDENT ADVISEMENT AND COURSE ENROLLMENT PROJECTIONS PROGRAM</strong></td>
<td>33</td>
</tr>
<tr>
<td>Assumptions</td>
<td>33</td>
</tr>
<tr>
<td>Logic Flow</td>
<td>33</td>
</tr>
<tr>
<td><strong>VII. UPDATING THE DATABASES</strong></td>
<td>50</td>
</tr>
<tr>
<td>COUPDT--Courses Database Update Program</td>
<td>50</td>
</tr>
<tr>
<td>COREUPD--Course Requirements Database Update Program</td>
<td>53</td>
</tr>
<tr>
<td>STUPDT--Student Database Update Program</td>
<td>65</td>
</tr>
<tr>
<td><strong>VIII. SYSTEM INTEGRITY AND SECURITY</strong></td>
<td>77</td>
</tr>
<tr>
<td><strong>IX. CONCLUSIONS AND RECOMMENDATIONS</strong></td>
<td>79</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>A</td>
<td>Listing and Output of BUILD_S</td>
</tr>
<tr>
<td>B</td>
<td>Listing and Output of BUILD_C</td>
</tr>
<tr>
<td>C</td>
<td>Listing and Output of BUILD_C_R</td>
</tr>
<tr>
<td>D</td>
<td>Listing of SADVISE</td>
</tr>
<tr>
<td>E</td>
<td>Listing of COUPDT</td>
</tr>
<tr>
<td>F</td>
<td>Listing of COREUPD</td>
</tr>
<tr>
<td>G</td>
<td>Listing of STUPDT</td>
</tr>
<tr>
<td>H</td>
<td>General User Instructions</td>
</tr>
<tr>
<td>I</td>
<td>User Instructions for SADVISE With Sample Output</td>
</tr>
<tr>
<td>J</td>
<td>User Instructions for COUPDT With Sample Output</td>
</tr>
<tr>
<td>K</td>
<td>User Instructions for COREUPD With Sample Output</td>
</tr>
<tr>
<td>L</td>
<td>User Instructions for STUPDT With Sample Output</td>
</tr>
<tr>
<td>M</td>
<td>Listing of Time-Sharing and Job Control Programs</td>
</tr>
<tr>
<td>N</td>
<td>Process For Recovery of Lost Data</td>
</tr>
<tr>
<td>REFERENCES</td>
<td></td>
</tr>
</tbody>
</table>
LIST OF FIGURES

1. Flowchart of BUILD C--the Program for Building the Courses Database ................................. 29
2. Flowchart of BUILD C R--the Program for Building the Course Requirements Database .................. 31
3. Flowchart of BUILD S--the Program for Building the Student Database .................................. 32
4. Flowchart of SADVISE--the Student Advisement and Course Enrollments Projection Program .......... 39
5. Flowchart of COUPDT--the Courses Database Update Program ................................................. 54
6. Flowchart of COREUPD--the Course Requirements Database Update Program ............................... 67
7. Flowchart of STUPDT--the Student Database Update Program .................................................... 73
8. Summary of Interactions ..........................
CHAPTER I

INTRODUCTION

In order for a student to graduate from Florida Technological University, the student must satisfy certain requirements. The basis of these requirements entails completing a prescribed set of courses which leads to proficiency in the student's major. This curriculum is a product of university requirements, college requirements, and departmental requirements. Thus, barring any complications such as switching majors, failing courses, etc., a student must complete approximately fifty university courses in order to receive a Bachelor's Degree.

At registration, each student is posed with the problem of deciding which courses to take during the approaching quarter. In order to make intelligent selections from the courses to be offered during the oncoming quarter, the student must consider: the appropriate major's prescribed curriculum; the list of courses to be offered with their respective times to be offered; and the prerequisite and corequisite structure of the courses being considered.

To help the student decide which courses to take, pre-registration trial-advisement interviews are set up between university professors, and students in their respective departments. Ideally, the student has already considered such factors as required courses, courses to be offered, and personal eligibility for courses to be
offered; and has tentatively decided on a feasible schedule of first-choice courses plus two or three second-choice courses before arriving for the student-professor interview. Unfortunately, this is rarely the case. More normally: the student arrives upset and confused--plans to take a particular set of courses have been foiled since the courses are not being offered. The student just picked up a schedule bulletin; discovered the absence of first choice courses; and does not have the slightest idea for an alternative schedule. Consequently, instead of a five to ten minute "approval" interview, the professor must partake in a forty-five minute research effort. The professor must obtain and examine the student's file, a university catalog, and the schedule bulletin in order to determine a feasible schedule for the now helpless and befuddled student.

This state of confusion during trial-advisement interviews results in several hours of the professor's time being wasted every quarter. This is an extremely expensive waste especially when other useful ways of spending this time are considered such as technical advisement to students in class, or research efforts for the university.

In order to alleviate this problem and provide professors with the information necessary to keep them constantly in a position to conduct an "approval" interview regardless of a student's state of preparation upon arrival, an interactive information system has been developed. Since it was judged as "not within the scope of a single
Master's thesis to develop an information system for the entire university, the target of this project is the College of Engineering undergraduate degree programs, excluding the Department of Engineering Technology.

Administrative personnel in the College of Engineering defined the following requirements of such an advisement information system:

1. Operable from a remote terminal - preferably interactive.
2. After considering the prerequisite and corequisite structure of courses to be offered, prepares a one page report on each student indicating that student's academic progress toward graduation and presents a trial advisement schedule applicable to the approaching quarter for that student.
3. Projects, for each course offered by the College of Engineering in connection with one of the B.S.E. programs, the number of students enrolled, or eligible to enroll, for each of the next four quarters.
4. User oriented--requires no previous knowledge of computers.
5. Capable of updating information concerning students, courses and course requirements--must facilitate easy addition, deletion, change, or simply, examination of any record of information.
6. Well documented for both users and future programming changes.
7. Capable of processing a single student advisement schedule and printing the results on the remote terminal or of processing all students in the data file and batching the printout.

Advice from supervisory committee chairman, Dr. Christian S. Bauer.
This information system will allow professors to advise their students on a management-by-exception basis. And although this system will require a part-time clerk, especially towards the end of every quarter, to update and maintain data files and to run the advisement program, the clerk's time is far less expensive than the wasted five to ten hours of every professor in the College of Engineering. In addition to distributing a copy to each student's advisor, a copy of the one page trial-advisement/progress report may be made available to the student for planning purposes concerning the next quarter's preregistration trial-advisement interview.
CHAPTER II

MANAGEMENT INFORMATION SYSTEMS RELATED TO STUDENT ADVISEMENT IN EDUCATIONAL INSTITUTIONS--A LITERATURE SURVEY

The process of advising students on the basis of courses completed versus courses required is an element of the functional type of decision known as operational control [1]. Operational control is defined as "the process of assuring that specific tasks are carried out effectively and efficiently" [2]. Ideally, operational control decision-making is supported by the same data bank which supports other functional types of decision-making in the university including administrative control, management control, and strategic planning [1]. In other words, an information system such as the one developed in this thesis, should have access to the university files which contain student information and course information, i.e., the registrar's files. Understandably, for security purposes, the programs developed in this thesis were forced to function without the aid of the registrar's files. In fact, the inaccessibility of the registrar's files was the major factor in preventing an advisement information system to be developed for the university as a whole.

"Student Graduation Progress Report", by Daryl Monroe [3], is an example of an information system, developed for the purpose of obtaining graduation checks, which accesses the university's records. The content of student records and of other related files are closely
related to the contents of the databases developed for the information system discussed in this thesis. Certain information, mainly concerning grades and grade averages, are not included in the system developed herein partially because it would be a duplication of information already available on student transcripts, but mainly because it was not deemed as necessary for purposes of student advisement since university administration notifies students if their grade averages decrease to an unacceptable level.

In Monroe's article, the detailed information available for each student analyzed was described as:

1. Student name and number.
2. Faculty advisor, major degree concentration, degree plan.
3. Course description, course number, grade, credit hours, and points for all required courses enrolled in to date.
4. A course description, course number, grade, and credit hours for all required courses necessary for this degree plan, but not enrolled in to date.
5. Summary information for the required courses including total hours attempted; total grade points earned, and a grade point average in the required course area.
6. Summary information in the areas referred to as general requirements. For example: a diploma student is required to take 9 hours of Social Science and/or Literature courses. Specific courses are not required, but the total hour requirements must be met.

Total hours attempted, total grade points earned, and grade point average in this specific area are provided.

Also, for each of the areas of general requirements, a total hour requirement for graduation is provided. If the student passes and does not duplicate any courses in this general required area, the total required will be at the minimum for this degree requirement. Each time the student either fails
a course or duplicates a course in this area, the hour requirement for graduation is incremented to an adjusted total.

7. A summary of all elective courses taken is provided including total hours attempted, total quality points, and a grade point average in the electives only.

8. A summary of all courses taken which count toward graduation not including non-credit courses such as laboratories, pre-college courses, or etc. This summary indicates total hours attempted, total quality points, and grade average.

9. The status of orientation is normally indicated, either with an "S" for satisfactory, a "U" for unsatisfactory, or as "Not taken for credit".

10. The total hours required for graduation of a specific student are indicated. This differs for each degree plan and is again indicated at the minimum (except when courses have been either failed or duplicated). The total is again incremented in the case of failure and/or duplication.

11. A total for the number of hours of failure is provided.

12. A total for the number of hours of duplication of courses is provided.

This student advisement information system was written for Mesa College in the SPS language for the IBM 1620 computer. The system is reportedly effective in the area of student advisement because of its accuracy, and because it saves large amounts of staff time.

The Mesa College system-description does not reveal what method is used to analyze the prerequisite and corequisite structure of courses. Since no mention is made of this consideration, it is assumed to be done by the advisors and students themselves. As was previously mentioned, the system developed herein, automates the process of determining student eligibility for a particular course.

A previous research effort by Montgomery [4], which closely parallels the system developed herein, and is, in fact, the
predecessor of the aforementioned system, attempts to satisfy the requirements of the system mentioned in Chapter I using a single FORTRAN program, run in batch mode, with all permanent records on computer-cards.

The main drawback to Montgomery's system is that it is only operable in batch mode. Updating is extremely vulnerable to misplacement of data cards and to mistakes since the user must be able to follow formatting instructions which may be confusing to the uninitiated.

Also, the prerequisite and corequisite structures of courses were handled through a "sequenced list of option courses" and a "sequenced table of core courses" [4]. This method's inflexibility resulted in an inaccurate model of the involved decision process. 1

Another drawback of Montgomery's system is that no method exists for indicating during which quarter a course is actually offered. Montgomery's system assumes that all courses are offered every quarter.

The advisement information system developed herein corrects the deficiencies in Montgomery's system by using a more accurate method of describing the prerequisite and corequisite structure for courses; by indicating the quarters in which each course is offered; by providing additional output for tracking the total number of hours

1This inaccuracy is evidenced in a sample output of Montgomery's program, specifically, Figure 18, page 69, where the courses ENGR 331 and ENGR 342 were scheduled for a student who had not satisfied the requirement of completing MATH 324. MATH 324 is a corequisite for ENGR 331 and a prerequisite for ENGR 342.
completed and still required in the Environmental Studies area; and by packaging the whole system including the data, in a user-oriented, interactive mode.

Kaimann, in *Structural Information Files* [5], describes a proposed structure for a database which contains information about students, teachers, courses, and schools (this effort was mainly aimed at secondary schools). The structures presented in connection with course information, however, did not consider any prerequisite information. Therefore, the structures for course information were incomplete for use in a student advisement program.

Kaimann's files utilized linkages through, for instance, a student's record containing the 'home address' in the disk file memory of a completed course. By "following" this pointer to the specified home address, information about the course could be obtained.¹ A student record contained the following information [5]:

1. The check character (P) identifying the record type.²
2. The unique number assigned to each student.
3. Demographic and other information.
4. Curriculum information for courses 1, 2, ..., n.
   a. The course home address.
   b. The grades that the student received and the date that the course was taken.
   c. The front tie relating to the succeeding student enrolled in the course.
5. Data overflow address.
6. Category linkage.
7. Synonym linkage.

¹This technique is referred to as 'randomized storage and retrieval' [6].

²As a pupil record type.
The "home address" of the course would have been initially calculated from a unique number assigned to the course. Likewise, records for students, teachers, and schools have home addresses initially computed from a unique number assigned to each student, teacher, and school respectively.

The information system developed herein uses a special feature of the programming language PL/I to "key" the different types of records, and uses certain programming techniques to "overlay" information of one database onto the information of another database. The result of these two techniques is the same as using home address and pointers. These techniques are discussed more fully in later chapters, especially Chapters IV and V.

C. J. Date's An Introduction to Database Systems [7] presents an example concerning an information system for the internal education system of a large industrial company. The proposed system considers the prerequisite and corequisite structure of the courses offered. Date's example uses IMS\(^1\), an IBM program product, and DL/1\(^2\), an IMS data sublanguage, to manipulate the information in the database described.

The example assumes that the company has an education department which is in charge of running a number of training courses for employees. Each course is offered at a number of locations within the company. Details of the information contained in the database are:

\(^1\)IMS stands for Information Management System/360
\(^2\)Data Language One.
1. For each course: course number (unique), course title, course description, details of prerequisite courses (if any), and details of all offerings (past and planned).

2. For each prerequisite course for a given course: course number and title.

3. For each offering of a given course: date, location, format (e.g., duration, full-time or half-time), details of all teachers, and details of all students.

4. For each teacher of a given offering: employee number and name.

5. For each student of a given offering: employee number.

Thus, the situation presented in Date's example is quite similar to the situation facing the system developed herein. Unfortunately, "in its basic form an IMS system provides facilities for the running of batch applications only" [7]. Therefore, since an interactive program must be supplied, IMS and DL/I were not used for the advice-ment program.\(^1\) However, the built-in functions of PL/I already mentioned provided similar convenience.

\(^1\)Another IBM program product, IQF-Interactive Query Facility [7] permits on-line retrieval only from an IMS database. However, as previously stated, an IMS database may be used in batch applications only.
CHAPTER III

PROGRAMMING LANGUAGE USED--PL/I

Before writing the programs for the advisement information system, consideration was given to which programming language should be used. FORTRAN was the only language personally known, and would obviously be awkward and difficult to work with for the purposes of data file handling and string handling. COBOL, although more capable of the task than FORTRAN, has a nasty reputation for being bulky. The next language investigated was PL/I (Programming Language/I). Because of the following reasons, PL/I was chosen as the language to support the advisement information system programs.

PL/I incorporates the best features of FORTRAN, COBOL and ALGOL (another of the higher level languages) as well as some new features such as "string manipulation, data structures, dynamic storage allocation, and extensive editing capabilities" [8]. PL/I provides the capability of effectively programming both scientific and commercial problems. "It is particularly useful for the increasing number of applications having both commercial and scientific characteristics, for example, information retrieval or command and control applications" [8]. An additional advantage of PL/I is that it provides the programmer with "structured programming" capabilities [6].

Structured programming, or top-down programming, promotes little reprogramming and reduces debugging efforts considerably.
Taken by itself, top-down programming is somewhat of a departure from the modular approach of assigning separate functions to be programmed by individual programmers, and then assembling them with a main "driver" routine. Instead, the top-down programming approach calls for blocking out the entire procedure first, identifying major blocks and branches in the process. Then, a skeleton program is developed which forces the logic of the program to proceed through all of the identified blocks from the "top-down". Then, programming actually begins, proceeding from the top blocks down. Llewellyn [6] states that the program sequence utilizing the top-down approach is:

1. Program initiation (array dimensioning, data input, other initializing tasks).
2. Calculations of the statements in the first block.
3. Separation of the task into two or more parts by a branching test.
4. Calculation of all blocks and loops in one branch. There may be additional branch tests in the first side of the first test.
5. Calculation of all blocks and loops in one side of the second test, if another one was required in step 4. Otherwise, the second side of the first test is explored. If the latter action is taken, there may be another branch test, this being, of course, on the second side of the first branch.
6. The calculations will tend to continue in this fashion. The main feature is that all calculations are performed in one branch before doing anything in the other. However, if another branch test is necessary while tracing out a branch, both sides of the later fork will be traced before working along the second side of the earlier branch.

Final results of structured-programming are the elimination of jumps in the logic, i.e., no GO TO statements, and "there should be only one entrance and one exit from each of the basic structures..." [6]. Therefore, the logic flow of the program is easier to follow and debug.
The types of statement blocks provided by PL/I which promote structured programming are:

1. BEGIN;
   XXXXXX;
   XXXXXX;
   END;
   (where 'XXXXXX' represents a sequence of PL/I statements).

2. DO;
   XXXXXX;
   XXXXXX;
   END;

3. IF XXXXXX THEN XXXXXX;
   ELSE XXXXXX;

4. IF XXXXXX THEN DO;
   XXXXXX;
   XXXXXX;
   END;
   ELSE BEGIN;
   XXXXXX;
   XXXXXX;
   END;
   (This is really a combination of 1, 2, and 3.)

5. DO I = J TO K BY L;
   XXXXXX;
   XXXXXX;
   END;

6. DO WHILE (condition);
   XXXXXX;
   XXXXXX;
   END;

Neither modularity nor structured programming were strictly enforced in the advisement information system programs. However, top-down programming was used as much as possible to minimize the number of GO TO statements and resulting logic jumps. Modular procedures were still, however, utilized, and in certain instances, even promoted a more understandable logic flow. In other words, the best of both
techniques were considered in writing the advisement information system programs.

Among the drawbacks of PL/I are:

1. The language requires a large number of data type specifications [6].
2. PL/I compilers are slow compared to FORTRAN and COBOL compilers [6].
3. There are buffer problems with GET LIST and PUT LIST statements when PL/I programs are run on an interactive mode.

The "drawback" of a large number of data type definitions parallels the bulky COBOL operation of "housekeeping". The fact that PL/I compilers are relatively slow only effects the development process of an interactive program since a program in "final form" is stored in a load module which is not recompiled at each run time. The final drawback of buffer problems arising in connection with interactive PL/I programs may possibly be solved with placing PUT SKIP statements after every PUT LIST. However, in the advisement information system programs, this problem was solved using two assembler routines:

1. TGET, which was substituted for GET LIST input statements.
2. TERMPUT, which was substituted for PUT LIST output statements.

Listings for these two assembler routines are found in Appendix M—Listings of Time-Sharing And Job Control Programs. These routines are simply called by a PL/I program, for example:

CALL TERMPUT('EXAMPLE');

This statement acts exactly as a batch PUT LIST statement. Without
the assembler routines, a PUT LIST statement may or may not print the contents contained in its parenthesis, depending on whether or not the output buffers are full and ready to be "dumped" onto the remote terminal printer.

TGET works in a similar manner. However, TGET has a restriction of only being able to "get" a character string. It is not capable of:

```
CALL TGET(NUM);
```

where NUM is a decimal number. Neither is TGET capable of "getting" an array or an arrayed structure, such as:

```
CALL TGET(STUDENT (I));
```

To solve this inadequacy of TGET the following technique was used:

```pl
DECLARE TEMP CHARACTER (20) VARYING;

CALL TGET(TMP);
STUDENT (I) = TEMP;

CALL TGET (TEMP);
NUM = DECIMAL(TEMP);
```

The use of the DECIMAL built-in PL/I function converts a numeric character string to its decimal equivalent.

If TERMPUT were to be used for either printing the contents of an array or an arrayed structure or printing a number, the reverse of this technique should be used, i.e.,
DECLARE TEMP CHARACTER(20) VARYING;

TEMP = STUDENT(I);
CALL TERMPUT(TEMP);

TEMP = CHAR(NUM, length);
CALL TERMPUT(TEMP);

Here, CHAR converts NUM to a character string. Length is the declared length of NUM. In other words, if NUM is declared as FIXED DECIMAL (5,1) then length = 5.

The TGET and TERMPUT statements are used for "interactive conversation", while PUT EDIT statements are used for final output reports. As will be further discussed under the user instructions for the various system programs (Appendix I, J, K, and L), the final output reports may be printed as a batch listing or at the terminal printer.

To output an entire structure in response to a 'print' update command (see Chapter VII--"Updating the Courses Database") the following procedure was used:

DECLARE SYSTEM FILE STREAM.
DECLARE 1 STUDENT UNALIGNED,
     2 SOCIAL_SECURITY CHARACTER(9),


OPEN FILE SYSTEM PRINT;

READ FILE (STDBASE) INTO STUDENT;

PUT FILE (SYSTEM) EDIT (STUDENT) (format);
CHAPTER IV

SYSTEMS ANALYSIS

Databases Required

Throughout the remaining discussion about the advisement information system programs, the word "database" should be understood as meaning an ordered set of logically similar records, each record having exactly the same structure. There are three databases which are accessed by the advisement information system programs. The student database, for example contains a number of records, each record pertaining to a particular student, but with all records possessing exactly the same structure.

A structure, in PL/I, is a hierarchical collection of data items. Data items that do not necessarily have identical characteristics, but which are bound by some logical relationship to one another, can be grouped together into a structure. Like an array, the entire structure is named, and that name can be used to refer to the entire structure. However, unlike an array, each element or array of elements within the structure also has a name [9]. The following databases were deemed necessary to fulfill the requirements of this system.

Student Database--Structure and Contents

The name of this database is STDBASE. The PL/I structure for the record of each student in the student database is:
DECLARE 1 STUDENT UNALIGNED,
2 SOCIAL SECURITY CHARACTER(9),
2 NAME CHARACTER(25),
2 MAJOR CHARACTER(20),
2 ADVISOR CHARACTER(26),
2 COURSES,
3 ENGR_CORE(33) CHARACTER(1),
3 ENVIR_STUDIES_SPE(8) CHARACTER(1),
3 ADV ENVIR_STUDIES(3) CHARACTER(1),
3 ENGR_OPTIONS(10) CHARACTER(1),
3 ENVIR_STUDIES_GEN(3) FIXED DECIMAL(3,1),
3 REST ELECTIVES FIXED DECIMAL(3,1),
2 TOTAL_HOURS FIXED DECIMAL(4,1);

This PL/I code reveals that the name of the entire structure is STUDENT. The code also reveals that the first data element of STUDENT is SOCIAL_SECURITY. The SOCIAL_SECURITY data element is defined as a character string consisting of exactly nine characters. In other words, a student's social security number is recorded in the student database as a nine-member character string, i.e., the numerals without the dashes.

The student's name is recorded within the twenty-five-character slot reserved for the data element NAME. NAME contains the student's last name, first name, and middle initial.

The MAJOR data element contains the accepted acronym for the student's major in the first four characters of the twenty-character string, i.e. EECS for Electrical Engineering, IEMS for Industrial Engineering, etc. The MAJOR data element contains the years of the catalog for which this major's degree plan was initiated in the eleventh through the fifteenth characters of the twenty-character string. For instance, where b signifies a blank:

IEMSbbbbbb76-77bbbb
IEMSbbbbbb75-76bbbb
Both of these majors have individual names. Even though they both are identified by the same acronyms, they are governed by different F.T.U. catalogs.

The ADVISOR data element has twenty-six characters reserved for the name of the student's advisor.

COURSES is a sub-structure of STUDENT. That is, even though it is not a data element, it contains several data elements. The first data element of STUDENT.COURSES being the array of data elements-ENGR_CORE. ENGR_CORE is dimensioned as having thirty-three character strings, each of length 1. Each one of these single-character strings represents the status of an engineering core course for the student. Possible contents of each single character slot with their corresponding interpretations are:

'C' - course completed
'T' - course was taken but credit was not received at this university
'E' - the student has completed all prerequisites for this course, and is eligible to take the course
'b', where b represents a blank
- the student is not eligible to take the course and has not taken it previously

The courses to which each element of ENGR_CORE correspond are defined in another structure called COURSES_REQUIRED. This structure will be further discussed under the section in this chapter titled "Course Requirements Database". Let it suffice at this point to mention that the appropriate structure will be retrieved for each student
according to the contents of the data element MAJOR of the STUDENT structure. Each element of ENGR_CORE will then correspond to a course name indicated in the COURSES_REQUIRED structure.

ENVIR_STUDIES_SPE is the second array of data elements of STUDENT.COURSES. It is dimensioned as having eight one-character strings which represent the status of certain environmental studies courses. The codes, "C", "T", "E", and "b", described for ENGR_CORE, also apply to ENVIR_STUDIES_SPE data elements. Also, there is a correspondence between these data elements and course names specified in the COURSES_REQUIRED structure of the appropriate record in the course requirements database.

The next two data element arrays: ADV_ENVIR_STUDIES, which corresponds to advanced environmental studies course areas; and ENGR_OPTIONS, which corresponds to engineering option courses, are dimensioned to 3 and 10 respectively. The elements of both are single-character strings which may contain the previously discussed codes 'T', 'C', 'E', and 'b' and which correspond to certain course names specified in the COURSES_REQUIRED structure of the appropriate record in the course requirements database.

The data element array ENVIR_STUDIES_GEN contains the actual credit hours earned by the student in each of three general areas in the environmental studies area. Again, each of these three areas correspond to the area names specified in the COURSES_REQUIRED structure of the record which corresponds to the student's major in the course requirements database.
The final two data elements of STUDENT.COURSES are: REST_ELECTIVES, which contains the total number of credit hours earned in restricted electives; and ELECTIVES, which contains the total number of credit hours earned in electives.

The final data element of STUDENT is TOTAL_HOURS, which contains the total number of credit hours earned to date toward graduation.

For purposes of testing the information system programs, 10 student records are contained in the student database.

Courses Database--Structure and Contents

The name of this database is CODBASE. The PL/I structure for the record of each course in the courses database is:

```
DECLARE 1 COURSE UNALIGNED,
  2 NAME CHARACTER(8),
  2 COREQUISITES(3) CHARACTER(8),
  2 PREREQUISITES(5) CHARACTER(8),
  2 SEASONS OFFERED(4) FIXED DECIMAL(1),
  2 CREDIT HOURS FIXED DECIMAL(1),
  2 DESCRIPTION CHARACTER(20);
```

The first data element of the course structure is NAME, which is declared as a character string of length 8. Due to programming considerations, the course name must begin with the three or four letter acronym representing the type of course, i.e. IEMS, EECS, ENG, etc., and must end with the course number. Thus, the course name must span the entire eight spaces allocated to it. For example:

---

1Through the technique of qualifying names, there will not be any confusion between the COURSE structure and the data element of the STUDENT structure named COURSE. The latter will be qualified when referenced in the program as STUDENT.COURSE.
The next two data element arrays, i.e., COREQUISITES and PREQUISITES, dimensioned three and five respectively, also contain eight-character names of courses which must be specified in a manner, as described above in the NAME description, such that the course spans the entire eight characters.

The data element array SEASONS_OFFERED contains four one-digit numbers. These four numbers represent whether or not a course is offered during the fall, winter, spring, and summer quarters respectively. A "0" represents a not-offered status, while a "1" represents an offered status.

The CREDIT_HOURS data element contains a one-digit number representing the number of credit hours the course is worth.

The DESCRIPTION data element contains a twenty-character description of the course.

The COURSES database contains information on all courses required to obtain a B.S.E. in any of the options offered by the College of Engineering.

Course Requirements Database--Structure and Contents

The name of this database is COREDBS. The PL/I structure for the record of each major's course requirements in the course requirements database is:

```
DECLARE 1 COURSES REQUIRED UNALIGNED,
         2 MAJOR CHARACTER(20),
```
The first data element of the COURSES_REQUIRED structure is MAJOR. This twenty character name of COURSES_REQUIRED.MAJOR must be specified in exactly the same manner as was STUDENT.MAJOR. In other words, the accepted acronym for the major is in the first four of twenty characters, while the governing catalog years, i.e., 75-76--including the dash, fill the eleventh through fifteenth characters.

The remaining data elements and data element arrays directly parallel the data elements and data element arrays under the STUDENT structure with corresponding names. However, this COURSES_REQUIRED structure contains the required name and number of hours for each course or course area. For example, consider the following correspondence between the STUDENT and the COURSES_REQUIRED structures:

the value of "1" or "0" in

STUDENT.COURSES.ENGR_CORE(1)

indicates the status of the course named by

COURSES_REQUIRED.ENGR_CORE(1).NAME for the indicated student.
STUDENT.COURSES.ELECTIVES contains the number of hours of electives the student has completed while, COURSES_REQUIRED.ELECTIVES contains the number of hours in electives which a student must complete in order to graduate.

The ENVIR_STUDIES_GEN data element array specifies a range of credit hours, i.e., in order to graduate, a student must complete a number of credit hours in each area which falls within the specified range.

The course requirements database currently contains records for every major offered under the 1976-1977 F.T.U. catalog.

Programs Required

Analysis revealed that turn-around times for the separate actions to be performed by the advisement information system would be expedited by creating relatively small single-purpose programs in contrast to one large program which would be capable of all required actions. In response to this analysis, a total of seven programs were written.

The first three programs were written to build the three databases according to the structures previously described. All three databases were created to permit either sequential or sequential indexed access. None of these three programs were stored in time-sharing files, and therefore, are only operable on batch mode.

The fourth program was written for the purpose of printing a one page advisement report for each student as well as printing a one page report projecting class enrollments for each course in the course
database for the four quarters of the school year. The program was written to converse interactively with the user. It is capable of either processing only a few students and printing their respective advisement reports on the remote terminal printer or processing all of the student records contained in the student database and printing their respective advisement reports on the batch printer.

The final three programs were each written to update one of the databases. All three of these programs were written to converse interactively with the user. Each of these three programs allows the user to add a record, delete a record, change any element of a record, or simply examine a record in the appropriate database. Upon completion of the update activity, each of these three programs lists the contents of the corresponding database.

Since comments have been extensively incorporated in each of the seven programs to aid in program documentation, highly detailed flowcharts are not provided in Chapters V through VII. Instead, verbal explanations accompanying flowcharts of a general nature are provided to permit a comprehensive overview of each of the seven programs. A detailed understanding of a particular program may be obtained by examining a listing of the programs included in one of the Appendices of this thesis.
CHAPTER V

BUILDING THE DATABASES

Building the Courses Database

Basically, the program used to build the courses database, reads data for each course into the COURSE structure from cards, and then prints each record before it writes the record for each course on a disk file. The program’s name is BUILD_C, and the name assigned to the disk file is CODBASE.

After writing all of the records, originally recorded on cards, on the disk file, the program then reads all of the records on the disk file and lists them. This final echo print of the data on the disk file assures the user that the information was recorded accurately. Figure 1 presents a flowchart of the BUILD_C program. A complete program listing, along with necessary job control cards and an output listing are presented in Appendix B.

Building the Course Requirements Database

The name of the program which builds the course requirements database (COREDBS) is BUILD_C_R. The logic flow of BUILD_C_R progresses through an initialization block and then into a read block. This first read block reads data from cards which are common to the course requirements of all the majors in the College of Engineering. This common data includes the names and credit hours of all
FIGURE 1: Flowchart of BUILD_C

--Building the Courses Database--CODBASE
engineering core courses, environmental studies courses and course areas, as well as advanced environmental studies course areas.

Program logic then flows through a do loop which begins with a second read block. This second read block reads data peculiar to each major. Then, all of the information is assigned to the appropriate COURSES_REQUIRED data elements. The structure is first printed and is then written on the COREDBS database. Because there are five different majors, the do loop is executed five times.

After all of the different majors have been written on the COREDBS database, the information is immediately read from the database and printed to assure that the different degree programs were accurately recorded. Necessary job control cards, the BUILD_C_R program listing, and the program output are displayed in Appendix C. Figure 2 presents a flowchart of the BUILD_C_R program.

Building the Student Database

The student database (STDBASE) is built by a program named BUILD_S. This program proceeds in a manner almost identical to that of the BUILD_C program. First, data for each student is read from cards into the STUDENT structure. The STUDENT structure is printed and then written on STDBASE. After all of the student records have been written on STDBASE, the records are read back and printed to assure that the information was accurately recorded.

Necessary job control cards, the BUILD_S program listing, and the program output are displayed in Appendix A. Figure 3 presents a flowchart of this program.
FIGURE 2: Flowchart of BUILD_C_R--
Building the Courses Database--COREDBS
START

INITIALIZATION ON ENDFILE (SYSIN) GO TO READ_F_1;

READ DATA INTO STUDENT STRUCTURE

PRINT STUDENT STRUCTURE

WRITE STUDENT STRUCTURE

READ STUDENT STRUCTURE ON ENDFILE (STDDBASE) GO TO EXIT;

PRINT STUDENT STRUCTURE

EXIT:

STOP

FIGURE 3: Flowchart of BUILD S--Building the student database--STDDBASE
Assumptions

During the development of the advisement information system, several assumptions were made. The first assumption made concerned transfer courses. Administrative personnel of the College of Engineering agreed that equating transfer credits from other universities would be accomplished through petitions and need not be the concern of the advisement information system programs. Therefore, the assumption was made that all courses recorded in the student record would be in terms of F.T.U.-offered courses.

Another assumption made, which is related to the assumption concerning equated courses, involved F.T.U. students who switched majors. Only those courses directly applicable to the student's new major will be transferred to the advisement record. Any substitutions will have to be resolved by petition.

An assumption was made in relation to the enrollment projection portion of the program that the student would enroll and pass every course which the advisement report acknowledged as being offered and for which the student was eligible.

Logic Flow

An initialization block declares the data types of all the
variables, structures, and files used in the program. Comments are written extensively throughout the program to clarify the meaning and usage of important variables, structures, and files used in the program as well as to further describe the program logic flow. The actual program listing appears in Appendix D.

As was previously mentioned, this program was written to interactively converse with the user. The next block of the program concerns this conversational programming. The first question the program asks the user is whether or not the user requires instructions and requests the user to answer with either "yes" or "no". If the user answers "yes", then a set of operating instructions are printed. If the user answers "no", then only very short prompting questions are used to converse with the user.

The next question asked by the program concerns whether the user wants to process only a few student records and have their advisement reports printed at the terminal, or process all of the student records in STDBASE and have their advisement reports printed at the batch printer. The user is requested to respond with either "all" or "some". If the user responds with "some", the program asks the user to supply the number of students to be processed. This number must be less than or equal to fifteen. The user is then asked to supply the social security numbers of each student to be processed as nine digit numbers without dashes. The program searches the student database (STDBASE) for the records which correspond to the social security numbers supplied by the user. These records are stored into a dimensioned structure named STUDENT_I.
The next program step, regardless of the answer to the "some" or "all" question, is to ask the user to supply either "fall", "winter", "spring", or "summer", to represent the approaching quarter. The program logic flow then proceeds through a section, labelled SEQUENT, which reads each course record from the courses database (CODBASE). Each course is tested to detect whether or not it will be offered during the approaching quarter. If the course will be offered, then the entire course record is stored in one of the records of a dimensioned structure named COURSE_0. At the same time, by calling the procedure named C_P_ASSIGN, another dimensioned structure is built to be used for prediction purposes. Depending on which quarter was specified, the structure name is either COURSE_PREDICT_FALL, COURSE_PREDICT_WINTER, COURSE_PREDICT_SPRING, or COURSE_PREDICT_SUMMER.

Next, if "some" student records were to be processed, then one element of the STUDENT_I structure is assigned to the STUDENT structure. Otherwise, if "all" student records were to be processed, a student record is read from the student database into the STUDENT structure. This assignment to the STUDENT structure occurs immediately after the statement labelled WHICH_STUDENT.

The next program block, beginning at LOOP1, records the index in the student record of all courses for which the student is eligible. The following program block, beginning at LOOP_1, records the index in the student record of all courses which the student has completed. By overlaying the COURSES_REQUIRED structure which corresponds to the student's major, the next block of statements, beginning at LOOP5, records the names of the courses for which the student is eligible and
is required to take. Beginning at LOOP_5, the names of all of the courses which the student has completed are recorded by overlaying the COURSES_REQUIRED structure which corresponds to the student's major.

At LOOP9, credit hours still required in the general areas of environmental studies, restricted electives, electives, and total hours necessary for graduation are computed.

At COMPARE, the courses for which the student is eligible are compared to the courses which are being offered during the approaching quarter. Whenever a match is made, the course name is stored in an array for printing purposes.

After the comparisons are completed, the student's advisement report is printed. If some of the student records were to be processed, the program logic flow returns to the statement labelled WHICH_STUDENT.

However, if all of the student records were to be processed, then the STUDENT structure is assigned to an element of another dimensioned structure named STUDENT_PREDICT. Then, a portion of the enrollment prediction segment is enacted. In this segment of the program, the procedure named PREDICT_ARRAY, the enrollment of the courses for which the student is eligible and which are to be offered during the approaching quarter is incremented by one. Then, the procedure named STUD_PRED is executed. STUD_PRED updates the contents of the appropriate element of the STUDENT_PREDICT structure by changing the eligible status to a completed status i.e., from "E" to "C", of any course for which the student is eligible and which is
being offered during the approaching quarter. Also, after examining the prerequisite and corequisite structure of all of the courses in the course database (CODBASE), the eligibility status for courses which have not been completed in the student's record is updated if necessary. Control then passes to the WHICH_STUDENT statement.

When all of the student records in STDBASE have been processed, the procedure named PREDICT is executed. First, the "next" quarter is designated for prediction purposes, i.e., if the quarter was originally specified as fall, then the "next" quarter is winter. Then the procedure named UPDATE_STATUS is called.

UPDATE_STATUS considers all of the courses the student has completed as well as all the courses which the student still requires, and for prediction purposes, "updates" the eligibility status of courses for which the student has just completed the prerequisites. This prediction of eligibility information is stored in the STUDENT_PREDICT structure. It should be noted that this "updating" process has a temporary nature and should not be confused with the permanent updating of the database information.

When "updating" is completed, control returns to PREDICT. Then PREDICT calls the procedure named PREDICT_ADVISE. PREDICT_ADVISE records the index of the courses for which the student record in the STUDENT_PREDICT structure is eligible. Then the names of the eligible courses are recorded.

