210

THE PRELIMINARY DESIGN OF A STUDENT ADVISORY SYSTEM

by

RONALD J. VIETH

B.S., Kansas State University, 1979
M.S., Kansas State University, 1984

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Approved by:

Roger T. Hartley
Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to express sincere gratitude to Dr. Roger Hartley, Professor in the Computer Science Department, Kansas State University. Dr. Hartley was an extremely cooperative and helpful factor in the development of the preliminary design of the Student Advisory System.

I would like to thank Dr. Clifford Stark, Professor in the Computer Science Department, Kansas State University, for helping to get the preliminary design of the Student Advisory System off the ground.

I would also like to thank Thomas Rodenbaugh, undergraduate student in the Computer Science Department, Kansas State University, for providing coded procedures used to support part of the Student Advisory System.

To my wife, Kaye, and four children, without their encouragement and cooperation this report would not have been written. Sincere appreciation goes to my wife for assisting in the editing of this report.

INTRODUCTION

The scope of this report is to develop and document the preliminary design of an automated student advisory system. The KSU Computer Science Department wants to develop an automated student advisory system which would essentially take the place of student advisors. The Student Advisory System may be implemented with the UNIX Operating System, but the decision to do so is not final. The system is to contain information which addresses seven different categories. The categories are as follows: Student, Course, Instructor, University, College, Department, and Curriculum. Each of the categories contain associated information that can assist university students in planning the pursuit of an educational goal. Students throughout the university may eventually be allowed access to the student advisory system.

This report documents all of the effort which has gone into the preliminary development of an automated student advisory system. The report documents work with the Course category which began in the spring 1984 Implementation Project course, CMPSC 690. The report covers the development of the Course category that took place in CMPSC 690 as well as the development of the other six categories. The products of the preliminary design effort include Entity-Relationship (E-R) diagrams and data structure diagrams. Structure declarations for each of the seven categories will be provided in the Appendix.

REQUIREMENTS REVIEW

The development of the student advisory system began with a review of the student advisory database requirements specification. The requirements specification is a listing of the major categories and their associated contents. The Student Advisory System requirements specification is provided below.

Numbers are provided which indicate the sizes of the different items. The numbers after each of the major categories indicate the number of occurrences of that category. For example, the Student category will accomodate about 1000 students. The numbers after each of the individual items indicate the information size of that item. For example, SSN under Student will contain 9 characters.

STUDENT ADVISORY SYSTEM REQUIREMENTS

STUDENT: (1000)

- name (30)
- SSN (9)
- declared major (6)
- entry date to college (6)
- ACT score (5)
- high school GPA (3)
- declared minor (if any) (6)
- standing (fr, so, jr, sr, grad etc.) (4)
- courses taken (with grade obtained) (8 * 50)
- individual curriculum (8 * 50)
- technical electives taken (8 * 25)
- total hours obtained (3)

GPA (4)

transfer credits (if any) (3)

major professor (grad only) (30)

committee members (grad only) (30 * 4)

interests (20 * 5)

career goals (20 * 5)

preferred workload (20 * 5)

history of query sessions (20 * 100)

bad times of day (20 * 2)

perceived strengths and weaknesses (20 * 6)

COURSE: (3500)

course number (8)

reference number (8)

name (30)

catalog description (20 * 10)

number of credits (1)

prerequisites (20)

instructor (30)

current enrollment (3)

maximum enrollment (3)

place (5)

time (9)

frequency offered (1)

INSTRUCTOR: (2000)

name (30)

department (5)

major area (20)

type of appointment (10)

courses usually taught (8 * 5)

evaluation record (10)

perceived strengths and weaknesses (20 * 6)

graduate faculty status (3)

UNIVERSITY:

entrance prerequisites (20 * 10)

math requirements (20 * 10)

degree requirements (20 * 10)

colleges (9)

COLLEGE: (9)

name (30)

general requirements (20 * 10)

special requirements (20 * 10)

departments (80)

DEPARTMENTS: (80)

name (30)

general requirements (20 * 10)

special requirements (20 * 10)

contact person (30)

curriculum (215)

CURRICULUM: (215)

curricula (10)

curricula requirements (20 * 10)

TYPES OF QUERIES

User queries will vary with the students. Basically, there are two types of students: new students and old students. Typically, new students will use the system for two purposes. The first purpose will be to ascertain an educational goal, and the second purpose will be to

ascertain a semester schedule. New student queries will access a greater portion of the student advisory database than does the queries of old students who have been at the university for some time. Typically, old students will use the system to ascertain a semester schedule or to determine what further requirements must be met in order to graduate. Thus the queries are basically goal oriented or schedule oriented.

Goal oriented queries should access several major categories of information contained in the database system. Goal oriented queries will address University, College, Department, and Curriculum. Accessing these categories will provide the user with information about general and specific requirements within the university.

Schedule oriented queries should access several major categories of information contained in the database system. Schedule oriented queries will address Student, Course, and Instructor. Accessing the Student category will provide the user with information about the student's current status. Accessing the Course category will provide the user with information about the course offerings. Accessing the Instructor category will provide the user with information about specific instructors.

Overall, the student advisory database system has to be versatile enough to handle queries from old students as well as new students. Basically, queries will be either goal oriented or schedule oriented. Each type of query will access different categories of information contained in the advisory system.

DEVELOPMENT OF COURSE

The E-R Diagram

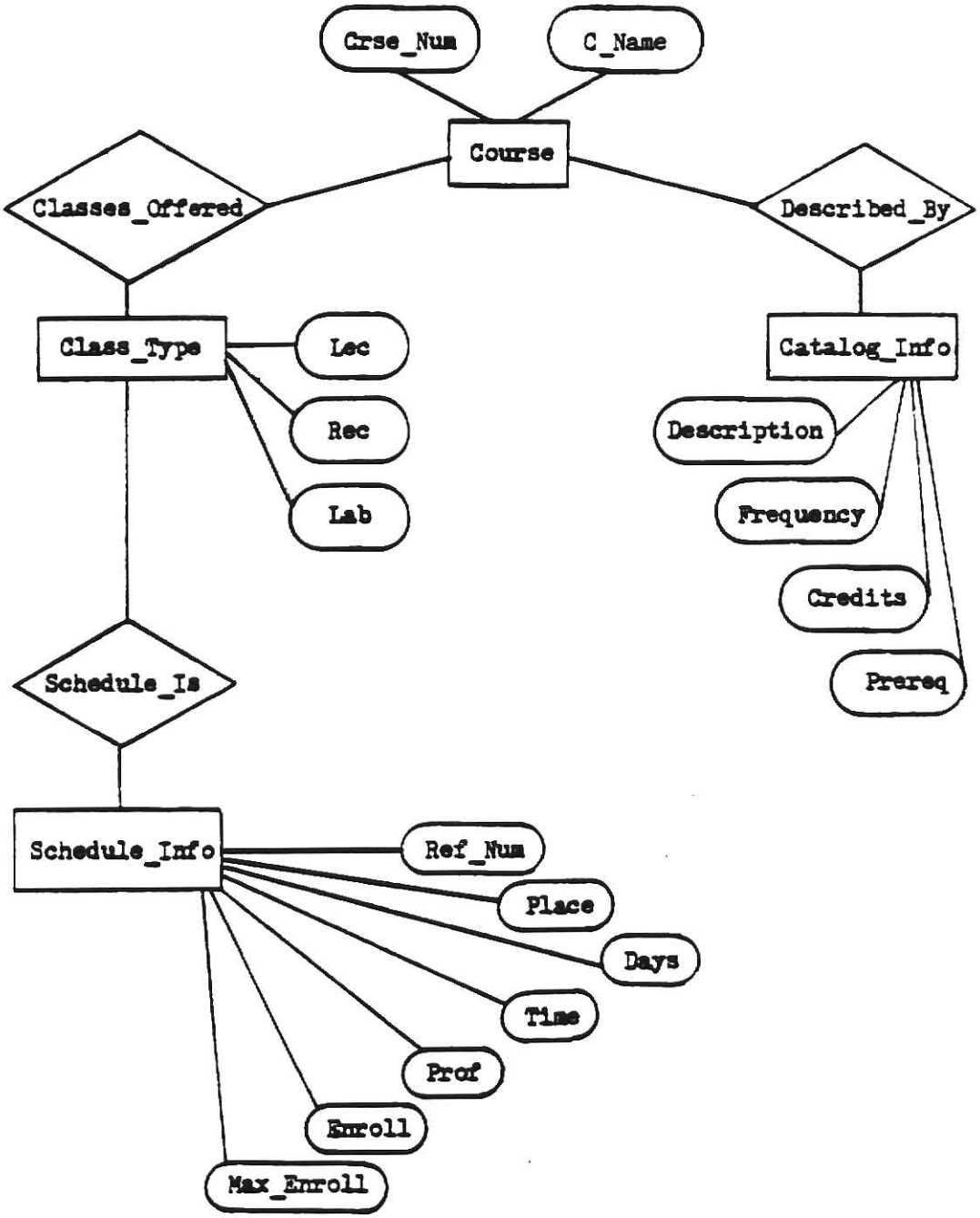
Initial development efforts began with a focus on the course category. Different E-R diagrams were proposed which modeled the course category. A final E-R diagram emerged, and it takes into account several changes suggested during the creation of the earlier E-R diagrams. The final E-R diagram for the Course category is provided in Figure 1. Points of controversy with earlier E-R diagrams involved where to place the instructor attribute and how to diagram class types (such as lectures, recitations, and laboratories all under one course name).

As the E-R diagram took on a final form, it became immediately obvious that the course category would include at least two files: a course information file and a schedule information file. With the idea of two files in mind, it was decided that the instructor attribute most appropriately belongs to the schedule information file since instructor's names appear in the line schedule in actuality.

Class types deserve special attention. Since some courses have lectures as well as recitations and laboratories, it was decided to try and show this possibility in the final E-R diagram. The portion of the E-R diagram that deals with class types comprises the bulk of the final E-R diagram, and this portion of the E-R diagram deals with the schedule information file.

The E-R diagram for the Course category includes four entities: Course, Class_Type, Catalog_Info, and

Figure 1.
COURSE E-R Diagram



Schedule_Info. The Course entity has two attributes: Crse_Num and C_Name. The Crse_Num attribute represents the course number for a given course. The C_Name attribute represents the name of a given course. The other entities in the Course E-R diagram hinge off of the course entity in one way or another.

The Class_Type entity is related directly to the Course entity. The Class_Type entity has three attributes: Lec, Rec, and Lab. The Lec attribute represents lectures. The Rec attribute represents recitations. The Lab attribute represents laboratories.

The Schedule_Info entity is related directly to the Class_Type entity. The Schedule_Info entity has seven attributes: Ref_Num, Place, Days, Time, Prof, Enroll, and Max_Enroll. The Ref_Num attribute represents the reference number for a given class type under one course. The Place attribute represents the building and room number in which a given course will be taught. The Days attribute represents the days of the week a particular class will meet. The Time attribute represents the time of the day a particular class will meet. The Prof attribute represents the name of the instructor teaching a given class. The Enroll attribute represents the number of students enrolled in a given class. The Max_Enroll attribute represents the maximum enrollment figure for a given class.

The Catalog_Info entity is related directly to the Course entity. The Catalog_Info entity has four attributes: Description, Frequency, Credits, and Prereq. The Description attribute represents the catalog description for

a given course. The Frequency attribute represents the semesters in which a given course is offered. The Credits attribute represents the number of credit hours a particular course is worth. The Prereq attribute represents the course prerequisites for a given course.

The Data Structure

After finalizing the E-R diagram, effort concentrated on developing a physical structure for the Course category. The initial physical structure proposals were modified several times and redrawn to handle all of the controversial aspects. As the class type structure deserved a special structuring effort, the decision was made to include some kind of offset pointer scheme for the different class type information which could appear under one course name. The decision was made to include a class field in the course information file structure which would have a length field and an offset pointer field. The length field indicates the number of records appearing in the schedule information file for a given course. The offset pointer field indicates where the course lecture, recitation, and laboratory records appear in the schedule information file.

