Academic Advisor Assistant

by

Dae-Chul Sohn

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

May 1995

Author [illegible]
Department of Electrical Engineering and Computer Science
May 26, 1995

Certified by [illegible]
Gill Pratt
Thesis Supervisor

Accepted by [illegible]
F.R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Academic Advisor Assistant

by
Dae-Chul Sohn

Submitted to the
Department of Electrical Engineering and Computer Science

May 25, 1995

in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

The Academic Advisor Assistant (AAA) is a system designed to help MIT students and academic advisors. In its current form, it performs a complete and detailed audit of the student's status in the pursuit of his degree, as well as generate a list of courses from which the student can make his selection for the upcoming term's schedule. If nothing else, AAA saves students and advisors from spending time at the end of each semester flipping through the Course Bulletin. AAA is a X-application, and the user interface has been designed with the non-EECS in mind.

Chapter 1 : Introduction

This chapter introduces the Academic Advisor Assistant (AAA)[1] by describing the disease[2] which the system intends to cure, illustrating the critical observation which makes the cure possible, and then briefly describing the cure as envisioned at the project's inception.

I. The Disease

At the end of each semester, MIT students take invaluable time away from their overwhelming end-of-term chores to complete a task known as pre-registration, in which they decide what courses to take in the upcoming term. The typical MIT student's approach to this task may be portrayed as a three-stage process.[3] First, the student must determine what courses he[4] *can* take, considering the offering of courses and prerequisites he has satisfied. He must then select a subset of those courses, considering which he *needs* to complete his degree requirements and which he *wants* to satisfy his intellectual curiosity or some other personal criteria. Finally, he must consult the class schedule to identify any time conflicts among his selections. If any are found, then he must go back and select different courses to resolve the conflict.

This task is conceptually simple. In reality, however, three factors make the task rather difficult. First, there are a large number of courses listed in the *Bulletin*[5], 3070 to be exact[6], and it is simply infeasible for anyone to consider all of them in making his selection. On the one hand, this difficulty is exaggerated since students will generally choose courses from their respective departments. Even still, the number of courses may be large; department of EECS[7], for example, lists 179 subjects. On the other hand, the difficulty is understated, since courses are inter-dependent via prerequisites and corequisites, such that they cannot all be independently considered.

Second, degree requirements can be somewhat complex, such that many students often are uncertain about their status in completing them. One of the best illustrations of this complexity can be found in the description of the degree requirements for VI-P, Master of Engineering in EECS:

> *No* subject can be counted *both* as part of the 17-subject GIRs *and* as part of the 285 units required beyond the GIRs. *Every* subject in the student's department program will count toward one or the other, but not *both*. Similarly, no single subject may be counted in more than one category of departmental restricted electives. (Bulletin, p. 161)

Third, the offering of couses varies from term to term, because some courses are offered only in alternate terms and some are offered only in alternate years.[8] Therefore, one must consider the consequences of choosing a particular bag of courses in the upcoming term on all of his future terms. For instance, if a student wants to take a particular course which is going to be offered

1. Not to be confused with the automobile service company.
2. An inconsequential allusion to a certain movie starring Sylvester Stallone.
3. This author boldly speaks for the typical MIT student here and elsewhere.
4. For sake of clear exposition, the masuline pronoun will be used to denote a generic third person in this paper.
5. The *Bulletin* referenced here and elsewhere in this paper is the 1994-95 edition.
6. This number does not include the SWE (Engineering School-Wide Electives), since they are already included in other sections of the *Bulletin*. Even without the SWE courses, this number (3070) includes a small amount of double counting, because of courses which have multiple course numbers. For example, 6.046J is the same subject as 18.410J, but both are listed.
7. Electrical Engineering and Computer Science.
8. Generally, courses in this latter group are also offered in only one of the terms (fall or spring). For example, 6.633 was offered only for the second term of the 1994-95 school year, and not offered at all in the 1995-96 school year.

three terms from present at the earliest, then he must be sure to satisfy all of that course's prerequisites in the meantime.

From a positive perspective, these difficulties, the large quantity of available courses, the complexity of degree requirements, and the time variability of course offerings, underscore the flexibility and breadth of course offerings at MIT, but from the perspective of the under-nourished and under-rested student, they can be a great source of stress, and make the road to graduation seem like a rocky terrain[9].

Fortunately, the student is not without help. Each student has an academic advisor who is usually a professor in the student's department and therefore generally much more knowledgeable about how one can achieve a certain set of academic goals within that department. Unfortunately, this advisor too faces the aforementioned difficulties. If the offering of courses never changed, and if degree requirements also remained constant, an advisor may be able to surmount the difficulties through several years of experience and the accompanied knowledge acquisition, but neither of those conditions are true.

II. The Observation

Underlying the difficulties is the somewhat surprising fact that contrary to popular belief, MIT students and advisors *are* normal people, and they therefore have a limited capacity for simultaneously considering a large number of interdependent things, if not in the engineering domain, certainly in the everyday one of paperwork.

Now consider Mr. Turing Machine, a hypothetical MIT student whose name just happens to coincide with the celebrated computation device, and who also happens to possess processing capability which also coincides with that of the device. Turing can complete pre-registration using the following algorithm:

1. Iterate over all the courses in the *Bulletin*, marking each course whose prequisites he has satisfied, and which is being offered in the upcoming term.
2. Iterate over all the marked courses, considering how each one would further his progress in the pursuit of his academic goals, as well as how it scores against various personal criteria.
3. Based on the scoring, select the best bag *B* of courses. The *Course Evaluation Guide* can provide useful information for this process.
4. Check *B* for temporal conflicts using the *Class Schedules*. If any conflict is found, go back to step 3 above and choose a different bag. Else, done.

Now suppose Turing was everyone's friend and willing to help out with everyone's preregistration. That should make things easier.

III. The Cure

Given a particular student's course history and digestible copies of the *Bulletin* and *Class Schedules*, even without resorting to any state-of-the-art Artificial Intelligence technology, a computer system could perform steps 1 and 4 above, since they basically involve mindless iteratation over data. A system could also perform the first part of step 2, and it could perform the latter part too if the student could specify his personal criteria in some form digestible to the system.

9. An inconsequential allusion to *Princess Bride*, an excellent movie which the reader *must* see!

So then, what remains is step 3. How can the system select the "best" bag? For example, for a freshman student just starting out in Course 6, how can the system determine that a bag containing both 6.001 and 6.002 is somehow worse than a bag containing only one of them, despite the fact that the former would result in greater progress in his degree program?

Obviously, the system could never deduce this without having the knowledge that 6.001 and 6.002 are both heavily time-consuming courses of significant difficulty. This knowledge could be expressed in several ways. First, we may express it descriptively as:

```
6.001 is TIME-CONSUMING and DIFFICULT
6.002 is TIME-CONSUMING and DIFFICULT
```

From this, the system could deduce that taking the two classes at once would be a bad idea. Alternatively, we may also represent it a negative imperative:

```
do not take (6.001, 6.002) together
```

Yet another way would be to represent it using conditionals:

```
IF 6.001 IN BAG, DO NOT ADD 6.002
IF 6.002 IN BAG, DO NOT ADD 6.001
```

Given the resources of the *Bulletin*, the *Class Schedule*, personal criteria, and knowledge for ranking bags of courses, then, a system could do virtually all of the dirty work entailed in pre-registration. This is the idea behind AAA. After all, why make brilliant MIT students spend time flipping pages, when they do not have enough time to eat and sleep, and when a stupid computer program could do it much faster?

The following illuminating example should help the reader understand AAA better. Suppose a student is currently a freshman pursuing a VI-P degree (Master of Engineering in EECS), that he has taken 8.01, 18.01, 3.091, and 7.012, and that he needs to pre-register for his second term, in which he does not want to take more than four courses. In absence of any knowledge, the system may find that the student can choose from hundreds or even thousands of good bags of courses. But suppose that we had the following rules in the knowledge base:

```
TAKE GIRs AS EARLY AS POSSIBLE
ONE HASS A TERM KEEPS THE DOCTOR AWAY
IF ((DEGREE_DESIRED = VI-P) AND (NOT TAKEN 6.001))
   ADD 6.001 TO THE BAG
```

Even with just these two rules, the system may deduce that the optimal bag is {8.02, 18.02, 6.001, HASS}. A drastic improvement from having to choose from hundreds or thousands.

Chapter 2 : Design

This chapter describes the design of AAA, beginning with a presentation of some fundamental design principles used to guide the entire design phase, followed by an overview of the entire system. Finally, the strategy for generating schedules is described.

I. Guiding Principles

The design of AAA was guided by several principles.

First, it was assumed that the Registrar's Office would not collaborate actively to boost either the functionality or the efficiency of AAA. One implication of this assumption was that AAA would have to process the *Bulletin* and the *Class Schedule* in whatever format they existed. Since nothing short of full-fledged language recognition could accomplish this[10], the second-best solution of implementing the parsing components of AAA to be as easy to modify as possible was followed.

Second, since not all students at MIT are as comfortable with computers as the Course 6 majors, the user interface would have to be very easy to use.

Third, because each degree program can have very particular requirements, AAA would have to be designed such that degree programs can be indepently added and removed, and that each program can be designed indepently of all others.

II. Overview of AAA

The figure below shows an overview of the entire system. The three boxes on the left represent the inputs to the system, and the two boxes on the right show the outputs.

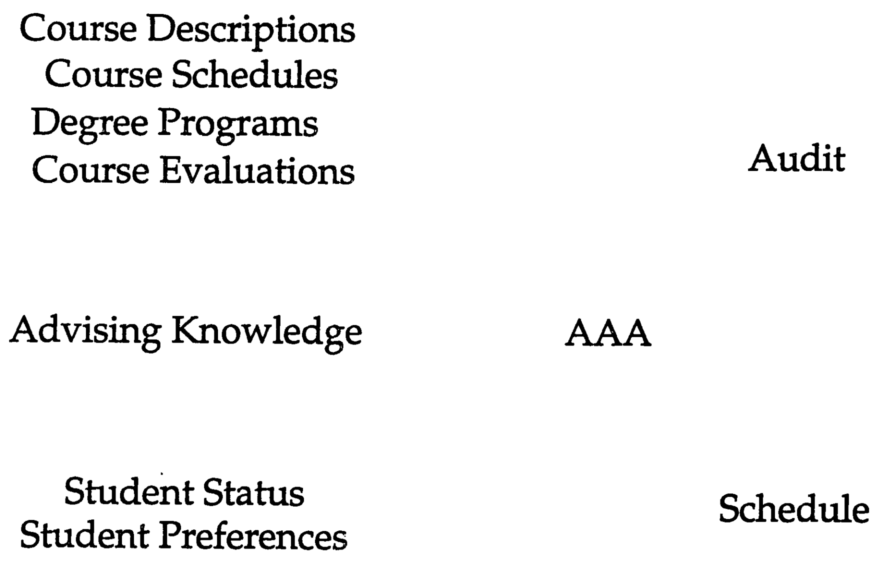

Figure 1. Overview of the Academic Advisor Assistant

10. A slight exaggeration.

II.A. The Inputs

The Course Descriptions are found in the *Course Bulletin*. Here is an example:

6.871 Knowledge-Based Applications Systems

Prereq.: 6.034; 6.036 or 6.824
G (2)
3-0-9 H-LEVEL Grad Credit

Development of programs containing a significant amount of knowledge about their application domain. Outline: 1) brief review of relevant AI techniques; 2) case studies from a number of application domains, chosen to illustrate principles of system development; 3) discussion of technical issues encountered in building a system, including selection of knowledge representation, knowledge acquisition, etc.; and 4) discussion of current and future research. Hands-on experience in building an expert system (term project). 8 Engineering Design Points.

R. Davis, P. Szolovits, H. E. Shrobe

The Course Schedule for the same course looks as follows:

```
6.871   (3 0 9)    KNOWLEDGE-BASED SYSTEMS                 LEC
                   TR9.30-11                            34-302
```

The Degree Programs are also found in the *Bulletin*. An example is not provided, lest the reader accuse this author of trying to make this report artificially long. The Course Evalutions, of course, can be found in the *Course Evaluation Guide*. Again, an example is not provided.

The Advising Knowledge is comprised of rules, such as those provided in Section III of Chapter 1.

The Student Status, specified by the student, is comprised of the following information:

Field		Example
Name	:	Marlboro
Year	:	junior
This-Term	:	spring 1995
Next-Term	:	fall 1995
Courses-Already-Taken	:	3.091, 18.01, 18.02, 8.01, 8.02, 6.041, 21F302, 21M301, 7.012, 21M302, 24.00, 21M240, 6.004, 6.170, 18.063, 6.035, 6.045J, 21M303, 21M700, 21W760, 6.001, 6.002, 18.06
Other-Units-Received	:	24
Thesis-Units-Completed	:	0
Phase-I-Completed?	:	yes
Phase-II-Completed?	:	no
Swimming-Requirment-Completed?	:	yes
Accumulated-PE-Points	:	8

The Student Preferences, again specified by the student, are comprised of the following information:

Field		Example
Desired-Date-of-Graduation	:	Spring 1999
Desired-Degree(s)	:	VI-P

Desired-Minor(s)	: NONE
Desired-Concentration	: Music
Maximum-Course-Load	: 54
Maximum-Class-Hours	: 20
Maximum-Lab-Hours	: 20
Maximum-Prep-Hours	: 20
Minimum-Course-Load	: 48
Minimum-Class-Hours	: none
Minimum-Lab-Hours	: none
Minimum-Prep-Hours	: none
Want-To-Take-Next-Term	: 6.111
Want-To-Take	: 6.871, 6.868J
Don't-Want-To-Take	: 6.821
Override-Permission-of-Instructor?	: yes
Consider-Undergrad-Courses?	: yes
Consider-Grad-Courses?	: yes

The field "Override-Permission-of-Instructor?" indicates whether or not the student wishes to consider the prerequisite, "Permission of instructor", satisfied.

II.B. The Outputs

The Audit output should list all categories of requirments for the student's degree program, and indicate the student's status in completing them. For example, for the student Marlboro, whose status and preferences are shown above, the VI-P Audit should show something like:

Req. Category	Status
GIR[11]	: Completed Calculus I; Calculus II; Physics I; Physics II; Chemistry; Biology; HASS-D 1/2/LO; Third HASS-D; HASS; REST; Phase I; Swimming Req.; Phys. Ed.; Concentration
Required	: Taken thus far : 6.001; 6.002; 6.004
Thesis Units	: No progress
Math Rest. Elec.[12]	: Completed by : 6.041; 18.06; 18.063
Lab Rest. Elec.	: Completed by : 6.170
Large Eng. Conc.[13]	: No progress
Small Eng. Concs.	: No progress
Grad-H Units	: No progress
Eng. Design Points	: Taken thus far : 32 points

Departmental program courses that also satisfy the GIRs :
18.06; 6.001 : Total of 27 units.

11. General Institute Requirements.
12. Math Restricted Electives.
13. Large Engineering Concentration.

Unrestricted Elec. : 21W760 : Subtotal of 12 units.
Other units received = 24. Total is 36 units.
Not yet completed.

Remaining to be completed in GIR : Laboratory; HASS-D 4/5; Phase II
Remaining to be completed in VI-P : Required; Thesis Units; Large Eng. Conc.; Small Eng. Concs; Grad-H Units; Eng. Design Points; Unrest. Elec.

The Generate Schedule output should have two components: a suggested bag of courses, and the weekly schedule that the schedule implies. For Marlboro, the suggested bag may be:

6.034	Artificial Intelligence	4-4-4	EC-1 Header
6.046J	Introduction to Algorithms	4-0-8	EC-7 Header
6.111	Introductory Digital Systems Laboratory	3-7-2	Institute Lab
17.241	Introduction to the American Political Process	3-0-9	HASS-D, 4

and that selection may imply the following schedule:

	Mon	Tue	Wed	Thu	Fri
10:00	**6.034-L**	6.034-R	**6.034-L**		**6.034-L**
10:30	*	*	*		*
11:00					**6.046J-R**
11:30					*
12:00	**6.111-L**	6.111-R	**6.111-L**		**6.111-L**
12:30	*	*	*		*
1:00					
1:30					
2:00					
2:30		**6.046J-L**		**6.046J-L**	
3:00	**17.241-L**	*	**17.241-L**	*	
3:30	*	*	*	*	

Items in bold cannot be moved.
6.034, 6.046J, and 17.241 have final examinations.
Other 6.034 recitation slots are : T9, T11, T12, T1, T2, T3.
Other 6.111 recitation slots are : T11, T1, T2, T3
17.241 recitations are "To be arranged".

III. Strategy for Schedule Generation

The following strategy is used to generate schedules:

1. Determine the set *P* of take-able courses by iterating through all courses, and for each course C, insert it into *P* if and onlf if:
   - C is offered in the upcoming term.
   - All of C's prerequisites have been satisfied by the student.
   - If the course has not been taken, unless the course is repeatable for credit.

2. Score the members of *P*, by determining for each *p* in *P*:
   - Number of General Institute Requirement courses it satisfies.
   - Number of degree program courses it satisfies.
   - Number of courses which have *p* as a prerequisite. These will be hereafter referred to as a course's *inverse* prerequisites.
   - The offering flexibility, as measured by how often the course is offered.
3. Based on the student's preferences, and based on the scores determined in Step #2, choose optimal bags of courses *B*. In doing this, also consider the impact of a particular choice on all future terms. That is, consider the questions, "How will taking the courses A, B, C, and D, affect the student's course selection in the following semester? How will it affect his ability to graduate when he wants?" Consult the *Course Evaluation Guide* to find additional information about each course.
4. Eliminate any members of *B* which has time conflicts.
5. Present a suitable number of *B*'s members to the student, arranged in decreasing order of optimality.

Chapter 3 : Implementation

I. Language Choice

AAA was implemented entirely in C. Although C was not the most appropriate language for all parts of the system, it was the most appropriate for the parsing and the computation components, and in all components, it was expected to provide faster execution than any other language. An ideal solution would have combined several languages, but the overhead and complexity of multi-language programming was not desirable, especially in light of the fact that a very large amount of computation was expected to be involved in the system.

The choice of a declarative language like C was expected to make the incorporation of rules into the system somewhat difficult, but the trade-off seemed worthwhile, since with some additional programming, a chaining engine could always be built in C, but a rule-based program could *never* (practically speaking) be optimized to the point where its execution speed could come even close to that of a C program.

As it turns out, the costs of this trade-off can be minimized, by using a system called CLIPS (C Language Integrated Production), whose existence and description was provided to this author by Dhananjay Ragade, a fellow 6.871 student:

> *"It's a forward chaining rule based system that we used to implement our San Francisco Tour Expert System.*
>
> *"It makes for easy integration into C, and uses a LISP-like syntax. It is pretty easy to use, yet is powerful, and fast.*
>
> *"Moreover, there is a ton of documentation on it, including hardcover textbooks."*

II. Limitations of the Current Release

The current release of AAA has several limitations, some of which are quite significant, and others which are virtually trivial. Taken together, these limitations prevent AAA from being able to generate schedules, and therefore reduces AAA to a mere auditor of courses. That is still an improvement over the current state of technology,[14] but obviously, far from what it was supposed to accomplish.

II.A. Lack of CEG and Class Schedule Data

The current release does not have the *Course Evaluation Guide* or the *Class Schedule* in its database. The lack of the former means that AAA can rank courses only based on its own scoring of them based on satisfaction of requirements, and the courses' inverse prerequisites and flexibilities of offering. The lack of the latter is much more serious, because without the Class Schedule, AAA has no way of determining whether any of the *B* optimal bags it determines is actually take-able.

II.B. Lack of Knowledge

The current release also lacks advising knowledge of any sort. This is, by far, the most serious limitation, since without knowledge to limits the potential choices of optimal bags, AAA must consider a computationally intractable number of courses.

14. Currently, the Registrar's Office fully audit's a student's GIR completion status. That is, a student is informed of exactlywhich GIRs he has completed and which he has not in every grade report. However, concerning the student's specific degree program, the grade report only indicates how many units beyond GIRs the student has completed.

For instance, suppose that for any given term, ten optimal courses can be identified. Supposing further that the student only wants to consider bags of courses which contain four courses, AAA must evaluate all 210 4-subsets of 10.[15] Furthermore, because AAA has to determine the impact of a particular choice on all future terms, the number of evaluations grows exponentially with the number of terms remaining in the student's MIT career. Specifically, this number is given by (branching factor) ^ (number of terms), and with branching factor equal to 210, and number of terms equal to eight, the number of evaluations is 210^8 = 3.78*10^18. Even at one million evaluations per second, it would take AAA 120,000 years to complete the computation. By that time, the human race will be extinct, not to mention the student.

Thus, the rectification of this limitation will be absolutely critical for AAA's to reach full functionality.[16] Admittedly, the significance of advising knowledge was not fully realized in the project's design phases. If it had been, it would have been addressed earlier, and would have existed in some form in the current release. This author stands guilty of lacking foresight, at the cost of seriously debilitating the Academic Advisor Assistant.

II.C. Minor Limitations

There are several other minor limitations in the current release.

First, AAA currently is not interactive. Instead of querying the user for his status and preferences, it reads the information from a file. This, of course, is a very minor limitation, and one that could be rectified easily.

Second, AAA does not audit minors or concentrations. This problem is only slightly more serious than the last. Removing this limitation entails parsing the *MIT Student's Guide to the Humanities, Arts, and Social Sciences.*

Third, AAA is currently hard-coded to work only for VI-P majors. Because a solid foundation of datatypes and functions are already in place, however, removing this limitation will be time-consuming, but not difficult.

III. User Interface

The following several pages contain pictures of the user interface. As shown, the interface is X-based. If the reader is wondering how this author could have had the audacity to spend time building an X-based interface when he failed to build any advising knowledge into the system, the answer is simple: It became clear early in the project that a text-based interface would not be optimal for the kind of functions AAA was designed to achieve; as mentioned earlier, the crucial significance of the advising knowledge did not crystallize until much later. Yes, that is a lousy excuse, and no, this author cannot provide anything better.

15. The number of 4-subsets of 10 is 10-choose-4, or 10!/6!4! = 210.

16. Perhaps more importantly, this limitation seriously jeopardizes this author's grade for 6.871.

Figure 2. The Start-Up Screen

Figure 3. General Institute Requirements

Figure 3. VI-P Requirements

Figure 4. Student Status

Figure 5. Student Preferences

Figure 6. Statistics

Figure 7. GIR Audit

Figure 8. VI-P Audit

Chapter 4 : Evaluation

This chapter presents a brief evaluation of the current release of AAA, as well as a description of the lessons learned through its development, and some hints of future enhancements.

I. The Good

Two aspects of the current release are sufficiently meritable to deserve some attention.

First, the auditing portion of AAA works particularly well. By itself, this is not so noteworthy, except that it is a direct consequence of a solid design. Thus, as far as auditing is concerned, new degree programs could be added quite effortlessly, if involving a bit of tedium. Furthermore, through some clever pre-processing of course descriptions, audit runs remarkably fast, requiring only one pass through the courses taken to determine the student's exact status.

Second, though this may be viewed as a generalization of the first point above, the over all design has kept true to the fundamental guiding principle of maintaining independence of degree programs. Thus, degree programs can indeed be added, deleted, and modified independently of others, as long as they obey a well-defined interface. An interesting implementation detail is that each degree program has been abstracted as an object, which communicates with the system by responding to messages.

II. The Bad

The worst aspect of AAA is really due to forces beyond this author's control. Namely, working on this project has revealed that the data in the *Course Bulletin* is surprisingly dirty. To begin with, there are a significant number of blatant bugs.[17] Furthermore, the data changes every year, as courses are cancelled and new ones are initiated, not to mention the adjustment of units and revision of course numbers.

To deal with this kind of dirty and dynamic data, AAA will have to be imbued with a fresh set of updated data each semester. Worse yet, until AAA includes a parser for all the data it uses, a computer programmer would be the only one qualified to perform this maintenance task. It is doubtful that such a person can be found in the ranks of the Registrar's Office, so either someone new will need to be hired, or some poor MIT student will have to volunteer his services.

III. The Ugly

The singularly ugly aspect of AAA, as mentioned before, is the intractability of the computation. Unless this problem is somehow circumvented, AAA has little chance of being embraced by the MIT community.

IV. Lessons Learned

Other than the painful lesson of identifying design weaknesses earlier on in a software engineering project, the only significant lesson learned was never to rely on an administrative office to provide clean data, or to understand what one is trying to do to help them. This author contacted the Registrar's Office early in the project for possible guidance and collaboration, but was received

17. For example, the course 6.931 lists 14.002 as a prerequisite, when there is no such course. In this case, the fix is obvious, but there are many more cases, in some of which the fix cannot be determined by any method except by consulting either the Registrar or the course secretary.

with deep-felt skepticism and disinterest. Essentially, the Registrar felt that the level of auditing the Office currently provides to the students is sufficient, despite the fact that it says absolutely nothing about the student's status in completing his degree program.

This author has not lost all hope, however. Once AAA has attained greater functionality, a well-prepared demonstration might just make a critical mass of converts at the Registrar's Office, thereby leading to a steady source of resources for maintaining AAA.

V. Future Direction

One exciting area of improvement is in that of specifying temporal preferences. Some examples of these may be (as might be specified by the student):

- I don't want to take 6.035 and 6.111 together.
- I don't want any classes on Mondays and Fridays
- I don't want to take any finals my senior year.

The ability to take into account such preferences in the schedule generation should really help AAA gain the support of students.

Chapter 5 : Conclusion

The Academic Advisor Assistant is a system with tremendous potential. At the least, AAA allows students to determine their exact status in the pursuit of their academic goals -- this much functionality is already provided by the current release. Once AAA reaches full functionality as specified in Chapter 2 of this report, however, AAA will serve as a truly knowledgeable assistant to both students and advisor alike, by summarizing all available information to deduce a set of courses from which the student can choose his courses, with the certainty that he would not be making some mistake which will create difficulties for him later on.

With further embellishments, AAA will allow students to specify temporal preferences, such that "morning" students can pick a schedule which will let them finish their courses by noon, say, while others can pick one which will let them sleep till noon. With even further embellishments, AAA will be able to suggest to the student that perhaps he should be considering a double major, because it is easily within his reach, or that the student could save $10000 in tuition by taking just one additional course in the upcoming term, which will allow him to graduate a term early.

In short, AAA has the potential to really improve the welfare of students and faculty at MIT. Indeed, if by eliminating the need for them to flip pages for an hour each semester, AAA allows them to do one more hour of research, or get one more hour of sleep, all of this author's efforts would have been well worth it. Amen.

Appendix : Source Code

This appendix contains all of the C source code which comprise AAA. Due to time constraints, much of it could not be commented sufficiently.

I. Header Files

I.A. constants.h

```
/**************************************************************************************************************
 * constants.h
 *
 * Author       : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *                               MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *
 * Abstract     : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *                designed and implemented in the spring of 1995 as the author's Master of Engineering thesis, under
 *                the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.  See main.c for
 *                a more complete description of AAA.
 *
 *                This file contains miscellaneous constants used by AAA in the following order:
 *
 *                     Location, Names, and Other Attributes of Files
 *                     Parsing Constants
 *                     Unparsing Constants
 *                     Maximum-Number-Of Constants
 *                     Maximum-Length-Of Constants
 *                     Boolean Constants
 *                     Useful Constants for Certain Chars and Strings
 *                     Special Scanf Strings
 *
 * Compiling Env : None.  This file is self-contained.
 *
 * Code Status                  : 0.5.0 WIP
 * Last Modification Date & Time : April 20, 1995
 *
 **************************************************************************************************************/

/**************************************************************************************************************
 ***** Location, Names, and Other Attributes of Files ********************************************************
 **************************************************************************************************************/

#define HOMEDIR                         "/mit/marlboro/Thesis/"          /* AAA's home directory           */
#define CRSE_DESC_DIR                   "CourseDescriptions/"            /* This directory must be in HOMEDIR  */
#define DESC_FILES_LISTING              "dir_listing"                    /* This file must be in CRSE_DESC_DIR */
#define DESC_FILES_LISTING_END_MARKER   "END"                            /* Marks the end of DESC_FILES_LISTING */
#define DEBUGGING_FLAGS_FILE            "Code/debugging_flags"           /* This file must be in HOMEDIR      */
#define VERSION                         "Version 1.0.0 Work in Progress" /* Identifies version of AAA         */
#define ACADEMIC_PROGRAMS_FILE          "academic_programs"              /* this must be in HOMEDIR           */
#define STUDENT_STATUS_FILE             "student_status"                 /* this must be in HOMEDIR           */
#define STUDENT_PREFS_FILE              "student_preferences"            /* this must be in HOMEDIR           */
#define BULLETIN_VERSION                "1994-95"
#define EARLIEST_VALID_YEAR             1995
#define LATEST_VALID_YEAR               3000
#define VERY_SHORT_NUM_MAX_NONE         0
#define VERY_SHORT_NUM_MIN_NONE         255

/**************************************************************************************************************
 ***** Parsing Constants - These may need be adjusted if the format of the Bulletin changes. ******************
 **************************************************************************************************************/

/*****
 ***** Note that some of the constants in the "Useful Constants for Certain Chars and Strings" section below
 ***** also play a non-trivial role in the parsing process.
 *****/

#define NULL_CN_SET       "--"
#define LINE_DELIMITER    "-------------------------------------------------------"
#define OR_LIST_DELIMITER "or"

#define SAME_SUBJ_AS_INIT_STR              "(Same"
#define MEETS_WITH_INIT_STR                "(Subject"
#define PREREQ_STR                            "Prereq.:"
#define TERM_OFFERED_NOT_OFFERED_STR          "Not offered"
#define SUBJECT_LEVEL_U_STR                   "U"
#define PASS_FAIL_STRING                      "[P/D/F]"
#define H_LEVEL_STRING                        "H-LEVEL Grad Credit"
#define PASS_FAIL_AND_H_LEVEL_STRING          "[P/D/F]  H-LEVEL Grad Credit"
#define REPEATABLE_FOR_CREDIT_INIT_STRING     "Can"
#define NO_ADDITIONAL_CREDIT_FOR_INIT_STRING  "Credit"

#define POI_INIT1   "Permission"
#define POI_INIT2   "permission"
#define POI_CODE    "POI"
#define POI_STRING  "Permission of Instructor"

#define EQUIV1        "Equivalent"
#define EQUIV2        "equivalent"
#define EQUIV_CODE    "EQUIV"
#define EQUIV_STRING  "Equivalent"

/**********************************************************************************************************
 ***** Unparsing Constants ********************************************************************************
 **********************************************************************************************************/

#define UNPARSE_LABEL   "%-29s : "
#define UNPARSE_INDENT  "\t\t\t\t"

/**********************************************************************************************************
 ***** Maximum-Number-Of Constants ************************************************************************
 **********************************************************************************************************/

#define MNO_CRSE_DESC_FILES    40
#define MNO_CRSES            4000
#define MNO_CNS_IN_SET        100
#define MNO_STRINGS_IN_BUFFER  10
#define MNO_DEGREES_IN_CAT    100
#define MNO_DEGREES            10
#define MNO_MINORS_SCI_ENG    100
#define MNO_OPTIONS            15
#define MNO_APPLICABLE        100
#define MNO_GIR_TYPE_CRSES   1000
#define MNO_TAKEN_ALL         100

/**********************************************************************************************************
 ***** Maximum-Length-Of Constants ************************************************************************
 **********************************************************************************************************/

#define MLO_FILENAME            100  /* Ex: /mit/marlboro/Thesis/CourseDescriptions/Course_1    */
#define MLO_CRSE_NUMBER          20  /* Ex: 6.001, 6.001-6.002                                  */
#define MLO_CRSE_NAME           150  /* Haydn, Mozart, and Beethoven (Revised Units)            */
#define MLO_ONE_STRING          100  /* The longest string that appears among the parsed files. */
#define MLO_BUFFER              300  /* Stream buffer.  3 * MLO_ONE_STRING should be enough.    */
#define MLO_UNITS                20  /* Ex: 3-12-3, Arranged                                    */
#define MLO_TERM_OFFERED        100  /* Ex: This Year : ( 1 IAP 2 SUMMER )                      */

/**********************************************************************************************************
 ***** Useful Constants for Certain Chars and Strings *****************************************************
 **********************************************************************************************************/

#define SPACE      ' '
#define COMMA      ','
#define SEMICOLON  ';'
#define HYPHEN     '-'
#define NEWLINE    10

#define SPACE_STR      " "
#define COMMA_STR      ","
#define SEMICOLON_STR  ";"

#define LINE_40    "----------------------------------------"
#define LINE_80    LINE_40 LINE_40
#define LINE_120   LINE_80 LINE_40

/**********************************************************************************************************
 ***** Special Scanf Strings ******************************************************************************
 **********************************************************************************************************/

/***** The Primitive *****/
```

```
#define SKIP_SPACE          "%*[ ]"
#define UP_TO_NEWLINE       "%[^\12]"
#define UP_TO_RIGHT_PAREN   "%[^)]"

/***** The Compound *****/

#define SKIP_SPACE_UP_TO_NEWLINE      SKIP_SPACE UP_TO_NEWLINE
#define SKIP_SPACE_UP_TO_RIGHT_PAREN  SKIP_SPACE UP_TO_RIGHT_PAREN

/*******************************************************************************************************************
 ***** Display Constants ******************************************************************************************
 *******************************************************************************************************************/

#define WINDOW_TITLE         "Academic Advisor Assistant"
#define WINDOW_ICON_TITLE    "AAA"
#define WINDOW_WIDTH         900
#define WINDOW_HEIGHT        700
#define WINDOW_BORDER_WIDTH  2

#define BUTTON_AREA_X           700

#define SCREEN_TITLE_Y     15
#define SCREEN_TITLE       "Academic Advisor Assistant"
#define SCREEN_TITLE_FONT  "-adobe-times-bold-i-normal--34-240-100-100-p-170-iso8859-1"

#define AUTHOR_NAME_Y      75
#define AUTHOR_NAME       "by Dae-Chul Sohn, MIT M.Eng. in EECS, 1995"
#define AUTHOR_NAME_FONT  "-adobe-new century schoolbook-bold-r-normal-*-14-*-100-100-*-*-iso8859-1"

#define GAP_BETWEEN_TOP_LINES 3
#define GAP_BETWEEN_SIDE_LINES 3
#define TOP_LINE_Y       65
#define TOP_LINE_GAP     10     /* distance from the top line to the top of the text area       */

#define SIDE_MARGIN       20
#define BOTTOM_MARGIN     20

#define TAB   40
#define TAB1  240
#define TAB2  250

#define BUTTON_WIDTH        180
#define BUTTON_HEIGHT       30
#define BUTTON_SPACE        15
#define BUTTON_INNER_GAP_X  3    /* x distance between outer and inner boxes */
#define BUTTON_INNER_GAP_Y  3    /* y distance between outer and inner boxes */
#define BUTTON_Y_GAP        10   /* the y distance between buttons           */
/*
#define BUTTON_FONT         "-b&h-lucidabright-demibold-r-normal--17-120-100-100-p-101-iso8859-1"
#define BUTTON_FONT         "-*-clean-*-*-*-*-16-*-*-*-*-*-*-*"
#define BUTTON_FONT         "-adobe-times-medium-r-*-*-17-*-*-*-*-*-*-*"
*/

#define BUTTON_PRESSED   1
#define BUTTON_RELEASED  2

#define BUTTON_FONT  "-adobe-times-bold-r-normal--14-140-75-75-p-77-iso8859-1"
#define TEXT_FONT  "-adobe-times-bold-r-normal--14-140-75-75-p-77-iso8859-1"

/***** These should be elsewhere *****/

#define NO_MAIN_SCREEN_OPTIONS 8
#define NO_AUDIT_SCREEN_OPTIONS 6

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

I.B. datatype_reps.h

```
/* first order */

typedef unsigned char  OneNumCode;
typedef unsigned short UShort;
typedef unsigned char  VeryShortNum;
typedef struct _AcadProgCat
{
  OneNumCode      code;
  char         ** members;
  VeryShortNum    count;
}
AcadProgCat;

typedef struct _AcadProgs
{
  AcadProgCat ** categories;
  VeryShortNum   count;
}
AcadProgs;

typedef struct _Degrees
{
  OneNumCode     type;
  VeryShortNum * members;
  VeryShortNum   count;
}
Degrees;

typedef struct _Stats
{
  UShort listed_count;
  UShort offered_count;
  UShort undergrad_count;
  UShort grad_count;
  UShort considered_count;
  UShort no_prereqs_count;
  UShort prereqs_satisfied_count;
  UShort taken_already_count;
  UShort takable_count;
}
Stats;

typedef struct _Stream
{
  FILE         * source;
  char         * buffer[MNO_STRINGS_IN_BUFFER];
  VeryShortNum   head;                              /***** first filled slot *****/
  VeryShortNum   tail;                              /***** first empty slot  *****/
  VeryShortNum   count;
}
Stream;

typedef struct _Term
{
  UShort        year;
  VeryShortNum  season;
}
Term;

/* Crse Stuff */

typedef struct _CrseSet
{
  char   ** members;
  UShort    count;
  UShort    capacity;
}
CrseSet;

typedef struct _CrseNumber
{
  char       *  number;
  OneNumCode    rel_to_next;
}
CrseNumber;

typedef struct _CrseNumberSet
{
  CrseNumber   ** members;
  VeryShortNum    count;
}
CrseNumberSet;
```

```
typedef struct _CrseBagItem
{
  CrseNumberSet * cns;
  UShort          ranking;
}
CrseBagItem;

typedef struct _CrseBag
{
  CrseBagItem  ** items;
  UShort          count;
}
CrseBag;

typedef struct _CrseDesc
{
  CrseNumberSet *  number;
  char          *  name;
  CrseNumberSet *  former_number;
  CrseNumberSet *  same_subj_as;
  CrseNumberSet *  meets_with;
  CrseNumberSet *  prerequisites;
  OneNumCode       term_offered;
  OneNumCode       subject_level;
  Boolean          units_arranged;
  VeryShortNum     class_hours;
  VeryShortNum     lab_hours;
  VeryShortNum     prep_hours;
  VeryShortNum     total_units;
  OneNumCode       GIR_fulfillment;
  OneNumCode       GIR_fulfillment_qualifier;
  UShort           GIR_fulfillment_total;
  Boolean          pass_fail;
  OneNumCode       h_level;
  Boolean          repeatable_for_credit;
  CrseNumberSet *  no_add_credit_for;
  VeryShortNum     inv_prereqs;
  OneNumCode       tag_first;
  OneNumCode       tag_second;
  OneNumCode       tag_third;
  int              tag_total;
}
CrseDesc;

typedef struct _StudStatus
{
  char          *  name;
  VeryShortNum     year;
  Term          *  this_term;
  Term          *  next_term;
  CrseSet       *  taken;
  CrseSet       *  taken_all;
  CrseDesc      ** corr_cds;
  VeryShortNum     other_units;
  VeryShortNum     thesis_units;
  Boolean          phase_I_complete;
  Boolean          phase_II_complete;
  Boolean          swimming_complete;
  VeryShortNum     PE_points;
}
StudStatus;

typedef struct _StudPrefs
{
  Term          * date_of_grad;
  Degrees       * majors;
  Degrees       * minors;
  Degrees       * conc;

  VeryShortNum    max_course_load;
  VeryShortNum    max_class_hours;
  VeryShortNum    max_lab_hours;
  VeryShortNum    max_prep_hours;

  VeryShortNum    min_course_load;
  VeryShortNum    min_class_hours;
  VeryShortNum    min_lab_hours;
  VeryShortNum    min_prep_hours;

  CrseSet * want_to_take_next_term;
  CrseSet * want_to_take;
  CrseSet * dont_want_to_take;

  Boolean         override_poi;
  Boolean         consider_undergrad;
  Boolean         consider_grad;
}
StudPrefs;

typedef struct _Button
{
  char         *  label;
  unsigned int    outer_x, outer_y;
  unsigned int    outer_w, outer_h;
  unsigned int    inner_x, inner_y;
  unsigned int    inner_w, inner_h;
}
Button;

typedef struct _Buttons
{
  Button **  members;
  int        n;
}
Buttons;

typedef struct _Env
{
  Display     * the_display;
  Window        the_window;
  GC            the_context;
  Pixmap        the_buffer;
  Font          font_id;              /* The text font */
  XFontStruct * font;
  int           font_ascent;
  int           font_descent;
  int           font_height;
  Font          button_font_id;
  XFontStruct * button_font;
  int           button_font_ascent;
  int           button_font_descent;
  int           button_font_height;
  int           text_area_top;
  int           text_area_bottom;
  int           text_area_left;
  int           text_area_right;
  int           posx;
  int           posy;
  int           black;
  int           white;
}
Env;

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

typedef struct _Units  /* Immutable */
{
  UShort   req;
  char   * units;
}
Units;

typedef struct _SubCS  /* Immutable */
{
  CrseSet      * set;
  char         * set_desc;
  VeryShortNum   req_num;
}
SubCS;

typedef struct _OneOfSCS
{
  SubCS        ** subcss;
  VeryShortNum    count;
}
OneOfSCS;

typedef struct _SubQualSCS
{
  SubCS   * subcs;
  SubCS   * including;
  CrseSet * excluding;
}
SubQualSCS;

typedef struct _SetOfSQSCS
{
  SubQualSCS   ** members;
  VeryShortNum    count;
  VeryShortNum    req_num;
} SetOfSQSCS;

typedef struct _GIR
{
  CrseSet *  categories[NO_GIR_TYPES];
  CrseSet *  all_hass;
  CrseSet *  all_phaseI;
  CrseSet *  all_phaseII;
}
GIR;

typedef struct _Strings
{
  char   ** members;
  UShort    count;
  UShort    capacity;
}
Strings;

typedef struct _AuditResult
{
  CrseSet *  applicable;
  UShort     appl_units;
  Strings *  completed;
  Strings *  not_completed;
  Strings *  not_audited;
}
AuditResult;

typedef struct _Essence
{
  Env        *  env;
  CrseDesc   ** cds;
  AcadProgs  *  ap;
  StudStatus *  ss;
  StudPrefs  *  sp;
  Stats      *  st;
  GIR        *  gir;
}
Essence;

typedef struct _RankedCrse
{
  char          * crse;
  VeryShortNum   inv_prereqs;
  VeryShortNum   off_flex;          /***** offering flexibility  *****/
  VeryShortNum   GIR_ff_count;      /***** GIR fulfillability    *****/
  UShort         GIR_ff_units;
  VeryShortNum   GIR_ff_cats;
  VeryShortNum   degree_ff_count;   /***** degree fulfillability *****/
  UShort         degree_ff_units;
  VeryShortNum   degree_ff_cats;
} RankedCrse;

/***** Unused ******************************************************************************************/

typedef struct _OverAll
{
  VeryShortNum  h_units;
  VeryShortNum  thesis_units;
  VeryShortNum  ED_points;
  VeryShortNum  unrest_elec;
} OverAll;

typedef struct _StateValidity
{
  Boolean stud_status_valid;
  Boolean stud_prefs_valid;
}
StateValidity;
```

I.C. datatype_reps_constants.h

```
enum DATATYPE_LABELS_ENUM {
  DT_BOOLEAN = 0,
  DT_TERM_PTR,
  DT_DEGREES_PTR,
  DT_VERY_SHORT_NUM,
  DT_CRSE_SET_PTR
  };

/*****************************************************************************************************************
 ***** CrseNumber ************************************************************************************************
 *****************************************************************************************************************/

enum CRSE_NUMBER_REL_TO_NEXT_ENUM {
  CRSE_NUMBER_REL_TO_NEXT_NONE = 0,
  CRSE_NUMBER_REL_TO_NEXT_RANGE,
  CRSE_NUMBER_REL_TO_NEXT_AND,
  CRSE_NUMBER_REL_TO_NEXT_OR
  };

/*****************************************************************************************************************
 ***** CrseDesc **************************************************************************************************
 *****************************************************************************************************************/

/***** Term Offered *********************************************************************************************/

/***** These are bit flags. *****/

enum TERM_OFFERED_ENUM {
  TERM_OFFERED_NONE          =   0,  /* 0000 0000 */
  TERM_OFFERED_THIS_FIRST    =   1,  /* 0000 0001 */
  TERM_OFFERED_THIS_IAP      =   2,  /* 0000 0010 */
  TERM_OFFERED_THIS_SECOND   =   4,  /* 0000 0100 */
  TERM_OFFERED_THIS_SUMMER   =   8,  /* 0000 1000 */
  TERM_OFFERED_NEXT_FIRST    =  16,  /* 0001 0000 */
  TERM_OFFERED_NEXT_IAP      =  32,  /* 0010 0000 */
  TERM_OFFERED_NEXT_SECOND   =  64,  /* 0100 0000 */
  TERM_OFFERED_NEXT_SUMMER   = 128   /* 1000 0000 */
  };

/***** These bit flags are used in calling parse_term_offered. *****/

#define THIS_YEAR   1  /* 01 */
#define NEXT_YEAR   2  /* 10 */
#define BOTH_YEARS  3  /* 11 */

/***** Subject Level ********************************************************************************************/

#define SUBJECT_LEVEL_U  1
#define SUBJECT_LEVEL_G  2

/***** Class Hours **********************************************************************************************/

#define CLASS_HOURS_UNITS_ARRANGED  -1  /* special value to denote Units Arranged */

/***** Degrees **************************************************************************************************/

enum DEGREE_TYPE_ENUM {
  DEGREE_TYPE_MAJOR = 0,
  DEGREE_TYPE_MINOR,
  DEGREE_TYPE_CONC
  };
```

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

enum RANKED_CRSE_SORTING_FIELDS_ENUM {
  SORTING_FIELD_NONE = 0,
  INV_PREREQS,
  CRSE,
  DEGREE_FF_UNITS,
  DEGREE_FF_COUNT,
  GIR_FF_UNITS,
  GIR_FF_COUNT,
  BOGUS
  };
```