Program control returns to the PREDICT procedure, which immediately calls the PREDICT_ARRAY procedure to increment by one the
enrollments of the courses whose names were recorded by PREDICT_ADVISE for the quarter currently under inspection. Control again returns to the PREDICT procedure where the STU_PRED procedure is called to update the student's record. Then, the next record in the STUDENT_PREDICT structure is processed.

When all of the student records in the STUDENT_PREDICT structure have been processed and updated, then the whole procedure is repeated for the next quarter. When there is enrollment information for all four quarters of the year, then, finally, the procedure named PRINT_PREDICT is executed.

PRINT_PREDICT prints a one page report of the projected enrollments of all the courses in CODBASE for the four quarters of the school year.

Figure 4 presents a flowchart diagram of SADVISE. Appendix I presents the user instructions for SADVISE, along with a sample output.
FIGURE 4: Flowchart of SADVISE--The Student Advisement Program
FIGURE 4: Flowchart of SADVISE--The Student Advisement Program (continued)
WHICH_STUDENT:

INDEX = INDEX + 1

OPTION_1

'SOME'

'STUDENT = STUDENT_I(INDEX);

STUDENT_DATABASE

READ
STUDENT
(SEQUENTIAL
ACCESS)

RECORD THE INDEX
OF ALL
ELIGIBLE COURSES

LOOP1:

RECORD THE INDEX
OF ALL
COMPLETED COURSES

READ
COURSES
REQUIRED
KEY
STUDENT_MAJOR

COREDBS
COURSE
REQUESTS
DATABASE

FIGURE 4: Flowchart of SADVISE--The
Student Advisement Program (continued)
FIGURE 4: Flowchart of SADVISE--The Student Advisement Program (continued)
CALL PRINT

PRINT: PROCEDURE:

TRIAL ADVISEMENT SCHEDULE

OPTION 1 = 'SOME' & ALL STUDENTS PROC

YES

GO TO EXIT

OPTION 1 = 'ALL'

STUDENT_PREDICT (INDX) = STUDENT, BY NAME

CALL PREDICT ARRAY: INCREMENTS ENROLLMENT OF ELIGIBLE OFFERED COURSES

GO TO WHICH_STUDENT ON ENDFILE STDBASE GO TO C_P

C_P:

CALL PREDICT

STOP

FIGURE 4: Flowchart of SADVISE--The Student Advisement Program (continued)
PREDICT_ARRAY: PROCEDURE;

START

INCREMENT BY 1
THE ENROLLMENTS
OF ALL ELIGIBLE
COURSES
TO BE OFFERED

CALL STUD_PRED

STUD_PRED: PROCEDURE

START

FOR ANY ELIGIBLE
OFFERED COURSE,
CHANGE THE STATUS
IN THE STUDENT
RECORD TO 'C'

STOP

STOP

FIGURE 4: Flowchart of Sadvise-- The Student Advisement Program (continued)
PREDICT: PROCEDURE;

START

START_P:

"ADVANCE"
THE QUARTER

DO INDXST = 1
TO INDX

READ COURSES
REQUIRED KEY
STUDENT
...PREDICT
(INDXST).MAJOR

CALL UPDATE_STATUS
(STUDENT_PREDICT
(INDXST));

CALL PREDICT_ADVISE
(STUDENT_PREDICT
(INDXST));

CALL PREDICT_ARRAY;

FIGURE 4: Flowchart of A SADVISE--The Student Advisement Program (continued)
FIGURE 4: Flowchart of SAUVISE--The Student Advisement Program (continued)
UPDATE_STATUS:PROCEDURE;

START

TEST1

EXAMINE THE STATUS
OF
EACH ENGINEERING
CORE COURSE--
DO I = 1 TO 33

NO

ST 'ATUS
YES

NAME

? = ''

?

YES

NAME

NO

PREREQUISITES

YES

STUDENT

COMPLETED ALL

PREREQUISITES

?

END TEST1

END OF DO

NO

CHANGE STATUS FROM

blank to

ELIGIBLE

END TEST1

NO

7

FIGURE 4: Flowchart of SADVISE--The Student Advisement Program
(continued)
FIGURE 4: Flowchart of SADVISE--The Student Advisement Program
(continued)
FIGURE 4: Flowchart of SADVISE--The Student Advisement Program
(continued)
CHAPTER VII

UPDATING THE DATABASES

CUFPIT--Courses Database Update Program

This program provides the user with the capability of adding a course, deleting a course, examining a course, or changing any part of a course record. In addition to declaring the data types of the variables used in this program, the initialization defines two structures: COURSE, which is the structure read from the course database; and COURSE_P, which is used for a temporary holding area for course records being added to CODBASE.

The first program instruction requests the user to supply one of the following update commands:

1. instruct - if instructions are needed, this must be the first command
2. delete - delete an existing course record from CODBASE
3. print - print a particular course record
4. change - change one or more data elements of an existing course record
5. add - add an entirely new course record to CODBASE
6. stop - the user is finished updating CODBASE

If the user responds with the instruct command, operating instructions are printed throughout all program steps. The print command generates another question asking for the name of the course
to be printed. After the user responds, the information contained in the specified course record is printed. Then another computer request for an update command is issued. In fact, after every command is executed, the program asks for another command until it receives a stop command.

Similarly, a delete command generates a request for the name of the course to be deleted. Also, when a change command is issued, the program requests the name of the course to be changed.

After a change command is issued and the appropriate course name supplied, the program reads the corresponding course record into the COURSE structure in the procedure named CHANGE. The user must then supply one of the following codes to signify the area of the COURSE structure to be changed:

- 0 - no change
- 1 - corequisites
- 2 - prerequisites
- 3 - seasons-offered
- 4 - credit hours
- 5 - description

Then, depending upon which section of the structure is to be changed, appropriate program-issued instructions are printed. These instructions will be brief unless the user specified "instruct" initially.

When an add command is issued, the program calls the procedure named ADD. First, a course is read from CODBASE into the COURSE structure. Next, the user is asked to supply the number of courses which
will be added to CODBASE. Then, in the procedure named READ_COURSE_P, a series of questions is asked to secure the various elements of information necessary to fill the COURSE_P structure. This series of questions is posed by the procedure named READ_COURSE_P, which is called by ADD. The questions ask for the course name, corequisites, prerequisites, seasons to be offered, the number of credit hours, and a course description.

Then, back in ADD, the new course, or one of the new courses, is inserted in the course database through the following process: COURSE_NAME is compared to COURSE_P_NAME. The structure with the "smallest" name, i.e., the name first in alphabetical order, is then written on a temporary sequential file (CODBSEP). If the COURSE structure contained the smallest name, then a new record must be read from CODBASE into COURSE. If the COURSE_P name was the smallest, then there are two different paths to take. If only one course was to be added, the process of reading from CODBASE into COURSE and then writing COURSE onto CODBSEP, will continue until the end-of-file for CODBASE is encountered. If more than one course was to be added, the entire process of asking for a new course name, corequisites, prerequisites, etc., will be repeated until all additions have been completed. When all of the courses originally existing on CODBASE as well as all of the added courses have been written on the temporary file named CODBSEP, then all of the courses are read from CODBSEP into the COURSE structure and, finally, written onto the course database (CODBASE) keyed by each course's name.
When a stop command is issued, a listing of the courses on
the course database will be printed on either the terminal's printer
or the batch printer, depending on the user's response to a final
question regarding printout routing. The user may answer this final
question with either "batch" or "term".

Figure 5 presents a flowchart of COUPDT. Appendix E provides
a program listing of COUPDT. And Appendix J presents user instruc-
tions and a sample output for COUPDT.

COREUPD--Course Requirements Database
    Update Program

The initialization block of COREUPD defines, in addition to
the variables used in the program, two important structures:
COURSES_REQUIRED, which is the structure for the records of the dif-
ferent majors on COREDBS (the course requirements database); and
COURSES_REQUIRED_TEMP, which is the structure used to hold records
being added to COREDBS.

There are also two files declared. COREDBS is declared as a
keyed, indexed file for possible direct access. RFQUPDT, a temporary
working file, is declared for sequential access.

The next block of program-user conversational programming is
identical to that of the courses update program--COUPDT. The update
commands, "instruct", "delete", "print", "change", "add", and "stop"
are used in this program for the same reasons and with similar con-
sequences as in COUPDT.

If the user issues an instruct command, operating instructions
are printed throughout all program steps. The print command generates
COUPDT: PROCEDURE OPTIONS (MAIN);

START

INITIALIZATION OF VARIABLES, STRUCTURES, AND FILES

GET_IN:

INPUT UPDATE COMMAND

INSTRUCTION

INSTRUCTION 'INSTRUCT'

INSTRUCT = 1:

GENERAL UPDATE PROGRAM USER INSTRUCTIONS

ELSE

CALL ADD

ELSE 1

FIGURE 5: Flowchart of COUPDT--Courses Database Update Program
CALL DELETIOO;
GO TO GET_IN;

CALL CHANGE;
GO TO GET_IN;

DISPLAY INSTRUCTIONS FOR INPUT OF COURSE NAME

COURSE NAME

INPUT COURSE NAME

COURSE RECORD

GO TO GET_IN;

FIGURE 5: Flowchart of COUPDT--Courses Database Update Program
(continued)
FIGURE 5: Flowchart of COUPDT--Courses Database Update Program
(continued)
ADD: PROCEDURE;

START

INPUT THE NUMBER OF COURSES TO BE ADDED

NUM_OF_ADDS

ENTER COURSE NAME

INSTRUCT?

YES

FURTHER INSTRUCTIONS FOR ENTERING COURSE NAME

NO

OPEN FILE (CODBSEP) FOR OUTPUT AS TEMPORARY WORKING FILE.

OPEN FILE (CODBASE) FOR SEQUENTIAL ACCESS INPUT; ON
ENDFILE (CODBASE) GO TO ADD_ON;

3

FIGURE 5: Flowchart of COUPDT--Courses Database Update Program

(continued)
READ COURSE P

READ COURSE P: PROCEDURE
PROMPTS USER TO SUPPLY INFO. FOR THE CURRICULUM OF THE NEW MAJOR
WRITE COURSE OR COURSE P, WHICHEVER IS SMALLEST

CALL READ COURSE P

TEST:
INSERT THE NEW COURSE IN THE LIST OF EXISTING COURSES

END_READ_C:
END OF DO WHEN COURSE P WRITTEN

FIGURE 5: Flowchart of COUPDT--Courses Database Update Program (continued)
DELETE ALL COURSE RECORDS IN CODBASE

RED:
READ COURSE ON ENDFILE
CODBSEP GO TO END_ADD

WRITE COURSE

CODBSEP
TEMP WORKING FILE

CODBASE
COURSES DATABASE

END_ADD:
END ADD;

FIGURE 5: Flowchart of COUPDT--Courses Database Update Program (continued)
DELETION: PROCEDURE

START

NUMBER OF DELETIONS =

DEL

DO I=1 TO DEL

COURSE NAME =

COURSE_NAME

DELETE COURSE

OODEBASE COURSES DATABASE

END OF DO

STOP

FIGURE 5: Flowchart of COUPDT--Courses Database Update Program (continued)
FIGURE 5: Flowchart of COUPDT—Courses Database Update Program (continued)
FIGURE 5: Flowchart of COUPDT--Courses Database Update Program (continued)
a request for the desired major's name. Once supplied with the name of the major, the program reads the corresponding record from the course requirements database (COREDBS) and prints the record. As soon as the print command, or any other update command, has been executed, the program asks the user for its next instruction.

Similar to the print command, the delete command generates a request for the name of the major to be deleted. Once supplied with the name of the major, the program deletes that major from COREDBS. Deletion is executed in the procedure named DELETES.

When a change command is issued, the program requests the name of the major to be changed in the procedure named CHANGE. Once supplied with the name of the major, the CHANGE procedure reads that major record from COREDBS. Then, the user is asked to specify a code which informs the program which section of the record is to be changed. These codes and their definitions are:

0 - no change
1 - one of the engineering core courses
2 - one of the specific courses for environmental studies
3 - one of the advanced environmental studies areas
4 - one of the engineering options courses
5 - one of the general areas of environmental studies
6 - the hours of restricted electives
7 - the hours of electives
8 - total number of hours needed for graduation
Then depending upon which section of the record is to be changed, appropriate questions are asked of the user. These questions ask for the name of the course to be changed in the case of an alteration in an area such as the engineering core courses. The appropriate set of courses is then searched. When the position of the course to be changed is located, the user is asked to supply the name of the new course and its corresponding credit hours.

When the user issues an "add" command, the main procedure calls the procedure named ADD. First, a major is read from COREDBS into the COURSES_REQUIRED structure. Next, ADD asks the user how many courses will be added. Then the READ_MAJOR procedure is executed to ask the user for all of the information needed to fill the COURSES_REQUIRED_TEMP structure. The user is asked to supply the new major's name; engineering options courses--names and credit hours; general areas of environmental studies--names and ranges of credit hours; and the number of credit hours in the areas of restricted electives, electives, and total hours necessary for graduation.

Then, back in ADD, the new major, or one of the new majors, is added to COREDBS. The process of adding a new major to COREDBS is almost identical to the process used in the COUPDT program to add a new course to CODBASE.

To add a new major, the first step is to compare COURSES_REQUIRED.MAJOR and COURSES_REQUIRED_TEMP.MAJOR in order to determine which name belongs first in an alphabetically ordered set. The major name which belongs first in alphabetical order is then written on the temporary file, REQUPT. If COURSES_REQUIRED.MAJOR is determined as
being the major to be first in order, then a new major must be read into the COURSES_REQUIRED structure from COREDBS to replace the one written on REQUPDT. If the program detects an end-of-file for COREDBS, then only new majors will be read and written on REQUPDT.

If, on the other hand, COURSES_REQUIRED_TEMP.MAJOR is determined as being the major to be first in alphabetical order, then the entire process, beginning with supplying the name of a new major, is repeated. If, at this point, all additions have been executed, then any majors remaining on COREDBS or in COURSES_REQUIRED must be written on REQUPDT.

When all of the original majors, as well as all of the added majors have been written on the temporary file, REQUPDT, then all of the majors are read from REQUPDT into COURSES_REQUIRED, and are then written onto the course requirements database (COREDBS) keyed according to each major's name.

When a stop command is issued, a listing of the majors on COREDBS is printed on either the terminal's printer or the batch printer, depending on the user's response to a final question regarding printout routing. The user may answer this final question with either "batch" or "term".

Figure 6 displays a flowchart of COREUPD. Appendix F presents a program listing of COREUPD, while Appendix K provides user instructions for and a sample output of COREUPD.

STUPDT--Student Database Update Program

The update commands which applied to the other two update programs, i.e. COUPDT and COREUPD, also apply to STUPDT. Therefore,
NOTE: SINCE THE PROCESSES OF ADDITION, DELETION, PRINTING, CHANGING, AND STOPPING FOR THIS PROGRAM PARALLEL THOSE PROCESSES USED IN COUPDT, THESE PROCEDURES ARE NOT DETAILED IN THIS FLOWCHART. PLEASE REFER TO FIGURE 5 FOR INFORMATION CONCERNING THE GENERAL FLOW OF THESE PROCESSES.

FIGURE 6: Flowchart of COREUPD--Course Requirements Database Update Program
Coupdt: procedure options(main);

START

initialization of variables, structures, and files

get in:

input update command

instruction

instruction

instruct = 1:

general update program user instruction

call add

FIGURE 6: Flowchart of Coreupd--Course Requirements Database Update Program (continued)
FIGURE 6: Flowchart of COREUPD--Course Requirements Database Update Program
(continued)
FIGURE 6: Flowchart of COREUPD--Course Requirements Database Update Program (continued)
explanations of the update commands "instruct", "print", "delete", and "stop" will not be repeated in this section.

The update commands "add" and "change" produce results similar to the ones in CUUPDU and COREUPD. However, there is a new aspect to these two commands when they are used in STUPDU. This new aspect concerns the updating of the eligibility status of courses. For further clarification, explanations of both "add" and "change" update commands are detailed below.

The general process of adding a student to the student database (STDBASE) parallels the process of adding a course to the course database, or a major to the courses required database. That is, the user must supply the new student's social security number (serves as the key for each student record), name, advisor's name, and major. In addition, the user must supply the name of every course the student has completed--"C", or taken--"T",¹ as well as the number of hours completed in the general areas of environmental studies, restricted electives, and general electives.

Before the new student record can be added to STDBASE, however, a procedure named UPDATE_STATUS is executed. The purpose of UPDATE_STATUS is two fold. One purpose is to calculate the total hours earned. To do this, UPDATE_STATUS uses the record from the courses required database corresponding to the student's major to overlay the student's record. The positions of all courses completed are recorded. By comparing these position indexes with the courses required record, 

¹"T" might apply to an equivalent course taken in high school.
UPDATE_STATUS is able to retrieve the credit hours of the courses which correspond to the indexes recorded. The credit hours retrieved are then totaled along with the credit hours recorded in the student record for the general areas of environmental studies, restricted electives, and electives. This sum is then stored in the TOTAL_HOURS data element of the student record.

The second purpose of UPDATE_STATUS is to determine, on the basis of courses completed or taken, those courses which the student requires and for which the student is eligible. The courses which are investigated are those which have a blank status in the student record. That is, those courses which have not been already marked as completed ('C'), taken ('T'), or eligible ('E').

To accomplish this status update, the student record is examined. When a course is found which possesses a blank status, the position is recorded and the courses required record which corresponds to the student's major is examined to obtain the name of the course. Next, that course's record is read from the courses database. The course's record is inspected to detect the existence of prerequisites. If no prerequisites exist, that course's status is immediately updated to eligible in the student's record. If there are prerequisites, their names are recorded. Then the courses-required database is checked to find the course's position, which corresponds to the position of that course in the student record. Next, that position in the student record is examined. If that position contains either 'C' or 'T', then the prerequisite has been satisfied. If the course had more than one prerequisite, then the other prerequisites are checked in the
same way. Finally, if all prerequisites have been completed or taken, then the course's position in the student record is marked with an "E" for eligible. All courses in the student record which possess a blank status are processed in this way. The one real weakness of this update system is that it is unable to check for the completion of such prerequisites as consent of instructor, senior status, junior status, or such things as high school chemistry.

After the student record has been completely processed by UPDATE_STATUS, the record is then added to the student database in much the same way as courses are added to the course database and as course requirements are added to the course requirements database.

The general process of changing a student record resembles the process of changing a course record or a course requirements record. However, when a student record is updated at the end of a quarter to record completed courses, the CHANGE procedure must also call the UPDATE_STATUS.

As was previously described, UPDATE_STATUS totals the hours completed and changes the status to eligible for any courses for which the student has completed all prerequisites or which do not require any prerequisites.

When a stop command is issued, the listing of all students on the database will automatically be routed to the batch printer.

Figure 7 displays a flowchart of the STUPDT program. Appendix G presents the program listing while Appendix L provides user instructions along with a sample output for STUPDT.
NOTE: SINCE THE PROCESSES OF ADDITION, DELETION, PRINTING, CHANGING, AND STOPPING FOR THIS PROGRAM PARALLEL THOSE PROCESSES USED IN COUPDT, THESE PROCEDURES ARE NOT DETAILED IN THIS FLOWCHART. PLEASE REFER TO FIGURE 5 FOR INFORMATION CONCERNING THE GENERAL FLOW OF THESE PROCESSES. ALSO, THE UPDATE_STATUS PROCEDURE IN THIS PROGRAM IS IDENTICAL TO THAT IN SADVISE. THEREFORE, PLEASE REFER TO FIGURE 4 FOR INFORMATION CONCERNING THE GENERAL FLOW OF UPDATE_STATUS.

FIGURE 7: Flowchart of STUPDT--Student Database Update Program
STUPDT: PROCEDURE OPTIONS (MAIN);

START

INITIALIZATION
OF VARIABLES,
STRUCTURES,
AND FILES

GET_IN:

INPUT
UPDATE
COMMAND

INSTRUCTION

INSTRUCT

INSTRUCT = 1:

GENERAL
UPDATE
PROGRAM USER
INSTRUCTIONS

GO TO
GET_IN

FIGURE 7: Flowchart of STUPDT--Student Database Update Program (continued)
FIGURE 7: Flowchart of STUPDT--Student Database Update Program (continued)
FIGURE 7: Flowchart of STUPDT--Student Database Update Program (continued)
CHAPTER VIII

SYSTEM INTEGRITY AND SECURITY

Throughout the interactive programs, numerous integrity constraints and security measures have been installed to protect the system from possible user or hardware errors.

The problem of integrity is the problem of ensuring—insofar as it can be ensured—that the data in the database is accurate at all times. There is a limit on the extent to which this objective can be achieved. In particular, the system cannot check the correctness of every individual value entered into the database...although it clearly can check each such value for plausibility...[7].

Therefore, the integrity constraints existing in the advisement information system programs check the plausibility of user inputs.

Security, in this case, refers to the measures taken to guard the information in the databases against total or partial loss. In order to preserve the information in the databases, each time a database is successfully updated, a copy of the database is made on another file. The databases and the names of their corresponding security files are:

- STDBASE - FTU.OU.P1095.BKUP.STDBASE
- CODBASE - FTU.OU.P1095.BKUP.CODBASE
- COREDDBS - FTU.OU.P1095.BKUP.COREDDBS

In case of a system crash due to hardware, software, or environmental causes resulting in the total or partial loss of one of the databases, the contents of the security file for that database may be copied into
the database. Then, the activity which was in progress during the crash must be started anew. Complete directions for recovery of information are available in Appendix N.

The user can help to prevent data losses as well as personal inconvenience by following these safeguards:

1. When operating any of the four system programs, proceed with utmost care when inputting data or user commands.

2. Before changing any existing record in any of the databases, first issue a print command to check the current contents of the record.

3. When updating one of the databases, it is best to change, add, or delete a few records; issue a stop command and thereby have the database copied on the security file. By "a few records" means ten, twenty, or even thirty additions, deletions, or changes in contrast to two hundred. This suggestion is made to protect the user from the inconvenience of having to repeat all of the instructions used to update, for instance, two hundred records, in case of a system crash. Processing only a "few" update commands at a time provides the user with a more useful backup system.
CHAPTER IX

CONCLUSIONS AND RECOMMENDATIONS

The interactive student advisement information system provides a convenient, user-oriented system capable of providing advisement reports for each student majoring in one of the engineering options. These advisement reports will be an effective aid for both the student and the student's advisor in defining a feasible, useful trial advisement schedule quickly and accurately.

The system is also capable of providing a fairly accurate prediction of course enrollments for each quarter over a period of one year for all courses offered in connection with one of the undergraduate engineering degree programs.

Although integrity constraints and security measures have been incorporated in all of the interactive programs, only time and system usage will tell whether or not these measures are sufficient. In the event of an unforeseen program "bug", or the desire for additional capabilities arising, it is hoped that the documentation contained herein will be sufficient to aid any future debugging or programming efforts. Figure 8 summarizes the interactions between all programs, time-sharing files, and databases used in the advisement system.
FIGURE 8: Program-Time Sharing Files-Database Interactions
Appendix A: Listing and Output of BUILD_S
BUILD_S:PROCEDURE OPTIONS(MAIN);

DECLARE 1 STUDENT UNALIGNED,
   2 SOCIAL_SECURITY CHARACTER(9),
   2 NAME CHARACTER (25),
   2 MAJOR CHARACTER (20),
   2 ADVISOR CHARACTER (26),
   2 COURSES,
   3 ENGR_CORE(33) CHARACTER(1),
   3 ENVIR_STUDIES_SPE(8) CHARACTER(1),
   3 ADV_ENVIR_STUDIES(3) CHARACTER(1),
   3 ENGR_OPTIONS(10) CHARACTER(1),
   3 ENVIR_STUDIES_GEN(3) FIXED DECIMAL(3,1),
   3 REST_ELECTIVES FIXED DECIMAL(3,1),
   3 ELECTIVES FIXED DECIMAL(3,1),
   2 TOTAL_HOURS FIXED DECIMAL(4,1);

DECLARE STD_BASE FILE RECORD SEQUENTIAL BUFFERED
KEYED ENVIRONMENT(F(147) INDEXED);

READ DATA FOR EACH STUDENT FROM CARDS AND
WRITE THE DATA ON THE FILE NAMED S_DBASE FOR STUDENT
DATABASE.
OPEN FILE(STDBASE) OUTPUT;
ON ENDFILE(SYSIN) GO TO READ_F_1;

READ_F: GET SKIP EDIT(STUDENT)
   (A(9),A(25),A(20),A(26),(54)A(1), (5)F(3,1),F(4,1));
PUT SKIP(2) EDIT(STUDENT)
   (X(1),A(9),A(25),A(20),A(26),SKIP,X(1),(54)A(1), (5)F(4,1),
    F(5,1));
WRITE FILE(STDBASE) FROM (STUDENT)
   KEYFROM(STUDENT,SOCIAL_SECURITY);
GO TO READ_F;

READ_F_1: CLOSE FILE(STDBASE);
OPEN FILE(STDBASE) INPUT;
ON ENDFILE(STDBASE) GO TO EXIT;

READ_F_2: READ FILE(STDBASE) INTO (STUDENT);
PUT SKIP(2) EDIT(STUDENT)
   (X(1),A(9),A(25),A(20),A(26),SKIP,X(1),(54)A(1), (5)F(4,1),
    F(5,1));
GO TO READ_F_2;

EXIT: CLOSE FILE(STDBASE);
END BUILD_SI.
<table>
<thead>
<tr>
<th>ID</th>
<th>Name and Department</th>
<th>Years</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>263041244</td>
<td>MARILYN SELLERS</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>275913256</td>
<td>ELIZABETH CONWAY</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>395244981</td>
<td>RICHARD SEVER</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>396139623</td>
<td>ROBERT LOUIS GALE</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>397873968</td>
<td>SHIRLEY MARY BRICKET</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>419365736</td>
<td>CARLEY JOAN TATE</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>438459113</td>
<td>WENDY MARSHA WILSON</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>462762322</td>
<td>JOHN ROBERT COSHING</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>556751312</td>
<td>DENNIS GENE UPTON</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>556962093</td>
<td>MASHAALLAH KHORSANDI</td>
<td>76-77</td>
<td></td>
</tr>
</tbody>
</table>

Note: The table entries include ID numbers and names, with the years of service and notes. The table is formatted as a list of names and ID numbers, with corresponding years and notes.
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Years</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>263041248</td>
<td>MARILYN SELLERS KHORSANDIEMCS</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>275913256</td>
<td>ELIZABETH CONWAY</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>395244981</td>
<td>RICHARD SEVER</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>396139623</td>
<td>ROBERT LOUIS BALE</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>397873968</td>
<td>SHIRLEY MARY BRICKET</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>418459111</td>
<td>CARLEY JOAN TATE</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>438659113</td>
<td>WENDY MARSHA WILSON</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>462762322</td>
<td>JOHN ROBERT COSHING</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>550751312</td>
<td>DENNIS GENE UPTON</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td>550962093</td>
<td>MASHAALLAH KHORSANDI</td>
<td>76-77</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dr. Christian S. Bauer</td>
<td>76-77</td>
<td>0.0 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td></td>
<td>Dr. Harold I. Klee</td>
<td>76-77</td>
<td>0.0 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td></td>
<td>Dr. Benjamin W. Lin</td>
<td>76-77</td>
<td>0.0 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
</tbody>
</table>
Appendix B: Listing and Output of BUILD_C
BUILD_C: PROCEDURE OPTIONS(MAIN);

BUILD_C: PROCEDURE OPTIONS(MAIN);
DECLARE 1 COURSE UNALIGNED;
   2 NAME CHARACTER(8),
   2 COREQUISITES(3) CHARACTER(8),
   2 PREREQUISITES(5) CHARACTER(8),
   2 SEASONS_OFFERED(4) FIXED DECIMAL(1),
   2 CREDIT_HOURS FIXED DECIMAL(1),
   2 DESCRIPTION CHARACTER(20);

DECLARE CODBASE FILE RECORD SEQUENTIAL HUFEHED
   KEYED ENVIRONMENT(F(97) INDEXED);
OPEN FILE(CODBASE) OUTPUT;
ON ENDFILE(SYSIN) GO TO READ_F_1;
READ_F: GET SKIP EDIT(COURSE)
   ((9)A(8),(5)F(1,0),X(3),A(20));
PUT SKIP(2) EDIT(COURSE)
   (X(1),(9)A(8),(5)F(2,0),X(3),A(20));
WRITE FILE(CODBASE) FROM(COURSE) KEYFROM(COURSE,NAME);
GO TO READ_F;
READ_F_1: CLOSE FILE (CODBASE);
OPEN FILE(CODBASE) INPUT;
ON ENDFILE(CODBASE) GO TO EXIT;
READ_F_2: READ FILE (CODBASE) INTO (COURSE);
PUT SKIP(2) EDIT(COURSE)
   (X(1),(9)A(8),(5)F(2,0),X(3),A(20));
GO TO READ_F_2;
EXIT: CLOSE FILE(CODBASE);
END BUILD_C;
<table>
<thead>
<tr>
<th>Course Code</th>
<th>Course Title</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>EECS 404</td>
<td>Electromagnetic Fields</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 405</td>
<td>Fundamentals of Electromagnetics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 410</td>
<td>Analytical Methods</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 415</td>
<td>Numerical Methods</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 420</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 425</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 430</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 440</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 450</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 460</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 470</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 480</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 490</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 500</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 510</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 520</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 530</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 540</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 550</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 560</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 570</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 580</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 590</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 600</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 610</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 620</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 630</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 640</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 650</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 660</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 670</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 680</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 690</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 700</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 710</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 720</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 730</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 740</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 750</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 760</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 770</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 780</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 790</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 800</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 810</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 820</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 830</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 840</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 850</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 860</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 870</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 880</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 890</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 900</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 910</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 920</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 930</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 940</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 950</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 960</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 970</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 980</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
<tr>
<td>EECS 990</td>
<td>Modern Physics</td>
<td>10000</td>
</tr>
</tbody>
</table>

Note: The above table is a simplified representation and does not include all courses and credits. The credits listed are placeholder values and should be replaced with the actual course credits.
Appendix C: Listing and Output of BUILD_C_R
BLD_C_R: PROCEDURE OPTIONS(MAIN);
   DECLARE 1 COURSES_REQUIRED UNALIGNED,
      2 MAJOR CHARACTER(20),
      2 ENGR_CORE(33),
      3 NAME CHARACTER(8),
      3 CREDIT_HOURS FIXED DECIMAL(1),
      2 ENVIR_STUDIES_SPE(8),
      3 NAME CHARACTER(8),
      3 CREDIT_HOURS FIXED DECIMAL(1),
      2 ADV_ENVIR_STUDIES(3),
      3 NAME CHARACTER(8),
      3 CREDIT_HOURS FIXED DECIMAL(1),
      2 ENGR_OPTIONS(8),
      3 NAME CHARACTER(8),
      3 CREDIT_HOURS FIXED DECIMAL(1),
      2 ENVIR_STUDIES_GEN(3),
      3 NAME CHARACTER(20),
      3 HOURS_LOW FIXED DECIMAL(3,1),
      3 HOURS_HIGH FIXED DECIMAL(3,1),
      2 REST_ELECTIVES FIXED DECIMAL(3,1),
      2 ELECTIVES FIXED DECIMAL(3,1),
      2 TOTAL_HOURS FIXED DECIMAL(4,1);
DECLARE MAJOR1(5) CHARACTER(20) UNALIGNED;
DECLARE 1 ENGR_CORE1(33) UNALIGNED;
  2 NAME CHARACTER(8),
  2 CREDIT_HOURS FIXED DECIMAL(1);
DECLARE 1 ENVIR_STUDIES_SPE1(8) LIKE ENGR_CORE1 UNALIGNED;
DECLARE 1 ADV_ENVIR_STUDIES1(3) LIKE ENGR_CORE1 UNALIGNED;
DECLARE 1 ENGR_OPTIONS1(8) LIKE ENGR_CORE1 UNALIGNED;
DECLARE 1 ENVIR_STUDIES_GEN1(3) UNALIGNED,
  2 NAME CHARACTER(20),
  2 HOURS_LOW FIXED DECIMAL(3,1),
  2 HOURS_HIGH FIXED DECIMAL(3,1);
DECLARE REST_ELECTIVES1 FIXED DECIMAL(3,1) UNALIGNED;
  ELECTIVES1 FIXED DECIMAL(3,1) UNALIGNED,
  TOTAL_HOURS1 FIXED DECIMAL(4,1) UNALIGNED;
DECLARE COREDBS FILE RECORD SEQUENTIAL BUFFERED
  KEYED ENVIRONMENT(F(567) INDEXED);
OPEN FILE(COREDBS) OUTPUT;
ON ENDFILE (SYSIN) GO TO READ_F_1;
GET SKIP EDIT(ENGR_CORE1) 
    ((33) (A(8) \cdot F(2, 0)));
GET SKIP EDIT(ENVIR_STUDIES_SPE1) ((8) (A(8) \cdot F(2, 0)));
GET EDIT(ENVIR_STUDIES_GEN1) ((3) (SKIP \cdot A(20) \cdot X(7) \cdot F(3, 1) \cdot X(7) \cdot F(3, 1)));
GET SKIP EDIT(ADV_ENVIR_STUDIES1) ((3) (A(9) \cdot F(2, 0)));
WRITE_F: DO I = 1 TO 5;
GET SKIP EDIT(MAJOR1(I)) (A(20));
GET SKIP EDIT(ENGR_OPTIONS1) ((9) (A(9) \cdot F(2, 0)));
GET SKIP EDIT(REST_EFFECTIVES, ELECTIVES, TOTAL_HOURS1) 
    ((2) (X(7) \cdot F(3, 1)) \cdot X(6) \cdot F(4, 1));
COURSES_REQUIRED, MAJOR = MAJOR1(I);
DO J = 1 TO 33:
    COURSES_REQUIRED, ENGR_CORE(J) = ENGR_CORE1(J) \cdot BY NAME;
END;
DO J = 1 TO A;
    COURSES_REQUIRED, ENVIR_STUDIES_SPE(J) = 
        ENVIR_STUDIES_SPE1(J) \cdot BY NAME;
END;
DO J = 1 TO 3;
COURSES_REQUIRED.ADV_ENVIR_STUDIES(J) = ADV_ENVIR_STUDIES1(J), BY NAME;
END;
DO J = 1 TO 3;
COURSES_REQUIRED.ENGR_OPTIONS(J) = ENGR_OPTIONS1(J), BY NAME;
END;
DO J = 1 TO 3;
COURSES_REQUIRED.ENVIR_STUDIES_GEN(J) = ENVIR_STUDIES_GEN1(J), BY NAME;
END;
COURSES_REQUIRED.REST_ELECTIVES = REST_ELECTIVES1;
COURSES_REQUIRED.ELECTIVES = ELECTIVES1;
COURSES_REQUIRED.TOTAL_HOURS = TOTAL_HOURS1;
PUT SKIP(2) EDIT(COURSES_REQUIRED)
(A(20), SKIP(2), (3) ((10) (X(1), A(8), F(3+0)), SKIP),
(3) (X(1), A(8), F(3+0)));
SKIP, (8) (X(1), A(8), F(3+0)), SKIP(2), (3) (X(1), A(8), F(3+0)),
SKIP(2), (9) (X(1), A(8), F(3+0)), SKIP(2), (3) (A(20), X(7), F(4+1),
X(7), F(4+1), SKIP), SKIP(2), (2) (SKIP, F(4+1)), SKIP, F(5,1));
WRITE FILE(COREDBS) FROM (COURSES_REQUIRED)
    KEYFROM(COURSES_REQUIRED.MAJOR);
END WRITE_F;
READ_F_1: CLOSE FILE (COREDBS);
PUT PAGE;
OPEN FILE(COREDBS) INPUT;
DO I = 1 TO 5;
READ_F_2: READ FILE (COREDBS) INTO (COURSES_REQUIRED);
PUT SKIP(2) EDIT(COURSES_REQUIRED)
      (A(20),SKIP(2),(3)((10)(X(1),A(8),F(3,0)))*SKIP),
      (3)(X(1),A(8),F(3,0)),
      SKIP,(8)(X(1),A(8),F(3,0)),SKIP(2),(3)(X(1),A(8),F(3,0)),
      SKIP(2),(8)(X(1),A(8),F(3,0)),SKIP(2),(3)(A(20),X(7),F(4,1)),
      X(7),F(4,1),SKIP),SKIP(2),(2)(SKIP,F(4,1)),SKIP,F(5,1));
END;
EXIT: CLOSE FILE(COREDBS);
END BLD_C_R;
<table>
<thead>
<tr>
<th>CEEs 76-77</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP</td>
<td>320</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>MATH</td>
<td>324</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>ENGR</td>
<td>332</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>ENG</td>
<td>310</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>BADM</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td>CEES</td>
<td>401</td>
</tr>
<tr>
<td></td>
<td>CEES</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>CUL &amp; MIST FNDS</td>
<td>7.0</td>
</tr>
<tr>
<td>SOCIAL SCIENCES</td>
<td>9.0</td>
</tr>
<tr>
<td>SCIENCE (EARTH / BIO)</td>
<td>3.0</td>
</tr>
</tbody>
</table>

17.0
0.0
192.0

<table>
<thead>
<tr>
<th>EECS 76-77</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP</td>
<td>320</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>MATH</td>
<td>324</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>ENGR</td>
<td>332</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>ENG</td>
<td>310</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>BADM</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td>EECS</td>
<td>321</td>
</tr>
<tr>
<td></td>
<td>EECS</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>CUL &amp; MIST FNDS</td>
<td>7.0</td>
</tr>
<tr>
<td>SOCIAL SCIENCES</td>
<td>9.0</td>
</tr>
<tr>
<td>SCIENCE (EARTH / BIO)</td>
<td>3.0</td>
</tr>
</tbody>
</table>