A special field in the course information file is the course description field. This field contains two pieces of information which are a length field and an offset pointer field. The length field indicates the length of character string information in a course description file. The offset pointer field indicates where the description for a given course appears in the course description file by the use of

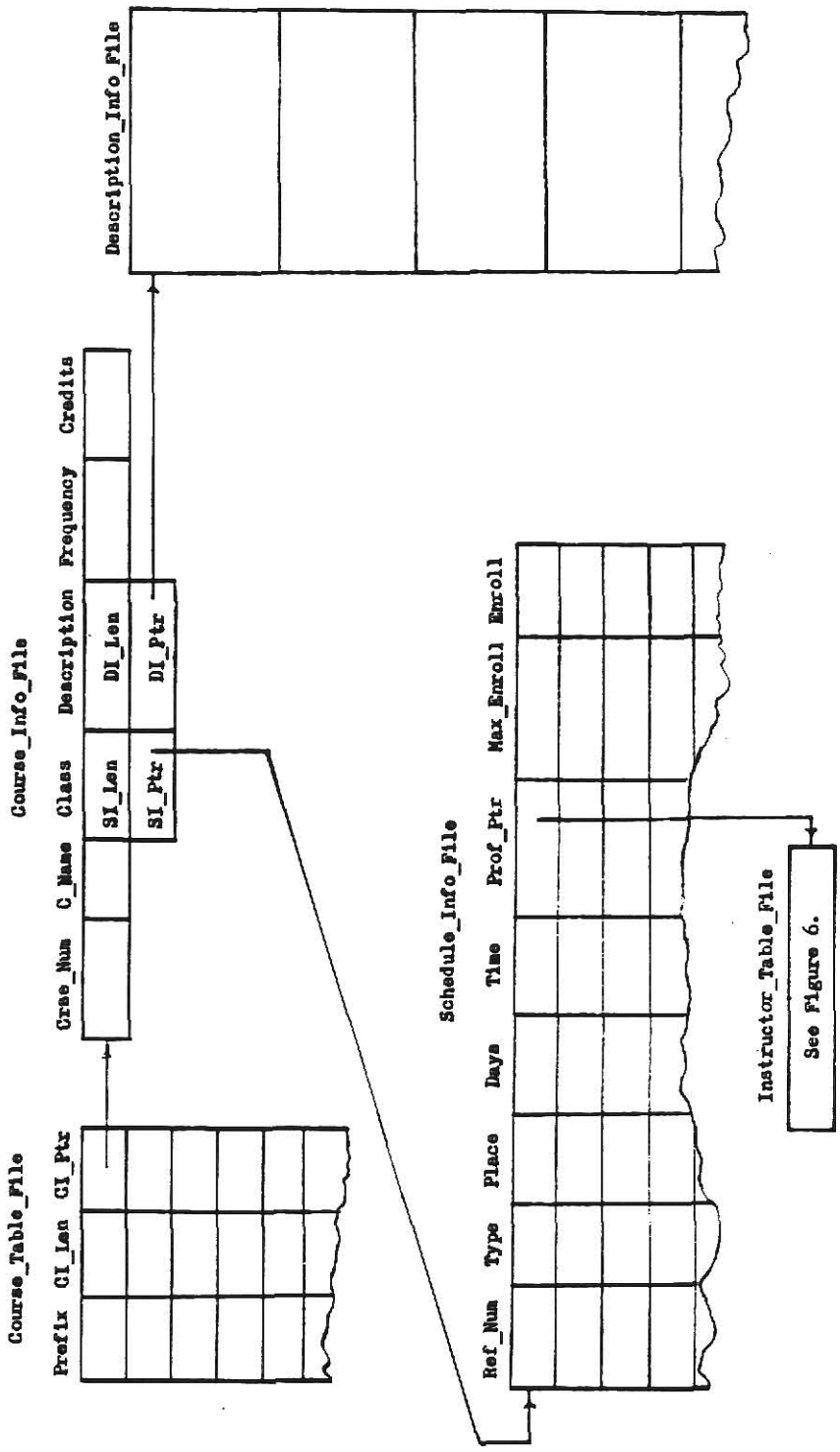
an offset from the top of the file.

Another development in the physical structure of the course category is the proposal to use a course table file for pointing at the appropriate course numbers in the course information file. This suggestion saves space in the course number field of the course information file. The course table file consists of a structure with three fields. The first field is a character field for representing the course number department prefix. The second field is a length field to indicate how many course numbers in the course information file pertain to a particular department. The third field consists of an offset pointer which points to the first course pertaining to a department in the course information file.

As it stands at this point in time, the physical structure of the course category for the student advisory database project consists of four files: Course_Table_File, Course_Info_File, Description_Info_File, Schedule_Info_File. All of the files are interrelated through means of some pointer information, and thus this structure bears out some of the flavor of a database. The physical structure of the Course category is provided in Figure 2.

The Course_Table_File is a record structure for getting at the appropriate course in the Course_Info_File. The Course_Table_File consists of three fields: Prefix, CI_Len, and CI_Ptr. The Prefix field contains the departmental prefix for given courses. The CI_Len field indicates the number of courses in the appropriate portion of the Course_Info_File. The CI_Ptr field indicates the amount of

FIGURE 2.
COURSE Data Structure



offset at which a particular set of courses from one department will appear from the top of the Course_Info_File.

The Course_Info_File is a record structure which consists of six fields: Crse_Num, C_Name, Class, Description, Frequency, and Credits. The Crse_Num field represents the course number for a given course. The C_Name field represents the name of a given course. The Class field is broken down into a subrecord structure with two fields: SI_Len and SI_Ptr. The SI_Len field indicates the number of classes in the Schedule_Info_File for a particular course. The SI_Ptr field indicates the amount of offset at which a particular course first appears from the top of the Schedule_Info_File.

The Description field is broken down into a subrecord structure which contains two fields: DI_Len and DI_Ptr. The DI_Len field indicates the length of the description of a particular course which is described in the Description_Info_File. The course description in the Description_Info_File also includes mention of the course prerequisites. The DI_Ptr field indicates the amount of offset at which a particular course description will appear from the top of the Description_Info_File. The Frequency field indicates the semesters in which a particular course is usually offered. The Credits field indicates the number of credit hours a particular course is worth.

The Description_Info_File is a textual file. This textual file contains course descriptions for all the courses taught at this university. The

Description_Info_File is pointed at by pointers in the Course_Info_File for each of the different courses.

The Schedule_Info_File consists of eight fields: Ref_Num, Type, Place, Days, Time, Prof_Ptr, Max_Enroll, and Enroll. The Ref_Num field indicates the reference number for a given class. The Type field indicates whether a class is a lecture, recitation, or laboratory. The Place field indicates the building and room number in which a particular class is to be taught. The Days field indicates the days of the week a particular class is to meet. The Time field indicates the times of the day a particular class is to meet. The Prof_Ptr field is a pointer field which points to the location of a table named Instructor_Table_File. The Max_Enroll field indicates the maximum number of students that can be enrolled in a particular class. The Enroll field indicates the total number of students enrolled in a particular class.

DEVELOPMENT STRATEGY EMERGES

After completing the preliminary design of the Course category, it was decided upon to attack the remaining categories in a more organized fashion. The general approach used in developing the E-R diagram for the course category was to draw a rough diagram which encompassed the requirements specification for that category. Several different E-R diagrams were drawn for the Course category. Each successive E-R diagram took into account a number of different refinements to prior versions. After devoting considerable time to just the Course category, hindsight

demonstrated the need to bridge the gap between having a requirements specification and developing an E-R diagram for a given category.

The strategy that emerged was to outline a given category from the given specification. Outlining each of the remaining six categories was a tremendous leap toward developing satisfactory E-R diagrams. Since the outlining strategy was merely an aid to obtaining appropriate E-R diagrams, the outlines used in this effort are not provided in this report.

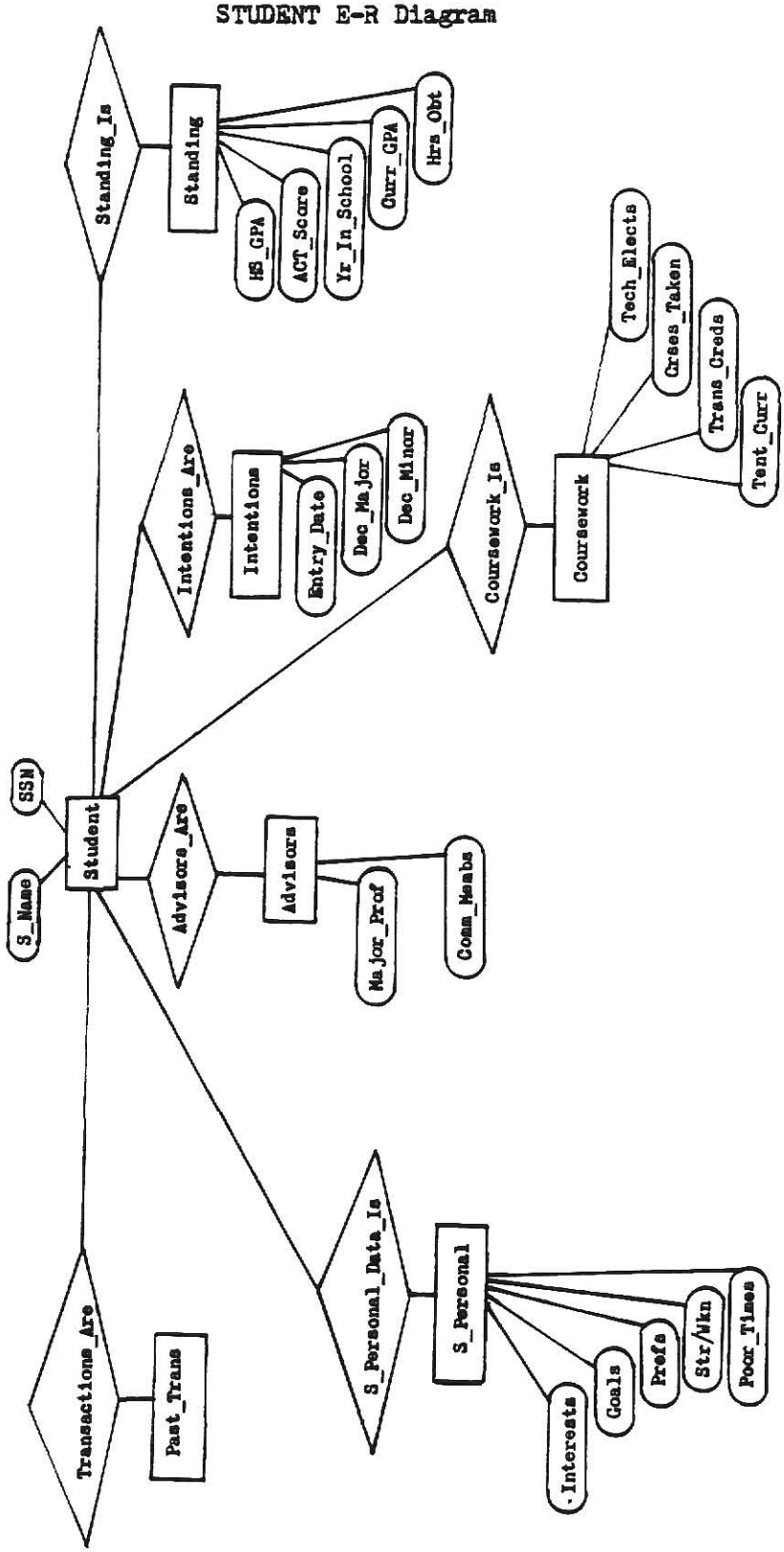
If one were to use the outline approach, the technique is simple. As with the bulk of the student advisory system categories, the technique is to use major outline heading as entities in the applicable E-R diagram. If a particular outline heading contains several subheadings, each subheading then becomes an entity related to the appropriate category.

DEVELOPMENT OF STUDENT

The E-R Diagram

The Student category was initially outlined into major headings in an effort to arrive at a suitable E-R diagram. The E-R diagram for the Student category is provided in Figure 3. The Student category was broken down into seven different entities: Student, Past_Transactions, Standing, Intentions, Advisors, Coursework, and S_Personal. The Student entity contains two attributes: S_Name and SSN. S_Name represents the name of the student, and SSN

FIGURE 3.
STUDENT E-R Diagram



represents the social security number of the student. The Student entity is also related to the remaining entities, and it appears to be the center of a spoke-like arrangement where all the other entities surround Student through different relationships. The Past_Transactions entity has no attributes.

The Standing entity contains five attributes: HS_GPA, ACT_Score, Yr_In_School, Curr_GPA, and Hrs_Obt. HS_GPA represents the student's high school grade point average. ACT_Score represents the student's American College Test score. Yr_In_School represents the student's year in college. Curr_GPA represents the student's current grade point average. Hrs_Obt represents the total number of hours a student has completed in college.

The Intentions entity has three attributes: Entry_Date, Dec_Major, and Dec_Minor. Entry_Date represents the date a student was admitted into a particular college within the university. Dec_Major represents a declared major area of study. Dec_Minor represents a declared minor area of study if applicable.