I.D. error_and_trace.h

```
/*************************************************************************************************************
 ***** Constants Pertaining to ErrorStrings ******************************************************************
 *************************************************************************************************************/

enum ERROR_ENUM {
  ERROR_NONE = 0,   /* this must be zero */

  ERROR_ALLOC_FAIL,

  ERROR_FOPEN_FAIL,
  ERROR_FCLOSE_FAIL,

  ERROR_PRINTF_FAIL,
  ERROR_SPRINTF_FAIL,

  ERROR_BAD_CRSE_DESC_FILE,
  ERROR_BAD_DEBUGGING_FLAGS_FILE,
  ERROR_BAD_DESC_FILES_LISTING,

  ERROR_COURSE_NUMBER_LONGER_THAN_7,

  ERROR_BUFFER_FULL,
  ERROR_BUFFER_NOT_EMPTY,

  ERROR_BAD_STUD_STAT_FILE,
  ERROR_INVALID_STUD_STAT_FIELD,
  ERROR_INVALID_STUD_STAT_YEAR_FIELD_VALUE,
  ERROR_INVALID_STUD_STAT_THIS_TERM_SEASON_VALUE,
  ERROR_INVALID_STUD_STAT_THIS_TERM_YEAR_VALUE,
  ERROR_INVALID_STUD_STAT_NEXT_TERM_SEASON_VALUE,
  ERROR_INVALID_STUD_STAT_NEXT_TERM_YEAR_VALUE,
  ERROR_SOMETHING_SAVED_ALREADY,
  ERROR_NOTHING_SAVED,

  ERROR_BAD_STUD_PREFS_FILE,
  ERROR_INVALID_STUD_PREFS_FIELD,
  ERROR_INVALID_STUD_PREFS_DATE_OF_GRAD_SEASON_VALUE,
  ERROR_INVALID_STUD_PREFS_DATE_OF_GRAD_YEAR_VALUE,

  ERROR_BAD_ACAD_PROGS_FILE,

  ERROR_INVALID_DEGREE_TYPE,
  ERROR_INVALID_DEGREE_NAME,

  ERROR_OUT_OF_BOUNDS,

  ERROR_RANGE_FOLLOWED_BY_RANGE,

  ERROR_UNKNOWN_MSG,

  ERROR_UNIDENTIFIED_CRSE_NUM,

  ERROR_CAPACITY_FULL,
  ERROR_CANNOT_APPEND_TO_NULL,

  ERROR_NOT_YET_IMPLEMENTED
  };

enum WARNING_ENUM {
  WARNING_NONE = 0,
  WARNING_BAD_CRSE_LIST,
  WARNING_UNIDENTIFIED_CRSE_NUM
  };

/***********************************************************************************************************************
 * Constants Pertaining to dfs
 *
 * There are two types of these.  The first, denoted by TRACE_function_name, determines whether or not the execution
 * of that function should be traced.  The second, denoted by PRINT_function_name_info, determines whether during
 * the execution of that function, the info should be printed.
 *
 * The numbers assigned to these constants follow this scheme (brackets indicate closed intervals):
 *
 *      TRACE_alpha : [0x000, 0x0FF]
 *
 *      PRINT_MAIN_info                  : [0x100, 0x10F]
 *      PRINT_PARSE_INTO_CRSE_DESCS_info : [0x110, 0x12F]
 *      PRINT_READ_STR_FROM_STREAMS_info : [0x130, 0x13F]
 *
 ***********************************************************************************************************************/

#define NO_DEBUGGING_FLAGS  0x300  /* It's a hash table */

/**** 0x000 Series : Main, Big Modules, Proc *****/

enum MAIN_ENUM {
  MAIN_TRACE = 0x000,
  MAIN_CRSE_DESC_FILENAMES,
  MAIN_NUM_OF_CRSE_DESCS,
  MAIN_CRSE_DESCS,
  MAIN_SORTED_CRSE_DESCS,
  MAIN_ACAD_PROGS,
  MAIN_STUD_STAT,
  MAIN_DP_REQUIRED,
  MAIN_DP_TAKEN,
  MAIN_DP_SATISFIED
  };

enum GENERATE_POSS_ENUM {
  GENERATE_POSS_TRACE = 0x010,
  GENERATE_POSS_RESULT
  };

enum CRSE_DESCS__PARSE_ENUM {
  CRSE_DESCS__PARSE_TRACE = 0x020,
  CRSE_DESCS__PARSE_CURR_FILE,
  CRSE_DESCS__PARSE_LINE,
  CRSE_DESCS__PARSE_NUMBER,
  CRSE_DESCS__PARSE_NAME,
  CRSE_DESCS__PARSE_FORMER_NUMBER,
  CRSE_DESCS__PARSE_SAME_SUBJ_AS,
  CRSE_DESCS__PARSE_MEETS_WITH,
  CRSE_DESCS__PARSE_PREREQUISITES,
  CRSE_DESCS__PARSE_TERM_OFFERED,
  CRSE_DESCS__PARSE_SUBJECT_LEVEL,
  CRSE_DESCS__PARSE_UNITS,
  CRSE_DESCS__PARSE_GIR_FULFILLMENT,
  CRSE_DESCS__PARSE_PASS_FAIL,
  CRSE_DESCS__PARSE_H_LEVEL,
  CRSE_DESCS__PARSE_REPEATABLE_FOR_CREDIT,
  CRSE_DESCS__PARSE_NO_CREDIT_FOR, /** 70 **/
  CRSE_DESCS__PARSE_VALIDATE_CNS
  };

/***** 0x100 Series : CrseNumber *****/

enum CRSE_NUMBER_ENUM {
  CRSE_NUMBER__PARSE_ARGS_ENTRY = 0x100,
  CRSE_NUMBER__PARSE_ARGS_EXIT
  };

/***** 0x110 Series : CrseNumberSet *****/

enum CRSE_NUMBER_SET_ENUM {
  CRSE_NUMBER_SET__PARSE_TRACE = 0x110,
  CRES_NUMBER_SET__INSERT_ARGS_ENTRY,
  CRES_NUMBER_SET__INSERT_ARGS_EXIT,
  CRSE_NUMBER_SET__IS_MEMBER_ARGS,
  CRSE_NUMBER_SET__SATISFIES_RECUR
  };

/***** 0x120 Series : Stream *****/

enum STREAM_ENUM {
  STREAM__PUT_ARGS_ENTRY = 0x120,
  STREAM__PUT_ARGS_EXIT,
  STREAM__PUT_STRINGS_ARS_ENTRY,
  STREAM__GET_TRACE
  };

enum DP_GIR_ENUM {
  DP_GIR_TRACE = 0x130,
  DP_GIR_TRACE_INDEX,
  DP_GIR_WAIT_FOR_CLICK,
  DP_GIR_SORT,
  DP_GIR_PRINT_ITER,
  DP_GIR_PRINT_CONSIDERING,
  DP_GIR_TRACE_HASSD_FSM
  };

enum DP_VI_P_ENUM {
  DP_VI_P_TRACE = 0x140,
  DP_VI_P_TRACE_INDEX,
  DP_VI_P_WAIT_FOR_CLICK
  };

enum STRINGS_ENUM {
  STRINGS__INSERT_ARGS_ENTRY = 0x150
  };

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

I.E. externs.h

```
/**********************************************************************************************************************
 * externs.h
 *
 * Author        : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *                               MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *
 * Abstract      : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *                 designed and implemented in the spring of 1995 as the author's Master of Engineering thesis, under
 *                 the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.  See main.c for
 *                 a more complete description of AAA.
 *
 *                 This file contains an extern declaration for every variable in tables.h.  See tables.h for
 *                 more details regarding these variables.
 *
 * Compiling Env : At least one of the .c source files must have included tables.h.
 *
 * Code Status                   : 0.5.0 WIP
 * Last Modification Date & Time : April 20, 1995
 *
 **********************************************************************************************************************/

extern const char       * CrseNumberSuffixes          [];
extern const char       * CrseNumberSpecProgPrefixes  [];
extern const char       * HLevelExceptStrings         [];
extern const UShort       GIRFulfillmentCodes         [];
extern       char       * GIRFulfillmentStrings       [];
extern       char       * GIRStrings                  [];
extern const OneNumCode   HASSFulfillmentCodes        [];
extern const OneNumCode   PhaseIFulfillmentCodes      [];
extern const OneNumCode   PhaseIIFulfillmentCodes     [];
extern const char       * ErrorStrings                [];

extern Boolean   dfs [];

extern const char * AcademicProgramsFields    [];
extern       char * StudStatusFileFields      [];
extern       char * StudStatusYearFieldValues [];
extern       char * SeasonNames               [];
extern      char * StudPrefsFileFields      [];
extern      char * MainScreenOptions        [];
extern      char * AuditScreenOptions       [];

extern const char * Messages[];

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

I.F. primitive_datatype_reps.h

```
typedef unsigned char  Boolean;
```

I.G. primitive_datatype_reps_constants.h

```
#define TRUE  1
#define FALSE 0
```

I.H. prototypes.h

```
/************************************************************************************************************
 * prototypes.h
 *
 * Author       : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *                               MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *
 * Abstract     : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *                designed and implemented in the spring of 1995 as the author's Master of Engineering thesis, under
 *                the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.  See main.c for
 *                a more complete description of AAA.
 *
 *                This file contains the prototypes for the functions which comprise AAA.  They are arranged by
 *                the source file in which they appear.  The source files, in turn, appear in ascending alphabetical
 *                order, except the major functions, whose prototypes are at the beginning.
 *
 * Compiling Env : #include "datatypes.h"
 *
 * Code Status                  : 0.5.0 WIP
 * Last Modification Date & Time : April 20, 1995
 *
 ************************************************************************************************************/

/************************************************************************************************************
 ***** dp_GIR ***********************************************************************************************
 ************************************************************************************************************/

int  dp_GIR                    (VeryShortNum, VeryShortNum, Essence *, AuditResult **);
int  dp_GIR_initialize         (GIR *);
int  dp_GIR_unparse_required_X (Env *);
int  dp_GIR_audit              (VeryShortNum, Essence *, AuditResult **);
void dp_GIR_clear_state        (GIR *);

/************************************************************************************************************
 ***** proc.c ***********************************************************************************************
 ************************************************************************************************************/

int  determine_filenames   (char ***, VeryShortNum *);
void handle_error          (int     , char *);
void handle_warning        (int     , char *);
void initialize_stats      (Stats *);
int  print_welcome_message (void);
int  read_debugging_flags  (void);
void show_interesting_info (Env *, StudPrefs *, Stats *);
void show_startup_info     (Env *);

/************************************************************************************************************
 ***** util.c ***********************************************************************************************
 ************************************************************************************************************/

/***** Parsing Utilities *****/

VeryShortNum parse_term_offered (char *, VeryShortNum, VeryShortNum, VeryShortNum *);
```

```
/***** Unparsing Utilities *****/

char * unparse_GIR_fulfillment       (CrseDesc *, Boolean);
char * unparse_h_level               (CrseDesc *, Boolean);
char * unparse_pass_fail             (CrseDesc *, Boolean);
char * unparse_repeatable_for_credit (CrseDesc *, Boolean);
char * unparse_subj_level            (CrseDesc *, Boolean);
char * unparse_term_offered          (CrseDesc *, Boolean);
char * unparse_units                 (CrseDesc *, Boolean);
char * Boolean_unparse               (Boolean   , Boolean);

/***** Misc Utilities *****/

void   convert_to_upper    (char  *);
void   get_mouse_event     (Env   *, XButtonEvent * );
char * now                 (void);
int    strptr_cmp          (char **, char        **);
void   wait_for_left_click (Env   *);

/*****************************************************************************************************************
 ***** AcadProgs *************************************************************************************************
 *****************************************************************************************************************/

int    AcadProgs_free    (AcadProgs *);
int    AcadProgs_parse   (char      *, AcadProgs *);
char * AcadProgs_unparse (AcadProgs *, Boolean);

/*****************************************************************************************************************
 ***** AuditResult ***********************************************************************************************
 *****************************************************************************************************************/

AuditResult * AuditResult_create  (void);
void          AuditResult_free    (AuditResult *);
void          AuditResult_unparse (AuditResult *);

/*****************************************************************************************************************
 ***** Buttons ***************************************************************************************************
 *****************************************************************************************************************/

void Buttons_free      (Buttons *);
void Buttons_display   (Env     *, char    * [], Buttons *);
int  Buttons_get_press (Env     *, Buttons *);

/*****************************************************************************************************************
 ***** CrseDesc **************************************************************************************************
 *****************************************************************************************************************/

void       CrseDesc_build_ht    (CrseDesc **, UShort);
int        CrseDesc_cmp         (CrseDesc **, CrseDesc  **);
void       CrseDesc_sort        (CrseDesc **, UShort);
void       CrseDesc_sort_on_gir (CrseDesc **, UShort);
void       CrseDesc_sort_on_tag (CrseDesc **, UShort);
int        CrseDesc_cmp_gir     (CrseDesc **, CrseDesc  **);
void       CrseDesc_determine_inv_prereqs (Essence *);
CrseDesc * CrseDesc_search    (char      *, CrseDesc  **, UShort);
int        CrseDesc_get_index (char      *, CrseDesc  **, UShort);
int        CrseDesc_parse     (char     **, VeryShortNum, CrseDesc ***, UShort *, GIR *);
char     * CrseDesc_unparse   (CrseDesc  *, Boolean);

/*****************************************************************************************************************
 ***** CrseNumber ************************************************************************************************
 *****************************************************************************************************************/
.
CrseNumber * CrseNumber_copy    (CrseNumber *);
int          CrseNumber_free    (CrseNumber *);
int          CrseNumber_cmp     (CrseNumber **, CrseNumber **);
Boolean      CrseNumber_similar (CrseNumber  *, CrseNumber *);
CrseNumber * CrseNumber_parse   (char        *, OneNumCode);
char       * CrseNumber_unparse (CrseNumber  *, Boolean);

/*****************************************************************************************************************
 ***** CrseNumberSet *********************************************************************************************
 *****************************************************************************************************************/

CrseNumberSet * CrseNumberSet_copy      (CrseNumberSet *);
int             CrseNumberSet_free      (CrseNumberSet *);
int            CrseNumberSet_insert    (CrseNumberSet *, CrseNumber     *);
int            CrseNumberSet_expand    (CrseNumberSet *, CrseNumber     *);
Boolean        CrseNumberSet_is_member (CrseNumber    *, CrseNumberSet  *);
Boolean        CrseNumberSet_satisfies (StudPrefs     *, CrseSet        *, CrseNumberSet *, Boolean, CrseNumberSet **);
int            CrseNumberSet_parse     (Stream        *, CrseNumberSet **);
char         * CrseNumberSet_unparse   (CrseNumberSet *, Boolean);
void           CrseNumberSet_unparse_X (Env           *, CrseNumberSet  *, int);
int            CrseNumberSet_validate  (CrseNumberSet *);

/*****************************************************************************************************************
 ***** CrseSet ***************************************************************************************************
 *****************************************************************************************************************/

CrseSet * CrseSet_create         (UShort);
CrseSet * CrseSet_new            (char    **);
CrseSet * CrseSet_copy           (CrseSet  *);
void      CrseSet_resize         (CrseSet  *, UShort);
void      CrseSet_free           (CrseSet  *);
void      CrseSet_insert         (CrseSet  *, char    *);
Boolean   CrseSet_is_member      (CrseSet  *, char    *);
int       CrseSet_index          (CrseSet  *, char    *);
CrseSet * CrseSet_merge          (CrseSet  *, CrseSet *);
void      CrseSet_append         (CrseSet  *, CrseSet *);
CrseSet * CrseSet_subtract       (CrseSet  *, CrseSet *);
CrseSet * CrseSet_remove_similar (CrseSet  *, CrseDesc **, UShort);
CrseSet * CrseSet_get_synonyms   (CrseSet  *, CrseDesc **, UShort);
UShort    CrseSet_get_units      (CrseSet  *, CrseDesc **, UShort, Boolean *);
void      CrseSet_sort           (CrseSet  *);
int       CrseSet_parse          (FILE    **, CrseSet **);
void      CrseSet_unparse        (CrseSet  *, Boolean);
void      CrseSet_unparse_X      (Env      *, CrseSet *, int);
void      CrseSet_augment_with_synonyms (CrseSet *, CrseDesc **, UShort);

/*****************************************************************************************************************
 ***** Degrees ***************************************************************************************************
 *****************************************************************************************************************/

int    Degrees_free      (Degrees *);
int    Degrees_parse     (char    *, AcadProgs *, OneNumCode , Degrees **);
char * Degrees_unparse   (Degrees *, AcadProgs *, Boolean);
void   Degrees_unparse_X (Env     *, Degrees   *, AcadProgs *, int);

/*****************************************************************************************************************
 ***** Display ***************************************************************************************************
 *****************************************************************************************************************/

void Display_create     (Env *);
void Display_initialize (Env *);

void Display_clear_screen    (Env *);
void Display_clear_text_area (Env *);

int  Display_print_string                  (Env *, char *);
int  Display_print_string_and_nl           (Env *, char *);
int  Display_print_string_at_x_of_cl       (Env *, char *, int);
int  Display_print_string_and_nl_at_x_of_cl (Env *, char *, int);
int  Display_print_string_at_pos           (Env *, char *, int, int);
int  Display_print_string_and_nl_at_pos    (Env *, char *, int, int);

int  Display_ps     (Env *, char *);
int  Display_psn    (Env *, char *);
int  Display_psx    (Env *, char *, int);
int  Display_psnx   (Env *, char *, int);
int  Display_pspos  (Env *, char *, int, int);
int  Display_psnpos (Env *, char *, int, int);

void Display_update_init_dim   (Env *, int, int, int, int);
void Display_update_init_final (Env *, int, int, int, int);

void Display_center_string (Env *, char *, int, int, int, char *, Boolean, Boolean, Font, XFontStruct *);

/*****************************************************************************************************************
 ***** GIR *******************************************************************************************************
 *****************************************************************************************************************/

void GIR_initialize (GIR *);
void GIR_sort       (GIR *);
```

```
/****************************************************************************************************
 ***** OneOfSCS **************************************************************************************
 ****************************************************************************************************/

OneOfSCS * OneOfSCS_create    (void);
void       OneOfSCS_free      (OneOfSCS *);
void       OneOfSCS_insert    (OneOfSCS *, SubCS    *);
Boolean    OneOfSCS_satisfied (OneOfSCS *, CrseSet  *, CrseSet **, VeryShortNum *, Boolean, Boolean, Env *, int);
void       OneOfSCS_unparse_X (Env      *, OneOfSCS *, int);

/****************************************************************************************************
 ***** RankedCrse ************************************************************************************
 ****************************************************************************************************/

void RankedCrse_initialize   (RankedCrse **, Essence *);
void RankedCrse_free         (RankedCrse  *);
void RankedCrse_determine_ff (RankedCrse  *, Essence *);
void RankedCrse_sort         (RankedCrse  *, UShort, OneNumCode);
void RankedCrse_unparse      (RankedCrse  *);

/****************************************************************************************************
 ***** SetOfSQSCS ************************************************************************************
 ****************************************************************************************************/

void         SetOfSQSCS_free            (SetOfSQSCS *);
SetOfSQSCS * SetOfSQSCS_create          (void);
void         SetOfSQSCS_insert          (SetOfSQSCS *, SubQualSCS *);
void         SetOfSQSCS_set_excludings  (SetOfSQSCS *, CrseSet    *);
void         SetOfSQSCS_free_excludings (SetOfSQSCS *);
Boolean      SetOfSQSCS_satisfied       (SetOfSQSCS *, CrseSet    *, CrseSet **, Boolean, Boolean, Env *, int);
void         SetOfSQSCS_unparse_X       (Env        *, SetOfSQSCS *, int, Boolean, Boolean);

/****************************************************************************************************
 ***** Streams ***************************************************************************************
 ****************************************************************************************************/

Stream  * Stream_new             (FILE   *);
int       Stream_close           (Stream *);
FILE    * Stream_end             (Stream *);
char    * Stream_get             (Stream *);
int       Stream_put             (Stream *, char *);
int       Stream_put_strings     (Stream *, char *);
Boolean   Stream_is_buffer_empty (Stream *);
char    * Stream_unparse         (Stream *, Boolean);

/****************************************************************************************************
 ***** Strings ***************************************************************************************
 ****************************************************************************************************/

Strings * Strings_create         (UShort);
void      Strings_free           (Strings  *);
void      Strings_insert         (Strings  *, char     *);
void      Strings_unparse        (Strings  *, Boolean);
void      Strings_unparse_X      (Env      *, Strings *, int);

/****************************************************************************************************
 ***** StudPrefs *************************************************************************************
 ****************************************************************************************************/

int    StudPrefs_initialize (StudPrefs *);
int    StudPrefs_clear      (StudPrefs *);
int    StudPrefs_parse      (char      *, AcadProgs *, StudPrefs *);
char * StudPrefs_unparse    (StudPrefs *, AcadProgs *, Boolean);
void   StudPrefs_unparse_X  (Env       *, StudPrefs *, AcadProgs *);

/****************************************************************************************************
 ***** StudStatus ************************************************************************************
 ****************************************************************************************************/

int    StudStatus_initialize        (StudStatus  *);
int    StudStatus_clear             (StudStatus  *);
int    StudStatus_save_state        (StudStatus  *);
int    StudStatus_restore_state     (StudStatus  *);
int    StudStatus_parse             (CrseDesc   **, Stats      *, char  *, StudStatus *);
void   StudStatus_compute_taken_all (CrseDesc   **, StudStatus *, Stats *);
char * StudStatus_unparse           (StudStatus  *, Boolean);
```

```
void   StudStatus_unparse_X         (Env         *, StudStatus *);

/*********************************************************************************************************
 ***** SubCS *********************************************************************************************
 *********************************************************************************************************/

SubCS   * SubCS_create    (CrseSet *, char    *, VeryShortNum);
SubCS   * SubCS_copy      (SubCS   *);
void      SubCS_free      (SubCS   *);
Boolean   SubCS_satisfied (SubCS   *, CrseSet *, CrseSet **, Boolean, Boolean, Env *, int);
void      SubCS_unparse   (SubCS   *, Boolean);
void      SubCS_unparse_X (Env     *, SubCS   *, int, Boolean);

/*********************************************************************************************************
 ***** SubQualSCS ****************************************************************************************
 *********************************************************************************************************/

SubQualSCS * SubQualSCS_create    (void);
SubQualSCS * SubQualSCS_copy      (SubQualSCS *);
void         SubQualSCS_free      (SubQualSCS *);
Boolean      SubQualSCS_satisfied (SubQualSCS *, CrseSet    *, CrseSet **, Boolean, Boolean, Env *, int);
void         SubQualSCS_unparse_X (Env        *, SubQualSCS *, int       , Boolean, Boolean);

/*********************************************************************************************************
 ***** Units *********************************************************************************************
 *********************************************************************************************************/

Units   * Units_create    (UShort , char *);
Boolean   Units_satisfied (Units *, UShort, Boolean, Boolean, Env *, int);
void      Units_unparse_X (Env   *, Units *, int);

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

I.I. table_constants.h

```
/*********************************************************************************************************
 * table_constants.h
 *
 * Author        : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *                                MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *
 * Abstract      : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *                 designed and implemented in the spring of 1995 as the author's Master of Engineering thesis, under
 *                 the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.  See main.c for
 *                 a more complete description of AAA.
 *
 *                 This file contains constants which are indices into the lookup tables given in tables.h.  Also
 *                 contained here are constants which are the cardinalities of the lookup tables.
 *
 *                 Any modification to this file may very well require an accompanying modification to tables.h.
 *
 * Compiling Env : None.  This file is self-contained.
 *
 * Notes         : NO = Number Of
 *
 * Code Status                  : 0.5.0 WIP
 * Last Modification Date & Time : April 20, 1995
 *
 *********************************************************************************************************/

/*********************************************************************************************************
 ***** Constants Pertaining to CrseNumberSuffixes ********************************************************
 *********************************************************************************************************/

#define NO_CRSE_NUMBER_SUFFIXES  6

/*********************************************************************************************************
 ***** Constants Pertaining to CrseNumberSpecProgPrefixes ************************************************
 *********************************************************************************************************/

#define NO_CRSE_NUMBER_SPEC_PROG_PREFIXES  16

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

```
 ***** Constants Pertaining to HLevelStrings ***************************************************************************
 **************************************************************************************************************************/

#define NO_H_LEVEL_EXCEPT_TYPES  6

/***** ???: The _EXCEPT_ are never referenced. *****/

enum H_LEVEL_ENUM {
  H_LEVEL_NONE = 0,
  H_LEVEL_GRAD_CREDIT,
  H_LEVEL_EXCEPT_18,
  H_LEVEL_EXCEPT_15,
  H_LEVEL_EXCEPT_2_6_8_12_13_16_18_22,
  H_LEVEL_EXCEPT_2_6_16_18_22
  };

/**************************************************************************************************************************
 ***** Constants Pertaining to GIRFulfillmentStrings ***********************************************************************
 **************************************************************************************************************************/

#define NO_GIR_TYPES  20

enum GIR_FUL_ENUM {
  GIR_FUL_NONE          = 0x0100,  /*  0000 0001 0000 0000  */
  GIR_FUL_HASS          = 0x2001,  /*  0000 0010 0000 0001  */
  GIR_FUL_REST          = 0x0400,  /*  0000 0100 0000 0000  */
  GIR_FUL_LAB           = 0x0800,  /*  0000 1000 0000 0000  */

  GIR_FUL_HASS_D_4      = 0x1010,  /*  0001 0000 0000 1000  */
  GIR_FUL_HASS_D_2      = 0x1004,  /*  0001 0000 0000 0010  */
  GIR_FUL_HASS_D_5      = 0x1020,  /*  0001 0000 0001 0000  */
  GIR_FUL_HASS_D_1      = 0x1002,  /*  0001 0000 0000 0001  */
  GIR_FUL_HASS_D_3      = 0x1008,  /*  0001 0000 0000 0100  */
  GIR_FUL_HASS_D_LO     = 0x1001,  /*  0001 0000 0010 0000  */

  GIR_FUL_PHYS1         = 0x0201,  /*  0010 0000 0000 0001  */
  GIR_FUL_HASS_PHASE_I  = 0x2002,  /*  0000 0010 0000 0010  */

  GIR_FUL_PHYS2         = 0x0202,  /*  0010 0000 0000 0010  */
  GIR_FUL_CALC1         = 0x0204,  /*  0010 0000 0000 0100  */
  GIR_FUL_CALC2         = 0x0208,  /*  0010 0000 0000 1000  */
  GIR_FUL_HASS_PHASE_II = 0x2004,  /*  0000 0010 0000 0100  */

  GIR_FUL_CHEM          = 0x0210,  /*  0010 0000 0001 0000  */
  GIR_FUL_BIO           = 0x0220,  /*  0010 0000 0010 0000  */
  GIR_FUL_PHASE_II      = 0x4000,  /*  0100 0000 0000 0000  */
  GIR_FUL_PHASE_I       = 0x8000   /*  1000 0000 0000 0000  */
  };

enum GIR_FF_ENUM {
  GIR_FF_NONE          = 0x01,
  GIR_FF_SCIENCE       = 0x02,
  GIR_FF_REST          = 0x04,
  GIR_FF_LAB           = 0x08,
  GIR_FF_HASS_D        = 0x10,
  GIR_FF_HASS          = 0x20,
  GIR_FF_PHASE_I       = 0x40,
  GIR_FF_PHASE_II      = 0x80
  };

enum GIR_FFQ_ENUM {
  GIR_FFQ_NONE          = 0x00,

  GIR_FFQ_PHYS1         = 0x01,
  GIR_FFQ_PHYS2         = 0x02,
  GIR_FFQ_CALC1         = 0x04,
  GIR_FFQ_CALC2         = 0x08,
  GIR_FFQ_CHEM          = 0x10,
  GIR_FFQ_BIO           = 0x20,

  GIR_FFQ_HASS_D_LO     = 0x01,
  GIR_FFQ_HASS_D_1      = 0x02,
  GIR_FFQ_HASS_D_2      = 0x04,
  GIR_FFQ_HASS_D_3      = 0x08,
  GIR_FFQ_HASS_D_4      = 0x10,
  GIR_FFQ_HASS_D_5      = 0x20,
  GIR_FFQ_HASS          = 0x01,
  GIR_FFQ_HASS_PHASE_I  = 0x02,
  GIR_FFQ_HASS_PHASE_II = 0x04

  };

enum GIR_FUL_INDEX_ENUM {
  GIR_FUL_INDEX_NONE = 0,
  GIR_FUL_INDEX_HASS,
  GIR_FUL_INDEX_REST,
  GIR_FUL_INDEX_LAB,

  GIR_FUL_INDEX_HASS_D_4,
  GIR_FUL_INDEX_HASS_D_2,
  GIR_FUL_INDEX_HASS_D_5,
  GIR_FUL_INDEX_HASS_D_1,
  GIR_FUL_INDEX_HASS_D_3,
  GIR_FUL_INDEX_HASS_D_LO,

  GIR_FUL_INDEX_PHYS1,
  GIR_FUL_INDEX_HASS_PHASE_I,

  GIR_FUL_INDEX_PHYS2,
  GIR_FUL_INDEX_CALC1,
  GIR_FUL_INDEX_CALC2,
  GIR_FUL_INDEX_HASS_PHASE_II,

  GIR_FUL_INDEX_CHEM,
  GIR_FUL_INDEX_BIO,
  GIR_FUL_INDEX_PHASE_II,
  GIR_FUL_INDEX_PHASE_I
  };

#define NO_HASS_FULFILLMENT_CODES      9
#define NO_PHASE_I_FULFILLMENT_CODES   2
#define NO_PHASE_II_FULFILLMENT_CODES  2

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

#define NO_ACAD_PROGS_FIELD_CODES 6

enum ACAD_PROGS_FIELD_CODES_ENUM {
  APF_MAJORS_FULL_NAMES = 0,
  APF_MAJORS_SHORT_NAMES,
  APF_MAJORS,
  APF_MINORS_SCI_ENG,
  APF_MINORS_HASS,
  APF_CONCS
  };

#define NO_STUD_STATUS_FILE_FIELD_CODES 11

enum STUD_STATUS_FILE_FIELD_CODES_ENUM {
  FIELD_NAME = 0,
  FIELD_YEAR,
  FIELD_THIS_TERM,
  FIELD_NEXT_TERM,
  FIELD_COURSES_ALREADY_TAKEN,
  FIELD_OTHER_UNITS_RECEIVED,
  FIELD_THESIS_UNITS_COMPLETED,
  FIELD_PHASE_I_COMPLETED,
  FIELD_PHASE_II_COMPLETED,
  FIELD_SWIMMING_REQUIREMENT_COMPLETED,
  FIELD_ACCUMULATED_PE_POINTS
  };

#define NO_STUD_STATUS_YEAR_FIELD_VALUE_CODES 5

enum STUD_STATUS_YEAR_FIELD_VALUE_CODES_ENUM {
  FRESHMAN = 0,
  SOPHOMORE,
  JUNIOR,
  SENIOR,
  FIFTH
  };

#define NO_SEASON_CODES 4
```

```
enum SEASON_CODES_ENUM  {
  FALL = 0,
  IAP,
  SPRING,
  SUMMER
  };

#define NO_STUD_PREFS_FILE_FIELD_CODES 18

enum STUD_PREFS_FILE_FIELD_CODES_ENUM {
  FIELD_DATE_OF_GRAD = 0,
  FIELD_MAJORS,
  FIELD_MINORS,
  FIELD_CONC,

  FIELD_MAX_COURSE_LOAD,
  FIELD_MAX_CLASS_HOURS,
  FIELD_MAX_LAB_HOURS,
  FIELD_MAX_PREP_HOURS,

  FIELD_MIN_COURSE_LOAD,
  FIELD_MIN_CLASS_HOURS,
  FIELD_MIN_LAB_HOURS,
  FIELD_MIN_PREP_HOURS,

  FIELD_WANT_TO_TAKE_NEXT_TERM,
  FIELD_WANT_TO_TAKE,
  FIELD_DONT_WANT_TO_TAKE,

  FIELD_OVERRIDE_POI,
  FIELD_CONSIDER_UNDERGRAD,
  FIELD_CONSIDER_GRAD
  };

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

enum MAIN_SCREEN_OPTIONS_ENUM {
  MS_OPT_DISP_INFO = 0,
  MS_OPT_PARSE_SSF,
  MS_OPT_PARSE_SPF,
  MS_OPT_DISP_SS,
  MS_OPT_DISP_SP,
  MS_OPT_SHOW_GIR,
  MS_OPT_SHOW_VI_P,
  MS_OPT_AUDIT_GIR,
  MS_OPT_AUDIT_VI_P,
  MS_OPT_GEN_SCH,
  MS_OPT_QUIT
  };

enum MSG_ENUM {
  MSG_NONE  = 0,
  MSG_INITIALIZE,
  MSG_UNPARSE_REQUIRED,
  MSG_UNPARSE_TAKEN,
  MSG_UNPARSE_SATISFIED,
  MSG_UNPARSE_REQUIRED_X,
  MSG_AUDIT
  };

enum FLAGS_ENUM {
  FLAG_NONE = 0,
  FLAG_DISPLAY_AUDIT_RESULTS
  };
```

I.J. tables.h

```
/******************************************************************************************************************
 * tables.h
 *
 * Author        : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *                              MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *
 * Abstract      : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *                designed and implemented in the spring of 1995 as the author's Master of Engineering thesis, under
 *                the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.  See main.c for
 *                a more complete description of AAA.
 *
 *                This file contains various lookup tables used by AAA.  The file table_constants.h contains
 *                constants directly related to this file.  Thus, any modification to this file may very well
 *                require an accompanying modification to that one.
 *
 * Compiling Env : #include "constants.h"
 *
 * Code Status                   : 0.5.0 WIP
 * Last Modification Date & Time : April 20, 1995
 *
 ***************************************************************************************************************/

/***************************************************************************************************************
 * CrseNumberSuffixes - Course Number Suffixes
 *
 * This table enumerates all possible strings that can appear as the second part of a course number, when the first
 * part is just a department designation (Ex: 6 ThG).
 *
 ***************************************************************************************************************/

const char * CrseNumberSuffixes[] = {
  "UR","URG","ThU","ThG","ThT","IND"
  };

/***************************************************************************************************************
 * CrseNumberSpecProgPrefixes - Course Number Special Program Prefixes
 *
 * This table enumerates all possible strings that can appear as the first part of a course number, when that part
 * is *not* one of the purely numerical department designations (Ex: 21M 240).
 *
 ***************************************************************************************************************/

const char * CrseNumberSpecProgPrefixes[] = {
  "HST","MAS","AS","MS","NS","SP","STS","TPP","TOX","SEM",
  "21A","21F","21H","21L","21M","21W"
  };

/***************************************************************************************************************
 * HLevelStrings - H-Level Strings
 *
 * This table enumerates all possible strings that can describe the H-Level information about a course.
 *
 * You may think it was a lousy hack to hard-code all these, but considering that there were only five possible,
 * (not counting NONE), writing a parser to decipher the roman numerals simply was not worth it.
 *
 ***************************************************************************************************************/

/***** The first and the last elements in this next table comprise a hack to ease unparsing. *****/

const char * HLevelExceptStrings[] = {
  "NONE",
  "H-LEVEL Grad Credit",                             /* 1522 of these */
  "(H except XVIII)",                                /* 14   of these in the Course Bulletin */
  "(H except XV)",                                   /* 1    of these */
  "(H except II, VI, VIII, XII, XIII, XVI, XVIII, XXII)",  /* 1    of these */
  "(H except II, VI, XVI, XVIII, XXII)"              /* 1    of these */
  };

/***************************************************************************************************************
 * GIRFulfillmentString - GIR Fulfillment Strings
 *
 * This table enumerates all possible strings that can describe the GIR fulfillment information about a course.
 *
 ***************************************************************************************************************/

const UShort GIRFulfillmentCodes[] = {
  0x0100, 0x2001, 0x0400, 0x0800,
  0x1010, 0x1004, 0x1020, 0x1002, 0x1008, 0x1001,
  0x0201, 0x2002,
  0x0202, 0x0204, 0x0208, 0x2004,
  0x0210, 0x0220, 0x4000, 0x8000
  };

char * GIRFulfillmentStrings[] = {
  "NONE",
  "HASS",                    /* 435 */
  "REST",                    /*  52 */
  "Institute LAB",           /*  52 */

  "HASS-D, Category 4",      /*  29 */
  "HASS-D, Category 2",      /*  18 */
  "HASS-D, Category 5",      /*  18 */
  "HASS-D, Category 1",      /*  14 */
  "HASS-D, Category 3",      /*  13 */
  "HASS-D, Language Option", /*  12 */

  "PHYSICS I",               /*   4 */
  "HASS, Phase One WRIT",    /*   4 */

  "PHYSICS II",              /*   3 */
  "CALC I",                  /*   3 */
  "CALC II",                 /*   3 */
  "HASS, Phase Two WRIT",    /*   3 */

  "CHEMISTRY",               /*   2 */
  "BIOLOGY",                 /*   1 */
  "Phase Two WRIT",          /*   1 */
  "Phase One WRIT"           /*   0 */
  };

char * GIRStrings[] = {  /* these are used for printing */
  "NONE",
  "HASS",
  "REST",
  "Institute LAB",

  "Category 4",
  "Category 2",
  "Category 5",
  "Category 1",
  "Category 3",
  "Language Option",

  "Physics I",
  " ",

  "Physics II",
  "Calculus I",
  "Calculus II",
  " ",

  "Chemistry",
  "Biology",
  " ",
  " "
  };

const OneNumCode HASSFulfillmentCodes[] = {
  GIR_FUL_INDEX_HASS,
  GIR_FUL_INDEX_HASS_D_4,
  GIR_FUL_INDEX_HASS_D_2,
  GIR_FUL_INDEX_HASS_D_5,
  GIR_FUL_INDEX_HASS_D_1,
  GIR_FUL_INDEX_HASS_D_3,
  GIR_FUL_INDEX_HASS_D_LO,
  GIR_FUL_INDEX_HASS_PHASE_I,
  GIR_FUL_INDEX_HASS_PHASE_II
  };

const OneNumCode PhaseIFulfillmentCodes[] = {
  GIR_FUL_INDEX_HASS_PHASE_I,
  GIR_FUL_INDEX_PHASE_I
  };

const OneNumCode PhaseIIFulfillmentCodes[] = {
  GIR_FUL_INDEX_HASS_PHASE_II,
  GIR_FUL_INDEX_PHASE_II
  };

/*******************************************************************************************************************
 * ErrorStrings - Error Strings
 *
 * This table includes all error messages that may be produced by AAA at run time.
 *
 *****************************************************************************************************************/

const char * ErrorStrings[] = {
  "NONE",

  "Memory allocation failure",

  "File open (fopen) failure",
  "File close (fclose) failure",

  "Printf failure",
  "Sprintf failure",

  "Bad course description file",
  "Bad debugging flags file",
  "Bad course description files listing file",

  "Course number is longer than 7 characters",

  "Stream buffer is full",
  "Stream buffer is not empty as presumed",

  "Bad student status file",
  "Invalid field in the student status file",
  "Invalid year for the year field in the student status file",
  "Invalid season for the This-Term field in the student status file",
  "Invalid year for the This-Term field in the student status file",
  "Invalid season for the Next-Term field in the student status file",
  "Invalid year for the Next-Term field in the student status file",
  "Attempt to save the StudStatus state when one is saved already",
  "Attempt to restore the StudStatus state when none has been saved",

  "Bad student preferences file",
  "Invalid field in the student preferences file",
  "Invalid season for the Desired-Date-of-Graduation field in the student preferences file",
  "Invalid year for the Desired-Date-of-Graduation field in the student preferences file",

  "Bad academic programs file",

  "Invalid degree type",
  "Invalid degree name",

  "Out of bounds error in the display",

  "An invalid course number in the form of A-B-C (i.e., a 'three-range')",

  "Unknown message sent to a degree module",

  "Unidentifiable course number encountered",

  "Capacity is full",
  "A NULL CrseSet cannot be appended to",

  "Not yet implemented"
  };

/*****************************************************************************************************************
 * dfs - Debugging Flags Set
 *
 * This is a hash table of debugging flags.
 *
 * For example, if (dfs[PRINT_MAIN_NUM_OF_CRSE_DESCS] == TRUE), then the number of course descriptions will be
 * printed to output at runtime.
 *
 *****************************************************************************************************************/

Boolean dfs [NO_DEBUGGING_FLAGS];  /* debugging flags */

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

const char * AcademicProgramsFields[] = {
  "MajorProgramsFullNames",
  "MajorProgramsShortNames",
  "MajorPrograms",
    "MinorProgramsInScienceAndEngineering",
    "MinorProgramsInHASS",
    "FieldOfConcentration"
    };

  char * StudStatusFileFields[] = {
    "Name",
    "Year",
    "This-Term",
    "Next-Term",
    "Courses-Already-Taken",
    "Other-Units-Received",
    "Thesis-Units-Completed",
    "Phase-I-Completed?",
    "Phase-II-Completed?",
    "Swimming-Requirment-Completed?",
    "Accumulated-PE-Points"
    };

  char * StudStatusYearFieldValues[] = {
    "FRESHMAN",
    "SOPHOMORE",
    "JUNIOR",
    "SENIOR",
    "GRAD"
    };

  char * SeasonNames[] = {
    "FALL",
    "IAP",
    "SPRING",
    "SUMMER"
    };

  char * StudPrefsFileFields[] = {
    "Desired-Date-of-Graduation",
    "Desired-Degree(s)",
    "Desired-Minor(s)",
    "Desired-Concentration",
    "Maximum-Course-Load",
    "Maximum-Class-Hours",
    "Maximum-Lab-Hours",
    "Maximum-Prep-Hours",
    "Minimum-Course-Load",
    "Minimum-Class-Hours",
    "Minimum-Lab-Hours",
    "Minimum-Prep-Hours",
    "Want-To-Take-Next-Term",
    "Want-To-Take",
    "Don't-Want-To-Take",
    "Override-Permission-of-Instructor?",
    "Consider-Undergrad-Courses?",
    "Consider-Grad-Courses?"
    };

  /*
  OneNumCode StudPrefsFileFieldsDatatypes = {
    DT_TERM_PTR,
    DT_DEGREES_PTR,
    DT_DEGREES_PTR,
    DT_DEGREES_PTR,

    DT_VERY_SHORT_NUM,
    DT_VERY_SHORT_NUM,
    DT_VERY_SHORT_NUM,
    DT_VERY_SHORT_NUM,
    DT_VERY_SHORT_NUM,
    DT_VERY_SHORT_NUM,
    DT_VERY_SHORT_NUM,
    DT_VERY_SHORT_NUM,

    DT_CRSE_SET_PTR,
    DT_CRSE_SET_PTR,
    DT_CRSE_SET_PTR,

    DT_BOOLEAN,
    DT_BOOLEAN,
  DT_BOOLEAN
  };
*/

char * MainScreenOptions[] = {
  /*2345678901234567890123456789*/
  "Display Interesting Info",
  "SPACE",
  "Parse Student Status File",
  "Parse Student Prefs File",
  "SPACE",
  "Display Student Status",
  "Display Student Prefs",
  "SPACE",
  "Show GIR Reqs",
  "Show VI-P Reqs",
  "SPACE",
  "Audit GIR",
  "Audit VI-P",
  "SPACE",
  "Generate Schedule".
  "SPACE",
  "Quit Me, Baby!",
  "END"
  };

char * Messages[] = {
  "None",
  "Initialize",
  "Unparse Required",
  "Unparse Taken",
  "Unparse Satisfied",
  "Unparse Required for X Display",
  "Audit"
  };

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

II. Datatypes

II.A. AcadProgs.c

```
/****************************************************************************************************************
 ***** General Info *********************************************************************************************
 ****************************************************************************************************************
 *****
 ***** File          : AcadProgs.c
 *****
 ***** Author        : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *****                                MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *****
 ***** Background    : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *****                 designed and implemented in the spring of 1995 as the author's Master of Engineering thesis,
 *****                 under the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.
 *****                 See main.c for a more complete description of AAA.
 *****
 *****                 This file contains the source code for the AcadProgs datatype.
 *****
 ***** Compiling Env : gcc -ansi -pedantic -c AcadProgs.c
 *****
 ***** Code Status                   : 1.0.0 WIP
 ***** Last Modification Date & Time : April 27, 1995
 *****
 ****************************************************************************************************************
 ***** Overview of Datatype *************************************************************************************
 ****************************************************************************************************************
 *****
 *****
 *****
 ****************************************************************************************************************
 ***** Operations on Datatype ***********************************************************************************
 ****************************************************************************************************************
 *****
 ***** int AcadProgs_free (AcadProgs * AP)
 *****
 ***** Requires : - none
 ***** Modifies : - AP
 ***** Effects  : - Frees all memory associated with AP, but not AP itself.
 *****            - Returns ERROR_NONE.
 *****
 ******************************************************************************************************
 *****
 ***** int AcadProgs_parse (char * fn, AcadProgs * AP)
 *****
 ***** Requires : - fn is the name of a file containing information about academic programs.
 *****            - AP is a non-NULL AcadProgs instance.
 ***** Modifies : - AP
 ***** Effects  : - Parses the academic program information in the file named by fn into AP.
 *****            - Returns any errors.
 ***** Bugs     : - Does not handle mid-line comments.
 *****
 ******************************************************************************************************
 *****
 ***** char * AcadProgs_unparse (AcadProgs * AP, Boolean should_return_result)
 *****
 ***** Requires : - none
 ***** Modifies : - stdout
 ***** Effects  : - If should_return_result is TRUE, returns a string representation of AP.
 *****            - Else, prints a string representation of AP to stdout.  Returns NULL.
 *****            - Exits via handle_error if error is incurred.
 *****
 ******************************************************************************************************
 ******************************************************************************************************
 ******************************************************************************************************/

/******************************************************************************************************
 ***** Header Files ***********************************************************************************
 ******************************************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"                        /* MLO_ */
#include "primitive_datatype_reps_constants.h"
#include "table_constants.h"                  /* NO_ACAD_PROGS_FIELD_CODES */
#include "error_and_trace.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"                    /* AcadProgs */
#include "externs.h"                          /* AcademicProgramsFields */

/******************************************************************************************************
 ***** AcadProgs_free *********************************************************************************
 ******************************************************************************************************/

int AcadProgs_free (AcadProgs * AP)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - AP
   ***** Effects  : - Frees all memory associated with AP, but not AP itself.
   *****            - Returns ERROR_NONE.
   *****/

  int i = 0;
  int j = 0;

  for (i = 0 ; i < AP->count ; i++) {
    for (j = 0 ; j < (AP->categories[i])->count ; j++)
      free ((AP->categories[i])->members[j]);
    free ((AP->categories[i])->members);
  }
  free (AP->categories);
}