14.0
0.0
192.0

<table>
<thead>
<tr>
<th>EMCS 76-77</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP</td>
<td>320</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>MATH</td>
<td>324</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>ENGR</td>
<td>332</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>ENGR</td>
<td>310</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>BADM</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td>EMCS</td>
<td>431</td>
</tr>
<tr>
<td></td>
<td>EMCS</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>CUL &amp; MIST FNDS</td>
<td>7.0</td>
</tr>
<tr>
<td>SOCIAL SCIENCES</td>
<td>9.0</td>
</tr>
</tbody>
</table>
CEES 76-77

COMP 302 3 ENGR 101 3 ENGR 103 4 ENGR 151 4 ENGR 152 3 ENGR 211 4 MATH 211 3 MATH 321 4 MATH 322 4 MATH 323 4
MATH 324 4 PATH 331 4 ENGR 310 4 ENGR 311 4 ENGR 312 5 ENGR 320 4 ENGR 321 4 ENGR 322 4 ENGR 323 4 ENGR 331 3
ENGR 332 4 ENGR 341 3 ENGR 342 3 ENGR 351 3 ENGR 352 3 ENGR 355 3 ENGR 361 3 ENGR 371 3 PHYS 344 3 PHYS 354 3 ENGR 431 3
ENG 310 3 ENGR 442 3 ENGR 443 3
ENG 101 4 SPE 101 3 HUM 201 4 ECON 201 4 0 0 0 0
BADM 3 ENGR 3 ED 3
CEES 401 3 CEES 402 3 CEES 411 4 CEES 412 4 CEES 414 3 0 0 0

CUL & HIST FNDTS 7.0 8.0
SOCIAL SCIENCES 9.6 10.0
SCIENCE(EARTH / BIO) 3.6 4.0

17.0
0.0
192.0

EECS 76-77

COMP 302 3 ENGR 101 3 ENGR 103 4 ENGR 151 4 ENGR 152 3 ENGR 211 4 MATH 211 3 MATH 321 4 MATH 322 4 MATH 323 4
MATH 324 4 PATH 331 4 ENGR 310 4 ENGR 311 4 ENGR 312 4 ENGR 320 4 ENGR 321 4 ENGR 322 4 ENGR 323 4 ENGR 331 3
ENGR 332 4 ENGR 341 3 ENGR 342 3 ENGR 351 3 ENGR 352 3 ENGR 355 3 ENGR 361 3 ENGR 371 3 PHYS 344 3 PHYS 354 3 ENGR 431 3
ENG 310 3 ENGR 442 3 ENGR 443 3
ENG 101 4 SPE 101 3 HUM 201 4 ECON 201 4 0 0 0 0
BADM 3 ENGR 3 ED 3
EECS 321 4 EECS 322 4 EECS 341 4 EECS 411 4 ENGR 421 4 0 0 0 0

CUL & HIST FNDTS 7.0 8.0
SOCIAL SCIENCES 9.6 10.0
SCIENCE(EARTH / BIO) 3.6 4.0

14.0
0.0
192.0

ENGR 76-77

COMP 302 3 ENGR 101 3 ENGR 103 4 ENGR 151 4 ENGR 152 3 ENGR 211 4 MATH 211 3 MATH 321 4 MATH 322 4 MATH 323 4
MATH 324 4 PATH 331 4 ENGR 310 4 ENGR 311 4 ENGR 312 5 ENGR 320 4 ENGR 321 4 ENGR 322 4 ENGR 323 4 ENGR 331 3
ENGR 332 4 ENGR 341 3 ENGR 342 3 ENGR 351 3 ENGR 352 3 ENGR 355 3 ENGR 361 3 ENGR 371 3 PHYS 344 3 PHYS 354 3 ENGR 431 3
ENG 310 3 ENGR 442 3 ENGR 443 3
ENG 101 4 SPE 101 3 HUM 201 4 ECON 201 4 0 0 0 0
BADM 3 ENGR 3 ED 3
EECS 431 4 EMCS 432 4 EMCS 472 4 EECS 411 4 IENG 447 3 0 0 0 0

CUL & HIST FNDTS 7.0 8.0
SOCIAL SCIENCES 9.6 10.0
101
331
341
442
101

302
324
132
310
111

E!',!GA
"ATH
ENGA
ENGA
SPE

~

.. EAS 342

101
331
34t
442
101

BlOt

3 ENGA

3
4
4
3
4

76•77

I

16.n
o.o
192.0

CUL ' t41ST FNOTS
SOCtAL SCtE~CES
SCIENCECEARTH I 810t

ME aS 341

8lf)'4

COfiP
-..aTH
EN(;A
ENG
ENG

MEAS

192.0

o.o

n.n

SCIE~CECEARTH

CUL ' HtST FNDTS
SOCIAl SCIENCES

4 IEMS 424

ENGA
"'ATH
ENGq
ENGR
SPE

-IEMS 301

3
4
4
3
4

3 ENGA

302
124
332
310
101

16•11

I 810)

BAOM ·

COfiP
'4ATH
ENGA
ENG
ENG

I EMS

11i.n
n.o
.192.0

SCIF~CElEAATH

ENGR
ENGA
ENGA
ENGA
HUM

103
310
342
443
201

1o.o
4.0

a.o

3 MEAS 423

4 MEAS 351

1.0
9.0
3.o

3

4 E~GA 151
4 ENGA 311
3 ENGA 351
3
4 ECON 201

e.o
10.0
4.0

3 IEMS 432

3

4
0

·0

0

4

4 MEAS 482

4 MATH 211
4 ENGR 321
3 ENGA 371

3 ENGR 211
5 ENGA 320
3 ENGR 361

3 IEMS 414

0

0

3 IEMS 434

4 MATH 211
4 ENGA 321
3 ENGA 371

3 ENGA 211
5 ENAA 320
3 ENGA 361

3 ENGA 152
4 ENG~ 312
3 E~GA 352

3 I EMS 447

4 E~GA 151 3 ENGA 152
4 ENGA 311 • 4 E~GA 312
3 E~GA 351 3 E~GA 352
3
4 ECON 201 4

4.0

3 EO

3 ENGA 103
4 E~GA 310
3 E~GR 342
3 ENGA 443
3 HUM 201

1.0
9.0
3.o

3 I EMS 461

3 EO

3
4
3
3
3

3.o

0

n .

3 MATH 321
4 ENGA 322
3 PHYS 344

4

0

3 MATH 321
4 ENGA 322
3 PHYS 344

0

4 MATH 322
4 ENGA 323
3 PHYS 354

0

0

4 MATH 322
4 . ENGA 323
3 f'HYS 354

4 MATH 323
4 ENGA 331
3 ENGA 431

4 MATH 323
4 ENGA 331
3 ENGA 431

4
3
3

\
4
3
3


Appendix D: Listing of SADVISE
**SADVISE** PROCEDURE OPTIONS (MAIN);

/* SADVISE -- AN INTERACTIVE PROGRAM WRITTEN IN PL/1 WHICH PROVIDES STUDENTS AND THEIR ADVISORS WITH AN ACCURATE CHECK OF THE STUDENT'S ACADEMIC PROGRESS TOWARDS GRADUATION AND PROVIDES ADMINISTRATIVE PERSONNEL IN THE COLLEGE OF ENGINEERING WITH PROJECTIONS OF COURSE ENROLLMENTS FOR FOUR QUARTERS.

VAR AND IMESARC ARE USED TO TRANSFER A RETURN CODE TO A STUDENT_P STRUCTURE TO THE STUDENT_PREDICT STRUCTURE AND EXTERNAL TIME SHARING CLIST. THE VALUE OF VAR DETERMINES WHETHER THE FINAL PRINTOUT WILL BE ROUTED TO THE TERMINAL OR TO THE BATCH PRINTER. IF VAR = 1 THEN THE PRINTOUT IS SENT TO THE BATCH PRINTER. IF VAR = 0 THEN THE PRINTOUT IS SENT TO THE TERMINAL. */

DECLARE VAR FIXED BINARY(31:0);
DECLARE IMESARC ENTRY(Fixed Binary(31:0));

/* TEMP IS A VARIABLE USED TO FACILITATE INPUT/OUTPUT USING THE ASSEMBLY ROUTINES -- TGET AND TTERMPT, WHICH ARE LISTED IN THE TIME SHARING FILE UNDER THIS USER I.D. WITH THE NAME TERMIO. */

DECLARE TEMP CHARACTER (9);
THE STUDENT STRUCTURE IS USED TO RETRIEVE INFORMATION FROM THE STUDENT DATABASE -- STDBASE.

DECLARE 1 STUDENT UNALIGNED,
   2 SOCIAL_SECURITY CHARACTER(9),
   2 NAME CHARACTER (25),
   2 MAJOR CHARACTER (20),
   2 ADVISOR CHARACTER (26),
   2 COURSES,
      3 ENGR_CORE(33) CHARACTER(1),
      3 ENVIR_STUDIES_SPE(8) CHARACTER(1),
      3 ADV_ENVIR_STUDIES(3) CHARACTER(1),
      3 ENGR_OPTIONS(10) CHARACTER(1),
      3 ENVIR_STUDIES_GEN(3) FIXED DECIMAL(3,1),
      3 REST_ELECTIVES FIXED DECIMAL(3,1),
      3 ELECTIVES FIXED DECIMAL(3,1),
   2 TOTAL_HOURS FIXED DECIMAL(4,1)
THE STUDENT_P STRUCTURE IS USED FOR PURPOSES OF PRINTING THE THE STUDENT RECORD.

DECLARE 1 STUDENT_P UNALIGNED,
2 SOCIAL_SECURITY CHARACTER(9),
2 NAME CHARACTER (25),
2 MAJOR CHARACTER (20),
2 ADVISOR CHARACTER (26),
2 COURSES,
3 ENGR_CORE(33) CHARACTER(1),
3 ENVIR_STUDIES_SPE(8) CHARACTER(1),
3 ADV_ENVIR_STUDIES(3) CHARACTER(1),
3 ENGR_OPTIONS(10) CHARACTER(1),
3 ENVIR_STUDIES_GEN(3) FIXED DECIMAL(3,1),
3 REST_ELECTIVES FIXED DECIMAL(3,1),
3 ELECTIVES FIXED DECIMAL(3,1),
2 TOTAL_HOURS FIXED DECIMAL(4,1);
/* THE STUDENT_1 STRUCTURE IS USED TO HOLD STUDENT RECORDS
   SPECIFIED AFTER THE SOME OPTION IS TAKEN. */

DECLARE 1 STUDENT_1(15) LIKE STUDENT UNALIGNED;

/* THE COURSE_PREDICT_ARRAY IS USED TO STORE THE ENROLLMENTS
   FOR EACH OF FOUR QUARTERS FOR A PARTICULAR STUDENT. */

DECLARE 1 COURSE_PREDICT_ARRAY(63) UNALIGNED,
   2 NAME CHARACTER(8),
   2 QUARTER(4),
   3 OFFERED FIXED DECIMAL(1),
   3 ENROLLMENT FIXED BINARY(4) INITIAL(0);
/* THE COURSE STRUCTURE IS USED TO RETRIEVE INFORMATION FROM THE COURSE DATABASE -- CODBASE. */

DECLARE 1 COURSE UNALIGNED,
   2 NAME CHARACTER(8),
   2 COREQUISITES(3) CHARACTER(8),
   2 PREREQUISITES(5) CHARACTER(8),
   2 SEASONS_OFFERED(4) FIXED DECIMAL(1),
   2 CREDIT_HOURS FIXED DECIMAL(1),
   2 DESCRIPTION CHARACTER(20);

/* THE COURSE_0 STRUCTURE CONTAINS ALL OF THE NAMES OF THE COURSES OFFERED DURING THE SPECIFIED QUARTER. */

DECLARE 1 COURSE_0(63) LIKE COURSE UNALIGNED;

/* PRINT_ARRAY IS USED, AS ITS NAME SUGGESTS, FOR PRINTING THE NAMES OF THE COURSES THE STUDENT IS ELIGIBLE TO TAKE AND WHICH WILL BE OFFERED DURING THE COMING QUARTER. */

DECLARE 1 PRINT_ARRAY(63) CHARACTER(26);
THE COURSES_REQUIRED STRUCTURE IS USED TO RETRIEVE INFORMATION FROM THE COURSE REQUIREMENTS DATABASE -- COREDBS.

DECLARE 1 COURSES_REQUIRED UNALIGNED,
  2 MAJOR CHARACTER(20),
  2 ENGR_CORE(33),
  3 NAME CHARACTER(8),
  3 CREDIT_HOURS FIXED DECIMAL(1),
  2 ENVIR_STUDIES_SPE(8),
  3 NAME CHARACTER(8),
  3 CREDIT_HOURS FIXED DECIMAL(1),
  2 ADV_ENVIR_STUDIES(3),
  3 NAME CHARACTER(8),
  3 CREDIT_HOURS FIXED DECIMAL(1),
  2 ENGR_OPTIONS(8),
  3 NAME CHARACTER(8),
  3 CREDIT_HOURS FIXED DECIMAL(1),
  2 ENVIR_STUDIES_GEN(3),
  3 NAME CHARACTER(20),
  3 HOURS_LOW FIXED DECIMAL(3,1),
  3 HOURS_HIGH FIXED DECIMAL(3,1),
  2 REST_ELECTIVES FIXED DECIMAL(3,1),
  2 ELECTIVES FIXED DECIMAL(3,1),
  2 TOTAL_HOURS FIXED DECIMAL(4,1);
RECORDS OF UP TO 200 STUDENTS FOR PREDICTION PURPOSES.
THAT IS, THE STUDENT_PREDICT(I) STRUCTURE IS 'UPDATED'
BY THE PROGRAM TO FACILITATE MAKING PROJECTIONS FOR
THE ENROLLMENTS OF COURSES OFFERED AS A SPECIFIC REQUIREMENT
FOR ONE OF THE OPTIONS OFFERED BY THE COLLEGE OF ENGINEERING.
IF MORE THAN 200 STUDENTS EXIST, THEN SIMPLY DIMENSION
STUDENT_PREDICT LARGER.

DECLARE 1 STUDENT_PREDICT(200) LIKE STUDENT UNALIGNED!

/* THE COURSE_PREDICT_FALL . . . COURSE_PREDICT_SUMMER
DIMENSIONED STRUCTURES ARE USED TO KEEP TRACK OF THE NAMES
AND ENROLLMENTS OF COURSES OFFERED DURING THE ASSOCIATED
QUARTER. */

DECLARE 1 COURSE_PREDICT_FALL(63) UNALIGNED,
   2 NAME CHARACTER(8) INITIAL(' '),
   2 ENROLLMENT FIXED BINARY(4) INITIAL(0);
DECLARE 1 COURSE_PREDICT_WINTER(63) UNALIGNED,
   2 NAME CHARACTER(8) INITIAL(' '),
   2 ENROLLMENT FIXED BINARY(4) INITIAL(0);
DECLARE 1 COURSE_PREDICT_SPRING(63) UNALIGNED,
   2 NAME CHARACTER(8) INITIAL(' '),
   2 ENROLLMENT FIXED BINARY(4) INITIAL(0);
DECLARE 1 COURSE_PREDICT_SUMMER(63) UNALIGNED,
   2 NAME CHARACTER(8) INITIAL(' '),
   2 ENROLLMENT FIXED BINARY(4) INITIAL(0);
MOST OF THE FOLLOWING VARIABLES ARE USED AS COUNTERS IN VARIOUS PARTS OF THE PROGRAM.

DECLARE OPTION_1 CHARACTER(4) VARYING,
    WHICH_PROC LABEL,
    NUM_OF_STUDENTS FIXED BINARY(3) INITIAL(0),
    SOC_SEC_NO(15) CHARACTER(9),
    QUARTER_COM CHARACTER(6) VARYING,
    (INDXC,INDX) FIXED BINARY(2) INITIAL(0),
    (IFAC,IFW,ISP,ISU) FIXED BINARY(2) INITIAL(0),
    (LBL1,LBL2) LABEL,
    (C_E_COURSES(33),
    S_E_COURSES(8),
    A_E_COURSES(3),
    O_E_COURSES(10)) FIXED BINARY(2) INITIAL(0),
    N_E_COURSES(56) CHARACTER(8),
    JCALL FIXED BINARY(2) INITIAL(0),
    HOURS_TO_GO(6) FIXED DECIMAL(5,1) INITIAL(0),
    JINDX FIXED DECIMAL(2) INITIAL(0),
    (C_C_COURSES(33),
    S_C_COURSES(8),
    A_C_COURSES(3),
    O_C_COURSES(10)) FIXED BINARY(2) INITIAL(0),
    C_COURSES(80) CHARACTER(8),
    (INDXP1,INDXP2,INDXP3,INDXP4,INDXP5,INDXP6)
    FIXED BINARY(2) INITIAL(0),
    (INDXC1,INDXC2,INDXC3,INDXC4,INDXC5) FIXED BINARY(2)
    INITIAL(0).

RLINES, NLINES, AND PRINT_VECTOR ARE USED TO PRINT THE NAMES OF COURSES WHICH THE STUDENT IS EITHER ELIGIBLE FOR OR HAS COMPLETED.

RLINES FIXED DECIMAL(5,1) INITIAL(0),
NLINES FIXED BINARY(4,0) INITIAL(0),
PRINT_VECTOR(8) CHARACTER(8).
/* ALL THREE OF THE DATABASES ARE DECLARED FOR DIRECT KEYED INPUT*/
/* THAT IS, THIS PROGRAM IS INCAPABLE OF UPDATING ANY OF THE DATABASES. */

DECLARE STDBASE FILE RECORD INPUT KEYED
   ENVIRONMENT(F(147) INDEXED).
DECLARE CODBASE FILE RECORD INPUT
   KEYED ENVIRONMENT(F(97) INDEXED).
DECLARE COREDBS FILE RECORD DIRECT INPUT
   KEYED ENVIRONMENT(F(567) Indexed).
ON ENDFILE(STDBASE) GO TO C_PI
ON ENDFILE(CODBASE) GO TO WHICH_STUDENT.

/* THE FOLLOWING ON KEY TESTS ARE INCLUDED TO NOTIFY THE USER OF INCORRECT INFORMATION WITHIN ONE OF THE DATABASES OR WITHIN THE PROGRAM. NEITHER OF THESE TESTS GUARD AGAINST INCORRECT USER INPUT SINCE INFORMATION FOR THESE KEYS ARE NOT INPUT BY THE USER */

ON KEY(CODBASE), BEGIN:
CALL TERMPUTC('FOR THE COURSE NAMED')
CALL TERMPUTC(COURSE .NAME)
CALL TERMPUTC('RECORD NOT FOUND IN THE COURSE DATABASE')
   GO TO FINISH
END:
ON KEY(COREDBS) BEGIN:
CALL TERMPUTC('FOR THE MAJOR NAMED')
CALL TERMPUTC(COURSES_REQUIRED .MAJOR)
CALL TERMPUTC('RECORD NOT FOUND IN COURSE REQUIREMENTS')
CALL TERMPUTC('DATABASE')
CLOSE FILE(COREDBS)
GO TO FINISH
END:
ON ERROR BEGIN:
   CALL IMESARC(VAR)
   STOP
END:
COURSE_PREDICT_ARRAY.QUARTER.ENROLLMENT = 0
COURSE_PREDICT_FALL.ENROLLMENT = 0
COURSE_PREDICT_WINTER.ENROLLMENT = 0
COURSE_PREDICT_SPRING_ENROLLMENT = 0
COURSE_PREDICT_SUMMER_ENROLLMENT = 0
CALL TERMPUT('')
CALL TERMPUT('AFTER EACH COMPUTER ISSUED REQUEST')
CALL TERMPUT('FOR INPUT, ENTER YOUR RESPONSE AND')
CALL TERMPUT('HIT THE RETURN KEY ON THE RIGHT HAND')
CALL TERMPUT('SIDE OF THE TERMINAL''S KEYBOARD')
OP1:
CALL TERMPUT('INDICATE WHICH OF THE FOLLOWING OPTIONS')
CALL TERMPUT('YOU DESIRE BY TYPING EITHER ALL')
CALL TERMPUT('OR SOME')
PUT SKIP(2)
CALL TERMPUT('ALL -INDICATES THE OPTION TO PROCESS')
CALL TERMPUT('ALL OF THE STUDENT RECORDS IN')
CALL TERMPUT('THE STUDENT DATABASE')
CALL TERMPUT('SOME -INDICATES THE OPTION TO')
CALL TERMPUT('PROCESS ONLY SOME OF THE')
CALL TERMPUT('STUDENT RECORDS')
CALL TGET(OPTION_1)

/* PROCESS ALL OR ONLY SOME OF THE STUDENT RECORDS ON THE
STUDENT DATABASE. */

IF OPTION_1 = 'ALL' THEN VAR = 1
ELSE IF OPTION_1 = 'SOME' THEN VAR = 0
QUA:
CALL TERMPUT('AND, IF YOU WOULD BE SO KIND, PLEASE')
CALL TERMPUT('INDICATE WHICH QUARTER IS FORTHCOMING')
CALL TERMPUT('BY ENTERING EITHER FALL, WINTER, SPRING')
CALL TERMPUT('OR SUMMER')
CALL TGET(QUARTER_COM)
IF QUARTER_COM='FALL' OR QUARTER_COM='WINTER' OR
QUARTER_COM='SPRING' OR QUARTER_COM='SUMMER'
THEN GO TO CONTI
ELSE CALL INPUT_MISTAKE_1
GO TO QUA
CONTI:
IF OPTION_1='ALL' THEN WHICH_PROC=SEQUENT
ELSE IF OPTION_1='SOME' THEN CALL INDIV!
ELSE CALL INPUT_MISTAKE_2
INPUT_MISTAKE_2: PROCEDURE
  CALL TERMPTU('"
  CALL TERMPUT(""
  CALL TERMPUT('YOU HAVE MADE AN ERROR')
  CALL TERMPUT('IN ENTERING YOUR RESPONSE TO THE ALL')
  CALL TERMPUT('OR SOME OPTION. THE INSTRUCTIONS WILL BE')
  CALL TERMPUT('GIVEN AGAIN. PLEASE RE-ENTER')
  WHICH_PROC=0P1;
END INPUT_MISTAKE_2

INPUT_MISTAKE_1: PROCEDURE
  PUT SKIP(4) LIST('YOU HAVE MADE AN ERROR IN')
  CALL TERMPUT('ENTERING YOUR RESPONSE TO WHICH QUARTER')
  CALL TERMPUT('IS FORTHCOMING. THE INSTRUCTIONS')
  CALL TERMPUT('WILL BE PRINTED AGAIN. PLEASE RE-ENTER')
END INPUT_MISTAKE_1
/* THIS PROCEDURE PROCESS THE FEW STUDENTS SPECIFIED IF THE 'SOME' IS TAKEN. */

ON KEY(STDBASE) BEGIN
  CALL TERMPUT('FOR SOCIAL SECURITY NUMBER = ');
  CALL TERMPUT(SOCIAL_SECURITY_NO);
  CALL TERMPUT('RECORD NOT FOUND IN STUDENT DATABASE--RE-ENTER ');
  CALL TERMPUT('INPUT ');
  CALL TGET(SOC_SEC_NO(I));
  GO TO READFSI;
END;

PUT SKIP(4);
CALL TERMPUT('YOU HAVE OPTED TO PROCESS A FEW STUDENT ');
CALL TERMPUT('RECORDS INDIVIDUALLY. PLEASE ENTER THE ');
CALL TERMPUT('TOTAL NUMBER OF STUDENTS YOU WISH TO ');
CALL TERMPUT('PROCESS. BY THE WAY, THIS NUMBER MUST BE ');
CALL TERMPUT('LESS THAN OR EQUAL TO 15. ');
MIST:
  CALL TGET(TMP);
  NUM_OF_STUDENTS = BIN(TMP);
  IF NUM_OF_STUDENTS <= 15 THEN GO TO SOCSEC;
  ELSE CALL TERMPUT('I REPEAT ... THE NUMBER MUST BE ');
  CALL TERMPUT('LESS THAN OR EQUAL TO 15. ');
  CALL TERMPUT('PLEASE RE-ENTER ');
  GO TO MIST;
SOCSEC:
  PUT PAGE;
  CALL TERMPUT('PLEASE ENTER A SOCIAL SECURITY NUMBER FOR ');
  CALL TERMPUT('EACH STUDENT ');
  CALL TERMPUT('ENTER IN NUMERICAL ORDER ');
  CALL TERMPUT('ALSO, ENTER EACH AS A CONTINUOUS 9 DIGIT ');
  CALL TERMPUT('NUMBER - OMIT DASHES OR SPACES ');
  DO I = 1 BY 1 TO NUM_OF_STUDENTS;
  CALL TGET(TMP);
  SOC_SEC_NO(I) = TMP;
END;
OPEN FILE(STDBASE) DIRECT;
DO I = 1 BY 1 TO NUM_OF_STUDENTS;
READFSI: READ FILE(STDBASE) INTO (STUDENT_P) KEY (SOC_SEC_NO(I));
STUDENT_I(I) = STUDENT_P, BY NAME
END
CLOSE FILE (STDBASE);
WHICH_PROC = SEQUENT;
END INDIV;
GO TO WHICH_PROC;

SEQUENT: OPEN FILE (CODBASE) SEQUENTIAL BUFFERED;
SEQ1: READ FILE (CODBASE) INTO (COURSE);
INDXC = INDXC + 1;
JINDX = JINDX + 1;
COURSE_PREDICT_ARRAY(JINDX).NAME = COURSE.NAME;
DO I = 1 BY 1 TO 4;

/* THIS SECTION DETERMINES WHICH COURSES WILL BE OFFERED DURING THE APPROACHING QUARTER AND STORES THE INFORMATION FOR THAT COURSE IN THE APPROPRIATE ELEMENT OF COURSE_O. */

COURSE_PREDICT_ARRAY(JINDX).QUARTER(I).OFFERED
= COURSE.SEASONS_OFFERED(I);
END;
IF QUARTER_COM = 'FALL' & COURSE.SEASONS_OFFERED(1) = 1
THEN COURSE_O(INDXC) = COURSE;
ELSE GO TO LB1;
GO TO CL_PRED;

LB1: IF QUARTER_COM = 'WINTER' & COURSE.SEASONS_OFFERED(2) = 1
THEN COURSE_O(INDXC) = COURSE;
ELSE GO TO LB2;
GO TO CL_PRED;

LB2: IF QUARTER_COM = 'SPRING' & COURSE.SEASONS_OFFERED(3) = 1
THEN COURSE_O(INDXC) = COURSE;
ELSE GO TO LB3;
GO TO CL_PRED;

LB3: IF QUARTER_COM = 'SUMMER' & COURSE.SEASONS_OFFERED(4) = 1
THEN COURSE_O(INDXC) = COURSE;
ELSE INDXC = INDXC - 1;

/*/ C_P_ASSIGN ASSIGNS COURSE RECORDS TO ELEMENTS OF THE ARRAYS ESTABLISHED FOR PREDICTION, I.E., COURSE_PREDICT_FALL . . . FOR THOSE COURSES OFFERED DURING THE ASSOCIATED QUARTER. */

CL_PRED: CALL C_P_ASSIGN;
GO TO SEQ1;
C_P.Assigni Procedure

FALL: If COURSE.SSEASONS_OFFERED(1) = 0 Then Go To WINTER
    IFA = IFA + 1;
    COURSE_PREDICT_FALL(IFIA).NAME = COURSE.NAME;
    COURSE_PREDICT_FALL(IFIA).ENROLLMENT = 0;

WINTER: If COURSE.SSEASONS_OFFERED(2) = 0 Then Go To SPRING
    IW = IW + 1;
    COURSE_PREDICT_WINTER(IW).NAME = COURSE.NAME;
    COURSE_PREDICT_WINTER(IW).ENROLLMENT = 0;

SPRING: If COURSE.SSEASONS_OFFERED(3) = 0 Then Go To SUMMER
    ISP = ISP + 1;
    COURSE_PREDICT_SPRING(ISP).NAME = COURSE.NAME;
    COURSE_PREDICT_SPRING(ISP).ENROLLMENT = 0;

SUMMER: If COURSE.SSEASONS_OFFERED(4) = 0 Then Go To STOP
    ISU = ISU + 1;
    COURSE_PREDICT_SUMMER(ISU).NAME = COURSE.NAME;
    COURSE_PREDICT_SUMMER(ISU).ENROLLMENT = 0;

STOP: END C_P.Assigni

WHICH_STUDENT: INDX = INDX + 1;

/* THIS SECTION PROCESSES THE INFORMATION IN EACH STUDENT RECORD TO BE EXAMINED. IF THE 'ALL' OPTION WAS SPECIFIED, THEN STDBASE IS READ SEQUENTIALLY. OTHERWISE, IF THE 'SOME' OPTION WERE SPECIFIED, THE DESIGNATED STUDENT RECORDS, TEMPORARILY STORED IN STUDENT_I(1) ARE ASSIGNED TO THE STUDENT STRUCTURE. INDX KEEPS TRACK OF THE NUMBER OF STUDENTS PROCESSED. */

IF OPTION_1 = 'SOME' THEN STUDENT = STUDENT_I(INDX);
ELSE READ FILE(STDBASE) INTO (STUDENT);

/* FIRST, THE INDEXES OF ALL COURSES FOR WHICH THE STUDENT IS ELIGIBLE ARE RECORDED FOR THE VARIOUS AREAS OF STUDY SUCH AS ENGINEERING CORE COURSES, SPECIFIC ENVIRONMENTAL STUDIES COURSES, ADVANCED ENVIRONMENTAL STUDIES COURSE AREAS, AND ENGINEERING OPTIONS COURSES. */

INDXP1 = 0;
LOOP1: DO I = 1 BY 1 TO 33;
INDXP1 = INDXP1 + 1;
IF STUDENT.COURSES.ENGR_CORE(I) = 'E' THEN
    C_E_COURSES(INDXP1) = I;
ELSE INDXP1 = INDXP1 - 1;
END LOOP1;

LOOP2:
DO I = 1 BY 1 TO 8;
INDXP2 = INDXP2 + 1;
IF STUDENT.COURSES.ENVIR_STUDIES_SPEC(I) = 'E' THEN
    S_E_COURSES(INDXP2) = I;
ELSE INDXP2 = INDXP2 - 1;
END LOOP2;

LOOP3:
DO I = 1 BY 1 TO 3;
INDXP3 = INDXP3 + 1;
IF STUDENT.COURSES.ADV_ENVIR_STUDIES(I) = 'E' THEN
    A_E_COURSES(INDXP3) = I;
ELSE INDXP3 = INDXP3 - 1;
END LOOP3;

LOOP4:
DO I = 1 BY 1 TO 10;
INDXP4 = INDXP4 + 1;
IF STUDENT.COURSES.ENGR_OPTIONS(I) = 'E' THEN
    O_E_COURSES(INDXP4) = I;
ELSE INDXP4 = INDXP4 - 1;
END LOOP4;

INDXC1 = 0;

/* THIS SECTION RECORDS THE INDEXES OF ALL COURSES FOR WHICH THE STUDENT HAS COMPLETED. */

LOOP_1:
DO I = 1 BY 1 TO 33;
INDXC1 = INDXC1 + 1;
IF STUDENT.COURSES.ENGR_CORE(I) = 'C' THEN
    C_C_COURSES(INDXC1) = I;
ELSE INDXC1 = INDXC1 - 1;
END LOOP_1;

INDXC2 = 0;

LOOP_2:
DO I = 1 BY 1 TO 8;
INDEXC2 = INDEXC2 + 1;
IF STUDENT.COURSES.ENVIRON_STUDIES_SPE(I) = 'C' THEN
    S_C_COURSES(INDEXC2) = I;
ELSE INDEXC2 = INDEXC2 - 1;
END LOOP_2;
INDEXC3 = 0;
LOOP_3:
    DO I = 1 BY 1 TO 31
    INDEXC3 = INDEXC3 + 1;
    IF STUDENT.COURSES.ADV_ENVIRON_STUDIES(I) = 'C' THEN
        A_C_COURSES(INDEXC3) = I;
    ELSE INDEXC3 = INDEXC3 - 1;
    END LOOP_3;
INDEXC4 = 0;
LOOP_4:
    DO I = 1 BY 1 TO 101
    INDEXC4 = INDEXC4 + 1;
    IF STUDENT.COURSES.ENGR_OPTIONS(I) = 'C' THEN
        O_C_COURSES(INDEXC4) = I;
    ELSE INDEXC4 = INDEXC4 - 1;
    END LOOP_4;
COURSES_REQUIRED.MAJOR = STUDENT.MAJOR;

/* READ THE RECORD FROM THE COURSE REQUIREMENTS DATABASE WHICH CORRESPONDS TO THE STUDENT'S MAJOR. THIS COURSES_REQUIRED STRUCTURE IS THEN USED TO 'OVERLAY' THE STUDENT'S RECORD IN ORDER TO DETERMINE THE NAMES OF THE COURSES WHOSE INDEXES WERE RECORDED IN THE TWO PREVIOUS SECTIONS. */

READ FILE(COREDBS) INTO (COURSES_REQUIRED)
KEY(COURSES_REQUIRED.MAJOR);

/* RECORD THE NAMES OF THE COURSES FOR WHICH THE STUDENT IS ELIGIBLE. */

LOOP5:
    DO I1 = 1 BY 1 TO INDEXP1
    N_E_COURSES(I1) = COURSES_REQUIRED.ENGR_CORE(C_E_COURSES(I1)).NAME;
    END LOOP5;
LOOP6:
    DO I2 = 1 BY 1 TO INDEXP2
    N_E_COURSES(INDEXP1 + I2) = COURSES_REQUIRED.
    ENVIR_STUDIES_SPE(S_E_COURSES(I2)).NAME;
    END LOOP6;
LOOP7:
    DO I3 = 1 BY 1 TO INDEXP3;

/* CODE */
N_E_COURSES(INDXP1 + INDXP2 + I3) =
COURSES_REQUIRED.ADV_ENVIR_STUDIES(A_E_COURSES(I3)).NAME;
END LOOP71

LOOP81
DO 14 = 1 BY 1 TO INDXP41
N_E_COURSES(INDXP1 + INDXP2 + INDXP3 + I4) =
COURSES_REQUIRED.ENGR_OPTIONS(O_E_COURSES(I4)).NAME;
END LOOP81
INDXP5 = INDXP1 + INDXP2 + INDXP3 + INDXP41

/* RECORD THE NAMES OF THE COURSES WHICH THE STUDENT HAS COMPLETED. */

LOOP_51
DO 11 = 1 BY 1 TO INDXC11
C_COURSES(I1) = COURSES_REQUIRED.ENGR_CORE
(C_C_COURSES(I1)).NAME;
END LOOP_51

LOOP_61
DO 12 = 1 BY 1 TO INDXC21
C_COURSES(INDXC1 + I2) = COURSES_REQUIRED
.ENVIR_STUDIES_SPE(S_C_COURSES(I2)).NAME;
END LOOP_61

LOOP_71
DO 13 = 1 BY 1 TO INDXC31
C_COURSES(INDXC1 + INDXC2 + I3) = COURSES_REQUIRED.
ADV_ENVIR_STUDIES(A_C_COURSES(I3)).NAME;
END LOOP_71

LOOP_81
DO 14 = 1 BY 1 TO INDXC41
C_COURSES(INDXC1 + INDXC2 + INDXC3 + I4) = COURSES_REQUIRED.
ENGR_OPTIONS(O_C_COURSES(I4)).NAME;
END LOOP_81
INDXC5 = INDXC1 + INDXC2 + INDXC3 + INDXC41
/* COMPUTE THE HOURS STILL REQUIRED (UNCOMPLETED) IN THE GENERAL AREAS OF GENERAL ENVIRONMENTAL STUDIES, RESTRICTED ELECTIVES, ELECTIVES, AND TOTAL HOURS REQUIRED FOR GRADUATION. */

LOOP91 DO I5 = 1 TO 3:
  HOURS_TO_GO(I5) = COURSES_REQUIRED.ENVIR_STUDIES_GEN(I5)
  HOURS_LOW = STUDENT.COURSES.ENVIR_STUDIES_GEN(I5);
END LOOP91;

HOURS_TO_GO(4) = COURSES_REQUIRED.REST_ELECTIVES
  - STUDENT.COURSES.REST_ELECTIVES;
HOURS_TO_GO(5) = COURSES_REQUIRED.ELECTIVES
  - STUDENT.COURSES.ELECTIVES;
HOURS_TO_GO(6) = COURSES_REQUIRED.TOTAL_HOURS
  - STUDENT.TOTAL_HOURS;
INDXP6 = 0;

/* COMPARE THE NAMES OF THE COURSES FOR WHICH THE STUDENT IS ELIGIBLE TO THE NAMES OF THE COURSES TO BE OFFERED DURING THE APPROACHING QUARTER. WHEN A MATCH IS FOUND, RECORD THAT COURSES NAME IN THE APPROPRIATE ELEMENT OF THE PRINT_ARRAY STRUCTURE. */

COMPARE1 DO I6 = 1 BY 1 TO INDXP5;
DO I7 = 1 BY 1 TO INDXC1;
INDXP6 = INDXP6 + 1;
IF N_E_COURSES(I6) = COURSE_0(I7).NAME THEN DO:
  IF COURSE_0(I7).COREQUISITES(I) = ' ' THEN
    PRINT_ARRAY(INDXP6) = COURSE_0(I7).NAME|||COURSE_0(I7).COREQUISITES(I)||||
  ELSE PRINT_ARRAY(INDXP6) = COURSE_0(I7).NAME;
END:
ELSE INDXP6 = INDXP6 - 1;
END;
END COMPARE1;

/* PRINT THE TRIAL ADVISEMENT SCHEDULE AND GRADUATION CHECK FOR THE STUDENT. */

CALL PRINT1;
/* IF THE 'SOME' OPTION WAS TAKEN, THEN END THE PROGRAM.  
   HOWEVER, IF THE 'ALL' OPTION WAS SPECIFIED, THEN ASSIGN THE  
   STUDENT STRUCTURE TO THE STUDENT_PREDICT STRUCTURE AND CALL  
   PREDICTARRAY. */