The Advisors entity has two attributes: Major_Prof and Comm_Membs. Major_Prof represents the major professor a student is working under provided the student is a graduate student. Comm_Membs represents the members on a graduate student's committee.

The Coursework entity has four attributes: Tech_Elects, Crses_Taken, Trans_Creds, and Tent_Curr. Tech_Elects represents the technical electives a student intends to complete. Crses_Taken represents the courses a

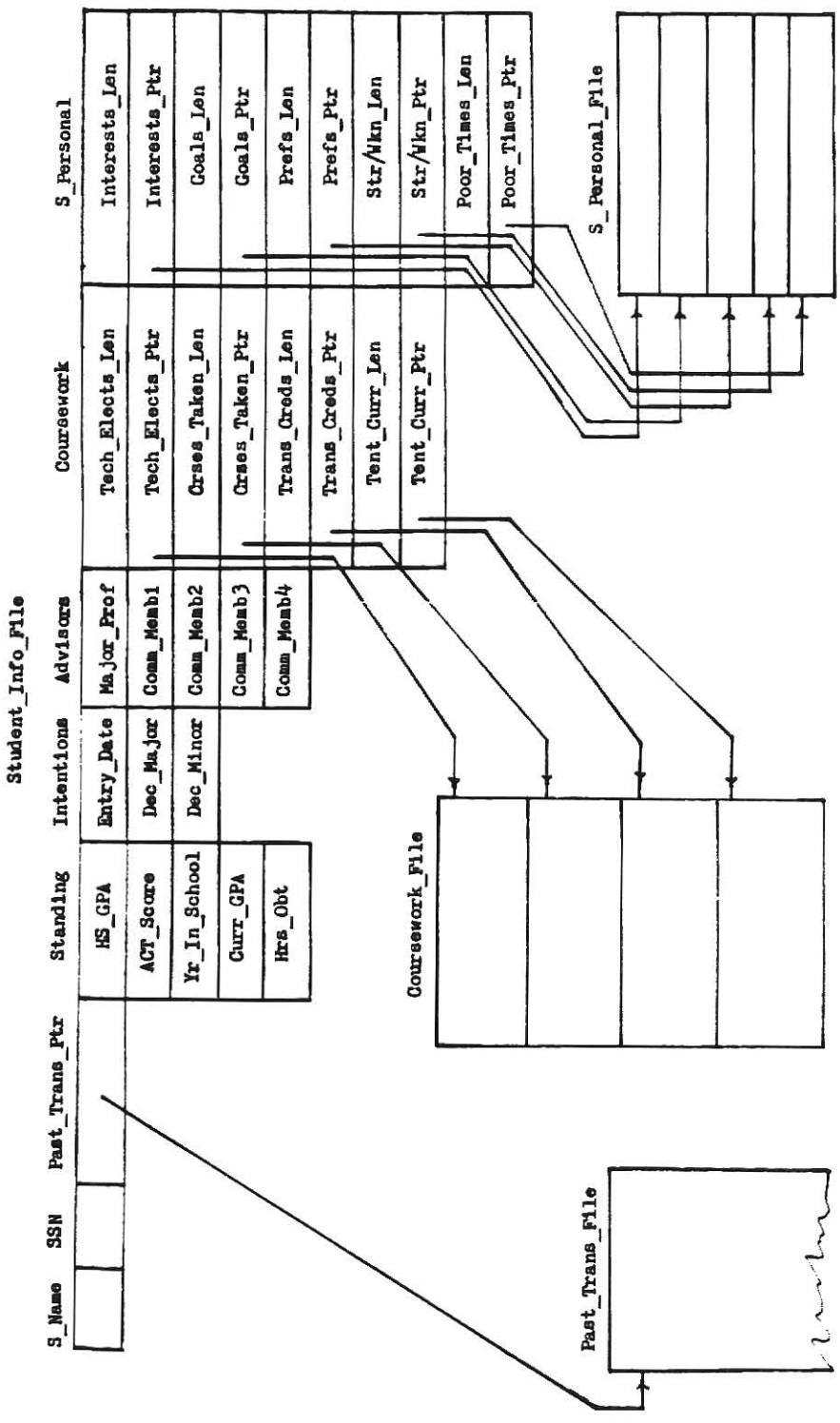
student has already completed. Trans_Creds represents the transfer credits a student has obtained in work at another college. Tent_Curr represents the tentative coursework a student intends to complete in order to obtain a given goal.

The S_Personal entity has five attributes: Interests, Goals, Prefs, Str/Wkn, and Poor_Times. Interests represents a students particular interests which may include favorite pasttimes and hobbies. Goals represents what educational and career goals a particular student has in mind. Prefs represents what preferences a student may have in regards to courses or instructors. Str/Wkn represents the student's perceived strengths and weaknesses. Poor_Times represents the times at which a student can not possibly be on campus.

The Data Structure

After completing the E-R diagram for the Student category, effort was focused on the development of an appropriate physical structure for this category. The physical structure of the Student category is provided in Figure 4. The overall structure of Student is a record structure with fields for the following: S_Name, SSN, Past_Trans_Ptr, Standing, Intentions, Advisors, Coursework, and S_Personal. The S_Name field contains the name of the student. The SSN field contains the social security number of the student. The Past_Trans_Ptr is a pointer which indicates the location of a file named Past_Trans_File for containing a history of all queries a particular student makes with the student advisory system.

Figure 4.
STUDENT Data Structure



The Standing field is broken down into a subrecord structure which contains five fields: HS_GPA, ACT_Score, Yr_In_School, Curr_GPA, and Hrs_Obt. The HS_GPA field indicates a particular student's high school grade point average. The ACT_Score field indicates a particular student's score on the American College Test. The Yr_In School field indicates the student's year in college. The Curr_GpA field indicates the student's current grade point average in college. The Hrs_Obt field indicates the total number of hours a particular student has completed.

The Intentions field is broken down into a subrecord structure which contains three fields: Entry_Date, Dec_Major, and Dec_Minor. The Entry_Date field indicates the date on which a student became enrolled in a particular college. The Dec_Major field indicates the student's desired area of major work. The Dec_Minor field indicates the student's desired area of minor work.

The Advisors field is broken down into a subrecord structure which contains five fields: Major_Prof, Comm_Memb1, Comm_Memb2, Comm_Memb3, and Comm_Memb4. The Major_Prof field will contain the name of a graduate student's major professor. Each of the other four fields contain the names of the graduate student's committee members.

The Coursework field is broken down into a subrecord structure which contains fields for each of the attributes of the Coursework entity. Each attribute has a length field and a pointer field. The length field represents the length of a portion of a file named Coursework_File which will

contain textual information relative to that attribute. The pointer field indicates the amount of offset at which the textual information relative to a given attribute appears from the top of the file named Coursework_File.

The S_Personal field is broken down into a subrecord structure which contains fields for each of the attributes of the S_Personal entity. Again the attributes are paired off into length and pointer fields. These length and pointer fields relate to a file named Personal_File which is a text file for containing the information pertinent to the Personal entity.

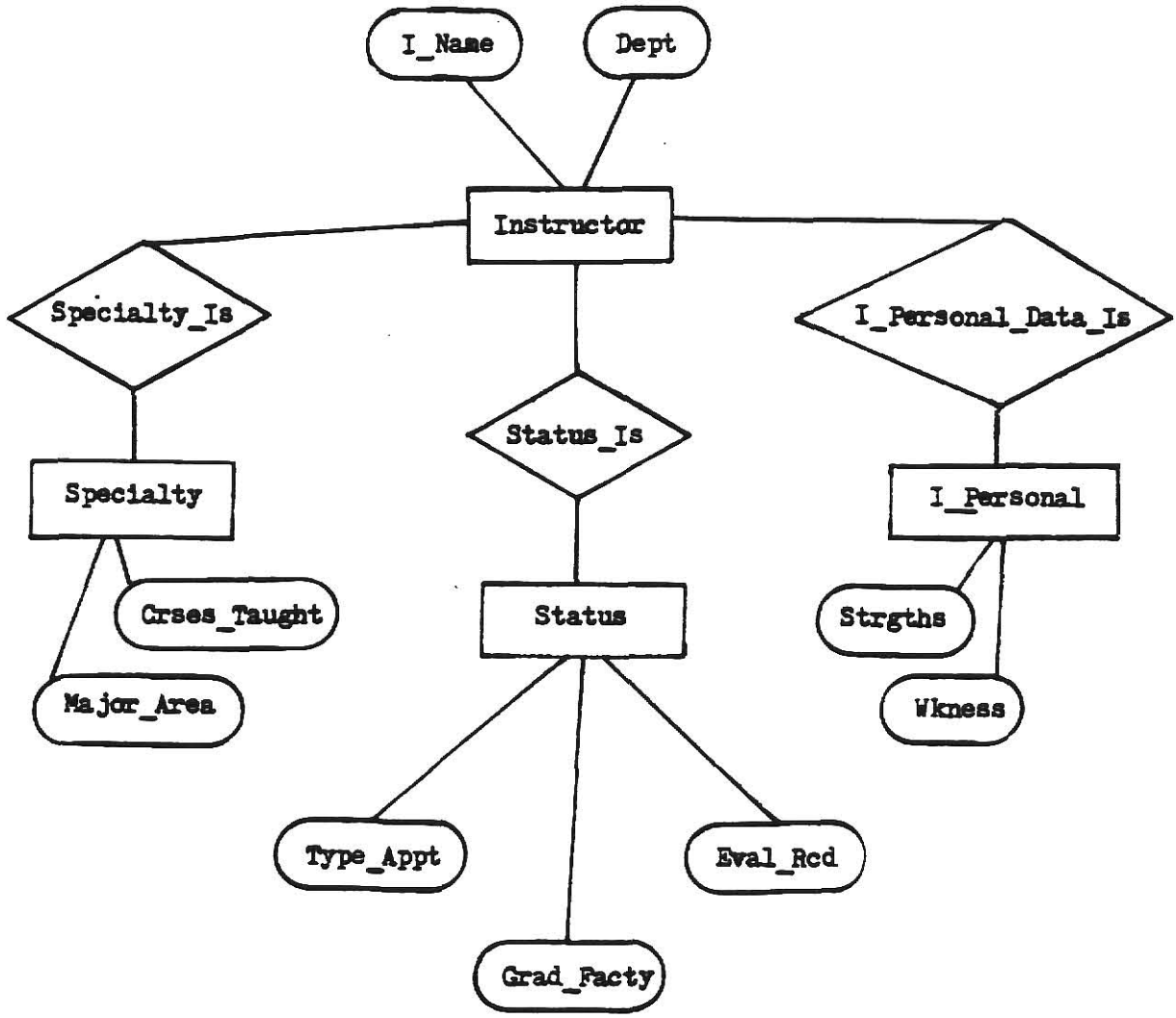
DEVELOPMENT OF INSTRUCTOR

The E-R Diagram

The Instructor category was initially outlined into major headings in an effort to arrive at a suitable E-R diagram. The E-R diagram for the Instructor category is provided in Figure 5. The Instructor category was broken down into four different entities: Instructor, Specialty, Status, and I_Personal. The Instructor entity contains two attributes: I_Name and Dept. I_Name represents the name of the instructor. Dept represents the name of a department the instructor works for. The Instructor entity is also related to the remaining entities, and it appears to be the center of a spoke-like arrangement where all of the other entities surround Instructor through different relationships.

The Specialty entity has two attributes: Crses_Taught

Figure 5.
INSTRUCTOR E-R Diagram



and Major_Area. Crses_Taught represents the courses a particular instructor usually teaches. Major_Area represents the area or areas of major interest to the particular instructor.

The Status entity has three attributes: Type_Appt, Grad_Facty, and Eval_Rcd. Type_Appt represents the particular type of teaching appointment for the particular instructor. Grad_Facty represents whether the particular instructor is graduate faculty. Eval_Rcd represents the evaluation record for a given instructor.