/******************************************************************************************************
 ***** AcadProgs_parse ********************************************************************************
 ******************************************************************************************************/

int AcadProgs_parse (char * fn, AcadProgs * AP)
```

```
{
  FILE * fp         = NULL;
  char   one_string [MLO_ONE_STRING];
  int    i          = 0;
  AcadProgCat * apc = NULL;

  fp = fopen(fn, "r");
  AP->categories = (AcadProgCat **) calloc (NO_ACAD_PROGS_FIELD_CODES, sizeof(AcadProgs *));
  AP->count      = 0;

  while (TRUE) {
    while (TRUE) {
      fscanf(fp, "%s", one_string);
      if ((*one_string != '#') &&
          (*one_string != NEWLINE)) break;
    }
    for (i = 0 ; i < NO_ACAD_PROGS_FIELD_CODES ; i++)
      if (!strcmp(one_string, AcademicProgramsFields[i])) break;
    if (i == NO_ACAD_PROGS_FIELD_CODES) return (ERROR_BAD_ACAD_PROGS_FILE);
    apc = (AcadProgCat *) malloc (sizeof(AcadProgCat));
    apc->code    = i;
    apc->members = (char **) calloc (MNO_DEGREES_IN_CAT, sizeof(char *));
    apc->count   = 0;
    fscanf(fp, "%s", one_string);
    if (strcmp(one_string, "(")) return (ERROR_BAD_ACAD_PROGS_FILE);
    while (TRUE) {
      fscanf (fp, "%*[\12]"); /* read newline */
      fscanf (fp, UP_TO_NEWLINE, one_string);
      if (!strcmp(one_string, ")")) break;
      else {
        apc->members[apc->count] = (char *) malloc ((strlen(one_string) + 1) * sizeof(char));
        strcpy(apc->members[(apc->count)++], one_string);
      }
    }
    apc->members = (char **) realloc (apc->members, apc->count * sizeof(char *));
    *(AP->categories + (AP->count)++) = apc;
  }
}

/******************************************************************************************************************
 ***** AcadProgs_unparse ******************************************************************************************
 ******************************************************************************************************************/

char * AcadProgs_unparse (AcadProgs * AP, Boolean should_return_result)
{
  char * c      = NULL;
  int i, j;

  if (should_return_result)
    handle_error (ERROR_NOT_YET_IMPLEMENTED, "executing AcadProgs_unparse with should_return_result = TRUE");
  else {
    for (i = 0 ; i < AP->count ; i++) {
      printf("%s\n", AcademicProgramsFields[(AP->categories[i])->code]);
      for (j = 0 ; j < (AP->categories[i])->count ; j++)
        printf("\t%s\n", (AP->categories[i])->members[j]);
    }}
}

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

II.B. AuditResult.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps_constants.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
```

```c
#include "prototypes.h"

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

AuditResult * AuditResult_create(void)
{
  AuditResult * retval = NULL;
  retval = (AuditResult *) calloc (1, sizeof(AuditResult));
  retval->applicable = CrseSet_create(MNO_APPLICABLE);
  retval->appl_units = 0;
  return (retval);
}

void AuditResult_free (AuditResult * ar)
{
  if (ar == NULL) return;
  CrseSet_free (ar->applicable);
  Strings_free (ar->completed);
  Strings_free (ar->not_completed);
  Strings_free (ar->not_audited);
  free (ar);
}

void AuditResult_unparse (AuditResult * ar)
{
  CrseSet_unparse (ar->applicable   , FALSE); printf("\n");
  Strings_unparse (ar->completed    , FALSE); printf("\n");
  Strings_unparse (ar->not_completed, FALSE); printf("\n");
  Strings_unparse (ar->not_audited  , FALSE); printf("\n");
}
```

II.C. Buttons.c

```c
/*****************************************************************************************************************
 ***** General Info ***********************************************************************************************
 *****************************************************************************************************************
 *****
 ***** File         : Buttons.c
 *****
 ***** Author       : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *****                               MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *****
 ***** Background   : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *****                designed and implemented in the spring of 1995 as the author's Master of Engineering thesis,
 *****                under the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.
 *****                See main.c for a more complete description of AAA.
 *****
 *****                This file contains the source code for the AcadProgs datatype.
 *****
 ***** Compiling Env : gcc -ansi -pedantic -c Buttons.c
 *****
 ***** Code Status                  : 1.0.0 WIP
 ***** Last Modification Date & Time : April 27, 1995
 *****
 *****************************************************************************************************************
 ***** Overview of Datatype ***************************************************************************************
 *****************************************************************************************************************
 *****
 *****
 *****
 *****************************************************************************************************************
 ***** Operations on Datatype *************************************************************************************
 *****************************************************************************************************************
 *****
 ***** void Buttons_free (Buttons * btns)
 *****
 *****************************************************************************************************************
 *****
 ***** void Buttons_display (Env * env, char * button_texts[], int n, Buttons * curr_buttons)
 *****
 *****************************************************************************************************************
 *****
 ***** int Buttons_get_press (Env * env, Buttons * btns)
 *****
 *****************************************************************************************************************
 *****
 ***** int determine_button_number (Buttons * btns, int x, int y)
 *****
 ***** INTERNAL ONLY
 *****
 ***************************************************************************************************************
 *****
 ***** void blink_button (Env * env, Buttons * btns, int btn_index, int cond)
 *****
 ***** INTERNAL ONLY
 *****
 ***************************************************************************************************************
 ***************************************************************************************************************
 ***************************************************************************************************************/

/***************************************************************************************************************
 ***** Header Files ********************************************************************************************
 ***************************************************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"
#include "primitive_datatype_reps_constants.h"
#include "table_constants.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "prototypes.h"

/***************************************************************************************************************
 ***** Protypes of Internal Functions **************************************************************************
 ***************************************************************************************************************/

static int  determine_button_number (Buttons *, int      , int);
static void blink_button            (Env     *, Buttons *, int, int);

/***************************************************************************************************************
 ***** Buttons_free ********************************************************************************************
 ***************************************************************************************************************/

void Buttons_free (Buttons * btns)
{
  int i = 0;

  if (btns == NULL) return;
  for (i = 0 ; i < btns->n ; i++) if (btns->members[i] != NULL) free (btns->members[i]);
  free (btns->members);
  free (btns);
}

/***************************************************************************************************************
 ***** Buttons_display *****************************************************************************************
 ***************************************************************************************************************/

void Buttons_display (Env * env, char * button_texts[], Buttons * curr_buttons)
{
  Button * one_button = NULL;
  int      i          = 0;
  int      posy       = 0;
  int      count      = 0;

  Buttons_free (curr_buttons);

  curr_buttons->members = (Button **) calloc (MNO_OPTIONS, sizeof(Button *));

  XSetForeground (env->the_display, env->the_context, env->white);
  posy = env->text_area_top;

  while (TRUE) {
    if (!strcmp(button_texts[i], "END")) break;
    if (!strcmp(button_texts[i], "SPACE")) {
      i++;
      posy += BUTTON_SPACE;
      continue;
    }
    one_button = (Button *) malloc (sizeof(Button));
    one_button->label   = button_texts[i];
    one_button->outer_x = BUTTON_AREA_X + ((WINDOW_WIDTH - BUTTON_AREA_X - BUTTON_WIDTH) / 2);
    one_button->outer_y = posy;
    one_button->outer_w = BUTTON_WIDTH;
    one_button->outer_h = BUTTON_HEIGHT;

    one_button->inner_x = one_button->outer_x + BUTTON_INNER_GAP_X;
    one_button->inner_y = one_button->outer_y + BUTTON_INNER_GAP_Y;
    one_button->inner_w = BUTTON_WIDTH  - 2 * BUTTON_INNER_GAP_X;
    one_button->inner_h = BUTTON_HEIGHT - 2 * BUTTON_INNER_GAP_Y;

    XDrawRectangle (env->the_display, env->the_buffer, env->the_context,
                    one_button->outer_x, one_button->outer_y, one_button->outer_w, one_button->outer_h);
    XDrawRectangle (env->the_display, env->the_buffer, env->the_context,
                    one_button->inner_x, one_button->inner_y, one_button->inner_w, one_button->inner_h);

    Display_center_string (env, BUTTON_FONT, one_button->inner_x, one_button->inner_w,
                   one_button->inner_y + ((one_button->inner_h - env->button_font_height) / 2),
                   one_button->label,
                   FALSE, FALSE, env->button_font_id, env->button_font);

    curr_buttons->members[count++] = one_button;
    posy += BUTTON_Y_GAP + BUTTON_HEIGHT;
    i++;
  }

  curr_buttons->members = (Button **) realloc (curr_buttons->members, count * sizeof(Button *));
  curr_buttons->n       = count;

  Display_update_init_final (env, BUTTON_AREA_X, env->text_area_top, WINDOW_WIDTH, WINDOW_HEIGHT);
}

/**************************************************************************************************************
 ***** Buttons_get_press **************************************************************************************
 **************************************************************************************************************/

int Buttons_get_press (Env * env, Buttons * btns)
{
  XButtonEvent     xbe;

  int     button_pressed    = -1;
  int     button_released   = -1;

  while (TRUE) {
    get_mouse_event (env, &xbe);
    if (xbe.button == 1) {
      if (xbe.type == ButtonPress) {
        if ((button_pressed  = determine_button_number(btns, xbe.x, xbe.y)) >= 0)
          blink_button (env, btns, button_pressed, BUTTON_PRESSED);
      } else {
        if (button_pressed >= 0) blink_button (env, btns, button_pressed, BUTTON_RELEASED);
        button_released = determine_button_number(btns, xbe.x, xbe.y);
      }
      if ((button_pressed == button_released) && (button_pressed >= 0)) break;
    }}
  return (button_pressed);
}

/**************************************************************************************************************
 ***** determine_button_number ********************************************************************************
 **************************************************************************************************************/

int determine_button_number (Buttons * btns, int x, int y)
{
  int      i;
  Button * b = NULL;

  for (i = 0; i < btns->n ; i++) {
    b = btns->members[i];
    if (((b->outer_x < x) && (x < (b->outer_x + BUTTON_WIDTH))) &&
        ((b->outer_y < y) && (y < (b->outer_y + BUTTON_HEIGHT)))) {
      return (i);
      break;
    }}
  return (-1);
}
```

```
/*****************************************************************************************************
 ***** blink_button ***********************************************************************************
 *****************************************************************************************************/

void blink_button (Env * env, Buttons * btns, int btn_index, int cond)
{
  Button * b          = NULL;

  b = btns->members[btn_index];
  XSetFont (env->the_display, env->the_context, env->button_font_id);

  if (cond == BUTTON_PRESSED) {
    XSetForeground (env->the_display, env->the_context, env->white);
    XFillRectangle (env->the_display, env->the_buffer,  env->the_context,
                    b->inner_x, b->inner_y, b->inner_w, b->inner_h);
    XSetForeground (env->the_display, env->the_context, env->black);
  } else {
    XSetForeground (env->the_display, env->the_context, env->black);
    XFillRectangle (env->the_display, env->the_buffer,  env->the_context,
                    b->inner_x, b->inner_y, b->inner_w, b->inner_h);
    XSetForeground (env->the_display, env->the_context, env->white);
    XDrawRectangle (env->the_display, env->the_buffer, env->the_context,
                    b->inner_x, b->inner_y, b->inner_w, b->inner_h);
  }
  Display_center_string (env, BUTTON_FONT, b->inner_x, b->inner_w,
                 b->inner_y + ((b->inner_h - env->button_font_height) / 2),
                 b->label,
                 FALSE, FALSE, env->button_font_id, env->button_font);

  Display_update_init_dim (env, b->inner_x, b->inner_y, b->inner_w + 1, b->inner_h + 1);
}

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

II.D. CrseDesc.c

```
/*****************************************************************************************************
 ***** General Info ***********************************************************************************
 *****************************************************************************************************
 *****
 ***** File          : CrseDesc.c
 *****
 ***** Author        : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *****                                MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *****
 ***** Background    : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *****                 designed and implemented in the spring of 1995 as the author's Master of Engineering thesis,
 *****                 under the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.
 *****                 See main.c for a more complete description of AAA.
 *****
 *****                 This file contains the source code for the CrseDesc datatype.
 *****
 ***** Compiling Env : gcc -ansi -pedantic -c CrseDesc.c
 *****
 ***** Code Status                   : 1.0.0 WIP
 ***** Last Modification Date & Time : April 27, 1995
 *****
 *****************************************************************************************************
 ***** Overview of Datatype ***************************************************************************
 *****************************************************************************************************
 *****
 ***** An instance of this datatype corresponds to a course description as provided in the MIT Course Bulletin.
 *****
 ***** The components of this datatype naturally correspond to the course attributes provided in the Bulletin.
 ***** There are some details worthy of note:
 *****
 ***** - Engineering Design (ED) points information not included, since that information exists only for
 *****   the Course 6 courses.
 *****
 ***** - Course description and the professors teaching the course are also not included.
```

```
*****
***** Example    : The following is the description from the Bulletin and the human-readable representation of the
*****              corresponding CrseDesc instance for the notorious 6.111.  In the CrseDesc instance, structs are
*****              represented by members enclosed inside brackets.
*****
*****                  **************************** Bulletin ***************************
*****
*****                  6.111 Introductory Digital Systems
*****                  Laboratory
*****                  -------------------------------------------------------
*****                  Prereq.: 6.002 or 6.071 or 16.040
*****                  U (1, 2)
*****                  3-7-2  Institute LAB
*****                  -------------------------------------------------------
*****                  Lectures and labs on digital logic, flipflops, PALs, counters,
*****                  timing, synchronization, finite-state machines and microprogrammed
*****                  systems prepare students for the design and implementation of a final
*****                  project of their choice, e.g., games, music, digital filters, graphics,
*****                  etc. Possible use of lab report for Phase II of the Writing Requirement.
*****                  Six extra units possible via adding 6.919 after project proposal.
*****                  12 Engineering Design Points.
*****                  J. L. Kirtley, Jr., D. E. Troxel
*****
*****                  *********************** CrseDesc instance ***********************
*****
*****                  number                        : [6.111, 1]
*****                  name                          : Introductory Digital Systems Laboratory
*****                  former_number                 : NULL
*****                  same_subj_as                  : NULL
*****                  meets_with                    : NULL
*****                  prerequisites                 : [6.002 or 6.071 or 16.040, 3]
*****                  term_offered                  : This Year : ( 1 2 )   Next Year : ( 1 2 )
*****                  subject_level                 : Undergraduate
*****                  units_arranged                : FALSE
*****                  class_hours                   : 3
*****                  lab_hours                     : 7
*****                  prep_hours                    : 2
*****                  GIR_fulfillment               : Institute LAB
*****                  pass_fail                     : FALSE
*****                  h_level                       : NONE
*****                  repeatable_for_credit         : FALSE
*****                  no_add_credit_for             : NULL
*****
*******************************************************************************************************************
***** Operations on Datatype **************************************************************************************
*******************************************************************************************************************
*****
***** int CrseDesc_cmp (CrseDesc ** cd1, CrseDesc ** cd2)
*****
***** Requires : - Both (*cd1)->number and (*cd2)->number are non-NULL CrseNumberSet instances.
***** Modifies : - none
***** Effects  : - Compares the string representations s1 and s2 of the course numbers of *cd1 and *cd2.
*****            - Returns < 0 if s1 < s2
*****                      = 0 if s1 = s2
*****                      > 0 if s1 > s2
*****
*******************************************************************************************************************
*****
***** CrseDesc * CrseDesc_search (CrseNumber * key, CrseDesc ** cds, UShort count)
*****
*******************************************************************************************************************
*****
***** CrseDesc * search_recur (CrseNumber * key, CrseDesc ** cds, int low, int high)
*****
***** INTERNAL ONLY
*****
*******************************************************************************************************************
*****
***** int CrseDesc_parse (char ** cd_filenames, VeryShortNum cd_files_count, CrseDesc *** cd, UShort * cd_count)
*****
***** Requires : - cd_filenames is an array of cd_files_count filenames, naming files which contain
*****              course descriptions in the Course Bulletin style.
***** Modifies : - *cd, *cd_count
***** Effects  : - Parses the files named by cd_filenames into an array of CrseDesc pointers pointed to by *cd.
*****            - Sets *cd_count to the number of course descriptions parsed.
*****            - Returns any errors except ERROR_BAD_CRSE_DESC_FILE.  For that error, exits via handle_error.
*****
```

```
 *****************************************************************************************************
 *****
 ***** char * CrseDesc_unparse (CrseDesc * cd, Boolean should_return_result)
 *****
 ***** Requires : - none
 ***** Modifies : - stdout
 ***** Effects  : - If should_return_result is TRUE, returns a string representation of cd.
 *****            - Else, prints a string representation of cd to stdout.  Returns NULL.
 *****            - Exits via handle_error if error is incurred.
 *****
 *****************************************************************************************************
 *****************************************************************************************************
 *****************************************************************************************************/

/*****************************************************************************************************
 ***** Header Files **********************************************************************************
 *****************************************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"                    /* MLO_, Parsing Constants, Unparsing Constants   */
#include "primitive_datatype_reps_constants.h"
#include "datatype_reps_constants.h"      /* TERM_OFFERED_, etc.                             */
#include "table_constants.h"              /* NO_                                             */
#include "error_and_trace.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"                /* OneNumCode, CrseNumber, etc.                    */
#include "prototypes.h"                   /* function prototypes                             */
#include "externs.h"                      /* CrseNumberSpecProgPrefixes, CrseNumberSuffixes */

/*****************************************************************************************************
 ***** The Hash Table Stuff - May God Help Us! *******************************************************
 *****************************************************************************************************/

#define SIZE_OF_HASH_TABLE  10008  /* prime number 10007 30011 */

typedef struct entry_t_
{
  UShort            index;
  struct entry_t_ * next;
} entry_t;

static entry_t * ht[SIZE_OF_HASH_TABLE];

/*****************************************************************************************************
 ***** Protypes of Internal Functions ****************************************************************
 *****************************************************************************************************/

static int  hf              (char *);
static void insert_into_ht  (int, UShort);

/*****************************************************************************************************
 ***** Hash Table Code *******************************************************************************
 *****************************************************************************************************/

int hf (char * one_word)
{
  int k = 0;
  int exponent = 0;
  int i, t, j, subk;
  char c;

  t = strlen(one_word);
  for (i = 0 ; i < t ; i++) {
    c = one_word[t - i - 1];
    if (c == '.') continue;
    if (isdigit(c)) subk = c - '0';
    else            subk = c - 'A';
    for (j = 0; j < exponent ; j++) subk *= 10;
    k += subk;
    exponent++;
  }
  return (k % (SIZE_OF_HASH_TABLE-1));
}

void insert_into_ht (int hv, UShort index)
{
  static int collisions = 0;
  entry_t * slot       = NULL;
  entry_t * prev_slot  = NULL;
  if ((slot = ht[hv]) != NULL) {
    while (slot != NULL) {
      prev_slot = slot;
      slot      = slot->next;
    }}
  if ((slot = calloc(1, sizeof(entry_t))) == NULL) handle_error(ERROR_ALLOC_FAIL, "");
  slot->index = index;
  if (prev_slot == NULL) ht[hv]          = slot;
  else                   prev_slot->next = slot;
}

/*************************************************************************************************************
 ***** CrseDesc_build_ht ******************************************************************************************
 *************************************************************************************************************/

void CrseDesc_build_ht (CrseDesc ** cds, UShort count)
{
  UShort i;
  for (i = 0 ; i < SIZE_OF_HASH_TABLE ; i++) ht[i] = NULL;
  for (i = 0 ; i < count ; i++)
    insert_into_ht (hf(cds[i]->number->members[0]->number), i);
}

/*************************************************************************************************************
 ***** CrseDesc_sort **********************************************************************************************
 *************************************************************************************************************/

void CrseDesc_sort (CrseDesc ** cds, UShort count)
{
  qsort ((void *) cds, count, sizeof(CrseDesc *), (int (*) (const void *, const void *)) CrseDesc_cmp);
}

void CrseDesc_sort_on_gir (CrseDesc ** cds, UShort count)
{
  char i, j;
  CrseDesc * key;
  if (count < 2) return;
  for (j = 1 ; j < count ; j++) {
    key = cds[j];
    for (i = j - 1 ; i >= 0 ; i--) {
      if (cds[i]->GIR_fulfillment_total <= key->GIR_fulfillment_total) break;
      else cds[i+1] = cds[i];
    }
    cds[i+1] = key;
  }
}

void CrseDesc_sort_on_tag (CrseDesc ** cds, UShort count)
{
  char i, j;
  CrseDesc * key;
  if (count < 2) return;
  for (j = 1 ; j < count ; j++) {
    key = cds[j];
    for (i = j - 1 ; i >= 0 ; i--) {
      if (cds[i]->tag_total <= key->tag_total) break;
      else cds[i+1] = cds[i];
    }
    cds[i+1] = key;
  }
}

/*
  qsort ((void *) cds, count, sizeof(CrseDesc *), (int (*) (const void *, const void *)) CrseDesc_cmp_gir);
*/

/*************************************************************************************************************
 ***** CrseDesc_cmp ***********************************************************************************************
 *************************************************************************************************************/
```

```
int CrseDesc_cmp_gir (CrseDesc ** cd1, CrseDesc ** cd2)
{
  if ((*cd1)->GIR_fulfillment != (*cd2)->GIR_fulfillment)
    return ((*cd1)->GIR_fulfillment - (*cd2)->GIR_fulfillment);
  else return ((*cd1)->GIR_fulfillment_qualifier - (*cd2)->GIR_fulfillment_qualifier);
}

int CrseDesc_cmp (CrseDesc ** cd1, CrseDesc ** cd2)
{
  /*****
   ***** Requires : - Both (*cd1)->number and (*cd2)->number are non-NULL CrseNumberSet instances.
   ***** Modifies : - none
   ***** Effects  : - Compares the string representations s1 and s2 of the course numbers of cd1 and cd2.
   *****            - Returns < 0 if s1 < s2
   *****                    = 0 if s1 = s2
   *****                    > 0 if s1 > s2
   *****/

  return (strcmp(
          ((((*cd1)->number)->members)[0])->number,
          ((((*cd2)->number)->members)[0])->number
          ));
}

/*******************************************************************************************************************
 ***** CrseDesc_search *********************************************************************************************
 *******************************************************************************************************************/

CrseDesc * CrseDesc_search (char * key, CrseDesc ** cds, UShort count)
{
  entry_t * slot = NULL;
  slot = ht[hf(key)];
  while (TRUE) {
    if (slot == NULL) return (NULL);
    if (!strcmp(cds[slot->index]->number->members[0]->number, key)) break;
    slot = slot->next;
  }
  return (cds[slot->index]);
}

/*******************************************************************************************************************
 ***** CrseDesc_get_index ******************************************************************************************
 *******************************************************************************************************************/

int CrseDesc_get_index (char * key, CrseDesc ** cds, UShort count)
{
  entry_t * slot = NULL;
  slot = ht[hf(key)];
  while ((slot != NULL) && strcmp(cds[slot->index]->number->members[0]->number, key)) slot = slot->next;
  if (slot == NULL) return (-1);
  return (slot->index);
}

/*******************************************************************************************************************
 ***** CrseDesc_determine_inv_prereqs ******************************************************************************
 *******************************************************************************************************************/

void CrseDesc_determine_inv_prereqs (Essence * ess)
{
  CrseDesc ** cds   = NULL;
  Stats    *  st    = NULL;
  char     *  c     = NULL;
  int         index = 0;

  int i, j;

  cds = ess->cds;
  st  = ess->st;

  for (i = 0 ; i < st->listed_count ; i++) {
    if (cds[i]->prerequisites != NULL) {
      for (j = 0 ; j < cds[i]->prerequisites->count ; j++) {
        c = cds[i]->prerequisites->members[j]->number;
        if (!strcmp(c, POI_CODE) || !strcmp(c, EQUIV_CODE)) continue;
        index = CrseDesc_get_index (c, cds, st->listed_count);
        if (index < 0) {
          printf("CrseDesc_determine_inv_prereqs : Could not find the course : %s\n", c);
          continue;
        }
        cds[index]->inv_prereqs++;
      }}}
}


/******************************************************************************************************************
 ***** CrseDesc_parse ********************************************************************************************
 ******************************************************************************************************************/

int CrseDesc_parse (char ** cd_filenames, VeryShortNum cd_files_count, CrseDesc *** cd, UShort * cd_count, GIR * gir)
{
  /*****
   ***** Requires : - cd_filenames is an array of cd_files_count filenames, naming files which contain
   *****              course descriptions in the Course Bulletin style.
   ***** Modifies : - cd, cd_count
   ***** Effects  : - Parses the files named by cd_filenames into an array of CrseDesc pointers pointed to by *cd.
   *****            - Sets *cd_count to the number of course descriptions parsed.
   *****            - Returns any errors except ERROR_BAD_CRSE_DESC_FILE.  For that error, exits via handle_error.
   ***** Notes    : - Keep in mind the following abbreviation used in this file: AAGM = At A Given Moment.
   *****            - All calls to Stream_is_buffer_empty are made to make sure that the "optimization" of calling
   *****              fscanf instead of Stream_get does not destroy the integrity of the buffer.
   *****            - Note that strtok has surprising side effects.
   *****/

  /****************************************************************************************************************
   ***** Local Variables ******************************************************************************************
   ****************************************************************************************************************/

  int      curr_file_idx  = 0;             /* ordinality of file being processed AAGM */
  char     curr_file_name [MLO_FILENAME];  /* the name of that file                   */
  FILE   * curr_file_ptr  = NULL;          /* pointer to that file                    */
  Stream * sp             = NULL;          /* stream whose source is curr_file_ptr    */

  CrseDesc * curr_crse_desc     = NULL;  /* pointer to course description being built AAGM    */
  UShort     curr_crse_desc_idx = 0;     /* ordinality of course description being built AAGM */

  CrseNumberSet * one_cns       = NULL;  /* the course number set being built AAGM */

  char           one_name       [MLO_CRSE_NAME];  /* holds the course name as it's being built   */
  VeryShortNum   term_offered   = 0;              /* holds the term offered as it's being built  */
  VeryShortNum   subject_level  = 0;              /* holds the subject level as it's being built */
  Stream       * null_source_sp = NULL;           /* stream whose source is NULL                 */
  char         * one_string     = NULL;           /* one string from input stream                */
  char           buffer         [MLO_BUFFER];     /* buffer of strings from input stream         */

  char           curr_char = 0;  /* these are used to detect the beginning of the next course description */
  char           last_char = 0;

  int status = 0;  /* status of function calls which return values */
  int i      = 0;  /* lcv                                          */

  /****************************************************************************************************************/
  /***** Function Body ********************************************************************************************/
  /****************************************************************************************************************/

  /***** Initialize. **********************************************************************************************/

  if (dfs[CRSE_DESCS__PARSE_TRACE]) printf("CrseDesc_parse : Initializing.\n");

  if ((*cd = (CrseDesc **) calloc(MNO_CRSES, sizeof(CrseDesc *))) == NULL) return (ERROR_ALLOC_FAIL);
  null_source_sp = Stream_new (NULL);

  /****************************************************************************************************************
   ***** Outer Loop - Iterate through each of the crse desc files. ************************************************
   ****************************************************************************************************************/

  for (curr_file_idx = 0; curr_file_idx < cd_files_count; curr_file_idx++) {

    /***** Open the current file. *****/

    if (dfs[CRSE_DESCS__PARSE_CURR_FILE]) printf("Processing %s...\n",*(cd_filenames + curr_file_idx));

    if ((sprintf(curr_file_name,"%s%s%s",HOMEDIR,CRSE_DESC_DIR,*(cd_filenames + curr_file_idx))) < 0)
      return (ERROR_SPRINTF_FAIL);
    if ((curr_file_ptr = fopen(curr_file_name, "r")) == NULL) handle_error (ERROR_FOPEN_FAIL, curr_file_name);
```

```
sp = Stream_new (curr_file_ptr);

if (dfs[CRSE_DESCS__PARSE_LINE]) printf ("%s\n",LINE_120);

/*******************************************************************************************************************
 ***** Main Loop ****************************************************************************************************
 *******************************************************************************************************************/

while (TRUE) {

  /*****************************************************************************************************************
   ***** Course Number **********************************************************************************************
   *****************************************************************************************************************/

  if (dfs[CRSE_DESCS__PARSE_TRACE]) printf("CrseDesc_parse : Processing Course Number.\n");

  /***** If EOF, we're done w/ this file; otherwise, allocate mem for a CD *****/

  if (CrseNumberSet_parse (sp, &one_cns) == EOF) break;

  if (dfs[CRSE_DESCS__PARSE_VALIDATE_CNS]) CrseNumberSet_validate (one_cns);
  if ((curr_crse_desc = (CrseDesc *) calloc(1, sizeof(CrseDesc))) == NULL) return (ERROR_ALLOC_FAIL);
  curr_crse_desc->number  = one_cns;

  if (dfs[CRSE_DESCS__PARSE_NUMBER])  printf(UNPARSE_LABEL "%s\n","Number", CrseNumberSet_unparse(one_cns,TRUE));

  /*****************************************************************************************************************
   ***** Course Name ************************************************************************************************
   *****************************************************************************************************************/

  if (dfs[CRSE_DESCS__PARSE_TRACE]) printf("CrseDesc_parse : Processing Course Name.\n");

  /***** Read in strings one at a time and put them into buffer until we encounter the delimiter. *****/

  strcpy(one_name,"");
  while (TRUE) {
    if ((one_string = Stream_get (sp)) == NULL) handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
    if (!strcmp(one_string, LINE_DELIMITER)) break;
    else {
      strcat(one_name, one_string);
      free (one_string);
      strcat(one_name, SPACE_STR);
    }}
  free (one_string);

  /***** Insert course name into crse_desc, leaving out final space. *****/

  if ((curr_crse_desc->name = (char *) malloc(sizeof(char) * (strlen(one_name)))) == NULL) return(ERROR_ALLOC_FAIL);
  one_name[strlen(one_name) - 1] = '\0';
  strcpy(curr_crse_desc->name, one_name);

  if (dfs[CRSE_DESCS__PARSE_NAME]) printf(UNPARSE_LABEL "%s\n", "Name", curr_crse_desc->name);

  /*****************************************************************************************************************
   ***** Former Number, Same Subject As, Meets With *****************************************************************
   *****************************************************************************************************************/

  if (dfs[CRSE_DESCS__PARSE_TRACE])
    printf("CrseDesc_parse : Processing Former Number, Same Subject As, and Meets With.\n");

  if (!Stream_is_buffer_empty(sp)) return (ERROR_BUFFER_NOT_EMPTY);

  while (TRUE) {

    if ((one_string = Stream_get (sp)) == NULL) handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);

    if (!strcmp(one_string, PREREQ_STR)) break;
    if (!strcmp(one_string, SAME_SUBJ_AS_INIT_STR)) {
      if (dfs[CRSE_DESCS__PARSE_TRACE]) printf("CrseDesc_parse : Processing Same Subject As.\n");
      if (fscanf(sp->source, "%*s%*s" SKIP_SPACE_UP_TO_RIGHT_PAREN, buffer) == EOF)    /***** "subject as" *****/
        handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
      if (status = Stream_put_strings(null_source_sp, buffer)) return (status);
      if (CrseNumberSet_parse(null_source_sp, &one_cns) != EOF)
        handle_error (ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
      if (dfs[CRSE_DESCS__PARSE_VALIDATE_CNS]) CrseNumberSet_validate (one_cns);
      curr_crse_desc->same_subj_as = one_cns;
    }
    else if (!strcmp(one_string, MEETS_WITH_INIT_STR)) {
      if (dfs[CRSE_DESCS__PARSE_TRACE]) printf("CrseDesc_parse : Processing Meets With.\n");
      if (fscanf(sp->source, "%*s%*s" SKIP_SPACE_UP_TO_RIGHT_PAREN, buffer) == EOF)    /***** "meets with" *****/
        handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
      if (status = Stream_put_strings(null_source_sp, buffer)) return (status);
      if (CrseNumberSet_parse(null_source_sp, &one_cns) != EOF)
        handle_error (ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
      if (dfs[CRSE_DESCS__PARSE_VALIDATE_CNS]) CrseNumberSet_validate (one_cns);
      curr_crse_desc->meets_with = one_cns;
    }
    else if (*one_string == '(') {  /***** Note that this case must come last. *****/
      if (dfs[CRSE_DESCS__PARSE_TRACE]) printf("CrseDesc_parse : Processing Former Number.\n");
      strcpy (buffer, (one_string + 1));
      while (strchr(one_string, ')') == NULL) {
        if ((one_string = Stream_get(sp)) == NULL) handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
        strcat (buffer, " ");
        strcat (buffer, one_string);
        free (one_string);
      }
      strtok(buffer, ")");  /***** This replaces the right paren with a '\0' *****/
      if (status = Stream_put_strings(null_source_sp, buffer)) return (status);
      if (CrseNumberSet_parse(null_source_sp, &one_cns) != EOF)
        handle_error (ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
      if (dfs[CRSE_DESCS__PARSE_VALIDATE_CNS]) CrseNumberSet_validate (one_cns);
      curr_crse_desc->former_number = one_cns;
    }
    else if (*one_string == ')') {
      free (one_string);
      continue;
    }
    else handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
  }

  if (dfs[CRSE_DESCS__PARSE_FORMER_NUMBER])
    printf(UNPARSE_LABEL "%s\n", "Former Number", CrseNumberSet_unparse (curr_crse_desc->former_number, TRUE));
  if (dfs[CRSE_DESCS__PARSE_SAME_SUBJ_AS])
    printf(UNPARSE_LABEL "%s\n", "Same Subject As", CrseNumberSet_unparse (curr_crse_desc->same_subj_as, TRUE));
  if (dfs[CRSE_DESCS__PARSE_MEETS_WITH])
    printf(UNPARSE_LABEL "%s\n", "Meets With", CrseNumberSet_unparse (curr_crse_desc->meets_with, TRUE));

  /**************************************************************************************************/
  /***** Prereqs ************************************************************************************/
  /**************************************************************************************************/

  if (CrseNumberSet_parse (sp, &one_cns) == EOF) handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
  if (dfs[CRSE_DESCS__PARSE_VALIDATE_CNS]) CrseNumberSet_validate (one_cns);
  curr_crse_desc->prerequisites = one_cns;
  if (dfs[CRSE_DESCS__PARSE_PREREQUISITES])
    printf(UNPARSE_LABEL "%s\n","Prerequisites", CrseNumberSet_unparse(one_cns, TRUE));

  /**************************************************************************************************/
  /***** Term Offered and Subject Level *************************************************************/
  /**************************************************************************************************/

  term_offered = TERM_OFFERED_NONE;

  /***** Skip through verbal junk. *****/

  do {
    if (*one_string == 'U') if (!strcmp(one_string, "U"))    break;
    if (*one_string == 'G') if (!strcmp(one_string, "G"))    break;
    if (*one_string == 'A') if (!strcmp(one_string, "Acad")) break;
    free (one_string);
  }
  while ((one_string = Stream_get (sp)) != NULL);
  if (one_string == NULL) handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);

  if (!Stream_is_buffer_empty(sp)) return (ERROR_BUFFER_NOT_EMPTY);

  if (strchr(one_string,'U') != NULL) {
    subject_level = SUBJECT_LEVEL_U;
    if ((fscanf(sp->source, UP_TO_NEWLINE, buffer)) == EOF)
      handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
    term_offered = parse_term_offered(buffer,term_offered, BOTH_YEARS, &subject_level);
  } else {
    if (strchr(one_string,'G') != NULL) {
      subject_level = SUBJECT_LEVEL_G;
    if (fscanf(sp->source, UP_TO_NEWLINE, buffer) == EOF)
      handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
    term_offered = parse_term_offered(buffer,term_offered, BOTH_YEARS, &subject_level);
  } else {
    if (fscanf(sp->source, "%*s%*s" UP_TO_NEWLINE, buffer) == EOF)           /***** "Year 1994-95:" *****/
      handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
    term_offered = parse_term_offered(buffer, term_offered, THIS_YEAR, &subject_level);
    if (fscanf(sp->source, "%*s%*s%*s" UP_TO_NEWLINE, buffer) == EOF)        /***** "Acad Year 1995-96:" *****/
      handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
    term_offered = parse_term_offered(buffer, term_offered, NEXT_YEAR, &subject_level);
  }
}

curr_crse_desc->term_offered  = term_offered;
curr_crse_desc->subject_level = subject_level;

if (dfs[CRSE_DESCS__PARSE_SUBJECT_LEVEL])
  printf(UNPARSE_LABEL "%s\n", "Subject Level", unparse_subj_level(curr_crse_desc, TRUE));
if (dfs[CRSE_DESCS__PARSE_TERM_OFFERED])
  printf(UNPARSE_LABEL "%s\n", "Term Offered", unparse_term_offered(curr_crse_desc, TRUE));

/************************************************************************************************************/
/***** Units ************************************************************************************************/
/************************************************************************************************************/

if (fscanf(sp->source, "%s", buffer) == EOF) handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);

if (!strcmp(buffer,"Units")) {
  curr_crse_desc->units_arranged = TRUE;
  if (fscanf(sp->source, "%*s") == EOF)                          /***** "arranged" *****/
    handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
} else {
  curr_crse_desc->class_hours = atoi(strtok(buffer, "-"));
  curr_crse_desc->lab_hours   = atoi(strtok(NULL,   "-"));
  curr_crse_desc->prep_hours  = atoi(strtok(NULL,   " "));
  curr_crse_desc->total_units =
    curr_crse_desc->class_hours + curr_crse_desc->lab_hours + curr_crse_desc->prep_hours;
}

if (dfs[CRSE_DESCS__PARSE_UNITS])
  printf(UNPARSE_LABEL "%s\n", "Units", unparse_units(curr_crse_desc, TRUE));

/************************************************************************************************************/
/***** GIR Fulfillment, Pass-Fail, H-Level Grad Credit *****************************************************/
/************************************************************************************************************/

if ((status = fscanf(sp->source, SKIP_SPACE_UP_TO_NEWLINE, buffer)) == EOF)
  handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);

/*****
 ***** In the course catalog, there are:
 *****
 *****      1522 courses with "H-LEVEL Grad Credit"
 *****       288         with "[P/D/F]"
 *****       106         with "[P/D/F]  H-LEVEL Grad Credit"
 *****        17         with one of HLevelExceptStrings
 *****
 *****       435         with "HASS" as the GIR fulfillment
 *****        92         with "HASS-D, Category ?"
 *****        52         with "REST"
 *****        52         with "Institute LAB"
 *****        12         with "HASS-D Language Option"
 *****       < 5 each    with others
 *****
 ***** The code below takes advantage of this distribution.
 *****/

if (status > 0) {
  if      (!strcmp(buffer,H_LEVEL_STRING))                 curr_crse_desc->h_level   = H_LEVEL_GRAD_CREDIT;
  else if (!strcmp(buffer,PASS_FAIL_STRING))               curr_crse_desc->pass_fail = TRUE;
  else if (!strcmp(buffer,PASS_FAIL_AND_H_LEVEL_STRING)) { curr_crse_desc->pass_fail = TRUE;
                                                             curr_crse_desc->h_level   = H_LEVEL_GRAD_CREDIT; }
  else {
    if (strstr(buffer,"except") != NULL) {
      for (i = 2; i < NO_H_LEVEL_EXCEPT_TYPES; i++)  /***** we start at 2, since 0 is NONE, 1 is H_LEVEL *****/
        if (!strcmp(buffer,HLevelExceptStrings[i])) { curr_crse_desc->h_level = i; break; }
    } else
```

```
      if (!strcmp(buffer, "HASS")) {
        curr_crse_desc->GIR_fulfillment_total     = GIRFulfillmentCodes[GIR_FUL_INDEX_HASS];
        curr_crse_desc->GIR_fulfillment           = GIRFulfillmentCodes[GIR_FUL_INDEX_HASS] >> 8;
        curr_crse_desc->GIR_fulfillment_qualifier = GIRFulfillmentCodes[GIR_FUL_INDEX_HASS] % 256;
        CrseSet_insert (gir->categories[GIR_FUL_INDEX_HASS], ((curr_crse_desc->number)->members[0])->number);
      } else
        for (i = 2; i < NO_GIR_TYPES; i++)          /***** we start at 2, since 0 in NONE, 1 is HASS *****/
          if (!strcmp(buffer,GIRFulfillmentStrings[i])) {
      curr_crse_desc->GIR_fulfillment_total     = GIRFulfillmentCodes[i];
      curr_crse_desc->GIR_fulfillment           = GIRFulfillmentCodes[i] >> 8;
      curr_crse_desc->GIR_fulfillment_qualifier = GIRFulfillmentCodes[i] % 256;
      CrseSet_insert (gir->categories[i], ((curr_crse_desc->number)->members[0])->number);
      break;
    }}}

if (dfs[CRSE_DESCS__PARSE_PASS_FAIL])
  printf(UNPARSE_LABEL "%s\n", "Pass/Fail?", unparse_pass_fail(curr_crse_desc, TRUE));
if (dfs[CRSE_DESCS__PARSE_H_LEVEL])
  printf(UNPARSE_LABEL "%s\n", "H-Level", unparse_h_level(curr_crse_desc, TRUE));
if (dfs[CRSE_DESCS__PARSE_GIR_FULFILLMENT])
  printf(UNPARSE_LABEL "%s\n", "GIR Fulfillment", unparse_GIR_fulfillment(curr_crse_desc, TRUE));

/*******************************************************************************************************************/
/***** Repeatable for Credit & No Additional Credit For ************************************************************/
/*******************************************************************************************************************/

if (fscanf(sp->source, "%s", buffer) == EOF) handle_error (ERROR_BAD_CRSE_DESC_FILE, curr_file_name);

if      (!strcmp(buffer,REPEATABLE_FOR_CREDIT_INIT_STRING)) curr_crse_desc->repeatable_for_credit = TRUE;
else if (!strcmp(buffer,NO_ADDITIONAL_CREDIT_FOR_INIT_STRING)) {
  if (fscanf(sp->source, "%*s%*s%*s%*s%*s" UP_TO_NEWLINE, buffer) == EOF) /***** "cannot also be received for" */
    handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
  if (status = Stream_put_strings(null_source_sp, buffer)) return (status);
  if (CrseNumberSet_parse(null_source_sp, &one_cns) != EOF)
    handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
  if (dfs[CRSE_DESCS__PARSE_VALIDATE_CNS]) CrseNumberSet_validate (one_cns);
  curr_crse_desc->no_add_credit_for = one_cns;
}

if (dfs[CRSE_DESCS__PARSE_REPEATABLE_FOR_CREDIT])
  printf(UNPARSE_LABEL "%s\n", "Repeatable for credit?", unparse_repeatable_for_credit(curr_crse_desc, TRUE));
if (dfs[CRSE_DESCS__PARSE_NO_CREDIT_FOR])
  printf(UNPARSE_LABEL "%s\n",
         "No additional credit for", CrseNumberSet_unparse (curr_crse_desc->no_add_credit_for, TRUE));

/***** Save the current course description and increment the count. *****/

*(*cd + curr_crse_desc_idx) = curr_crse_desc;
curr_crse_desc_idx++;
if (dfs[CRSE_DESCS__PARSE_LINE]) printf ("%s\n",LINE_120);

/*******************************************************************************************************************/
/******* Advance to the next course description. *******************************************************************/
/*******************************************************************************************************************/

/***** Look for the line delimiter. *****/

if (strcmp(buffer, LINE_DELIMITER)) {
  do {
    if (!strcmp(one_string, LINE_DELIMITER)) break;
    free (one_string);
  }
  while ((one_string = Stream_get (sp)) != NULL);
}
if (one_string == NULL) handle_error(ERROR_BAD_CRSE_DESC_FILE, curr_file_name);
free (one_string);

/***** Look for two consecutive newlines using the fastest implementation I could think of. *****/
/***** The bitwise-and test outputs FALSE 1% of the time. */

last_char = NEWLINE - 1;

while ((curr_char = getc(sp->source)) != EOF)
  if ((curr_char & last_char) == NEWLINE)
    if (curr_char == NEWLINE)
      if (last_char == NEWLINE) break;
      else last_char = curr_char;
```

```
          else last_char = curr_char;
        else last_char = curr_char;

      if (curr_char == EOF) break;

    } /***** End of Main Loop *****/

    if (status = Stream_close(sp)) return (status);
  }

  if (status = Stream_close (null_source_sp)) return (status);

  if ((*cd = (CrseDesc **) realloc (*cd, curr_crse_desc_idx * sizeof(CrseDesc *))) == NULL)
    return (ERROR_ALLOC_FAIL);

  *cd_count = curr_crse_desc_idx;
  return(ERROR_NONE);
}

/*****************************************************************************************************************
 ***** CrseDesc_unparse ******************************************************************************************
 *****************************************************************************************************************/

char * CrseDesc_unparse (CrseDesc * cd, Boolean should_return_result)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - stdout
   ***** Effects  : - If should_return_result is TRUE, returns a newly allocated string representation of cd.
   *****            - Else, prints a string representation of cd to stdout.  Returns NULL.
   *****            - Exits via handle_error if error is incurred.
   *****/

  char * c      = NULL;
  int    status = ERROR_NONE;

  if (should_return_result) {
    handle_error (ERROR_NOT_YET_IMPLEMENTED, "trying to execute CrseDesc_unparse with should_return_result = TRUE");
  } else {
    status = printf(UNPARSE_LABEL "%s\n", "Number",
             c = CrseNumberSet_unparse         (cd->number,TRUE))             ; free(c);
    status = printf(UNPARSE_LABEL "%s\n", "Name",
             cd->name);
    status = printf(UNPARSE_LABEL "%s\n", "Former Number",
             c = CrseNumberSet_unparse         (cd->former_number, TRUE))     ; free(c);
    status = printf(UNPARSE_LABEL "%s\n", "Same Subject As",
             c = CrseNumberSet_unparse         (cd->same_subj_as, TRUE))      ; free(c);
    status = printf(UNPARSE_LABEL "%s\n", "Meets With",
             c = CrseNumberSet_unparse         (cd->meets_with, TRUE))        ; free(c);
    status = printf(UNPARSE_LABEL "%s\n", "Prerequisites",
             c = CrseNumberSet_unparse         (cd->prerequisites, TRUE))     ; free(c);
    status = printf(UNPARSE_LABEL "%s\n", "Subject Level",
             c = unparse_subj_level            (cd, TRUE))                    ; free(c);
    status = printf(UNPARSE_LABEL "%s\n", "Term Offered",
             c = unparse_term_offered          (cd, TRUE))                    ; free(c);
    status = printf(UNPARSE_LABEL "%s\n", "Units",
             c = unparse_units                 (cd, TRUE))                    ; free(c);
    status = printf(UNPARSE_LABEL "%s\n", "Pass/Fail?",
             c = unparse_pass_fail             (cd, TRUE))                    ; free(c);
    status = printf(UNPARSE_LABEL "%s\n", "H-Level",
             c = unparse_h_level               (cd, TRUE))                    ; free(c);
    status = printf(UNPARSE_LABEL "%s\n", "GIR Fulfillment",
             c = unparse_GIR_fulfillment       (cd, TRUE))                    ; free(c);
    status = printf(UNPARSE_LABEL "%s\n", "Repeatable for credit?",
             c = unparse_repeatable_for_credit (cd, TRUE))                    ; free(c);
    status = printf(UNPARSE_LABEL "%s\n", "No additional credit for",
             c = CrseNumberSet_unparse         (cd->no_add_credit_for, TRUE)); free(c);
    status = printf ("%s\n",LINE_120);
    if (status < 0) handle_error (ERROR_PRINTF_FAIL, "executing CrseDesc_unparse");
    return (NULL);
  }
}

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

II.E. CrseNumber.c