IF OPTION_1 = 'SOME' & INDEX >= NUM_OF_STUDENTS THEN GO TO EXIT
ELSE IF OPTION_1 = 'ALL' THEN DO
   STUDENT_PREDICT(INDEX) = STUDENT_BY_NAME
   INDEXST = INDEX
   CALL PREDICT_ARRAY

/* PREDICT_ARRAY INCREMENTS BY ONE THE ENROLLMENTS OF ALL  
   COURSES FOR WHICH THE STUDENT IS ELIGIBLE AND WHICH WILL  
   BE OFFERED DURING THE COMING QUARTER. */

END
   GO TO WHICH_STUDENT

C_P:
   CLOSE FILE(CODBASE)
   INDEX = INDEX - 1

/* WHEN THE 'ALL' OPTION WAS TAKEN AND ALL STUDENT RECORDS IN  
   THE STUDENT DATABASE HAVE BEEN PROCESSED, THEN CALL THE PREDICT  
   PROCEDURE. THIS PROCEDURE PRINTS A ONE PAGE FOUR QUARTER  
   ENROLLMENT PROJECTION REPORT. */
/* AFTER THE PREDICTION PROCESS HAS BEEN COMPLETED, THE  
   PROGRAM IS FINISHED. */

CALL PREDICT

EXIT:
   GO TO FINISH
PREDICT_ARRAY: PROCEDURE

/* THIS PROCEDURE INCREMENTS BY ONE THE ENROLLMENTS OF COURSES FOR WHICH THE STUDENT IS ELIGIBLE AND WHICH WILL BE OFFERED DURING THE APPROACHING QUARTER. */

/* AFTER THIS INFORMATION IS STORED, THE STATUS OF THE COURSE IN THE STUDENT RECORD IS CHANGED FROM 'E' TO 'C', I.E., FROM ELIGIBLE TO COMPLETED, FOR PREDICTION PURPOSES BY CALLING THE STUD_PRED PROEDURE. */

IF QUARTER_COM = 'FALL' THEN GO TO FALL_11
ELSE IF QUARTER_COM = 'WINTER' THEN GO TO WINTER_11
ELSE IF QUARTER_COM = 'SPRING' THEN GO TO SPRING_11
ELSE IF QUARTER_COM = 'SUMMER' THEN GO TO SUMMER_11

FALL_11:
DO I8 = 1 BY 1 TO INDXPSI
DO I9 = 1 BY 1 TO IFA:
IF N_E_COURSES(I9) = COURSE_PREDICT_FALL(I9).NAME
THEN COURSE_PREDICT_FALL(I9).ENROLLMENT = COURSE_PREDICT_FALL(I9).ENROLLMENT + 1;
IF N_E_COURSES(I8) = COURSE_PREDICT_FALL(I9).NAME
THEN CALL STUD_PRED
END FALL_11
GO TO E_P_A1

WINTER_11:
DO I8 = 1 BY 1 TO INDXPSI
DO I9 = 1 BY 1 TO IWI:
IF N_E_COURSES(I9) = COURSE_PREDICT_WINTER(I9).NAME
THEN COURSE_PREDICT_WINTER(I9).ENROLLMENT = COURSE_PREDICT_WINTER(I9).ENROLLMENT + 1;
IF N_E_COURSES(I8) = COURSE_PREDICT_WINTER(I9).NAME
THEN CALL STUD_PRED
END WINTER_11
GO TO E_P_A1

SPRING_11:
DO I8 = 1 BY 1 TO INDXPSI
DO I9 = 1 BY 1 TO ISP1:
IF N_E_COURSES(I9) = COURSE_PREDICT_SPRING(I9).NAME
THEN COURSE_PREDICT_SPRING(I9).ENROLLMENT = COURSE_PREDICT_SPRING(I9).ENROLLMENT + 1;
IF N_E_COURSES(I8) = COURSE_PREDICT_SPRING(I9).NAME
THEN CALL STUD_PRED
END SPRING_11
GO TO E_P_A1

SUMMER_11:
DO I8 = 1 BY 1 TO INDXPSI
DO I9 = 1 BY 1 TO ISU1:
IF N_E_COURSES(I9) = COURSE_PREDICT_SUMMER(I9).NAME
THEN COURSE_PREDICT_SUMMER(I9).ENROLLMENT = COURSE_PREDICT_SUMMER(I9).ENROLLMENT + 1;
IF N_E_COURSES(I8) = COURSE_PREDICT_SUMMER(I9).NAME
THEN CALL STUD_PRED
END SUMMER_11
E_P_A1: END PREDICT_ARRAY;
FOR THE APPROPRIATE QUARTER, THE ENTIRE PROCESS OF 'ADVISING' THE STUDENT IS REPEATED, THIS TIME BY THE PREDICT_ADVICE PROCEDURE WHICH PARALLELS THE WHICH_STUDENT SECTION OF THE MAIN PROCEDURE. BEFORE THIS ADVISEMENT PROCESS CAN TAKE PLACE, THE STUDENT RECORD MUST BE UPDATED SINCE COURSES STATUSES HAVE BEEN CHANGED FROM 'E' TO 'C'. TO DO THIS, THE PROGRAM CALLS THE UPDATE_STATUS PROCEDURE.

```
ON KEY(COREDBS) BEGIN
 CALL TERMPUT('FOR THE MAJOR NAMED');
 CALL TERMPUT(STUDENT_PREDICT(INDXST).MAJOR);
 CALL TERMPUT('RECORD NOT FOUND IN COURSE REQUIREMENTS');
 CALL TERMPUT(IDATABASE.1);
 CLOSE FILE(COREDBS);
END;

START_P;
 IF QUARTER_COM = 'FALL' THEN QUARTER_COM = 'WINTER';
 ELSE IF QUARTER_COM = 'WINTER' THEN QUARTER_COM = 'SPRING';
 ELSE IF QUARTER_COM = 'SPRING' THEN QUARTER_COM = 'SUMMER';
 ELSE IF QUARTER_COM = 'SUMMER' THEN QUARTER_COM = 'FALL';
 DO INDXST = 1 TO INDX;
 READ FILE(COREDBS) INTO (COURSES_REQUIRED)
 KEY (STUDENT_PREDICT(INDXST).MAJOR);
 CALL UPDATE_STATUS(STUDENT_PREDICT(INDXST));
 CALL PREDICT_ADVISE(STUDENT_PREDICT(INDXST));
 CALL PREDICT_ARRAY;
END;
```

THIS ENTIRE PROJECTION ADVISEMENT PROCESS CONTINUES UNTIL ALL FOUR QUARTERS HAVE BEEN PROCESSED. THEN PRINT_PREDICT IS EXECUTED TO PRINT THE ENROLLMENT PROJECTION REPORT.

```
JCALL = JCALL + 1;
 IF JCALL >= 3 THEN CALL PRINT_PREDICT;
 ELSE GO TO START_P;
END PREDICT;
```
STUD_PREDI PROCEDURE

/* THIS PROCEDURE CHANGES THE STATUS OF COURSES FROM ELIGIBLE TO COMPLETED WHEN THE STUDENT IS 'SCHEDULED' TO TAKE IT. */

IF I0 <= INDXP1 THEN
  STUDENT_PREDICT(INDXST).COURSES.ENG_R_CORE(C_E_COURSES(I0)) = 'C'
ELSE IF I0 <= INDXP1 + INDXP2 THEN
  STUDENT_PREDICT(INDXST).COURSES.ENVIR_STUDIES_SPE(S_E_COURSES(I0 - INDXP1)) = 'C'
ELSE IF I0 <= INDXP1 + INDXP2 + INDXP3 THEN
  STUDENT_PREDICT(INDXST).COURSES.ADV_ENVIR_STUDIES(A_E_COURSES(I0 - (INDXP1 + INDXP2))) = 'C'
ELSE IF I0 <= INDXP1 + INDXP2 + INDXP3 + INDXP4 THEN
  STUDENT_PREDICT(INDXST).COURSES.ENG_R_OPTIONS(O_E_COURSES(I0 - (INDXP1 + INDXP2 + INDXP3))) = 'C'
END STUD_PREDI
PREDICT_ADVICE: PROCEDURE(STUDENT); /*
 THIS PROCEDURE DIRECTLY PARALLELS THE SECTION IN THE MAIN
 PROCEDURE WHICH ADVISES STUDENTS. */

DECLARE 1 STUDENT UNALIGNED,
  2 SOCIAL_SECURITY CHARACTER(9),
  2 NAME CHARACTER(25),
  2 MAJOR CHARACTER(20),
  2 ADVISOR CHARACTER(26),
  2 COURSES,
    3 ENGR_CORE(33) CHARACTER(1),
    3 ENVIR_STUDIES_SPE(B) CHARACTER(1),
    3 ADV_ENVIR_STUDIES(3) CHARACTER(1),
    3 ENGR_OPTIONS(10) CHARACTER(1),
    3 ENVIR_STUDIES_GEN(3) FIXED DECIMAL(3,1),
    3 REST_ELECTIVES FIXED DECIMAL(3,1),
    3 ELECTIVES FIXED DECIMAL(3,1),
  2 TOTAL_HOURS FIXED DECIMAL(4,1);

INDXP1 = 0;

LOOP1: DO I = 1 BY 1 TO 33;
  INDXP1 = INDXP1 + 1;
  IF STUDENT.COURSES.ENGRCORE(I) = 'E' THEN
    C_E_COURSES(INDXP1) = I;
  ELSE INDXP1 = INDXP1 - 1;
END LOOP1;

INDXP2 = 0;

LOOP2: DO I = 1 BY 1 TO 8;
  INDXP2 = INDXP2 + 1;
  IF STUDENT.COURSES.ENVIR_STUDIES_SPE(I) = 'E' THEN
    S_E_COURSES(INDXP2) = I;
  ELSE INDXP2 = INDXP2 - 1;
END LOOP2;

INDXP3 = 0;

LOOP3: DO I = 1 BY 1 TO 3;
INDXP3 = INDXP3 + 1
IF STUDENT.COURSES.ADV_ENVIR_STUDIES(I) = 'E' THEN
  A_E_COURSES(INDXP3) = I
ELSE INDXP3 = INDXP3 - 1
END LOOP3;
INDXP4 = 0;
LOOP4:
  DO I = 1 BY 1 TO 101
  INDXP4 = INDXP4 + 1
  IF STUDENT.COURSES.ENGR_OPTIONS(I) = 'E' THEN
    O_E_COURSES(INDXP4) = I
  ELSE INDXP4 = INDXP4 - 1
  END LOOP4;
READ FILE(COREDBS) INTO (COURSES_REQUIRED)
  KEY (STUDENT.MAJOR);
LOOP5:
  DO I1 = 1 BY 1 TO INDXP1;
  _E_COURSES(I1) = COURSES_REQUIRED.ENGR_CORE(C_E_COURSES(I1)).NAME;
END LOOP5;
LOOP6:
  DO I2 = 1 BY 1 TO INDXP2;
  _E_COURSES(INDXP1 + I2) = COURSES_REQUIRED.ENVIR_STUDIES_SPE(C_E_COURSES(I2)).NAME;
END LOOP6;
LOOP7:
  DO I3 = 1 BY 1 TO INDXP3;
  _E_COURSES(INDXP1 + INDXP2 + I3) = COURSES_REQUIRED.ADV_ENVIR_STUDIES(A_E_COURSES(I3)).NAME;
END LOOP7;
LOOP8:
  DO I4 = 1 BY 1 TO INDXP4;
  _E_COURSES(INDXP1 + INDXP2 + INDXP3 + I4) = COURSES_REQUIRED.ENGR_OPTIONS(O_E_COURSES(I4)).NAME;
END LOOP8;
INDXP5 = INDXP1 + INDXP2 + INDXP3 + INDXP4; END PREDICT_ADVISE;
PRINT_PREDICT: PROCEDURE:

/* THIS PROCEDURE PRINTS A ONE PAGE REPORT WHICH DISPLAYS PROJECTIONS OF CLASS ENROLLMENTS IN ALL COURSES SPECIFIED AS REQUIREMENTS FOR ANY OF THE OPTIONS IN THE ENGINEERING COLLEGE. */

PUT PAGE:
PUT SKIP(10) EDIT ('*PROJECTED COURSE ENROLLMENT*') (X(50),A)
PUT SKIP(2) EDIT ('QUARTER', 'QUARTER') (X(38), A..X(53), A)
PUT SKIP(2) EDIT ('COURSE', 'FALL', 'WINTER', 'SPRING', 'SUMMER', 'COURSE', 'FALL', 'WINTER', 'SPRING', 'SUMMER')

BUILD: DO I = 1 BY 1 TO JINDEX!
ONE: IF COURSE_PREDICT_ARRAY(I),QUARTER(1),OFFERED = 1 THEN GO TO TWO!
     IFA1 = IFA1 + 1!
     COURSE_PREDICT_ARRAY(I),QUARTER(1),ENROLLMENT = COURSE_PREDICT_FALL(IFA1),ENROLLMENT!
TWO: IF COURSE_PREDICT_ARRAY(I),QUARTER(2),OFFERED = 1 THEN GO TO THREE!
     IW1 = IW1 + 1!
     COURSE_PREDICT_ARRAY(I),QUARTER(2),ENROLLMENT = COURSE_PREDICT_WINTER(IW1),ENROLLMENT!
THREE: IF COURSE_PREDICT_ARRAY(I),QUARTER(3),OFFERED = 1 THEN GO TO FOUR!
     ISP1 = ISP1 + 1!
     COURSE_PREDICT_ARRAY(I),QUARTER(3),ENROLLMENT = COURSE_PREDICT_SPRING(ISP1),ENROLLMENT!
FOUR: IF COURSE_PREDICT_ARRAY(I),QUARTER(4),OFFERED = 1
THEN GO TO END_BUILD;

\[
\text{ISU1} = \text{ISU1} + 1; \\
\text{COURSE\_PREDICT\_ARRAY(I)} \cdot \text{QUARTER(4)} \cdot \text{ENROLLMENT} = \text{COURSE\_PREDICT\_SUMMER(ISU1)} \cdot \text{ENROLLMENT}; \\
\text{END\_BUILD}; \text{END\_BUILD}; \\
\text{RLINES} = \text{JINDEX} / 2.1; \\
\text{NLINES} = \text{CEIL(JINDEX / 2.1)}; \\
\text{PUT SKIP}; \\
\text{PRINT_21} \quad \text{DO I = 1 BY 1 TO NLINES}; \\
\text{IF RLINES < NLINES & I = NLINES THEN GO TO ONE_ONLY}; \\
\text{PUT SKIP EDIT(COURSE\_PREDICT\_ARRAY(I)).NAME,} \\
\text{(COURSE\_PREDICT\_ARRAY(I)).QUARTER(J) \cdot ENROLLMENT DO J = 1 TO 4), COURSE\_PREDICT\_ARRAY(NLINES + I)).NAME,} \\
\text{(COURSE\_PREDICT\_ARRAY(NLINES + I)).QUARTER(J) \cdot ENROLLMENT} \\
\text{DO J1 = 1 TO 4)) (2(X(12) \cdot A \cdot X(4)),} \\
\text{F(4 \cdot 0), 3(X(6) \cdot F(4 \cdot 0)))}; \\
\text{END PRINT_21 END ONE_ONLY}; \\
\text{END \_END\_PRINT\_P\_P};
PRINT: PROCEDURE:

/* THIS PROCEDURE PRINTS A ONE PAGE ADVISEMENT AND GRADUATION CHECK REPORT FOR A STUDENT. */

PUT PAGE:
PUT SKIP(10) EDIT('S.S. NO.: STUDENT.SOCIAL_SECURITY:
 'ADVISOR: STUDENT.ADVISOR) (X(6) * X(6) * X(62) * A * A)
((21) _ _ ) (26) _ _ ) (SKIP(0) * X(15) * A * X(64) * A)
PUT EDIT('NAME:', STUDENT.NAME) (SKIP(6) * A * A)
((25) _ _ ) (SKIP(0) * X(11) * A) (**MAJOR:', STUDENT.MAJOR)
(SKIP * X(6) * A * X(2) * A) ( (24) _ _ ) (SKIP(0) * X(12) * A)
('TOTAL HOURS REQUIRED: ', COURSES_REQUIRED * TOTAL_HOURS)
(SKIP * X(6) * A * X(2) * F(5 * 1))
((9) _ _ ) (SKIP(0) * X(27) * A)
('TOTAL HOURS COMPLETED: ', STUDENT.TOTAL_HOURS)
(SKIP * X(6) * A * X(1) * F(5 * 1))( (8) _ _ ) (SKIP(0) * X(28) * A)
PUT EDIT((120) _ _ ) (SKIP * X(6) * A)

PUT EDIT('ENGINEERING', 'BASIC', 'ADVANCED', 'ENGINEERING',
'CORE', 'ENVIRONMENTAL', 'ENVIRONMENTAL',
'OPTION', 'STUDIES', 'STUDIES', (120) _ _ )
(SKIP * X(15) * A * X(23) * A * X(21) * A * X(22) * A *
X(16) * A * X(20) * A * SKIP * X(48) * A * X(22) * A * SKIP * X(6) * A)
('REQUIRED', 'REQUAED', 'REQUAED', 'REQUIRED',
'COMPLETED', 'COMPLETED', 'COMPLETED', 'COMPLETED',
'COMPLETED', 'ELIGIBLE', 'ELIGIBLE', 'ELIGIBLE',
'ELIGIBLE',
(120) _ _ ) (SKIP * X(24) * A * 3 * X(22) * A) * SKIP * X(9) * A * 7 * X(10) * A) * SKIP * X(24) * A * 3 * X(22) * A) * SKIP * X(6) * A)

IC1 = 01
IP1 = 01
IC2 = INDXC1
IP2 = INDXP1
IC3 = INDXC1 + INDXC21
IP3 = INDXP1 + INDXP21
IC4 = INDXC1 + INDXC2 + INDXC31
IP4 = INDXP1 + INDXP2 + INDXP31

START: IC1 = IC1 + 1
IP1 = IP1 + 1
IC2 = IC2 + 1
IP2 = IP2 + 1
IC3 = IC3 + 1
IP3 = IP3 + 1
IC4 = IC4 + 1
IP4 = IP4 + 1
IF IC1 > INDXCl & IP1 > INDXP1 & IC2 > INDXCl & INDXC2 & IP2 > INDXP1 & INDXP2 & IC3 > INDXCl & INDXC2 & INDXC3 & IP3 > INDXP1 & INDXP2 & INDXP3 & IC4 > INDXC5 & IP4 > INDXPS THEN GO TO FINI;
ELSE PRINT_VECTOR(1) = ' ' IF IC1 > INDXCl THEN PRINT_VECTOR(1) = C_COURSES(IC1);
ELSE PRINT_VECTOR(2) = N_E_COURSES(IP1); IF IP1 > INDXP1 THEN PRINT_VECTOR(2) = ' ' IF IC2 > INDXCl + INDXC2 THEN PRINT_VECTOR(3) = ' ' ELSE PRINT_VECTOR(3) = C_COURSES(IC2);
ELSE PRINT_VECTOR(4) = N_E_COURSES(IP2); IF IP2 > INDXP1 + INDXP2 THEN PRINT_VECTOR(4) = ' ' ELSE PRINT_VECTOR(4) = C_COURSES(IC3);
ELSE PRINT_VECTOR(5) = C_COURSES(IC3); IF IC3 > INDXCl + INDXC2 + INDXC3 THEN PRINT_VECTOR(5) = ' ' ELSE PRINT_VECTOR(5) = N_E_COURSES(IP3); IF IC4 > INDXCS THEN PRINT_VECTOR(6) = ' ' ELSE PRINT_VECTOR(6) = C_COURSES(IC4);
ELSE PRINT_VECTOR(7) = N_E_COURSES(IP4); IF IC4 > INDXCS THEN PRINT_VECTOR(7) = ' ' ELSE PRINT_VECTOR(7) = C_COURSES(IC4);
ELSE PRINT_VECTOR(8) = N_E_COURSES(IP4); IF IP4 > INDXPS THEN PRINT_VECTOR(8) = ' ' ELSE PRINT_VECTOR(8) = C_COURSES(IP6); PUT EDIT((PRINT_VECTOR(1)) DO I = 1 TO 8)) (SKIP,X(99),A,7(X(7),A));
GO TO START;
FINI;
PUT EDIT((120)' '(120)' '),(SKIP(0)*X(6),A,SKIP,X(6),A)
PUT EDIT('REQUIRED & ELIGIBLE COURSES NEXT QUARTER')(120)
 1,1)(SKIP(2)*X(50)*A*SKIP*X(6)*A)
PUT EDIT(('PRINT_ARRAY(I) DO I = 1 TO INDXP6))
(SKIP*X(6)*A*X(4)*A))
PUT EDIT((120)'_'*(120)'_' (SKIP(0)*X(6)*A*SKIP*X(6)*A))
PUT EDIT('ENVIRONMENTAL STUDIES', 'RESTRICTED',
'GENERAL AREAS', 'ELECTIVES', (120)'_'
(COURSES_REQUIRED.ENVIR_STUDIESGEN(I), NAME 00 I = 1 TO 3))
(SKIP*X(37)*A*X(21)*A*X(5)*A*SKIP*X(41)*A*X(25)*A*SKIP(0)*
  X(6)*A*SKIP*X(16), A*X(11)*A*X(11)*A)
PUT EDIT('REQUIRED',
(COURSES_REQUIRED.ENVIR_STUDIESGEN(I), HOURS_LOW DO I = 1 TO 3),
COURSES_REQUIRED.REST_ELECTIVES, COURSES_REQUIRED.ELECTIVES)
(SKIP*X(6)*A*X(9) F(3*1)*X(19)*F(3*1)*X(17) F(3*1)*X(15) F(4*1),
X(11)) F(3*1))
PUT EDIT('COMPLETED', (STUDENT.ENVIR_STUDIESGEN(I)
  DO I = 1 TO 3), STUDENT.REST_ELECTIVES, STUDENT.ELECTIVES)
(SKIP*X(6)*A*X(8) F(3*1)*X(19) F(3*1)*X(17) F(3*1)*X(15) F(4*1),
X(11)) F(3*1))
PUT EDIT('REMAINING', (HOURS_TO_GO(I) DO I = 1 TO 5))
(SKIP*X(6)*A*X(8) F(3*1)*X(19) F(3*1)*X(17) F(3*1)*X(15) F(4*1),
X(11)) F(3*1))
END PRINT
UPDATE_STATUS: PROCEDURE(STUDENT); 

/* THIS PROCEDURE UPDATES THE STUDENT_PREDICT STRUCTURE. IT ASSUMES THAT THE STUDENT WILL ENROLL IN AND COMPLETE ALL COURSES FOR WHICH THE STUDENT IS ELIGIBLE AND WHICH WILL BE OFFERED DURING THE COMING QUARTER. */

DECLARE PRERSAT FIXED DECIMAL(1,0); 
OPEN FILE(CODBASE) DIRECT INPUT; 
DECLARE 1 STUDENT UNALIGNED, 
    2 SOCIAL_SECURITY CHARACTERC(9), 
    2 NAME CHARACTER(25), 
    2 MAJOR CHARACTER(20), 
    2 ADVISOR CHARACTER(26), 
    2 COURSES, 
    3 ENGR_CORE (33) CHARACTER(1), 
    3 ENVIR_STUDIES_SPE (8) CHARACTER(1), 
    3 ADV_ENVIR_STUDIES (3) CHARACTER(1), 
    3 ENGR_OPTIONS (10) CHARACTER(1), 
    3 ENVIR_STUDIES_GEN (3) FIXED DECIMAL (3.1), 
    3 ELECTIVES FIXED DECIMAL (3.1), 
    2 TOTAL_HOURS FIXED DECIMAL (4.1); 

TEST1: DO I = 1 TO 33;
// THIS SECTION EXAMINES THE STATUS OF ENGINEERING CORE
 IN THE
 STUDENT'S RECORD. IF THE STATUS IS BLANK THEN OVERLAY THE
 RECORD FROM THE COURSE REQUIREMENTS DATABASE WHICH CORRESPONDS
 TO THE STUDENT'S MAJOR TO DETERMINE THE NAME OF THE COURSE.
 IF THE COURSE HAS PREREQUISITES, CHECK TO SEE IF THE STUDENT
 HAS COMPLETED THESE. IF SO, CHANGE THE STATUS OF THE COURSE
 FROM BLANK TO COMPLETED -- 'C'.

 PRERSAT = 0;
 IF STUDENT.COURSES.ENGR_CORE(I) = ' ' THEN DO:
 IF COURSES_REQUIRED.ENGR_CORE(I).NAME = ' '
 THEN GO TO END_TEST1;
 COURSE.NAME = COURSES_REQUIRED.ENGR_CORE(I).NAME;
 READ FILE(COBASE) INTO(COURSE) KEY(COURSE.NAME);
 DO IJ = 1 TO 51
 IF COURSE.PREREQUISITES(IJ) = ' ' THEN GO TO EL1;
 DO IK = 1 TO 33:
 IF COURSES_REQUIRED.ENGR_CORE(IK).NAME = COURSE.PREREQUISITES(IJ) THEN
 IF STUDENT.COURSES.ENGR_CORE(IK) = 'C'
 | STUDENT.COURSES.ENGR_CORE(IK) = 'T' THEN
 PRERSAT = 1;
 ELSE PRERSAT = 0;
 END;
 ELSE PRERSAT = 0;
 END;
 EL1:
 IF PRERSAT = 0 THEN GO TO END_TEST1;
 END;
 END_TEST1;
 END TEST1;
 TEST2: DO I = 1 TO 8;

/* FOR EACH SPECIFIC ENVIRONMENTAL STUDIES COURSE, IF THE
STATUS IS BLANK IN THE STUDENT RECORD, THEN CHANGE THE STATUS
TO COMPLETED -- 'C' IF THE STUDENT HAS COMPLETED ALL NECESSARY
PREREQUISITES. */

PRERSAT = 0
IF STUDENT.COURSES.ENVIR_STUDIES_SPE(I) = ' ' THEN DO
    IF COURSES_REQUIRED.ENVIR_STUDIES_SPE(I).NAME = ' ' THEN GO TO END_TEST2
    COURSE.NAME = COURSES_REQUIRED.ENVIR_STUDIES_SPE(I).NAME
    READ FILE(CODBASEx) INTO (COURSE) KEY (COURSE Namex)
    DO IJ = 1 TO 51
        IF COURSE.PREREQUISITES(IJ) = ' ' THEN GO TO EL2
        DO IK = 1 TO 81
            IF COURSES_REQUIRED.ENVIR_STUDIES_SPE(IK).NAME = COURSE.PREREQUISITES(IJ) THEN IF
            STUDENT.COURSES.ENVIR_STUDIES_SPE(IK) = 'C' THEN
                PRERSAT = 1
            ELSE PRERSAT = 0
            END
        IF PRERSAT = 0 THEN GO TO END_TEST2
        END
    EL2: IF IJ <= 1 THEN STUDENT.COURSES.ENVIR_STUDIES_SPE(I) = 'E'
        ELSE IF PRERSAT = 1 THEN
            STUDENT.COURSES.ENVIR_STUDIES_SPE(I) = 'E'
        END
    END
END_TEST2: END_TEST2:
TEST3: DO I = 1 TO 81
FOR EACH ENGINEERING OPTIONS COURSE, IF THE COURSES STATUS IS BLANK IN THE STUDENT RECORD, THEN CHANGE THE STATUS TO COMPLETED -- 'C' IF THE STUDENT HAS COMPLETED ALL NECESSARY PREREQUISITES.*

PRERAT = 0
IF STUDENT.COURSES.ENGR_OPTIONS(I) = ' ' THEN DO:
    IF COURSES_REQUIRED.ENGR_OPTIONS(I).NAME = ' ' THEN GO TO END_TEST3;
    COURSE.NAME = COURSES_REQUIRED.ENGR_OPTIONS(I).NAME;
    READ FILE(CODBASE) INTO (COURSE) KEY (COURSE.NAME);
    DO IJ = 1 TO 51
        IF COURSE.PREREQUISITES(IJ) = ' ' THEN GO TO EL3;
        DO IK = 1 TO 81
            IF COURSES_REQUIRED.ENGR_OPTIONS(IK).NAME = COURSE.PREREQUISITES(IJ) THEN IF
                STUDENT.COURSES.ENGR_OPTIONS(IK) = 'C'
                | STUDENT.COURSES.ENGR_OPTIONS(IK) = 'T' THEN PRERAT = 1|
                ELSE PRERAT = 0|
            END;
        END;
    END;
    IF PRERAT = 0 THEN GO TO END_TEST3;
END;
EL3:
IF IJ <= 1 THEN STUDENT.COURSES.ENGR_OPTIONS(I) = 'E';
ELSE IF PRERAT = 1 THEN
    STUDENT.COURSES.ENGR_OPTIONS(I) = 'E';
END;
END_TEST3:
END TEST3;
CLOSE FILE(CODBASE);
END UPDATE_STATUS;
FINISH: CALL IHESARC(VAR);
END SADVISE;

Appendix E: Listing of COUPDT
**COUPDT**: PROCEDURE OPTIONS (MAIN)!

/* COUPDT -- COURSE DATABASE UPDATE PROGRAM -- PROVIDES THE USER WITH THE CAPABILITY OF UPDATING THE COURSE DATABASE BY ADDING NEW COURSES, DELETING COURSES, CHANGING A PORTION OF A COURSE RECORD, OR JUST PRINTING A COURSE RECORD. */

/* SYSTERM IS USED TO PRINT A STRUCTURE DURING USER I/O */

DCL SYSTERM FILE STREAM!

/* IHESARC AND VAR ARE USED TO TRANSFER A RETURN CODE TO A EXTERNAL TIME-SHARING CLIST. THE VALUE OF VAR DETERMINES WHETHER THE FINAL PRINTOUT WILL BE ROUTED TO THE TERMINAL OR TO THE BATCH PRINTER. IF VAR = 1 THEN THE PRINTOUT IS SENT TO THE BATCH PRINTER. IF VAR = 0 THEN THE PRINTOUT IS SENT TO THE TERMINAL. */

DCL IHESARC ENTRY (FIXED BINARY (31, 0))!

/* TEMP IS A VARIABLE USED TO FACILITATE INPUT/OUTPUT USING THE ASSEMBLER ROUTINES -- TGET AND TERMPUT, WHICH ARE LISTED IN THE TIME-SHARING FILE UNDER THIS USER I.D. WITH THE NAME TERMIO1 AND TERMIO2. */

DECLARE TEMP CHARACTER (20) VARYING!
DCL VAR FIXED BIN (31, 0)
DECLARE INSTRUCTION CHARACTER (10) VARYING!
THE COURSE STRUCTURE IS USED TO RETRIEVE INFORMATION FROM THE COURSES DATABASE.

DECLARE 1 COURSE UNALIGNED,
   2 NAME CHARACTER(8),
   2 COREQUISITES(3) CHARACTER(8),
   2 PREREQUISITES(5) CHARACTER(8),
   2 SEASONS_OFFERED(4) FIXED DECIMAL(1),
   2 CREDIT_HOURS FIXED DECIMAL(1),
   2 DESCRIPTION CHARACTER(20);

THE COURSE_P STRUCTURE IS USED AS A TEMPORARY HOLDING AREA FOR COURSES BEING ADDED TO THE COURSES DATABASE -- CODBASE.

DECLARE 1 COURSE_P UNALIGNED LIKE COURSE1
DECLARE ADDITIONS CHARACTER(3) VARYING,
   FINISHED CHARACTER(3) VARYING,
   DELET CHARACTER(3) VARYING,
   CHANGES CHARACTER(3) VARYING;
DECLARE NUM_OF_ADDS FIXED DECIMAL(2,0);

CODBASE -- THE PERMANENT COURSES DATABASE.
CODBSEP -- A TEMPORARY FILE USED TO SORT ALL COURSES BEFORE RECREATING THE COURSES DATABASE WITH BOTH OLD AND NEW COURSES.

DECLARE (CODBASE) FILE RECORD KEYED ENVIRONMENT(F(97) INDEXED);
DECLARE (CODBSEP) FILE RECORD SEQUENTIAL BUFFERED ENVIRONMENT(F(97) CONSECUTIVE REWIN)
DECLARE (SEAS,
   COREQ,
   PREREQ) CHARACTER(3) VARYING;
DECLARE (NU_COREQ,
   NU_PREREQ) FIXED DECIMAL(1,0);
DECLARE NCOURSE FIXED DECIMAL(3,0) INITIAL (0);
/* BECAUSE OF THE FREQUENCY OF USER INSTRUCTIONS, MOST SECTIONS OF THIS PROGRAM DO NOT REQUIRE ADDITIONAL DOCUMENTATION. */

START: PUT PAGE:
   OPEN FILE(SYSTEM) PRINT:
   CALL TERMPUT("THIS PROGRAM, IN CONJUNCTION WITH CORRECT USER IN
   PUT,")
   CALL TERMPUT("UPDATES THE COURSE DATABASE.")
   CALL TERMPUT("IF YOU NEED OPERATING INSTRUCTIONS, TYPE")
   CALL TERMPUT("INSTRUCT.")
   GET_IN:
   CALL TERMPUT("PROVIDE AN UPDATE COMMAND.")
   CALL TOET(INSTRUCTION)
   IF INSTRUCTION = 'INSTRUCT' THEN DO:
      INSTRUCT = 1
      CALL TERMPUT("WHEN THE PROGRAM ASKS FOR AN")
      CALL TERMPUT("UPDATE COMMAND, SUPPLY ONE OF")
      CALL TERMPUT("THE FOLLOWING COMMANDS AND THE")
      CALL TERMPUT("ASSOCIATED ACTION WILL BE EXECUTED.")
      CALL TERMPUT("INSTRUCT - INSTRUCTIONS WILL BE")
      CALL TERMPUT("PRINTED THROUGHOUT THE")
      CALL TERMPUT("PROGRAM.")
      CALL TERMPUT("ADD - ADDS A COURSE RECORD")
      CALL TERMPUT("DELETE - DELETES A COURSE")
      CALL TERMPUT("PRINT - PRINTS A RECORD")
      CALL TERMPUT("CHANGE - PROVIDES THE USER WITH")
      CALL TERMPUT("THE CAPABILITY TO CHANGE")
      CALL TERMPUT("ANY PART OF A COURSE")
      CALL TERMPUT("STOP - STOPS THE PROGRAM AND")
      CALL TERMPUT("Provides THE USER WITH")
      CALL TERMPUT("A FINAL LISTING OF ALL")
      CALL TERMPUT("COURSES DATABASE.")
   END:

137
ELSE IF INSTRUCTION = 'ADD' THEN CALL ADD;
ELSE IF INSTRUCTION = 'DELETE' THEN CALL DELETION;
ELSE IF INSTRUCTION = 'CHANGE' THEN CALL CHANGE;
ELSE IF INSTRUCTION = 'PRINT' THEN DO:
  IF INSTRUCT = 1 THEN DO:
    CALL TERMPUT('INPUT THE COURSE NAME YOU!');
    CALL TERMPUT('WANT PRINTED AS 8');
    CALL TERMPUT('CHARACTERS! AAAA XXX!');
    CALL TERMPUT('WHERE AAAA STANDS FOR THE');
    CALL TERMPUT('ACCEPTED ACRONYM FOR THE');
    CALL TERMPUT('COURSE, WHILE XXX STANDS FOR');
    CALL TERMPUT('THE COURSE NUMBER. FOR EXAMPLE');
    CALL TERMPUT(' ITEMS 420');
  END;
ELSE CALL TERMPUT('COURSE NAME?');
CALL TGET(COURSE,NAME);
OPEN FILE (CODBASE) DIRECT INPUT
READ FILE (CODBASE) INTO (COURSE) KEY
(COURSE.NAME);
CLOSE FILE (CODBASE);
PUT SKIP FILE(SYSTEM);
PUT FILE(SYSTEM) EDIT ('NAME', 'CO-1', 'CO-2', 'CO-3',
  'PR-1', 'PR-2', 'PR-3', 'PR-4', 'PR-5', 'F', 'W', 'SP', 'SU',
  'HR', 'DESCRIPTION',
  (X(2) A 8) (X(4) A) X(2) X(5) X(1) A X(4) A);
PUT SKIP FILE(SYSTEM) EDIT(COURSE) X(1),
  (9) A (8) F (1 0) F (2 0) F (3 0) F (2 0) F (4 0) F (2 0) A;
PUT FILE(SYSTEM) SKIP;
PUT FILE(SYSTEM) SKIP;
END;
ELSE IF INSTRUCTION = 'STOP' THEN DO
OPEN FILE (CODBASE) SEQUENTIAL BUFFERED INPUT
ON ENDFILE (CODBASE) GO TO EXIT
PUT PAGE
PUT SKIP
PUT SKIP EDIT ('NAME', 'CO-1', 'CO-2', 'CO-3', 'PR-1', 'PR-2', 'PR-3', 'PR-4', 'PR-5', 'F', 'W', 'SP', 'SU', 'HR', 'DESCRIPTION')
(X(2), A, (8), (X(4), A), X(2), (5), (X(1), A), X(4), A)
FREAD_FILE READ FILE (CODBASE) INTO (COURSE)
PUT SKIP EDIT (COURSE)
(X(1), (9), A(8), F(1, 0), F(2, 0), F(3, 0), F(2, 0), F(4, 0), X(2), A)
GO TO FREAD_FILE
EXIT CLOSE FILE (CODBASE)
GO TO FINAL
END
GO TO GET_IN
ADDI PROCEDURE!
PUT PAGE!
ADD_INSTRI: CALL TERMPUT('PLEASE INPUT THE TOTAL NUMBER OF');
CALL TERMPUT('COURSES YOU WANT TO ADD.');
CALL TGET(TEMP);
NUM_OF_ADDS = DEC(TEMP);
CALL TERMPUT('ENTER COURSE NAMES IN THE FORM');
CALL TERMPUT('AAAA XXX');
IF INSTRUCT = 1 THEN DO;
CALL TERMPUT('FOR EXAMPLE: IEMS 431');
CALL TERMPUT('IN OTHER WORDS THE ENTIRE COURSE NAME');
CALL TERMPUT('MUST OCCUPY 8 SPACES.');
CALL TERMPUT('ALSO, BE SURE TO ENTER THE COURSES IN');
CALL TERMPUT('ALPHABETICAL AND NUMERICAL ORDER.');
CALL TERMPUT('FOR EXAMPLE:');
CALL TERMPUT('IEMS 433');
CALL TERMPUT('IEMS 434');
CALL TERMPUT('MATH 333');
CALL TERMPUT('MEAS 215');
CALL TERMPUT('MEAS 314');
END;
/* THIS SECTION USES THE TEMPORARY FILE -- CODBSEP, TO SORT
THE COURSES. BOTH OLD AND NEW. ALL COURSES ARE THEN
WRITTEN ON CODBASE -- THE PERMANENT COURSES DATABASE. */
OPEN FILE(CODBSEP) OUTPUT;
ON ENDFILE (CODBASE) GO TO ADD_ON;
OPEN FILE(CODBASE) SEQUENTIAL BUFFERED INPUT;
READ FILE(CODBASE) INTO (COURSE);
READ_C: DO IAD = 1 TO NUM_OF_ADDS;
CALL TERMPUT('COURSE NAME = ');
CALL READ_COURSE_P;
TEST: IF COURSE.NAME <= COURSE_P.NAME THEN DO;
WRITE FILE(CODBSEP) FROM (COURSE);
NCOURSE = NCOURSE + 1;
READ FILE(CODBASE) INTO (COURSE);
END;
ELSE DO;
WRITE FILE(CODBSEP) FROM (COURSE_P);
NCOURSE = NCOURSE + 1;
GO TO END_READ_C;
END;
GO TO TEST;
END_READ_C: END READ_C;
WRITE_F: WRITE FILE (CODBSEP) FROM (COURSE);  
      NCOURSE = NCOURSE + 1;  
READ FILE (CODBASE) INTO (COURSE);  
GO TO WRITE_F;  