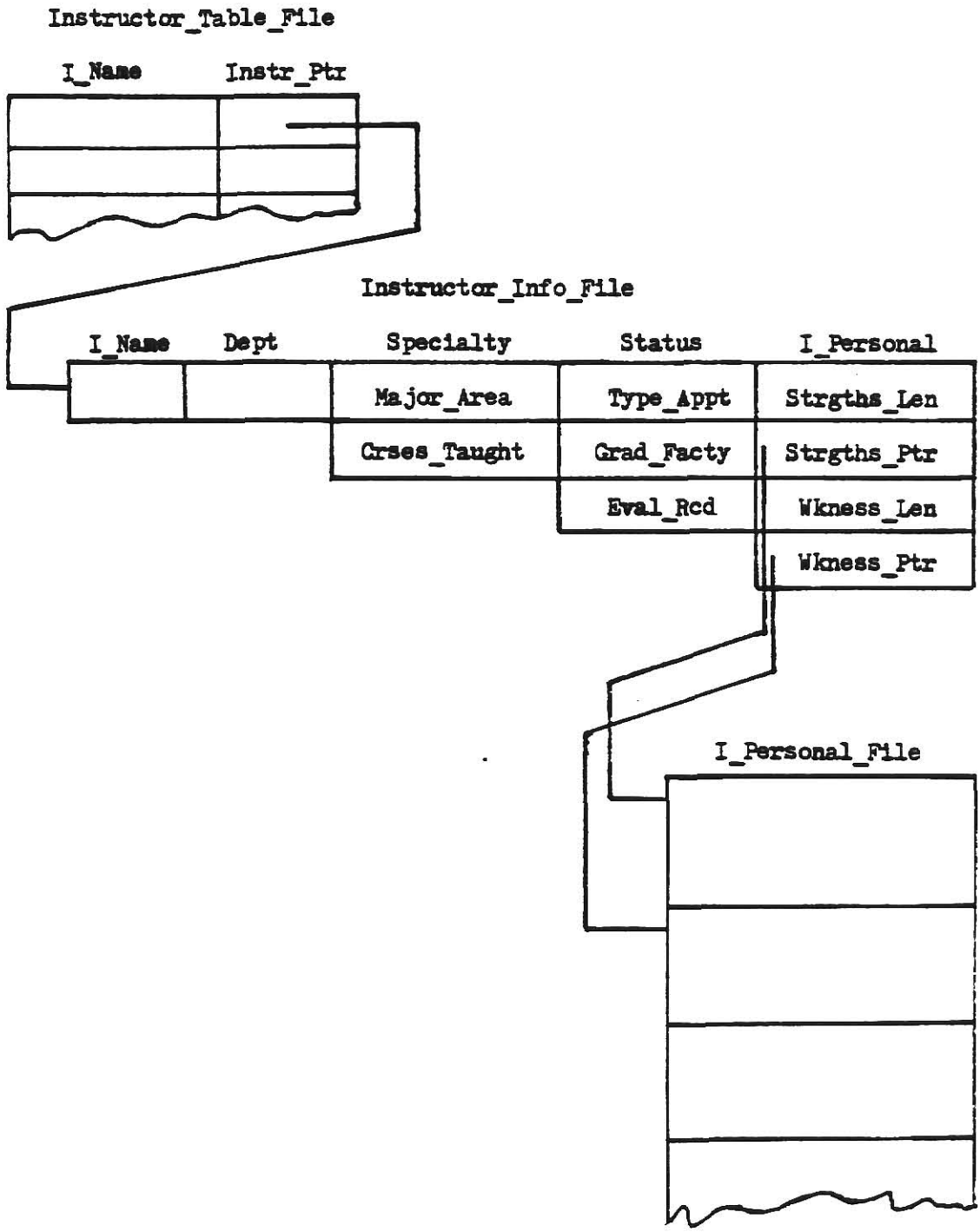
The I_Personal entity has two attributes: Strgths and Wkness. Strgths represents the perceived strengths of a particular instructor. Wkness represents the perceived weaknesses of a particular instructor.

The Data Structure

After completing the E-R diagram for the Instructor category, effort was focused on the development of an appropriate physical structure for this category. The physical structure of the Instructor category is provided in Figure 6. The overall structure of Instructor is two record structures and a text file: Instructor_Table_File, Instructor_Info_File, and I_Personal_File.

The Instructor_Table_File is a record structure which contains two fields: I_Name and Instr_Ptr. The I_Name field will contain the name of a given instructor and all instructors are placed in the Instructor_Table_File in alphabetical order. The Instr_Ptr field indicates the amount of offset at which the particular instructor

Figure 6.
INSTRUCTOR Data Structure



information appears from the top of the file named Instructor_Info_File.

The Instructor_Info_File contains five fields: I_Name, Dept, Specialty, Status, and I_Personal. The I_Name field will contain the name of a given instructor and all instructors are placed in the Instructor_Info_File. The Dept field contains the name of the department the instructor works for. The Specialty field is broken down into a subrecord structure which contains two fields: Major_Area and Crses_Taught. The Major_Area field contains the name of the major area in which an instructor is most interested. The Crses_Taught field contains a small list of courses a particular instructor usually teaches. The Status field is broken down into a subrecord structure which contains three fields: Type_Appt, Grad_Facty, and Eval_Rcd. The Type_Appt field represents a particular instructor's type of appointment. The Grad_Facty field represents whether a particular instructor is on the graduate faculty. The Eval_Rcd field contains evaluation record information.

The I_Personal field is broken down into a subrecord structure with four fields: Strgths_Len, Strgths_Ptr, Wkness_Len, and Wkness_Ptr. The fields are pairs of length and pointer fields for indicating appropriate information about the strength and weaknesses of a particular instructor. The strengths and weaknesses information is contained in a textual file named I_Personal File.

As was just stated, the I_Personal_File is a textual file for containing information about the strengths and

weaknesses of given instructors. The I_Personal_File will contain strengths and weaknesses information for all instructors in the university.

DEVELOPMENT OF UNIVERSITY

The E-R Diagram

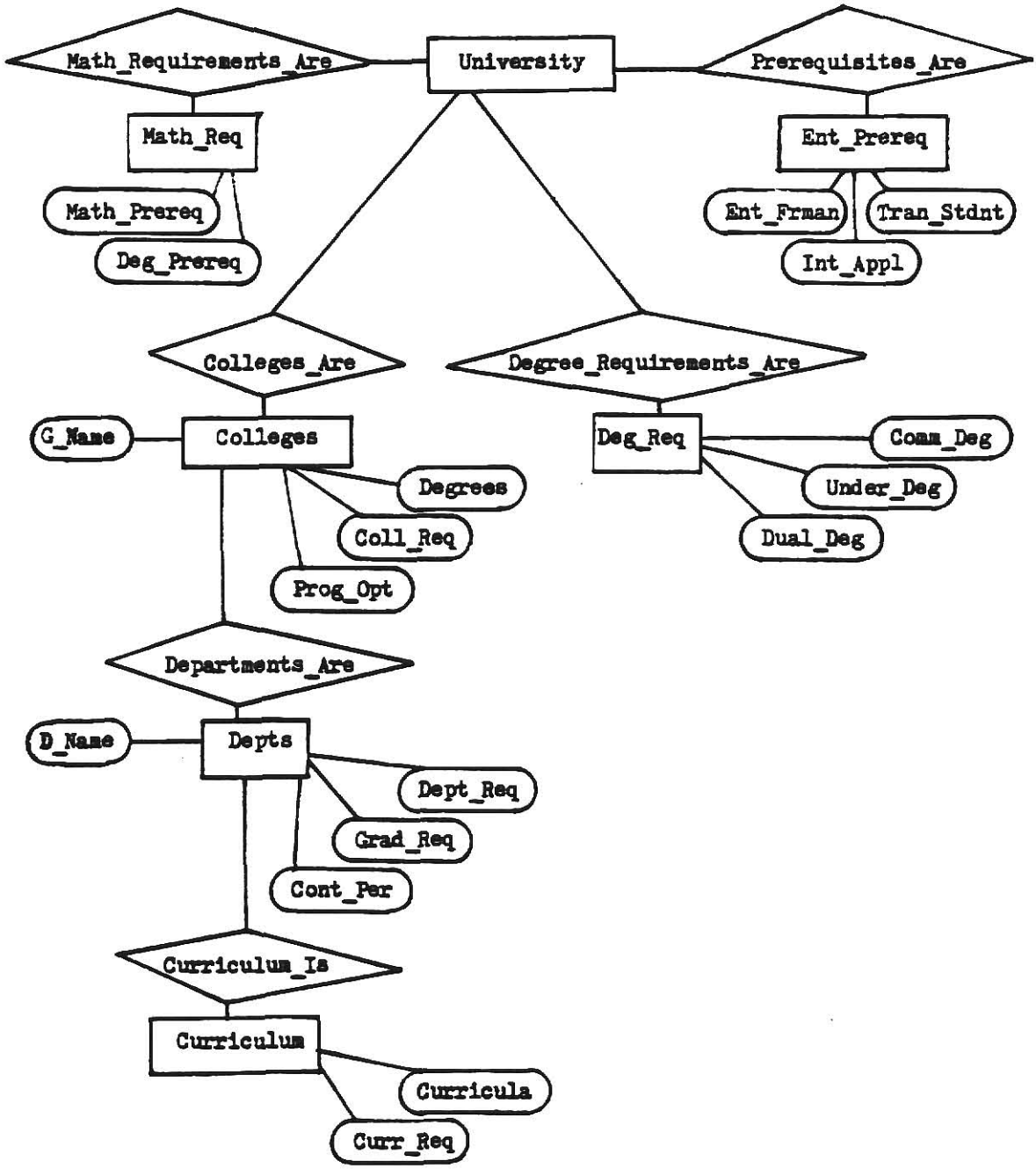
The University category was initially outlined into major headings in an effort to arrive at a suitable E-R diagram. See the appropriate portion of the E-R diagram provided in Figure 7. The University category was broken down into four different entities: Ent_Prereq, Math_Req, Deg_Req, and Colleges.

The Ent_Prereq entity has three attributes: Ent_Frman, Tran_Stdnt, and Int_Appl. The Ent_Frman attribute represents the entrance requirements for entering freshmen. The Tran_Stdnt attribute represents the entrance requirements for transfer students. The Int_Appl attribute represents the entrance requirements for international applicants.

The Math_Req entity has two attributes: Math_Prereq and Deg_Prereq. The Math_Prereq attribute represents the mathematical background that a student should have before entering college. The Deg_Prereq attribute represents any other prerequisites that a particular student should be able to meet for any particular degree.

The Deg_Req entity has three attributes: Comm_Deg, Under_Deg, and Dual_Deg. The Comm_Deg attribute represents common degree requirements for all college students. The

Figure 7.
UNIVERSITY/COLLEGE/DEPARTMENT/CURRICULUM E-R Diagram



Under_Deg attribute represents the different requirements for undergraduate degrees. The Dual_Deg attribute represents the requirements for dual degrees.

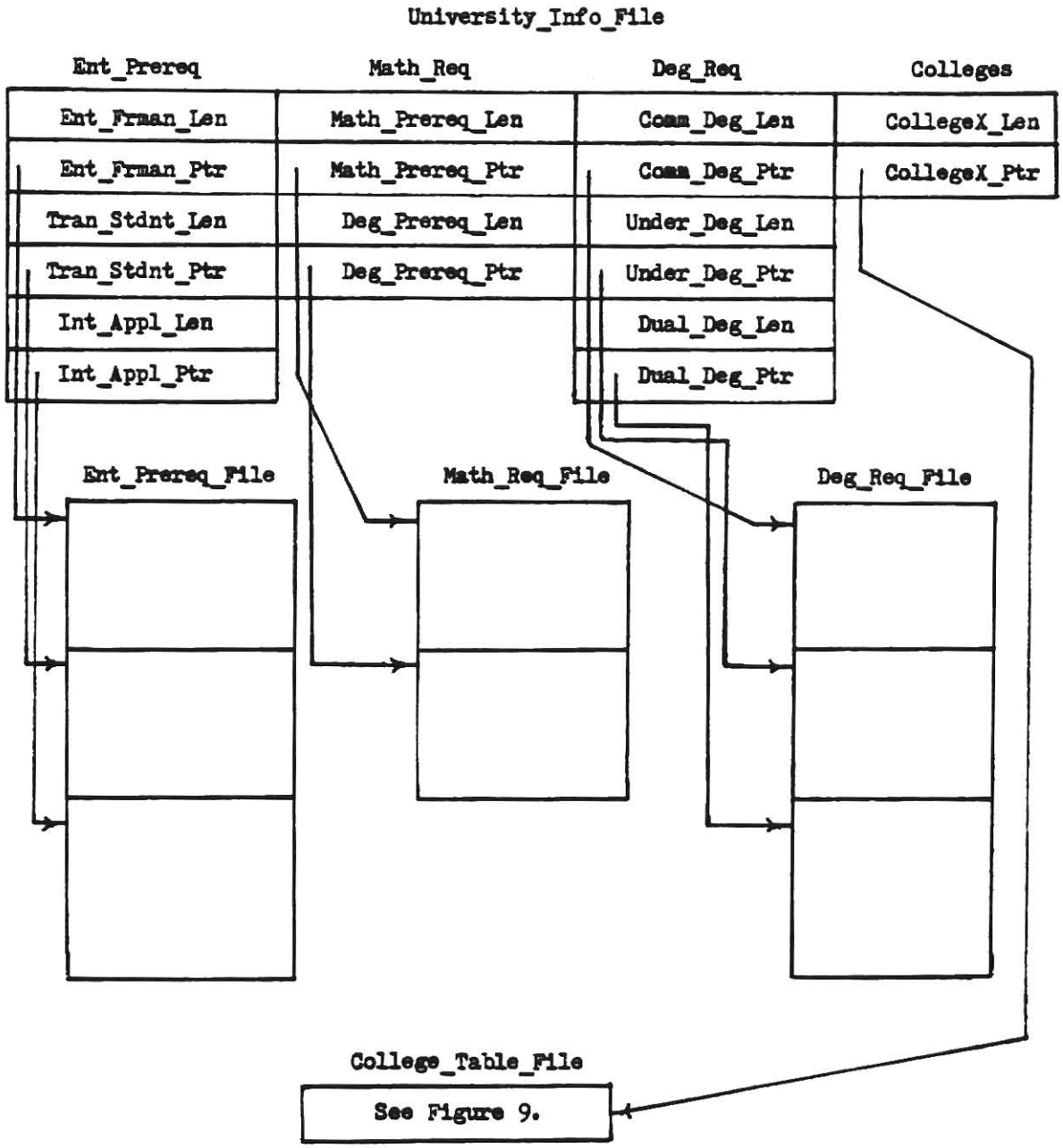
The Data Structure

After completing the E-R diagram which included the University entity, effort focused on developing the physical structure for the University category. The physical structure of the University category is provided in Figure 8. The overall structure of University is one record structure and three text files.