```
/*****************************************************************************************************
 ***** General Info *********************************************************************************
 ****************************************************************************************************
 *****
 ***** File          : CrseNumber.c
 *****
 ***** Author        : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *****                                MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *****
 ***** Background    : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *****                 designed and implemented in the spring of 1995 as the author's Master of Engineering thesis,
 *****                 under the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.
 *****                 See main.c for a more complete description of AAA.
 *****
 *****                 This file contains the source code for the CrseNumber datatype.
 *****
 ***** Compiling Env : gcc -ansi -pedantic -c CrseNumber.c
 *****
 ***** Code Status                   : 1.0.0 WIP
 ***** Last Modification Date & Time : April 27, 1995
 *****
 ****************************************************************************************************
 ***** Overview of Datatype *************************************************************************
 ****************************************************************************************************
 *****
 ***** An instance of this datatype corresponds to a course number in a particular context (i.e., list).
 *****
 ***** This datatype has two components.  The 'number' component is simply the human-readable string
 ***** representation of the course number.  The 'rel_to_next' component indicates the course number's
 ***** relation to the next course number in the context.
 *****
 ***** Suppose that a prequisite list for a course X is (8.01, 18.02).  That list would correspond to two
 ***** CrseNumber instances, say A and B, as follows:
 *****
 *****      A.number      = 8.01
 *****      A.rel_to_next = CRSE_NUMBER_REL_TO_NEXT_AND
 *****
 *****      B.number      = 18.02
 *****      B.rel_to_next = CRSE_NUMBER_REL_TO_NEXT_NONE
 *****
 ****************************************************************************************************
 ***** Operations on Datatype ***********************************************************************
 ****************************************************************************************************
 *****
 ***** CrseNumber * CrseNumber_copy (CrseNumber * cn)
 *****
 ***** Requires : - none
 ***** Modifies : - none
 ***** Effects  : - Returns a copy (newly allocated) of cn.
 *****
 ****************************************************************************************************
 *****
 ***** int CrseNumber_free (CrseNumber * cn)
 *****
 ***** Requires : - none
 ***** Modifies : - cn
 ***** Effects  : - Frees all memory associated with cn, including cn itself.
 *****            - Returns ERROR_NONE.
 *****
 ****************************************************************************************************
 *****
 ***** int CrseNumber_cmp (CrseNumber ** cn1, CrseNumber ** cn2)
 *****
 ***** Requires : - Both *cn1 and *cn2 are non-NULL CrseNumber instances.
 ***** Modifies : - none
 ***** Effects  : - Compares the string representations s1 and s2 of *cn1 and *cn2.
 *****            - Returns < 0 if s1 < s2
 *****                      = 0 if s1 = s2
 *****                      > 0 if s1 > s2
 *****            - The comparison is not just lexicographic.  For example, this function would return >, given
 *****              18.01 and 5.60.
 ***** Bugs     : - For prefixed courses, it does a simple strcmp, instead of comparing the numeric
 *****              suffixes correctly.
 *****
 ****************************************************************************************************
 *****
 ***** Boolean CrseNumber_similar (CrseNumber * cn1, CrseNumber * cn2)
 *****
 ***** Requires : - Both cn1->number and cn2->number are non-NULL strings.
 ***** Modifies : - none
 ***** Effects  : - Returns TRUE if cn1->number and cn2->number are equal, in the eyes of strcmp.
 *****
 ********************************************************************************************************
 *****
 ***** CrseNumber * CrseNumber_parse (char * n, OneNumCode r)
 *****
 ***** Requires : - r is one of CRSE_NUMBER_REL_TO_NEXT_ENUM.
 ***** Modifies : - none
 ***** Effects  : - Creates a new CrseNumber cn.
 *****            - Sets cn.number to n, and cn.rel_to_next to r.
 *****            - Returns cn.
 *****            - Exits via handle_error if any error is incurred.
 *****
 ********************************************************************************************************
 *****
 ***** char * CrseNumber_unparse (CrseNumber * cn, Boolean should_return_result)
 *****
 ***** Requires : - cn is a non-NULL CrseNumber
 ***** Modifies : - stdout
 ***** Effects  : - If should_return_result is TRUE, returns a string representation of cn.
 *****            - Else, prints a string representation of cn to stdout.  Returns NULL.
 *****            - Exits via handle_error if error is incurred.
 *****
 ********************************************************************************************************
 ********************************************************************************************************
 ********************************************************************************************************/

/*******************************************************************************************************
 ***** Header Files *************************************************************************************
 ********************************************************************************************************/

#include <stdio.h>                              /* FILE                 */
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"                          /* MNO_STRINGS_IN_BUFFER */
#include "primitive_datatype_reps_constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "externs.h"                            /* dfs                  */
#include "prototypes.h"                         /* handle_error         */

/*******************************************************************************************************
 ***** CrseNumber_copy **********************************************************************************
 ********************************************************************************************************/

CrseNumber * CrseNumber_copy (CrseNumber * cn)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - none
   ***** Effects  : - Returns a copy (newly allocated) of cn.
   *****/
  if (cn == NULL) return (NULL);
  else            return (CrseNumber_parse (cn->number, cn->rel_to_next));
}

/*******************************************************************************************************
 ***** CrseNumber_free **********************************************************************************
 ********************************************************************************************************/

int CrseNumber_free (CrseNumber * cn)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - cn
   ***** Effects  : - Frees all memory associated with cn, including cn itself.
   *****            - Returns ERROR_NONE.
   *****/

  free(cn->number);
  free(cn);
  return (ERROR_NONE);
}

/*****************************************************************************************************************
 ***** CrseNumber_cmp ***********************************************************************************************
 *****************************************************************************************************************/

int CrseNumber_cmp (CrseNumber ** cn1, CrseNumber ** cn2)
{
  /*****
   ***** Requires : - Both *cn1 and *cn2 are non-NULL CrseNumber instances.
   ***** Modifies : - none
   ***** Effects  : - Compares the string representations s1 and s2 of *cn1 and *cn2.
   *****            - Returns < 0 if s1 < s2
   *****                      = 0 if s1 = s2
   *****                      > 0 if s1 > s2
   *****            - The comparison is not just lexicographic.  For example, this function would return >, given
   *****              18.01 and 5.60.
   ***** Bugs     : - For prefixed courses, it does a simple strcmp, instead of comparing the numeric
   *****              suffixes correctly.
   *****/

  char   str1[MLO_CRSE_NUMBER];
  char   str2[MLO_CRSE_NUMBER];
  char * prefix1 = NULL;
  char * prefix2 = NULL;
  char * suffix1 = NULL;
  char * suffix2 = NULL;

  prefix1 = (*cn1)->number;
  prefix2 = (*cn2)->number;

  if (isalpha(*(prefix1))) {
    if (isalpha(*(prefix2))) return (strcmp(prefix1, prefix2));
    else return (1);
  } else {
    if (isalpha(*(prefix2))) return (-1);
    else {
      strcpy(str1, prefix1);
      strcpy(str2, prefix2);
      prefix1 = strtok (str1, ".AFHLMWUTI");
      suffix1 = strtok (NULL, "-");
      prefix2 = strtok (str2, ".AFHLMWUTI");
      suffix2 = strtok (NULL, "-");
      if (atoi(prefix1) != atoi(prefix2)) {
        if (atoi(prefix1) < atoi(prefix2)) return (-1);
        else return (1);
      } else return (strcmp(suffix1, suffix2));
    }}
}

/*****************************************************************************************************************
 ***** CrseNumber_similar *******************************************************************************************
 *****************************************************************************************************************/

Boolean CrseNumber_similar (CrseNumber * cn1, CrseNumber * cn2)
{
  /*****
   ***** Requires : - Both cn1->number and cn2->number are non-NULL strings.
   ***** Modifies : - none
   ***** Effects  : - Returns TRUE if cn1->number and cn2->number are equal, in the eyes of strcmp.
   *****/

  if (!strcmp(cn1->number, cn2->number)) return (TRUE);
  else                                   return (FALSE);
}

/*****************************************************************************************************************
 ***** CrseNumber_parse *********************************************************************************************
 *****************************************************************************************************************/

CrseNumber * CrseNumber_parse (char * n, OneNumCode r)
{
  /*****
   ***** Requires : - r is one of CRSE_NUMBER_REL_TO_NEXT_ENUM.
   ***** Modifies : - none
   ***** Effects  : - Creates a new CrseNumber cn.
   *****            - Sets cn.number to n, and cn.rel_to_next to r.
   *****            - Returns cn.
   *****            - Exits via handle_error if any error is incurred.
   *****/

  CrseNumber * cn = NULL;

  if (dfs[CRSE_NUMBER__PARSE_ARGS_ENTRY]) printf("CrseNumber_parse : Entering : n = %s\n", n);

  if ((cn = (CrseNumber *) malloc(sizeof(CrseNumber))) == NULL)
    handle_error (ERROR_ALLOC_FAIL, "executing CrseNumber_parse");
  if ((cn->number = (char *) malloc((strlen(n) + 1) * sizeof(char))) == NULL)
    handle_error (ERROR_ALLOC_FAIL, "executing CrseNumber_parse");
  strcpy(cn->number,n);
  cn->rel_to_next = r;

  if (dfs[CRSE_NUMBER__PARSE_ARGS_EXIT])
    printf("CrseNumber_parse : Exiting : cn->number = %s.....cn->rel_to_next = %d\n", cn->number, cn->rel_to_next);

  return (cn);
}

/**********************************************************************************************************
 ***** CrseNumber_unparse *********************************************************************************
 **********************************************************************************************************/

char * CrseNumber_unparse (CrseNumber * cn, Boolean should_return_result)
{
  /*****
   ***** Requires : - cn is a non-NULL CrseNumber
   ***** Modifies : - stdout
   ***** Effects  : - If should_return_result is TRUE, returns a string representation of cn.
   *****              - In this case, the return value *must not* be freed.  If it is, the CrseNumber will be
   *****                destroyed.
   *****            - Else, prints a string representation of cn to stdout.  Returns NULL.
   *****            - Exits via handle_error if error is incurred.
   *****/

  int status = 0;

  if (should_return_result) {
    if      (!strcmp(cn->number,POI_CODE))   return (POI_STRING);
    else if (!strcmp(cn->number,EQUIV_CODE)) return (EQUIV_STRING);
    else                                     return (cn->number);
  }
  else {
    if      (!strcmp(cn->number,POI_CODE))   status = printf(POI_STRING);
    else if (!strcmp(cn->number,EQUIV_CODE)) status = printf(EQUIV_STRING);
    else                                     status = printf("%s", cn->number);
    if (status < 0) handle_error(ERROR_PRINTF_FAIL, "executing CrseNumber_unparse");
    return (NULL);
  }
}

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

II.F. CrseNumberSet.c

```
/**********************************************************************************************************
 ***** General Info ***************************************************************************************
 **********************************************************************************************************
 *****
 ***** File           : CrseNumberSet.c
 *****
 ***** Author         : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *****                                 MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *****
 ***** Background     : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *****                  designed and implemented in the spring of 1995 as the author's Master of Engineering thesis,
*****               under the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.
*****               See main.c for a more complete description of AAA.
*****
*****               This file contains the source code for the CrseNumberSet datatype.
*****
***** Compiling Env : gcc -ansi -pedantic -c CrseNumberSet.c
*****
***** Code Status                  : 1.0.0 WIP
***** Last Modification Date & Time : April 27, 1995
*****
*****************************************************************************************************************
***** Overview of Datatype **************************************************************************************
*****************************************************************************************************************
*****
***** An instance of this datatype corresponds to a set of course numbers.
*****
***** This datatype has two components.  The 'members' components is an array of pointers to CrseNumber instances
***** which comprise the set represented by the instance.  The 'count' component is the cardinality of the set.
*****
***** For example, the set of courses {8.01, 18.02} would correspond to the CrseNumberSet instance cns, where:
*****
*****       cns.count = 2
*****       cns.members[0] = CrseNumber corresponding to 8.01
*****       cns.members[1] = CrseNumber corresponding to 18.02
*****
***** Note that the CrseNumber instances maintain context information.  See CrseNumber.c for details.
*****
*****************************************************************************************************************
***** Operations on Datatype ************************************************************************************
*****************************************************************************************************************
*****
***** CrseNumberSet * CrseNumberSet_copy (CrseNumberSet * cns)
*****
***** Requires : - none
***** Modifies : - none
***** Effects  : - Returns a copy (newly allocated) of cns.
*****
*****************************************************************************************************************
*****
***** int CrseNumberSet_free (CrseNumberSet * cns)
*****
***** Requires : - none
***** Modifies : - cns
***** Effects  : - Frees all memory associated with cns, including cns itself.
*****            - Returns ERROR_NONE.
*****
*****************************************************************************************************************
*****
***** int CrseNumberSet_insert (CrseNumberSet * cns, CrseNumber * cn)
*****
***** Requires : - none
***** Modifies : - cns
***** Effects  : - Inserts cn into cns at the end.
*****            - Increments cns->count
*****            - Returns ERROR_NONE.
*****
*****************************************************************************************************************
*****
***** int CrseNumberSet_expand (CrseNumberSet * cns, CrseNumber * cn)
*****
***** Requires : -
***** Modifies : -
***** Effects  : -
*****
*****************************************************************************************************************
*****
***** Boolean CrseNumberSet_is_member (CrseNumber * cn, CrseNumberSet * cns)
*****
***** Requires : - none
***** Modifies : - none
***** Effects  : - Returns TRUE if one of the CrseNumber instances in cns is same as cn, from the eyes of
*****              CrseNumber_similar.  Return FALSE otherwise.
*****
*****************************************************************************************************************
*****
***** Boolean CrseNumberSet_satisfies (StudPrefs * sp, CrseNumberSet * taken, CrseNumberSet * req,
*****                      CrseNumberSet * satisfied_by)
```

```
 *****
 ***** Requires : -
 ***** Modifies : -
 ***** Effects  : - Returns TRUE if the courses in taken satisfy the courses in req; FALSE otherwise.
 *****            - Also returns the subset of taken which satisfy the courses in req in satisfied_by (newly
 *****              allocated.)
 *****
 *******************************************************************************************************
 *****
 ***** Boolean satisfies_recur (StudPrefs * sp, CrseNumberSet * taken, CrseNumberSet * req,
 *****                          CrseNumberSet * satisfied_by, int index)
 *****
 ***** INTERNAL ONLY
 *****
 *******************************************************************************************************
 *****
 ***** int CrseNumberSet_parse (Stream * sp, CrseNumberSet ** cns)
 *****
 ***** Requires : - none
 ***** Modifies : - sp, cns
 ***** Effects  : - Parses the course number set at the beginning of sp and returns the resulting course
 *****              numbers set in cns.
 *****            - Returns EOF if it is encountered at an acceptable position.
 *****            - Returns ERROR_NONE if no errors occurs.
 *****            - Otherwise, may return ERROR_ALLOC_FAIL or ERROR_BAD_CRSE_DESC_FILE.
 *****            - Note that whatever was in *cns at entry is destroyed.
 *****
 *******************************************************************************************************
 *****
 ***** char * CrseNumberSet_unparse (CrseNumberSet * cns, Boolean should_return_result)
 *****
 ***** Requires : - none
 ***** Modifies : - stdout
 ***** Effects  : - If should_return_result is TRUE, returns a newly-allocated string representation of cns.
 *****            - Else, prints a string representation of cns to stdout.  Returns NULL.
 *****            - Exits via handle_error if error is incurred.
 *****
 *******************************************************************************************************
 *****
 ***** void CrseNumberSet_unparse_X (Env * env, CrseNumberSet * cns, int tab)
 *****
 ***** Requires : -
 ***** Modifies : -
 ***** Effects  : -
 *****
 *******************************************************************************************************
 *****
 ***** int CrseNumberSet_validate (CrseNumberSet * cns)
 *****
 ***** Requires : - none
 ***** Modifies : - stderr
 ***** Effects  : - For each course in cns, checks to see if its number is longer than 7 characters.
 *****            - If one is found, prints an error message and the unparsed course number to stderr, and returns
 *****              ERROR_COURSE_NUMBER_LONGER_THAN_7.
 *****            - Else, returns ERROR_NONE.
 *****
 *******************************************************************************************************
 *******************************************************************************************************
 *******************************************************************************************************/

/*******************************************************************************************************
 ***** Header Files ************************************************************************************
 *******************************************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"                        /* MLO_                                          */
#include "primitive_datatype_reps_constants.h"
#include "datatype_reps_constants.h"          /* CRSE_NUMBER_REL_TO_NEXT_, TERM_OFFERED_       */
#include "table_constants.h"                  /* NO_CRSE_NUMBER_                               */
#include "error_and_trace.h"                  /* error and trace                               */
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"                    /* CrseNumberSet                                 */
#include "prototypes.h"                       /* function prototypes                           */
```

```
#include "externs.h"                          /* CrseNumberSpecProgPrefixes, CrseNumberSuffixes */

/***********************************************************************************************************
 ***** Protypes of Internal Functions ***********************************************************************
 ***********************************************************************************************************/

static Boolean satisfies_recur (StudPrefs *, CrseSet *, CrseNumberSet *, Boolean, CrseNumberSet *, int);

/***********************************************************************************************************
 ***** CrseNumberSet_copy ***********************************************************************************
 ***********************************************************************************************************/

CrseNumberSet * CrseNumberSet_copy (CrseNumberSet * cns)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - none
   ***** Effects  : - Returns a copy (newly allocated) of cns.
   *****/

  CrseNumberSet * retval = NULL;
  int             i      = 0;

  if (cns == NULL) return (NULL);
  retval = (CrseNumberSet *) malloc (sizeof(CrseNumberSet));
  retval->count   = cns->count;
  retval->members = (CrseNumber **) calloc (cns->count, sizeof(CrseNumber *));
  for (i = 0; i < cns->count ; i++) retval->members[i] = CrseNumber_copy(cns->members[i]);

  return (retval);
}

/***********************************************************************************************************
 ***** CrseNumberSet_free ***********************************************************************************
 ***********************************************************************************************************/

int CrseNumberSet_free (CrseNumberSet * cns)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - cns
   ***** Effects  : - Frees all memory associated with cns, including cns itself.
   *****            - Returns ERROR_NONE.
   *****/

  int i = 0;

  if (cns == NULL) return (ERROR_NONE);
  for (i = 0; i < cns->count; i++) CrseNumber_free(cns->members[i]);
  free(cns->members);
  free(cns);
  return (ERROR_NONE);
}

/***********************************************************************************************************
 ***** CrseNumberSet_insert *********************************************************************************
 ***********************************************************************************************************/

int CrseNumberSet_insert (CrseNumberSet * cns, CrseNumber * cn)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - cns
   ***** Effects  : - Inserts cn into cns at the end.
   *****            - Increments cns->count
   *****            - Returns ERROR_NONE.
   *****/

  if (dfs[CRES_NUMBER_SET__INSERT_ARGS_ENTRY])
    printf("CrseNumberSet_insert : Entering : cn->number = %s...cns->count = %d\n",cn->number, cns->count);

  (cns->members)[cns->count++] = cn;

  if (dfs[CRES_NUMBER_SET__INSERT_ARGS_EXIT])
    printf("CrseNumberSet_insert : Exiting : cn->number = %s...cns->count = %d\n",cn->number, cns->count);

  return (ERROR_NONE);
}

/*****************************************************************************************************************
 ***** CrseNumberSet_expand ***************************************************************************************
 *****************************************************************************************************************/

int CrseNumberSet_expand (CrseNumberSet * cns, CrseNumber * cn)
{
  if ((cns->members = (CrseNumber **) realloc (cns->members, (++cns->count) * sizeof(CrseNumber *))) == NULL)
    return (ERROR_ALLOC_FAIL);
  cns->members[cns->count - 1] = cn;
  return (ERROR_NONE);
}

/*****************************************************************************************************************
 ***** CrseNumber_Set_is_member ***********************************************************************************
 *****************************************************************************************************************/

Boolean CrseNumberSet_is_member (CrseNumber * cn, CrseNumberSet * cns)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - none
   ***** Effects  : - Returns TRUE if one of the CrseNumber instances in cns is same as cn, from the eyes of
   *****             CrseNumber_similar.  Return FALSE otherwise.
   *****/

  int     i      = 0;
  Boolean retval = FALSE;

  if (dfs[CRSE_NUMBER_SET__IS_MEMBER_ARGS])
    printf("CrseNumberSet_is_member : Checking membership of %s.\n", cn->number);

  if (cns != NULL)
    for (i = 0; i < cns->count ; i++) if (CrseNumber_similar(cn, cns->members[i])) { retval = TRUE; break; }

  if (dfs[CRSE_NUMBER_SET__IS_MEMBER_ARGS])
    printf("CrseNumberSet_is_member : Returning %s.\n", Boolean_unparse (retval, TRUE));

  return (retval);
}

/*****************************************************************************************************************
 ***** CrseNumber_Set_satisfies ***********************************************************************************
 *****************************************************************************************************************/

Boolean CrseNumberSet_satisfies (StudPrefs * sp, CrseSet * taken, CrseNumberSet * req,
                     Boolean build_satisfied_by, CrseNumberSet ** satisfied_by)
{
  /*****
   ***** Requires : -
   ***** Modifies : -
   ***** Effects  : - Returns TRUE if the courses in taken satisfy the courses in req; FALSE otherwise.
   *****            - Also returns the subset of taken which satisfy the courses in req in satisfied_by (newly
   *****              allocated.)
   *****/

  Boolean         retval = FALSE;

  if (req == NULL) return (TRUE);
  else if (req->count == 0) return (TRUE);

  if (build_satisfied_by) {
    *satisfied_by = (CrseNumberSet *) malloc (sizeof(CrseNumberSet));
    (*satisfied_by)->count   = 0;
    (*satisfied_by)->members = (CrseNumber **) calloc (MNO_CNS_IN_SET, sizeof(CrseNumber *));
    retval = satisfies_recur (sp, taken, req, build_satisfied_by, *satisfied_by, 0);
    if ((*satisfied_by)->count == 0) CrseNumberSet_free (*satisfied_by);
    else (*satisfied_by)->members = (CrseNumber **) realloc ((*satisfied_by)->members,
                                 ((*satisfied_by)->count * sizeof(CrseNumber *)));
    return (retval);
  } else return (satisfies_recur (sp, taken, req, build_satisfied_by, NULL, 0));
}

/*****************************************************************************************************************
 ***** CrseNumber_Set_satisfies_recur *****************************************************************************
 *****************************************************************************************************************/
```

```
Boolean satisfies_recur (StudPrefs * sp, CrseSet * taken, CrseNumberSet * req,
                 Boolean build_satisfied_by, CrseNumberSet * satisfied_by, int index)
{
  CrseNumber * cn  = NULL;
  CrseNumber * cn1 = NULL;

  cn = req->members[index];
  switch (cn->rel_to_next) {

  case CRSE_NUMBER_REL_TO_NEXT_NONE:
    if (dfs[CRSE_NUMBER_SET__SATISFIES_RECUR]) printf("satisfies_recur : None encountered.\n");
    if (!strcmp(cn->number,POI_CODE)) return (sp->override_poi);
    if (CrseSet_is_member(taken, cn->number)) {
      if (build_satisfied_by) CrseNumberSet_insert (satisfied_by, CrseNumber_copy(cn));
      return (TRUE);
    } else return (FALSE);

  case CRSE_NUMBER_REL_TO_NEXT_RANGE:
    if (dfs[CRSE_NUMBER_SET__SATISFIES_RECUR]) printf("satisfies_recur : Range encountered.\n");
    cn1 = req->members[++index];
    switch (cn1->rel_to_next) {
    case CRSE_NUMBER_REL_TO_NEXT_NONE:
      if (CrseSet_is_member(taken, cn->number)) {
        if (build_satisfied_by) CrseNumberSet_insert (satisfied_by, CrseNumber_copy(cn));
        if (CrseSet_is_member(taken, cn1->number)) {
          if (build_satisfied_by) CrseNumberSet_insert (satisfied_by, CrseNumber_copy(cn1));
          return (TRUE);
        } else return (FALSE);
      } else {
        if (CrseSet_is_member(taken, cn1->number))
          if (build_satisfied_by) CrseNumberSet_insert (satisfied_by, CrseNumber_copy(cn1));
        return (FALSE);
      }
    case CRSE_NUMBER_REL_TO_NEXT_RANGE:
      handle_error (ERROR_RANGE_FOLLOWED_BY_RANGE, "executing CrseNumberSet:satisfied_recur");
    case CRSE_NUMBER_REL_TO_NEXT_AND:
      if (CrseSet_is_member(taken, cn->number)) {
        if (build_satisfied_by) CrseNumberSet_insert (satisfied_by, CrseNumber_copy(cn));
        if (CrseSet_is_member(taken, cn1->number)) {
          if (build_satisfied_by) CrseNumberSet_insert (satisfied_by, CrseNumber_copy(cn1));
          return (satisfies_recur (sp, taken, req, build_satisfied_by, satisfied_by, index + 1));
        } else return (FALSE);
      } else {
        if (CrseSet_is_member(taken, cn1->number))
          if (build_satisfied_by) CrseNumberSet_insert (satisfied_by, CrseNumber_copy(cn1));
        return (FALSE);
      }
    case CRSE_NUMBER_REL_TO_NEXT_OR:
      if (CrseSet_is_member(taken, cn->number)) {
        if (build_satisfied_by) CrseNumberSet_insert (satisfied_by, CrseNumber_copy(cn));
        if (CrseSet_is_member(taken, cn1->number)) {
          if (build_satisfied_by) CrseNumberSet_insert (satisfied_by, CrseNumber_copy(cn1));
          while (cn1->rel_to_next == CRSE_NUMBER_REL_TO_NEXT_OR)
            cn1 = req->members[++index];
          switch (cn1->rel_to_next) {
          case CRSE_NUMBER_REL_TO_NEXT_NONE:
            return (TRUE);
          case CRSE_NUMBER_REL_TO_NEXT_RANGE:
            handle_error (ERROR_NOT_YET_IMPLEMENTED, "encountering Range in CrseNumberSet:satisfied_recur");
          case CRSE_NUMBER_REL_TO_NEXT_AND:
            return (satisfies_recur (sp, taken, req, build_satisfied_by, satisfied_by, index + 1));
          }
        } else return (FALSE);
      } else {
        if (CrseSet_is_member(taken, cn1->number))
          if (build_satisfied_by) CrseNumberSet_insert (satisfied_by, CrseNumber_copy(cn1));
        return (FALSE);
      }
    }

  case CRSE_NUMBER_REL_TO_NEXT_AND:
    if (dfs[CRSE_NUMBER_SET__SATISFIES_RECUR]) printf("satisfies_recur : And encountered.\n");
    if (CrseSet_is_member(taken, cn->number)) {
      if (build_satisfied_by) CrseNumberSet_insert (satisfied_by, CrseNumber_copy(cn));
      return (satisfies_recur (sp, taken, req, build_satisfied_by, satisfied_by, index + 1));
    } else return (FALSE);
  case CRSE_NUMBER_REL_TO_NEXT_OR:
    if (dfs[CRSE_NUMBER_SET__SATISFIES_RECUR]) printf("satisfies_recur : Or encountered.\n");
    if (CrseSet_is_member(taken, cn->number)) {
      cn1 = cn;
      while (cn1->rel_to_next == CRSE_NUMBER_REL_TO_NEXT_OR)
        cn1 = req->members[++index];
      switch (cn1->rel_to_next) {
      case CRSE_NUMBER_REL_TO_NEXT_NONE:
        if (build_satisfied_by) CrseNumberSet_insert (satisfied_by, CrseNumber_copy(cn));
        return (TRUE);
      case CRSE_NUMBER_REL_TO_NEXT_RANGE:
        handle_error (ERROR_NOT_YET_IMPLEMENTED, "encountering Range in CrseNumberSet:satisfied_recur");
      case CRSE_NUMBER_REL_TO_NEXT_AND:
        if (build_satisfied_by) CrseNumberSet_insert (satisfied_by, CrseNumber_copy(cn));
        return (satisfies_recur (sp, taken, req, build_satisfied_by, satisfied_by, index + 1));
      }
    }
    else return (satisfies_recur (sp, taken, req, build_satisfied_by, satisfied_by, index + 1));
  }
}

/*******************************************************************************************************************
 ***** CrseNumber_Set_parse *******************************************************************************************
 *******************************************************************************************************************/

int CrseNumberSet_parse (Stream * sp, CrseNumberSet ** cns)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - sp, cns
   ***** Effects  : - Parses the course number set at the beginning of sp and returns the resulting course
   *****              numbers set in cns.
   *****            - Returns EOF if it is encountered at an acceptable position.
   *****            - Returns ERROR_NONE if no errors occurs.
   *****            - Otherwise, may return ERROR_ALLOC_FAIL or ERROR_BAD_CRSE_DESC_FILE.
   *****            - Note that whatever was in *cns at entry is destroyed.
   *****
   ***** Notes    : - This code is rather hairy and full of assumptions and leaps-of-faith.  Thus, it is very heavily
   *****              documented.  I hope it helps.
   *****
   *****            - The parsing is based on the premise that there are the following types of course number sets:
   *****
   *****                   Type #  Type Name                          Typical Instance
   *****                   ------  ---------------------------------  ----------------------------------------------
   *****                   1       Null                               NULL_CN_SET (i.e., --)
   *****                   2       Prefixed Singleton                 MAS 731J
   *****                                                              21M 240
   *****                   3       Prefixed Range                     MAS 961-964
   *****                   4       Suffixed Singleton                 6 UR
   *****                   5       General Case Singleton             8.01
   *****                   6       General Case Range                 6.001-6.002
   *****
   *****            - Note the subtle difference between Type 3 and Type 6.  For Type 3, only the starting point
   *****              has the prefix, which indicates the department.  For Type 6, both starting and ending points
   *****              are fully qualified.  This is not always true in the files put out by the Registrar's.  So,
   *****              some massaging of the input files will be inevitably necessary.
   *****
   *****            - There are two cases whose treatment may be seen as exceptions from the general strategy:
   *****
   *****                   Exception A : We incur this exception when instead of a valid course number, we encounter
   *****                                 OR_LIST_DELIMITER (i.e., "or")
   *****
   *****                   Exception B : We incur this exception when we process the two MIT courses whose names
   *****                                 begin with a number, and therefore defy our parsing strategy.  These
   *****                                 courses are 4.641 ("19th-Century Art") and 4.651 ("20th-Century Art").
   *****
   *****                   Exception C : We incur this exception when instead of a valid course number, we encounter
   *****                                 POI_INIT1 or POI_INIT2 (i.e., "Permission" or "permission").  In this case,
   *****                                 we parse this "course" as POI_CODE.
   *****
   *****                   Exception D : We incur this exception when instead of a valid course number, we encounter
   *****                                 EQUIV1 or EQUIV2 (i.e., "Equivalent" or "equivalent").  In this case,
   *****                                 we parse this "course" as EQUIV_CODE.
   *****
   *****            - This function may be "properly" exited in the following circumstances:
   *****
 *****                   Proper Exit A : At the beginning of the Main Loop, when we are looking for the beginning
 *****                                   of a course number, we encounter an EOF.  In this case, we deduce that
 *****                                   we have reached the end of the input streams and quit.
 *****                                   This case very rarely happens.
 *****
 *****                   Proper Exit B : The course number we are looking at is the NULL_CN_SET (i.e., "--").
 *****                                   Here, we free cns and return ERROR_NONE.
 *****
 *****                   Proper Exit C : We realize that the first character of the string we are looking at is
 *****                                   not a number, and that the string does not fit one of the special cases
 *****                                   in which a course number begins with a letter.  Thus, we deduce that we,
 *****                                   in fact, are not looking at a course number.  So, we push the string in
 *****                                   unused and exit.  This case is by far the most common route of exit
 *****                                   (about 99%).
 *****
 *****                   Proper Exit D : We deduce that the course number we are examining is of Type 4.  Since
 *****                                   we assume that Type 4 course numbers can only occur as singletons
 *****                                   (i.e., not in a or-delimited, comma-delimited, etc. lists), we exit.
 *****                                   This case is rare (about 1%).
 *****
 *****                   Proper Exit E : What we thought was Type 4 turns out to be not.  In this case, we just
 *****                                   push the strings into unused and exit, since what is not Type 4 cannot
 *****                                   be either Type 5 or 6.  This case never actually occurs.
 *****/

/*******************************************************************************************************
 ***** Local Variables *********************************************************************************
 *******************************************************************************************************/

char       * one_str = NULL;               /* a single string read from the stream */
char         cn      [MLO_CRSE_NUMBER];    /* course number being built             */
char         prefix  [MLO_CRSE_NUMBER];    /* prefix for Types 2 & 3                */

char * token  = NULL;  /* used to hold the return value of strtok */
char * strptr = NULL;  /* these two are used to share code... i don't know how else to say it */
int    offset = 0;

int  i       = 0;  /* lcv                          */
int  status  = 0;  /* status of function calls     */

/*******************************************************************************************************
 ***** Function Body ***********************************************************************************
 *******************************************************************************************************/

if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : Entering.\n");

if ((*cns = (CrseNumberSet *) malloc(sizeof(CrseNumberSet))) == NULL) return (ERROR_ALLOC_FAIL);
if (((*cns)->members = (CrseNumber **) calloc(MNO_CNS_IN_SET,sizeof(CrseNumber *))) == NULL) return(ERROR_ALLOC_FAIL);
(*cns)->count = 0;

/*******************************************************************************************************
 ***** Main Loop ***************************************************************************************
 *******************************************************************************************************/

while (TRUE) {

  if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : At the top of Main Loop.\n");
  if ((one_str = Stream_get (sp)) == NULL) {

    /***************************************************************************************************
     * Proper Exit A
     *
     * An EOF at this point indicates that we are at the end of a valid course description file.  So, we
     * wrap up *cns simply return EOF.
     *
     ***************************************************************************************************/

    if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : Exiting via A.\n");
    if ((*cns)->count > 0) {
      if (((*cns)->members = (CrseNumber **) realloc ((*cns)->members, sizeof(CrseNumber *) * (*cns)->count)) == NULL)
        return (ERROR_ALLOC_FAIL);
      else return (EOF);
    } else {
      CrseNumberSet_free (*cns);
      *cns = NULL;
      return (EOF);
    }
```

```
}

if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : Processing the string %s.\n", one_str);

if (!strcmp(one_str, NULL_CN_SET)) {

  /*******************************************************************************************************
   * Type 1 : Null & Proper Exit B
   *
   * In this case, we have zero courses in the set.  So, we free cns (which we allocated above) and
   * return (ERROR_NONE).
   *
   *******************************************************************************************************/

  if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : Type 1 detected.\n");
  CrseNumberSet_free (*cns);
  *cns = NULL;
  free (one_str);
  if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : Exiting through B.\n");
  return (ERROR_NONE);
}

if ((!isdigit(*one_str)) || (!strncmp(one_str, "21", 2))) {  /***** A bit of optimization. *****/

  if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : Entering optimized section.\n");

  /*******************************************************************************************************
   * Detecting Prefixed (Types 2 and 3)
   *
   * To check whether the course number at hand is one of these types, we compare the first string, now contained
   * in one_str, with the elements of CrseNumberSpecProgPrefixes.  If there is a match, then we deduce that we
   * have one of these types.
   *
   *******************************************************************************************************/

  for (i = 0; i < NO_CRSE_NUMBER_SPEC_PROG_PREFIXES; i++)
    if (!strcmp(one_str,CrseNumberSpecProgPrefixes[i])) {

      /***************************************************************************************************
       * Types 2 and 3 : Part I
       *
       * - Put the first string (contained in one_str) into cn.
       * - Get one more string.
       * - If EOF was encountered, then we have an invalid course number in the file, since one of these special
       *   prefixes must be followed by another string.
       * - Determine whether we have Type 2 or Type 3, by looking for a hyphen in the second string.  If we do find
       *   a hyphen, we deduce that we have Type 3.  Otherwise, we deduce that we have Type 2.
       *
       ***************************************************************************************************/

      if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : Types 2 & 3 detected.\n");
      strcpy(cn, one_str);
      free(one_str);
      if ((one_str = Stream_get (sp)) == NULL) return (ERROR_BAD_CRSE_DESC_FILE);
      if (strchr(one_str, HYPHEN) != NULL) {

        /*************************************************************************************************
         * Type 3 : Prefixed Range : Part I
         *
         * - Save the prefix.
         * - Get the starting point by fetching that portion of one_str preceding the hyphen.
         * - Create the complete course number by concatenating the prefix and the fetched starting point.
         * - Create a new CrseNumber with CRSE_NUMBER_REL_TO_NEXT_RANGE as the second
         *   argument, to indicate that the relation of this CrseNumber to the next one is that the pair
         *   comprise a range.
         * - Insert it into cns.
         *
         *************************************************************************************************/

        strcpy(prefix, cn);
        token = strtok(one_str,"-");
        strcat(cn, token);
        CrseNumberSet_insert (*cns, CrseNumber_parse (cn, CRSE_NUMBER_REL_TO_NEXT_RANGE));

        /*************************************************************************************************
         * Type 3 : Prefixed Range : Part II
         *
```

```
     * We now have to create a CrseNumber for the ending point of the range, but we note that one CrseNumber
     * will have to be created for the Type 2 case as well.  So, we capitalize on this by simply setting up
     * the state to share the same code with Type 2.
     *
     * The effect of the statements below is to make the processing below start at the character right past
     * the hyphen, while setting cn to be equal to the prefix saved from above.
     *
     *****************************************************************************************************/

    strcpy(cn, prefix);
    strptr = NULL;
    offset = strlen(token) + 1;
  } else {

    /*****************************************************************************************************
     * Type 2 : Prefixed Singleton
     *
     * If there is no hyphen in one_str, then we simply have a singleton.  So, we set up the state such that
     * the processing below will start at the beginning of one_str.
     *
     *****************************************************************************************************/

    strptr = one_str;
    offset = 0;
  }

  /*******************************************************************************************************
   * Types 2 and 3 : Part II
   *
   * We now create a CrseNumber for the ending point of the range (if we are looking at Type 3)
   *                                    prefixed singleton        (if we are looking at Type 2):
   * - Examine (one_str + offset):
   *   - If there is a comma:
   *     - Fetch that portion between strptr and the comma.
   *     - Create the complete course number by concatenating that portion to cn.
   *     - Create a new CrseNumber with CRSE_NUMBER_REL_TO_NEXT_AND as the second
   *       argument, to indicate that the relation to the next CrseNumber will be an "and" relation.
   *   - Else, if there is a semicolon:
   *     - Do the same as above, except that we fetch the portion between strptr and the semicolon.
   *   - Else, we simply fetch everything after strptr, and again do the same as above, except that
   *     this time, we make a new CrseNumber with CRSE_NUMBER_REL_TO_NEXT_NONE, to indicate that this
   *     CrseNumber does not have any relation with the next one.  In fact, in this case, we expect the
   *     current CrseNumber to be the last to be inserted into the cns.
   * - Proceed to the next iteration of the Main Loop.
   *
   *******************************************************************************************************/

  if (strchr(one_str + offset, COMMA) != NULL) {
    strcat(cn, strtok(strptr, COMMA_STR));
    CrseNumberSet_insert (*cns, CrseNumber_parse (cn, CRSE_NUMBER_REL_TO_NEXT_AND));
  } else {
    if (strchr(one_str + offset, SEMICOLON) != NULL) {
      strcat(cn, strtok(strptr, SEMICOLON_STR));
      CrseNumberSet_insert (*cns, CrseNumber_parse (cn, CRSE_NUMBER_REL_TO_NEXT_AND));
    } else {
      strcat(cn, strtok(strptr, ""));
      CrseNumberSet_insert (*cns, CrseNumber_parse (cn, CRSE_NUMBER_REL_TO_NEXT_NONE));
    }
  }
  free(one_str);
  break;
 }
if (i < NO_CRSE_NUMBER_SPEC_PROG_PREFIXES) continue;

if (!strcmp(one_str, OR_LIST_DELIMITER)) {

  /*********************************************************************************************************
   * Exception A
   *
   * In this case, the rel_to_next of the last CrseNumber we inserted into cns was incorrectly set to
   * CRSE_NUMBER_REL_TO_NEXT_NONE.  So, we fix it up, and we continue with the next iteration of the Main Loop.
   *
   *********************************************************************************************************/

  if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : Exception A incurred.\n");
  free(one_str);
  (((*cns)->members)[(*cns)->count - 1])->rel_to_next = CRSE_NUMBER_REL_TO_NEXT_OR;
```

```
        continue;
      }
    }  /***** End of optimization *****/

    if (
        (!strcmp(one_str,"19th-Century")) ||     /***** 4.641 *****/
        (!strcmp(one_str,"20th-Century"))        /***** 4.651 *****/
        ) {

      /*************************************************************************************************************
       * Exception B
       *
       * In this case, we know we have already finished parsing the course number set, since we are into the course
       * name portion.  Thus, we save one_str in unused and exit the Main Loop.
       *
       *************************************************************************************************************/

      if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : Exception B incurred.\n");
      if (status = Stream_put (sp, one_str)) handle_error (status, "executing CrseNumberSet_parse");
      free(one_str);
      break;
    }

    if (
        (!strcmp(one_str, POI_INIT1)) ||
        (!strcmp(one_str, POI_INIT2))
        ) {

      /*************************************************************************************************************
       * Exception C
       *
       * We parse these as POI_CODE.
       * ??? we assume that POI always comes at the end
       *************************************************************************************************************/

      if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : Exception C incurred.\n");
      free(one_str);
      free(Stream_get (sp));   /***** of          *****/
      free(Stream_get (sp));   /***** instructor *****/
      CrseNumberSet_insert (*cns, CrseNumber_parse (POI_CODE, CRSE_NUMBER_REL_TO_NEXT_NONE));
      break;
    }

    if (
        (!strcmp(one_str, EQUIV1)) ||
        (!strcmp(one_str, EQUIV2))
        ) {

      /*************************************************************************************************************
       * Exception D
       *
       * We parse these as EQUIV_CODE.
       * ??? we assume that EQUIV always comes at the end
       *************************************************************************************************************/

      if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : Exception D incurred.\n");
      free(one_str);
      CrseNumberSet_insert (*cns, CrseNumber_parse (EQUIV_CODE, CRSE_NUMBER_REL_TO_NEXT_NONE));
      break;
    }

    /***************************************************************************************************************
     * Proper Exit C
     *
     * Now, if the first character of one_str is not a digit, then we deduce that we are not looking at a valid
     * course number, and that we therefore must be done parsing the course number set.  Thus, we save one_str
     * in unused, and exit the Main Loop.
     *
     * Before we go, we make sure the rel_to_next of the last one was properly set to CRSE_NUMBER_REL_TO_NEXT_NONE.
     * This is necessary, for example, for 1.125, whose prereqs are "1.124J or Knowledge of C++".  The "or" leads
     * us to believe that there is another course number coming, but there isn't.
     *
     ***************************************************************************************************************/

    if (!isdigit(*one_str)) {
      if (status = Stream_put (sp, one_str))  handle_error (status, "executing CrseNumberSet_parse");
      free(one_str);
    if ((*cns)->count > 0)
      (((*cns)->members)[(*cns)->count - 1])->rel_to_next = CRSE_NUMBER_REL_TO_NEXT_NONE;
    if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : Exiting via C.\n");
    break;
  }

  /*****************************************************************************************************
   * Detecting Suffixed (Type 4) - Part I
   *
   * To check whether the course number at hand is one of these types, we first see if strlen(one_str) is less
   * than or equal to 2.  This is a necessary but not sufficent condition for Type 4.
   *
   *****************************************************************************************************/

  if (strlen(one_str) <= 2) {

    /***************************************************************************************************
     * Detecting Suffixed (Type 4) - Part II
     *
     * Now, we save one_str in cn, and then, to confirm that we have Type 4, we fetch one more string from the
     * streams.  If EOF is returned, we have a bad input file.  Otherwise, we compare that string to the elements
     * of CrseNumberSuffixes.  If there is a match, we deduce that we have Type 4.
     *
     ***************************************************************************************************/

    strcpy(cn, one_str);
    free(one_str);
    if ((one_str = Stream_get (sp)) == NULL) return (ERROR_BAD_CRSE_DESC_FILE);
    for (i = 0; i < NO_CRSE_NUMBER_SUFFIXES; i++)
      if (!strcmp(one_str,CrseNumberSuffixes[i])) break;
    if (i < NO_CRSE_NUMBER_SUFFIXES) {

      /*************************************************************************************************
       * Type 4 & Proper Exit D
       *
       * This case is very simple.  After creating the full course number by concatenating the second string
       * (the suffix) to the first, we create a CrseNumber with CRSE_NUMBER_REL_TO_NEXT_NONE as the second argument.
       * We then exit the Main Loop.
       *
       * We know that the course will have nothing following it, since these Type 4 courses always occur as
       * singletons.
       *
       *************************************************************************************************/

      if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : Type 4 detected.  Exiting via D.\n");
      strcat(cn, one_str);
      free(one_str);
      CrseNumberSet_insert (*cns, CrseNumber_parse (cn, CRSE_NUMBER_REL_TO_NEXT_NONE));
      break;
    } else {

      /*************************************************************************************************
       * Proper Exit E
       *
       * Since it turns out that we were not looking at a Type 4, we save the strings we examined and exit the
       * Main Loop, since we are apparently done.
       *
       * This exit never actually happens.
       *
       *************************************************************************************************/

      if (status = Stream_put (sp, one_str)) handle_error (status, "executing CrseNumberSet_parse");
      free(one_str);
      if (status = Stream_put (sp, cn)) handle_error (status, "executing CrseNumberSet_parse");
      break;
    }
  }

  /*****************************************************************************************************
   * General Case (Types 5 and 6)
   *
   * If we are still in this Main Loop, then we deduce that we have either Type 5 or 6.  Processing here is
   * virtually identical to that in "Types 2 and 3 : Part II" above, except that when there is a range (Type 6),
   * both ends will be fully valid course numbers.
   *
   *****************************************************************************************************/
```

```
    if (dfs[CRSE_NUMBER_SET__PARSE_TRACE]) printf("CrseNumberSet_parse : Type 5 & 6 detected.\n");

    if (strchr(one_str, '-') != NULL) {
      token = strtok(one_str,"-");
      strcpy(cn,token);
      CrseNumberSet_insert (*cns, CrseNumber_parse (cn, CRSE_NUMBER_REL_TO_NEXT_RANGE));
      strptr = NULL;
      offset = strlen(token) + 1;
    } else {
      strptr = one_str;
      offset = 0;
    }
    if (strchr(one_str + offset, COMMA) != NULL) {
      strcpy(cn, strtok(strptr, COMMA_STR));
      CrseNumberSet_insert (*cns, CrseNumber_parse (cn, CRSE_NUMBER_REL_TO_NEXT_AND));
    } else {
      if (strchr(one_str + offset, SEMICOLON) != NULL) {
        strcpy(cn, strtok(strptr, SEMICOLON_STR));
        CrseNumberSet_insert (*cns, CrseNumber_parse (cn, CRSE_NUMBER_REL_TO_NEXT_AND));
      } else {
        strcpy(cn, strtok(strptr, ""));
        CrseNumberSet_insert (*cns, CrseNumber_parse (cn, CRSE_NUMBER_REL_TO_NEXT_NONE));
      }
    }
    free(one_str);
  }

  /*************************************************************************************************************
   ***** End of Main Loop ****************************************************************************************
   *************************************************************************************************************/

  if (((*cns)->members = (CrseNumber **) realloc ((*cns)->members, sizeof(CrseNumber *) * (*cns)->count)) == NULL)
    handle_error(ERROR_ALLOC_FAIL, "executing CrseNumberSet_parse");

  if (dfs[CRSE_NUMBER_SET__PARSE_TRACE])
    printf("CrseNumberSet_parse : Exiting at bottom of function body with (*cns)->count = %d.\n", (*cns)->count);

  return (ERROR_NONE);
}

/***************************************************************************************************************
 ***** CrseNumberSet_unparse ***********************************************************************************
 ***************************************************************************************************************/

char * CrseNumberSet_unparse (CrseNumberSet * cns, Boolean should_return_result)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - stdout
   ***** Effects  : - If should_return_result is TRUE, returns a newly-allocated string representation of cns.
   *****            - Else, prints a string representation of cns to stdout.  Returns NULL.
   *****            - Exits via handle_error if error is incurred.
   *****/

  char   buffer [MLO_BUFFER];
  char * retval = NULL;
  char * c      = NULL;
  int    i      = 0;
  int    status = 0;

  if (should_return_result) {
    if (cns == NULL) strcpy(buffer, "NULL");
    else {
      if (cns->count == 0) strcpy(buffer, "--");
      else {
        strcpy(buffer,"");
        for (i = 0; i < cns->count ; i++) {
          strcat(buffer, CrseNumber_unparse(cns->members[i], TRUE));
          switch ((cns->members[i])->rel_to_next) {
          case CRSE_NUMBER_REL_TO_NEXT_NONE  : break;
          case CRSE_NUMBER_REL_TO_NEXT_RANGE : strcat(buffer, "-");  break;
          case CRSE_NUMBER_REL_TO_NEXT_AND   : strcat(buffer, "; "); break;
          case CRSE_NUMBER_REL_TO_NEXT_OR    : strcat(buffer, " or "); break;
          }
        }}}
    if ((retval = (char *) malloc (sizeof(char) * (strlen(buffer) + 1))) == NULL)
      handle_error(ERROR_ALLOC_FAIL, "executing CrseNumberSet_unparse");
    strcpy (retval, buffer);
    return (retval);
  } else {
    if (cns == NULL) {
      if (printf("NULL") < 0) handle_error(ERROR_PRINTF_FAIL, "executing CrseNumberSet_unparse");
    } else {
      if (cns->count == 0) {
        if (printf("--") < 0) handle_error(ERROR_PRINTF_FAIL  "executing CrseNumberSet_unparse");
      } else
        for (i = 0; i < cns->count ; i++) {
          c = CrseNumber_unparse(cns->members[i], TRUE);
          switch ((cns->members[i])->rel_to_next) {
          case CRSE_NUMBER_REL_TO_NEXT_NONE  : status = printf("%s",     c); break;
          case CRSE_NUMBER_REL_TO_NEXT_RANGE : status = printf("%s-",    c); break;
          case CRSE_NUMBER_REL_TO_NEXT_AND   : status = printf("%s; ",   c); break;
          case CRSE_NUMBER_REL_TO_NEXT_OR    : status = printf("%s or ", c); break;
          }
          if (status < 0) handle_error(ERROR_PRINTF_FAIL, "executing CrseNumberSet_unparse");
        }
    }
    return (NULL);
  }
}

/*************************************************************************************************************
 ***** CrseNumberSet_unparse_X *******************************************************************************
 *************************************************************************************************************/

void CrseNumberSet_unparse_X (Env * env, CrseNumberSet * cns, int tab)
{
  char * c      = NULL;
  char   buffer [MLO_BUFFER];
  int    status = 0;
  int    i      = 0;

  if (cns == NULL) Display_print_string_at_x_of_cl (env, "NULL", tab);
  else {
    if (cns->count == 0) Display_print_string_at_x_of_cl (env, "--", tab);
    else {
      Display_print_string_at_x_of_cl (env, "", tab);
      for (i = 0; i < cns->count ; i++) {
        c = CrseNumber_unparse(cns->members[i], TRUE);
        switch ((cns->members[i])->rel_to_next) {
        case CRSE_NUMBER_REL_TO_NEXT_NONE  : status = sprintf(buffer, "%s; ",   c); break;
        case CRSE_NUMBER_REL_TO_NEXT_RANGE : status = sprintf(buffer, "%s-",    c); break;
        case CRSE_NUMBER_REL_TO_NEXT_AND   : status = sprintf(buffer, "%s; ",   c); break;
        case CRSE_NUMBER_REL_TO_NEXT_OR    : status = sprintf(buffer, "%s or ", c); break;
        }
        if (status < 0) handle_error(ERROR_SPRINTF_FAIL, "executing CrseNumberSet_unparse");
        if (Display_print_string (env, buffer) == ERROR_OUT_OF_BOUNDS) {
          Display_print_string_and_nl (env, "");
          Display_print_string_at_x_of_cl (env, buffer, tab);
        }}}}
}

/*************************************************************************************************************
 ***** CrseNumberSet_validate ********************************************************************************
 *************************************************************************************************************/

int CrseNumberSet_validate (CrseNumberSet * cns)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - stderr
   ***** Effects  : - For each course in cns, checks to see if its number is longer than 7 characters.
   *****            - If one is found, prints an error message and the unparsed course number to stderr, and returns
   *****              ERROR_COURSE_NUMBER_LONGER_THAN_7.
   *****            - Else, returns ERROR_NONE.
   *****/

  int i = 0;

  if (cns == NULL) return (ERROR_NONE);
  for (i = 0; i < cns->count; i++)
    if (strlen((cns->members[i])->number) > 7) {
      fprintf(stderr, "%-10s: ********** POSSIBLE ERROR **********\n",(cns->members[i])->number);
      return (ERROR_COURSE_NUMBER_LONGER_THAN_7);
    }
  return (ERROR_NONE);
}

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

II.G. CrseSet.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps_constants.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "prototypes.h"

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

CrseSet * CrseSet_create (UShort capacity)
{
  CrseSet * retval = NULL;
  if ((retval = (CrseSet *) calloc (1, sizeof(CrseSet))) == NULL)
    handle_error (ERROR_ALLOC_FAIL, "executing CrseSet_create");
  else {
    retval->members  = (char **) calloc (capacity, sizeof(char *));
    retval->count    = 0;
    retval->capacity = capacity;
    return (retval);
  }
}

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

CrseSet * CrseSet_new (char * strs[])
{
  CrseSet * retval = NULL;
  int       count  = 0;
  int       i      = 0;
  while (strcmp(strs[count], "END")) count++;
  retval = CrseSet_create(count);
  for (i = 0 ; i < count ; i++) retval->members[i] = strs[i];
  retval->count = count;
  return (retval);
}

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

CrseSet * CrseSet_copy (CrseSet * cs)
{
  CrseSet * retval = NULL;
  int       i      = 0;
  if (cs == NULL) return (NULL);
  retval          = CrseSet_create(cs->capacity);
  for (i = 0 ; i < cs->count ; i++) {
    retval->members[i] = (char *) malloc ((strlen(cs->members[i]) + 1) * sizeof (char));
    strcpy (retval->members[i], cs->members[i]);
  }
  retval->count   = cs->count;
  return (retval);
}

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

void CrseSet_resize (CrseSet * cs, UShort new_size)
{
  cs->capacity = new_size;
  cs->members  = (char **) realloc (cs->members, sizeof(char *) * new_size);
}
```