ADD_ON1: IF IAD <= NUM_OF_ADDS THEN DO;  
      WRITE FILE (CODBSEP) FROM (COURSE_P);  
      NCOURSE = NCOURSE + 1;  
END;  

IA1:  
      IAD = IAD + 1;  
      IF IAD <= NUM_OF_ADDS THEN DO;  
        CALL READ_COURSE_P;  
        WRITE FILE (CODBSEP) FROM (COURSE_P);  
        NCOURSE = NCOURSE + 1;  
      END;  
      IF IAD <= NUM_OF_ADDS THEN GO TO IA1;  
      CLOSE FILE (CODBSEP);  
      CLOSE FILE (CODBASE);  
      OPEN FILE (CODBSEP) INPUT;  
      OPEN FILE (CODBASE) SEQUENTIAL BUFFERED UPDATE;  
      ON ENDFILE (CODBSEP) GO TO END_ADD;  
      ON ENDFILE (CODBASE) GO TO END_DELETE;  

DELE: READ FILE (CODBASE) INTO (COURSE);  
      DELETE FILE (CODBSEP);  
      GO TO DELE;  

END_DELETE: CLOSE FILE (CODBASE);  
      OPEN FILE (CODBSEP) SEQUENTIAL BUFFERED OUTPUT;  

RED1: READ FILE (CODBASE) INTO (COURSE);  
      CALL TERMINPUT (COURSE, NAME);  
      WRITE FILE (CODBSEP) FROM (COURSE) KEYFROM (COURSE, NAME);  
      GO TO RED1;  

END_ADD: CLOSE FILE (CODBSEP);  
      CLOSE FILE (CODBASE);  
END ADD;
/* This section prompts the user for all necessary information for a course being added to the courses database. All information is temporarily stored in the course_p structure. */

READ_COURSE_P: PROCEDURE;
    CALL TGET (COURSE_P.NAME);
    CALL TERMPUT('ARE THERE ANY COREQUISITES FOR THIS COURSE?');
    CALL TGET (COREQ);
    IF COREQ = 'NO' THEN DO I = 1 TO 3;
        COURSE_P.COREQUISITES(I) = '
    END;
    ELSE DO;
        CALL TERMPUT('HOW MANY COREQUISITES ARE THERE?');
        CALL TGET (TEMP);
        NU_COREQ = DEC(TEMP);
        CALL TERMPUT('LIST THE COREQUISITES');
        DO I = 1 TO NU_COREQ;
            CALL TGET (TEMP);
            COURSE_P.COREQUISITES(I) = TEMP;
        END;
        DO I = NU_COREQ + 1 TO 3;
            COURSE_P.COREQUISITES(I) = '
        END;
    END;
    CALL TERMPUT('ARE THERE ANY PREREQUISITES FOR THIS COURSE?');
    CALL TGET (PREREQ);
    IF PREREQ = 'NO' THEN DO I = 1 TO 5;
        COURSE_P.PREREQUISITES(I) = '
    END;
    ELSE DO;
        CALL TERMPUT('HOW MANY PREREQUISITES ARE THERE?');
        CALL TGET (TEMP);
        NU_PREREQ = DEC(TEMP);
        CALL TERMPUT('LIST THE PREREQUISITES');
        DO I = 1 TO NU_PREREQ;
            CALL TGET (TEMP);
            COURSE_P.PREREQUISITES(I) = TEMP;
        END;
DO I = NU_PREREQ+1 TO 5
COURSE_P.PREREQUISITES(I) = 0
END;
END;
CALL TERMPUT('WILL THIS COURSE BE OFFERED IN THE FALL?');
CALL TGET(SEAS);
IF SEAS = 'NO' THEN COURSE_P.SEASONS_OFFERED(1) = 0;
ELSE COURSE_P.SEASONS_OFFERED(1) = 1;
CALL TERMPUT('WILL THIS COURSE BE OFFERED IN THE WINTER?');
CALL TGET(SEAS);
IF SEAS = 'NO' THEN COURSE_P.SEASONS_OFFERED(2) = 0;
ELSE COURSE_P.SEASONS_OFFERED(2) = 1;
CALL TERMPUT('WILL THIS COURSE BE OFFERED IN THE SPRING?');
CALL TGET(SEAS);
IF SEAS = 'NO' THEN COURSE_P.SEASONS_OFFERED(3) = 0;
ELSE COURSE_P.SEASONS_OFFERED(3) = 1;
CALL TERMPUT('WILL THIS COURSE BE OFFERED IN THE SUMMER?');
CALL TGET(SEAS);
IF SEAS = 'NO' THEN COURSE_P.SEASONS_OFFERED(4) = 0;
ELSE COURSE_P.SEASONS_OFFERED(4) = 1;
CALL TERMPUT('HOW MANY CREDIT HOURS IS THIS COURSE WORTH?');
CALL TGET(TEMP);
    COURSE_P.CREDIT_HOURS = DEC(TEMP);
CALL TERMPUT('PROVIDE A DESCRIPTIVE NAME FOR THIS COURSE');
    CALL TERMPUT('IN 20 CHARACTERS OR LESS');
    CALL TGET(COURSE_P.DESCRIPTION);
END READ_COURSE_P;
DECLARE DEL FIXED DECIMAL(3,0);
OPEN FILE (CODBASE) DIRECT UPDATE;
PUT PAGE;
CALL TERMPUT('HOW MANY COURSES DO YOU WISH TO DELETE?');
CALL TGET(TEMP);
DEL = DEC(TEMP);
DO ID = 1 TO DEL;
CALL TERMPUT('WHAT IS THE NAME (IN EXACTLY 8 CHARACTERS)?');
CALL TGET(COURSE.NAME);
DELETE FILE (CODBASE) KEY (COURSE.NAME);
END;
CLOSE FILE(CODBASE);
END DELETION;
CHANGE: PROCEDURE

/* THIS PROCEDURE ENABLES THE USER TO CHOOSE WHICH SECTION OF
A PARTICULAR COURSE RECORD NEEDS TO BE CHANGED. THE USER
DETERMINES WHICH AREA IS CHANGED BY SUPPLYING A CHANGE
CODE. */

DECLARE NUM_OF_C FIXED DECIMAL(3,0)
DECLARE CODE FIXED DECIMAL (1,0)
DECLARE ANSW CHARACTER(4) VARYING
OPEN FILE (CODBASE) DIRECT UPDATE;
CALL TERMPUT('HOW MANY COURSES DO YOU WISH TO CHANGE?');
CALL TGET(TMP);
NUM_OF_C = DEC(TMP);
NCHANGES: DO IC = 1 TO NUM_OF_C;
CALL TERMPUT('WHAT IS THE NAME OF THE COURSE TO BE CHANGED?');
CALL TGET (COURSE.NAME);
READ FILE (CODBASE) INTO (COURSE) KEY (COURSE.NAME);
IF INSTRUCT = 1 THEN DO;
CALL TERMPUT('PLEASE INPUT THE AREA OF THE COURSE INFORMATION');
CALL TERMPUT('YOU WISH TO CHANGE BY ONE OF THE FOLLOWING');
CALL TERMPUT('APPROPRIATE CODES');
CALL TERMPUT('1 - COREQUISITES');
CALL TERMPUT('2 - PREREQUISITES');
CALL TERMPUT('3 - SEASONS OFFERED');
CALL TERMPUT('4 - CREDIT HOURS');
CALL TERMPUT('5 - DESCRIPTION');
END;
CHG: CALL TERMPUT('CHANGE CODE=');
READ_C: CALL TGET(TMP);
CODE=DEC(TMP);
IF CODE = 1 THEN CALL CHA_COR;
ELSE IF CODE = 2 THEN CALL CHA_PRE;
ELSE IF CODE = 3 THEN CALL CHA_SEA;
ELSE IF CODE = 4 THEN CALL CHA_CRE;
ELSE IF CODE = 5 THEN CALL CHA_DES;
ELSE DO;
CALL TERMPUT('YOU HAVE MADE AN ERROR IN TYPING THE CHANGE');
CALL TERMPUT('CODE, PLEASE RE-ENTER');
GO TO READ_C;
END;
CALL TERMPUT('ARE THERE ANY MORE CHANGES TO THIS COURSE?');
CALL TERMPUT('ANSWER YES OR NO');
CALL TGET(ANSW);
IF ANSW = 'YES' THEN GO TO CHG;
REWRITE FILE (CODBASE) FROM (COURSE) KEY (COURSE.NAME);
END NCHANGES;
CLOSE FILE (CODBASE);
CHA_CORI: PROCEDURE
CALL TERMPUT('INPUT ALL THREE COREQUISITES.'));
IF INSTRUCT = 1 THEN DO;
CALL TERMPUT('IF, FOR INSTANCE, THERE IS ONLY ONE');
CALL TERMPUT(' COREQUISITE, THEN HIT RETURN FOR THE LAST TWO');
END;
DO ICOR = 1 TO 3;
CALL TGET(TEMP);
    COURSE.COREQUISITES(ICOR)=TEMP;
END;
END CHA_CORI
CHA_PREI PROCEDURE:
    CALL TERMPUT('INPUT ALL FIVE PREREQUISITES.');
    IF INSTRUCT = 1 THEN DO:
        CALL TERMPUT('AGAIN, IF, FOR INSTANCE, THERE ARE ONLY TWO.');
        CALL TERMPUT('PREREQUISITES, THEN HIT RETURN FOR THE LAST.');
        CALL TERMPUT('THREE.');
        END;
    DO IPRE = 1 TO 5:
        CALL TGET(TEMP);
        COURSE.PREREQUISITES(IPRE) = TEMP;
    END;
    END CHA_PREI
CHA_SEA: PROCEDURE
CALL TERMPUT("INDICATE FOR EACH QUARTER WHETHER OR NOT");
CALL TERMPUT("THE COURSE WILL BE OFFERED BY INPUTTING 1");
CALL TERMPUT("FOR OFFERED AND 0 FOR NOT OFFERED.");
IF INSTRUCT = 1 THEN DO:
CALL TERMPUT("THE FIRST QUARTER BEING FALL, THEN WINTER,");
CALL TERMPUT("SPRING, AND FINALLY, SUMMER.");
END;
DO ISEA = 1 TO 4;
CALL TGET(TEMP);
    COURSE*SEASONS_OFFERED(ISEA) = DEC(TEMP);
END;
END CHA_SEA;
CHA_CRE1 PROCEDURE:
CALL TEMPLPUT(*INDICATE THE TOTAL CREDIT HOURS FOR THIS*);
CALL TEMPLPUT(*COURSE*);
CALL TGET(TEMP);
    COURSE.CREDIT_HOURS=DEC(TEMP);
END CHA_CRE1
CHA_DESI: PROCEDURE:
CALL TERMPUT('INPUT THE NEW COURSE DESCRIPTION OF 20 CHARACTERS OR LESS. ')
CALL TERMPUT(COURSE.DESCRIPTION)
END CHA_DESI:
END CHANGE:
FINAL: IF INSTRUCT = 1 THEN DO:
  CALL TERMPUT('*TO ROUTE THE FINAL PRINTOUT*');
  CALL TERMPUT('*TO BATCH, TYPE BATCH*');
  CALL TERMPUT('*TO ROUTE THE FINAL PRINTOUT*');
  CALL TERMPUT('*TO THE TERMINAL, TYPE TERM.*');
END;
CALL TERMPUT('*BATCH OR TERMINAL PRINTOUT--*');
CALL TGET(TEMP);
IF TEMP = 'BATCH' THEN VAR = 1;
ELSE IF TEMP = 'TERM' THEN VAR = 0;
ELSE DO:
  CALL TERMPUT('*INPUT ERROR--RE-ENTER*');
  GO TO FINAL;
END;
FINISH: CALL BACKUP;
CALL IMESARC(VAR);
END COUPDT;
Appendix F: Listing of COREUPD
COREUPD: PROCEDURE OPTIONS (MAIN);

/* COREUPD -- COURSE REQUIREMENTS DATABASE UPDATE PROGRAM -- PROVIDES THE USER WITH THE CAPABILITY OF UPDATING THE COURSE REQUIREMENTS DATABASE BY ADDING, DELETING, CHANGING, OR JUST PRINTING A MAJOR'S RECORD. */

/* VAR AND IHESARC ARE USED TO DETERMINE WHETHER OR NOT THE FINAL PRINTOUT IS ROUTED TO BATCH OR THE TERMINAL. */

DECLARE VAR FIXED BINARY(31,0);
DECLARE IHESARC ENTRY (FIXED BINARY(31,0));
DECLARE INSTRUCTION CHARACTER(10) VARYING;
DECLARE INSTRUCT FIXED BINARY(1,0);

/* SYSTERM IS USED TO PRINT A STRUCTURE DURING USER-COMPUTER CONVERSATION. */

DECLARE SYSTERM FILE STREAM;
THE COURSES_REQUIRED STRUCTURE IS USED TO RETRIEVE A RECORD FROM THE COURSE REQUIREMENTS DATABASE. */

DECLARE 1 COURSES_REQUIRED UNALIGNED,
  2 MAJOR CHARACTER(20),
  2 ENGR_CORE(33),
  3 NAME CHARACTER(8),
  3 CREDIT_HOURS FIXED DECIMAL(1),
  2 ENVIR_STUDIES_SPE(8),
  3 NAME CHARACTER(8),
  3 CREDIT_HOURS FIXED DECIMAL(1),
  2 ADV_ENVIR_STUDIES(3),
  3 NAME CHARACTER(8),
  3 CREDIT_HOURS FIXED DECIMAL(1),
  2 ENGR_OPTIONS(8),
  3 NAME CHARACTER(8),
  3 CREDIT_HOURS FIXED DECIMAL(1),
  2 ENVIR_STUDIES_GEN(3),
  3 NAME CHARACTER(20),
  3 HOURS_LOW FIXED DECIMAL(3.1),
  3 HOURS_HIGH FIXED DECIMAL(3.1),
  2 REST_ELECTIVES FIXED DECIMAL(3.1),
  2 ELECTIVES FIXED DECIMAL(3.1),
  2 TOTAL_HOURS FIXED DECIMAL(4.1) !

COURSES_REQUIRED_TEMP IS USED AS A TEMPORARY HOLDING STRUCTURE FOR COURSES BEING ADDED TO THE COURSE REQUIREMENTS DATABASE. -- COREDBS. */

DECLARE 1 COURSES_REQUIRED_TEMP LIKE COURSES_REQUIRED UNALIGNED;

DECLARE (ADDITIONS,
  DELETIONS,
  CHANGES,
  FINISHED) CHARACTER(3) VARYING;

TEMP AND TEMPI ARE VARIABLES USED TO FACILITATE I/O USING THE ASSEMBLER ROUTINES. -- TOET AND TERPUT. */

DECLARE TEMPI CHARACTER(8);
DECLARE TEMP CHARACTER(20) VARYING;
DECLARE (IAD, NMAJOR, NCOURSE, CORE, DELET) FIXED Binary(3, 0) INITIAL(0);

COREDBS -- THE PERMANENT COURSE REQUIREMENTS DATABASE.
REQUPDT -- A TEMPORARY FILE USED TO SORT BOTH OLD AND NEW MAJOR RECORDS BEFORE RECREATING THE PERMANENT DATABASE. */

DECLARE (COREDBS) FILE RECORD KEYED ENVIRONMENT(F(567) INDEXED);
DECLARE (REQUPDT) FILE RECORD SEQUENTIAL BUFFERED ENVIRONMENT(F(567) CONSECUTIVE REWIND);
/* TEST CERTAIN POSSIBLE 'ON' CONDITIONS TO ENSURE SYSTEM
SAFETY. */

ON KEY(COREDBS) BEGIN
CALL TЕРМРUT("RECORD NOT FOUND IN DATABASE--RE-ENTER INPUT");
CLOSE FILE(COREDBS);
GO TO GET_IN;
END;
OPEN FILE (SYSTEM) PRINT;
CALL ТЕРМРUT("THIS PROGRAM, IN CONJUNCTION WITH CORRECT ");
CALL ТЕРМРUT("USER INPUT, UPDATES THE COURSE REQUIREMENTS");
CALL ТЕРМРUT("DATABASE. IF INSTRUCTIONS ARE NEEDED.");
CALL ТЕРМРUT("RESPOND TO THE REQUEST FOR AN UPDATE");
CALL ТЕРМРUT("COMMAND WITH INSTRUCT.");
GET_INSTR
CALL TERMPUT('PROVIDE AN UPDATE COMMAND.**')
CALL TGET(INSTRUCTION)
IF INSTRUCTION = 'INSTRUCT' THEN DO:
  INSTRUCT = 1;
  CALL TERMPUT('WHEN THE PROGRAM ASKS FOR AN**')
  CALL TERMPUT('UPDATE COMMAND, SUPPLY ONE OF*')
  CALL TERMPUT('THE FOLLOWING COMMANDS AND THE**')
  CALL TERMPUT('ASSOCIATED ACTION WILL BE EXECUTED.**')
  CALL TERMPUT('INSTRUCT - INSTRUCTIONS WILL BE**')
  CALL TERMPUT('PRINTED THROUGHOUT THE**')
  CALL TERMPUT('ADD - ADDS A MAJOR RECORD**')
  CALL TERMPUT('DELETE - DELETES A MAJOR**')
  CALL TERMPUT('RECORD.')
  CALL TERMPUT('PRINT - PRINTS A RECORD.')
  CALL TERMPUT('CHANGE - PROVIDES THE USER WITH')
  CALL TERMPUT('THE CAPABILITY TO CHANGE**')
  CALL TERMPUT('ANY PART OF A MAJOR**')
  CALL TERMPUT('RECORD.')
  CALL TERMPUT('STOP - STOPS THE PROGRAM AND**')
  CALL TERMPUT('PROVIDES THE USER WITH')
  CALL TERMPUT('A FINAL LISTING OF ALL**')
  CALL TERMPUT('MAJOR RECORDS IN THE**')
  CALL TERMPUT('COURSE REQUIRE DATABASE.**)')
END
ELSE IF INSTRUCTION = 'ADD' THEN CALL ADD
ELSE IF INSTRUCTION = 'DELETE' THEN CALL DELETES
ELSE IF INSTRUCTION = 'CHANGE' THEN CALL CHANGE
ELSE IF INSTRUCTION = 'PRINT' THEN DO;
   IF INSTRUCT = 1 THEN DO;
      CALL TERPUTF('INPUT THE MAJOR NAME YOU\n');
      CALL TERPUTF('WANT PRINTED AS 15 CHARACTERS\n');
      CALL TERPUTF('AAAABBBBBBXX-XX\n');
      CALL TERPUTF('WHERE AAAA STANDS FOR THE ACCEPTED ACRONYM\n');
      CALL TERPUTF('FOR THE MAJOR. B STANDS FOR A BLANK.\n');
      CALL TERPUTF('XX-XX STANDS FOR THE GOVERNING CATALOG\n');
      CALL TERPUTF('YEARS. FOR EXAMPLE: ITEMS 76-77\n');
   END;
ELSE CALL TERPUTF('MAJOR NAME\n');
   CALL TGET(COURSES_REQUIRED,MAJOR)
OPEN FILE (COREDBS) DIRECT INPUT;
READ FILE (COREDBS) INTO (COURSES_REQUIRED) KEY
(COURSES_REQUIRED,MAJOR);
CLOSE FILE (COREDBS);
PUT FILE (SYSTEM) EDIT
(COURSES_REQUIRED,MAJOR) (X(5),A(20))
('ENGINEERING CORE COURSES') (SKIP(2),X(5),A)
(COURSES_REQUIRED,ENGR_CORE) (SKIP(3),X(5),A(8),F(3,0)))
('SPECIFIC ENVIRONMENTAL STUDIES COURSES') (SKIP(2),X(5),A)
(COURSES_REQUIRED,ENVIR_STUDIES_SPE) (3) (SKIP(3),X(5),A(8),F(3,0))
(A(8),F(3,0)))
('ADVANCED ENVIRONMENTAL STUDIES AREAS') (SKIP(2),X(5),A)
(COURSES_REQUIRED,ADV_ENVIR_STUDIES) (SKIP,
(3) (X(5),A(8),F(3,0)))
('ENGINEERING OPTIONS COURSES') (SKIP(2),X(5),A)
(COURSES_REQUIRED,ENGR_OPTIONS) (3) (SKIP(3),X(5),A(8),F(3,0))
('GENERAL ENVIRONMENTAL STUDIES AREAS') (SKIP(2),X(5),A)
(COURSES_REQUIRED,ENVIR_STUDIES_GEN) (SKIP(3),SKIP(2),X(5),A(20),
F(5,1),F(5,1)))
('HOURS IN RESTRICTED ELECTIVES =\n', COURSES_REQUIRED,REST_ELECTIVES,\n'TOTAL HOURS REQUIRED =\n', COURSES_REQUIRED,TOTAL_HOURS) (3) (SKIP(2),X(5),A,F(6,1)))
PUT FILE (SYSTEM) SKIP!
PUT FILE (SYSTEM) SKIP!
END;
ELSE IF INSTRUCTION = 'STOP' THEN DO
OPEN FILE (COREDBS) SEQUENTIAL BUFFERED INPUT;
on ENDFILE (COREDBS) GO TO EXIT;

FREAD_F: READ FILE (COREDBS) INTO (COURSES_REQUIRED);
PUT PAGE;
PUT SKIP(5) EDIT
(COURSES_REQUIRED.MAJOR) (X(5) A(20))
("ENGINEERING CORE COURSES") (SKIP(2) X(5) A)
(COURSES_REQUIRED.ENGRCORE) (SKIP, (3) (X(5) A(8) F(3,0)))
("SPECIFIC ENVIRONMENTAL STUDIES COURSES") (SKIP(2) X(5) A)
(COURSES_REQUIRED.ENVIR_STUDIES_SPE)((3) (SKIP, (3) (X(5), A(8) F(3,0)))))
("ADVANCED ENVIRONMENTAL STUDIES AREAS") (SKIP(2) X(5) A)
(COURSES_REQUIRED.ADV_ENVIR_STUDIES) (SKIP, (3) (X(5) A(8) F(3,0)))
("ENGINEERING OPTIONS COURSES") (SKIP(2) X(5) A)
(COURSES_REQUIRED.ENGRC_OPTIONS) (SKIP, (3) (X(5) A(8) F(3,0)))
("GENERAL ENVIRONMENTAL STUDIES AREAS") (SKIP(2) X(5) A)
(COURSES_REQUIRED.ENVIR_STUDIES_GEN) (SKIP, (3) (SKIP(2) X(5) A(20), F(5,1) F(5,1)))
("HOURS IN RESTRICTED ELECTIVES =" COURSES_REQUIRED.REST_ELECTIVES,
"HOURS IN ELECTIVES =" COURSES_REQUIRED.ELECTIVES,
"TOTAL HOURS REQUIRED =" COURSES_REQUIRED.TOTAL_HOURS) (3) (SKIP(2) X(5) A F(6,1))
GO TO FREAD_F;

EXIT: CLOSE FILE (COREDBS);
GO TO FINAL;
END;
GO TO GET_IN;
ADD1 PROCEDURE1

/* THIS PROCEDURE ENABLES THE USER TO A NEW MAJOR TO THE COURSE REQUIREMENTS DATABASE. IT PROMPTS THE USER FOR ALL INFORMATION NECESSARY TO FILL THE COURSES_REQUIRED STRUCTURE. */

CALL TERMPUT(' ')
CALL TERMPUT(' ')
ADD_INST: CALL TERMPUT('INPUT THE TOTAL NUMBER OF MAJORS YOU WISH*');
CALL TERMPUT('TO ADD = ANSWER MUST BE AN INTEGER NUMBER*');
CALL GET(TEMP);
    NUM_OF_ADDS = DEC(TEMP);
IF INSTRUCT = 1 THEN DO:
    CALL TERMPUT('ENTER THE NAME OF THE NEW MAJOR*');
    CALL TERMPUT('THIS NAME MUST BE ENTERED IN THE FOLLOWING*');
    CALL TERMPUT('FORMAT = AAAABBBBBXX-XX, IE:A FOUR LETTER*');
    CALL TERMPUT('ABBREVIATION OF THE MAJORS NAME,FOLLOWED BY*');
    CALL TERMPUT('AND THEN THE CATALOG YEARS*');
    CALL TERMPUT('WITH A DASH IN BETWEEN THE TWO DATES*');
    CALL TERMPUT('FOR EXAMPLE - EMCS 77-78*');
END:
ELSE CALL TERMPUT('NEW MAJOR NAME=');
OPEN FILE(REQUPT) OUTPUT;
OPEN FILE(COREDBS) SEQUENTIAL BUFFERED INPUT;
ON ENDFILE(COREDBS) GO TO ADD_ON;
READ FILE (COREDBS) INTO (COURSES_REQUIRED);
READ_CI: DO IAD = 1 TO NUM_OF_ADDS:
    CALL TERMPUT('ENTER THE NAME OF THE NEW MAJOR*');
    CALL READ MAJOR;

/* SORT THE MAJORS AND INSERT THE NEW MAJOR. */

TEST: IF COURSES_REQUIRED.MAJOR <= COURSES_REQUIRED_TEMP.MAJOR THEN DO:
    WRITE FILE(REQUPT) FROM (COURSES_REQUIRED);
    NMAJOR = NMAJOR + 1;
    READ FILE (COREDBS) INTO (COURSES_REQUIRED);
END:
ELSE DO:
    WRITE FILE (REQUPT) FROM (COURSES_REQUIRED_TEMP);
    NMAJOR = NMAJOR + 1;
    GO TO END_READ_CI;
END;
GO TO TEST;
END_READ_CI: END READ_CI;
WRITE_F: WRITE FILE (REQUPT) FROM (COURSES_REQUIRED);
NMAJOR = NMAJOR + 1;
READ FILE (COREDBS) INTO (COURSES_REQUIRED);
GO TO WRITE_F;
RECREATE THE PERMANENT DATABASE, THIS TIME INCLUDING THE NEW MAJOR(S).

ADD_ON: IF IAD <= NUM_OF_ADDS THEN
   DO:
      WRITE FILE (REQPUDT) FROM (COURSES_REQUIRED_TEMP); 
      NMAJOR = NMAJOR + 1; 
      END;
   IA:
      IAD = IAD + 1;
      IF IAD <= NUM_OF_ADDS THEN
         DO:
            CALL READ_MAJOR;
            WRITE FILE (REQPUDT) FROM (COURSES_REQUIRED_TEMP);
            NMAJOR = NMAJOR + 1;
         END;
      IF IAD <= NUM_OF_ADDS THEN GO TO IA;
   CLOSE FILE (REQPUDT);
   CLOSE FILE (COREDBS);
   OPEN FILE (REQPUDT) INPUT;
   ON ENDFILE (REQPUDT) GO TO END_ADD;
   OPEN FILE (COREDBS) SEQUENTIAL BUFFERED UPDATE;
   ON ENDFILE (COREDBS) GO TO END_DEL;
   NCOURSE = NMAJOR - NUM_OF_ADDS;
   DEL:
      READ FILE (COREDBS) INTO (COURSES_REQUIRED);
      DELETE FILE (COREDBS); 
      GO TO DEL;
   END_DEL:
   CLOSE FILE (COREDBS); 
   OPEN FILE (COREDBS) SEQUENTIAL BUFFERED OUTPUT;
   DO I = 1 TO NMAJOR;
      READ FILE (REQPUDT) INTO (COURSES_REQUIRED);
      WRITE FILE (COREDBS) FROM (COURSES_REQUIRED);
      KEYFROM (COURSES_REQUIRED.MAJOR);
   END;
   END_ADD:
   CLOSE FILE (REQPUDT);
   CLOSE FILE (COREDBS); 
END ADD;
READ MAJOR: PROCEDURE;
/* THIS IS THE PROCEDURE WHICH PROMPTS THE USER FOR THE NECESSARY INFORMATION REQUIRED TO FILL THE COURSE_P STRUCTURE SO THAT A NEW COURSE MAY BE ADDED TO THE CODBASE. */

CALL TGET (COURSES_REQUIRED_TEMP. MAJOR);
DO I = 1 TO 3;
   COURSES_REQUIRED_TEMP. ENGR_CORE(I) = COURSES_REQUIRED. ENGR_CORE(I), BY NAME;
END;
DO I = 1 TO 31;
   COURSES_REQUIRED_TEMP. ENVIR_STUDIES_SPE(I) = COURSES_REQUIRED. ENVIR_STUDIES_SPE(I), BY NAME;
END;
DO I = 1 TO 31;
   COURSES_REQUIRED_TEMP. ADV_ENVIR_STUDIES(I) = COURSES_REQUIRED. ADV_ENVIR_STUDIES(I), BY NAME;
END;
CALL TERMPUT('LIST THE ENGINEERING OPTION COURSES');
DO I = 1 TO 31;
   CALL TERMPUT('WHAT IS THE COURSE NAME?');
   CALL TGET(TEMP);
   COURSES_REQUIRED_TEMP. ENGR_OPTIONS(I). NAME = TEMP;
   CALL TERMPUT('HOW MANY HOURS IS THIS COURSE WORTH?');
   CALL TGET(TEMP);
   COURSES_REQUIRED_TEMP. ENGR_OPTIONS(I). CREDIT_HOURS = DEC(TEMP);
END;
IF INSTRUCT = 1 THEN DO;
   CALL TERMPUT('LIST THE GENERAL ENVIRONMENTAL STUDIES');
   CALL TERMPUT('COURSES ALONG WITH LOWER AND UPPER LIMIT ');
   CALL TERMPUT('OF HOURS REQUIRED FOR EACH, FOR EXAMPLE');
   CALL TERMPUT('CUL & HIST FNDTS');
   CALL TERMPUT(' 17');
   CALL TERMPUT(' 19');
END;
ELSE CALL TERMPUT('*GENERAL ENVIRONMENTAL STUDIES*');
DO I = 1 TO 3;
CALL TGET(TEMP);
COURSES_REQUIRED_TEMP.ENVIR_STUDIES_GEN(I).NAME = TEMP;
CALL TERMPUT('*WHAT IS THE LOWER LIMIT OF HOURS REQUIRED?*');
CALL TGET(TEMP);
COURSES_REQUIRED_TEMP.ENVIR_STUDIES_GEN(I).HOURS_LOW = DEC(TEMP);
CALL TERMPUT('*WHAT IS THE UPPER LIMIT OF HOURS REQUIRED?*');
CALL TGET(TEMP);
COURSES_REQUIRED_TEMP.ENVIR_STUDIES_GEN(I).HOURS_HIGH = DEC(TEMP);
ENDI;
CALL TERMPUT('*HOW MANY HOURS OF RESTRICTED ELECTIVES ARE THERE?*');
CALL TGET(TEMP);
COURSES_REQUIRED_TEMP.REST_ELECTIVES = DEC(TEMP);
CALL TERMPUT('*HOW MANY HOURS OF ELECTIVES ARE REQUIRED?*');
CALL TGET(TEMP);
COURSES_REQUIRED_TEMP.ELECTIVES = DEC(TEMP);
CALL TERMPUT('*HOW MANY HOURS ARE REQUIRED TO GRADUATE?*');
CALL TGET(TEMP);
COURSES_REQUIRED_TEMP.TOTAL_HOURS = DEC(TEMP);
END READ_MAJOR;
DELETE PROCEDURE
OPEN FILE (COREDBS) DIRECT UPDATE
CALL TERMPUT('"
CALL TERMPUT('HOW MANY MAJORS DO YOU WISH TO DELETE?’)
CALL TGET(TEMP)
DELET = DEC(TEMP)
DO I = 1 TO DELET;
CALL TERMPUT('WHAT IS THE NAME AND YEARS OF THE MAJOR?’)
CALL TGET(COURSES_REQUIRED.MAJOR)
DELETE FILE(COREDBS) KEY (COURSES_REQUIRED.MAJOR)
END
CLOSE FILE(COREDBS)
END DELETE
CHANGE PROCEDURE
  DECLARE NUM_OF_CODE FIXED DECIMAL(2,0)
  DECLARE ANSW CHARACTER(3) VARYING
  OPEN FILE(COREDBS) DIRECT UPDATE
  CALL TERMPUT('HOW MANY MAJORS DO YOU WISH TO CHANGE?')
  CALL TGET(TEMP)
  NUM_OF_C = DEC(TEMP)
NCHANGES DO ICH = 1 TO NUM_OF_C
  CALL TERMPUT('WHAT IS THE NAME AND THE YEAR OF THE MAJOR?')
  CALL TERMPUT('YOU WISH TO CHANGE?')
  CALL TGET(COURSES_REQUIRED,MAJOR)
  READ FILE(COREDBS) INTO (COURSES_REQUIRED) KEY (COURSES_REQUIRED,MAJOR)
  CALL TERMPUT wollte
  IF INSTRUCT = 1 THEN DO
    CHG
    CALL TERMPUT('INPUT THE AREA OF THE COURSES REQUIRED')
    CALL TERMPUT('FOR THIS MAJOR YOU WISH TO CHANGE')
    CALL TERMPUT('ACCORDING TO ONE OF THE FOLLOWING')
    CALL TERMPUT('APPROPRIATE CODES')
    CALL TERMPUT(0 'NO CHANGES')
    CALL TERMPUT(1 'ENGINEERING CORE')
    CALL TERMPUT(2 'ENVIRONMENTAL STUDIES SPECIFIC')
    CALL TERMPUT(3 'ADVANCED ENVIRONMENTAL STUDIES')
    CALL TERMPUT(4 'ENGINEERING OPTION COURSES')
    CALL TERMPUT(5 'ENVIRONMENTAL STUDIES--GENERAL')
    CALL TERMPUT(6 'RESTRICTED ELECTIVES--HOURS ONLY')
    CALL TERMPUT(7 'ELECTIVES--HOURS ONLY')
    CALL TERMPUT(8 'TOTAL HOURS REQUIRED TO GRADUATE')
  END
  CALL TERMPUT('CHANGE CODE=')
READ_CH1 CALL TGET(TEMP)
CODE = DEC(TEMP)
IF CODE = 0 THEN GO TO PGM_END
ELSE IF CODE = 1 THEN CALL CHA_EC
ELSE IF CODE = 2 THEN CALL CHA_ESS
ELSE IF CODE = 3 THEN CALL CHA_AES
ELSE IF CODE = 4 THEN CALL CHA_EOL
ELSE IF CODE = 5 THEN CALL CHA_ESG
ELSE IF CODE = 6 THEN CALL CHA_RE
ELSE IF CODE = 7 THEN CALL CHA_E
ELSE IF CODE = 8 THEN CALL CHA_TH
ELSE DO
  CALL TERMPUT('YOU HAVE MADE AN ERROR IN TYPING THE CHANGE')
  CALL TERMPUT('PLEASE RE-ENTER.')
  GO TO READ_CH1
END