The record structure is named University_Info_File and contains four fields: Ent_Prereq, Math_Req, Deg_Req, and Colleges. The Ent_Prereq field is broken down into a subrecord structure with six fields. The six fields of Ent_Prereq are three pairs of length and pointer fields as follows: Ent_Frman_Len, Ent_Frman_Ptr, Tran_Stdnt_Len, Tran_Stdnt_Ptr, Int_Appl_Len, and Int_Appl_Ptr. Each length field indicates the length of the appropriate information appearing in a textual file name Ent_Prereq_File. Each pointer field indicates the amount of offset at which the appropriate information appears from the top of the Ent_Prereq_File.

The Math_Req field is broken down into a subrecord structure with four fields: Math_Prereq_Len, Math_Prereq_Ptr, Deg_Prereq_Len, and Deg_Prereq_Ptr. The four fields of Math_Req are two pairs of length and pointer fields which indicate appropriate information about math prerequisites and degree prerequisites contained in a text

Figure 8.
UNIVERSITY Data Structure



file named Math_Req_File.

The Deg_Req field is broken down into a subrecord structure with six fields: Comm_Deg_Len, Comm_Deg_Ptr, Under_Deg_Len, Under_Deg_Ptr, Dual_Deg_Len, and Dual_Deg_Ptr. The six fields of Deg_Req are three pairs of length and pointer fields which indicate appropriate information about degree requirements contained in a text file name Deg_Req_File.

The Colleges field is broken down into a subrecord structure with two fields: CollegeX_Len and CollegeX_Ptr. The CollegeX_Len field indicates the number of colleges appearing in a table named College_Table_File. The CollegeX_Ptr field indicates the locations of the table named College_Table_File.

DEVELOPMENT OF COLLEGE

The E-R Diagram

The College category was initially outlined into major headings in an effort to arrive at a suitable E-R diagram. See the appropriate portion of the E-R diagram provided in Figure 7. The Colleges entity represents the different colleges within the university. The College entity contains four attributes: G_Name, Degrees, Coll_Req, and Prog_Opt. The G_Name attribute represents the name of a particular college within the university. The Degrees attribute represents the different degrees which can be obtained from a given college within the university. The Coll_Req attribute represents the requirements that pertain to a

particular college. The Prog_Opt attribute represents different program options that are available within a given college.

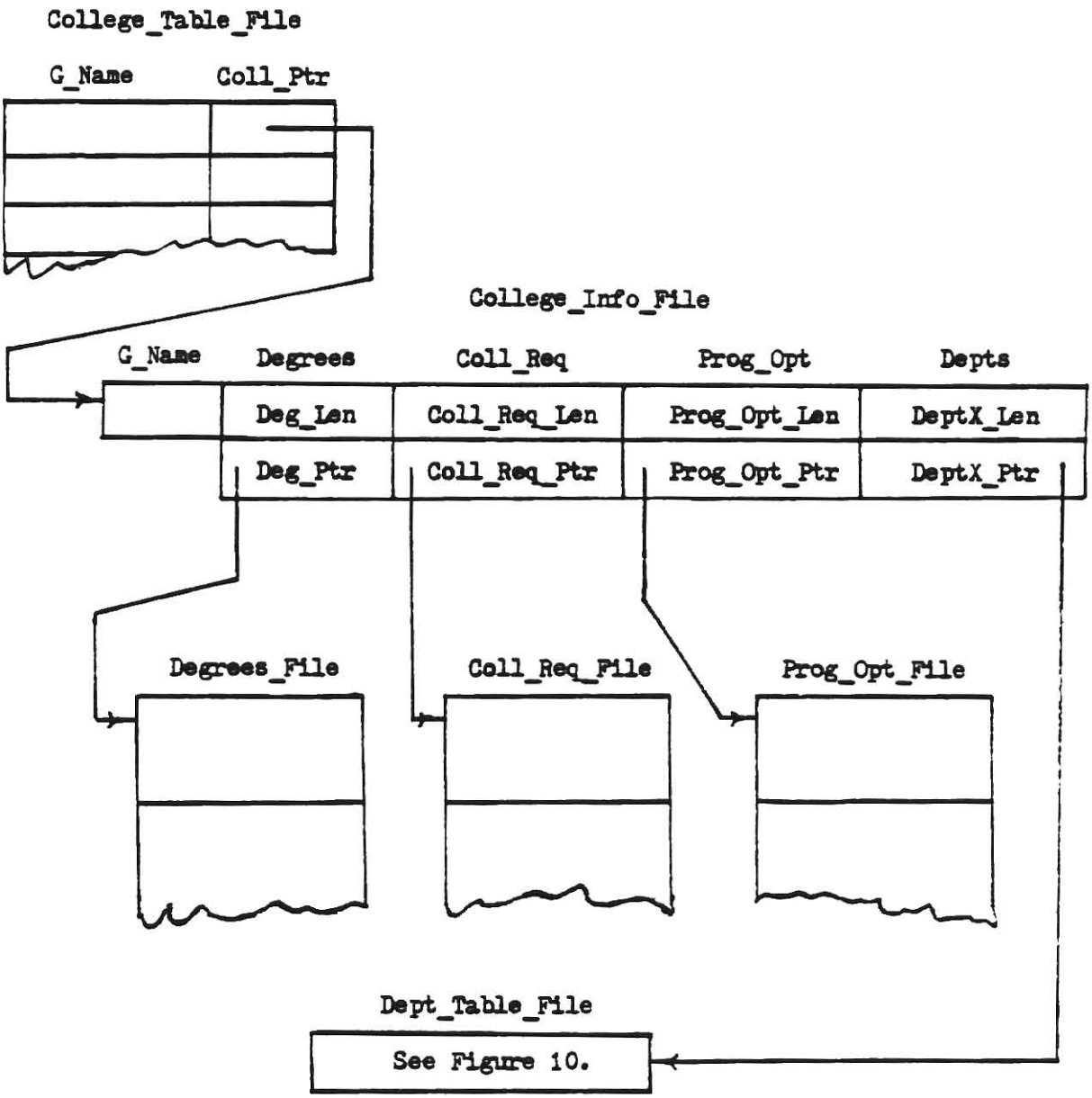
The Data Structure

After completing the E-R diagram which included the College entity, effort focused on development of an appropriate physical structure for this category. The physical structure of the College category is provided in Figure 9. The overall physical structure for the Course category consists of two record structures and three files as follows: College_Table_File, College_Info_File, Degrees_File, Coll_Req_File, and Prog_Opt_File.

The College_Table_File is a record structure which contains two fields: G_Name and Coll_Ptr. The G_Name field indicates the name of a particular college within the table. College names appear in this table in alphabetical order. The Coll_Ptr field indicates the amount of offset at which particular college information appears from the top of the College_Info_File.

The College_Info_File is a record structure which contains five fields: G_Name, Degrees, Coll_Req, Prog_Opt, and Depts. The G_Name field indicates the name of a given college contained in College_Info_File. The Degrees field is broken down into a subrecord structure with two fields: Deg_Len and Deg_Ptr. The Deg_Len field indicates the length of information about particular degree requirements contained in the Degrees_File. The Deg_Ptr field indicates the amount of offset at which particular degree requirements

Figure 9.
COLLEGE Data Structure



appear from the top of the Degrees_File.

The Coll_Req field is broken down into a subrecord structure with two fields: Coll_Req_Len and Coll_Req_Ptr. The Coll_Req_Len field indicates the length of information about particular college requirements contained in the Coll_Req_File. The Coll_Req_Ptr field indicates the amount of offset at which particular college requirements appear from the of the Coll_Req_File.

The Prog_Opt field is broken down into a subrecord structure with two fields: Prog_Opt_Len and Prog_Opt_Ptr. The Prog_Opt_Len field indicates the length of information about particular program options contained in the Prog_Opt_File. The Prog_Opt_Ptr field indicates the amount of offset at which particular program options appear from the top of the Prog_Opt_File.

The Depts field is broken down into a subrecord structure with two fields: DeptX_Len and DeptX_Ptr. The DeptX_Len field indicates the number of departments contained in a table named Dept_Table_File. The DeptX_Ptr field indicates the location of the table named Dept_Table_File.

DEVELOPMENT OF DEPARTMENT

The E-R Diagram

The Department category was initially outlined into major heading in an effort to arrive at a suitable E-R diagram. See the appropriate portion of the E-R diagram provided in Figure 7. The Depts entity represents the

different departments contained within a given college. The Depts entity contains four attributes: D_Name, Dept_Req, Grad_Req, and Cont_Per. The D_Name attribute represents the name of a particular department within a given college. The Dept_Req attribute represents the different requirements that exist for a given department. The Grad_Req attribute represents the requirements within a department for graduate students. The Cont_Per represents the name of a contact person for a given department.