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

void CrseSet_free (CrseSet * cs)
{
  int i = 0;
  if (cs == NULL) return;
  for (i = 0 ; i < cs->capacity ; i++) free (cs->members[i]);
  free (cs->members);
  free (cs);
}

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

void CrseSet_insert (CrseSet * cs, char * crse)
{
  if ((cs->count + 1) > cs->capacity) handle_error (ERROR_CAPACITY_FULL, "executing CrseSet_insert");
  cs->members[cs->count] = (char *) malloc ((strlen(crse) + 1) * sizeof(char));
  strcpy(cs->members[cs->count++], crse);
}

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

Boolean CrseSet_is_member (CrseSet * cs, char * crse)
{
  int i = 0;
  for (i = 0 ; i < cs->count ; i++)
    if (cs->members[i][0] == crse[0])
      if (cs->members[i][1] == crse[1])
        if (cs->members[i][2] == crse[2])
          if (!strcmp(cs->members[i], crse)) return (TRUE);
  return (FALSE);
}

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

int CrseSet_index (CrseSet * cs, char * crse)
{
  int i = 0;
  for (i = 0 ; i < cs->count ; i++)
    if (cs->members[i][0] == crse[0])
      if (cs->members[i][1] == crse[1])
        if (cs->members[i][2] == crse[2])
          if (!strcmp(cs->members[i], crse)) return (i);
  return (-1);
}

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

CrseSet * CrseSet_merge (CrseSet * cs1, CrseSet * cs2)
{
  CrseSet * retval = NULL;
  int       i      = 0;
  if      (cs1 == NULL) return (CrseSet_copy(cs2));
  else if (cs2 == NULL) return (CrseSet_copy(cs1));
  else {
    retval = CrseSet_create(cs1->capacity + cs2->capacity);
    for (i = 0 ; i < cs1->count ; i++) {
      retval->members[i] = (char *) malloc ((strlen(cs1->members[i]) + 1) * sizeof (char));
      strcpy (retval->members[i], cs1->members[i]);
    }
    for (i = 0 ; i < cs2->count ; i++) {
      retval->members[cs1->count + i] = (char *) malloc ((strlen(cs2->members[i]) + 1) * sizeof (char));
      strcpy (retval->members[cs1->count + i], cs2->members[i]);
    }}
  retval->count = cs1->count + cs2->count;
  return (retval);
}

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

void CrseSet_append (CrseSet * cs1, CrseSet * cs2)
{
  int i = 0;
  if (cs2 == NULL) return;
  if (cs1 == NULL) handle_error (ERROR_CANNOT_APPEND_TO_NULL, "executing CrseSet_append");
  if ((cs2->count + cs1->count) > cs1->capacity) handle_error (ERROR_CAPACITY_FULL, "executing CrseSet_append");
  for (i = 0 ; i < cs2->count ; i++) {
    cs1->members[cs1->count + i] = (char *) malloc ((strlen(cs2->members[i]) + 1) * sizeof (char));
    strcpy (cs1->members[cs1->count + i], cs2->members[i]);
  }
  cs1->count += cs2->count;
}

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

CrseSet * CrseSet_subtract (CrseSet * cs1, CrseSet * cs2)
{
  /***** Computes cs1 - cs2 *****/
  CrseSet * retval = NULL;
  int       i      = 0;
  if (cs1 == NULL) return (NULL);
  if (cs2 == NULL) return (CrseSet_copy(cs1));
  retval = CrseSet_create(cs1->capacity);
  for (i = 0 ; i < cs1->count ; i++) {
    if (!CrseSet_is_member(cs2, cs1->members[i])) {
      CrseSet_insert (retval, cs1->members[i]);
    }}
  return (retval);
}

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

CrseSet * CrseSet_remove_similar (CrseSet * cs, CrseDesc ** cds, UShort cds_count)
{
  CrseSet  * retval = NULL;
  char     * c      = NULL;
  CrseDesc * cd     = NULL;
  int i, j;

  retval = CrseSet_create(cs->capacity);
  for (i = 0 ; i < cs->count ; i++) {
    if (!CrseSet_is_member(retval, cs->members[i])) {
      cd = CrseDesc_search (cs->members[i], cds, cds_count);
      if (cd == NULL) handle_error (ERROR_UNIDENTIFIED_CRSE_NUM, "executing CrseSet_remove_similar");
      if (cd->same_subj_as == NULL) {
        CrseSet_insert (retval, cs->members[i]);
        continue;
      }
      for (j = 0 ; j < (cd->same_subj_as)->count ; j++) {
        c = ((cd->same_subj_as)->members[j])->number;
        if (CrseSet_is_member(retval, c)) break;
      }
      if (j == (cd->same_subj_as)->count) CrseSet_insert (retval, cs->members[i]);
    }}
  return (retval);
}

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

CrseSet * CrseSet_get_synonyms (CrseSet * cs, CrseDesc ** cds, UShort cds_count)
{
  char     * c      = NULL;
  CrseDesc * cd     = NULL;
  int i, j;
  CrseSet * retval = NULL;

  retval = CrseSet_create(1000);  /* ??? needs to be resized later. */
  for (i = 0 ; i < cs->count ; i++) {
    cd = CrseDesc_search (cs->members[i], cds, cds_count);
    if (cd == NULL) handle_error (ERROR_UNIDENTIFIED_CRSE_NUM, "executing CrseSet_get_synonyms");
    if (cd->same_subj_as != NULL) {
      for (j = 0 ; j < (cd->same_subj_as)->count ; j++)
        CrseSet_insert (retval, ((cd->same_subj_as)->members[j])->number);
    }}
  CrseSet_resize (retval, retval->count);
  return (retval);
}

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

void CrseSet_augment_with_synonyms (CrseSet * cs, CrseDesc ** cds, UShort cds_count)
{
  CrseSet * temp = NULL;
  temp = CrseSet_get_synonyms (cs, cds, cds_count);
  CrseSet_resize (cs, cs->capacity + temp->capacity);
  CrseSet_append (cs, temp);
  CrseSet_free (temp);
}

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

UShort CrseSet_get_units (CrseSet * cs, CrseDesc ** cds, UShort cds_count, Boolean * encountered_units_arranged)
{
  CrseDesc * cd = NULL;
  UShort retval = 0;
  int j;
  *encountered_units_arranged = FALSE;
  for (j = 0 ; j < cs->count ; j++) {
    cd = CrseDesc_search(cs->members[j], cds, cds_count);
    if (cd == NULL) handle_error (ERROR_UNIDENTIFIED_CRSE_NUM, "executing CrseSet_get_units");
    if (cd->units_arranged) *encountered_units_arranged = TRUE;
    else retval += cd->total_units;
  }
  return (retval);
}

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

void CrseSet_sort (CrseSet * cs)
{
  qsort ((void *) cs->members, cs->count, sizeof(char *),
        (int (*) (const void *, const void *)) strptr_cmp);
}

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

int CrseSet_parse (FILE ** fp, CrseSet ** result)
{
  Stream        * S      = NULL;
  CrseNumberSet * cns    = NULL;
  int             status = 0;
  int             i      = 0;
  S = Stream_new (*fp);
  if ((status = CrseNumberSet_parse (S, &cns)) != ERROR_NONE) return (status);
  *fp = Stream_end (S);
  *result = CrseSet_create(cns->count);
  for (i = 0 ; i < cns->count ; i++) CrseSet_insert (*result, (cns->members[i])->number);
  return (ERROR_NONE);
}

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

void CrseSet_unparse (CrseSet * cs, Boolean srr)
{
  int i = 0;
  if (srr) handle_error (ERROR_NOT_YET_IMPLEMENTED, "executing CrseSet_unparse with srr = TRUE");
  else {
    if (cs == NULL) printf("NULL");
    else for (i = 0 ; i < cs->count ; i++) printf("%s ", cs->members[i]);
  }
}

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

void CrseSet_unparse_X (Env * env, CrseSet * set, int tab)
{
  int    i      = 0;
  char   buffer [MLO_BUFFER];
  if (set == NULL) Display_psx (env, "NULL", tab);
  else {
    if (set->count == 0) Display_psx (env, "--", tab);
    else {
      Display_psx (env, "", tab);
      for (i = 0; i < set->count ; i++) {
        if (i < (set->count - 1)) sprintf(buffer, "%s; ", set->members[i]);
        else                      sprintf(buffer, "%s", set->members[i]);
        if (Display_ps (env, buffer) == ERROR_OUT_OF_BOUNDS) {
          Display_psn (env, "");
          Display_psx (env, buffer, tab);
        }}}}
```

```
}

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

II.H. Degrees.c

```
/****************************************************************************************************************
 ***** General Info **********************************************************************************************
 ****************************************************************************************************************
 *****
 ***** File          : Degrees.c
 *****
 ***** Author        : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *****                              MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *****
 ***** Background    : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *****                 designed and implemented in the spring of 1995 as the author's Master of Engineering thesis,
 *****                 under the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.
 *****                 See main.c for a more complete description of AAA.
 *****
 *****                 This file contains the source code for the Degrees datatype.
 *****
 ***** Compiling Env : gcc -ansi -pedantic -c Degrees.c
 *****
 ***** Code Status                   : 1.0.0 WIP
 ***** Last Modification Date & Time : April 27, 1995
 *****
 ****************************************************************************************************************
 ***** Overview of Datatype **************************************************************************************
 ****************************************************************************************************************
 *****
 *****
 *****
 ****************************************************************************************************************
 ***** Operations on Datatype ************************************************************************************
 ****************************************************************************************************************
 *****
 ***** int Degrees_free (Degrees * D)
 *****
 ***** Requires : - none
 ***** Modifies : - D
 ***** Effects  : - Frees all memory associated with D, including D itself.
 *****            - Returns ERROR_NONE.
 *****
 ****************************************************************************************************************
 *****
 ***** int Degrees_parse (char * buffer, OneNumCode type, Degrees ** D)
 *****
 ***** Requires : -
 ***** Modifies : - D
 ***** Effects  : -
 *****
 ****************************************************************************************************************
 *****
 ***** char * Degrees_unparse (Degrees * D, Boolean should_return_result)
 *****
 ***** Requires : - D is a non-NULL Degrees instance.
 ***** Modifies : - stdout
 ***** Effects  : - If should_return_result is TRUE, returns a string representation of D.
 *****            - Else, prints a string representation of D to stdout.  Returns NULL.
 *****            - Exits via handle_error if error is incurred.
 *****
 ****************************************************************************************************************
 *****
 ***** void Degrees_unparse_X (Env * env, Degrees * D, AcadProgs * AP, int tab)
 *****
 ***** Requires : -
 ***** Modifies : - D
 ***** Effects  : -
 *****
 ****************************************************************************************************************
 ****************************************************************************************************************
 ****************************************************************************************************************/

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

```
 ***** Header Files ***********************************************************************************************
 *****************************************************************************************************************/

#include <stdio.h>            /* FILE                */
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"                          /* MLO_ONE_STRING */
#include "primitive_datatype_reps_constants.h"
#include "datatype_reps_constants.h"            /* DEGREE_TYPE_   */
#include "table_constants.h"                    /* APF_           */
#include "error_and_trace.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"                      /* Degrees        */
#include "prototypes.h"                         /* handle_error   */
#include "externs.h"                            /* dfs            */

/*****************************************************************************************************************
 ***** Degrees_free ***********************************************************************************************
 *****************************************************************************************************************/

int Degrees_free (Degrees * D)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - D
   ***** Effects  : - Frees all memory associated with D, including D itself.
   *****            - Returns ERROR_NONE.
   *****/

  free (D->members);
  free (D);
  return (ERROR_NONE);
}

/*****************************************************************************************************************
 ***** Degrees_parse **********************************************************************************************
 *****************************************************************************************************************/

int Degrees_parse (char * buffer, AcadProgs * AP, OneNumCode type, Degrees ** D)
{
  /*****
   ***** Requires : -
   ***** Modifies : - buffer, D
   ***** Effects  : -
   *****/

  char * c = NULL;
  VeryShortNum i = 0;
  int cat = 0;

  *D = (Degrees *) malloc (sizeof(Degrees));
  (*D)->type    = type;
  (*D)->members = (VeryShortNum *) calloc (MNO_DEGREES, sizeof(VeryShortNum));
  (*D)->count   = 0;

  switch (type) {
  case DEGREE_TYPE_MAJOR: cat = APF_MAJORS         ; break;
  case DEGREE_TYPE_MINOR: cat = APF_MINORS_SCI_ENG ; break;
  case DEGREE_TYPE_CONC : cat = APF_CONCS          ; break;
    default             : return (ERROR_INVALID_DEGREE_TYPE);
  }

  if ((c = strtok(buffer, ";\0")) != NULL) {
    do {
      while (*c == SPACE) c++;
      for (i = 0 ; i < (AP->categories[cat])->count ; i++)
        if (!strcmp(c, (AP->categories[cat])->members[i])) {
          (*D)->members[((*D)->count)++] = i;
          break;
        }
      if (i == (AP->categories[cat])->count) {
        if (type != DEGREE_TYPE_MINOR) return (ERROR_INVALID_DEGREE_NAME);
        else {
          for (i = 0 ; i < (AP->categories[APF_MINORS_HASS])->count ; i++)
            if (!strcmp(c, (AP->categories[APF_MINORS_HASS])->members[i])) {
```

```
            (*D)->members[((*D)->count)++] = MNO_MINORS_SCI_ENG + i;
            break;
          }
        if (i == (AP->categories[APF_MINORS_HASS])->count) return (ERROR_INVALID_DEGREE_NAME);
      }}
    } while ((c = strtok(NULL, ";\0")) != NULL);
    (*D)->members = (VeryShortNum *) realloc ((*D)->members,((*D)->count * sizeof(VeryShortNum)));
  } else free ((*D)->members);

  return (ERROR_NONE);
}

/**************************************************************************************************************
 ***** Degrees_unparse ****************************************************************************************
 **************************************************************************************************************/

char * Degrees_unparse (Degrees * D, AcadProgs * AP, Boolean should_return_result)
{
  /*****
   ***** Requires : - D is a non-NULL Degrees instance.
   ***** Modifies : - stdout
   ***** Effects  : - If should_return_result is TRUE, returns a string representation of D.
   *****            - Else, prints a string representation of D to stdout.  Returns NULL.
   *****            - Exits via handle_error if error is incurred.
   *****/

  int cat = 0;
  int i   = 0;

  if (should_return_result)
    handle_error (ERROR_NOT_YET_IMPLEMENTED, "executing Degrees_unparse with should_return_result = TRUE");
  else {
    switch (D->type) {
    case DEGREE_TYPE_MAJOR: cat = APF_MAJORS         ; break;
    case DEGREE_TYPE_MINOR: cat = APF_MINORS_SCI_ENG ; break;
    case DEGREE_TYPE_CONC : cat = APF_CONCS          ; break;
      default             : handle_error (ERROR_INVALID_DEGREE_TYPE, "executing Degrees_unparse");
    }

    for (i = 0 ; i < D->count ; i++) {
      if (D->type == DEGREE_TYPE_MINOR) {
        if (D->members[i] < MNO_MINORS_SCI_ENG)
          printf("%s; ", (AP->categories[cat])->members[D->members[i]]);
        else
          printf("%s; ", (AP->categories[APF_MINORS_HASS])->members[D->members[i] - MNO_MINORS_SCI_ENG]);
      } else {
        printf("%s; ", (AP->categories[cat])->members[D->members[i]]);
      }
    }
  }
  return (NULL);
}

/**************************************************************************************************************
 ***** Degrees_unparse_X **************************************************************************************
 **************************************************************************************************************/

void Degrees_unparse_X (Env * env, Degrees * D, AcadProgs * AP, int tab)
{
  int    cat    = 0;
  int    i      = 0;
  char   buffer [MLO_BUFFER];

  if      (D == NULL)      { Display_print_string_at_x_of_cl (env, "NULL", tab); return; }
  else if (D->count == 0) { Display_print_string_at_x_of_cl (env, "--", tab);   return; }
  else Display_print_string_at_x_of_cl (env, "", tab);

  switch (D->type) {
  case DEGREE_TYPE_MAJOR: cat = APF_MAJORS         ; break;
  case DEGREE_TYPE_MINOR: cat = APF_MINORS_SCI_ENG ; break;
  case DEGREE_TYPE_CONC : cat = APF_CONCS          , break;
    default             : handle_error (ERROR_INVALID_DEGREE_TYPE, "executing Degrees_unparse_X");
  }

  for (i = 0 ; i < D->count ; i++) {
    if (D->type == DEGREE_TYPE_MINOR) {
      if (D->members[i] < MNO_MINORS_SCI_ENG)
```

```
        sprintf(buffer, "%s; ", (AP->categories[cat])->members[D->members[i]]);
      else
        sprintf(buffer,"%s; ", (AP->categories[APF_MINORS_HASS])->members[D->members[i] - MNO_MINORS_SCI_ENG]);
    } else {
      sprintf(buffer, "%s; ", (AP->categories[cat])->members[D->members[i]]);
    }
    if (Display_print_string (env, buffer) == ERROR_OUT_OF_BOUNDS) {
      Display_print_string_and_nl (env, "");
      Display_print_string_at_x_of_cl (env, buffer, tab);
    }
  }
}

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

II.I. Display.c

```
/*************************************************************************************************************
 ***** General Info ******************************************************************************************
 *************************************************************************************************************
 *****
 ***** File         : Display.c
 *****
 ***** Author       : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *****                                MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *****
 ***** Background   : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *****                designed and implemented in the spring of 1995 as the author's Master of Engineering thesis,
 *****                under the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.
 *****                See main.c for a more complete description of AAA.
 *****
 *****                This file contains the source code for the Display datatype.
 *****
 ***** Compiling Env : gcc -ansi -pedantic -c Display
 *****
 ***** Code Status                 : 1.0.0 WIP
 ***** Last Modification Date & Time : April 27, 1995
 *****
 *************************************************************************************************************
 ***** Overview of Datatype **********************************************************************************
 *************************************************************************************************************
 *****
 *****
 *****
 *************************************************************************************************************
 ***** Operations on Datatype ********************************************************************************
 *************************************************************************************************************
 *****
 *****
 *************************************************************************************************************
 *************************************************************************************************************
 *************************************************************************************************************/

/*************************************************************************************************************
 ***** Header Files ******************************************************************************************
 *************************************************************************************************************/

#include <stdio.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xatom.h>
#include "constants.h"
#include "primitive_datatype_reps_constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "prototypes.h"

/*************************************************************************************************************
 ***** Protypes of Internal Functions ************************************************************************
 *************************************************************************************************************/
```

```
static int print_string_general (Env *, char *, int, int, Boolean);

/*****************************************************************************************************************
 ***** Display_create ********************************************************************************************
 *****************************************************************************************************************/

void Display_create (Env * env)
{
  XSizeHints   size_hints;
  XEvent       event;

  env->the_display = XOpenDisplay(NULL);

/*
  env->black = WhitePixel (env->the_display, DefaultScreen(env->the_display));
  env->white = BlackPixel (env->the_display, DefaultScreen(env->the_display));
*/

  env->black = BlackPixel (env->the_display, DefaultScreen(env->the_display));
  env->white = WhitePixel (env->the_display, DefaultScreen(env->the_display));

  env->the_window = XCreateSimpleWindow (env->the_display,  /***** This call generates a CreateNotify event.  *****/
                       DefaultRootWindow(env->the_display),
                       0, 0, WINDOW_WIDTH, WINDOW_HEIGHT,
                       WINDOW_BORDER_WIDTH, env->white, env->black);

  size_hints.width       = WINDOW_WIDTH;   /***** these two (width and height) are obsolete              *****/
  size_hints.height      = WINDOW_HEIGHT;  /***** but should be set anyway for compatibility with old wms *****/
  size_hints.base_width  = WINDOW_WIDTH;
  size_hints.base_height = WINDOW_HEIGHT;
  size_hints.min_width   = WINDOW_WIDTH;
  size_hints.min_height  = WINDOW_HEIGHT;
  size_hints.max_width   = WINDOW_WIDTH;
  size_hints.max_height  = WINDOW_HEIGHT;
  size_hints.flags       = USSize | PMinSize | PMaxSize;

  XSetStandardProperties (env->the_display, env->the_window,
                        WINDOW_TITLE, WINDOW_ICON_TITLE, None, /***** no icon pixmap for now *****/
                        NULL, 0,                               /***** argv and argc          *****/
                &size_hints);

  XSelectInput (env->the_display, env->the_window,
              ExposureMask | ButtonPressMask | ButtonReleaseMask);

  XMapRaised (env->the_display, env->the_window);
  XNextEvent (env->the_display, &event);  /***** Process the CreateNotify event to make window appear. *****/
}

/*****************************************************************************************************************
 ***** Display_initialize ****************************************************************************************
 *****************************************************************************************************************/

void Display_initialize (Env * env)
{
  XGCValues       the_GC_values;
  Screen        * scr     = NULL;
  int             scr_num = 0;

  scr_num = DefaultScreen    (env->the_display);
  scr     = ScreenOfDisplay (env->the_display, scr_num);

  the_GC_values.background = env->black;
  env->the_context         = XCreateGC (env->the_display, env->the_window, GCBackground, &the_GC_values);

  env->the_buffer = XCreatePixmap (env->the_display, env->the_window,
                      WINDOW_WIDTH, WINDOW_HEIGHT, DefaultDepthOfScreen (scr));

  XSetForeground (env->the_display, env->the_context, env->black);
  XFillRectangle (env->the_display, env->the_buffer,  env->the_context, 0, 0, WINDOW_WIDTH, WINDOW_HEIGHT);

  XSetForeground (env->the_display, env->the_context, env->white);
  Display_center_string (env, SCREEN_TITLE_FONT, 0, WINDOW_WIDTH, SCREEN_TITLE_Y, SCREEN_TITLE, TRUE, TRUE, 0, NULL);

  XDrawLine (env->the_display, env->the_buffer, env->the_context,
           0, TOP_LINE_Y, WINDOW_WIDTH, TOP_LINE_Y);
  XDrawLine (env->the_display, env->the_buffer, env->the_context,
           0, TOP_LINE_Y - GAP_BETWEEN_TOP_LINES, WINDOW_WIDTH, TOP_LINE_Y - GAP_BETWEEN_TOP_LINES);

  XDrawLine (env->the_display, env->the_buffer, env->the_context,
           BUTTON_AREA_X, TOP_LINE_Y, BUTTON_AREA_X, WINDOW_HEIGHT);
  XDrawLine (env->the_display, env->the_buffer, env->the_context,
           BUTTON_AREA_X + GAP_BETWEEN_SIDE_LINES, TOP_LINE_Y, BUTTON_AREA_X + GAP_BETWEEN_SIDE_LINES, WINDOW_HEIGHT);

  Display_update_init_dim (env, 0, 0, WINDOW_WIDTH, WINDOW_HEIGHT);

  env->font_id              = XLoadFont(env->the_display, TEXT_FONT);
  env->font                 = XLoadQueryFont(env->the_display, TEXT_FONT);
  env->font_ascent          = ((env->font)->max_bounds).ascent;
  env->font_descent         = ((env->font)->max_bounds).descent;
  env->font_height          = env->font_ascent + env->font_descent;
  env->button_font_id       = XLoadFont(env->the_display, BUTTON_FONT);
  env->button_font          = XLoadQueryFont(env->the_display, BUTTON_FONT);
  env->button_font_ascent   = ((env->button_font)->max_bounds).ascent;
  env->button_font_descent  = ((env->button_font)->max_bounds).descent;
  env->button_font_height   = env->button_font_ascent + env->button_font_descent;
  env->text_area_top        = TOP_LINE_Y + TOP_LINE_GAP;
  env->text_area_left       = SIDE_MARGIN;
  env->text_area_right      = BUTTON_AREA_X - SIDE_MARGIN;
  env->text_area_bottom     = WINDOW_HEIGHT - BOTTOM_MARGIN;
  env->posx                 = 0;
  env->posy                 = 0;
}

/*****************************************************************************************************************
 ***** Display_clear_ ********************************************************************************************
 *****************************************************************************************************************/

void Display_clear_screen (Env * env)
{
  XSetForeground (env->the_display, env->the_context, env->black);
  XFillRectangle (env->the_display, env->the_buffer,  env->the_context,
                0, env->text_area_top,
                WINDOW_WIDTH, WINDOW_HEIGHT);
  XSetForeground (env->the_display, env->the_context, env->white);
  Display_update_init_final (env, 0, env->text_area_top, WINDOW_WIDTH, WINDOW_HEIGHT);

  env->posx = 0;
  env->posy = 0;
}

void Display_clear_text_area (Env * env)
{
  XSetForeground (env->the_display, env->the_context, env->black);
  XFillRectangle (env->the_display, env->the_buffer,  env->the_context,
                env->text_area_left, env->text_area_top,
                (env->text_area_right - env->text_area_left + 1),
                (env->text_area_bottom - env->text_area_top + 1));
  XSetForeground (env->the_display, env->the_context, env->white);
  Display_update_init_final (env, env->text_area_left, env->text_area_top, env->text_area_right, env->text_area_bottom);

  env->posx = 0;
  env->posy = 0;
}

/*****************************************************************************************************************
 ***** Display_print_string **************************************************************************************
 *****************************************************************************************************************/

int Display_print_string (Env * env, char * str)
{
  return (print_string_general(env, str, env->posx, env->posy, FALSE));
}

int Display_print_string_and_nl (Env * env, char * str)
{
  return (print_string_general(env, str, env->posx, env->posy, TRUE));
}

int Display_print_string_at_x_of_cl (Env * env, char * str, int x)
{
  return (print_string_general(env, str, x, env->posy, FALSE));
}
```

```
int Display_print_string_and_nl_at_x_of_cl (Env * env, char * str, int x)
{
  return (print_string_general(env, str, x, env->posy, TRUE));
}

int Display_print_string_at_pos (Env * env, char * str, int x, int y)
{
  return (print_string_general(env, str, x, y, FALSE));
}

int Display_print_string_and_nl_at_pos (Env * env, char * str, int x, int y)
{
  return (print_string_general(env, str, x, y, TRUE));
}

/***** Shorter aliases *************************************************************************************/

int Display_ps (Env * env, char * str)
{
  return (print_string_general(env, str, env->posx, env->posy, FALSE));
}

int Display_psn (Env * env, char * str)
{
  return (print_string_general(env, str, env->posx, env->posy, TRUE));
}

int Display_psx (Env * env, char * str, int x)
{
  return (print_string_general(env, str, x, env->posy, FALSE));
}

int Display_psnx (Env * env, char * str, int x)
{
  return (print_string_general(env, str, x, env->posy, TRUE));
}

int Display_pspos (Env * env, char * str, int x, int y)
{
  return (print_string_general(env, str, x, y, FALSE));
}

int Display_psnpos (Env * env, char * str, int x, int y)
{
  return (print_string_general(env, str, x, y, TRUE));
}

/*****************************************************************************************************************
 ***** Display_update_ ********************************************************************************************
 *****************************************************************************************************************/

void Display_update_init_dim (Env * env, int initx, int inity, int width, int height)
{
  XCopyArea(env->the_display, env->the_buffer, env->the_window, env->the_context,
            initx, inity, width, height, initx, inity);
}

void Display_update_init_final (Env * env, int initx, int inity, int finalx, int finaly)
{
  XCopyArea(env->the_display, env->the_buffer, env->the_window, env->the_context,
            initx, inity, (finalx - initx + 1), (finaly - inity + 1), initx, inity);
}

/*****************************************************************************************************************
 ***** Display_center_string **************************************************************************************
 *****************************************************************************************************************/

void Display_center_string (Env * env, char * font_name, int x_begin, int x_width, int y_pos, char * string,
                    Boolean load, Boolean unload, Font prev_font_id, XFontStruct * prev_font)
{
  static Font          font_id;
  static XFontStruct * font;

  if (load) {
    font_id = XLoadFont(env->the_display, font_name);
    font    = XLoadQueryFont(env->the_display, font_name);
  } else if (prev_font_id != 0) {
    font_id = prev_font_id;
    font    = prev_font;
  }

  XSetFont    (env->the_display, env->the_context, font_id);
  XDrawString (env->the_display, env->the_buffer, env->the_context,
               x_begin + (x_width - XTextWidth(font, string, strlen(string))) / 2,
               y_pos + (font->max_bounds).ascent,
               string, strlen(string));

  if (unload) {
    XUnloadFont (env->the_display, font_id);
    XFreeFont (env->the_display, font);
    font_id = 0;
    font    = NULL;
  }
}

/*****************************************************************************************************************
 ***** Internal Functions ****************************************************************************************
 *****************************************************************************************************************/

int print_string_general (Env * env, char * str, int x, int y, Boolean print_newline)
{
  int pixel_y      = 0;
  int width_of_str = 0;

  if (str != NULL) {
    XSetFont (env->the_display, env->the_context, env->font_id);
    width_of_str = XTextWidth(env->font, str, strlen(str));
    pixel_y      = env->text_area_top + y * env->font_height;

    if ((env->text_area_left + x + width_of_str) > env->text_area_right)  return (ERROR_OUT_OF_BOUNDS);
    if ((pixel_y + env->font_height)             > env->text_area_bottom) return (ERROR_OUT_OF_BOUNDS);

    XDrawString (env->the_display, env->the_buffer, env->the_context,
           (env->text_area_left + x), (pixel_y + env->font_ascent), str, strlen(str));
    Display_update_init_final (env, x, pixel_y, (env->text_area_left + x + width_of_str), (pixel_y + env->font_height));
  }

  if (print_newline) {
    env->posx = 0;
    env->posy = y + 1;
  } else {
    env->posx = x + width_of_str;
    env->posy = y;
  }

  return (ERROR_NONE);
}

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

II.J. GIR.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps_constants.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "prototypes.h"

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

void GIR_initialize (GIR * gir)
{
  int i = 0;
  for (i = 0 ; i < NO_GIR_TYPES ; i++) gir->categories[i] = CrseSet_create(MNO_GIR_TYPE_CRSES);
  gir->all_hass    = CrseSet_create(MNO_GIR_TYPE_CRSES);
  gir->all_phaseI  = CrseSet_create(MNO_GIR_TYPE_CRSES);
  gir->all_phaseII = CrseSet_create(MNO_GIR_TYPE_CRSES);

}

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

void GIR_sort (GIR * gir)
{
  int i = 0;
  for (i = 0 ; i < NO_GIR_TYPES ; i++)
    qsort ((void *) (gir->categories[i])->members, (gir->categories[i])->count, sizeof(char *),
         (int (*) (const void *, const void *)) strptr_cmp);
  qsort ((void *) (gir->all_hass)->members, (gir->all_hass)->count, sizeof(char *),
         (int (*) (const void *, const void *)) strptr_cmp);
  qsort ((void *) (gir->all_phaseI)->members, (gir->all_phaseI)->count, sizeof(char *),
         (int (*) (const void *, const void *)) strptr_cmp);
  qsort ((void *) (gir->all_phaseII)->members, (gir->all_phaseII)->count, sizeof(char *),
         (int (*) (const void *, const void *)) strptr_cmp);
}

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

II.K. OneOfSCS.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps_constants.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "prototypes.h"

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

OneOfSCS * OneOfSCS_create (void)
{
  OneOfSCS * retval = NULL;
  if ((retval = (OneOfSCS *) calloc (1, sizeof(OneOfSCS))) == NULL)
    handle_error (ERROR_ALLOC_FAIL, "executing OneOfSCS_create");
  else return (retval);
}

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

void OneOfSCS_free (OneOfSCS * ooscs)
{
  int i = 0;
  if (ooscs == NULL) return;
  for (i = 0 ; i < ooscs->count ; i++) SubCS_free (ooscs->subcss[i]);  /***** ??? should this be freed *****/
  free (ooscs->subcss);
  free (ooscs);
}

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

void OneOfSCS_insert (OneOfSCS * ooscs, SubCS * subcs)
{
  if (ooscs->subcss == NULL) {
    ooscs->subcss    = (SubCS **) malloc (sizeof(SubCS *));
    ooscs->count     = 1;
    ooscs->subcss[0] = subcs;
  } else {
    ooscs->subcss                    = (SubCS **) realloc (ooscs->subcss, (++(ooscs->count) * sizeof(SubCS *)));
    ooscs->subcss[ooscs->count - 1] = subcs;
  }
}
```

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

Boolean OneOfSCS_satisfied (OneOfSCS * ooscs, CrseSet * taken, CrseSet ** applicable, VeryShortNum * index,
                    Boolean print_if_succ, Boolean print_if_fail, Env * env, int tab)
{
  VeryShortNum i = 0;
  CrseSet * cs = NULL;

  *applicable = CrseSet_create(MNO_APPLICABLE);
  for (i = 0; i < ooscs->count ; i++) {
    if (SubCS_satisfied(ooscs->subcss[i], taken, &cs, FALSE, FALSE, NULL, 0)) {
      CrseSet_append(*applicable, cs);
      *index = i;
      if (print_if_succ) {
        Display_psx (env, "Completed by : ", tab);
        CrseSet_unparse_X (env, *applicable, env->posx);
        Display_ps (env, " of ");
        Display_ps (env, (ooscs->subcss[*index])->set_desc);
      }
      return (TRUE);
    } else CrseSet_append (*applicable, cs);
  }
  if (print_if_fail) Display_psx (env, "No progress", tab);
  return (FALSE);
}

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

void OneOfSCS_unparse_X (Env * env, OneOfSCS * oo, int tab)
{
  int i = 0;
  Display_psnx (env, "One of the following :", tab);
  for (i = 0 ; i < oo->count ; i++) {
    SubCS_unparse_X (env, oo->subcss[i], tab + TAB, TRUE);
    Display_psn      (env, "");
  }
}

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

II.L. RankedCrse.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps_constants.h"
#include "datatype_reps_constants.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "prototypes.h"

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

int inv_prereqs_cmp      (RankedCrse *, RankedCrse *);
int crse_cmp             (RankedCrse *, RankedCrse *);
int degree_ff_units_cmp  (RankedCrse *, RankedCrse *);
int degree_ff_count_cmp  (RankedCrse *, RankedCrse *);
int GIR_ff_units_cmp     (RankedCrse *, RankedCrse *);
int GIR_ff_count_cmp     (RankedCrse *, RankedCrse *);

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

void RankedCrse_initialize (RankedCrse ** rc, Essence * ess)
{
  CrseNumberSet * cns       = NULL;
  VeryShortNum    term_tbc  = 0;      /* term To Be Checked */
  int             i         = 0;
  Boolean         satisfied = FALSE;
  RankedCrse_free (*rc);
  initialize_stats (ess->st);

  if ((*rc = (RankedCrse *) calloc (ess->st->listed_count, sizeof(RankedCrse))) == NULL)
    handle_error (ERROR_ALLOC_FAIL, "executing generate_poss");

  if      ((ess->ss->this_term)->season == FALL)   term_tbc = TERM_OFFERED_THIS_SECOND; /* ??? what about IAP & S */
  else if ((ess->ss->this_term)->season == SPRING) term_tbc = TERM_OFFERED_NEXT_FIRST;

  for (i = 0 ; i < ess->st->listed_count; i++) {

    if ((ess->cds[i]->term_offered & term_tbc) == term_tbc) {

      ++(ess->st->offered_count);

      if (ess->cds[i]->subject_level == SUBJECT_LEVEL_U) {
        ++(ess->st->undergrad_count);
        if (ess->sp->consider_undergrad == FALSE) continue;
      } else {
        ++(ess->st->grad_count);
        if (ess->sp->consider_grad == FALSE) continue;
      }

      ++(ess->st->considered_count);

      /***** ??? what if the course can be repeated for credit? *****/

      if (CrseSet_is_member(ess->ss->taken_all, ((ess->cds[i]->number)->members[0])->number)) {
        ++(ess->st->taken_already_count);
        continue;
      }

      if (CrseNumberSet_satisfies(ess->sp, ess->ss->taken_all, ess->cds[i]->prerequisites, FALSE, NULL)) {
        if ((cns = ess->cds[i]->prerequisites) == NULL) ++(ess->st->no_prereqs_count);
        else {
          if (cns->count == 0) ++(ess->st->no_prereqs_count);
          else                 ++(ess->st->prereqs_satisfied_count);
        }
        (*rc)[ess->st->takable_count].crse        = ess->cds[i]->number->members[0]->number;
        (*rc)[ess->st->takable_count].inv_prereqs = ess->cds[i]->inv_prereqs;
        ++(ess->st->takable_count);
      }}}
  *rc = (RankedCrse *) realloc (*rc, ess->st->takable_count * sizeof(RankedCrse));
}

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

void RankedCrse_free (RankedCrse * rc)
{
  free (rc);
}

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

void RankedCrse_determine_ff (RankedCrse * rc, Essence * ess)
{
  CrseSet  *  orig_taken;
  CrseSet  *  orig_taken_all;

  UShort orig_taken_count;
  UShort orig_taken_all_count;
  UShort orig_taken_cap;
  UShort orig_taken_all_cap;

  CrseSet  *  taken;
  CrseSet  *  taken_all;

  CrseDesc    * cd = NULL;
  char        * c  = NULL;
  AuditResult * ar_gir = NULL;
  AuditResult * ar_dp  = NULL;
  AuditResult * orig_gir = NULL;
  AuditResult * orig_dp  = NULL;

  UShort positive_dpffc  = 0;
  UShort positive_girffc = 0;
  int i, j;

  /***** Audit the originals. *****/

  dp_GIR (MSG_AUDIT, FLAG_NONE, ess, &orig_gir);
  dp_VI_P(MSG_AUDIT, FLAG_NONE, ess, &orig_dp);

  /***** Save the state. *****/

  orig_taken     = ess->ss->taken;
  orig_taken_all = ess->ss->taken_all;

  orig_taken_count     = ess->ss->taken->count;
  orig_taken_all_count = ess->ss->taken_all->count;
  orig_taken_cap       = ess->ss->taken->capacity;
  orig_taken_all_cap   = ess->ss->taken_all->capacity;