CALL TERMPUT('ARE THERE ANY MORE CHANGES TO THIS COURSE?')
CALL TERMPUT('YES OR NO')
CALL TGET(ANSW)
IF ANSW = 'YES' THEN GO TO CHG1
REWRITE FILE(COREDRS) FROM (COURSES_REQUIRED) KEY (COURSES_REQUIRED.MAJOR)
END CHANGES
CLOSE FILE (COREDRS)
CHA_EC: PROCEDURE;
CALL TERMPUT(*OLD ENGINEERING CORE COURSE NAME=*);
CALL TGET(TEMP);
CALL TERMPUT(*WHAT IS THE NEW NAME??);
CALL TGET(TEMP1);
DO I = 1 TO 33;
   IF TEMP = COURSES_REQUIRED.ENGR_CORE(I).NAME THEN DO;
      COURSES_REQUIRED.ENGR_CORE(I).NAME = TEMP1;
   CALL TERMPUT(*HOW MANY HOURS IS THIS COURSE WORTH??);
   CALL TGET(TEMP);
   COURSES_REQUIRED.ENGR_CORE(I).CREDIT_HOURS =
      DEC(TEMP);
   GO TO END_CHA;
END;
END;
END_CHA: END CHA_EC;
CHA_ESSI: PROCEDURE
    CALL TGETPUT("OLD ENVIRONMENTAL STUDIES COURSE NAME=");
    CALL TGET(TEMP);
    CALL TGETPUT("WHAT IS THE NEW NAME?");
    CALL TGET(TEMP);
    DO I = 1 TO 8;
        IF TEMP = COURSES_REQUIRED.ENVIRONMENTAL_STUDIES_SPEC(I).NAME THEN DO;
            COURSES_REQUIRED.ENVIRONMENTAL_STUDIES_SPEC(I).NAME = TEMP;
        CALL TGETPUT("HOW MANY HOURS IS THIS COURSE WORTH?");
        CALL TGET(TEMP);
            COURSES_REQUIRED.ENVIRONMENTAL_STUDIES_SPEC(I).CREDIT_HOURS = DEC(TEMP);
        GO TO END_CHA:
    END;
END;
END_CHA: END CHA_ESSI;
CHA_AESI: PROCEDURE;
    CALL TGETPUT('OLD ADVANCED ENVIRONMENTAL COURSE AREA NAME=*');
    CALL TGET(TEMP);
    CALL TGETPUT('WHAT IS THE NEW NAME?');
    CALL TGET(TEMP);
    DO I = 1 TO 3;
        IF TEMP = COURSES_REQUIRED.ADV_ENVIR_STUDIES(I).NAME THEN DO;
            COURSES_REQUIRED.ADV_ENVIR_STUDIES(I).NAME = TEMP;
        CALL TGETPUT('HOW MANY HOURS IS THIS COURSE WORTH??');
        CALL TGET(TEMP);
        COURSES_REQUIRED.ADV_ENVIR_STUDIES(I).CREDIT_HOURS =
            DEC(TEMP);
        GO TO END_CHAI;
    END;
END;
END_CHAI; END CHA_AESI;
CHA_E0: PROCEDURE
   CALL TERMPUT('*OLD ENGINEERING OPTION COURSE NAME=*');
   CALL TGET(TEMP);
   CALL TERMPUT('*WHAT IS THE NEW NAME??*');
   CALL TGET(TEMP1);
   DO I = 1 TO 81
   IF TEMP = COURSES_REQUIRED.ENG_R_OPTIONS(I).NAME THEN DO:
      COURSES_REQUIRED.ENG_R_OPTIONS(I).NAME=TEMP1;
      CALL TERMPUT('*HOW MANY HOURS IS THIS COURSE WORTH??*');
      CALL TGET(TEMP);
      COURSES_REQUIRED.ENG_R_OPTIONS(I).CREDIT_HOURS = DEC(TEMP);
      GO TO END_CHAI
   END:
END:
END_CHAI: END CHA_E0
CHA_ESG: PROCEDURE:
   CALL TERMPUT(*OLD GENERAL ENVIRONMENTAL STUDIES*);
   CALL TERMPUT(*COURSE AREA NAME IN 20 CHARACTERS OR LESS*);
   CALL TGET(TEMP);
   CALL TERMPUT(*WHAT IS THE NEW NAME??*);
   CALL TGET(TEMP1);
   DO I = 1 TO 31
      IF TEMP = COURSES_REQUIRED.ENVIR_STUDIES_GEN(I).NAME THEN DO;
         COURSES_REQUIRED.ENVIR_STUDIES_GEN(I).NAME = TEMP1;
      CALL TERMPUT(*WHAT IS THE LOWER LIMIT OF HOURS REQUIRED??*);
      CALL TGET(TEMP);
         COURSES_REQUIRED.ENVIR_STUDIES_GEN(I).HOURS_LOW = DEC(TEMP);
      CALL TERMPUT(*WHAT IS THE UPPER LIMIT OF HOURS REQUIRED??*);
      CALL TGET(TEMP);
         COURSES_REQUIRED.ENVIR_STUDIES_GEN(I).HOURS_HIGH = DEC(TEMP);
      GO TO END_CHA;
   END;
   END
END_CHA: END CHA_ESG;

170
PROCEDURE:
CALL TERMINPUT('HOW MANY HOURS OF RESTRICTED ELECTIVES?');
CALL TERMINPUT('ARE ALLOWED?');
CALL TGET(TEMP);
COURSES_REQUIRED.REST_ELECTIVES = DEC(TEMP);
END CHA_RE1
CHA_E:  PROCEDURE;
    CALL TTERMPUT(*HOW MANY HOURS OF ELECTIVES ARE REQUIRED*);
    CALL TGET(TEMP);
    COURSES_REQUIRED.ELECTIVES = DEC(TEMP);
END CHA_E;
PROCEDURE:
CALL TERMUPR("HOW MANY TOTAL HOURS ARE REQUIRED?");
CALL TGET(TEMP);
COURSES_REQUIRED, TOTAL_HOURS = DEC(TEMP);
END CHA_TH1;
PGM_END: END CHANGE
FINAL: IF INSTRUCT = 1 THEN DO:
   CALL TERMPUT("TO ROUTE THE FINAL PRINTOUT TO BATCH");
   CALL TERMPUT("TYPE BATCH TO ROUTE THE FINAL ");
   CALL TERMPUT("PRINTOUT TO THE TERMINAL, TYPE TERM.");
END;
CALL TERMPUT("BATCH OR TERMINAL PRINTOUT--");
CALL TGET(TEMP);
IF TEMP = 'BATCH' THEN VAR = 1;
ELSE IF TEMP = 'TERM' THEN VAR = 0;
ELSE DO:
   CALL TERMPUT("INPUT ERROR--RE-ENTER");
   GO TO FINAL;
END;
FINISH: CALL BACKUP;
CALL IHESARC(VAR);
END COREJPD
Appendix G: Listing of STUPDT
STUPDT: PROCEDURE OPTIONS(MAIN);

/* STUPDT -- STUDENT UPDATE PROGRAM -- A PROGRAM WRITTEN IN
PL/1 TO PROVIDE THE USER WITH THE CAPABILITY OF CORRECTING
AS WELL NORMAL COURSE STATUS UPDATING */

DECLARE SYSTEM FILE STREAM;

/* THE STUDENT STRUCTURE IS USED TO RETRIEVE INFORMATION FROM
THE STUDENT DATABASE -- STDBASE */

DECLARE 1 STUDENT UNALIGNED,
  2 SOCIAL_SECURITY CHARACTER(9),

  2 NAME CHARACTER(25),
  2 MAJOR CHARACTER(20),
  2 ADVISOR CHARACTER(26),
  2 COURSES,
    3 ENGR_CORE(33) CHARACTER(1),
    3 ENVIR_STUDIES_SPE(8) CHARACTER(1),
    3 ADV_ENVIR_STUDIES(3) CHARACTER(1),
    3 ENGR_OPTIONS(10) CHARACTER(1),
    3 ENVIR_STUDIES_GEN(3) FIXED DECIMAL(3,1),
    3 REST_ELECTIVES FIXED DECIMAL(3,1),
    3 ELECTIVES FIXED DECIMAL(3,1),
  2 TOTAL_HOURS FIXED DECIMAL(4,1);

/* THE STUDENT_P STRUCTURE IS USED TO TEMPORARILY HOLD INFORMATION
PERTAINING TO A STUDENT RECORD BEING PRINTED FROM THE STUDENT
DATABASE. */

DECLARE 1 STUDENT_P LIKE STUDENT UNALIGNED;

/* VAR AND IHERSARC ARE USED TO TRANSFER A RETURN CODE TO AN
EXTERNAL TIME SHARING CLIST. THE VALUE OF VAR DETERMINES
WHETHER THE FINAL PRINTOUT WILL BE ROUTED TO THE TERMINAL OR
TO THE BATCH PRINTER. IF VAR = 1 THEN THE PRINTOUT IS SENT
TO THE BATCH PRINTER. IF VAR = 0 THEN THE PRINTOUT IS SENT
TO THE TERMINAL. */

DECLARE VAR FIXED BINARY(31,0);
DECLARE IHERSARC ENTRY(FIXED BINARY(31,0));
/* The COURSES_REQUIRED structure is used to retrieve records corresponding to a student's major from the course requirements database. */

DECLARE 1 COURSES_REQUIRED·UNALIGNED,
2 MAJOR CHARACTER(20),
2 ENGR_CORE(33),
3 NAME CHARACTER(8),
3 CREDIT_HOURS FIXED DECIMAL(1),
2 ENVIR_STUDIES_SPE(8),
3 NAME CHARACTER(8),
3 CREDIT_HOURS FIXED DECIMAL(1),
2 ADV_ENVIR_STUDIES(3),
3 NAME CHARACTER(8),
3 CREDIT_HOURS FIXED DECIMAL(1),
2 ENGR_OPTIONS(8),
3 NAME CHARACTER(8),
3 CREDIT_HOURS FIXED DECIMAL(1),
2 ENVIR_STUDIES_GEN(3),
3 NAME CHARACTER(20),
3 HOURS_LOW FIXED DECIMAL(3,1),
3 HOURS_HIGH FIXED DECIMAL(3,1),
2 REST_ELECTIVES FIXED DECIMAL(3,1),
2 ELECTIVES FIXED DECIMAL(3,1),
2 TOTAL_HOURS FIXED DECIMAL(4,1)
-- THE COURSE STRUCTURE IS USED TO RETRIEVE INFORMATION CONCERNING A CERTAIN COURSE FROM THE COURSE DATABASE.

DECLARE 1 COURSE UNALIGNED,
  2 NAME CHARACTER(8),
  2 COREQUISITES(3) CHARACTER(8),
  2 PREREQUISITES(5) CHARACTER(8),
  2 SEASONS_OFFERED(4) FIXED DECIMAL(1),
  2 CREDIT_HOURS FIXED DECIMAL(1),
  2 DESCRIPTION CHARACTER(20);
THE FOLLOWING VARIABLES ARE USED AS COUNTERS IN VARIOUS SECTIONS OF THE PROGRAM. */

DECLARE (NUMCORT, NUMEST, NUMAET, NUMEO) FIXED BINARY(2,0) INITIAL(0);
DECLARE (NUMCORE, NUMESS, NUMAES, NUMEO) FIXED BINARY(2,0) INITIAL(0);

CNAME IS USED AS TEMPORARY STORAGE FOR A COURSE NAME INPUT BY THE USER. THIS NAME IS THEN COMPARED TO THE NAMES OF COURSES IN, FOR INSTANCE, THE RECORD IN THE COURSE REQUIREMENTS DATABASE CORRESPONDING TO THE STUDENT'S MAJOR. */

DECLARE (CNAME) CHARACTER(8);
DECLARE (ICOUNT, WROTE) FIXED BINARY(2,0) INITIAL(0);
DECLARE CODE FIXED BINARY(1,0) INITIAL(0);
DECLARE STATUS FIXED BINARY(1,0) INITIAL(0);

THE STUDENT_TEMP STRUCTURE IS USED TO TEMPORARILY HOLD INFORMATION PERTAINING TO A STUDENT RECORD BEING ADDED TO THE STUDENT DATABASE. */

DECLARE 1 STUDENT_TEMP LIKE STUDENT UNALIGNED;

TEMP IS USED TO READ AND PRINT USER I/O TO AND FROM THE TERMINAL. */

DECLARE TEMP CHARACTER(25) VARYING;
DECLARE INSTRUCT FIXED BINARY(1,0) INITIAL(0);
DECLARE INSTRUCTION CHARACTER(10) VARYING;
DECLARE SOCIAL_SECURITY_NO CHARACTER(9);

STDTEMP IS A TEMPORARY FILE USED TO HELP INSERT NEW STUDENT RECORDS TO STDBASE -- THE STUDENT DATABASE. */

THE COURSES DATABASE AND THE COURSE REQUIREMENTS DATABASE ARE BOTH USED AS INPUT ONLY. STDTEMP AND STDBASE MAY BE USED AS BOTH INPUT AND OUTPUT FILES. */

DECLARE (COREDBS) FILE RECORD KEYED DIRECT INPUT ENVIRONMENT(F(567) INDEXED);
DECLARE (CDBASE) FILE RECORD KEYED DIRECT INPUT ENVIRONMENT(F(97) INDEXED);
DECLARE (STDTEMP) FILE RECORD SEQUENTIAL BUFFERED ENVIRONMENT(F(147) CONSECUTIVE REWIND);
DECLARE (STDBASE) FILE RECORD KEYED ENVIRONMENT(F(147) INDEXED);
/* CERTAIN *ON* CONDITIONS ARE TESTED TO HELP ENSURE USER I/O ACCURACY. */

OPEN FILE(SYSTEM) PRINT1
ON ERROR BEGIN: VAR = 11
    CALL IHESARC(VAR);
END1
ON KEY(CODBASEx) BEGIN1
CALL TERMPLUT("FOR THE COURSE NAMED")1

CALL TERMPLUT(COURSE, NAME1;
CALL TERMPLUT("RECORD NOT FOUND IN THE COURSE DATABASE")1;
CALL TERMPLUT("--RE-ENTER INPUT")1;
    CLOSE FILE(CODBASEx;
    GO TO GET_IN1;
    END1;
ON KEY(STDBASE) BEGIN;
CALL TERMPLUT("FOR SOCIAL SECURITY NUMBER = ");
CALL TERMPLUT(SOCIAL_SECURITY_NUM0X);
CALL TERMPLUT("RECORD NOT FOUND IN STUDENT DATABASE--RE-ENTER ");
CALL TERMPLUT("INPUT")1;
    CLOSE FILE(STDBASE);1
    GO TO GET_IN1;
    END;
ON KEY(COREDBS) BEGIN1;
CALL TERMPLUT("FOR THE MAJOR NAMED");
CALL TERMPLUT(COURSES_REQUIRED, MAJOR);
CALL TERMPLUT("RECORD NOT FOUND IN COURSE REQUIREMENTS")1;
CALL TERMPLUT("DATABASE--RE-ENTER INPUT")1;
    CLOSE FILE(COREDBS);1
    GO TO GET_IN1;
    END1;
CALL TERMPUT('');
CALL TERMPUT('');
CALL TERMPUT('THIS PROGRAM IN CONJUNCTION WITH CORRECT USER');
CALL TERMPUT('INPUT, UPDATES THE STUDENT DATABASE.');
CALL TERMPUT('IF YOU NEED OPERATING INSTRUCTIONS, TYPE');
CALL TERMPUT('INSTRUCT.');

GET_IN: CALL TERMPUT('PROVIDE AN UPDATE COMMAND');
CALL TGET(INSTRUCTION);
IF INSTRUCTION = 'INSTRUCT' THEN INSTRUCT = 1;
ELSE IF INSTRUCTION = 'PRINT' THEN DO;
   CALL TERMPUT('WHAT IS THE STUDENT S SOCIAL SECURITY NUMBER?');
   CALL TGET(SOCIAL_SECURITY_NO);
   OPEN FILE(STDBASE) DIRECT INPUT;
/
   FIND THAT STUDENT S RECORD IN THE STUDENT DATABASE
   AND RETRIEVE THE INFORMATION IN THAT RECORD USING
   THE STUDENT STRUCTURE. */

   READ FILE(STDBASE) INTO(STUDENT) KEY (SOCIAL_SECURITY_NO);
   CLOSE FILE(STDBASE);
   PUT FILE(SYSTEM) EDIT(STUDENT) (SKIP,X(1),A(9),A(25),
      A(20),A(26),SKIP,X(1),X(54),A(1),X(5),F(4),F(5),F(5));
   PUT FILE(SYSTEM) SKIP;
   PUT FILE(SYSTEM) SKIP;
END;
/* FIND THE STUDENT'S RECORD CORRESPONDING TO THE SOCIAL SECURITY NUMBER INPUT BY THE USER, AND DELETE IT. */

ELSE IF INSTRUCTION = 'DELETE' THEN DO;
    CALL TGET('WHAT IS THE STUDENT'S SOCIAL SECURITY NUMBER?');
    CALL TGET(SOCIAL_SECURITY_NO);
    OPEN FILE(STDBASE) DIRECT UPDATE;
    DELETE FILE(STDBASE). KEY(SOCIAL_SECURITY_NO);
    CLOSE FILE(STDBASE);
END;
/* WHEN THE USER ISSUES A STOP COMMAND, PRINT ALL
   STUDENT RECORDS IN THE STUDENT DATABASE TO ENSURE
   THE USER OF CORRECT I/O. */

ELSE IF INSTRUCTION = 'STOP' THEN DO:
  PUT PAGE;
  OPEN FILE(STDBASE)SEQUENTIAL BUF INPUT:
  ON ENDFILE(STDBASE) GO TO EXIT;
  READ_F:
    READ FILE(STDBASE) INTO(STUDENT);
    PUT SKIP(2) EDIT(STUDENT)
       (X(1),A(9),A(25),A(20),A(26),SKIP,X(1),
        (54)A(1),(5)F(4*1),F(5*1))
    GO TO READ_F;
END;

/* IF THE USER ISSUES AN ADD COMMAND, THE FIRST STEP IS TO
INITIALIZE THE TEMPORARY HOLDING STRUCTURE FOR THE
NEW STUDENT'S INFORMATION -- STUDENT_P. */

ELSE IF INSTRUCTION = 'ADD' THEN DO;
  STUDENT_P.SOCIAL_SECURITY = 'I
  STUDENT_P.NAME = 'I
  STUDENT_P.MAJOR = 'I
  STUDENT_P.ADVISOR = 'I
  DO I = 1 TO 33;
    STUDENT_P.COURSES.ENG_DECOR(I) = 'I
  END;
  DO I = 1 TO 8;
    STUDENT_P.COURSES.ENVIR_STUDIES_SPE(I) = 'I
  END;
  DO I = 1 TO 3;
    STUDENT_P.COURSES.ADV_ENVIR_STUDIES(I) = 'I
  END;
  DO I = 1 TO 10;
    STUDENT_P.COURSES.ENG_OPTIONS(I) = 'I
  END;
  DO I = 1 TO 3;
    STUDENT_P.COURSES.ENVIR_STUDIES_GEN(I) = 0.
  END;
  STUDENT_P.COURSES.REST_ELECTIVES = 0.1
  STUDENT_P.COURSES.ELECTIVES = 0.1
  STUDENT_P.TOTAL_HOURS = 0.1
END;
THEN, ASK THE USER TO SUPPLY ALL NECESSARY INFORMATION FOR THE NEW STUDENT'S RECORD, GIVING INSTRUCTIONS IF THEY WERE INITIALLY REQUESTED. /*

IF INSTRUCT = 1 THEN DO
    CALL TERMINPUT('*
    CALL TERMINPUT('INPUT THE NEW STUDENTS SOCIAL SECURITY'
    CALL TERMINPUT('NUMBER AS A 9 DIGIT NUMBER--I.E.*WITHTHE
    CALL TERMINPUT('DASHES.*
    END
ELSE CALL TERMINPUT('SOCIAL SECURITY NUMBER=*
CALL TGET(STUDENT_P,SOCIAL_SECURITY)
IF INSTRUCT = 1 THEN DO
    CALL TERMINPUT('*
    CALL TERMINPUT('INPUT THE NEW STUDENTS NAME IN 25*
    CALL TERMINPUT('CHARACTERS OR LESS.*
    END
ELSE CALL TERMINPUT('*STUDENT NAME=
CALL TGET(STUDENT_P,NAME)
IF INSTRUCT = 1 THEN DO
    CALL TERMINPUT('*)
    CALL TERMINPUT('INPUT THE STUDENTS MAJOR IN THE FOLLOWING*'
    CALL TERMINPUT('FORMAT CONSISTING OF 15 SPACES*'
    CALL TERMINPUT('*
    CALL TERMINPUT('AAAAABBBBBBXX-XX*
    CALL TERMINPUT('*
    CALL TERMINPUT('WHERE AAAA REPRESENTS THE ACCEPTED 4 LETTER*'
    CALL TERMINPUT('ACCRONYM OF THE MAJOR,E.G.,IEMS,EECS,ETC.*'
    CALL TERMINPUT('AND,WHERE BBBBBB REPRESENTS 6 SPACES.*'
    CALL TERMINPUT('AND,WHERE XX-XX REPRESENTS THE YEARS OF *'
    CALL TERMINPUT('THE GOVERNING CATALOG, E.G., 76-77*'
    CALL TERMINPUT('*
    CALL TERMINPUT('AN EXAMPLE IS: IEMS 76-77*'
END
ELSE CALL TERMPUT(' *STUDENTS MAJOR=*');
CALL TGET(STUDENT_P, MAJOR);
OPEN FILE(COREDBS) DIRECT INPUT;
READ FILE(COREDBS) INTO (COURSES_REQUIRED)
   KEY (STUDENT_P, MAJOR);
CLOSE FILE(COREDBS);
IF INSTRUCT = 1 THEN DO;
   CALL TERMPUT('');
   CALL TERMPUT(' *INPUT THE STUDENTS ADVISORS NAME IN*');
   CALL TERMPUT(' 26 CHARACTERS OR LESS*');
END;
ELSE CALL TERMPUT(' *STUDENTS ADVISORS NAME=*');
CALL TGET(STUDENT_P, ADVISOR);

CALL INPUT_COURSES(STUDENT_P);

CALL INPUT_HOURS(STUDENT_P);

CALL UPDATE_STATUS(STUDENT_P);
/* NOW, SORT ALL OF THE STUDENT RECORDS -- BOTH NEW AND OLD
WITH REFERENCE TO THEIR SOCIAL SECURITY NUMBERS.
THEN, RECREATE THE STUDENT DATABASE FILE, THIS TIME
WITH THE NEW STUDENT RECORD(S). */

OPEN FILE (STDBASE) SEQUENTIAL BUFFERED INPUT;
ON ENDFILE (STDBASE) GO TO END_ADD;
OPEN FILE (STDTMP) OUTPUT;
WROTE = 0;
READ FILE (STDBASE) INTO (STUDENT);
TEST: IF STUDENT.SOCIAL_SECURITY <= STUDENT_P.SOCIAL_SECURITY
THEN DO:
  WRITE FILE (STDTMP) FROM (STUDENT);
  READ FILE (STDBASE) INTO (STUDENT);
END;
ELSE DO:
  WRITE FILE (STDTMP) FROM (STUDENT_P);
  WROTE = 1;
END;
IF WROTE = 0 THEN GO TO TEST;
ELSE DO:
WRITE_F: WRITE FILE (STDTMP) FROM (STUDENT);
READ FILE (STDBASE) INTO (STUDENT);
GO TO WRITE_F;
END_ADD: CLOSE FILE (STDBASE);
CLOSE FILE (STDTMP);
OPEN FILE (STDTMP) INPUT;
OPEN FILE (STDBASE) SEQUENTIAL BUFFERED UPDATE;
ON ENDFILE (STDTMP) GO TO END_ADD;
ON ENDFILE (STDBASE) GO TO END_DEL;
DEL: READ FILE (STDBASE) INTO (STUDENT);
DELETE FILE (STDBASE) INTO (STUDENT);
GO TO DEL;
END_DEL: CLOSE FILE (STDBASE);
OPEN FILE (STDBASE) SEQUENTIAL BUFFERED OUTPUT;
READ_FS: READ FILE (STDTMP) INTO (STUDENT);
CALL TERMINPET (STUDENT.SOCIAL_SECURITY);
WRITE_FILE (STDBASE) FROM (STUDENT) KEYS FROM
(STUDENT.SOCIAL_SECURITY);
GO TO READ_FS;
END_AD: CLOSE FILE (STDTMP);
CLOSE FILE (STDBASE);
END;
END;
ELSE IF INSTRUCTION = 'CHANGE' THEN DO;
  STATUS = 0;
  IF INSTRUCTION = 1 THEN DO;
    CALL TERMPUT('');
    CALL TERMPUT('WHAT IS THE SOCIAL SECURITY NUMBER OF THE');
    CALL TERMPUT('STUDENT WHOSE RECORD IS TO BE CHANGED?');
    CALL TERMPUT('INPUT THE NUMBER AS A 9 DIGIT INTEGER?');
    CALL TERMPUT('I.E., OMIT DASHES OR SPACES?');
  END;
  ELSE CALL TERMPUT('SOCIAL SECURITY NUMBER=?');
  CALL TGET(STUDENT,SOCIAL_SECURITY);
  OPEN FILE (STDBASE) DIRECT UPDATE;
  READ FILE (STDBASE) INTO (STUDENT) KEY
  (STUDENT,SOCIAL_SECURITY);
  OPEN FILE (COREDBS) DIRECT INPUT;
  READ FILE (COREDBS) INTO (COURSES_REQUIRED)
  KEY (STUDENT,MAJOR);
  CLOSE FILE (COREDBS);
  IF INSTRUCTION = 1 THEN DO;
    CALL TERMPUT('');
    CALL TERMPUT('INPUT THE CODE CORRESPONDING TO THE AREA');
    CALL TERMPUT('OF THE STUDENTS RECORD YOU WISH TO CHANGE?');
    CALL TERMPUT('TO EXIT THE CHANGE LOOP--ENTER 0');
    CALL TERMPUT('1 - NAME');
    CALL TERMPUT('2 - MAJOR');
    CALL TERMPUT('3 - ADVISORS NAME');
    CALL TERMPUT('4 - SPECIFIC COURSE STATUS ');
    CALL TERMPUT('5 - GENERAL AREA CREDIT HOURS');
    CALL TERMPUT('0 - NO CHANGE');
    CALL TERMPUT('NOTE THAT THE SOCIAL SECURITY NUMBER CAN NOT');
    CALL TERMPUT('BE CHANGED');
  END;
END;
THE INSTRUCTIONS TO BE PRINTED UPON REQUEST ARE
SUFFICIENT TO EXPLAIN THIS SECTION OF THE PROGRAM.

GET_CODE: CALL TERMPUT('CHANGE CODE=*');
CALL TGET(TEMP);
CODE = BIN(TEMP);
IF CODE = 1 THEN DO;
CALL TERMPUT('STUDENT NAME=*');
CALL TGET(STUDENT.NAME);
END;
ELSE IF CODE = 2 THEN DO;
IF INSTRUCT = 1 THEN DO;
CALL TERMPUT('INPUT THE STUDENT NAME=');
CALL TERMPUT('FOLLOWING FORMAT: AAAABBBBBXX-XX');
CALL TERMPUT('WHERE AAAA STANDS FOR THE LEFT');
CALL TERMPUT('JUSTIFIED 3-4 LETTER ACRONYM FOR THE');
CALL TERMPUT('COURSE, BBBBB STANDS FOR 6 SPACES.');
CALL TERMPUT('XX-XX STANDS FOR THE YEARS OF THE');
CALL TERMPUT('GOVERNING CATALOG, FOR EXAMPLE:');
CALL TERMPUT(' ITEMS 76-77');
END;
CALL TERMPUT('MAJOR=*');
CALL TGET(STUDENT.MAJOR);
END;
ELSE IF CODE = 3 THEN DO;
CALL TERMPUT('ADVISORS NAME-*');
CALL TGET(STUDENT.ADVISOR);
END;
ELSE IF CODE = 4 THEN DO;
CALL INPUT_COURSES(STUDENT);
STATUS = 1;
END;
ELSE IF CODE = 5 THEN DO;
CALL INPUT_HOURS(STUDENT);
STATUS = 1;
END;
ELSE IF CODE = 0 THEN GO TO END_CHANGE;
ELSE CALL TERMPUT('ERROR--RE-ENTER CHANGE CODE*');
GO TO GET_CODE;
END_CHANGE: IF STATUS = 1 THEN
CALL UPDATE_STATUS(STUDENT);
WRITE FILE(STDBASE) FROM(STUDENT) KEY
(STUDENT,SOCIAL_SECURITY);
CLOSE FILE(STDBASE);
END;
ELSE CALL TERMPUT('ERROR--RE-ENTER UPDATE COMMAND*');
GO TO GET_IN;
EXIT: CLOSE FILE(STDBASE); IF INSTRUCT = 1 THEN DO:
CALL TERMPUT('*INDICATE BY TYPING BATCH OR TERM WHETHER YOU*')
CALL TERMPUT('*WANT THE FINAL LISTING TO BE PRINTED ON THE*')
CALL TERMPUT('*BATCH PRINTER OR AT THE TERMINAL*')
CALL TERMPUT('*IT IS STRONGLY SUGGESTED THAT IF THERE ARE*')
CALL TERMPUT('*MANY STUDENTS, THAT THE PRINTOUT BE ROUTED*')
CALL TERMPUT('*TO THE BATCH PRINTER*')
END:
ELSE CALL TERMPUT('*BATCH OR TERM=*')

BATTERM: CALL TGET(TMP)
IF TEMP = 'BATCH' THEN VAR = 1
ELSE IF TEMP = 'TERM' THEN VAR = 0
ELSE DO:
CALL TERMPUT('*ERROR--RE-ENTER TERM OR BATCH*')
GO TO BATTERM
END:

/* END OF PROGRAM -- ROUTE THE FINAL PRINTOUT. */
INPUT_COURSES: PROCEDURE (STUDENT);

/* THIS PROCEDURE PROMPTS THE USER TO SUPPLY ANY CHANGES 
IN THE STATUS OF A COURSE IN THE STUDENT RECORD. */

DECLARE 1 STUDENT UNALIGNED,
1 SOCIAL_SECURITY CHARACTER(9),
2 NAME CHARACTER(25),
2 MAJOR CHARACTER(20),
2 ADVISOR CHARACTER(26),
2 COURSES,
3 ENGR_CORE(33) CHARACTER(1),
3 ENVIR_STUDIES_SPEC(8) CHARACTER(1),
3 ADV_ENVIR_STUDIES(3) CHARACTER(1),
3 ENGR_OPTIONS(10) CHARACTER(1),
3 ENVIR_STUDIES_GEN(3) FIXED DECIMAL(3,1),
3 REST_ELECTIVES FIXED DECIMAL(3,1),
3 ELECTIVES FIXED DECIMAL(3,1),
2 TOTAL_HOURS FIXED DECIMAL(4,1);

IF INSTRUCT = 1 THEN DO:
CALL TERMINPUT('A COURSE STATUS IN A STUDENT RECORD MAY HAVE');
CALL TERMINPUT('ONE OF THE FOLLOWING VALUES');
CALL TERMINPUT('C - COMPLETED FOR CREDIT');
CALL TERMINPUT('T - TAKEN, BUT RECEIVED NO CREDIT, E.G., A');
CALL TERMINPUT('HIGH SCHOOL COURSE EQUIVALENT');
CALL TERMINPUT('E - STUDENT IS ELIGIBLE TO TAKE THE COURSE');
CALL TERMINPUT('B - WHERE B IS ACTUALLY A BLANK, SIGNIFIES');
CALL TERMINPUT('AN UNCOMPLETED, INELIGIBLE COURSE');
END;

IF INSTRUCT = 1 THEN DO:
CALL TERMINPUT('INPUT THE NAME OF EACH COURSE IN THE FORM');
CALL TERMINPUT('AAAA XXX');
CALL TERMINPUT('FOR EXAMPLE');
CALL TERMINPUT('IEMS 420');
CALL TERMINPUT('FOR ENG 310');
CALL TERMINPUT('NOTICE THAT THE ALPHABETIC PART OF THE');
CALL TERMINPUT('NAME IS LEFT JUSTIFIED IN THE EIGHT SPACES');
CALL TERMINPUT('WHILE THE NUMERIC PART IS RIGHT JUSTIFIED.');
END;
CALL TERMPUT('COURSE NAME = ?');
CALL TGET(CNAME);
DO I = 1 TO 33;
   IF STUDENT.COURSES.ENGR_CORE(I) = 'E' THEN
      STUDENT.COURSES.ENGR_CORE(I) = '\'';
END;
DO I = 1 TO 81;
   IF STUDENT.COURSES.ENVIR_STUDIES_SPEC(I) = 'E' THEN
      STUDENT.COURSES.ENVIR_STUDIES_SPEC(I) = '\'';
END;
DO I = 1 TO 101;
   IF STUDENT.COURSES.ENGR_OPTIONS(I) = 'E' THEN
      STUDENT.COURSES.ENGR_OPTIONS(I) = '\'';
END;
NSTAT:
CALL TERMPUT('*NEW STATUS = '?'');
CALL TGET(TEMP);
IF LENGTH(TEMP) /= 1 THEN DO;
CALL TERMPUT('*INPUT ERROR ------');
CALL TERMPUT('*A NEW COURSE STATUS MAY BE ONLY 1 CHARACTER*');
CALL TERMPUT('*RE-ENTER NEW STATUS*');
GO TO NSTAT;
END;
IF TEMP /= 'C' & TEMP /= 'T' & TEMP /= 'E' & TEMP /= '1'
THEN DO;
CALL TERMPUT('*INPUT ERROR ------');
CALL TERMPUT('*NEW STATUS MUST BE EITHER BLANK,T,C,OR,E*');
CALL TERMPUT('*RE-ENTER NEW STATUS*');
GO TO NSTAT;
END;
DO I = 1 TO 331;
IF CNAME = COURSES_REQUIRED.ENG CORE (I).NAME THEN DO;
   STUDENT.COURSES.ENG CORE (I) = TEMP;
   GO TO END_INPUT;
END;
END;
DO I = 1 TO 81;
IF CNAME = COURSES_REQUIRED.ENVIR STUDIES_SPE (I).NAME
THEN DO;
   STUDENT.COURSES.ENVIR STUDIES_SPE (I) = TEMP;
   GO TO END_INPUT;
END;
END;
DO I = 1 TO 31;
IF CNAME = COURSES_REQUIRED.ADV ENVIR STUDIES (I).NAME
THEN DO;
   STUDENT.COURSES.ADV ENVIR STUDIES (I) = TEMP;
   GO TO END_INPUT;
END;
END;
DO I = 1 TO 81;
IF CNAME = COURSES_REQUIRED.ENG.Options (I).NAME
THEN DO;
   STUDENT.COURSES.ENG.Options (I) = TEMP;
   GO TO END_INPUT;
END;
END;
END_INPUT: CALL TERMPUT('*MORE STATUS CHANGES? = YES OR NO.' );
CALL TGET(TEMP);
IF TEMP = 'YES' THEN GO TO CNAM;
END INPUT_COURSES;
INPUT_HOURS: PROCEDURE(STUDENT);

/* THIS PROCEDURE PROMPTS THE USER TO SUPPLY ANY CHANGES IN THE NUMBER OF HOURS COMPLETED IN GENERAL COURSE AREAS. */

DECLARE 1 STUDENT UNALIGNED,
2 SOCIAL_SECURITY CHARACTER(9),
2 NAME CHARACTER(25),
2 MAJOR CHARACTER(20),
2 ADVISOR CHARACTER(26),
2 COURSES,
3 ENGR_CORE(33) CHARACTER(1),
3 ENVIR_STUDIES_SPE(8) CHARACTER(1),
3 ADV_ENVIR_STUDIES(3) CHARACTER(1),
3 ENGR_OPTIONS(10) CHARACTER(1),
3 ENVIR_STUDIES_GEN(3) FIXED DECIMAL(3,1),
3 REST_ELECTIVES FIXED DECIMAL(3,1),
2 ELECTIVES FIXED DECIMAL(3,1),
2 TOTAL_HOURS FIXED DECIMAL(4,1);

LABL1: CALL TERMPUT('INPUT POSITIVE OR NEGATIVE TO INDICATE TYPE OF');
CALL TERMPUT('CHANGE. ');
CALL TGET(TMP);  
IF TEMP = 'POSITIVE' THEN J = 1;
ELSE IF TEMP = 'NEGATIVE' THEN J = 0;
ELSE DO;
   CALL TERMPUT('INPUT ERROR--RE-ENTER. ');
   GO TO LABL1;
END;
IF TEMP = 'NEGATIVE' THEN GO TO NEG1;
/* IF THE CHANGE IN HOURS COMPLETED IS POSITIVE, THEN GO TO POS. OTHERWISE, IF THE CHANGE IN HOURS IS NEGATIVE, THEN GO TO NEG. */

POS:
DO I = 1 TO 31
  TEMP = COURSES_REQUIRED.ENVIR_STUDIES_GEN(I).NAME;
  CALL TERMPUT('HOW MANY HOURS HAVE BEEN COMPLETED IN');
  CALL TERMPUT(TEMP);
  CALL TGET(TEMP);
  STUDENT.COURSES.ENVIR_STUDIES_GEN(I) = DEC(TEMP) +
      STUDENT.COURSES.ENVIR_STUDIES_GEN(I);
END;
  CALL TERMPUT('HOW MANY HOURS OF RESTRICTED ELECTIVES');
  CALL TERMPUT(TEMP);
  CALL TGET(TEMP);
  STUDENT.COURSES.REST_ELECTIVES = DEC(TEMP) +
      STUDENT.COURSES.REST_ELECTIVES;
  CALL TERMPUT('HOW MANY HOURS OF ELECTIVES WERE COMPLETED');
  CALL TGET(TEMP);
  STUDENT.COURSES.ELECTIVES = DEC(TEMP) +
      STUDENT.COURSES.ELECTIVES;
GO TO END_INI;

NEG:
DO I = 1 TO 31
  TEMP = COURSES_REQUIRED.ENVIR_STUDIES_GEN(I).NAME;
  CALL TERMPUT('HOW MANY HOURS HAVE BEEN CHANGED IN');
  CALL TERMPUT(TEMP);
  CALL TGET(TEMP);
  STUDENT.COURSES.ENVIR_STUDIES_GEN(I) = DEC(TEMP) -
      STUDENT.COURSES.ENVIR_STUDIES_GEN(I);
END;
  CALL TERMPUT('HOW MANY HOURS OF RESTRICTED ELECTIVES');
  CALL TERMPUT(TEMP);
  CALL TGET(TEMP);
  STUDENT.COURSES.REST_ELECTIVES = DEC(TEMP) -
      STUDENT.COURSES.REST_ELECTIVES;
  CALL TERMPUT('HOW MANY HOURS OF ELECTIVES WERE CHANGED');
  CALL TGET(TEMP);
  STUDENT.COURSES.ELECTIVES = DEC(TEMP) -
      STUDENT.COURSES.ELECTIVES;
END_INI: END INPUT_HOURS;
UPDATE_STATUS: PROCEDURE (STUDENT) *