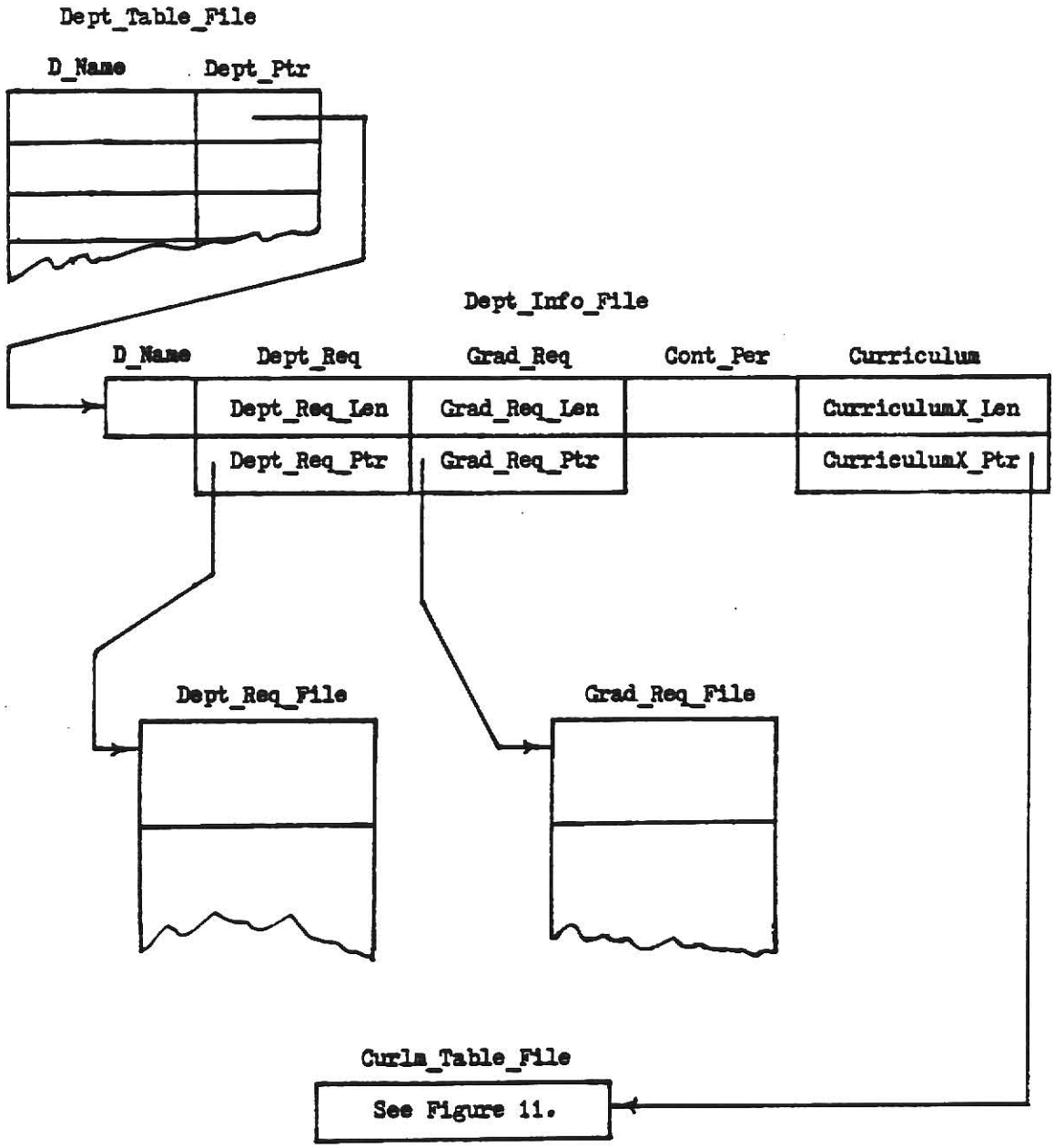
The Data Structure

After completingg the E-R diagram which included the Department entity, effort focused on the development of an appropriate physical structure for this category. The physical structure of the Department category is provided in Figure 10. The overall physical structure for the Department category consists of two record structures and two text files as follows: Dept_Table_File, Dept_Info_File, Dept_Req_File, and Grad_Req_File.

The table named Dept_Table_File is a record structure which contains two fields: D_Name and Dept_Ptr. The D_Name field indicates the name of the department in the table. Department names are placed in the table in an alphabetical order. The Dept_Ptr field indicates the amount of offset at which a particular department appears from the top of the Dept_Info_File.

The record structure Dept_Info_File consists of five fields: D_Name, Dept_Req, Grad_Req, Cont_Per, and Curriculum. The D_Name field indicates the name of a

Figure 10.
DEPARTMENT Data Structure



particular department within the Dept_Info_File. The Cont_Per field indicates the name of a contact person within a given department.

The Dept_Req field is broken down into a subrecord structure which contains two fields: Dept_Req_Len and Dept_Req_Ptr. The Dept_Req_Len field indicates the length of departmental requirements which appears in a text file named Dept_Req_File. The Dept_Req_Ptr field indicates the amount of offset at which particular department requirements appear from the top of Dept_Req_File.

The Grad_Req field is broken down into a subrecord structure which contains two fields: Grad_Req_Len and Grad_Req_Ptr. The Grad_Req_Len field indicates the length of graduate requirements information which appears in a text file named Grad_Req_File. The Grad_Req_Ptr field indicates the amount of offset at which particular graduate requirements information appears from the top of Grad_Req_File.

The Curriculum field is broken down into a subrecord structure which contains two fields: CurriculumX_Len and CurriculumX_Ptr. The CurriculumX_Len field indicates the number of different curriculas a particular department has to offer as contained in a table named Curlm_Table_File. The CurriculumX_Ptr field indicates the location of the table named Curlm_Table_File.

DEVELOPMENT OF CURRICULUM

The E-R Diagram

The Curriculum category was not initially outlined into major headings in order to arrive at a suitable E-R diagram. See the appropriate portion of the E-R diagram provided in Figure 7. The Curriculum entity represents the different curricula offered within a particular department. The Curriculum entity has two attributes: Curricula and Curr_Req. The Curricula attribute represents the different curricula in a given department. The Curr_Req attribute represents the particular requirements that must be met in a given curricula.

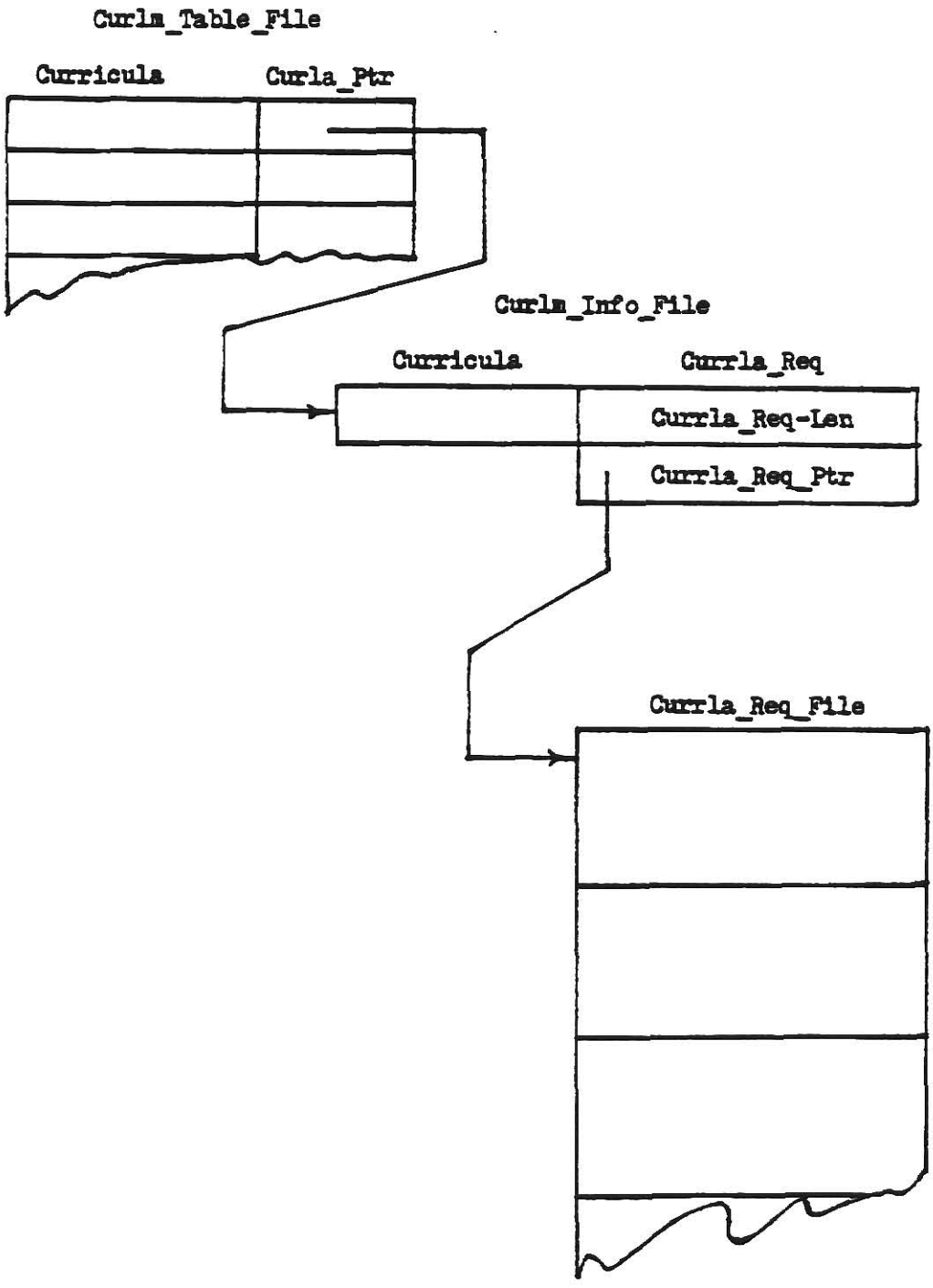
The Data Structure

After completing the E-R diagram which included the Curriculum entity, effort focused on the development of an appropriate physical structure for the Curriculum category. The physical structure of the Curriculum category is provided in Figure 11. The overall physical structure of the Curriculum category consists of two record structures and a text file as follows: Curlm_Table_File, Curlm_Info_File, and Currla_Req_File.

The table named Curlm_Table_File is a record structure which contains two fields: Curricula and Curla_Ptr. The Curricula field indicates the name of different curricula offered within a given department. The names of the different curricula appear in Curlm_Table_File in an alphabetical order. The Curla_Ptr field indicates the amount of offset at which particular curricula appears from the top of Curlm_Info_File.

The Curlm_Info_File is a record structure which

Figure 11.
CURRICULUM Data Structure



consists of two fields: Curricula and Currla_Req. The Curricula field is the name of the particular curricula offered within a given department. The Currla_Req field is broken down into a subrecord structure which consists of two fields: Currla_Req_Len and Currla_Req_Ptr. The Currla_Req_Len field indicates the length of the particular curricula requirements information appearing in a text file named Currla_Req_File. The Currla_Req_Ptr field indicates the amount of offset at which particular curricula requirements appear from the top of Currla_Req_File.

INTERACTION BETWEEN CATEGORIES

Basically the seven different categories comprise three different databases. The Student category can be thought of as one database. The Instructor and Course categories comprise the second database. The University, College, Department, and Curriculum categories comprise the third database. Since the Student category pertains to a particular student and no other students when a user accesses the Student Advisory System, it is feasible to think of the Student database as personal information maintained only on a student's own personal medium such as a floppy disc. The idea of the use of a floppy disc for the Student category information is sound for the sake of privacy. As the Course category interacts with the Instructor category, it is feasible to think of these two categories as one database. Since the University category interacts with the College, Department, and Curriculum categories, it is feasible to think of these four categories

as one database.

INTERACTION BETWEEN DATABASES

Basically the Student database will interact with each of the other two databases in the Student Advisory System. The Student database interacts with the other databases because it needs to maintain a history of all queries a given student performs with the system. Since there are no pointers from either the Course or Instructor data structures to the University, College, Department, or Curriculum data structures, it is not likely that any interaction will exist between these two databases.

FUTURE WORK

The next phase in the development of the Student Advisory System should deal with the implementation procedures associated with all the different files designed thus far. Procedures have been coded but not tested for the Course category. The coded procedures for the Course category are discussed in the Appendix.

As was discovered in writing programs for the Course category, extra data structures may have to be developed for accessing certain files in the system. At the preliminary design level of this system, it is rather difficult to ascertain whether any additional tables or files are needed other than the ones that have been provided thus far.

CONCLUSION

The seven different categories specified in the

requirements specification have been analyzed and developed into appropriate E-R diagrams and data structures. Data structure declarations for the different categories are provided in this report, but they are more appropriately presented in the Appendix. The technique used to develop the majority of the categories was to use an outlining approach to bridge the gap between having a requirements specification and creating an E-R diagram.

The data structures for each of the seven categories consist of an arrangement of flat files which model each of the different E-R diagrams. The different flat files essentially form three separate databases for the Student Advisory System. The only interaction between the databases is between the Student database and each of the other two databases.

BIBLIOGRAPHY

1. Cook, John L., Master's Report, "Graduate Student Records Relational Data Base Design," Kansas State University, 1982.