  /*****
   ***** Now, we fast copy the three relevant fields into local work space.  Note that these fast-made
   ***** copies must be handled carefully, so as not to destroy the integrity of the originals.
   *****/

  taken     = CrseSet_create (orig_taken_count     + 1  );
  taken_all = CrseSet_create (orig_taken_all_count + 100);

  for (j = 0 ; j < orig_taken_count     ; j++) taken->members[j]     = orig_taken->members[j];
  for (j = 0 ; j < orig_taken_all_count ; j++) taken_all->members[j] = orig_taken_all->members[j];

  taken->count = orig_taken_count + 1;

  /***** Make the local work space *the* work space for ss. *****/

  ess->ss->taken     = taken;
  ess->ss->taken_all = taken_all;

  /***** God Help Us! ************************************************************************************/

  RankedCrse_sort (rc, ess->st->takable_count, INV_PREREQS);
  for (i = 0 ; i < 10 ; i++) RankedCrse_unparse (&rc[i]);

  printf("RankedCrse_determine_ff : Beginning at %s.\n", now());

  for (i = 0 ; i < ess->st->takable_count ; i++) {

    /***** Insert rc[i].crse into the local work space. *****/
/*
    memcpy (taken->members    , orig_taken->members    , orig_taken_count     * sizeof(char *));
    memcpy (taken_all->members, orig_taken_all->members, orig_taken_all_count * sizeof(char *));
*/

    taken->members[orig_taken_count] = rc[i].crse;
    cd = CrseDesc_search (rc[i].crse, ess->cds, ess->st->listed_count);
    if (cd == NULL) exit (-1);
    taken_all->members[orig_taken_all_count] = rc[i].crse;

    taken_all->count = orig_taken_all_count + 1;
    if (cd->same_subj_as != NULL) {
      for (j = 0 ; j < (cd->same_subj_as)->count ; j++)
        taken_all->members[taken_all->count + j] = ((cd->same_subj_as)->members[j])->number;
      taken_all->count += (cd->same_subj_as)->count;
    }

    dp_GIR (MSG_AUDIT, FLAG_NONE, ess, &ar_gir);
    dp_VI_P(MSG_AUDIT, FLAG_NONE, ess, &ar_dp);
    rc[i].GIR_ff_count     = ar_gir->applicable->count - orig_gir->applicable->count;
    rc[i].GIR_ff_units     = ar_gir->appl_units        - orig_gir->appl_units;
    rc[i].GIR_ff_cats      = ar_gir->completed->count  - orig_gir->completed->count;
    rc[i].degree_ff_count  = ar_dp->applicable->count  - orig_dp->applicable->count;
    rc[i].degree_ff_units  = ar_dp->appl_units         - orig_dp->appl_units;
    rc[i].degree_ff_cats   = ar_dp->completed->count   - orig_dp->completed->count;

    AuditResult_free (ar_gir);
    AuditResult_free (ar_dp);
  }

  printf("RankedCrse_determine_ff : Finished at %s.\n", now());
  printf("RankedCrse_determine_ff : Courses having non-zero Degree Fulfillment Differential\n");
  RankedCrse_sort (rc, ess->st->takable_count, DEGREE_FF_COUNT);
  while (rc[positive_dpffc++].degree_ff_count > 0);
  positive_dpffc--;
  RankedCrse_sort (rc, positive_dpffc, CRSE);
  for (i = 0 ; i < positive_dpffc ; i++) RankedCrse_unparse(&rc[i]);

  printf("RankedCrse_determine_ff : Courses having non-zero GIR Fulfillment Differential\n");
  RankedCrse_sort (rc, ess->st->takable_count, GIR_FF_COUNT);
  while (rc[positive_girffc++].GIR_ff_count > 0);
  positive_girffc--;
  RankedCrse_sort (rc, positive_girffc, CRSE);
  for (i = 0 ; i < positive_girffc ; i++) RankedCrse_unparse(&rc[i]);

  ess->ss->taken     = orig_taken;
  ess->ss->taken_all = orig_taken_all;

  free (taken->members);
  free (taken_all->members);
  free (taken);
  free (taken_all);
}

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

void RankedCrse_sort (RankedCrse * rc, UShort count, OneNumCode sorting_field)
{
  if (sorting_field == INV_PREREQS)
    qsort ((void *) rc, count, sizeof(RankedCrse), (int (*) (const void *, const void *)) inv_prereqs_cmp);
  if (sorting_field == CRSE)
    qsort ((void *) rc, count, sizeof(RankedCrse), (int (*) (const void *, const void *)) crse_cmp);
  if (sorting_field == DEGREE_FF_UNITS)
    qsort ((void *) rc, count, sizeof(RankedCrse), (int (*) (const void *, const void *)) degree_ff_units_cmp);
  if (sorting_field == DEGREE_FF_COUNT)
    qsort ((void *) rc, count, sizeof(RankedCrse), (int (*) (const void *, const void *)) degree_ff_count_cmp);
  if (sorting_field == GIR_FF_UNITS)
    qsort ((void *) rc, count, sizeof(RankedCrse), (int (*) (const void *, const void *)) GIR_ff_units_cmp);
  if (sorting_field == GIR_FF_COUNT)
    qsort ((void *) rc, count, sizeof(RankedCrse), (int (*) (const void *, const void *)) GIR_ff_count_cmp);
}

int inv_prereqs_cmp     (RankedCrse * rc1, RankedCrse * rc2) { return (rc2->inv_prereqs - rc1->inv_prereqs);}
int degree_ff_units_cmp (RankedCrse * rc1, RankedCrse * rc2) { return (rc2->degree_ff_units - rc1->degree_ff_units); }
int degree_ff_count_cmp (RankedCrse * rc1, RankedCrse * rc2) { return (rc2->degree_ff_count - rc1->degree_ff_count); }
int GIR_ff_units_cmp    (RankedCrse * rc1, RankedCrse * rc2) { return (rc2->GIR_ff_units - rc1->GIR_ff_units); }
int GIR_ff_count_cmp    (RankedCrse * rc1, RankedCrse * rc2) { return (rc2->GIR_ff_count - rc1->GIR_ff_count); }

int crse_cmp (RankedCrse * rc1, RankedCrse * rc2)
{
  char   str1[MLO_CRSE_NUMBER];
  char   str2[MLO_CRSE_NUMBER];
  char * prefix1 = NULL;
  char * prefix2 = NULL;
  char * suffix1 = NULL;
  char * suffix2 = NULL;

  prefix1 = rc1->crse;
  prefix2 = rc2->crse,

  if (isalpha(*(prefix1))) {
    if (isalpha(*(prefix2))) return (strcmp(prefix1, prefix2));
    else return (1);
  } else {
    if (isalpha(*(prefix2))) return (-1);
    else {
      strcpy(str1, prefix1);
      strcpy(str2, prefix2);
      prefix1 = strtok (str1, ".AFHLMWUTI");
      suffix1 = strtok (NULL, "-");
      prefix2 = strtok (str2, ".AFHLMWUTI");
      suffix2 = strtok (NULL, "-");
      if (atoi(prefix1) != atoi(prefix2)) {
        if (atoi(prefix1) < atoi(prefix2)) return (-1);
        else return (1);
      } else return (strcmp(suffix1, suffix2));
    }}
}
```

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

void RankedCrse_unparse (RankedCrse * rc)
{
  printf("\tcrse = %s\t(%2d, %2d, %2d, %2d, %2d, %2d, %2d, %2d)\n", rc->crse,
         rc->inv_prereqs, rc->off_flex,
         rc->GIR_ff_count, rc->GIR_ff_units, rc->GIR_ff_cats,
         rc->degree_ff_count, rc->degree_ff_units, rc->degree_ff_cats);
/*
  printf("      inv_prereqs     = %d\n", rc->inv_prereqs);
  printf("      off_flex        = %d\n", rc->off_flex);
  printf("      GIR_ff_count    = %d\n", rc->GIR_ff_count);
  printf("      GIR_ff_units    = %d\n", rc->GIR_ff_units);
  printf("      degree_ff_count = %d\n", rc->degree_ff_count);
  printf("      degree_ff_units = %d\n", rc->degree_ff_units);
*/
}

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

II.M. SetOfSQSCS.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps_constants.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "prototypes.h"

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

void SetOfSQSCS_free (SetOfSQSCS * sosqscs)
{
  int i;
  if (sosqscs == NULL) return;
  for (i = 0 ; i < sosqscs->count ; i++) SubQualSCS_free (sosqscs->members[i]);
  free (sosqscs->members);
  free (sosqscs);
}

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

SetOfSQSCS * SetOfSQSCS_create (void)
{
  SetOfSQSCS * retval = NULL;
  if ((retval = (SetOfSQSCS *) calloc (1, sizeof(SetOfSQSCS))) == NULL)
    handle_error (ERROR_ALLOC_FAIL, "executing SetOfSQSCS_create");
  else return (retval);
}

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

void SetOfSQSCS_insert (SetOfSQSCS * sosqscs, SubQualSCS * sqscs)
{
  if (sosqscs->members == NULL) {
    sosqscs->members    = (SubQualSCS **) malloc (sizeof(SubQualSCS *));
    sosqscs->count      = 1;
    sosqscs->members[0] = SubQualSCS_copy (sqscs);
  } else {
    sosqscs->members = (SubQualSCS **) realloc (sosqscs->members, (++(sosqscs->count) * sizeof(SubQualSCS *)));
    sosqscs->members[sosqscs->count - 1] = SubQualSCS_copy (sqscs);
  }
}

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

void SetOfSQSCS_set_excludings (SetOfSQSCS * sosqscs, CrseSet * excl)
{
  int i;
  for (i = 0 ; i < sosqscs->count ; i++) (sosqscs->members[i])->excluding = CrseSet_copy (excl);
}

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

void SetOfSQSCS_free_excludings (SetOfSQSCS * sosqscs)
{
  int i;
  for (i = 0 ; i < sosqscs->count ; i++) CrseSet_free ((sosqscs->members[i])->excluding);
}

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

Boolean SetOfSQSCS_satisfied (SetOfSQSCS * sosqscs, CrseSet * taken, CrseSet ** applicable,
                      Boolean print_if_succ, Boolean print_if_fail, Env * env, int tab)
{
  CrseSet ** local_applicables    = NULL;
  int       * indices             = NULL;
  int count      = 0;
  int saved_posx = 0;
  int i, j;

  local_applicables = (CrseSet **) calloc(sosqscs->count, sizeof(CrseSet *));
  indices           = (int      *) calloc(sosqscs->count, sizeof(int      ));
  *applicable       = CrseSet_create(MNO_APPLICABLE);

  for (i = 0 ; i < sosqscs->count ; i++) {
    if (SubQualSCS_satisfied(sosqscs->members[i], taken, &(local_applicables[i]), FALSE, FALSE, NULL, 0)) {
      CrseSet_append (*applicable, local_applicables[i]);
      indices[count++] = i;
      if (count >= sosqscs->req_num) {
        if (print_if_succ) {
          Display_psx (env, "Completed by : ", tab);
          saved_posx = env->posx;
          for (j = 0 ; j < count ; j++) {
            CrseSet_unparse_X (env, local_applicables[indices[j]], saved_posx);
            Display_psn (env, "");
            Display_psx (env, "of ", saved_posx);
            Display_ps  (env, ((sosqscs->members[indices[j]])->subcs)->set_desc);
            if (j < (count - 1)) Display_psn (env, "");
          }}
        return (TRUE);
      }
    }
  }
  if (print_if_fail) {
    if (count > 0) {
      Display_psx (env, "Completed : ", tab);
      saved_posx = env->posx;
      for (j = 0 ; j < count ; j++) {
        CrseSet_unparse_X (env, local_applicables[indices[j]], saved_posx);
        Display_psn (env, "");
        Display_psx (env, "of ", saved_posx);
        Display_ps  (env, ((sosqscs->members[indices[j]])->subcs)->set_desc);
        if (j < (count - 1)) Display_psn (env, "");
      }}
    else Display_psx (env, "No progress", tab);
  }

  for (i = 0 ; i < sosqscs->count ; i++) CrseSet_free (local_applicables[i]);
  free (local_applicables);
  free (indices);

  return (FALSE);
}

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

void SetOfSQSCS_unparse_X (Env * env, SetOfSQSCS * sosqscs, int tab,
                   Boolean subcs_desc_only, Boolean including_desc_only)
{
  char buffer[MLO_BUFFER];
  int  saved_posx;
  int  i;
  if (sosqscs == NULL) Display_psx (env, "NULL", tab);
  else {
    sprintf(buffer, "%d of ", sosqscs->req_num);
    Display_psx (env, buffer, tab);
    saved_posx = env->posx;
    for (i = 0 ; i < sosqscs->count ; i++) {
      SubQualSCS_unparse_X (env, sosqscs->members[i], saved_posx, subcs_desc_only, including_desc_only);
      Display_psn (env, "");
    }
  }
}

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

II.N. Stream.c

```
/****************************************************************************************************************
 ***** General Info *******************************************************************************************
 ****************************************************************************************************************
 *****
 ***** File          : Stream.c
 *****
 ***** Author        : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *****                                MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *****
 ***** Background    : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *****                 designed and implemented in the spring of 1995 as the author's Master of Engineering thesis,
 *****                 under the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.
 *****                 See main.c for a more complete description of AAA.
 *****
 *****                 This file contains the source code for the Stream datatype.
 *****
 ***** Compiling Env : gcc -ansi -pedantic -c Stream.c
 *****
 ***** Code Status                : 1.0.0 WIP
 ***** Last Modification Date & Time : April 27, 1995
 *****
 ****************************************************************************************************************
 ***** Overview of Datatype ***********************************************************************************
 ****************************************************************************************************************
 *****
 ***** The Stream datatype basically provides a buffered FILE *.
 *****
 ***** The Stream datatype has five components:
 *****
 *****      'source' : The FILE * which is the input source for the stream.
 *****      'buffer' : The buffer, which is an array of strings.
 *****      'head'   : Provides the index to the first filled slot in the buffer.
 *****      'tail'   : Provides the index to the first empty slot in the buffer.
 *****      'count'  : Provides the number of strings currently in the buffer.
 *****
 ***** With a null FILE * as the source, a Stream may be used as a FIFO queue of strings.
 *****
 ****************************************************************************************************************
 ***** Operations on Datatype **********************************************************************************
 ****************************************************************************************************************
 *****
 ***** Stream * Stream_new (FILE * input_file)
 *****
 ***** Requires   - none
 ***** Modifies   - none
 ***** Effects  : - Creates a new stream S and sets S->source to input_file.  Returns S.
 *****            - Exits via handle_error if error is incurred.
 *****
 ****************************************************************************************************************
 *****
 ***** int Stream_close (Stream * S)
 *****
 ***** Requires : - S is a non-NULL Stream.
 ***** Modifies . - S
 ***** Effects  : - Closes the stream S and frees all allocated memory associated with S.
 *****            - Returns any errors.
 *****
 ****************************************************************************************************************
 *****
 ***** FILE * Stream_end (Stream * S)
 *****
```

```
***** Requires : - S is a non-NULL Stream.
***** Modifies : - S
***** Effects  : - Frees the stream S and returns the source of S.  This is just like Stream_close except that the
*****              source is "salvaged."
*****
*************************************************************************************************************************
*****
***** char * Stream_get (Stream * S)
*****
***** Requires : - S is a non-NULL Stream.
***** Modifies : - S
***** Effects  : - Fetches a single string from S and returns it.
*****            - Returns NULL if EOF occurs.
*****            - Exits via handle_error if a different error is incurred.
*****
*************************************************************************************************************************
*****
***** int Stream_put (Stream * S, char * str)
*****
***** Requires : - S is a non-NULL Stream.
***** Modifies : - S
***** Effects  : - Puts str at the tail of S.
*****            - Returns ERROR_BUFFER_FULL if buffer is full.
*****            - Returns ERROR_ALLOC_FAIL if memory allocation fails.
*****            - Returns ERROR_NONE otherwise.
*****
*************************************************************************************************************************
*****
***** int Stream_put_strings (Stream * S, char * strs)
*****
***** Requires : - S is a non-NULL Stream.
*****            - strs is a series of strings delimited by either spaces or newlines.
***** Modifies : - S, strs
***** Effects  : - Puts the strings in strs at the tail of S.
*****            - Returns any errors that may be returned by Stream_put.
*****            - strs is destoryed in the processing.
*****
*************************************************************************************************************************
*****
***** Boolean Stream_is_buffer_empty (Stream * S)
*****
***** Requires : - none
***** Modifies : - none
***** Effects  : - Returns TRUE if the S->buffer is empty, FALSE otherwise.
*****
*************************************************************************************************************************
*****
***** char * Stream_unparse (Stream * S, Boolean should_return_result)
*****
***** Requires : - none
***** Modifies : - stdout
***** Effects  : - If should_return_result is TRUE, returns a newly-allocated string representation of S.
*****            - Else, prints a string representation of S to stdout.  Returns NULL.
*****            - Exits via handle_error if error is incurred.
*****
*************************************************************************************************************************
*************************************************************************************************************************
************************************************************************************************************************/

/************************************************************************************************************************
***** Header Files ******************************************************************************************************
************************************************************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"                          /* MNO_ and MLO_     */
#include "primitive_datatype_reps_constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"                      /* CrseNumber, etc.  */
#include "prototypes.h"                         /* handle_error      */
#include "externs.h"                            /* dfs               */
```

```
/*******************************************************************************************************************
 ***** Stream_new ***************************************************************************************************
 *******************************************************************************************************************/

Stream * Stream_new (FILE * input_file)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - none
   ***** Effects  : - Creates a new stream S and sets S->source to input_file.  Returns S.
   *****            - Exits via handle_error if error is incurred.
   *****/

  Stream * S = NULL;

  if ((S = (Stream *) calloc (1, sizeof(Stream))) == NULL) handle_error (ERROR_ALLOC_FAIL, "executing Stream_new");
  S->source = input_file;
  return (S);
}

/*******************************************************************************************************************
 ***** Stream_close *************************************************************************************************
 *******************************************************************************************************************/

int Stream_close (Stream * S)
{
  /*****
   ***** Requires : - S is a non-NULL Stream.
   ***** Modifies : - S
   ***** Effects  : - Closes the stream S and frees all allocated memory associated with S.
   *****            - Returns any errors.
   *****/

  if (Stream_end(S) != NULL) return (fclose (S->source));
  else                       return (ERROR_NONE);
}

/*******************************************************************************************************************
 ***** Stream_end ***************************************************************************************************
 *******************************************************************************************************************/

FILE * Stream_end (Stream * S)
{
  /*****
   ***** Requires : - S is a non-NULL Stream.
   ***** Modifies : - S
   ***** Effects  : - Frees the stream S and returns the source of S.  This is just like Stream_close except that the
   *****              source is "salvaged."
   *****/

 FILE * fp    = NULL;
 int    i     = 0;

 for (i = 0; i < S->count; i++) free (S->buffer[(S->head + i) % MNO_STRINGS_IN_BUFFER]);
 fp = S->source;
 free (S);
 return (fp);
}

/*******************************************************************************************************************
 ***** Stream_get ***************************************************************************************************
 *******************************************************************************************************************/

char * Stream_get (Stream * S)
{
  /*****
   ***** Requires : - S is a non-NULL Stream.
   ***** Modifies : - S
   ***** Effects  : - Fetches a single string from S and returns it.
   *****            - Returns NULL if EOF occurs.
   *****            - Exits via handle_error if a different error is incurred.
   *****/

  char * retval                = NULL;
  char   temp [MLO_ONE_STRING];

  if (S->count > 0) {
    if ((retval = (char *) malloc ((strlen(S->buffer[S->head]) + 1) * sizeof(char))) == NULL)
      handle_error (ERROR_ALLOC_FAIL, "executing Stream_get");
    strcpy(retval, S->buffer[S->head]);
    free (S->buffer[S->head]);
    S->head = ++(S->head) % MNO_STRINGS_IN_BUFFER;
    (S->count)--;
    if (dfs[STREAM__GET_TRACE]) printf("Stream_get : Returning %s from buffer.\n", retval);
    return (retval);
  } else {
    if (S->source == NULL) return (NULL);
    else {
      if (fscanf(S->source. "%s", temp) == EOF) return (NULL);
      else {
        if ((retval = (char *) malloc ((strlen(temp) + 1) * sizeof(char))) == NULL)
          handle_error (ERROR_ALLOC_FAIL, "executing Stream_get");
        strcpy(retval, temp);
        if (dfs[STREAM__GET_TRACE]) printf("Stream_get : Returning %s from source.\n", retval);
        return (retval);
      }
    }
  }
}

/*****************************************************************************************************************
 ***** Stream_put *************************************************************************************************
 *****************************************************************************************************************/

int Stream_put (Stream * S, char * str)
{
  /*****
   ***** Requires : - S is a non-NULL Stream.
   ***** Modifies : - S
   ***** Effects  : - Puts str at the tail of S.
   *****            - Returns ERROR_BUFFER_FULL if buffer is full.
   *****            - Returns ERROR_ALLOC_FAIL if memory allocation fails.
   *****            - Returns ERROR_NONE otherwise.
   *****/

  char * c = NULL;

  if (dfs[STREAM__PUT_ARGS_ENTRY])
    printf("Stream_put : Entering : str = %s.....S->count = %d.....S->tail = %d\n",str, S->count, S->tail);

  if (++(S->count) > MNO_STRINGS_IN_BUFFER) return (ERROR_BUFFER_FULL);
  else {
    if ((c = (char *) malloc (sizeof(char) * (strlen(str) + 1))) == NULL) return (ERROR_ALLOC_FAIL);
    strcpy(c, str);
    S->buffer[S->tail] = c;
    S->tail = ++(S->tail) % MNO_STRINGS_IN_BUFFER;

    if (dfs[STREAM__PUT_ARGS_EXIT])
      printf("Stream_put : Exiting : str = %s.....S->count = %d.....S->tail = %d\n",str, S->count, S->tail);

    return (ERROR_NONE);
  }
}

/*****************************************************************************************************************
 ***** Stream_put_strings *****************************************************************************************
 *****************************************************************************************************************/

int Stream_put_strings (Stream * S, char * strs)
{
  /*****
   ***** Requires : - S is a non-NULL Stream.
   *****            - strs is a series of strings delimited by either spaces or newlines.
   ***** Modifies : - S, strs
   ***** Effects  : - Puts the strings in strs at the tail of S.
   *****            - Returns any errors that may be returned by Stream_put.
   *****            - strs is destoryed in the processing.
   *****/

  char * c      = NULL;
  int    status = 0;

  if (dfs[STREAM__PUT_STRINGS_ARS_ENTRY]) printf("Stream_put_strings : Entering : strs = %s\n",strs);
  c = strtok(strs," ");
  do if ((status = Stream_put (S, c)) != ERROR_NONE) return (status);
  while ((c = strtok(NULL, " \12")) != NULL);
  return (ERROR_NONE);
}

/*****************************************************************************************************************
 ***** Stream_is_buffer_empty ************************************************************************************
 *****************************************************************************************************************/

Boolean Stream_is_buffer_empty (Stream * S)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - none
   ***** Effects  : - Returns TRUE if the S->buffer is empty, FALSE otherwise.
   *****/

  if (S->count == 0) return (TRUE);
  else               return (FALSE);
}

/*****************************************************************************************************************
 ***** Stream_unparse ********************************************************************************************
 *****************************************************************************************************************/

char * Stream_unparse (Stream * S, Boolean should_return_result)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - stdout
   ***** Effects  : - If should_return_result is TRUE, returns a newly-allocated string representation of S.
   *****            - Else, prints a string representation of S to stdout.  Returns NULL.
   *****            - Exits via handle_error if error is incurred.
   *****/

  char * c      = NULL;
  int    i      = 0;
  int    status = 0;

  if (should_return_result) {
    if ((c = (char *) malloc (sizeof(char) * MNO_STRINGS_IN_BUFFER * (MLO_ONE_STRING + 3))) == NULL)
      handle_error (ERROR_ALLOC_FAIL, "executing Stream_unparse");
    if (S->source == NULL) strcpy(c, "source = NULL...");
    else                   strcpy(c, "source = FILE *...");
    for (i = 0; i < S->count ; i++) {
      strcat(c, S->buffer[(S->head + i) % MNO_STRINGS_IN_BUFFER]);
      strcat(c, "...");
    }
    return (c);
  } else {
    if (S->source == NULL) status = printf("source = NULL...");
    else                   status = printf("source = FILE *...");
    for (i = 0; i < S->count ; i++) status = printf("%s...", S->buffer[(S->head + i) % MNO_STRINGS_IN_BUFFER]);
    if (status < 0) handle_error (ERROR_PRINTF_FAIL, "executing Stream_unparse");
    return (NULL);
  }
}

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

II.O. Strings.c

```
/***** Strings, above all, are immutable as hell. ************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
```

```
#include "primitive_datatype_reps_constants.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "prototypes.h"
#include "externs.h"

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

Strings * Strings_create (UShort capacity)
{
  Strings * retval = NULL;
  if ((retval = (Strings *) calloc (1, sizeof(Strings))) == NULL)
    handle_error (ERROR_ALLOC_FAIL, "executing Strings_create");
  else {
    retval->members  = (char **) calloc (capacity, sizeof(char *));
    retval->count    = 0;
    retval->capacity = capacity;
    return (retval);
  }
}

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

void Strings_free (Strings * strs)
{
  if (strs == NULL) return;
  free (strs->members);
  free (strs);
}

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

void Strings_insert (Strings * strs, char * crse)
{
  if (dfs[STRINGS__INSERT_ARGS_ENTRY]) {
    printf("Strings_insert : strs = ");
    Strings_unparse (strs, FALSE);
    printf("\n");
    printf("Strings_insert : crse = %s\n", crse);
  }
  if ((strs->count + 1) > strs->capacity) handle_error (ERROR_CAPACITY_FULL, "executing Strings_insert");
  strs->members[strs->count++] = crse;
}

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

void Strings_unparse (Strings * strs, Boolean srr)
{
  int i = 0;
  if (srr) handle_error (ERROR_NOT_YET_IMPLEMENTED, "executing Strings_unparse with srr = TRUE");
  else {
    if (strs == NULL) printf("NULL");
    else for (i = 0 ; i < strs->count ; i++) printf("%s ", strs->members[i]);
  }
}

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

void Strings_unparse_X (Env * env, Strings * set, int tab)
{
  int    i      = 0;
  char   buffer [MLO_BUFFER];
  if (set == NULL) Display_psx (env, "NULL", tab);
  else {
    if (set->count == 0) Display_psx (env, "--", tab);
    else {
      Display_psx (env, "", tab);
      for (i = 0; i < set->count ; i++) {
        if (i < (set->count - 1)) sprintf(buffer, "%s; ", set->members[i]);
        else                      sprintf(buffer, "%s", set->members[i]);
        if (Display_ps (env, buffer) == ERROR_OUT_OF_BOUNDS) {
          Display_psn (env, "");
          Display_psx (env, buffer, tab);
        }}}}
}

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

II.P. StudPrefs.c

```
/*********************************************************************************************************
***** General Info ***************************************************************************************
**********************************************************************************************************
*****
***** File           : StudPrefs.c
*****
***** Author         : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
*****                             MIT Master of Engineering in Electrical Engineering and Computer Science '95
*****
***** Background     : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
*****                  designed and implemented in the spring of 1995 as the author's Master of Engineering thesis,
*****                  under the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.
*****                  See main.c for a more complete description of AAA.
*****
*****                  This file contains the source code for the StudPrefs datatype.
*****
***** Compiling Env : gcc -ansi -pedantic -c StudPrefs.c
*****
***** Code Status                  : 1.0.0 WIP
***** Last Modification Date & Time : April 27, 1995
*****
**********************************************************************************************************
***** Overview of Datatype *******************************************************************************
**********************************************************************************************************
*****
***** An instance of this datatype captures the preference information for a student.
*****
**********************************************************************************************************
***** Operations on Datatype *****************************************************************************
**********************************************************************************************************
*****
***** int StudPrefs_initialize (StudPrefs * sp)
*****
***** Requires : - sp is a non-NULL StudPrefs instance.
***** Modifies : - sp
***** Effects  : - Sets all of sp's components to "zero" values.
*****            - Returns ERROR_NONE.
*****
**********************************************************************************************************
*****
***** int StudPrefs_clear (StudPrefs * sp)
*****
***** Requires : - sp is a non-NULL StudPrefs instance.
***** Modifies : - sp
***** Effects  : - Frees any memory allocated for sp's components.
*****            - Returns ERROR_NONE.
*****
**********************************************************************************************************
*****
***** int StudPrefs_parse (char * fn, StudPrefs * sp)
*****
***** Requires : - fn is the name of a file containing student status information.
*****            - sp is a non-NULL StudPrefs instance.
***** Modifies : - sp
***** Effects  : - Parses the student information in the file named by fn into sp.
*****            - Returns any errors.
***** Bugs     : - Does not handle mid-line comments.
*****
**********************************************************************************************************
*****
***** char * StudPrefs_unparse (StudPrefs * sp, Boolean should_return_result)
*****
***** Requires : - none
***** Modifies : - stdout
***** Effects  : - If should_return_result is TRUE, returns a newly-allocated string representation of sp.
*****            - Else, prints a string representation of sp to stdout.  Returns NULL.
*****            - Exits via handle_error if error is incurred.
*****
**********************************************************************************************************
*****
***** void StudPrefs_unparse_X  (Env * env, StudPrefs * sp, AcadProgs * AP)
*****
```

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

/******************************************************************************************************************
 ***** Header Files ***********************************************************************************************
 ******************************************************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"                      /* MLO_                          */
#include "primitive_datatype_reps_constants.h"
#include "datatype_reps_constants.h"        /* CRSE_NUMBER_REL_TO_NEXT_AND    */
#include "table_constants.h"                /* FIELD_COURSES_ALREADY_TAKEN, etc. */
#include "error_and_trace.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"                  /* StudPrefs                     */
#include "prototypes.h"
#include "externs.h"                        /* StudPrefsFileFields           */

/******************************************************************************************************************
 ***** StudPrefs_initialize ***************************************************************************************
 ******************************************************************************************************************/

int StudPrefs_initialize (StudPrefs * sp)
{
  /*****
   ***** Requires : - sp is a non-NULL StudPrefs instance.
   ***** Modifies : - sp
   ***** Effects  : - Sets all of sp's components to "zero" values.
   *****            - Returns ERROR_NONE.
   *****/

  sp->date_of_grad = NULL;
  sp->majors       = NULL;
  sp->minors       = NULL;
  sp->conc         = NULL;

  sp->max_course_load  = 0;
  sp->max_class_hours  = 0;
  sp->max_lab_hours    = 0;
  sp->max_prep_hours   = 0;

  sp->min_course_load  = 0;
  sp->min_class_hours  = 0;
  sp->min_lab_hours    = 0;
  sp->min_prep_hours   = 0;

  sp->want_to_take_next_term = NULL;
  sp->want_to_take           = NULL;
  sp->dont_want_to_take      = NULL;

  return (ERROR_NONE);
}

/******************************************************************************************************************
 ***** StudPrefs_clear ********************************************************************************************
 ******************************************************************************************************************/

int StudPrefs_clear (StudPrefs * sp)
{
  /*****
   ***** Requires : - sp is a non-NULL StudPrefs instance.
   ***** Modifies : - sp
   ***** Effects  : - Frees any memory allocated for sp's components.
   *****            - Returns ERROR_NONE.
   *****/

  if (sp->date_of_grad           != NULL) free (sp->date_of_grad);
  if (sp->majors                 != NULL) Degrees_free (sp->majors);
  if (sp->minors                 != NULL) Degrees_free (sp->minors);
  if (sp->conc                    != NULL) Degrees_free (sp->conc);
  if (sp->want_to_take_next_term != NULL) CrseSet_free (sp->want_to_take_next_term);
  if (sp->want_to_take            != NULL) CrseSet_free (sp->want_to_take);
  if (sp->dont_want_to_take       != NULL) CrseSet_free (sp->dont_want_to_take);
  return (ERROR_NONE);
}

/*****************************************************************************************************************
 ***** StudPrefs_parse *******************************************************************************************
 *****************************************************************************************************************/

int StudPrefs_parse (char * fn, AcadProgs * AP, StudPrefs * sp)
{
  /*****
   ***** Requires : - fn is the name of a file containing student status information.
   *****            - sp is a non-NULL StudPrefs instance.
   ***** Modifies : - sp
   ***** Effects  : - Parses the student information in the file named by fn into sp.
   *****            - Returns any errors.
   ***** Bugs     : - Does not handle mid-line comments.
   *****/

  FILE   * fp         = NULL;
  char     one_string [MLO_ONE_STRING];

  int num    = 0;
  int i      = 0;
  int status = 0;

  StudPrefs_clear (sp);

  if ((fp = fopen(fn, "r")) == NULL) return (ERROR_FOPEN_FAIL);
  while (TRUE) {
    if (fscanf(fp, "%s", one_string) == EOF) break;
    if (*one_string == '#') {
      if (fscanf(fp, UP_TO_NEWLINE, one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
      continue;
    }
    for (i = 0; i < NO_STUD_PREFS_FILE_FIELD_CODES ; i++) if (!strcmp(one_string, StudPrefsFileFields[i])) break;
    if (i == NO_STUD_PREFS_FILE_FIELD_CODES) return (ERROR_INVALID_STUD_PREFS_FIELD);
    if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);  /***** : *****/

    switch (i) {

    case FIELD_DATE_OF_GRAD:  /*************************************************************************************/

      if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
      convert_to_upper (one_string);
      for (i = 0; i < NO_SEASON_CODES ; i++)
        if (!strcmp(one_string, SeasonNames[i])) break;
      if (i == NO_SEASON_CODES) return (ERROR_INVALID_STUD_PREFS_DATE_OF_GRAD_SEASON_VALUE);
      if ((sp->date_of_grad = (Term *) malloc (sizeof(Term))) == NULL) return (ERROR_ALLOC_FAIL);
      (sp->date_of_grad)->season = i;
      if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
      (sp->date_of_grad)->year = atoi (one_string);
      if (((sp->date_of_grad)->year < EARLIEST_VALID_YEAR) ||
          ((sp->date_of_grad)->year > LATEST_VALID_YEAR))
        return (ERROR_INVALID_STUD_PREFS_DATE_OF_GRAD_YEAR_VALUE);
      break;

    case FIELD_MAJORS:  /*******************************************************************************************/

      if (fscanf(fp, SKIP_SPACE_UP_TO_NEWLINE, one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
      if ((status = Degrees_parse (one_string, AP, DEGREE_TYPE_MAJOR, &(sp->majors))) != ERROR_NONE) return (status);
      break;

    case FIELD_MINORS:  /*******************************************************************************************/

      if (fscanf(fp, SKIP_SPACE_UP_TO_NEWLINE, one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
      if ((status = Degrees_parse (one_string, AP, DEGREE_TYPE_MINOR, &(sp->minors))) != ERROR_NONE) return (status);
      break;

    case FIELD_CONC:  /*********************************************************************************************/

      if (fscanf(fp, SKIP_SPACE_UP_TO_NEWLINE, one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
      if ((status = Degrees_parse (one_string, AP, DEGREE_TYPE_CONC, &(sp->conc))) != ERROR_NONE) return (status);
      break;
```

```
case FIELD_MAX_COURSE_LOAD:  /*************************************************************************************/

  if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
  convert_to_upper (one_string);
  if (!strcmp(one_string, "NONE")) sp->max_course_load = VERY_SHORT_NUM_MAX_NONE;
  else                             sp->max_course_load = atoi(one_string);
  break;

case FIELD_MAX_CLASS_HOURS:  /*************************************************************************************/

  if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
  convert_to_upper (one_string);
  if (!strcmp(one_string, "NONE")) sp->max_class_hours = VERY_SHORT_NUM_MAX_NONE;
  else                             sp->max_class_hours = atoi(one_string);
  break;

case FIELD_MAX_LAB_HOURS:  /***************************************************************************************/

  if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
  convert_to_upper (one_string);
  if (!strcmp(one_string, "NONE")) sp->max_lab_hours = VERY_SHORT_NUM_MAX_NONE;
  else                             sp->max_lab_hours = atoi(one_string);
  break;

case FIELD_MAX_PREP_HOURS:  /**************************************************************************************/

  if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
  convert_to_upper (one_string);
  if (!strcmp(one_string, "NONE")) sp->max_prep_hours = VERY_SHORT_NUM_MAX_NONE;
  else                             sp->max_prep_hours = atoi(one_string);
  break;

case FIELD_MIN_COURSE_LOAD:  /*************************************************************************************/

  if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
  convert_to_upper (one_string);
  if (!strcmp(one_string, "NONE")) sp->min_course_load = VERY_SHORT_NUM_MIN_NONE;
  else                             sp->min_course_load = atoi(one_string);
  break;

case FIELD_MIN_CLASS_HOURS:  /*************************************************************************************/

  if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
  convert_to_upper (one_string);
  if (!strcmp(one_string, "NONE")) sp->min_class_hours = VERY_SHORT_NUM_MIN_NONE;
  else                             sp->min_class_hours = atoi(one_string);
  break;

case FIELD_MIN_LAB_HOURS:  /***************************************************************************************/

  if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
  convert_to_upper (one_string);
  if (!strcmp(one_string, "NONE")) sp->min_lab_hours = VERY_SHORT_NUM_MIN_NONE;
  else                             sp->min_lab_hours = atoi(one_string);
  break;

case FIELD_MIN_PREP_HOURS:  /**************************************************************************************/

  if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
  convert_to_upper (one_string);
  if (!strcmp(one_string, "NONE")) sp->min_prep_hours = VERY_SHORT_NUM_MIN_NONE;
  else                             sp->min_prep_hours = atoi(one_string);
  break;

case FIELD_WANT_TO_TAKE_NEXT_TERM:  /******************************************************************************/

  if ((status = CrseSet_parse (&fp, &(sp->want_to_take_next_term))) != ERROR_NONE)
    handle_warning (WARNING_BAD_CRSE_LIST, "StudStatus:parse");
  CrseSet_sort (sp->want_to_take_next_term);
  break;

case FIELD_WANT_TO_TAKE:  /****************************************************************************************/

  if ((status = CrseSet_parse (&fp, &(sp->want_to_take))) != ERROR_NONE)
    handle_warning (WARNING_BAD_CRSE_LIST, "StudStatus:parse");
  CrseSet_sort (sp->want_to_take);
```

```
      break;

    case FIELD_DONT_WANT_TO_TAKE:  /*****************************************************************************/

      if ((status = CrseSet_parse (&fp, &(sp->dont_want_to_take))) != ERROR_NONE)
        handle_warning (WARNING_BAD_CRSE_LIST, "StudStatus:parse");
      CrseSet_sort (sp->dont_want_to_take);
      break;

    case FIELD_OVERRIDE_POI:  /**********************************************************************************/

      if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
      convert_to_upper (one_string);
      if (!strcmp(one_string, "YES")) sp->override_poi = TRUE;
      else                            sp->override_poi = FALSE;
      break;

    case FIELD_CONSIDER_UNDERGRAD:  /****************************************************************************/

      if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
      convert_to_upper (one_string);
      if (!strcmp(one_string, "YES")) sp->consider_undergrad = TRUE;
      else                            sp->consider_undergrad = FALSE;
      break;

    case FIELD_CONSIDER_GRAD:  /*********************************************************************************/

      if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_PREFS_FILE);
      convert_to_upper (one_string);
      if (!strcmp(one_string, "YES")) sp->consider_grad = TRUE;
      else                            sp->consider_grad = FALSE;
      break;
    }
  }
  return (fclose (fp));
}

/*************************************************************************************************************
 ***** StudPrefs_unparse ************************************************************************************
 *************************************************************************************************************/

char * StudPrefs_unparse (StudPrefs * sp, AcadProgs * AP, Boolean should_return_result)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - stdout
   ***** Effects  : - If should_return_result is TRUE, returns a newly-allocated string representation of sp.
   *****            - Else, prints a string representation of sp to stdout.  Returns NULL.
   *****            - Exits via handle_error if error is incurred.
   *****/

  char * c         = NULL;
  VeryShortNum vsn = 0;

  if (should_return_result)
    handle_error (ERROR_NOT_YET_IMPLEMENTED, "executing StudPrefs_unparse with should_return_result = TRUE");
  else {
    handle_error (ERROR_NOT_YET_IMPLEMENTED, "executing StudPrefs_unparse with should_return_result = FALSE");
  }
}
/*************************************************************************************************************
    if (printf(UNPARSE_LABEL "%s %d\n", StudPrefsFileFields[FIELD_DATE_OF_GRAD],
               SeasonNames[(sp->date_of_grad)->season], (sp->date_of_grad)->year) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");

    if (printf(UNPARSE_LABEL, StudPrefsFileFields[FIELD_MAJORS]) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");
    Degrees_unparse (sp->majors, AP, FALSE);
    printf("\n");

    if (printf(UNPARSE_LABEL, StudPrefsFileFields[FIELD_MINORS]) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");
    Degrees_unparse (sp->minors, AP, FALSE);
    printf("\n");

    if (printf(UNPARSE_LABEL, StudPrefsFileFields[FIELD_CONC]) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");
    Degrees_unparse (sp->conc, AP, FALSE);
    printf("\n");

    vsn = sp->max_course_load;
    if (printf(UNPARSE_LABEL, StudPrefsFileFields[FIELD_MAX_COURSE_LOAD]) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");
    if (vsn != VERY_SHORT_NUM_MAX_NONE) printf("%d\n", vsn); else printf("NONE\n");

    vsn = sp->max_class_hours;
    if (printf(UNPARSE_LABEL, StudPrefsFileFields[FIELD_MAX_CLASS_HOURS]) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");
    if (vsn != VERY_SHORT_NUM_MAX_NONE) printf("%d\n", vsn); else printf("NONE\n");

    vsn = sp->max_lab_hours;
    if (printf(UNPARSE_LABEL, StudPrefsFileFields[FIELD_MAX_LAB_HOURS]) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");
    if (vsn != VERY_SHORT_NUM_MAX_NONE) printf("%d\n", vsn); else printf("NONE\n");

    vsn = sp->max_prep_hours;
    if (printf(UNPARSE_LABEL, StudPrefsFileFields[FIELD_MAX_PREP_HOURS]) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");
    if (vsn != VERY_SHORT_NUM_MAX_NONE) printf("%d\n", vsn); else printf("NONE\n");

    vsn = sp->min_course_load;
    if (printf(UNPARSE_LABEL, StudPrefsFileFields[FIELD_MIN_COURSE_LOAD]) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");
    if (vsn != VERY_SHORT_NUM_MIN_NONE) printf("%d\n", vsn); else printf("NONE\n");

    vsn = sp->min_class_hours;
    if (printf(UNPARSE_LABEL, StudPrefsFileFields[FIELD_MIN_CLASS_HOURS]) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");
    if (vsn != VERY_SHORT_NUM_MIN_NONE) printf("%d\n", vsn); else printf("NONE\n");

    vsn = sp->min_lab_hours;
    if (printf(UNPARSE_LABEL, StudPrefsFileFields[FIELD_MIN_LAB_HOURS]) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");
    if (vsn != VERY_SHORT_NUM_MIN_NONE) printf("%d\n", vsn); else printf("NONE\n");

    vsn = sp->min_prep_hours;
    if (printf(UNPARSE_LABEL, StudPrefsFileFields[FIELD_MIN_PREP_HOURS]) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");
    if (vsn != VERY_SHORT_NUM_MIN_NONE) printf("%d\n", vsn); else printf("NONE\n");

    if (printf(UNPARSE_LABEL "%s\n", StudPrefsFileFields[FIELD_WANT_TO_TAKE_NEXT_TERM],
               c = CrseNumberSet_unparse (sp->want_to_take_next_term, TRUE)) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");
    free (c);
    if (printf(UNPARSE_LABEL "%s\n", StudPrefsFileFields[FIELD_WANT_TO_TAKE],
               c = CrseNumberSet_unparse (sp->want_to_take, TRUE)) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");
    free (c);
    if (printf(UNPARSE_LABEL "%s\n", StudPrefsFileFields[FIELD_DONT_WANT_TO_TAKE],
               c = CrseNumberSet_unparse (sp->dont_want_to_take, TRUE)) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");
    free (c);

    if (printf(UNPARSE_LABEL "%s\n",
               StudPrefsFileFields[FIELD_OVERRIDE_POI], Boolean_unparse(sp->override_poi,TRUE)) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudPrefs_unparse");

    return (NULL);
  }
}
 ******************************************************************************************************/

/*****************************************************************************************************
 ***** StudPrefs_unparse_X ***************************************************************************
 *****************************************************************************************************/

void StudPrefs_unparse_X  (Env * env, StudPrefs * sp, AcadProgs * AP)
{
  char * c         = NULL;
  VeryShortNum vsn = 0;
  char buffer[MLO_BUFFER];

  Display_ps  (env, StudPrefsFileFields[FIELD_DATE_OF_GRAD]);
  Display_psx (env, ":", TAB1);
Display_psx (env, SeasonNames[(sp->date_of_grad)->season], TAB2);
sprintf     (buffer, " %d", (sp->date_of_grad)->year);
Display_psn (env, buffer);

Display_ps        (env, StudPrefsFileFields[FIELD_MAJORS]);
Display_psx       (env, ":", TAB1);
Degrees_unparse_X (env, sp->majors, AP, TAB2);
Display_psn       (env, "");

Display_ps        (env, StudPrefsFileFields[FIELD_MINORS]);
Display_psx       (env, ":", TAB1);
Degrees_unparse_X (env, sp->minors, AP, TAB2);
Display_psn       (env, "");

Display_ps        (env, StudPrefsFileFields[FIELD_CONC]);
Display_psx       (env, ":", TAB1);
Degrees_unparse_X (env, sp->conc, AP, TAB2);
Display_psn       (env, "");

Display_ps  (env, StudPrefsFileFields[FIELD_MAX_COURSE_LOAD]);
Display_psx (env, ":", TAB1);
if ((vsn = sp->max_course_load) != VERY_SHORT_NUM_MAX_NONE) {
  sprintf (buffer, "%d", vsn); Display_psnx (env, buffer, TAB2);
} else                         Display_psnx (env, "NONE", TAB2);

Display_ps  (env, StudPrefsFileFields[FIELD_MAX_CLASS_HOURS]);
Display_psx (env, ":", TAB1);
if ((vsn = sp->max_class_hours) != VERY_SHORT_NUM_MAX_NONE) {
  sprintf (buffer, "%d", vsn); Display_psnx (env, buffer, TAB2);
} else                         Display_psnx (env, "NONE", TAB2);

Display_ps  (env, StudPrefsFileFields[FIELD_MAX_LAB_HOURS]);
Display_psx (env, ":", TAB1);
if ((vsn = sp->max_lab_hours) != VERY_SHORT_NUM_MAX_NONE) {
  sprintf (buffer, "%d", vsn); Display_psnx (env, buffer, TAB2);
} else                         Display_psnx (env, "NONE", TAB2);

Display_ps  (env, StudPrefsFileFields[FIELD_MAX_PREP_HOURS]);
Display_psx (env, ":", TAB1);
if ((vsn = sp->max_prep_hours) != VERY_SHORT_NUM_MAX_NONE) {
  sprintf (buffer, "%d", vsn); Display_psnx (env, buffer, TAB2);
} else                         Display_psnx (env, "NONE", TAB2);

Display_ps  (env, StudPrefsFileFields[FIELD_MIN_COURSE_LOAD]);
Display_psx (env, ":", TAB1);
if ((vsn = sp->min_course_load) != VERY_SHORT_NUM_MIN_NONE) {
  sprintf (buffer, "%d", vsn); Display_psnx (env, buffer, TAB2);
} else                         Display_psnx (env, "NONE", TAB2);

Display_ps  (env, StudPrefsFileFields[FIELD_MIN_CLASS_HOURS]);
Display_psx (env, ":", TAB1);
if ((vsn = sp->min_class_hours) != VERY_SHORT_NUM_MIN_NONE) {
  sprintf (buffer, "%d", vsn); Display_psnx (env, buffer, TAB2);
} else                         Display_psnx (env, "NONE", TAB2);

Display_ps  (env, StudPrefsFileFields[FIELD_MIN_LAB_HOURS]);
Display_psx (env, ":", TAB1);
if ((vsn = sp->min_lab_hours) != VERY_SHORT_NUM_MIN_NONE) {
  sprintf (buffer, "%d", vsn); Display_psnx (env, buffer, TAB2);
} else                         Display_psnx (env, "NONE", TAB2);

Display_ps  (env, StudPrefsFileFields[FIELD_MIN_PREP_HOURS]);
Display_psx (env, ":", TAB1);
if ((vsn = sp->min_prep_hours) != VERY_SHORT_NUM_MIN_NONE) {
  sprintf (buffer, "%d", vsn); Display_psnx (env, buffer, TAB2);
} else                         Display_psnx (env, "NONE", TAB2);

Display_ps        (env, StudPrefsFileFields[FIELD_WANT_TO_TAKE_NEXT_TERM]);
Display_psx       (env, ":", TAB1);
CrseSet_unparse_X (env, sp->want_to_take_next_term, TAB2);
Display_psn       (env, "");

Display_ps        (env, StudPrefsFileFields[FIELD_WANT_TO_TAKE]);
Display_psx       (env, ":", TAB1);
CrseSet_unparse_X (env, sp->want_to_take, TAB2);
Display_psn       (env, "");
```

```
  Display_ps        (env, StudPrefsFileFields[FIELD_DONT_WANT_TO_TAKE]);
  Display_psx       (env, ":", TAB1);
  CrseSet_unparse_X (env, sp->dont_want_to_take, TAB2);
  Display_psn       (env, "");

  Display_ps   (env, StudPrefsFileFields[FIELD_OVERRIDE_POI]);
  Display_psx  (env, ":", TAB1);
  Display_psnx (env, Boolean_unparse(sp->override_poi, TRUE), TAB2);

  Display_ps   (env, StudPrefsFileFields[FIELD_CONSIDER_UNDERGRAD]);
  Display_psx  (env, ":", TAB1);
  Display_psnx (env, Boolean_unparse(sp->consider_undergrad, TRUE), TAB2);

  Display_ps   (env, StudPrefsFileFields[FIELD_CONSIDER_GRAD]);
  Display_psx  (env, ":", TAB1);
  Display_psnx (env, Boolean_unparse(sp->consider_grad, TRUE), TAB2);

}
/*****************************************************************************************************************
 *****************************************************************************************************************
 *****************************************************************************************************************/
```