DECLARE PRERSAT FIXED DECIMAL (1.0); OPEN FILE (CDBASE) DIRECT INPUT;
DECLARE 1 STUDENT UNALIGNED,
  2 SOCIAL_SECURITY CHARACTER (9),
  2 NAME CHARACTER (25),
  2 MAJOR CHARACTER (20),
  2 ADVISOR CHARACTER (26),
  2 COURSES,
  3 ENGR_Core (33) CHARACTER (1),
  3 ENVIR_STUDIES_SPE (8) CHARACTER (1),
  3 ADV_ENVIR_STUDIES (3) CHARACTER (1),
  3 ENGR_OPTIONS (10) CHARACTER (1),
  3 ENVIR_STUDIES_GEN (3) FIXED DECIMAL (3.1),
  2 REST_ELECTIVES FIXED DECIMAL (3.1),
  2 TOTAL_HOURS FIXED DECIMAL (4.1);

STUDENT.TOTAL_HOURS = 0.0;
DO I = 1 TO 33;
  IF STUDENT.COURSES.ENGR_Core (I) = 'C' THEN
      STUDENT.TOTAL_HOURS = STUDENT.TOTAL_HOURS + COURSES_REQUIRED.ENGR_Core (I) * CREDIT_HOURS;
END;
DO I = 1 TO 8;
  IF STUDENT.COURSES.ENVIR_STUDIES_SPE (I) = 'C' THEN
      STUDENT.TOTAL_HOURS = STUDENT.TOTAL_HOURS + COURSES_REQUIRED.ENVIR_STUDIES_SPE (I) * CREDIT_HOURS;
END;
DO I = 1 TO 3;
  IF STUDENT.COURSES.ADV_ENVIR_STUDIES (I) = 'C' THEN
      STUDENT.TOTAL_HOURS = STUDENT.TOTAL_HOURS + COURSES_REQUIRED.ADV_ENVIR_STUDIES (I) * CREDIT_HOURS;
END;
DO I = 1 TO 10;
  IF STUDENT.COURSES.ENGR_OPTIONS (I) = 'C' THEN
      STUDENT.TOTAL_HOURS = STUDENT.TOTAL_HOURS + COURSES_REQUIRED.ENGR_OPTIONS (I) * CREDIT_HOURS;
END;
STUDENT.TOTAL_HOURS = STUDENT.TOTAL_HOURS + STUDENT.COURSES.REST_ELECTIVES;
STUDENT.TOTAL_HOURS = STUDENT.TOTAL_HOURS + STUDENT.COURSES.ELECTIVES;
/* Test Engineering Core Courses. If the student has not completed a course and the status is not registered as eligible -- 'E', then check to see if the course has any prerequisites. If so, check to see if the student has completed them. If so, change the blank status to 'E'. */

TEST1: DO I = 1 TO 331
  PRERSAT = 01
  IF STUDENT.COURSES.ENGR_CORE(I) = ' ' THEN DO1
    IF COURSES_REQUIRED.ENGR_CORE(I).NAME = ' ' THEN GO TO END_TEST1;
    COURSE.NAME = COURSES_REQUIRED.ENGR_CORE(I).NAME;
    READ FILE(CODBASE) INTO(COURSE) KEY(COURSE.NAME);
    DO IJ = 1 TO 51
      IF COURSE.PREREQUISITES(I) = ' ' THEN GO TO E11;
      DO IK = 1 TO 331
        IF COURSES_REQUIRED.ENGR_CORE(IK).NAME = COURSE.PREREQUISITES(IJ) THEN
          IF STUDENT.COURSES.ENGR_CORE(IK) = 'C' THEN PRERSAT = 11
          ELSE PRERSAT = 01;
        END;
      END;
    END;
    IF PRERSAT = 0 THEN GO TO END_TEST1;
  END;
E11: IF IJ <= 1 THEN STUDENT.COURSES.ENGR_CORE(I) = 'E';
    ELSE IF PRERSAT = 1 THEN
      STUDENT.COURSES.ENGR_CORE(I) = 'E';
END;
END_TEST1: END TEST1;
/* THIS SECTION PROCEEDS IN A MANNER PARALLEL TO THAT OF TEST1,  
   BUT THIS TIME FOR SPECIFIC ENVIRONMENTAL STUDIES COURSES. */

TEST2: DO I = 1 TO 81
     PRERSAT = 0
     IF STUDENT.COURSES.ENVIR_STUDIES_SPE(I) = "" THEN DO
         IF COURSES_REQUIRED.ENVIR_STUDIES_SPE(I).NAME = ""
             THEN GO TO END_TEST2;
         COURSE.NAME = COURSES_REQUIRED.ENVIR_STUDIES_SPE(I).NAME;
         READ FILE(CODEBASE) INTO (COURSE) KEY (COURSE.NAME);
         DO IJ = 1 TO 51
             IF COURSE.PREREQUISITES(IJ) = "" THEN GO TO EL21;
         DO IK = 1 TO 81
             IF COURSES_REQUIRED.ENVIR_STUDIES_SPE(IK).NAME = 
                 COURSE.PREREQUISITES(IJ)
                 THEN IF
                     STUDENT.COURSES.ENVIR_STUDIES_SPE(IK) = "C"
                     | STUDENT.COURSES.ENVIR_STUDIES_SPE(IK) = "T"
                     THEN 
                         PRERSAT = 1;
                     ELSE PRERSAT = 0;
         END;
     IF PRERSAT = 0 THEN GO TO END_TEST2;
     END;
EL21: IF IJ <= 1 THEN STUDENT.COURSES.ENVIR_STUDIES_SPE(I) = "E";
     ELSE IF PRERSAT = 1 THEN 
         STUDENT.COURSES.ENVIR_STUDIES_SPE(I) = "E";
     END;
END_TEST2: END TEST2;
/* THIS SECTION PROCEEDS IN A MANNER PARALLEL TO THAT OF TEST1* AND TEST2, BUT THIS TIME FOR ENGINEERING OPTIONS COURSES. */

TEST3: DO I = 1 TO 8*
PRERSAT = 0!
IF STUDENT.COURSES.ENG options(I) = ' ' THEN DO!
IF COURSES_required.ENG options(I).NAME = ' ' THEN GO TO END TEST3!
COURSE.NAME = COURSES_required.ENG options(I).NAME!
READ FILE(CODBASE) INTO (COURSE) KEY (COURSE.NAME)!
DO IJ = 1 TO 5:
IF COURSE.PREREQUISITES(IJ) = ' ' THEN GO TO EL3!
DO IK = 1 TO 8:
IF COURSES_required.ENG options(IK).NAME = COURSE.PREREQUISITES(IJ) THEN IF
STUDENT.COURSES.ENG options(IK) = 'C'
STUDENT.COURSES.ENG options(IK) = 'T' THEN
PRERSAT = 1!
ELSE PRERSAT = 0!
END!
IF PRERSAT = 0 THEN GO TO END TEST3!
END!
EL3: IF IJ <= 1 THEN STUDENT.COURSES.ENG options(I) = 'E'!
ELSE IF PRERSAT =1 THEN
STUDENT.COURSES.ENG options(I) = 'E'!
END!
END TEST3: END TEST3!
CLOSE FILE(CODBASE)!
END UPDATE_STATUS!
FINISH: CALL BACKUP!
CALL IMESARC(VAR)!
END STUPOT!
Appendix H: General User Instructions
GENERAL USER INSTRUCTIONS

User Instructions

The most attractive feature of the student advisement system is the fact that it is highly user oriented. The user is not required to do any programming. In order to use the system, the user must first have access to a computer terminal which, in turn, accesses F.T.U.'s IBM 360-75. Then, the user must turn on the terminal and hit the return key in order to communicate with the computer. The following "conversation" must then be executed between the user and the computer. Words in all capital letters have been printed by the computer. Words in all small letters have been typed by the user. After every user response is typed, the user must hit the terminal's return key to send the response to the computer.

IKJ54012A ENTER LOGON -
logon
ENTER USERID -
zf0414 size(200)
ENTER PASSWORD -
free

ZF0414 LOGON IN PROGRESS AT 10:17:17 ON NOVEMBER 20, 1976.
READY
t exec $
READY
After the computer responds with READY, the user must type in the name of the program to be run:

- *sadvice* - student advisement and enrollment projection program
- *coupdt* - course database update program
- *coreupd* - course requirements database update program
- *stupdt* - student database update program

After the program name is typed, the computer will respond with some messages before initial instructions are printed.

The user should not be concerned with these messages, but should continue, following user instructions for the appropriate program as presented.

Instructions for each of the four interactive programs in the system are presented in Appendices I, J, K, and L. Each user manual is in the form of an example program run. It is suggested that the
the first time through the program the user type INSTRUCT as the
first update command. Issuing this command results in instructions
being printed throughout the run to guide the user. All program ins
structions are straight forward and quite detailed. Subsequent
runs will proceed more quickly if the user omits the INSTRUCT command.
Then, only short, prompting questions will be issued and the user
may refer to the first run instructions or to the appropriate user's
manual.

At the end of each update program, the user is requested to
specify whether a batch computer printout or a terminal printout is
desired. This is up to the user. However, if as in the instance
of the student update program, there are large numbers of records to
be printed, it is suggested that the user request a batch output.

If a batch listing is requested, the user may inquire as to
the status of the job by typing:

   @ o

or by typing:

   st

The @ o command is the most useful. It may result in the following
sequence of responses:

  READY
  @ o
  JOB 839 ZF0414TH,,PL1L(31.6)
  READY
  @ o
  JOB 839 ZF0414TH,,PL1L(23.8)
  READY
  @ o
  JOB 839 ZF0414TH,,PL1L(13.9)
Thus, the job number is provided as well as a progress report. The PLL(XX.X) section indicates the number of CPU seconds remaining. This section may be replaced by XAUE-N where N will represent the number of jobs ahead of this job.

These batch printouts will be routed to the remote 2 printer in Room 113 of the computer center.

Whenever an update program is successfully executed, a copy of the database is written on tape. A successful completion of this copy routine will be printed with the name ZF0414BK. In order to obtain the listing of this copy routine, type:

```
output job number
OUTPUT
rr(2)
READY
```

This series of user commands routes the printout to remote printer 2.

When all programs needed have been run, the user must type "logoff" to end communication with the computer.
**Programmer's Guide**

In case of program additions or changes, type:

```
qed2 program name
```

where "program name" is either SADVISE, STUPDT, COUPDT, or COREUPD.

Using the IBM text editor language, changes may then be made to the program. End the programming changes with:

```
end s
SAVED
READY
```

Then, the program must be recompiled and stored in the partitioned data set called $.LOAD. This is done, simply, by typing:

```
thesub program name r(2)
```

When this job is finished, the new listing may be obtained from the remote 2 printer. If the changes resulted in any syntax errors, the program will not be saved in the load module.

If logical errors should occur, type:

```
list syspnt
```

This command will list the error at the terminal. If the printout is long, hit the attention key and route the printout to remote 2 by typing:

```
print syspnt r(2)
```
Appendix I: User Instructions For SADVISE with Sample Output
READY
sadvise
UTILITY DATA SET NOT FREED, IS NOT ALLOCATED
ATTR-LIST-NAME THOSE NOT FOUND

AFTER EACH COMPUTER ISSUED REQUEST
FOR INPUT, ENTER YOUR RESPONSE AND
HIT THE RETURN KEY ON THE RIGHT HAND
SIDE OF THE TERMINAL'S KEYBOARD
INDICATE WHICH OF THE FOLLOWING OPTIONS
YOU DESIRE BY TYPING EITHER ALL
OR SOME.

ALL - INDICATES THE OPTION TO PROCESS
    ALL OF THE STUDENT RECORDS IN
    THE STUDENT DATABASE.
SOME - INDICATES THE OPTION TO
    PROCESS ONLY SOME OF THE
    STUDENT RECORDS.

all

AND, IF YOU WOULD BE SO KIND, PLEASE
INDICATE WHICH QUARTER IS FORTHCOMING
BY ENTERING EITHER FALL, WINTER, SPRING
OR SUMMER

fall

READY
<table>
<thead>
<tr>
<th></th>
<th>ENGR 101</th>
<th>ENG 101</th>
<th>ENGR 151</th>
<th>SPE 101</th>
<th>MATH 321</th>
<th>HUM 201</th>
<th>ENG 310</th>
<th>ECON 201</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ENGINEERING CORE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>REQUIRED &amp; ELIGIBLE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ADDITIONAL ENVIRONMENTAL STUDIES</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>REQUIRED &amp; ELIGIBLE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ADVANCED ENVIRONMENTAL STUDIES</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ENGINEERING OPTION</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>REQUIRED &amp; ELIGIBLE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>REQUIRED &amp; ELIGIBLE COURSES NEXT QUARTER</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ENGINEERING STUDIES</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GENERAL AREAS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CUL &amp; HIST FNDTS</strong></td>
<td>7.0</td>
<td>9.0</td>
<td>3.0</td>
<td>15.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td><strong>SOCIAL SCIENCES</strong></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td><strong>SCIENCE (EARTH / BIO)</strong></td>
<td>7.0</td>
<td>9.0</td>
<td>3.0</td>
<td>15.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
## Projected Course Enrollment

<table>
<thead>
<tr>
<th>COURSE</th>
<th>FALL</th>
<th>WINTER</th>
<th>SPRING</th>
<th>SUMMER</th>
<th>COURSE</th>
<th>FALL</th>
<th>WINTER</th>
<th>SPRING</th>
<th>SUMMER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEES 401</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ENGR 342</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CEES 402</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ENGR 351</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CEES 411</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ENGR 352</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CEES 412</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ENGR 361</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CEES 414</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ENGR 371</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>COMP 102</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ENGR 421</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>COMP 302</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ENGR 431</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ECON 201</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ENGR 442</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EECS 321</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ENGR 443</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EECS 322</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>MATH 201</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>EECS 341</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IEMS 301</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>EECS 411</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IEMS 414</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EMCS 431</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IEMS 424</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EMCS 432</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IEMS 432</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EMCS 472</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IEMS 434</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ENG 101</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>IEMS 447</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ENG 310</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>IEMS 461</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>ENGR 101</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>MATH 211</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ENGR 103</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>MATH 321</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>ENGR 151</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>MATH 322</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>ENGR 152</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>MATH 323</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>ENGR 211</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>MATH 324</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ENGR 310</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>MATH 331</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ENGR 311</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>MEAS 341</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ENGR 312</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>MEAS 342</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ENGR 320</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>MEAS 351</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ENGR 321</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>MEAS 423</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ENGR 322</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>MEAS 482</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ENGR 323</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>PHYS 344</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ENGR 331</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>PHYS 354</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ENGR 332</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SPE 101</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ENGR 341</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
AFTER EACH COMPUTER ISSUED REQUEST FOR INPUT, ENTER YOUR RESPONSE AND HIT THE RETURN KEY ON THE RIGHT HAND SIDE OF THE TERMINAL'S KEYBOARD. INDICATE WHICH OF THE FOLLOWING OPTIONS YOU DESIRE BY TYPING EITHER "ALL" OR "SOME.

"ALL" INDICATES THE OPTION TO PROCESS ALL OF THE STUDENT RECORDS IN THE STUDENT DATABASE.

"SOME" INDICATES THE OPTION TO PROCESS ONLY SOME OF THE STUDENT RECORDS.

AND, IF YOU WOULD BE SO KIND, PLEASE INDICATE WHICH QUARTER IS FORTHCOMING BY ENTERING EITHER FALL, WINTER, SPRING OR SUMMER.

YOU HAVE OPTED TO PROCESS A FEW STUDENT RECORDS INDIVIDUALLY. PLEASE ENTER THE TOTAL NUMBER OF STUDENTS YOU WISH TO PROCESS. BY THE WAY, THIS NUMBER MUST BE LESS THAN OR EQUAL TO 15.

PLEASE ENTER A SOCIAL SECURITY NUMBER FOR EACH STUDENT. ENTER IN NUMERICAL ORDER. ALSO, ENTER EACH AS A CONTINUOUS 9 DIGIT NUMBER - OMIT DASHES OR SPACES.

263041248

ATTR-LIST-NAME SYS NOT FOUND
ADVANCE PAPER TO TOP OF FORM AND HIT RETURN. WHEN PRINT IS FINISHED REMOVE LISTING AND HIT RETURN.
S.S. NO.: 263041248
NAME: MARILYN SELLERS KHORSANDI
MAJOR: EICS 76-77
TOTAL HOURS REQUIRED: 192.0
TOTAL HOURS COMPLETED: 4.0

ENGINEERING CORE

| ENGR 101 | ENG 101 | SPE 101 | HUM 201 | ECON 201 |

BASIC ENVIRONMENTAL STUDIES

ADVANCED ENVIRONMENTAL STUDIES

ENGINEERING OPTION

| ENGR 151(COR->MATH 211) | ENG 310 | SPE 101 | HUM 201 |

REQUIRED & ELIGIBLE COURSES NEXT QUARTER

| ENGR 151 | ENG 310 | SPE 101 | HUM 201 |

ENVIRONMENTAL STUDIES

<table>
<thead>
<tr>
<th>CUL &amp; HIST FNDS</th>
<th>SOCIAL SCIENCES</th>
<th>SCIENCE(earth / bio)</th>
<th>RESTRICTED ELECTIVES</th>
<th>ELECTIVES</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUIRED 7.0</td>
<td>SOCIAL SCIENCES 9.0</td>
<td>SCIENCE(EARTH / BIO) 3.0</td>
<td>15.0</td>
<td>0.0</td>
</tr>
<tr>
<td>COMPLETED 3.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>REMAINING 4.0</td>
<td>9.0</td>
<td>3.0</td>
<td>15.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Appendix J: User Instructions for COUPDT with Sample Output
THIS PROGRAM, IN CONJUNCTION WITH CORRECT USER INPUT, 
UPDATES THE COURSE DATABASE. 
IF YOU NEED OPERATING INSTRUCTIONS, TYPE 
INSTRUCT. 
PROVIDE AN UPDATE COMMAND.

WHEN THE PROGRAM ASKS FOR AN 
UPDATE COMMAND, SUPPLY ONE OF 
THE FOLLOWING COMMANDS AND THE 
ASSOCIATED ACTION WILL BE EXECUTED. 
INSTRUCT - INSTRUCTIONS WILL BE 
PRINTED THROUGHOUT THE 
PROGRAM. 
ADD - ADDS A COURSE RECORD 
DELETE - DELETES A COURSE 
RECORD. 
PRINT - PRINTS A RECORD. 
CHANGE - PROVIDES THE USER WITH 
THE CAPABILITY TO CHANGE 
ANY PART OF A COURSE 
RECORD. 
STOP - STOPS THE PROGRAM AND 
PROVIDES THE USER WITH 
A FINAL LISTING OF ALL 
COURSE RECORDS IN THE 
COURSES DATABASE. 
PROVIDE AN UPDATE COMMAND.

PLEASE INPUT THE TOTAL NUMBER OF 
COURSES YOU WANT TO ADD. 

1
ENTER COURSE NAMES IN THE FORM
AAAA XXX
FOR EXAMPLE: IEMS 431
IN OTHER WORDS THE ENTIRE COURSE NAME
MUST OCCUPY 8 SPACES.
ALSO, BE SURE TO ENTER THE COURSES IN
ALPHABETICAL AND NUMERICAL ORDER.
FOR EXAMPLE:
IEMS 433
IEMS 434
MATH 333
MEAS 215
MEAS 314
COURSE NAME =
eng 103
ARE THERE ANY COREQUISITES FOR THIS COURSE?
no
ARE THERE ANY PREREQUISITES FOR THIS COURSE?
yes
HOW MANY PREREQUISITES ARE THERE?
1
LIST THE PREREQUISITES
eng 101
WILL THIS COURSE BE OFFERED IN THE FALL?
yes
WILL THIS COURSE BE OFFERED IN THE WINTER?
yes
WILL THIS COURSE BE OFFERED IN THE SPRING?
yes
WILL THIS COURSE BE OFFERED IN THE SUMMER?
yes
HOW MANY CREDIT HOURS IS THIS COURSE WORTH?

3

PROVIDE A DESCRIPTIVE NAME FOR THIS COURSE IN 20 CHARACTERS OR LESS.

freshman english

CEES 401
CEES 402
CEES 411
CEES 412
CEES 414
COMP 102
COMP 302
ECON 201
EECS 321
EECS 322
EECS 341
EECS 411
EMCS 431
EMCS 432
EMCS 472
ENG 101
ENG 103
ENG 310
ENGR 101
ENGR 103
ENGR 151
ENGR 152
ENGR 211
ENGR 310
ENGR 311
ENGR 312
ENGR 320
ENGR 321
ENGR 322
ENGR 323
ENGR 331
ENGR 332
ENGR 341
ENGR 342
ENGR 351
ENGR 352
ENGR 361
ENGR 371
ENGR 421  
ENGR 431  
ENGR 442  
ENGR 443  
HUM 201  
IEMS 301  
IEMS 414  
IEMS 424  
IEMS 432  
IEMS 434  
IEMS 447  
IEMS 461  
MATH 211  
MATH 321  
MATH 322  
MATH 323  
MATH 324  
MATH 331  
MEAS 341  
MEAS 342  
MEAS 351  
MEAS 423  
MEAS 482  
PHYS 344  
PHYS 354  
SPE 101  
PROVIDE AN UPDATE COMMAND.

delete

HOW MANY COURSES DO YOU WISH TO DELETE?

1

WHAT IS THE NAME (IN EXACTLY 8 CHARACTERS) OF THE COURSE TO BE DELETED?

eng 103

PROVIDE AN UPDATE COMMAND.

print
INPUT THE COURSE NAME YOU WANT PRINTED AS 8 CHARACTERS: AAAA XXX
WHERE AAAA STANDS FOR THE ACCEPTED ACRONYM FOR THE COURSE, WHILE XXX STANDS FOR THE COURSE NUMBER. FOR EXAMPLE:

<table>
<thead>
<tr>
<th>eng</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>CO-1</td>
</tr>
<tr>
<td>ENG 101</td>
<td></td>
</tr>
</tbody>
</table>

PROVIDE AN UPDATE COMMAND.

print

INPUT THE COURSE NAME YOU WANT PRINTED AS 8 CHARACTERS: AAAA XXX WHERE AAAA STANDS FOR THE ACCEPTED ACRONYM FOR THE COURSE, WHILE XXX STANDS FOR THE COURSE NUMBER. FOR EXAMPLE:

<table>
<thead>
<tr>
<th>math</th>
<th>321</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>CO-1</td>
</tr>
<tr>
<td>MATH 321</td>
<td>MATH 211</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PROVIDE AN UPDATE COMMAND.
HOW MANY COURSES DO YOU WISH TO CHANGE?

3

WHAT IS THE NAME OF THE COURSE TO BE CHANGED?

engr 103

PLEASE INPUT THE AREA OF THE COURSE INFORMATION YOU WISH TO CHANGE BY ONE OF THE FOLLOWING APPROPRIATE CODES:

1 - COREQUISITES
2 - PREREQUISITES
3 - SEASONS OFFERED
4 - CREDIT HOURS
5 - DESCRIPTION

CHANGE CODE=

1

INPUT ALL THREE COREQUISITES. IF, FOR INSTANCE, THERE IS ONLY ONE COREQUISITE, THEN HIT RETURN FOR THE LAST TWO

c.i.

ARE THERE ANY MORE CHANGES TO THIS COURSE? ANSWER YES OR NO

yes

CHANGE CODE=

2

INPUT ALL FIVE PREREQUISITES. AGAIN, IF, FOR INSTANCE, THERE ARE ONLY TWO PREREQUISITES, THEN HIT RETURN FOR THE LAST THREE.

ARE THERE ANY MORE CHANGES TO THIS COURSE? ANSWER YES OR NO

no
WHAT IS THE NAME OF THE COURSE TO BE CHANGED?

engr 443

PLEASE INPUT THE AREA OF THE COURSE INFORMATION YOU WISH TO CHANGE BY ONE OF THE FOLLOWING APPROPRIATE CODES:

1 - COREQUISITES
2 - PREREQUISITES
3 - SEASONS OFFERED
4 - CREDIT HOURS
5 - DESCRIPTION

CHANGE CODE=

1

INPUT ALL THREE COREQUISITES. IF, FOR INSTANCE, THERE IS ONLY ONE COREQUISITE, THEN HIT RETURN FOR THE LAST TWO

ARE THERE ANY MORE CHANGES TO THIS COURSE? ANSWER YES OR NO

no

WHAT IS THE NAME OF THE COURSE TO BE CHANGED?

phys 354

PLEASE INPUT THE AREA OF THE COURSE INFORMATION YOU WISH TO CHANGE BY ONE OF THE FOLLOWING APPROPRIATE CODES:

1 - COREQUISITE
2 - PREREQUISITES
3 - SEASONS OFFERED
4 - CREDIT HOURS
5 - DESCRIPTION

CHANGE CODE=

2

INPUT ALL FIVE PREREQUISITES. AGAIN, IF, FOR INSTANCE, THERE ARE ONLY TWO PREREQUISITES, THEN HIT RETURN FOR THE LAST THREE.
Are there any more changes to this course? Answer yes or no

No

Provide an update command.

Stop

To route the final printout to batch, type batch;
To route the final printout to the terminal, type term.
Batch or terminal printout--

Batch

Ready
Appendix K: User Instructions for COREUPD With Sample Output
ATTR-LIST-NAME THIS NOT FOUND

THIS PROGRAM, IN CONJUNCTION WITH CORRECT USER INPUT, UPDATES THE COURSE REQUIREMENTS DATABASE. IF INSTRUCTIONS ARE NEEDED, RESPOND TO THE REQUEST FOR AN UPDATE COMMAND WITH INSTRUCT. PROVIDE AN UPDATE COMMAND.

instruct

WHEN THE PROGRAM ASKS FOR AN UPDATE COMMAND, SUPPLY ONE OF THE FOLLOWING COMMANDS AND THE ASSOCIATED ACTION WILL BE EXECUTED. INSTRUCT - INSTRUCTIONS WILL BE PRINTED THROUGHOUT THE PROGRAM.

ADD - ADDS A MAJOR RECORD
DELETE - DELETES A MAJOR RECORD.
PRINT - PRINTS A RECORD.
CHANGE - PROVIDES THE USER WITH THE CAPABILITY TO CHANGE ANY PART OF A MAJOR RECORD.
STOP - STOPS THE PROGRAM AND PROVIDES THE USER WITH A FINAL LISTING OF ALL MAJOR RECORDS IN THE COURSE REQMTS DATABASE.

PROVIDE AN UPDATE COMMAND.

add

INPUT THE TOTAL NUMBER OF MAJORS YOU WISH TO ADD - ANSWER MUST BE AN INTEGER NUMBER

1
ENTER THE NAME OF THE NEW MAJOR.
THIS NAME MUST BE ENTERED IN THE FOLLOWING
FORMAT - AAAABBBBBXX-XX. IE., A FOUR LETTER
ABBREVIATION OF THE MAJORS NAME, FOLLOWED BY
6 BLANKS (THE BS) AND THEN THE CATALOG YEARS
WITH A DASH IN BETWEEN THE TWO, DATES
FOR EXAMPLE - EMCS 77-78
ENTER THE NAME OF THE NEW MAJOR

iems 77-78
LIST THE ENGINEERING OPTION COURSES
WHAT IS THE COURSE NAME?

iems 301
HOW MANY HOURS IS THIS COURSE WORTH?
4
WHAT IS THE COURSE NAME?

iems 424
HOW MANY HOURS IS THIS COURSE WORTH?
3
WHAT IS THE COURSE NAME?

iems 461
HOW MANY HOURS IS THIS COURSE WORTH?
3
WHAT IS THE COURSE NAME?

iems 432
HOW MANY HOURS IS THIS COURSE WORTH?
3
WHAT IS THE COURSE NAME?

iems 447
HOW MANY HOURS IS THIS COURSE WORTH?
3
WHAT IS THE COURSE NAME?
lems 434
HOW MANY HOURS IS THIS COURSE WORTH?
3
WHAT IS THE COURSE NAME?
lems 414
HOW MANY HOURS IS THIS COURSE WORTH?
4
WHAT IS THE COURSE NAME?

LIST THE GENERAL ENVIRONMENTAL STUDIES COURSES ALONG WITH LOWER AND UPPER LIMIT OF HOURS REQUIRED FOR EACH, FOR EXAMPLE:
CUL & HIST FNDTS
17
19
WHAT IS THE COURSE-AREA NAME?
cul & hist fndts
WHAT IS THE LOWER LIMIT OF HOURS REQUIRED?
7
WHAT IS THE UPPER LIMIT OF HOURS REQUIRED?
8
WHAT IS THE COURSE-AREA NAME?
social sciences
WHAT IS THE LOWER LIMIT OF HOURS REQUIRED? 9
WHAT IS THE UPPER LIMIT OF HOURS REQUIRED? 10
WHAT IS THE COURSE-AREA NAME?
science(earth / bio)
WHAT IS THE LOWER LIMIT OF HOURS REQUIRED? 3
WHAT IS THE UPPER LIMIT OF HOURS REQUIRED? 4
HOW MANY HOURS OF RESTRICTED ELECTIVES ARE REQUIRED? 16
HOW MANY HOURS OF ELECTIVES ARE REQUIRED? 0
HOW MANY HOURS ARE REQUIRED TO GRADUATE? 192
PROVIDE AN UPDATE COMMAND.
print
**INPUT THE MAJOR NAME YOU WANT PRINTED AS 15 CHARACTERS:**

AAABBBBBBBXX-XX

**WHERE AAAA STANDS FOR THE ACCEPTED ACRONYM FOR THE MAJOR. B STANDS FOR A BLANK. XX-XX STANDS FOR THE GOVERNING CATALOG YEARS. FOR EXAMPLE:** IEMS 76-77

**Iems 77-78**

**IEMS 77-78**

**ENGINEERING CORE COURSES**

<table>
<thead>
<tr>
<th>Course</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP 302</td>
<td>3</td>
</tr>
<tr>
<td>ENGR 151</td>
<td>3</td>
</tr>
<tr>
<td>MATH 211</td>
<td>3</td>
</tr>
<tr>
<td>MATH 323</td>
<td>4</td>
</tr>
<tr>
<td>ENGR 310</td>
<td>4</td>
</tr>
<tr>
<td>ENGR 320</td>
<td>4</td>
</tr>
<tr>
<td>ENGR 323</td>
<td>4</td>
</tr>
<tr>
<td>ENGR 341</td>
<td>3</td>
</tr>
<tr>
<td>ENGR 352</td>
<td>3</td>
</tr>
<tr>
<td>PHYS 344</td>
<td>3</td>
</tr>
<tr>
<td>ENG 310</td>
<td>3</td>
</tr>
</tbody>
</table>

**SPECIFIC ENVIRONMENTAL STUDIES COURSES**

<table>
<thead>
<tr>
<th>Course</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENG 101</td>
<td>4</td>
</tr>
<tr>
<td>SPE 101</td>
<td>3</td>
</tr>
<tr>
<td>ENGR 201</td>
<td>0</td>
</tr>
<tr>
<td>HUM 201</td>
<td>4</td>
</tr>
</tbody>
</table>

**ADVANCED ENVIRONMENTAL STUDIES AREAS**

<table>
<thead>
<tr>
<th>Course</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BADM</td>
<td>3</td>
</tr>
<tr>
<td>ED</td>
<td>3</td>
</tr>
</tbody>
</table>

**ENGINEERING OPTIONS COURSES**

<table>
<thead>
<tr>
<th>Course</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEMS 301</td>
<td>4</td>
</tr>
<tr>
<td>IEMS 424</td>
<td>3</td>
</tr>
<tr>
<td>IEMS 432</td>
<td>3</td>
</tr>
<tr>
<td>IEMS 447</td>
<td>3</td>
</tr>
<tr>
<td>IEMS 414</td>
<td>4</td>
</tr>
</tbody>
</table>

**GENERAL ENVIRONMENTAL STUDIES AREAS**

<table>
<thead>
<tr>
<th>Course</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUL &amp; HIST FNDS</td>
<td>7.0</td>
</tr>
<tr>
<td>SOCIAL SCIENCES</td>
<td>9.0</td>
</tr>
<tr>
<td>SCIENCE(EARTH / BIO)</td>
<td>3.0</td>
</tr>
</tbody>
</table>

**HOURS IN RESTRICTED ELECTIVES = 16.0**

**HOURS IN ELECTIVES = 0.0**

**TOTAL HOURS REQUIRED = 192.0**

**PROVIDE AN UPDATE COMMAND.**

`delete`
HOW MANY MAJORS DO YOU WISH TO DELETE?
1

WHAT IS THE NAME AND YEARS OF THE MAJOR?
Items 77-78

PROVIDE AN UPDATE COMMAND.

change

HOW MANY MAJORS DO YOU WISH TO CHANGE?
1

WHAT IS THE NAME AND THE YEAR OF THE MAJOR YOU WISH TO CHANGE?
Items 76-77

INPUT THE AREA OF THE COURSES REQUIRED FOR THIS MAJOR YOU WISH TO CHANGE ACCORDING TO ONE OF THE FOLLOWING APPROPRIATE CODES:

0 - NO CHANGES
1 - ENGINEERING CORE
2 - ENVIRONMENTAL STUDIES SPECIFIC
3 - ADVANCED ENVIRONMENTAL STUDIES
4 - ENGINEERING OPTION COURSES
5 - ENVIRONMENTAL STUDIES--GENERAL
6 - RESTRICTED ELECTIVES--HOURS ONLY
7 - ELECTIVES--HOURS ONLY
8 - TOTAL HOURS REQUIRED TO GRADUATE

CHANGE CODE=
1

OLD ENGINEERING CORE COURSE NAME=
comp 302

WHAT IS THE NEW NAME?
comp 302
HOW MANY HOURS IS THIS COURSE WORTH?
3
ARE THERE ANY MORE CHANGES TO THIS COURSE?—
YES OR NO
no
PROVIDE AN UPDATE COMMAND.
stop
TO ROUTE THE FINAL PRINTOUT TO BATCH
TYPE BATCH; TO ROUTE THE FINAL PRINTOUT TO THE TERMINAL, TYPE TERM.
BATCH OR TERMINAL PRINTOUT--
batch
READY
<table>
<thead>
<tr>
<th>COURSE</th>
<th>TRANSCRIPT</th>
<th>COURSE</th>
<th>TRANSCRIPT</th>
<th>COURSE</th>
<th>TRANSCRIPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP 302</td>
<td>3</td>
<td>ENGR 101</td>
<td>3</td>
<td>ENGR 103</td>
<td>4</td>
</tr>
<tr>
<td>ENGR 151</td>
<td>3</td>
<td>ENGR 152</td>
<td>3</td>
<td>ENGR 211</td>
<td>4</td>
</tr>
<tr>
<td>MATH 211</td>
<td>3</td>
<td>MATH 321</td>
<td>4</td>
<td>MATH 322</td>
<td>4</td>
</tr>
<tr>
<td>MATH 323</td>
<td>4</td>
<td>MATH 324</td>
<td>4</td>
<td>MATH 331</td>
<td>4</td>
</tr>
<tr>
<td>ENGR 310</td>
<td>4</td>
<td>ENGR 311</td>
<td>4</td>
<td>ENGR 312</td>
<td>5</td>
</tr>
<tr>
<td>ENGR 320</td>
<td>4</td>
<td>ENGR 321</td>
<td>4</td>
<td>ENGR 322</td>
<td>4</td>
</tr>
<tr>
<td>ENGR 323</td>
<td>4</td>
<td>ENGR 331</td>
<td>3</td>
<td>ENGR 332</td>
<td>4</td>
</tr>
<tr>
<td>ENGR 341</td>
<td>3</td>
<td>ENGR 342</td>
<td>3</td>
<td>ENGR 351</td>
<td>3</td>
</tr>
<tr>
<td>ENGR 352</td>
<td>3</td>
<td>ENGR 361</td>
<td>3</td>
<td>ENGR 371</td>
<td>3</td>
</tr>
<tr>
<td>PHYS 344</td>
<td>3</td>
<td>PHYS 354</td>
<td>3</td>
<td>ENGR 431</td>
<td>3</td>
</tr>
<tr>
<td>ENGR 310</td>
<td>3</td>
<td>ENGR 442</td>
<td>3</td>
<td>ENGR 443</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COURSE</th>
<th>TRANSCRIPT</th>
<th>COURSE</th>
<th>TRANSCRIPT</th>
<th>COURSE</th>
<th>TRANSCRIPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENG 101</td>
<td>4</td>
<td>SPE 101</td>
<td>3</td>
<td>HUM 201</td>
<td>4</td>
</tr>
<tr>
<td>ECON 201</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COURSE</th>
<th>TRANSCRIPT</th>
<th>COURSE</th>
<th>TRANSCRIPT</th>
<th>COURSE</th>
<th>TRANSCRIPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>BADM 3</td>
<td>ENGR 3</td>
<td>ED 3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COURSE</th>
<th>TRANSCRIPT</th>
<th>COURSE</th>
<th>TRANSCRIPT</th>
<th>COURSE</th>
<th>TRANSCRIPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEMS 301</td>
<td>4</td>
<td>IEMS 424</td>
<td>3</td>
<td>IEMS 461</td>
<td>3</td>
</tr>
<tr>
<td>IEMS 432</td>
<td>3</td>
<td>IEMS 447</td>
<td>3</td>
<td>IEMS 434</td>
<td>3</td>
</tr>
<tr>
<td>IEMS 414</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COURSE</th>
<th>TRANSCRIPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUL &amp; HIST FNDTS</td>
<td>7.0 8.0</td>
</tr>
<tr>
<td>SOCIAL SCIENCES</td>
<td>9.0 10.0</td>
</tr>
<tr>
<td>SCIENCE(EARTH / BIO)</td>
<td>3.0 4.0</td>
</tr>
</tbody>
</table>

HOURS IN RESTRICTED ELECTIVES = 16.0
HOURS IN ELECTIVES = 0.0
TOTAL HOURS REQUIRED = 192.0
Appendix I: User Instructions for STUPDT With Sample Output
THIS PROGRAM IN CONJUNCTION WITH CORRECT USER INPUT, UPDATES THE STUDENT DATABASE. IF YOU NEED OPERATING INSTRUCTIONS, TYPE INSTRUCT. PROVIDE AN UPDATE COMMAND

instruct

BY INPUTTING ONE OF THE FOLLOWING COMMANDS THE ASSOCIATED UPDATE ACTION IS EXECUTED:

PRINT - PRINTS STUDENT RECORD AFTER SPECIFYING SOCIAL SECURITY NUMBER
INSTRUCT - PRINTS EXTRA INSTRUCTION THROUGHOUT THE PROGRAM.
DELETE- DELETES SPECIFIED RECORD
ADD - ADDS A STUDENT RECORD
CHANGE - PROVIDES USER WITH CAPABILITY TO CHANGE ANY PART OF A STUDENT RECORD
STOP - ENDS PROGRAM - GENERATES A COMPLETE LISTING OF THE DATABASE TO ENSURE THE USER THAT INPUTS WERE CORRECTLY RECORDED

PROVIDE AN UPDATE COMMAND

add

INPUT THE NEW STUDENTS SOCIAL SECURITY NUMBER AS A 9 DIGIT NUMBER--I.E., WITHOUT DASHES.