2. Kroenke, David M., Database Processing, Science Research Associates, Inc., A subsidiary of IBM, Second Edition, 1983.

3. Ullman, Jeffrey D., Principles of Database Systems, Computer Science Press, Inc., Second Edition, 1982.

APPENDIX A

COURSE STRUCTURE DECLARATIONS
COURSE PROCEDURES CODE
COURSE PROCEDURES DISCUSSION

```
COURSE STRUCTURE DECLARATIONS

struct course_table_file §
   char prefix [4];          /* represents course prefix */
   int ci_len;               /* represents number of lines */
   int ci_ptr;               /* represents offset pointer */
   †;
struct course_info_file §
   char crse_num [2];        /* represents course number */
   char c_name [29];         /* represents course name */
   struct class §
      int si_len;            /* represents number of lines */
      int si_ptr;            /* represents offset pointer */
      †;
   struct description §
      int di_len;            /* represents number of lines */
      int di_ptr;            /* represents offset pointer */
      †;
   char frequency;           /* s=spring, m=summer, f=fall */
   int credits;              /* represents number of hours */
   †;
struct schedule_info_file §
   int ref_num;              /* represents reference number */
   char type [2];            /* lec=lecture, rec=recitation,
                                lab=laboratory */
   char place [3];           /* building and room number */
   char days [4];            /* example is mtwhf */
   char time [8];            /* example is 1130-0130 */
   int prof_ptr;             /* represents offset pointer */
   int max_enroll;           /* represents maximum number */
   int enroll;               /* represents current number */
```

```
   †;
struct description_info_file §
   char di_text [19];          /* represents line of text */
   †;
struct descrip_table_file §
   char prefix [4];            /* represents course prefix */
   int crse_num;               /* represents course number */
   int di_len;                 /* represents number of lines */
   int di_ptr;                 /* represent offset pointer */
   †;
struct temp_descrip_file §
   char td_text [19];          /* represents line of text */
   †;
```

COURSE PROCEDURES CODE

```
get_course(dept,c_num);
    /* this function finds the location of the course number
*/
  §
    int num_byte; /* variable for the read */
    int c_num; /* parameter */
    char dept[4]; /* parameter */
    course_table ct_rec; /* represents course table file
record */
    course_info ci_rec; /* represents course info file
record */
    char prefix[4]; /* the department in the course table
file */
    int number; /* the course number */
    int offset; /* the offset into the files */
    int lb,ub,mp; /* binary search operators */
    /* perform a search on the course table for the
department */
    lb=0;
    ub=0;
    while(lb<=ub)§
      lseek(ct_file,0,0);
      mp=(lb+ub)/2;
      lseek(ct_file,mp*sizeof(ct_rec),0);
      num_byte=read(ct_file,ct_rec,sizeof(ct_rec));
      if(dept < ct_rec.prefix)
        ub=mp-1;
      else
```

```
        if(dept>ct_rec.prefix)
          lb=mp+1;
        else
          lb=ub+1;
    †
  /* test to see if the proper record was found */
    if (dept=ct_rec.prefix) §
      if (number != 0) §
        /* perform a binary search  on  the course info file
*/
        lb=ct_rec.ci_ptr; /* init the variables */
        ub= lb+ct_rec.ci_length;
        /* begin searching */
        while(lb<= ub)§
          lseek(ci_file,lb*sizeof(ci_rec),0);
          mp=(lb+ub)/2;
          lseek(ci_file,mp*sizeof(ci_rec),0);
          num_byte=read(ci_file,ci_rec,sizeof(ci_rec));
          if(c_num < ci_rec.course_num)
            ub=mp-1;
          else
            if(c_num > ci_rec.course_num)
              lb=mp+1;
            else
              lb=ub+1;
        † /* end while */
      † /* end if */
    † /* end if */
  offset=mp*sizeof(ci_rec);
```

```
  return(offset); † /* function */
  copy_description(dept,c_num);
  /* Perform linear search on the description table */
    int c_num,num_byte;
    char dept[4];
   §
   course_table table_rec;
    numbyte = read(desc_info,table_rec,sizeof(table_rec));
    while((table_rec.prefix != dept) && (numbyte)) §
      numbyte = read(desc_info,table_rec,sizeof(table_rec));
      if (numbyte != 0) §
      /* Perform a linear search o the course numbers */
        while((table_rec.course_num != c_num) &&  (numbyte))
§
          numbyte                                          =
read(desc_info,table_rec,sizeof(desc_rec));
          if (numbyte != 0) §
            offset := table_rec.ptr;
            num_rec := table_rec.length;
            /* Transfer the records to the new file */
            lseek(desc_info,(num_rec*sizeof(desc_rec),0);
            while(num_rec != 0) §
              num_byte                                     =
read(desc_file,desc_array,sizeof(desc_array));
              write(tem_desc_file,desc_array,sizeof(desc_array));
              num_rec = num_rec - 1;
            †
          †
        †
```

```
      †
    †
    table_rec.offset = tell(tem_desc_file);
    lseek(desc_info,-sizeof(table_rec),1);
    write(desc_info,table_rec,sizeof(table_rec));
    return(num_byte); /* a do nothing statement */
  † table_add(dept,c_num) /* this function adds a new course
to the table that  */  /*  holds the info to the description
file            */
  §
  char dept[4];
  int c_num; /* parameter */
  struct table_rec §
       char prefix[4];
       int course_num;
       int length;
       int offset;
  †
    table_rec rec1,rec2;
    int numbyte;
    l_seek(desc_table,0,0);
  /* find the position of the new entry */
    numbyte= read(desc_table,rec1,sizeof(rec1));
    while((rec1.prefix != dept) && (numbyte));
      numbyte = read(desc_table,rec1,sizeof(rec1));
  /* insert the record into the table */
    rec2.prefix = dept;
    rec2.course_num = c_num;
    rec2.lenth = 0;
```

```
    rec2.offset = 0;
    /* position the file pointer to the previous record */
    lseek(desc_table,-sizeof(rec2),1);
    while(num_byte != 0) §
      write(desc_table,rec2,sizeof(rec2));
      rec2.prefix = rec1.prefix;
      rec2.c_num = rec1.c_num;
      rec2.lenth = rec1.lenth;
      rec2.offset = rec1.offset;
      numbyte = read(desc_table,rec1,sizeof(rec1));
  †
  return(num_byte) /* a do nothing statement */
 † new_description(dept,c_num,desc_line)
  /* this procedure replaces the description of the
department */
  /* and the course number in the description file
*/
 §
  char dept[4];
  int c_num,num_byte;
  char desc_line[19];
  char line[19];
  table_rec rec1;
  /* if c num <> o then don't search for the c_num */
  if (c_num != 0 ) §
    /* init the pointers in the files used */
    lseek(desc_table,0,0);
    lseek(tem_desc_file,0,2);
    /* find the position in the description table */
```

```
    numbyte = read(desc_table,recl,sizeof(recl));
    while(recl.prefix != dept) && (recl.course_num != c-num)
      num_byte= read(desc_table,recl,sizeof(recl));
    recl.length=0;
    recl.offset = 0;
    position=(1/(sizeof(recl)) * tell(tem_desc_file);
  † /* end if */ /* place the contents of the array into the
file */
  write(tem_desc_file,desc_line,sizeof(desc_line));
  lseek(desc_table,-sizeof(recl),1);
  num_byte=read(desc_table_file,recl,sizeof(recl));
  recl.length = recl.length + 1;
  lseek(desc_table,-sizeof(recl),1);
  write(desc_table,recl,sizeof(recl));
  return(num_byte); /* a do nothing statement */ †
  table_delete(dept,c_num)
  /* This function deletes a table entry */
  §
    struct table_rec §
        char prefix[4];
        int course_num;
        int length;
        int offset;
      †
    char dept[4]; /* the department */
    int c_num; /* the course number */
    table_rec recl;,
    int numbyte;
  /* Find the position of the entry to be deleted */
```

```
    lseek(desc_table,0,0);
    numbyte = read(desc_table,recl,sizeof(recl));
    while(recl.prefix != dept)
      numbyte = read(desc_table,recl,sizeof(recl));
    while((recl.prefix  !=  dept)  &&  (recl.course_num   !=
c_num))
      numbyte = read(desc_table,rec,sizeof(recl))
    lseek(desc_table,-sizeof(recl)*2,1)
    while(numbyte != 0) §
      write(desc_table,recl,sizeof(recl));
      numbyte= read(desc_table,recl,sizeof(recl));
      numbyte= read(desc_table,recl,sizeof(recl));
      lseek(desc_table,-(sizeof(recl)*2),1);
    †
   return(numbyte); /* a do nothing statement */
  †
```

COURSE PROCEDURES DISCUSSION

The next phase in the development of the course category addresses implementation level procedures. At first glance, several different programming requirements were considered. The initial idea was to address the accessing of a course, the inserting and deleting of courses, the inserting and deleting of descriptions, the inserting and deleting of schedules, and description updating. After considerable thought and discussion, it was finally resolved that a yearly update routine would have to be written by an applications programmer which would address course insertion and course deletion. An application programmer will also have to write a semester update routine which handles schedule deletion and schedule insertion. The implementation level programs will address accessing a course as well as description information updating.

Actual code has been written which deals with accessing a particular course and updating description information. A function entitled Get_Course deals with a user's request to obtain a particular record related to a given department and course number. Four functions entitled Copy_Description, Table_Add, New_Description, and Table_Delete deal with the necessity to updating course description information. These functions are discussed in more detail in the following paragraphs.

The Get_Course function uses two input parameters. It uses a parameter named Dept which is used in a binary search on the Course_Table_File. The binary search of the Course_Table_File looks for the appropriate department in

order to get to the right area of course numbers in the Course_Info_File. The binary search for Dept yields an offset value which points to the appropriate area of records in the Course_Info_File. The Get_Course function also uses a parameter named C_Num which is used in another binary search on the Course_Info_File. The binary search of the Course_Info_File looks for the appropriate course number under a given department. The binary search with C_Num returns an offset value which indicates the exact record in the Course_Info_File that pertains to a given department and course number. Thus Get_Course consists of two binary searches: One binary search with Dept on the Course_Table_File and another binary search with C_Num on the Course_Info_File. Function Get_Course returns an offset value (displacement from the top of the file) for the location of a desired course number within a given department. Note: There is a special circumstance regarding the input parameter C_Num. If C_Num equals zero, the binary search for a particular course number is bypassed. This feature allows the user to scan a number of different course number records within a given department.

Before actual code was written for the updating of description information, the question arose as to how to access the description information in the Descrip_Info_File. The only means of accessing this file to this point in time is through the appropriate fields of the Course_Info_File associated with each course number. This was deemed as extremely awkward, and an alternative approach was developed which includes another table. This latest table is named

Descrip_Table_File, and it was designed solely for updating description information. In addition, a temporary description file is also used for the same updating purpose. The temporary description file is named Temp_Descrip_File.

The Descrip_Table_File consists of a structure with four fields. The fields are as follows: Prefix, Course_Num, DI_Len, and DI_Ptr. The Prefix field represents a department. The Course_Num field represents a course number. The DI_Len field represents the length of a course description in the Descrip_Info_File. The DI_Ptr field represents the offset to a given description in the Descrip_Info_File.

The main purpose of the Descrip_Table_File is for indicating where descriptions are contained in the Descrip_Info_File. As new descriptions are added or old descriptions are deleted, the Descrip_Table_File must be modified accordingly. This brings up the reason for a Temp_Descrip_File. The Temp_Descrip_File is used to temporarily hold updated description information. If an old description is valid, it is copied into the Temp_Descrip_File from the Descrip_Info_File. If a new description is desired, a new description is entered into the Temp_Descrip_File a line at a time. As new descriptions are entered into the Temp_Descrip_File, modifications must be made on the DI_Len and DI_Ptr fields of the Descrip_Table_File to account for the new information. Once all course descriptions are properly placed in the Temp_Descrip_File, it is renamed to replace the old Descrip_Info_File.

As mentioned earlier, four functions address the description information update. Each of these functions work with the Descrip_Info_File, Descrip_Table_File, and Temp_Descrip_File. All four functions are used only as procedures. They return a value for Num_Byte, but it is not meant to be used for anything purposeful.

The Copy_Description function is used to copy a valid course description from the Descrip_Info_File into the Temp_Descrip_File. This function uses two input parameters named Dept and C_Num. The parameters are used with a linear search on the Descrip_Table_File. The linear search is used to find the location of the description from the Descrip_Table_File. This function then copies the valid course description pointed at in the Descrip_Info_File. It copies the course description one line at a time. It then updates the DI_Ptr field in the Descrip_Table_File.

The Table_Add function is used in the event that a new course must be added to the courses already in the system. This function uses two input parameters named Dept and C_Num. When desireable to add a new department and course number, the new information is inserted in front of the first course number for a given department in the Descrip_Table_File. In order to do this insertion, every successive entry must be moved down one slot in the Descrip_Table_File. This function will always be used when adding a new description to the Temp_Descrip_File.