II.Q. StudStatus.c

```
/*****************************************************************************************************************
 ***** General Info **********************************************************************************************
 *****************************************************************************************************************
 *****
 ***** File         : StudStatus.c
 *****
 ***** Author       : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *****                                MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *****
 ***** Background   : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *****                designed and implemented in the spring of 1995 as the author's Master of Engineering thesis,
 *****                under the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.
 *****                See main.c for a more complete description of AAA.
 *****
 *****                This file contains the source code for the StudStatus datatype.
 *****
 ***** Compiling Env : gcc -ansi -pedantic -c StudStatus.c
 *****
 ***** Code Status                  : 1.0.0 WIP
 ***** Last Modification Date & Time : April 27, 1995
 *****
 *****************************************************************************************************************
 ***** Overview of Datatype **************************************************************************************
 *****************************************************************************************************************
 *****
 ***** An instance of this datatype captures the status information for a student.
 *****
 ***** For example, if the information is:
 *****
 *****     Name                          : Shinpark
 *****     Year                          : freshman
 *****     Courses-Already-Taken         : 18.01, 18.02. 18.03. 18.06,
 *****                                     8.01. 8.02,
 *****                                     5.11, 5.60,
 *****                                     18.313. 21W735. 21F222
 *****                                     END
 *****     Phase-I-Completed?            : yes
 *****     Phase-II-Completed?           : no
 *****     Accumulated-PE-Points         : 12
 *****
 ***** the StudStatus instance corresponding to this is:
 *****
 *****     name                          : Shinpark
 *****     year                          : FRESHMAN
 *****     taken                         : {18.01, 18.02, 18.03, 18.06, 8.01, 8.02,
 *****                                      5.11, 5.60, 18.313, 21W735, 21F222}
 *****     phase_I_complete              : TRUE
 *****     phase_II_complete             : FALSE
 *****     PE_points                     : 12
 *****
 *****************************************************************************************************************
 ***** Operations on Datatype ***********************************************************************************
 ****************************************************************************************************************
 *****
 ***** int StudStatus_initialize (StudStatus * ss)
 *****
 ***** Requires : - ss is a non-NULL StudStatus instance.
 ***** Modifies : - ss
 ***** Effects  : - Sets all of ss's components to "zero" values.
 *****            - Returns ERROR_NONE.
 *****
 ****************************************************************************************************************
 *****
 ***** int StudStatus_clear (StudStatus * ss)
 *****
 ***** Requires : - ss is a non-NULL StudStatus instance.
 ***** Modifies : - ss
 ***** Effects  : - Frees any memory allocated for ss's components; i.e., ss->name and ss->taken.
 *****            - Returns ERROR_NONE.
 *****
 ****************************************************************************************************************
 *****
 ***** int StudStatus_parse (char * fn, StudStatus * stud_stat)
 *****
 ***** Requires : - fn is the name of a file containing student status information.
 *****            - stud_stat is a non-NULL StudStatus instance.
 ***** Modifies : - stud_stat
 ***** Effects  : - Parses the student information in the file named by fn into stud_stat.
 *****            - Returns any errors.
 ***** Bugs     : - Does not handle mid-line comments.
 *****
 ****************************************************************************************************************
 *****
 ***** char * StudStatus_unparse (StudStatus * ss. Boolean should_return_result)
 *****
 ***** Requires : - none
 ***** Modifies : - stdout
 ***** Effects  : - If should_return_result is TRUE, returns a newly-allocated string representation of ss.
 *****            - Else, prints a string representation of ss to stdout.  Returns NULL.
 *****            - Exits via handle_error if error is incurred.
 *****
 ****************************************************************************************************************
 *****
 ***** void StudStatus_unparse_X (Env * env, StudStatus * ss)
 *****
 ***** Requires : -
 ***** Modifies : -
 ***** Effects  : -
 *****
 ****************************************************************************************************************
 ****************************************************************************************************************
 ****************************************************************************************************************/

/****************************************************************************************************************
 ***** Header Files ***********************************************************************************************
 ****************************************************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"                          /* MLO_                               */
#include "primitive_datatype_reps_constants.h"
#include "datatype_reps_constants.h"            /* CRSE_NUMBER_REL_TO_NEXT_AND        */
#include "table_constants.h"                    /* FIELD_COURSES_ALREADY_TAKEN, etc. */
#include "error_and_trace.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"                      /* StudStatus                         */
#include "prototypes.h"
#include "externs.h"                            /* StudStatusFileFields               */

/****************************************************************************************************************
 ***** State Variables ********************************************************************************************
 ****************************************************************************************************************/

static StudStatus saved_ss;
static Boolean    something_saved;
```

```
/*****************************************************************************************************************
 ***** StudStatus_initialize **************************************************************************************
 *****************************************************************************************************************/

int StudStatus_initialize (StudStatus * ss)
{
  /*****
   ***** Requires : - ss is a non-NULL StudStatus instance.
   ***** Modifies : - ss
   ***** Effects  : - Sets all of ss's components to 'zero' values.
   *****            - Returns ERROR_NONE.
   *****/

  ss->name              = NULL;
  ss->year              = 0;
  ss->this_term         = NULL;
  ss->next_term         = NULL;
  ss->taken             = NULL;
  ss->taken_all         = NULL;
  ss->phase_I_complete  = FALSE;
  ss->phase_II_complete = FALSE;
  ss->PE_points         = 0;

  something_saved = FALSE;

  return (ERROR_NONE);
}

/*****************************************************************************************************************
 ***** StudStatus_clear *******************************************************************************************
 *****************************************************************************************************************/

int StudStatus_clear (StudStatus * ss)
{
  /*****
   ***** Requires : - ss is a non-NULL StudStatus instance.
   ***** Modifies : - ss
   ***** Effects  : - Frees any memory allocated for ss's components; i.e., ss->name and ss->taken.
   *****            - Returns ERROR_NONE.
   *****/

  if (ss->name      != NULL) free         (ss->name);
  if (ss->this_term != NULL) free         (ss->this_term);
  if (ss->next_term != NULL) free         (ss->next_term);
  if (ss->taken     != NULL) CrseSet_free (ss->taken);
  if (ss->taken_all != NULL) CrseSet_free (ss->taken_all);
  if (ss->corr_cds  != NULL) free         (ss->corr_cds);
  return (ERROR_NONE);
}

/*****************************************************************************************************************
 ***** StudStatus_save_state **************************************************************************************
 *****************************************************************************************************************/

int StudStatus_save_state (StudStatus * ss)
{
  /***** Note that there is *no* copying here. *****/

  if (something_saved) return (ERROR_SOMETHING_SAVED_ALREADY);
/*
  saved_ss.name              = ss->name;
  saved_ss.year              = ss->year;
  saved_ss.this_term         = ss->this_term;
  saved_ss.next_term         = ss->next_term;
*/
  saved_ss.taken             = ss->taken;
  saved_ss.taken_all         = ss->taken_all;
/*
  saved_ss.corr_cds          = ss->corr_cds;
  saved_ss.other_units       = ss->other_units;
  saved_ss.thesis_units      = ss->thesis_units;
  saved_ss.phase_I_complete  = ss->phase_I_complete;
  saved_ss.phase_II_complete = ss->phase_II_complete;
  saved_ss.swimming_complete = ss->swimming_complete;
  saved_ss.PE_points         = ss->PE_points;
*/
```

```
  something_saved = TRUE;
  return (ERROR_NONE);
}

/************************************************************************************************************************
 ***** StudStatus_restore_state ****************************************************************************************
 ************************************************************************************************************************/

int StudStatus_restore_state (StudStatus * ss)
{
  /***** Note that there is *no* copying here. *****/

  if (!something_saved) return (ERROR_NOTHING_SAVED);

/*
  ss->name              = saved_ss.name;
  ss->year              = saved_ss.year;
  ss->this_term         = saved_ss.this_term;
  ss->next_term         = saved_ss.next_term;
*/
  ss->taken             = saved_ss.taken;
  ss->taken_all         = saved_ss.taken_all;
/*
  ss->corr_cds          = saved_ss.corr_cds;
  ss->other_units       = saved_ss.other_units;
  ss->thesis_units      = saved_ss.thesis_units;
  ss->phase_I_complete  = saved_ss.phase_I_complete;
  ss->phase_II_complete = saved_ss.phase_II_complete;
  ss->swimming_complete = saved_ss.swimming_complete;
  ss->PE_points         = saved_ss.PE_points;
*/

  something_saved = FALSE;
  return (ERROR_NONE);
}

/************************************************************************************************************************
 ***** StudStatus_parse ************************************************************************************************
 ************************************************************************************************************************/

int StudStatus_parse (CrseDesc ** crse_descs, Stats * stats, char * fn, StudStatus * stud_stat)
{
  /*****
   ***** Requires : - fn is the name of a file containing student status information.
   *****            - stud_stat is a non-NULL StudStatus instance.
   ***** Modifies : - stud_stat
   ***** Effects  : - Parses the student information in the file named by fn into stud_stat.
   *****            - Returns any errors.
   ***** Bugs     : - Does not handle mid-line comments.
   *****/

  FILE     * fp       = NULL;
  CrseDesc * cd       = NULL;
  char     one_string [MLO_ONE_STRING];

  int num    = 0;
  int i      = 0;
  int j      = 0;
  int status = 0;

  StudStatus_clear (stud_stat);

  if ((fp = fopen(fn, "r")) == NULL) return (ERROR_FOPEN_FAIL);
  while (TRUE) {
    if (fscanf(fp, "%s", one_string) == EOF) break;
    if (*one_string == '#') {
      if (fscanf(fp, UP_TO_NEWLINE, one_string) == EOF) return (ERROR_BAD_STUD_STAT_FILE);
      continue;
    }
    for (i = 0; i < NO_STUD_STATUS_FILE_FIELD_CODES ; i++) if (!strcmp(one_string, StudStatusFileFields[i])) break;
    if (i == NO_STUD_STATUS_FILE_FIELD_CODES) return (ERROR_INVALID_STUD_STAT_FIELD);
    if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_STAT_FILE);  /***** : *****/

    switch (i) {

    case FIELD_NAME:  /*********************************************************************************************/
```

```
      if (fscanf(fp, "%s". one_string) == EOF) return (ERROR_BAD_STUD_STAT_FILE);
      if ((stud_stat->name = (char *) malloc ((strlen(one_string) + 1) * sizeof (char))) == NULL)
        return (ERROR_ALLOC_FAIL);
      strcpy(stud_stat->name, one_string);
      break;

    case FIELD_YEAR:  /*****************************************************************************************/

      if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_STAT_FILE);
      convert_to_upper (one_string);
      for (i = 0; i < NO_STUD_STATUS_YEAR_FIELD_VALUE_CODES ; i++)
        if (!strcmp(one_string, StudStatusYearFieldValues[i])) break;
      if (i == NO_STUD_STATUS_YEAR_FIELD_VALUE_CODES) return (ERROR_INVALID_STUD_STAT_YEAR_FIELD_VALUE);
      stud_stat->year = i;
      break;

    case FIELD_THIS_TERM:  /************************************************************************************/

      if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_STAT_FILE);
      convert_to_upper (one_string);
      for (i = 0; i < NO_SEASON_CODES ; i++)
        if (!strcmp(one_string, SeasonNames[i])) break;
      if (i == NO_SEASON_CODES) return (ERROR_INVALID_STUD_STAT_THIS_TERM_SEASON_VALUE);
      if ((stud_stat->this_term = (Term *) malloc (sizeof(Term))) == NULL) return (ERROR_ALLOC_FAIL);
      (stud_stat->this_term)->season = i;
      if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_STAT_FILE);
      (stud_stat->this_term)->year = atoi (one_string);
      if (((stud_stat->this_term)->year < EARLIEST_VALID_YEAR) ||
          ((stud_stat->this_term)->year > LATEST_VALID_YEAR)) return (ERROR_INVALID_STUD_STAT_THIS_TERM_YEAR_VALUE);
      break;

    case FIELD_NEXT_TERM:  /************************************************************************************/

      if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_STAT_FILE);
      convert_to_upper (one_string);
      for (i = 0; i < NO_SEASON_CODES ; i++)
        if (!strcmp(one_string, SeasonNames[i])) break:
      if (i == NO_SEASON_CODES) return (ERROR_INVALID_STUD_STAT_NEXT_TERM_SEASON_VALUE);
      if ((stud_stat->next_term = (Term *) malloc (sizeof(Term))) == NULL) return (ERROR_ALLOC_FAIL);
      (stud_stat->next_term)->season = i;
      if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_STAT_FILE);
      (stud_stat->next_term)->year = atoi (one_string);
      if (((stud_stat->next_term)->year < EARLIEST_VALID_YEAR) ||
          ((stud_stat->next_term)->year > LATEST_VALID_YEAR)) return (ERROR_INVALID_STUD_STAT_NEXT_TERM_YEAR_VALUE);
      break;

    case FIELD_COURSES_ALREADY_TAKEN:  /*******************************************************************************/

      if ((status = CrseSet_parse (&fp, &(stud_stat->taken))) != ERROR_NONE)
        handle_warning (WARNING_BAD_CRSE_LIST, "StudStatus:parse");
      CrseSet_sort (stud_stat->taken);

      StudStatus_compute_taken_all(crse_descs, stud_stat, stats);

      break;

    case FIELD_OTHER_UNITS_RECEIVED: /*******************************************************************************/

      if (fscanf(fp, "%d", &num) == EOF) return (ERROR_BAD_STUD_STAT_FILE);
      stud_stat->other_units = num;
      break;

    case FIELD_THESIS_UNITS_COMPLETED: /*****************************************************************************/

      if (fscanf(fp, "%d", &num) == EOF) return (ERROR_BAD_STUD_STAT_FILE);
      stud_stat->thesis_units = num;
      break;

    case FIELD_PHASE_I_COMPLETED:  /*********************************************************************************/

      if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_STAT_FILE);
      convert_to_upper (one_string);
      if (!strcmp(one_string, "YES")) stud_stat->phase_I_complete = TRUE;
      else                            stud_stat->phase_I_complete = FALSE;
      break;

    case FIELD_PHASE_II_COMPLETED:  /********************************************************************************/
```

```
      if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_STAT_FILE);
      convert_to_upper (one_string);
      if (!strcmp(one_string, "YES")) stud_stat->phase_II_complete = TRUE;
      else                            stud_stat->phase_II_complete = FALSE;
      break;

    case FIELD_SWIMMING_REQUIREMENT_COMPLETED: /*********************************************************************/

      if (fscanf(fp, "%s", one_string) == EOF) return (ERROR_BAD_STUD_STAT_FILE);
      convert_to_upper (one_string);
      if (!strcmp(one_string, "YES")) stud_stat->swimming_complete = TRUE;
      else                            stud_stat->swimming_complete = FALSE;
      break;

    case FIELD_ACCUMULATED_PE_POINTS:  /*************************************************************************/

      if (fscanf(fp, "%d", &num) == EOF) return (ERROR_BAD_STUD_STAT_FILE);
      stud_stat->PE_points = num;
      break;
    }
  }
  return (fclose (fp));
}

/*****************************************************************************************************************
 ***** StudStatus_compute_taken_all ******************************************************************************
 *****************************************************************************************************************/

void StudStatus_compute_taken_all (CrseDesc ** crse_descs, StudStatus * stud_stat, Stats * stats)
{
  CrseDesc * cd = NULL;
  int i, j;

  stud_stat->taken_all = CrseSet_copy (stud_stat->taken);
  CrseSet_resize (stud_stat->taken_all, MNO_TAKEN_ALL);

  if (stud_stat->taken != NULL) {
    for (i = 0 ; i < (stud_stat->taken)->count ; i++) {
      cd = CrseDesc_search ((stud_stat->taken)->members[i], crse_descs, stats->listed_count);
      if (cd == NULL) handle_warning (WARNING_UNIDENTIFIED_CRSE_NUM, "StudStatus_compute_taken_all");
      else {
        if (cd->same_subj_as == NULL) continue;
        for (j = 0 ; j < (cd->same_subj_as)->count ; j++)
          CrseSet_insert (stud_stat->taken_all, ((cd->same_subj_as)->members[j])->number);
      }
    }
    CrseSet_sort (stud_stat->taken_all);
  }
/*
  stud_stat->corr_cds = (CrseDesc **) calloc ((stud_stat->taken_all)->count, sizeof(CrseDesc *));
  for (i = 0 ; i < (stud_stat->taken_all)->count ; i++) {
    cd = CrseDesc_search ((stud_stat->taken_all)->members[i], crse_descs, stats->listed_count);
    if (cd == NULL) handle_warning (WARNING_UNIDENTIFIED_CRSE_NUM, "StudStatus_compute_taken_all");
    stud_stat->corr_cds[i] = cd;
  }
*/
}

/*****************************************************************************************************************
 ***** StudStatus_unparse ****************************************************************************************
 *****************************************************************************************************************/

char * StudStatus_unparse (StudStatus * ss, Boolean should_return_result)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - stdout
   ***** Effects  : - If should_return_result is TRUE, returns a newly-allocated string representation of ss.
   *****            - Else, prints a string representation of ss to stdout.  Returns NULL.
   *****            - Exits via handle_error if error is incurred.
   *****/

  char * c      = NULL;

  if (should_return_result)
    handle_error (ERROR_NOT_YET_IMPLEMENTED, "executing StudStatus_unparse with should_return_result = TRUE");
  else {
    handle_error (ERROR_NOT_YET_IMPLEMENTED, "executing StudStatus_unparse with should_return_result = FALSE");
/*
    if (printf(UNPARSE_LABEL "%s\n", StudStatusFileFields[FIELD_NAME], ss->name) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudStatus_unparse");
    if (printf(UNPARSE_LABEL "%s\n", StudStatusFileFields[FIELD_YEAR], StudStatusYearFieldValues[ss->year]) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudStatus_unparse");
    if (printf(UNPARSE_LABEL "%s %d\n", StudStatusFileFields[FIELD_THIS_TERM],
               SeasonNames[(ss->this_term)->season], (ss->this_term)->year) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudStatus_unparse");
    if (printf(UNPARSE_LABEL "%s\n",
               StudStatusFileFields[FIELD_COURSES_ALREADY_TAKEN], c = CrseNumberSet_unparse (ss->taken, TRUE)) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudStatus_unparse");
    free (c);
    if (printf(UNPARSE_LABEL "%s\n",
               StudStatusFileFields[FIELD_PHASE_I_COMPLETED], Boolean_unparse(ss->phase_I_complete,TRUE)) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudStatus_unparse");
    if (printf(UNPARSE_LABEL "%s\n",
               StudStatusFileFields[FIELD_PHASE_II_COMPLETED], Boolean_unparse(ss->phase_II_complete,TRUE)) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudStatus_unparse");
    if (printf(UNPARSE_LABEL "%d\n", StudStatusFileFields[FIELD_ACCUMULATED_PE_POINTS], ss->PE_points) < 0)
      handle_error (ERROR_PRINTF_FAIL, "executing StudStatus_unparse");
    return (NULL);
*/
  }
}

/*****************************************************************************************************************
 ***** StudStatus_unparse_X ***************************************************************************************
 *****************************************************************************************************************/

void StudStatus_unparse_X (Env * env, StudStatus * ss)
{
  char * c      = NULL;
  char buffer[MLO_BUFFER];

  Display_ps   (env, StudStatusFileFields[FIELD_NAME]);
  Display_psx  (env, ":", TAB1);
  Display_psnx (env, ss->name, TAB2);

  Display_ps   (env, StudStatusFileFields[FIELD_YEAR]);
  Display_psx  (env, ":", TAB1);
  Display_psnx (env, StudStatusYearFieldValues[ss->year], TAB2);

  Display_ps  (env, StudStatusFileFields[FIELD_THIS_TERM]);
  Display_psx (env, ":", TAB1);
  Display_psx (env, SeasonNames[(ss->this_term)->season], TAB2);
  sprintf     (buffer, " %d", (ss->this_term)->year);
  Display_psn (env, buffer);

  Display_ps  (env, StudStatusFileFields[FIELD_NEXT_TERM]);
  Display_psx (env, ":", TAB1);
  Display_psx (env, SeasonNames[(ss->next_term)->season], TAB2);
  sprintf     (buffer, " %d", (ss->next_term)->year);
  Display_psn (env, buffer);

  Display_ps        (env, StudStatusFileFields[FIELD_COURSES_ALREADY_TAKEN]);
  Display_psx       (env, ":", TAB1);
  CrseSet_unparse_X (env, ss->taken, TAB2);
  Display_psn       (env, "");

  Display_ps        (env, "Taken-All (Debugging)");
  Display_psx       (env, ":", TAB1);
  CrseSet_unparse_X (env, ss->taken_all, TAB2);
  Display_psn       (env, "");

  Display_ps   (env, StudStatusFileFields[FIELD_THESIS_UNITS_COMPLETED]);
  Display_psx  (env, ":", TAB1);
  sprintf      (buffer, "%d", ss->thesis_units);
  Display_psnx (env, buffer, TAB2);

  Display_ps   (env, StudStatusFileFields[FIELD_OTHER_UNITS_RECEIVED]);
  Display_psx  (env, ":", TAB1);
  sprintf      (buffer, "%d", ss->other_units);
  Display_psnx (env, buffer, TAB2);

  Display_ps   (env, StudStatusFileFields[FIELD_PHASE_I_COMPLETED]);
  Display_psx  (env, ":", TAB1);
  Display_psnx (env, Boolean_unparse(ss->phase_I_complete,TRUE), TAB2);

  Display_ps   (env, StudStatusFileFields[FIELD_PHASE_II_COMPLETED]);
  Display_psx  (env, ":", TAB1);
  Display_psnx (env, Boolean_unparse(ss->phase_II_complete,TRUE), TAB2);

  Display_ps   (env, StudStatusFileFields[FIELD_SWIMMING_REQUIREMENT_COMPLETED]);
  Display_psx  (env, ":", TAB1);
  Display_psnx (env, Boolean_unparse(ss->swimming_complete,TRUE), TAB2);

  Display_ps   (env, StudStatusFileFields[FIELD_ACCUMULATED_PE_POINTS]);
  Display_psx  (env, ":", TAB1);
  sprintf      (buffer, "%d", ss->PE_points);
  Display_psnx (env, buffer, TAB2);
}

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

II.R. SubCS.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps_constants.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "prototypes.h"

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

SubCS * SubCS_create (CrseSet * set, char * set_desc, VeryShortNum req)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - none
   ***** Effects  : - Returns a pointer to a newly allocated SubCS scs, whose members are set to the arguments.
   *****            - If req is zero, then scs->req_num is set to set->count.
   *****/
  SubCS * retval = NULL;
  if ((retval = (SubCS *) malloc (sizeof(SubCS))) == NULL)
    handle_error (ERROR_ALLOC_FAIL, "executing SubCS_create");
  if (req == 0) req = set->count;
  retval->set      = set;
  retval->set_desc = set_desc;
  retval->req_num  = req;
  return (retval);
}

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

SubCS * SubCS_copy (SubCS * scs)
{
  SubCS * retval = NULL;
  retval = SubCS_create(scs->set, scs->set_desc, scs->req_num);
  return (retval);
}

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

void SubCS_free (SubCS * subcs)
{
  /***** Note that the members are *not* freed. *****/
  free (subcs);
}

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

```
Boolean SubCS_satisfied(SubCS * subcs, CrseSet * taken, CrseSet ** applicable,
                Boolean print_if_succ, Boolean print_if_fail, Env * env, int tab)
{
  int count = 0;
  int i     = 0;

  if (subcs == NULL) return (TRUE);
  *applicable = CrseSet_create(MNO_APPLICABLE);
  for (i = 0 ; i < taken->count; i++)
    if (CrseSet_is_member(subcs->set, taken->members[i])) {
      CrseSet_insert (*applicable, taken->members[i]);
      if (++count >= subcs->req_num) {
        if (print_if_succ) {
          Display_psx (env, "Completed by : ", tab);
          CrseSet_unparse_X (env, *applicable, env->posx);
        }
        return (TRUE);
      }
    }
  if (print_if_fail) {
    if ((*applicable)->count > 0) {
      Display_psx (env, "Taken thus far : ", tab);
      CrseSet_unparse_X (env, *applicable, env->posx);
    }
    else Display_psx (env, "No progress", tab);
  }
  return (FALSE);
}

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

void SubCS_unparse (SubCS * scs, Boolean desc_only)
{
  if (desc_only) {
    printf("%d of %s", scs->req_num, scs->set_desc);
  } else {
    printf("%d of ", scs->req_num);
    CrseSet_unparse (scs->set, FALSE);
  }
}

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

void SubCS_unparse_X (Env * env, SubCS * scs, int tab, Boolean desc_only)
{
  char buffer[MLO_BUFFER];
  if (scs == NULL) Display_psx (env, "NULL", tab);
  else {
    if (desc_only) {
      sprintf(buffer, "%d of %s", scs->req_num, scs->set_desc);
      Display_psx (env, buffer, tab);
    } else {
      sprintf(buffer, "%d of ", scs->req_num);
      Display_psx       (env, buffer, tab);
      CrseSet_unparse_X (env, scs->set, env->posx);
    }}
}

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

II.S. SubQualSCS.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps_constants.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "prototypes.h"
```

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

SubQualSCS * SubQualSCS_create (void)
{
  SubQualSCS * retval = NULL;
  if ((retval = (SubQualSCS *) calloc (1, sizeof(SubQualSCS))) == NULL)
    handle_error (ERROR_ALLOC_FAIL, "executing SubQualSCS_create");
  else return (retval);
}

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

SubQualSCS * SubQualSCS_copy (SubQualSCS * sqscs)
{
  SubQualSCS * retval = NULL;
  retval = SubQualSCS_create();
  retval->subcs     = SubCS_copy   (sqscs->subcs);
  retval->including = SubCS_copy   (sqscs->including);
  retval->excluding = CrseSet_copy (sqscs->excluding);
  return (retval);
}

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

void SubQualSCS_free (SubQualSCS * sqscs)
{
  if (sqscs == NULL) return;
  SubCS_free   (sqscs->subcs);
  SubCS_free   (sqscs->including);
  CrseSet_free (sqscs->excluding);
}

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

Boolean SubQualSCS_satisfied (SubQualSCS * sqscs, CrseSet * taken, CrseSet ** applicable,
                      Boolean print_if_succ, Boolean print_if_fail, Env * env, int tab)
{
  CrseSet * stripped = NULL;
  CrseSet * cs       = NULL;
  CrseSet * temp     = NULL;
  Boolean   retval   = FALSE;
  CrseSet * local_applicable = NULL;
  int i;

  stripped = CrseSet_subtract (taken, sqscs->excluding);
  /***** First, check if including is satisfied. *****/

  if (!SubCS_satisfied(sqscs->including, stripped, &cs, FALSE, FALSE, NULL, 0)) {

    /***** If it is not, then determine applicable by checking for subcs and return FALSE. *****/

    SubCS_satisfied(sqscs->subcs, stripped, applicable, FALSE, FALSE, NULL, 0);
    retval = FALSE;

  } else {

    /***** If it is, then check if subcs is satisfied. *****/

    if (!SubCS_satisfied(sqscs->subcs, stripped, &local_applicable, FALSE, FALSE, NULL, 0)) {

      /***** If it is not, we return FALSE. *****/

      *applicable = local_applicable;
      retval = FALSE;

    } else {

      /***** If it is, we make sure that cs (the applicable to including) is included in our result. *****/

      if (cs == NULL) *applicable = local_applicable;
      else {
        *applicable = CrseSet_copy (cs);                   /** put cs into result **/
        temp = CrseSet_subtract (local_applicable, cs);    /** temp = local_applicable - cs **/
        CrseSet_resize (*applicable, (sqscs->subcs)->req_num);
        for (i = 0 ; i < ((sqscs->subcs)->req_num - (cs->count)) ; i++)
          CrseSet_insert (*applicable, temp->members[i]);  /** complete the required number **/
        CrseSet_free (local_applicable);
      }

      retval = TRUE;

      if (print_if_succ) {
        Display_psx (env, "Completed by : ", tab);
        CrseSet_unparse_X (env, *applicable, env->posx);
      }
    }}

  CrseSet_free (stripped);
  CrseSet_free (cs);
  CrseSet_free (temp);

  if (print_if_fail && !retval) {
    if (*applicable != NULL) {
      if ((*applicable)->count > 0) {
        Display_psx (env, "Taken thus far : ", tab);
        CrseSet_unparse_X (env, *applicable, env->posx);
      }
      else Display_psx (env, "No progress", tab);
    }
  }

  return (retval);
}

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

void SubQualSCS_unparse_X (Env * env, SubQualSCS * sqscs, int tab, Boolean subcs_desc_only, Boolean including_desc_only)
{
  if (sqscs == NULL) Display_psx  (env, "NULL", tab);
  else {
    SubCS_unparse_X   (env, sqscs->subcs, tab, subcs_desc_only);
    if (sqscs->including != NULL) {
      if (((sqscs->including)->set)->count > 0) {
        Display_psn       (env, "");
        Display_psx       (env, "Including : ", tab);
        SubCS_unparse_X   (env, sqscs->including, env->posx, including_desc_only);
      }}
    if (sqscs->excluding != NULL) {
      if ((sqscs->excluding)->count > 0) {
        Display_psn       (env, "");
        Display_psx       (env, "Excluding : ", tab);
        CrseSet_unparse_X (env, sqscs->excluding, env->posx);
      }}}
}

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

II.T. Units.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps_constants.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "prototypes.h"

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

Units * Units_create (UShort req, char * units)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - none
   ***** Effects  : - Returns a pointer to a newly allocated Units, whose req and units are set to req and units.
   *****/
  Units * retval = NULL;
  if ((retval = (Units *) malloc (sizeof(Units))) == NULL)
    handle_error (ERROR_ALLOC_FAIL, "executing Units_create");
  retval->req   = req;
  retval->units = units;
}

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

Boolean Units_satisfied (Units * u, UShort req,
                  Boolean print_if_succ, Boolean print_if_fail, Env * env, int tab)
{
  /*****
   ***** Requires : - u is non-NULL.
   ***** Modifies : - none
   ***** Effects  : - Returns TRUE if req >= u->req, FALSE otherwise.
   *****/
  char buffer [MLO_BUFFER];
  if (req >= u->req) {
    if (print_if_succ) {
      sprintf(buffer, "Completed with : %d %s", req, u->units);
      Display_psx (env, buffer, tab);
    }
    return (TRUE);
  } else {
    if (print_if_fail) {
      if (req > 0) {
        sprintf(buffer, "Taken thus far : %d %s", req, u->units);
        Display_psx (env, buffer, tab);
      }
      else Display_psx (env, "No progress in this field", tab);
    }
    return (FALSE);
  }
}

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

void Units_unparse_X (Env * env, Units * u, int tab)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - Display
   ***** Effects  : - Prints a string representation of u to Display at position tab of the current line.
   *****/
  char buffer[MLO_BUFFER];
  if (u == NULL) sprintf(buffer, "NULL");
  else           sprintf(buffer, "%d %s", u->req, u->units);
  Display_psx (env, buffer, tab);
}

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

III. Functions & Procedures

III.A. dp_GIR.c

```
/*****************************************************************************************************************
 * dp_GIR.c
 *
 * Author        : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *                               MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *
 * Abstract      : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *                 designed and implemented in the spring of 1995 as the author's Master of Engineering thesis, under
 *                 the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.  See main.c for
 *                 a more complete description of AAA.
 *
 * Compiling Env : gcc -ansi -pedantic -c dp_GIR.c
 *
 * Code Status            : 0.5.0 WIP
 * Last Modification Date : April 20, 1995
 *
 * A lot of this code may look ugly and redundant.  That was not so.  It used to be very beautiful.  But I had to
 * make it ugly and redundant to make it efficient, okay@%^&?  Trust me.  It took incredible effort to make it
 * this ugly.
 *
 ****************************************************************************************************************/

/****************************************************************************************************************
 ***** Header Files ***********************************************************************************************
 ****************************************************************************************************************/

#include <stdio.h>                    /* printf */
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"
#include "primitive_datatype_reps_constants.h"
#include "datatype_reps_constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"                 /* CrseNumber, Boolean */
#include "prototypes.h"
#include "externs.h"

/****************************************************************************************************************
 ***** Local Constants ********************************************************************************************
 ****************************************************************************************************************/

enum REQ_CAT_TYPE_CODES_ENUM {
  UNDEFINED = 0,
  SUB_CS,
  ONE_OF_SCS,
  SUB_QUAL_SCS,
  UNITS,
  BOOLEAN
  };

enum REQ_CAT_CODES_ENUM {
  CALCULUS_I = 0,      /***** SubCS      *****/
  CALCULUS_II,         /***** SubCS      *****/
  PHYSICS_I,           /***** SubCS      *****/
  PHYSICS_II,          /***** SubCS      *****/
  CHEMISTRY,           /***** SubCS      *****/
  BIOLOGY,             /***** SubCS      *****/
  LABORATORY,          /***** SubCS      *****/
  HASSD_1OR2ORLO,      /***** OneOfSCS   *****/
  HASSD_4OR5,          /***** OneOfSCS   *****/
  HASSD_REM,           /***** OneOfSCS   *****/
  HASS,                /***** SubQualSCS *****/
  REST,                /***** SubCS      *****/
  PHASE_I,             /***** SubCS      *****/
  PHASE_II,            /***** SubCS      *****/
  CONCENTRATION,       /***** UNDEFINED  *****/
  SWIMMING,            /***** Boolean    *****/
  PHYSICAL_EDUCATION   /***** Units      *****/
  };

#define NO_REQ_CATS           17
#define MNO_CRSES_TO_DISPLAY  10

#define GIR_TAB1  0
#define GIR_TAB2  150

/****************************************************************************************************************
 ***** Local Datatypes ********************************************************************************************
 ****************************************************************************************************************/

typedef union _ReqCatUnion {
  SubCS      * subcs;
  OneOfSCS   * ooscs;
  SubQualSCS * sqscs;
  Units      * units;
  Boolean      bool;
} ReqCatUnion;

/****************************************************************************************************************
 ***** Local Tables ***********************************************************************************************
 ****************************************************************************************************************/
```

```
static char * ReqCatStr[] = {
  "Calculus I",
  "Calculus II",
  "Physics I",
  "Physics II",
  "Chemistry",
  "Biology",
  "Laboratory",
  "HASS-D 1/2/LO",
  "HASS-D 4/5",
  "Third HASS-D",
  "HASS",
  "REST",
  "Phase I",
  "Phase II",
  "Concentration",
  "Swimming Req.",
  "Phys. Ed."
  };

static const VeryShortNum ReqCatType[] = {
  SUB_CS,
  SUB_CS,
  SUB_CS,
  SUB_CS,
  SUB_CS,
  SUB_CS,
  SUB_CS,
  ONE_OF_SCS,
  ONE_OF_SCS,
  ONE_OF_SCS,
  SUB_QUAL_SCS,
  SUB_CS,
  SUB_CS,
  SUB_CS,
  UNDEFINED,
  BOOLEAN,
  UNITS
  };

/*****************************************************************************************************************
 ***** State Variables ********************************************************************************************
 *****************************************************************************************************************/

static ReqCatUnion ReqCat          [NO_REQ_CATS];
static Boolean     ReqCatSatisfied [NO_REQ_CATS];

/*****************************************************************************************************************
 ***** dp_GIR *****************************************************************************************************
 *****************************************************************************************************************/

int dp_GIR (VeryShortNum message, VeryShortNum flag, Essence * ess, AuditResult ** ar)
{
  int status = 0;

  switch (message) {
  case MSG_INITIALIZE           : status = dp_GIR_initialize         (ess->gir); break;
  case MSG_UNPARSE_REQUIRED_X   : status = dp_GIR_unparse_required_X (ess->env); break;
  case MSG_AUDIT                : status = dp_GIR_audit              (flag, ess, ar) ; break;
  default                       : handle_error (ERROR_UNKNOWN_MSG, "received by dp_GIR");
  }
  return (status);
}

/*****************************************************************************************************************
 ***** dp_GIR_initialize ******************************************************************************************
 *****************************************************************************************************************/

int dp_GIR_initialize (GIR * gir)
{
  int i, t;

  /***** ReqCat *************************************************************************************************/

  for (i = 0 ; i < NO_REQ_CATS ; i++)
    switch (ReqCatType[i]) {
    case SUB_CS:              ReqCat[i].subcs   = NULL; break;
    case ONE_OF_SCS:          ReqCat[i].ooscs   = NULL; break;
    case SUB_QUAL_SCS:        ReqCat[i].sqscs   = NULL; break;
    case UNITS:               ReqCat[i].units   = NULL; break;
    }

  ReqCat[CALCULUS_I].subcs  = SubCS_create(gir->categories[t = GIR_FUL_INDEX_CALC1], GIRStrings[t], 1);
  ReqCat[CALCULUS_II].subcs = SubCS_create(gir->categories[t = GIR_FUL_INDEX_CALC2], GIRStrings[t], 1);
  ReqCat[PHYSICS_I].subcs   = SubCS_create(gir->categories[t = GIR_FUL_INDEX_PHYS1], GIRStrings[t], 1);
  ReqCat[PHYSICS_II].subcs  = SubCS_create(gir->categories[t = GIR_FUL_INDEX_PHYS2], GIRStrings[t], 1);
  ReqCat[CHEMISTRY].subcs   = SubCS_create(gir->categories[t = GIR_FUL_INDEX_CHEM] , GIRStrings[t], 1);
  ReqCat[BIOLOGY].subcs     = SubCS_create(gir->categories[t = GIR_FUL_INDEX_BIO]  , GIRStrings[t], 1);
  ReqCat[LABORATORY].subcs  = SubCS_create(gir->categories[t = GIR_FUL_INDEX_LAB]  , GIRStrings[t], 1);

  ReqCat[HASSD_1OR2ORLO].ooscs = OneOfSCS_create ();
  OneOfSCS_insert (ReqCat[HASSD_1OR2ORLO].ooscs,
           SubCS_create(gir->categories[t = GIR_FUL_INDEX_HASS_D_1] , GIRStrings[t], 1));
  OneOfSCS_insert (ReqCat[HASSD_1OR2ORLO].ooscs,
           SubCS_create(gir->categories[t = GIR_FUL_INDEX_HASS_D_2] , GIRStrings[t], 1));
  OneOfSCS_insert (ReqCat[HASSD_1OR2ORLO].ooscs,
           SubCS_create(gir->categories[t = GIR_FUL_INDEX_HASS_D_LO], GIRStrings[t], 1));

  ReqCat[HASSD_4OR5].ooscs = OneOfSCS_create ();
  OneOfSCS_insert (ReqCat[HASSD_4OR5].ooscs,
           SubCS_create(gir->categories[t = GIR_FUL_INDEX_HASS_D_4] , GIRStrings[t], 1));
  OneOfSCS_insert (ReqCat[HASSD_4OR5].ooscs,
           SubCS_create(gir->categories[t = GIR_FUL_INDEX_HASS_D_5] , GIRStrings[t], 1));

  ReqCat[REST].subcs     = SubCS_create(gir->categories[t = GIR_FUL_INDEX_REST] , GIRStrings[t], 2);
  ReqCat[PHASE_I].subcs  = SubCS_create(gir->all_phaseI , "Phase I Subjects" , 1);
  ReqCat[PHASE_II].subcs = SubCS_create(gir->all_phaseII, "Phase II Subjects", 1);

  ReqCat[SWIMMING].bool = TRUE;  /***** what we require is that this be TRUE *****/
  ReqCat[PHYSICAL_EDUCATION].units = Units_create (8, "points");

  dp_GIR_clear_state (gir);

  return (ERROR_NONE);
}

/*****************************************************************************************************************
 ***** dp_GIR_clear_state ****************************************************************************************
 *****************************************************************************************************************/

void dp_GIR_clear_state (GIR * gir)
{
  VeryShortNum i, t;

  OneOfSCS_free (ReqCat[HASSD_REM].ooscs);
  ReqCat[HASSD_REM].ooscs = OneOfSCS_create ();
  OneOfSCS_insert (ReqCat[HASSD_REM].ooscs,
           SubCS_create(gir->categories[t = GIR_FUL_INDEX_HASS_D_3] , GIRStrings[t], 1));

  SubQualSCS_free (ReqCat[HASS].sqscs);
  ReqCat[HASS].sqscs = SubQualSCS_create ();
  (ReqCat[HASS].sqscs)->subcs = SubCS_create(gir->all_hass, "HASS Subjects", 5);

  for (i = 0 ; i < NO_REQ_CATS ; i++) ReqCatSatisfied[i] = FALSE;
}

/*****************************************************************************************************************
 ***** unparse_*_X ***********************************************************************************************
 *****************************************************************************************************************/

int dp_GIR_unparse_required_X (Env * env)
{
  int  i      = 0;

  Display_psn (env, "The requirements for GIR are as follows:");
  Display_psn (env, "");
  for (i = 0; i < NO_REQ_CATS; i++) {
    Display_psx (env, ReqCatStr[i], GIR_TAB1);
    if (i == HASSD_REM) { Display_psnx (env, "One from the 'Unused' categories", GIR_TAB2); continue; }
    if (i == HASS)      { Display_psnx (env, "Five more HASS subjects"         , GIR_TAB2); continue; }
    switch (ReqCatType[i]) {
    case UNDEFINED    : Display_psnx (env, "Undefined", GIR_TAB2) ; break;
    case SUB_CS       :
      if (((ReqCat[i].subcs)->set)->count > MNO_CRSES_TO_DISPLAY)
```

```
          SubCS_unparse_X (env, ReqCat[i].subcs, GIR_TAB2, TRUE);
      else SubCS_unparse_X (env, ReqCat[i].subcs, GIR_TAB2, FALSE);
      Display_psn (env, "");
      if (i == PHASE_I)  { Display_psnx (env, "or Equivalent", GIR_TAB2 + TAB); continue; }
      if (i == PHASE_II) { Display_psnx (env, "or Equivalent", GIR_TAB2 + TAB); continue; }
      break;
    case ONE_OF_SCS   : OneOfSCS_unparse_X   (env, ReqCat[i].ooscs, GIR_TAB2) ; break;
    case SUB_QUAL_SCS :
      SubQualSCS_unparse_X (env, ReqCat[i].sqscs, GIR_TAB2, TRUE, FALSE) ;
      Display_psn (env, "");
      break;
    case UNITS        : Units_unparse_X      (env, ReqCat[i].units, GIR_TAB2) ; Display_psn (env, ""); break;
    case BOOLEAN      :
      Display_psnx (env, "Can be satisfied by passing the swim test,", GIR_TAB2);
      Display_psnx (env, "or by taking a swimming PE class.", GIR_TAB2);
      break;
    }}
  return (ERROR_NONE);
}