123456789

INPUT THE NEW STUDENTS NAME IN 25 CHARACTERS OR LESS.

john doe jr.
INPUT THE NEW STUDENTS NAME IN 25 CHARACTERS OR LESS.

johndoe jr.

INPUT THE STUDENTS MAJOR IN THE FOLLOWING FORMAT CONSISTING OF 15 SPACES:

AAAABBBBBXX-XX

WHERE AAAA REPRESENTS THE ACCEPTED 4 LETTER ACCRONYM OF THE MAJOR, E.G., IEMS, EECS, ETC.
AND, WHERE BBBBBB REPRESENTS 6 SPACES.
AND, WHERE XX-XX REPRESENTS THE YEARS OF THE GOVERNING CATALOG, E.G., 76-77

AN EXAMPLE IS: IEMS 76-77

IEMS 76-77

INPUT THE STUDENTS ADVISORS NAME IN 26 CHARACTERS OR LESS

dr. christian s. bauer

A COURSE STATUS IN A STUDENT RECORD MAY HAVE ONE OF THE FOLLOWING VALUES:

C - COMPLETED FOR CREDIT
T - TAKEN, BUT RECEIVED NO CREDIT, E.G., A HIGH SCHOOL COURSE EQUIVALENT
E - STUDENT IS ELIGIBLE TO TAKE THE COURSE
B - WHERE B IS ACTUALLY A BLANK, SIGNIFIES AN UNCOMPLETED, INELIGIBLE COURSE

INPUT THE NAME OF EACH COURSE IN THE FORM:

AAAA XXX

FOR EXAMPLE:

IEMS 420
OR ENG 310

NOTICE THAT THE ALPHABETIC PART OF THE NAME IS LEFT JUSTIFIED IN THE EIGHT SPACES
WHILE THE NUMERIC PART IS RIGHT JUSTIFIED.

COURSE NAME = ?
math 211
NEW STATUS = ?
c
MORE STATUS CHANGES? - YES OR NO.
yes
COURSE NAME = ?
math 321
NEW STATUS = ?
c
MORE STATUS CHANGES? - YES OR NO.
no
INPUT POSITIVE OR NEGATIVE TO INDICATE TYPE OF CHANGE.
positive
HOW MANY HOURS HAVE BEEN COMPLETED IN CUL & HIST FNCTS
3
HOW MANY HOURS HAVE BEEN COMPLETED IN SOCIAL SCIENCES
0
HOW MANY HOURS HAVE BEEN COMPLETED IN SCIENCE(EARTH / BIO)
0
HOW MANY HOURS OF RESTRICTED ELECTIVES WERE COMPLETED=
0
HOW MANY HOURS OF ELECTIVES WERE COMPLETED=
0
BY INPUTTING ONE OF THE FOLLOWING COMMANDS THE ASSOCIATED UPDATE ACTION IS EXECUTED:

**PRINT** - PRINTS STUDENT RECORD AFTER SPECIFYING SOCIAL SECURITY NUMBER

**INSTRUCT** - PRINTS EXTRA INSTRUCTION THROUGHOUT THE PROGRAM.

**DELETE** - DELETES SPECIFIED RECORD

**ADD** - ADDS A STUDENT RECORD

**CHANGE** - PROVIDES USER WITH CAPABILITY TO CHANGE ANY PART OF A STUDENT RECORD

**STOP** - ENDS PROGRAM - GENERATES A COMPLETE LISTING OF THE DATABASE TO ENSURE THE USER THAT INPUTS WERE CORRECTLYRecorded

PROVIDE AN UPDATE COMMAND

**change**

WHAT IS THE SOCIAL SECURITY NUMBER OF THE STUDENT WHOSE RECORD IS TO BE CHANGED=

INPUT THE NUMBER AS A 9 DIGIT INTEGER I.E., OMIT DASHES OR SPACES

**123456789**

INPUT THE CODE CORRESPONDING TO THE AREA OF THE STUDENT'S RECORD YOU WISH TO CHANGE:

TO EXIT THE CHANGE LOOP--ENTER 0

1 - NAME
2 - MAJOR
3 - ADVISORS NAME
4 - SPECIFIC COURSE STATUS
5 - GENERAL AREA CREDIT HOURS
0 - NO CHANGE

NOTE THAT THE SOCIAL SECURITY NUMBER CAN NOT BE CHANGED
CHANGE CODE = 
3

ADVISORS NAME =

dr. harold i. klee

CHANGE CODE =

4

A COURSE STATUS IN A STUDENT RECORD MAY HAVE ONE OF THE FOLLOWING VALUES:
C - COMPLETED FOR CREDIT
T - TAKEN, BUT RECEIVED NO CREDIT, E.G., A HIGH SCHOOL COURSE EQUIVALENT
E - STUDENT IS ELIGIBLE TO TAKE THE COURSE
B - WHERE B IS ACTUALLY A BLANK, SIGNIFIES AN UNCOMPLETED, INELIGIBLE COURSE

INPUT THE NAME OF EACH COURSE IN THE FORM:
AAAA XXX

FOR EXAMPLE:
IEMS 420
OR ENG 310

NOTICE THAT THE ALPHABETIC PART OF THE NAME IS LEFT JUSTIFIED IN THE EIGHT SPACES WHILE THE NUMERIC PART IS RIGHT JUSTIFIED.

COURSE NAME = ?

comp 302

NEW STATUS = ?

c

MORE STATUS CHANGES? - YES OR NO.

no

CHANGE CODE =

0

PROVIDE AN UPDATE COMMAND
WHAT IS THE STUDENT'S SOCIAL SECURITY NUMBER?

123456789

123456789 JOHN DOE JR. IEMS 76-77 DR. HAROLD L. KLEE
CEEE ECCE E EEEE E E 3.0 0.0 0.0 0.0 0.0 0.0

BY INPUTTING ONE OF THE FOLLOWING COMMANDS
THE ASSOCIATED UPDATE ACTION IS EXECUTED:

PRINT - PRINTS STUDENT RECORD AFTER
SPECIFYING SOCIAL SECURITY NUMBER
INSTRUCT - PRINTS EXTRA INSTRUCTION
THROUGHOUT THE PROGRAM.
DELETE - DELETES SPECIFIED RECORD
ADD - ADDS A STUDENT RECORD
CHANGE - PROVIDES USER WITH CAPABILITY TO
CHANGE ANY PART OF A STUDENT
RECORD
STOP - ENDS PROGRAM - GENERATES A COMPLETE
LISTING OF THE DATABASE TO ENSURE
THE USER THAT INPUTS WERE
CORRECTLY RECORDED

PROVIDE AN UPDATE COMMAND
WHAT IS THE STUDENTS SOCIAL SECURITY NUMBER?

123456789

BY INPUTTING ONE OF THE FOLLOWING COMMANDS THE ASSOCIATED UPDATE ACTION IS EXECUTED:
PRINT - PRINTS STUDENT RECORD AFTER SPECIFYING SOCIAL SECURITY NUMBER
INSTRUCT - PRINTS EXTRA INSTRUCTION THROUGHOUT THE PROGRAM
DELETE- DELETES SPECIFIED RECORD
ADD - ADDS A STUDENT RECORD
CHANGE - PROVIDES USER WITH CAPABILITY TO CHANGE ANY PART OF A STUDENT RECORD
STOP - ENDS PROGRAM - GENERATES A COMPLETE LISTING OF THE DATABASE TO ENSURE THE USER THAT INPUTS WERE CORRECTLY RECORDED

PROVIDE AN UPDATE COMMAND

stop

INDICATE BY TYPING BATCH OR TERM WHETHER YOU WANT THE FINAL LISTING TO BE PRINTED ON THE BATCH PRINTER OR AT THE TERMINAL. IT IS STRONGLY SUGGESTED THAT IF THERE ARE MANY STUDENTS, THAT THE PRINTOUT BE ROUTED TO THE BATCH PRINTER.

batch

READY
<table>
<thead>
<tr>
<th>Name</th>
<th>Program</th>
<th>Years</th>
<th>DOIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MARILYN SELLERS</td>
<td>EMCS</td>
<td>76-77</td>
<td>DR. CHRISTIAN S. BAUER 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>ELIZABETH CONWAY</td>
<td>EMCS</td>
<td>76-77</td>
<td>DR. HAROLD I. KLEE 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>RICHARD SEVER</td>
<td>IEMS</td>
<td>76-77</td>
<td>DR. BENJAMIN W. LIN 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>ROBERT LOUIS BALE</td>
<td>EMCS</td>
<td>76-77</td>
<td>DR. CHRISTIAN S. BAUER 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>SHIRLEY MARY BRICKET</td>
<td>IEMS</td>
<td>76-77</td>
<td>DR. ROBERT D. DOERING 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>CARLEY JOAN TATE</td>
<td>EMCS</td>
<td>76-77</td>
<td>DR. CHRISTIAN S. BAUER 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>WENDY MARSHA WILSON</td>
<td>EMCS</td>
<td>76-77</td>
<td>DR. HAROLD I. KLEE 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>JOHN ROBERT COSHING</td>
<td>IEMS</td>
<td>76-77</td>
<td>DR. GEORGE F. SCHRADE 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>DENNIS GENE UPTON</td>
<td>IEMS</td>
<td>76-77</td>
<td>DR. ROBERT D. DOERING 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>MASHAALLAH KHORSANDI</td>
<td>IEMS</td>
<td>76-77</td>
<td>DR. BENJAMIN W. LIN 0.0 0.0 0.0 0.0 0.0 0.0</td>
</tr>
</tbody>
</table>
Appendix M: Listing of Time-Sharing and Job Control Programs
listcat

$.CLIST
$.LOAD @OBJ.J.DATA @OBJ.J.DATA BKASM.DATA BKUP.DATA COPY.DATA COREUPD.$,DATA COUPDT.$,DATA IRSTAX.FORT KAREN.DATA LISTIT.$,DATA NOTE NULLFILE PROCLIB.DATA PRT.DATA RESTORE.DATA SADVISE.$,DATA STUPDT.$,DATA SYSSPNT TEMP.DATA TERMIO1.$,DATA TERMIO2.$,DATA TESTIT.DATA THEJCL.DATA READY
<table>
<thead>
<tr>
<th>DSORG</th>
<th>ALLOC</th>
<th>UNUSED</th>
<th>EXTENTS</th>
<th>DSNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>PO</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>ZFO414.$CLIST</td>
</tr>
<tr>
<td>PO</td>
<td>30</td>
<td>1</td>
<td>1</td>
<td>ZFO414.$LOAD</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ZFO414.BKASM.DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ZFO414.BKUP.DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ZFO414.COPY.DATA</td>
</tr>
<tr>
<td>PS</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>ZFO414.COREUPD.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>ZFO414.COPDTS.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>ZFO414.IRSTAX.FORT</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ZFO414.KAREN.DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ZFO414.LISTIT.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ZFO414.OBJ.DATA</td>
</tr>
<tr>
<td>PO</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>ZFO414.PROCLIB.DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ZFO414.PRT.DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ZFO414.RESTORE.DATA</td>
</tr>
<tr>
<td>PS</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>ZFO414.SADVISE.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ZFO414.SOURCE.BACKUP</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ZFO414.SRCJCL.DATA</td>
</tr>
<tr>
<td>PS</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>ZFO414.STUPD.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>ZFO414.SYSPNT</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ZFO414.TERM101.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ZFO414.TERM102.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ZFO414.TESTIT.DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ZFO414.THEJCL.DATA</td>
</tr>
</tbody>
</table>

23     89    10    29  **TOTAL**
11std $clist mem

ZF0414.$CLIST
--RECFM-LRECL-BLKSIZE-DSORG
   VB  255  13030  PO
--VOLUMES--
   FTUPK1
--MEMBERS--
   @
   ALTCMPRS
   BATCH
   CMD
   CMDS
   CMPRS
   COREUPD
   COUPDT
   DIP
   ERROR
   FA
   INITDSE
   LISTIT
   NEW
   NOTE
   PRNT
   QED2
   RESTORE
   SADVISE
   STUPDT
   TEMPNAME
   THESUB
   WHOIS

   THE FOLLOWING ALIAS NAMES EXIST WITHOUT TRUE NAMES
   ALIAS($)
$.CLIST(ALTCMPRS)
00005 FREE F(SYSPRINT)
00006 FREEALL
00010 REUSE F(INOUT) DA($.CLIST)
00015 ALLOC F(SYSPRINT) DA(ERROR) SPACE(10,10) BLOCK(121)
00020 REUSE F(SYSIN) DA(COPY.DATA)
00030 FETCH IEBCOPY
00034 CALL $.LOAD(TESTRC)
00035 WHEN SYSRC(GT 0) EXEC $.CLIST(ERROR)
00040 REUSE F(INOUT) DA($.LOAD)
00050 FETCH IEBCOPY
00054 CALL $.LOAD(TESTRC)
00055 WHEN SYSRC(GT 0) EXEC $.CLIST(ERROR)
00057 D ERROR
00090 REUSE F(SYSIN) DA(*)
00095 REUSE F(SYSPRINT) DA(*)
00100 FREEALL
00110 EXEC $
READY
An example of use of altcmpsr:

```
listspc

<table>
<thead>
<tr>
<th>DSORG</th>
<th>ALLOC</th>
<th>UNUSED</th>
<th>EXTENTS</th>
<th>DSNAMES</th>
</tr>
</thead>
<tbody>
<tr>
<td>PO</td>
<td>2</td>
<td>1</td>
<td>ZF0414. LIST</td>
<td></td>
</tr>
<tr>
<td>PO</td>
<td>51</td>
<td>1</td>
<td>ZF0414. LOAD</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>10</td>
<td>9</td>
<td>ZF0414. OBJ. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. OBJ. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. BKASM. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. BKUP. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. COPY. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>10</td>
<td>5</td>
<td>ZF0414. COREUPD. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>10</td>
<td>6</td>
<td>ZF0414. COUPDT. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>5</td>
<td>3</td>
<td>ZF0414. IRSTAX. FORT</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. KAREN. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. LISTIT. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. NOTE</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. NUIFILE</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>5</td>
<td>5</td>
<td>ZF0414. PROCLIB. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. PRT. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. RESTORE. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>10</td>
<td>2</td>
<td>ZF0414. SADVISE. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>10</td>
<td>4</td>
<td>ZF0414. STUPDT. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>10</td>
<td>10</td>
<td>ZF0414. SYSPNT</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. TMMP. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. TERMIN1. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. TERMIN2. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. TESTIT. DATA</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414. THEJCL. DATA</td>
<td></td>
</tr>
</tbody>
</table>

25 138 38 31 ** TOTAL **
```
altcmprs

COMMAND ALTTCMPRS NOT FOUND
READY

exec $

READY

altcmprs

RETURN CODE 00 (0)
RETURN CODE 00 (0)
READY

11stspc

<table>
<thead>
<tr>
<th>DSORG</th>
<th>ALLOC</th>
<th>UNUSED EXTENTS</th>
<th>DSNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD</td>
<td>2</td>
<td>1</td>
<td>ZF0414.$.CLIST</td>
</tr>
<tr>
<td>PD</td>
<td>51</td>
<td>21</td>
<td>ZF0414.$.LOAD</td>
</tr>
<tr>
<td>PS</td>
<td>10</td>
<td>9</td>
<td>ZF0414.@OBJ.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.@OBJ.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.BKASM.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.BKUP.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.CPY.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>10</td>
<td>5</td>
<td>ZF0414.COREUPD.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>10</td>
<td>6</td>
<td>ZF0414.COUPLNT.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>5</td>
<td>3</td>
<td>ZF0414.RSTAX.FOPT</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.KAREN.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.LISTIT.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.NOTE</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.NULL.FILE</td>
</tr>
<tr>
<td>PD</td>
<td>5</td>
<td>5</td>
<td>ZF0414.PROCLIB.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.PRT.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.RESTORE.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>10</td>
<td>2</td>
<td>ZF0414.SADVISE.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>10</td>
<td>4</td>
<td>ZF0414.STUPDT.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>10</td>
<td>10</td>
<td>ZF0414.SYSPNT</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.TEMP.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.TERM01.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.TERM02.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.TESTIT.$DATA</td>
</tr>
<tr>
<td>PS</td>
<td>1</td>
<td>1</td>
<td>ZF0414.TFUCL.$DATA</td>
</tr>
</tbody>
</table>
| 25    | 138   | 59             | 31                      | * * * TOTAL * * *

READY
$.CLIST(@)
00010 PROC 1 MODULE MON USER(P.) MSG(JOB(&MON.&USER.&NOI.)
00020 CALL ZU0205.$LOAD(&MODULE.)
00300 FREEALL
00303 READY
$.CLIST(BATCH)
00010 FREE DA(SYSPT)
00020 SUB PRT.DATA REMOTE(2) NOHOLD
00030 FREEALL
00040 REUSE F(SYSPRINT) DA(*)
00050 END
READY
$.CLIST(COREUPD)
00010 FREEALL
00020 FREE F(SYSTERM)
00030 FREE ATTRL(THESE,THIS,TERM)
00040 REUSE F(SYSIN) DA(*)
00050 ATTR THESE KEYLEN(20)
00060 ATTR THIS RECFM(F B) LRECL(567) BLKSIZE(5670)
00070 FREE F(SYSPRINT)
00080 REUSE F(SYSPRINT) DA(SYSPNT)
00090 ALLOC F(REQUPDT) SPACE(10,10) BLOCK(5670) USING(THIS)
00100 ALLOC F(COREDBS) DA('FTU.OU.P1095.COREDBS') USING( THESE)
00110 ATTR TERM LRECL(121)
00120 ALLOC F(SYSTERM) DA(*) USING(TERM)
00130 CALL $.LOAD(COREUPD)
00140 WHEN SYSRC(GT 0) BATCH
00150 FREEALL
00160 TERM LINESIZE(132)
00170 EXEC $.CLIST(LISTIT)
00180 FREEALL
00190 END
READY
$.CLIST(COUPDT)
00020 FREEALL
00030 FREE F(SYSTEM)
00040 FREE ATTRL(THAT, THEM, TERM)
00050 REUSE F(SYSIN) DA(*)
00060 ATTR THEM KEYLEN(8)
00070 ALLOC F(CODBASE) DA('FTU.OU.P1095.CODBASE') USING(THEM)
00080 ATTR THAT RECFM(F B) LRECL(97) BLKSIZE(970)
00090 ALLOC F(CODESEP) SPACE(10,10) BLOCK(970) USING(THAT)
00100 FREE F(SYSPRINT)
00110 REUSE F(SYSPRINT) DA(SYSPNT)
00120 ATTR TERM LRECL(121)
00130 ALLOC F(SYSTEM) DA(*) USING(TERM)
00140 CALL $.LOAD(COUPDT)
00150 WHEN SYSRC(GT 0) BATCH
00160 FREEALL
00170 TERM LINESIZE(132)
00180 EXEC $.CLIST(LISTIT)
00190 FREEALL
00200 END
READY
$.CLIST(LISTIT)
00004 FREE A(SYS)
00005 ATTR SYS LRECL(133) BUFNO(1)
00010 REUSE F(SYSIN) DA(SYSPPNT)
00020 ALLOC F(SYSPRINT) DA(*) USING(SYS)
00030 CALL $.LOAD(LISTIT)
00040 FREEALL
READY
$.CLIST(SADVISE)
00010 FREEALL
00020 FREE F(SYSIN) DA(*)
00030 FREE ATTRL( THEM, THOSE, THESE, TERM)
00040 REUSE F(SYSIN) DA(*)
00050 ATTR THEM KEYLEN(8)
00060 ATTR THOSE KEYLEN(9)
00070 ATTR THESE KEYLEN(20)
00080 ALLOC F(CODBASEx) DA('FTU.OU.P1095.CODBASE') USING(THEM)
00090 FREE F(SYSPRINT)
00100 REUSE F(SYSPRINT) DA(SYSPNT)
00110 ALLOC F(STDDBASE) DA('FTU.OU.P1095.STDBASE') USING(THOSE)
00120 ALLOC F(COREDBS) DA('FTU.OU.P1095.COREDBS') USING(THOSE)
00130 ATTR TERM LRECL(121)
00140 ALLOC F(SYSUTMM) DA(*) USING(TERM)
00150 CALL $.LOAD(SADVISE)
00160 WHEN SYSRC(GT 0) BATCH
00170 FREEALL
00180 TERM LINESIZE(132)
00190 EXEC $.CLIST(LISTIT)
00200 FREEALL
00210 END
READY
.$.CLIST(STUPDT)
00010 FREEALL
00020 FREE F(SYstem)
00030 FREE ATTRL(Them,STEMPT,THOSE,THESE,TERM)
00040 REUSE F(SYSIN) DA(*)
00050 ATTR THEM KEYLEN(8)
00060 ATTR STEMPT LRECL(147)
00070 ATTR THOSE KEYLEN(9)
00080 ATTR THESE KEYLEN(20)
00090 ALLOC F(CODBASE) DA('FTU.OU.P1095.CODBASE') USING(Them)
00100 FREE F(SYSPRINT)
00110 REUSE F(SYSPRINT) DA(SYSPNT)
00120 ALLOC F(STDDBASE) DA('FTU.OU.P1095.STDBASE') USING(THOSE)
00130 ALLOC F(STDTEMP) SPACE(10,10) BLOCK(1470) USING(STEMPT)
00140 ALLOC F(COREDBS) DA('FTU.OU.P1095.COREDBS') USING(THOSE)
00150 ATTR TERM LRECL(121)
00160 ALLOC F(SYSTERM) DA(*) USING(TERM)
00170 CALL $.LOAD(STUPDT)
00180 WHEN SYSRC(GT 0) BATCH
00190 FREEALL
00200 TERM LINESIZE(132)
00210 EXEC $.CLIST(LISTIT)
00220 FREEALL
00230 END
READY
$.CLIST(THESUB)
00010 PROC  I NAME C(C) T(50) I(30) R(1)
00020 QED THEJCL.DATA
00025 5 //ZF0414TH JOB (,,FTU,&T.,&I.,3), 'MARILYN,KHORSANDI', CLASS=&C.,
00030 41 //PL1L.SYSIN DD DSN=ZF0414.$NAME..$.DATA,DISP=SHR
00035 50 //LKED.SYSLMOD DD DSN=ZF0414.$LOAD(&NAME.),DISP=SHR
00040 END S
00050 FREEALL
00060 SUB THEJCL.DATA REMOTE(&R.)
00070 @ 0
00080 END
READY
CSECT
USING *,12
SAVE (14,12)
LR 12,15
ST 13,SAVE+4
LA 13,SAVE
L 2,16 CVT
L 2,0(,2) TCBQRDs
L 2,4(,2) TCB
L 2,124(,2) JSTCB
L 2,112(,2)
L 2,24(,2) CPPL
MVC CPPL+4(12),4(2) MOVE TO MY CPPL
LA 1,CPPL
LINK EP=SUBMIT
L 13,SAVE+4
RETURN (14,12)
CBUFF DC Y(L'BUFF1+4)
BUFF1 DC H'7'
BFUP DATA DC C'SUBMIT BKUP.DATA REMOTE(1) HOLD'
SAVE DC 18F'0'
CPPL DC A(CBUFF)
DC 3F'0'
END BACKUP
//
READY
BKUP DATA

00010 //ZF0414.BK JOB (,,FTU,5,5), 'MARILYN',K.', CLASS=1, NOTIFY=ZF0414,

00020 MSGLEVEL=(1,1)

00030 /•SETUP TAPE=1

00040 //PROCLIB DD DSN=ZF0414.PROCLIB.DATA, DISP=SHR

00050 EXEC BACKUP

00060 READY
LISTIT.$.DATA
00010 LIST: PROC OPTIONS(MAIN);
00020 DCL BUFF CHAR(132) VAR;
00030 ON ENDFILE(SYSIN) BEGIN;
00040 PUT SKIP;
00050 PUT SKIP;
00060 CALL TGET(BUFF);
00070 STOP;
00080 END;
00090 CALL TERMPUT('ADVANCE PAPER TO TOP OF FORM AND');
00100 CALL TERMPUT('HIT RETURN. WHEN PRINT IS FINISHED');
00110 CALL TERMPUT('REMOVE LISTING AND HIT RETURN. ');
00120 CALL TGET(BUFF);
00125 OPEN FILE(SYSPRINT) LINESIZE(132);
00130 DO WHILE ('1'B);
00140 GET EDIT(BUFF)(COL(1),A(132));
00145 BUFF=SUBSTR(BUFF,2);
00220 PUT EDIT(BUFF)(COL(1),A);
00230 END;
00240 END LIST;
READY
PRT.DATA
00010 //ZF0414P JOB (,,FTU,20,40,5), 'MARILYN', CLASS=A, NOTIFY=ZF0414
00020 /*JOBPARM PSS=NO
00030 // EXEC PGM=COPYSOR
00040 //SYSIN DD DUMMY
00050 //SYSPRINT DD SYSOUT=A
00060 //SYSUT1 DD DSN=ZF0414.SYSPNT,DISP=SHR
00070 //SYSUT2 DD SYSOUT=(J,,2400)
00080 //
READY
READY
11st restore.data

RESTORE.DATA
00010 //ZF0414RS JOB (.,.,FTU,10,10),CLASS=A,NOTIFY=ZF0414,
00020 //MSGLEVEL=(1,1)
00030 /*SETUP TAPE=1
00040 //PROCLIB DD DSN=ZF0414.PROCLIB.DATA,DISP=SHP
00050 // EXEC RESTORE
00060 /*
READY
list source.backup

SOURCE.BACKUP
00010 //ZF0414TP JOB (,,FTU,5,5), 'MARILYN', CLASS=2 NOTIFY=ZF0414
00020 //SETUP TAPE=1
00030 //STEP1 EXEC PGM=COPYSAME
00040 //SYSIN DD DUMMY
00050 //SYSPRINT DD SYSOUT=A
00060 //SYSUT1 DD DSN=ZF0414.SADVISE.$DATA, DISP=SHR
00070 //SYSUT2 DD UNIT=(TAPE,,DEFER), VOL=(,RETAIN), DISP=(NEW,CATLG),
00080 // DCB=(RECFM=FB,LRECL=80,BLKSIZ=8000), LABEL=(1,SL),
00090 // DSN=FTU.OU.P1095.SADVISE.SOURCE
00100 //STEP2 EXEC PGM=COPYSAME
00110 //SYSIN DD DUMMY
00120 //SYSPRINT DD SYSOUT=A
00130 //SYSUT1 DD DSN=ZF0414.STUPDT.$DATA, DISP=SHR
00140 //SYSUT2 DD UNIT=(TAPE,,DEFER), VOL=(,RETAIN,REF=*,STEP1.SYSUT2),
00150 // DCB=(RECFM=FB,LRECL=80,BLKSIZ=8000), LABEL=(2,SL),
00160 // DSN=FTU.OU.P1095.STUPDT.SOURCE, DISP=(NEW,CATLG)
00170 //STEP3 EXEC PGM=COPYSAME
00180 //SYSIN DD DUMMY
00190 //SYSPRINT DD SYSOUT=A
00200 //SYSUT1 DD DSN=ZF0414.COUPDT.$DATA, DISP=SHR
00210 //SYSUT2 DD UNIT=(TAPE,,DEFER), VOL=(,RETAIN,REF=*,STEP1.SYSUT2),
00220 // DCB=(RECFM=FB,LRECL=80,BLKSIZ=8000), LABEL=(3,SL),
00230 // DSN=FTU.OU.P1095.COUPDT.SOURCE, DISP=(NEW,CATLG)
00240 //STEP4 EXEC PGM=COPYSAME
00250 //SYSIN DD DUMMY
00260 //SYSPRINT DD SYSOUT=A
00270 //SYSUT1 DD DSN=ZF0414.COREUPD.$DATA, DISP=SHR
00280 //SYSUT2 DD UNIT=(TAPE,,DEFER), VOL=(,RETAIN,REF=*,STEP1.SYSUT2),
00290 // DCB=(RECFM=FB,LRECL=80,BLKSIZ=8000), LABEL=(4,SL),
00300 // DSN=FTU.OU.P1095.COREUPD.SOURCE, DISP=(NEW,CATLG)
READY
LIST SRCJCL.DATA
SRCJCL.DATA
00010 //ZF0414BS JOB (,,,FTU,80,30),'MARILYN',CLASS=C,NOTIFY=ZF0414
00020 /*SETUP TAPE=1
00030 // EXEC PL1LFCL,PARM.PL1L='NOOL,X,A,NT'
00060 //LKD. SYSIN DD DSN=ZF0414.OBJ.DATA,DISP=(OLD,KEEP)
READY

LIST $.CLIST(RESTSRC)
$.CLIST(RESTSRC)
00010 PROC 1 PGM
00020 QED SRCJCL DATA
00030 40 //PL1L.SYSIN DD DSN=FTU.OU.P1095.&PGM..SOURCE,DISP=(OLD,KEEP)
00040 50 //LKD.SYSLMOD DD DSN=ZF0414.$LOAD(&PGM.),DISP=(OLD,KEEP)
00050 SAVE
00060 END
00070 SUB SRCJCL.DATA REMOTE(1)
00080 ST
READY
START temp.data

TEMP.DATA
00010 //ZF0414AS JOB (,,FTU), 'MARILYN', CLASS=R, NOTIFY=ZF0414
00011 /*JOBPARM IO=15, CPU=12
00020 // EXEC ASMG2, PARM= 'DECK, NOLOAD'
00040 //ASM.SYSIN DD DSM=ZF0414.BKASM.DATA, DISP=SUP
00050 //
READY
READY

list termio1.$\$.data

TERMIO1.$\$.DATA

<table>
<thead>
<tr>
<th>00010</th>
<th>TERMPUT CSECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00020</td>
<td>DS OH</td>
</tr>
<tr>
<td>00030</td>
<td>STM 14,12,12(13)</td>
</tr>
<tr>
<td>00040</td>
<td>BALR 12,0</td>
</tr>
<tr>
<td>00050</td>
<td>USING *,12</td>
</tr>
<tr>
<td>00060</td>
<td>ST 13,SAV+4</td>
</tr>
<tr>
<td>00070</td>
<td>LA 13,SAV</td>
</tr>
<tr>
<td>00080</td>
<td>ST 1,R1SAVE</td>
</tr>
<tr>
<td>00090</td>
<td>L 1,0(1)</td>
</tr>
<tr>
<td>00100</td>
<td>*</td>
</tr>
<tr>
<td>00110</td>
<td>L 3,0(1)</td>
</tr>
<tr>
<td>00120</td>
<td>LH 2,6(1)</td>
</tr>
<tr>
<td>00130</td>
<td>DS 0H</td>
</tr>
<tr>
<td>00140</td>
<td>LR 1,3</td>
</tr>
<tr>
<td>00150</td>
<td>LR 0,2</td>
</tr>
<tr>
<td>00160</td>
<td>SVC 93</td>
</tr>
<tr>
<td>00170</td>
<td>L 13,SAV+4</td>
</tr>
<tr>
<td>00180</td>
<td>LM 14,12,12(13)</td>
</tr>
<tr>
<td>00190</td>
<td>BR 14</td>
</tr>
<tr>
<td>00200</td>
<td>SAV DS 18F</td>
</tr>
<tr>
<td>00210</td>
<td>R1SAVE DC 1F'0'</td>
</tr>
<tr>
<td>00220</td>
<td>END</td>
</tr>
</tbody>
</table>

ENTRY POINT
FORCE ALIGNMENT
SAVE CALLER'S REGS
ESTABLISH ADDRESSABILITY

SET UP SAVE AREA
AND PUT ITS ADDRESS IN REG 13
SAVE PARM LIST ADDR
R1 POINTS TO CHAR STRING DOPE VECTOR
R3 POINTS TO CHAR STRING
R2 POINTS TO ITS LENGTH
LOAD OPTIONS & BUFFER ADDR
LOAD TJID & BUFFER SIZE
ISSUE TGET/PUTP SVC
RESTORE REG 13
RETURN CALLER'S REGS
RETURN TO CALLER
MY SAVE AREA
00010 TGET    CSECT
00020 DS      0H
00030 STM     14,12,12(13)
00040 BALR    12,0
00050 USING   *+12
00060 ST      13,SAV+4
00070 LA      13,SAV
00080 ST      1,R1SAVE
00090 L       1,0(1)
00100 L       3,0(1)
00110 LH      7,6(1)
00120 LH      2,4(1)
00130 CR      7,2
00140 BE      FIXED
00150 MVI     VARSW,X'FF'
00160 FIXED   LA    6,1
00170 N       3,=X'00FFFFF'
00180 O       3,=X'80000000'
00190 TGET    (3),(2),R
00200 L       2,R1SAVE
00210 L       3,4(2)
00220 L       2,0(2)
00230 CLI     VARSW,X'FF'
00240 BNE     +%8
00250 STH     1,6(2)

FORCE ALIGNMENT
SAVE CALLERS REGS
ESTABLISH ADDRESsABILITY
SET UP SAVE AREA
AND PUT ITS ADDRESS IN REG 13
DOPE VECTOR

SNARK IT UP

268
TGET (continued)

RETURN TO CALLER'S REGS

END OF DATA

BCTR
L
EX
MVI
LM
BR
MVC
OC
DC
DS
DC
END

00260
00270
00280
00290
00300
00310
00320
00330
00340
00350
00360
00370
00380
00390
00400
00410
00420
00430
00440
00450
00460
00470
00480
00490
00410
00420
00430
00440
00450
00460
00470
00480
00490
00500
00510
00520
00530
00540
00550
00560
00570
00580
00590
00510
00520
00530
00540
00550
00560
00570
00580
00590
00600
00610
00620
00630
00640
00650
00660
00670
00680
00690
00610
00620
00630
00640
00650
00660
00670
00680
00690
00700
00710
00720
00730
00740
00750
00760
00770
00780
00790
00710
00720
00730
00740
00750
00760
00770
00780
00790
00800
00810
00820
00830
00840
00850
00860
00870
00880
00890
00810
00820
00830
00840
00850
00860
00870
00880
00890
00900
00910
00920
00930
00940
00950
00960
00970
00980
00990
00910
00920
00930
00940
00950
00960
00970
00980
00990
01000
01010
01020
01030
01040
01050
01060
01070
01080
01090
01010
01020
01030
01040
01050
01060
01070
01080
01090
01100
01110
01120
01130
01140
01150
01160
01170
01180
01190
01110
01120
01130
01140
01150
01160
01170
01180
01190
01200
01210
01220
01230
01240
01250
01260
01270
01280
01290
01300
01310
01320
01330
01340
01350
01360
01370
01380
01390
01400
01410
01420
01430
01440
01450
01460
01470
01480
01490
01410
01420
01430
01440
01450
01460
01470
01480
01490
01500
01510
01520
01530
01540
01550
01560
01570
01580
01590
01510
01520
01530
01540
01550
01560
01570
01580
01590
READY

list testIt.data

TESTIT.DATA

00010 CALL BACKUP
00020 STOP
00030 END

READY
READY

11st thejcl.data

THEJCL.DATA
00005 //ZF0414TH JOB (,,FTU,50,30,3),'MARILYN,KHORSANDI',CLASS=C,
00006 // NOTIFY=ZF0414
00007 /*JOBPARM PSS=NO
00010 // EXEC PL1LFCL,PARM.PLL='NOL,XREF,A,NT',DPRTY=(14,12)
00041 // PL1L.SYSIN DD DSN=ZF0414.COREUPD.*.DATA,DISP=SHR
00050 // LKED.SYSLMOD DD DSN=ZF0414.*.LOAD(COREUPD),DISP=SHR
00060 // LKED.SYSIN DD DSN=ZF0414.@OBJ.DATA,DISP=SHR
00070 /*
READY
Appendix N: Process For Recovery of Lost Data
In case of a system crash resulting in loss of data from one of the databases, the user may restore the data in the databases by executing the following sequence:

READY
restore
READY

The restore command restores all data previously stored in the databases.

For purposes of backing up the source programs for SADVISE, COUPDT, COREUPD, and STUPDT, the following corresponding files were created:

SADVISE--FTU.OU.P1095.SADVISE.SOURCE
COUPDT--FTU.OU.P1095.COUPDT.SOURCE
COREUPD--FTU.OU.P1095.COREUPD.SOURCE
STUPDT--FTU.OU.P1095.STUPDT.SOURCE
REFERENCES