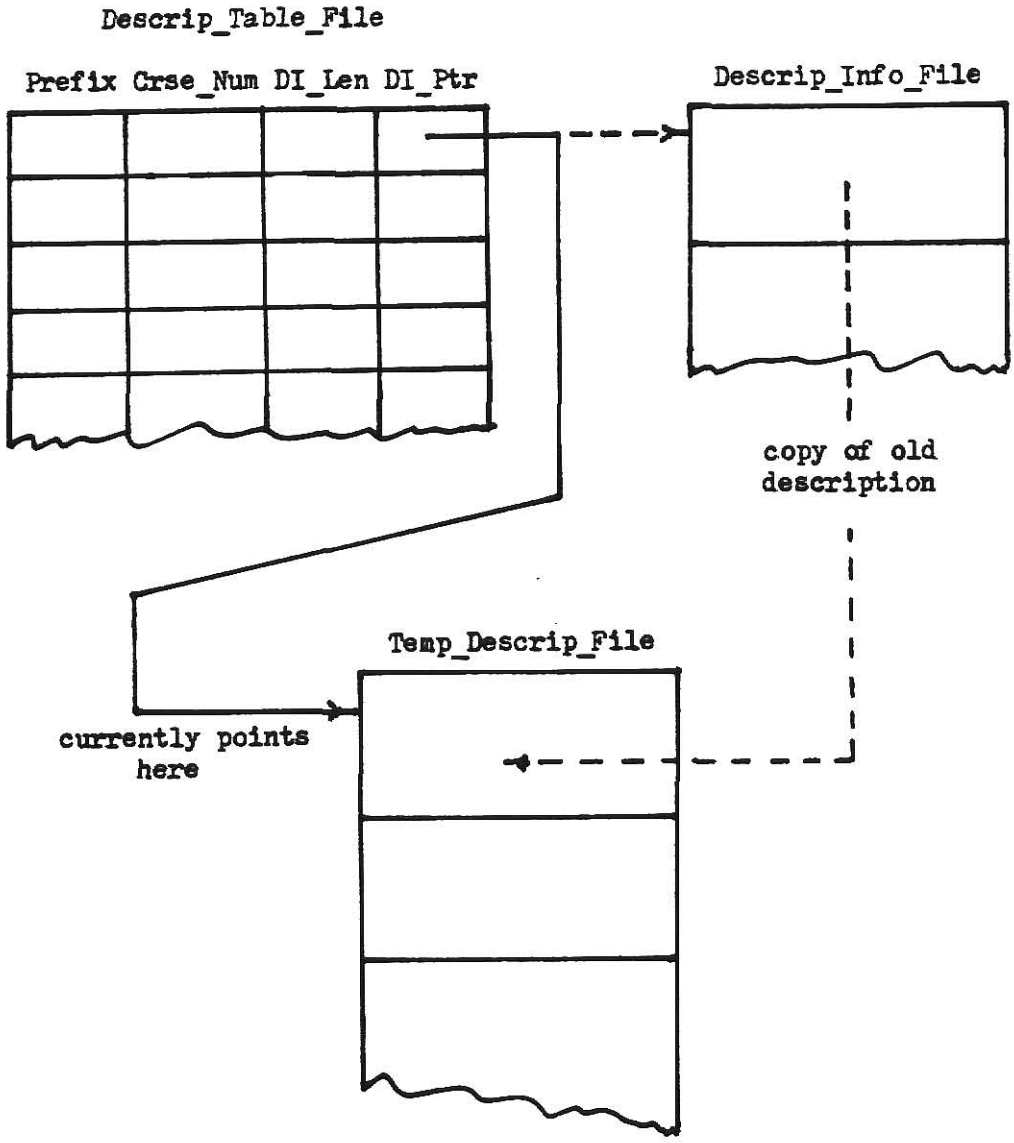
The New_Description function is used to add a new description to the Temp_Descrip_File. This function uses three input parameters named Dept, C_Num, and Desc_Line.

The parameter Desc_Line is an array of twenty characters used when creating a line of description for the generation of a new description. The New_Description function then adds a line of new description information to the Temp_Descrip_File. The New_Description function also performs a linear search on the Descrip_Table_File for the purpose of locating where to update the DI_Len and DI_Ptr fields that are pertinent to a new description entry. The input parameter C_Num is used to indicate how to modify the DI_Len and DI_Ptr fields of the Descrip_Table_File. When first entering a new description for a given course number, C_Num will be greater than zero. When C_Num is greater than zero and one new line of course description has been entered into the Temp_Descrip_File, DI_Len is assigned a value of one and DI_Ptr is assigned the offset value which points to the beginning of the new description. When C_Num is equal to zero and another new line of course description has been entered into the Temp_Descrip_File, DI_Len is incremented by one for each successive entry but nothing more is done to DI_Ptr. In this manner the DI_Len field of the Descrip_Table_File for the new course will indicate the appropriate length of the entire new course description.

The Table_Delete function is used when desireable to delete a course description from the Descrip_Info_File. This function uses two input parameters named Dept and C_Num. It uses a linear search with Dept and C_Num on the Descrip_Table_File in order to find the entry to delete. Once an entry has been deleted, it then compacts the remaining entries in the Descrip_Table_File by moving the

rest of the entries up one slot.

Figure A-1.
COURSE Data Structure Supplement



APPENDIX B

STUDENT STRUCTURE DECLARATIONS

STUDENT STRUCTURE DECLARATIONS

```
struct student_info_file §

   char s_name [29];          /* represents student name */

   int ssn [8];               /* represents social security
                                 number */
   int past_trans_ptr;        /* represents offset pointer */

   struct standing §

      int hs_gpa [2];         /* represents high school grade
                                 point average */
      int act_score [4];      /* represents American College
                                 Test score */
      int yr_in_school [3];   /* represents year in school */

      int curr_gpa [4];       /* represents current GPA */

      int hrs_obt [2];        /* represents hours obtained */

      †;

   struct intentions §

      int entry_date [5];     /* represents college entrance
                                 date */
      char dec_major [5];     /* represents declared major */

      char dec_minor [5];     /* represents declared minor */

      †;

   struct advisors §

      char major_prof [29];   /* represents major professor */

      char comm_memb1 [29];   /* represents committee member */

      char comm_memb2 [29];   /* represents committee member */

      char comm_memb3 [29];   /* represents committee member */

      char comm_memb4 [29]    /* represents committee member */

      †;

   struct coursework §

      int tech_elects_len;    /* represents number of lines */

      int tech_elects_ptr;    /* represents offset pointer */

      int crses_taken_len;    /* represents number of lines */

      int crses_taken_ptr;    /* represents offset pointer */
```

```
      int trans_creds_len;      /* represents number of lines */
      int trans_creds_ptr;      /* represents offset pointer */
      int tent_curr_len;        /* represents number of lines */
      int tent_curr_ptr;        /* represents offset pointer */
      };
   struct s_personal {
      int interests_len;        /* represents number of lines */
      int interests_ptr;        /* represents offset pointer */
      int goals_len;            /* represents number of lines */
      int goals_ptr;            /* represents offset pointer */
      int prefs_len;            /* represents number of lines */
      int prefs_ptr;            /* represents offset pointer */
      int str/wkn_len;          /* represents number of lines */
      int str/wkn_ptr;          /* represents offset pointer */
      int poor_times_len;       /* represents number of lines */
      int poor_times_ptr;       /* represents offset pointer */
      };
   };
struct past_trans_file {
   char pt_text [19];           /* represents line of text */
   };
struct coursework_file {
   char cw_text [19];           /* represents line of text */
   };
struct s_personal_file {
   char sp_text [19];           /* represents line of text */
   };
```

APPENDIX C

INSTRUCTOR STRUCTURE DECLARATIONS

```
INSTRUCTOR STRUCTURE DECLARATIONS

struct instructor_table_file §
   char i_name [29];            /* represents instructor name */
   int instr_ptr;               /* represents offset pointer */
   †;
struct instructor_info_file §
   char i_name [29];            /* represents instructor name */
   char dept [4];               /* represents department name */
   struct specialty §
      char major_area [19];     /* represents major specialty */
      char crses_taught [39];   /* represents course taught */
      †;
   struct status §
      char type_appt [9];       /* represents type of appointment */
      char grad_facty [2];      /* yes means graduate faculty */
      char eval_rcd [9];        /* represents evaluation record */
      †;
   struct i_personal §
      int strgths_len;          /* represents number of lines */
      int strgths_ptr;          /* represents offset pointer */
      int wkness_len;           /* represents number of lines */
      int wkness_ptr;           /* represents offset pointer */
      †;
   †;
struct i_personal_file §
   char ip_text [19];           /* represents line of text */
   †;
```

APPENDIX D

UNIVERSITY STRUCTURE DECLARATIONS

UNIVERSITY STRUCTURE DECLARATIONS

```
struct university_info_file §
   struct ent_prereq §
      int ent_frman_len;      /* represents number of lines */
      int ent_frman_ptr;      /* represents offset pointer */
      int tran_stdnt_len;     /* represents number of lines */
      int tran_stdnt_ptr;     /* represents offset pointer */
      int int_appl_len;       /* represents number of lines */
      int int_appl_ptr;       /* represents offset pointer */
      †;
   struct math_req §
      int math_prereq_len;    /* represents number of lines */
      int math_prereq_ptr;    /* represents offset pointer */
      int deg_prereq_len;     /* represents number of lines */
      int deg_prereq_ptr;     /* represents offset pointer */
      †;
   struct deg_req §
      int comm_deg_len;       /* represents number of lines */
      int comm_deg_ptr;       /* represents offset pointer */
      int under_deg_len;      /* represents number of lines */
      int under_deg_ptr;      /* represents offset pointer */
      int dual_deg_len;       /* represents number of lines */
      int dual_deg_ptr;       /* represents offset pointer */
      †;
   struct colleges §
      int collegex_len;       /* represents number of lines */
      int collegex_ptr;       /* represents offset pointer */
      †;
   †;
```

```
struct ent_prereq_file §
   char ep_text [19];          /* represents line of text */
   †;
struct math_req_file §
   char mr_text [19];          /* represents line of text */
   †;
struct deg_req_file §
   char dr_text [19];          /* represents line of text */
   †;
```

APPENDIX E

COLLEGE STRUCTURE DECLARATIONS

```
COLLEGE STRUCTURE DECLARATIONS

struct college_table_file §
   char g_name [29];          /* represents college name */
   int coll_ptr;              /* represents offset pointer */
   †;
struct college_info_file §
   char g_name [29];          /* represents college name */
   struct degrees §
      int deg_len;            /* represents number of lines */
      int deg_ptr;            /* represents offset pointer */
      †;
   struct coll_req §
      int coll_req_len;       /* represents number of lines */
      int coll_req_ptr;       /* represents offset pointer */
      †;
   struct prog_opt §
      int prog_opt_len;       /* represents number of lines */
      int prog_opt_ptr;       /* represents offset pointer */
      †;
   struct depts §
      int deptx_len;          /* represents number of lines */
      int deptx_ptr;          /* represents offset pointer */
      †;
   †;
struct degrees_file §
   char df_text [19];         /* represents line of text */
   †;
struct coll_req_file §
   cSOR YR_text [19];        /* represents line of text */
```

```
    †;
struct prog_opt_file §
    char po_text [19];          /* represents line of text */
    †;
```

APPENDIX F

DEPARTMENT STRUCTURE DECLARATIONS

DEPARTMENT STRUCTURE DECLARATIONS

```
struct dept_table_file §
   char d_name [29];          /* represents department name */
   int dept_ptr;              /* represents offset pointer */
   †;
struct dept_info_file §
   char d_name [29];          /* represents department name */
   struct dept_req §
      int dept_req_len;       /* represents number of lines */
      int dept_req_ptr;       /* represents offset pointer */
      †;
   struct grad_req §
      int grad_req_len;       /* represents number of lines */
      int grad_req_ptr;       /* represents offset pointer */
      †;
   char cont_per [29];        /* represents contact person */
   struct curriculum §
      int curriculumx_len;    /* represents number of lines */
      int curriculumx_ptr;    /* represents offset pointer */
      †;
   †;
struct dept_req_file §
   char dr_text [19];         /* represents line of text */
   †;
struct grad_req_file §
   char gr_test [19];         /* represents line of text */
   †;
```

APPENDIX G

CURRICULUM STRUCTURE DECLARATIONS

CURRICULUM STRUCTURE DECLARATIONS

```
struct curlm_table_file §
   char curricula [9];          /* represents name of curricula */
   int curla_ptr;               /* represents offset pointer */
   †;
struct curlm_info_file §
   char curricula [9];          /* represents name of curricula */
   struct currla_req §
      int currla_req_len;       /* represents number of lines */
      int currla_req_ptr;       /* represents offset pointer */
      †;
   †;
struct currla_req_file §
   char cr_text [19];           /* represents line of text */
   †;
```

THE PRELIMINARY DESIGN OF A STUDENT ADVISORY SYSTEM

by

RONALD J. VIETH

B.S., Kansas State University, 1979
M.S., Kansas State University, 1984

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

ABSTRACT

The scope of this report is to develope and document the preliminary design of an automated Student Advisory System. The Student Advisory System is to support information pertinent to seven major categories: Course, Student, Instructor, University, College, Department, and Curriculum.

The preliminary design of the Student Advisory System entails mapping the requirements specifications for each of the categories into Entity-Relationship diagrams, data structure diagrams, and the appropriate structure declarations.

The report documents work which began with this project in the spring semester of 1984. The 1984 spring semester work dealt with one category of the seven different categories. The report goes on to develope the other six categories and documents the results of this effort.