/************************************************************************************************************
 ***** dp_GIR_audit *****************************************************************************************
 ************************************************************************************************************/

int dp_GIR_audit (VeryShortNum flag, Essence * ess, AuditResult ** ar)
{
  Env        *  env;
  CrseDesc   ** cds;
  StudStatus *  ss;
  StudPrefs  *  sp;
  Stats      *  st;
  GIR        *  gir;

  Boolean   satisfied          = FALSE;
  CrseSet  * excluding         = NULL;
  CrseSet  * applicable        = NULL;
  CrseSet  * considering       = NULL;
  CrseDesc * considering_cds   [MNO_APPLICABLE];

  CrseDesc * cd = NULL;
  char     * c  = NULL;

  VeryShortNum index = 0;
  VeryShortNum considering_count = 0;
  VeryShortNum satisfied_count = 0;
  VeryShortNum hassd_fsm = 0;
  VeryShortNum REST_count = 0;
  VeryShortNum HASS_count = 0;

  int  cpx1   = 0;
  int  cpx2   = 0;
  char buffer [MLO_BUFFER];

  OneNumCode i, j, t;

  /***** Initialize. ******************************************************************************************/

  if (dfs[DP_GIR_TRACE]) printf("dp_GIR : Setting local variables for Essence members...\n");
  env = ess->env;
  cds = ess->cds;
  ss  = ess->ss;
  sp  = ess->sp;
  st  = ess->st;
  gir = ess->gir;

  *ar = AuditResult_create();

  /***** Verbose Mode *****************************************************************************************/

  if (flag == FLAG_DISPLAY_AUDIT_RESULTS) {

    excluding   = CrseSet_create(MNO_APPLICABLE);
    considering = CrseSet_create(ss->taken_all->count);
    dp_GIR_clear_state (gir);

    Display_psx  (env, "REQ. CATEGORY", GIR_TAB1);
    Display_psnx (env, "STATUS", GIR_TAB2);
```

```
Display_psn  (env, "");

for (i = 0 ; i < ss->taken_all->count ; i++) {
  c = ss->taken_all->members[i];
  if ((cd = CrseDesc_search (c, cds, st->listed_count)) == NULL) printf ("dp_GIR:audit : Could not find %s.\n", c);
  else {
    if (cd->GIR_fulfillment > 0) {
      CrseSet_insert (considering, c);
      considering_cds[considering_count++] = cd;
    }}}
CrseDesc_sort_on_gir (considering_cds, considering_count);

/***** CALCULUS_I through LABORATORY *****************************************************************************/

for (i = CALCULUS_I ; i <= LABORATORY ; i++) {
  if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing %s...\n", ReqCatStr[i]);
  if (dfs[DP_GIR_WAIT_FOR_CLICK]) wait_for_left_click (env);
  Display_psx (env, ReqCatStr[i], GIR_TAB1);
  if (SubCS_satisfied (ReqCat[i].subcs, considering, &applicable, TRUE, TRUE, env, GIR_TAB2)) {
    ReqCatSatisfied[i] = TRUE;
    satisfied_count++;
  } /* else Display_psx (env, "No progress", GIR_TAB2); */
  CrseSet_append ((*ar)->applicable, applicable);
  CrseSet_free (applicable);
  Display_psn  (env, "");
}

/***** HASSD_1OR2ORLO ********************************************************************************************/

if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing %s...\n", ReqCatStr[HASSD_1OR2ORLO]);
if (dfs[DP_GIR_WAIT_FOR_CLICK]) wait_for_left_click (env);
Display_psx (env, ReqCatStr[HASSD_1OR2ORLO], GIR_TAB1);
if (OneOfSCS_satisfied (ReqCat[HASSD_1OR2ORLO].ooscs,considering,&applicable,&index, TRUE, TRUE, env, GIR_TAB2)) {
  ReqCatSatisfied[HASSD_1OR2ORLO] = TRUE;
  satisfied_count++;
  CrseSet_append (excluding, applicable);
  if ((index == 0) || (index == 2))
    OneOfSCS_insert (ReqCat[HASSD_REM].ooscs,
             SubCS_create(gir->categories[t = GIR_FUL_INDEX_HASS_D_2] , GIRStrings[t], 1));
  if ((index == 1) || (index == 2))
    OneOfSCS_insert (ReqCat[HASSD_REM].ooscs,
             SubCS_create(gir->categories[t = GIR_FUL_INDEX_HASS_D_1] , GIRStrings[t], 1));
}
CrseSet_append ((*ar)->applicable, applicable);
CrseSet_free (applicable);
Display_psn  (env, "");

/***** HASSD_4OR5 ************************************************************************************************/

if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing %s...\n", ReqCatStr[HASSD_4OR5]);
if (dfs[DP_GIR_WAIT_FOR_CLICK]) wait_for_left_click (env);
Display_psx (env, ReqCatStr[HASSD_4OR5], GIR_TAB1);
if (OneOfSCS_satisfied (ReqCat[HASSD_4OR5].ooscs, considering, &applicable, &index, TRUE, TRUE, env, GIR_TAB2)) {
  ReqCatSatisfied[HASSD_4OR5] = TRUE;
  satisfied_count++;
  CrseSet_append (excluding, applicable);
  if (index == 0)
    OneOfSCS_insert (ReqCat[HASSD_REM].ooscs,
             SubCS_create(gir->categories[t = GIR_FUL_INDEX_HASS_D_5] , GIRStrings[t], 1));
  if (index == 1)
    OneOfSCS_insert (ReqCat[HASSD_REM].ooscs,
             SubCS_create(gir->categories[t = GIR_FUL_INDEX_HASS_D_4] , GIRStrings[t], 1));
}
CrseSet_append ((*ar)->applicable, applicable);
CrseSet_free (applicable);
Display_psn  (env, "");

/***** HASSD_REM *************************************************************************************************/

if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing %s...\n", ReqCatStr[HASSD_REM]);
if (dfs[DP_GIR_WAIT_FOR_CLICK]) wait_for_left_click (env);
Display_psx (env, ReqCatStr[HASSD_REM], GIR_TAB1);
if (OneOfSCS_satisfied (ReqCat[HASSD_REM].ooscs, considering, &applicable, &index, TRUE, TRUE, env, GIR_TAB2)) {
  ReqCatSatisfied[HASSD_REM] = TRUE;
  satisfied_count++;
  CrseSet_append (excluding, applicable);
}
```

```
CrseSet_append ((*ar)->applicable, applicable);
CrseSet_free (applicable);
Display_psn  (env, "");

/***** HASS *************************************************************************************************/

if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing %s...\n", ReqCatStr[HASS]);
if (dfs[DP_GIR_WAIT_FOR_CLICK]) wait_for_left_click (env);
Display_psx (env, ReqCatStr[HASS], GIR_TAB1);
(ReqCat[HASS].sqscs)->excluding = excluding;
if (SubQualSCS_satisfied (ReqCat[HASS].sqscs, considering, &applicable, TRUE, TRUE, env, GIR_TAB2)) {
  ReqCatSatisfied[HASS] = TRUE;
  satisfied_count++;
} /* else Display_psx (env, "No progress", GIR_TAB2); */
CrseSet_append ((*ar)->applicable, applicable);
CrseSet_free (applicable);
Display_psn  (env, "");

/***** REST *************************************************************************************************/

if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing %s...\n", ReqCatStr[REST]);
if (dfs[DP_GIR_WAIT_FOR_CLICK]) wait_for_left_click (env);
Display_psx (env, ReqCatStr[REST], GIR_TAB1);
if (SubCS_satisfied (ReqCat[REST].subcs, considering, &applicable, TRUE, TRUE, env, GIR_TAB2)) {
  ReqCatSatisfied[REST] = TRUE;
  satisfied_count++;
} /* else Display_psx (env, "No progress", GIR_TAB2); */
CrseSet_append ((*ar)->applicable, applicable);
CrseSet_free (applicable);
Display_psn  (env, "");

/***** PHASE_I **********************************************************************************************/

if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing %s...\n", ReqCatStr[PHASE_I]);
if (dfs[DP_GIR_WAIT_FOR_CLICK]) wait_for_left_click (env);
Display_psx (env, ReqCatStr[PHASE_I], GIR_TAB1);
if (SubCS_satisfied (ReqCat[PHASE_I].subcs, considering, &applicable, TRUE, FALSE, env, GIR_TAB2)) {
  ReqCatSatisfied[PHASE_I] = TRUE;
  satisfied_count++;
} else {
  if (ss->phase_I_complete) {
    ReqCatSatisfied[PHASE_I] = TRUE;
    satisfied_count++;
    Display_psx (env, "Completed by Equivalent", GIR_TAB2);
  } else {
    Display_psx (env, "No progress", GIR_TAB2);
  }}
CrseSet_append ((*ar)->applicable, applicable);
CrseSet_free (applicable);
Display_psn  (env, "");

/***** PHASE_II *********************************************************************************************/

if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing %s...\n", ReqCatStr[PHASE_II]);
if (dfs[DP_GIR_WAIT_FOR_CLICK]) wait_for_left_click (env);
Display_psx (env, ReqCatStr[PHASE_II], GIR_TAB1);
if (SubCS_satisfied (ReqCat[PHASE_II].subcs, considering, &applicable, TRUE, FALSE, env, GIR_TAB2)) {
  ReqCatSatisfied[PHASE_II] = TRUE;
  satisfied_count++;
} else {
  if (ss->phase_II_complete) {
    ReqCatSatisfied[PHASE_II] = TRUE;
    satisfied_count++;
    Display_psx (env, "Completed by Equivalent", GIR_TAB2);
  } else {
    Display_psx (env, "No progress", GIR_TAB2);
  }}
CrseSet_append ((*ar)->applicable, applicable);
CrseSet_free (applicable);
Display_psn  (env, "");

/***** SWIMMING *********************************************************************************************/

if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing %s...\n", ReqCatStr[SWIMMING]);
if (dfs[DP_GIR_WAIT_FOR_CLICK]) wait_for_left_click (env);
Display_psx (env, ReqCatStr[SWIMMING], GIR_TAB1);
if (ss->swimming_complete == ReqCat[SWIMMING].bool) {
    ReqCatSatisfied[SWIMMING] = TRUE;
    satisfied_count++;
    Display_psx (env, "Completed", GIR_TAB2);
  } else {
    Display_psx (env, "Not yet completed", GIR_TAB2);
  }
  Display_psn  (env, "");

  /***** PHYSICAL_EDUCATION ******************************************************************************/

  if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing %s...\n", ReqCatStr[PHYSICAL_EDUCATION]);
  if (dfs[DP_GIR_WAIT_FOR_CLICK]) wait_for_left_click (env);
  Display_psx (env, ReqCatStr[PHYSICAL_EDUCATION], GIR_TAB1);
  if (Units_satisfied (ReqCat[PHYSICAL_EDUCATION].units, ss->PE_points, TRUE, TRUE, env, GIR_TAB2)) {
    ReqCatSatisfied[PHYSICAL_EDUCATION] = TRUE;
    satisfied_count++;
  }
  Display_psn  (env, "");

  /* Verbose Mode does not yet return units. */
}

/***** Silent Mode *****************************************************************************************/

else {

  /***** Initialize ***************************************************************************************/

  for (i = 0 ; i < NO_REQ_CATS ; i++) ReqCatSatisfied[i] = FALSE;

  for (i = 0 ; i < ss->taken_all->count ; i++) {
    c = ss->taken_all->members[i];
    if ((cd = CrseDesc_search (c, cds, st->listed_count)) == NULL) printf ("dp_GIR:audit : Could not find %s.\n", c);
    else if (cd->GIR_fulfillment > 0) considering_cds[considering_count++] = cd;
  }
  CrseDesc_sort_on_gir (considering_cds, considering_count);

  if (dfs[DP_GIR_PRINT_CONSIDERING]) {
    printf("dp_GIR : Considering : ");
    for (i = 0 ; i < considering_count ; i++)
      printf("%s; ", considering_cds[i]->number->members[0]->number);
    printf("\n");
  }

  /***** Let's get the easy ones first. ******************************************************************/

  if (ss->phase_I_complete) {
    ReqCatSatisfied[PHASE_I] = TRUE;
    satisfied_count++;
  }
  if (ss->phase_II_complete) {
    ReqCatSatisfied[PHASE_II] = TRUE;
    satisfied_count++;
  }
  if (ss->swimming_complete) {
    ReqCatSatisfied[SWIMMING] = TRUE;
    satisfied_count++;
  }
  if (ss->PE_points >= 8) {
    ReqCatSatisfied[PHYSICAL_EDUCATION] = TRUE;
    satisfied_count++;
  }

  j = 0;
  cd = considering_cds[j];

  while (TRUE) {

    if (j == considering_count) break;

    /***** Sciences : Physics I & II, Calculus I & II, Chemistry, Biology ******************************************/

    if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing the Sciences...\n");
    if ((cd->GIR_fulfillment & GIR_FF_SCIENCE) == GIR_FF_SCIENCE) {
      while ((cd->GIR_fulfillment_qualifier & GIR_FFQ_PHYS1) == GIR_FFQ_PHYS1) {
        if (!ReqCatSatisfied[PHYSICS_I]) {
          ReqCatSatisfied[PHYSICS_I] = TRUE;
        satisfied_count++;
        CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
        (*ar)->appl_units += cd->total_units;
      }
      cd = considering_cds[++j];
      if (j == considering_count) break;
    }}
  if (j == considering_count) break;

  if ((cd->GIR_fulfillment & GIR_FF_SCIENCE) == GIR_FF_SCIENCE) {
    while ((cd->GIR_fulfillment_qualifier & GIR_FFQ_PHYS2) == GIR_FFQ_PHYS2) {
      if (!ReqCatSatisfied[PHYSICS_II]) {
        ReqCatSatisfied[PHYSICS_II] = TRUE;
        satisfied_count++;
        CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
        (*ar)->appl_units += cd->total_units;
      }
      cd = considering_cds[++j];
      if (j == considering_count) break;
    }}
  if (j == considering_count) break;

  if ((cd->GIR_fulfillment & GIR_FF_SCIENCE) == GIR_FF_SCIENCE) {
    while ((cd->GIR_fulfillment_qualifier & GIR_FFQ_CALC1) == GIR_FFQ_CALC1) {
      if (!ReqCatSatisfied[CALCULUS_I]) {
        ReqCatSatisfied[CALCULUS_I] = TRUE;
        satisfied_count++;
        CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
        (*ar)->appl_units += cd->total_units;
      }
      cd = considering_cds[++j];
      if (j == considering_count) break;
    }}
  if (j == considering_count) break;

  if ((cd->GIR_fulfillment & GIR_FF_SCIENCE) == GIR_FF_SCIENCE) {
    while ((cd->GIR_fulfillment_qualifier & GIR_FFQ_CALC2) == GIR_FFQ_CALC2) {
      if (!ReqCatSatisfied[CALCULUS_II]) {
        ReqCatSatisfied[CALCULUS_II] = TRUE;
        satisfied_count++;
        CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
        (*ar)->appl_units += cd->total_units;
      }
      cd = considering_cds[++j];
      if (j == considering_count) break;
    }}
  if (j == considering_count) break;

  if ((cd->GIR_fulfillment & GIR_FF_SCIENCE) == GIR_FF_SCIENCE) {
    while ((cd->GIR_fulfillment_qualifier & GIR_FFQ_CHEM) == GIR_FFQ_CHEM) {
      if (!ReqCatSatisfied[CHEMISTRY]) {
        ReqCatSatisfied[CHEMISTRY] = TRUE;
        satisfied_count++;
        CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
        (*ar)->appl_units += cd->total_units;
      }
      cd = considering_cds[++j];
      if (j == considering_count) break;
    }}
  if (j == considering_count) break;

  if ((cd->GIR_fulfillment & GIR_FF_SCIENCE) == GIR_FF_SCIENCE) {
    while ((cd->GIR_fulfillment_qualifier & GIR_FFQ_BIO) == GIR_FFQ_BIO) {
      if (!ReqCatSatisfied[BIOLOGY]) {
        ReqCatSatisfied[BIOLOGY] = TRUE;
        satisfied_count++;
        CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
        (*ar)->appl_units += cd->total_units;
      }
      cd = considering_cds[++j];
      if (j == considering_count) break;
    }}
  if (j == considering_count) break;

  /***** REST ****************************************************************************************/

  if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing the RESTs...\n");
```

```
while ((cd->GIR_fulfillment & GIR_FF_REST) == GIR_FF_REST) {
  if (!ReqCatSatisfied[REST]) {
    if (++REST_count == 2) {
      ReqCatSatisfied[REST] = TRUE;
      satisfied_count++;
    }
    CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
    (*ar)->appl_units += cd->total_units;
  }
  cd = considering_cds[++j];
  if (j == considering_count) break;
}
if (j == considering_count) break;

/***** LAB ******************************************************************************************/

if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing the LAB...\n");
while ((cd->GIR_fulfillment & GIR_FF_LAB) == GIR_FF_LAB) {
  if (!ReqCatSatisfied[LABORATORY]) {
    ReqCatSatisfied[LABORATORY] = TRUE;
    satisfied_count++;
    CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
    (*ar)->appl_units += cd->total_units;
  }
  cd = considering_cds[++j];
  if (j == considering_count) break;
}
if (j == considering_count) break;

/***** HASS_D ***************************************************************************************/

hassd_fsm = 0;
if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing the HASS-Ds...\n");
while ((cd->GIR_fulfillment & GIR_FF_HASS_D) == GIR_FF_HASS_D) {

  if (dfs[DP_GIR_TRACE_HASSD_FSM]) printf("dp_GIR : Entering HASS-D FSM State # %d...\n", hassd_fsm);
  switch (hassd_fsm) {
  case 0 :
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_LO) == GIR_FFQ_HASS_D_LO) { hassd_fsm = 7; break; }
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_1 ) == GIR_FFQ_HASS_D_1 ) { hassd_fsm = 1; break; }
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_2 ) == GIR_FFQ_HASS_D_2 ) { hassd_fsm = 2; break; }
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_3 ) == GIR_FFQ_HASS_D_3 ) { hassd_fsm = 3; break; }
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_4 ) == GIR_FFQ_HASS_D_4 ) { hassd_fsm = 4; break; }
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_5 ) == GIR_FFQ_HASS_D_5 ) { hassd_fsm = 5; break; }
    break;
  case 1 :
    if (!ReqCatSatisfied[HASSD_1OR2ORLO]) {
      ReqCatSatisfied[HASSD_1OR2ORLO] = TRUE;
      satisfied_count++;
    } else HASS_count++;
    CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
    (*ar)->appl_units += cd->total_units;
    cd = considering_cds[++j];
    if (j == considering_count) break;
    if ((cd->GIR_fulfillment & GIR_FF_HASS_D) != GIR_FF_HASS_D) break;
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_1) == GIR_FFQ_HASS_D_1) break;
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_2) == GIR_FFQ_HASS_D_2) { hassd_fsm = 6; break; }
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_3) == GIR_FFQ_HASS_D_3) { hassd_fsm = 6; break; }
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_4) == GIR_FFQ_HASS_D_4) { hassd_fsm = 4; break; }
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_5) == GIR_FFQ_HASS_D_5) { hassd_fsm = 5; break; }
    break;
  case 2 :
    if (!ReqCatSatisfied[HASSD_1OR2ORLO]) {
      ReqCatSatisfied[HASSD_1OR2ORLO] = TRUE;
      satisfied_count++;
    } else HASS_count++;
    CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
    (*ar)->appl_units += cd->total_units;
    cd = considering_cds[++j];
    if (j == considering_count) break;
    if ((cd->GIR_fulfillment & GIR_FF_HASS_D) != GIR_FF_HASS_D) break;
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_2) == GIR_FFQ_HASS_D_2) break;
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_3) == GIR_FFQ_HASS_D_3) { hassd_fsm = 3; break; }
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_4) == GIR_FFQ_HASS_D_4) { hassd_fsm = 4; break; }
    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_5) == GIR_FFQ_HASS_D_5) { hassd_fsm = 5; break; }
    break;
  case 3 :
```

```
  if (!ReqCatSatisfied[HASSD_REM]) {
    ReqCatSatisfied[HASSD_REM] = TRUE;
    satisfied_count++;
  } else HASS_count++;
  CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
  (*ar)->appl_units += cd->total_units;
  cd = considering_cds[++j];
  if (j == considering_count) break;
  if ((cd->GIR_fulfillment & GIR_FF_HASS_D) != GIR_FF_HASS_D) break;
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_3) == GIR_FFQ_HASS_D_3) break;
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_4) == GIR_FFQ_HASS_D_4) { hassd_fsm = 5; break; }
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_5) == GIR_FFQ_HASS_D_5) { hassd_fsm = 5; break; }
  break;
case 4 :
  if (!ReqCatSatisfied[HASSD_4OR5]) {
    ReqCatSatisfied[HASSD_4OR5] = TRUE;
    satisfied_count++;
  } else HASS_count++;
  CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
  (*ar)->appl_units += cd->total_units;
  cd = considering_cds[++j];
  if (j == considering_count) break;
  if ((cd->GIR_fulfillment & GIR_FF_HASS_D) != GIR_FF_HASS_D) break;
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_4) == GIR_FFQ_HASS_D_4) break;
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_5) == GIR_FFQ_HASS_D_5) { hassd_fsm = 8; break; }
  break;
case 5 :
  if (!ReqCatSatisfied[HASSD_4OR5]) {
    ReqCatSatisfied[HASSD_4OR5] = TRUE;
    satisfied_count++;
  } else HASS_count++;
  CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
  (*ar)->appl_units += cd->total_units;
  cd = considering_cds[++j];
  break;
case 6 :
  if (!ReqCatSatisfied[HASSD_REM]) {
    ReqCatSatisfied[HASSD_REM] = TRUE;
    satisfied_count++;
  }
  CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
  (*ar)->appl_units += cd->total_units;
  cd = considering_cds[++j];
  if (j == considering_count) break;
  if ((cd->GIR_fulfillment & GIR_FF_HASS_D) != GIR_FF_HASS_D) break;
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_2) == GIR_FFQ_HASS_D_2) break;
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_3) == GIR_FFQ_HASS_D_3) break;
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_4) == GIR_FFQ_HASS_D_4) { hassd_fsm = 9; break; }
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_5) == GIR_FFQ_HASS_D_5) { hassd_fsm = 9; break; }
  break;
case 7 :
  if (!ReqCatSatisfied[HASSD_1OR2ORLO]) {
    ReqCatSatisfied[HASSD_1OR2ORLO] = TRUE;
    satisfied_count++;
  } else HASS_count++;
  CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
  (*ar)->appl_units += cd->total_units;
  cd = considering_cds[++j];
  if (j == considering_count) break;
  if ((cd->GIR_fulfillment & GIR_FF_HASS_D) != GIR_FF_HASS_D) break;
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_LO) == GIR_FFQ_HASS_D_LO) break;
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_1 ) == GIR_FFQ_HASS_D_1 ) { hassd_fsm = 10; break; }
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_2 ) == GIR_FFQ_HASS_D_2 ) { hassd_fsm = 6;  break; }
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_3 ) == GIR_FFQ_HASS_D_3 ) { hassd_fsm = 11; break; }
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_4 ) == GIR_FFQ_HASS_D_4 ) { hassd_fsm = 4;  break; }
  if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_5 ) == GIR_FFQ_HASS_D_5 ) { hassd_fsm = 9;  break; }
  break;
case 8 :
  if (!ReqCatSatisfied[HASSD_REM]) {
    ReqCatSatisfied[HASSD_REM] = TRUE;
    satisfied_count++;
  } else HASS_count++;
  CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
  (*ar)->appl_units += cd->total_units;
  cd = considering_cds[++j];
  break;
case 9 :
      if (!ReqCatSatisfied[HASSD_4OR5]) {
        ReqCatSatisfied[HASSD_4OR5] = TRUE;
        satisfied_count++;
      } else HASS_count++;
      CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
      (*ar)->appl_units += cd->total_units;
      cd = considering_cds[++j];
      break;
    case 10 :
      if (!ReqCatSatisfied[HASSD_REM]) {
        ReqCatSatisfied[HASSD_REM] = TRUE;
        satisfied_count++;
      } else HASS_count++;
      CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
      (*ar)->appl_units += cd->total_units;
      cd = considering_cds[++j];
      if (j == considering_count) break;
      if ((cd->GIR_fulfillment & GIR_FF_HASS_D) != GIR_FF_HASS_D) break;
      if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_1) == GIR_FFQ_HASS_D_1) break;
      if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_2) == GIR_FFQ_HASS_D_2) break;
      if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_3) == GIR_FFQ_HASS_D_3) break;
      if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_4) == GIR_FFQ_HASS_D_4) { hassd_fsm = 9; break; }
      if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_5) == GIR_FFQ_HASS_D_5) { hassd_fsm = 9; break; }
      break;
    case 11 :
      if (!ReqCatSatisfied[HASSD_REM]) {
        ReqCatSatisfied[HASSD_REM] = TRUE;
        satisfied_count++;
      } else HASS_count++;
      CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
      (*ar)->appl_units += cd->total_units;
      cd = considering_cds[++j];
      if (j == considering_count) break;
      if ((cd->GIR_fulfillment & GIR_FF_HASS_D) != GIR_FF_HASS_D) break;
      if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_3) == GIR_FFQ_HASS_D_3) break;
      if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_4) == GIR_FFQ_HASS_D_4) { hassd_fsm = 9; break; }
      if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_D_5) == GIR_FFQ_HASS_D_5) { hassd_fsm = 9; break; }
      break;
    }
    if (j == considering_count) break;
  }
  if (j == considering_count) break;

  /***** HASS ******************************************************************************************/

  if (HASS_count >= 5) {
    ReqCatSatisfied[HASS] = TRUE;
    satisfied_count++;
  }

  if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing the HASSs...\n");
  while ((cd->GIR_fulfillment & GIR_FF_HASS) == GIR_FF_HASS) {

    if (!ReqCatSatisfied[HASS]) {
      CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
      (*ar)->appl_units += cd->total_units;
      if (++HASS_count >= 5) {
        ReqCatSatisfied[HASS] = TRUE;
        satisfied_count++;
      }}

    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_PHASE_I) == GIR_FFQ_HASS_PHASE_I) {
      if (!ReqCatSatisfied[PHASE_I]) {
        CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
        (*ar)->appl_units += cd->total_units;
        ReqCatSatisfied[PHASE_I] = TRUE;
        satisfied_count++;
      }}

    if ((cd->GIR_fulfillment_qualifier & GIR_FFQ_HASS_PHASE_II) == GIR_FFQ_HASS_PHASE_II) {
      if (!ReqCatSatisfied[PHASE_II]) {
        CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
        (*ar)->appl_units += cd->total_units;
        ReqCatSatisfied[PHASE_II] = TRUE;
        satisfied_count++;
      }}
```

```
        cd = considering_cds[++j];
        if (j == considering_count) break;
      }
      if (j == considering_count) break;

      /***** PHASE_I *******************************************************************************************/

      if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing the PHASE_I...\n");
      while ((cd->GIR_fulfillment & GIR_FF_PHASE_I) == GIR_FF_PHASE_I) {
        if  (!ReqCatSatisfied[PHASE_I]) {
          CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
          (*ar)->appl_units += cd->total_units;
          ReqCatSatisfied[PHASE_I] = TRUE;
          satisfied_count++;
        }
        cd = considering_cds[++j];
        if (j == considering_count) break;
      }
      if (j == considering_count) break;

      /***** PHASE_II ******************************************************************************************/

      if (dfs[DP_GIR_TRACE_INDEX]) printf("dp_GIR : Auditing the PHASE_II...\n");
      while ((cd->GIR_fulfillment & GIR_FF_PHASE_II) == GIR_FF_PHASE_II) {
        if  (!ReqCatSatisfied[PHASE_II]) {
          CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
          (*ar)->appl_units += cd->total_units;
          ReqCatSatisfied[PHASE_II] = TRUE;
          satisfied_count++;
        }
        cd = considering_cds[++j];
        if (j == considering_count) break;
      }
      if (j == considering_count) break;

      break;
    }
  }

  /***** Fill in AuditResults ******************************************************************************/

  (*ar)->completed     = Strings_create (satisfied_count);
  (*ar)->not_completed = Strings_create (NO_REQ_CATS - satisfied_count);
  (*ar)->not_audited   = Strings_create (1);

  for (i = 0 ; i < NO_REQ_CATS ; i++) {
    if (ReqCatType[i] == UNDEFINED) Strings_insert ((*ar)->not_audited, ReqCatStr[i]);
    else {
      if (ReqCatSatisfied[i]) Strings_insert ((*ar)->completed, ReqCatStr[i]);
      else                    Strings_insert ((*ar)->not_completed, ReqCatStr[i]);
    }}
  if (flag == FLAG_DISPLAY_AUDIT_RESULTS) {
    Display_psn (env, "");
    Display_ps  (env, "Remaining to be completed ");
    cpx1 = env->posx;
    Display_ps  (env, ": ");
    cpx2 = env->posx;
    Strings_unparse_X (env, (*ar)->not_completed, cpx2);
    Display_psn (env, "");
    Display_ps  (env, "Not audited ");
    Display_psx (env, ": ", cpx1);
    Strings_unparse_X (env, (*ar)->not_audited, cpx2);
  }
  CrseSet_free (considering);
  return (ERROR_NONE);

}

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

III.B. dp_VI_P.c

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

```
 * dp_VI_P.c
 *
 * Author        : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *                               MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *
 * Abstract      : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *                 designed and implemented in the spring of 1995 as the author's Master of Engineering thesis, under
 *                 the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.  See main.c for
 *                 a more complete description of AAA.
 *
 * Compiling Env : gcc -ansi -pedantic -c dp_VI_P.c
 *
 * Code Status            : 0.5.0 WIP
 * Last Modification Date : April 20, 1995
 *
 *************************************************************************************************************/

/*************************************************************************************************************
 ***** Header Files ******************************************************************************************
 *************************************************************************************************************/

#include <stdio.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"
#include "primitive_datatype_reps_constants.h"
#include "datatype_reps_constants.h"
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "prototypes.h"
#include "externs.h"

/*************************************************************************************************************
 ***** Local Constants ***************************************************************************************
 *************************************************************************************************************/

enum REQ_CAT_TYPE_CODES_ENUM {
  UNDEFINED = 0,
  SUB_CS,
  ONE_OF_SCS,
  SUB_QUAL_SCS,
  SET_OF_SUB_QUAL_SCS,
  UNITS
  };

enum REQ_CAT_CODES_ENUM {
  REQUIRED = 0,            /***** SubCS            *****/
  THESIS,                  /***** Units            *****/
  RE_MATH,                 /***** SubQualSCS       *****/
  RE_LAB,                  /***** SubCS            *****/
  RE_EC_LARGE,             /***** SetOfSubQualSCS *****/
  RE_EC_SMALL,             /***** SetOfSubQualSCS *****/
  RE_EC_TWO_MORE,          /***** SubQualSCS       *****/
  GRAD_H_UNITS,            /***** Units            *****/
  ED_POINTS,               /***** Units            *****/
  UNRESTRICTED_ELECTIVES   /***** Units            *****/
  };

#define NO_REQ_CATS                      10
#define MNO_CRSES_TO_DISPLAY             20
#define MNO_EC_HEADER_SUBJECTS            5
#define MNO_TAKEN_SUBJECTS_IN_ONE_EC     10

#define DP_TAB1  0
#define DP_TAB2  150

enum VI_P_T1_ENUM {
  VI_P_T1_REQUIRED     = 0x01,
  VI_P_T1_RE_MATH      = 0x02,
  VI_P_T1_LAB          = 0x04,
  VI_P_T1_EC_HEADER    = 0x08,
  VI_P_T1_EC_NONHEADER = 0x10
  };
```

```
enum VI_P_T2_ENUM {
  VI_P_T2_RE_MATH_REQUIRED = 0x01,
  VI_P_T2_RE_MATH_REGULAR  = 0x02,

  VI_P_T2_EC_CAT1 = 0x01,
  VI_P_T2_EC_CAT2 = 0x02,
  VI_P_T2_EC_CAT3 = 0x04,
  VI_P_T2_EC_CAT4 = 0x08,
  VI_P_T2_EC_CAT5 = 0x10,
  VI_P_T2_EC_CAT6 = 0x20,
  VI_P_T2_EC_CAT7 = 0x40
  };

/*****************************************************************************************************************
 ***** Local Datatypes *******************************************************************************************
 *****************************************************************************************************************/

typedef union _ReqCatUnion {
  SubCS      * subcs;
  OneOfSCS   * ooscs;
  SubQualSCS * sqscs;
  SetOfSQSCS * sosqscs;
  Units      * units;
} ReqCatUnion;

typedef struct _EngConcAuditState {
  Boolean        header_seen;
  CrseDesc     * members[MNO_TAKEN_SUBJECTS_IN_ONE_EC];
  VeryShortNum   count;
  VeryShortNum   dfc;   /* Distance From Completion */
} EngConcAuditState;

/*****************************************************************************************************************
 ***** Local Tables **********************************************************************************************
 *****************************************************************************************************************/

static char * ReqCatStr[] = {
  "Required",
  "Thesis Units",
  "Math Rest. Elec.",
  "Lab Rest. Elec.",
  "Large Eng. Conc.",
  "Small Eng. Concs.",
  "Eng. Conc. Elec.",
  "Grad-H Units",
  "Eng. Design Points",
  "Unrest. Elec."
  };

static const VeryShortNum ReqCatType[] = {
  SUB_CS,
  UNITS,
  SUB_QUAL_SCS,
  SUB_CS,
  SET_OF_SUB_QUAL_SCS,
  SET_OF_SUB_QUAL_SCS,
  SUB_QUAL_SCS,
  UNITS,
  UNITS,
  UNITS
  };

static char * Required[] = {
  "6.001", "6.002", "6.003", "6.004", "18.03", "END"
  };

static char * RestElecMathSet[] = {
  "6.041", "18.313", "6.042J", "18.04", "18.06", "18.063", "18.100", "END"
  };

static char * RestElecMathSetIncl[] = {
  "6.041", "18.313", "6.042J", "END"
  };

static char * RestElecLabSet[]    = {
  "6.100", "6.101", "6.111", "6.115", "6.121J", "6.151", "6.152J",
  "6.161", "6.163", "6.170", "6.182", "END"
  };

static char * ECNames[] = {
  "Artificial Intelligence and Applications",
  "Bioelectrical Engineering",
  "Communication, Control, and Signal Processing",
  "Computer Systems and Architecture Engineering",
  "Devices, Circuits and Systems",
  "Electrodynamics and Energy Systems",
  "Theoretical Computer Science",
  };

static char * ECHeaderSubjects[] = {
  "6.034", "6.021J", "6.011", "6.033", "6.012", "6.013", "6.046J", "END"  /
***** ??? Notice that 6.014 is left out! *****/
  };

static char * EC1Subjects[] = {
  "6.036", "6.037", "6.801", "6.802", "9.39",
  "6.824", "6.835", "6.858J", "6.863J", "6.865J", "6.866", "6.867", "6.868J", "6.871", "END"
  };
static char * EC2Subjects[] = {
  "6.022J", "6.023J", "6.501", "6.801", "7.05", "9.35",
  "6.343", "6.345", "6.541J", "6.542J", "6.551J", "6.552J", "6.555J", "6.561J", "6.566J", "6.863J", "6.865J", "END"
  };
static char * EC3Subjects[] = {
  "6.302",
  "6.231", "6.233J", "6.234J", "6.238", "6.241", "6.242",
  "6.243J", "6.244", "6.251J", "6.252J", "6.262", "6.263",
  "6.264J", "6.281J", "6.335", "6.336", "6.341", "6.343",
  "6.344", "6.345", "6.432", "6.433", "6.435", "6.441",
  "6.451", "6.453", "6.454", "6.455J", "6.456J", "6.686",
  "6.687", "END"
  };
static char * EC4Subjects[] = {
  "6.035", "6.313",
  "6.371", "6.821", "6.823", "6.826", "6.845", "6.846",
  "6.847", "6.853", "END"
  };
static char * EC5Subjects[] = {
  "6.017", "6.151", "6.152J", "6.301", "6.302", "6.313", "6.720",
  "6.331", "6.333", "6.334", "6.371", "6.372", "6.373",
  "6.730", "6.732", "6.751", "6.763", "6.771", "6.772",
  "6.774", "6.775", "6.776J", "6.781", "END"
  };
static char * EC6Subjects[] = {
  "6.014", "6.312", "6.601", "22.061",
  "6.334", "6.453", "6.561J", "6.630", "6.631", "6.633",
  "6.634", "6.635", "6.637", "6.641", "6.651J", "6.661",
  "6.662J", "6.671", "6.673", "6.683J", "6.685", "6.686", "END"
  };
static char * EC7Subjects[] = {
  "6.044J", "6.045J", "18.433",
  "6.336", "6.830J", "6.840J", "6.841J", "6.848J",
  "6.851J", "6.852J", "6.854J", "6.858J", "6.875J", "END"
  };

static char * EDSubjects[] = {
  "6.001",  "6.002",  "6.003",  "6.004",
  "6.012",  "6.021J", "6.022J", "6.023J",
  "6.033",  "6.035",  "6.037",  "6.071",
  "6.101",  "6.111",  "6.115",  "6.121J",
  "6.151",  "6.152J", "6.161",  "6.163", "6.170", "6.182",
  "6.233J", "6.234J", "6.301",  "6.302",
  "6.313",  "6.331",  "6.345",
  "6.371",  "6.372",  "6.373",
  "6.542J", "6.551J", "6.566J",
  "6.775",  "6.823",  "6.845",  "6.846",
  "6.863J", "6.871",
  "END"
  };
static OneNumCode EDPoints[] = {
  4,4,4,4,
  4,4,2,3,
  4,8,2,4,
  12,12,12,12,
  12,6,12,12,12,12,
  3,6,4,4,
  2,12,4,
  4,2,2,
  4,4,6,
  4,4,6,4,
  8,8
  };

static OneNumCode ECCatTag[] = {
  VI_P_T2_EC_CAT1,
  VI_P_T2_EC_CAT2,
  VI_P_T2_EC_CAT3,
  VI_P_T2_EC_CAT4,
  VI_P_T2_EC_CAT5,
  VI_P_T2_EC_CAT6,
  VI_P_T2_EC_CAT7
  };

/*****************************************************************************************************************
 ***** Derived Tables ********************************************************************************************
 *****************************************************************************************************************/

static CrseSet * EngConcHeader    = NULL;
static CrseSet * EngConcNonHeader [7];
static CrseSet * EngConcSubjects  [7];
static CrseSet * EngConcAll       = NULL;
static CrseSet * EDPointsSubjects = NULL;

/*****************************************************************************************************************
 ***** State Variables *******************************************************************************************
 *****************************************************************************************************************/

static ReqCatUnion   ReqCat           [NO_REQ_CATS];
static Boolean       ReqCatSatisfied  [NO_REQ_CATS];

/*****************************************************************************************************************
 ***** function prototypes ***************************************************************************************
 *****************************************************************************************************************/

extern int dp_VI_P             (VeryShortNum, VeryShortNum. Essence *, AuditResult **);

static int  initialize         (Essence *);
static int  unparse_required_X (Env *);
static int  audit              (VeryShortNum, Essence *, AuditResult **);
static void clear_state        (void);

/*****************************************************************************************************************
 ***** dp_VI_P ***************************************************************************************************
 *****************************************************************************************************************/

int dp_VI_P (VeryShortNum message. VeryShortNum flag, Essence * ess, AuditResult ** ar)
{
  int status = 0;

  switch (message) {
  case MSG_INITIALIZE          : status = initialize         (ess)     ; break;
  case MSG_UNPARSE_REQUIRED_X  : status = unparse_required_X (ess->env); break;
  case MSG_AUDIT               : status = audit              (flag, ess, ar) ; break;
  default                      : handle_error (ERROR_UNKNOWN_MSG. "received by dp_GIR");
  }
  return (status);
}

/*****************************************************************************************************************
 ***** initialize ************************************************************************************************
 *****************************************************************************************************************/

int initialize (Essence * ess)
{
  VeryShortNum t, i, j, k;

  SubQualSCS * sqscs1 = NULL;
  SubQualSCS * sqscs2 = NULL;
  SubCS      * scs    = NULL;
  CrseSet    * cs     = NULL;
  CrseDesc   * cd     = NULL;
  char       * c      = NULL;
```

```
/***** ReqCat ****************************************************************************************/

for (i = 0 ; i < NO_REQ_CATS ; i++)
  switch (ReqCatType[i]) {
  case SUB_CS:              ReqCat[i].subcs   = NULL; break;
  case UNITS:               ReqCat[i].units   = NULL; break;
  case SUB_QUAL_SCS:        ReqCat[i].sqscs   = NULL; break;
  case ONE_OF_SCS:          ReqCat[i].ooscs   = NULL; break;
  case SET_OF_SUB_QUAL_SCS: ReqCat[i].sosqscs = NULL; break;
  }

/***** Required and Thesis Units *********************************************************************/

ReqCat[REQUIRED].subcs = SubCS_create(CrseSet_new(Required), ReqCatStr[REQUIRED], 0);  /* CrseSet needs to be freed */
for (i = 0 ; i < ReqCat[REQUIRED].subcs->set->count ; i++) {
  cd = CrseDesc_search (ReqCat[REQUIRED].subcs->set->members[i], ess->cds, 0);
  cd->tag_first = VI_P_T1_REQUIRED;
  cd->tag_total = cd->tag_first << 16;
}
ReqCat[THESIS].units   = Units_create(24, "units");

/***** Restricted Electives : Math *******************************************************************/

ReqCat[RE_MATH].sqscs = SubQualSCS_create();
(ReqCat[RE_MATH].sqscs)->subcs =
  SubCS_create(CrseSet_new(RestElecMathSet), ReqCatStr[RE_MATH], 3);
(ReqCat[RE_MATH].sqscs)->including =
  SubCS_create(CrseSet_new(RestElecMathSetIncl), "Required Math", 1);
for (i = 0 ; i < (ReqCat[RE_MATH].sqscs)->subcs->set->count ; i++) {
  cd = CrseDesc_search ((ReqCat[RE_MATH].sqscs)->subcs->set->members[i], ess->cds, 0);
  cd->tag_first  = VI_P_T1_RE_MATH;
  cd->tag_second = VI_P_T2_RE_MATH_REGULAR;
  cd->tag_total  = (cd->tag_first << 16) | (cd->tag_second << 8);
}
for (i = 0 ; i < (ReqCat[RE_MATH].sqscs)->including->set->count ; i++) {
  cd = CrseDesc_search ((ReqCat[RE_MATH].sqscs)->including->set->members[i], ess->cds, 0);
  cd->tag_first  = VI_P_T1_RE_MATH;
  cd->tag_second = VI_P_T2_RE_MATH_REQUIRED;
  cd->tag_total  = (cd->tag_first << 16) | (cd->tag_second << 8);
}

/***** Restricted Electives : Lab ********************************************************************/

ReqCat[RE_LAB].subcs = SubCS_create(CrseSet_new(RestElecLabSet), ReqCatStr[RE_LAB], 1);
for (i = 0 ; i < ReqCat[RE_LAB].subcs->set->count ; i++) {
  cd = CrseDesc_search (ReqCat[RE_LAB].subcs->set->members[i], ess->cds, 0);
  cd->tag_first = VI_P_T1_LAB;
  cd->tag_total = cd->tag_first << 16;

}

/***** Engineering Concentrations ********************************************************************/

EngConcHeader = CrseSet_new(ECHeaderSubjects);
for (i = 0 ; i < EngConcHeader->count ; i++) {
  cd = CrseDesc_search (EngConcHeader->members[i], ess->cds, 0);
  cd->tag_first  = VI_P_T1_EC_HEADER;
  cd->tag_second = ECCatTag[i];
  cd->tag_total  = (cd->tag_first << 16) | (cd->tag_second << 8);
}

EngConcNonHeader[0] = CrseSet_new(EC1Subjects);
EngConcNonHeader[1] = CrseSet_new(EC2Subjects);
EngConcNonHeader[2] = CrseSet_new(EC3Subjects);
EngConcNonHeader[3] = CrseSet_new(EC4Subjects);
EngConcNonHeader[4] = CrseSet_new(EC5Subjects);
EngConcNonHeader[5] = CrseSet_new(EC6Subjects);
EngConcNonHeader[6] = CrseSet_new(EC7Subjects);

for (i = 0 ; i < 7 ; i++)
  for (j = 0 ; j < EngConcNonHeader[i]->count ; j++) {
    cd = CrseDesc_search (EngConcNonHeader[i]->members[j], ess->cds, 0);
    cd->tag_first  = VI_P_T1_EC_NONHEADER;
    cd->tag_second = ECCatTag[i];
    cd->tag_total  = (cd->tag_first << 16) | (cd->tag_second << 8);
  }
  EngConcAll = CrseSet_copy(EngConcHeader);
  for (i = 0 ; i < 7 ; i++) {
    CrseSet_sort (EngConcNonHeader[i]);
    EngConcSubjects[i] = CrseSet_copy(EngConcNonHeader[i]);
    CrseSet_resize (EngConcSubjects[i], (EngConcSubjects[i])->capacity + 1);
    CrseSet_insert(EngConcSubjects[i], EngConcHeader->members[i]);
    CrseSet_sort (EngConcSubjects[i]);
    CrseSet_resize (EngConcAll, EngConcAll->capacity + (EngConcNonHeader[i])->capacity);
    CrseSet_append (EngConcAll, EngConcNonHeader[i]);
  }
  CrseSet_sort (EngConcAll);

  /***** Large Engineering Concentration and Small Engineering Concentrations *****/

  SetOfSQSCS_free (ReqCat[RE_EC_LARGE].sosqscs);
  ReqCat[RE_EC_LARGE].sosqscs = SetOfSQSCS_create ();
  (ReqCat[RE_EC_LARGE].sosqscs)->req_num = 1;

  SetOfSQSCS_free (ReqCat[RE_EC_SMALL].sosqscs);
  ReqCat[RE_EC_SMALL].sosqscs = SetOfSQSCS_create ();
  (ReqCat[RE_EC_SMALL].sosqscs)->req_num = 2;

  for (i = 0 ; i < 7 ; i++) {
    cs = CrseSet_create(MNO_EC_HEADER_SUBJECTS);
    CrseSet_insert (cs, ECHeaderSubjects[i]);
    scs = SubCS_create(cs, "Header", 1);
    sqscs1 = SubQualSCS_create();
    sqscs2 = SubQualSCS_create();
    sqscs1->subcs = SubCS_create (EngConcSubjects[i], ECNames[i], 3);
    sqscs2->subcs = SubCS_create (EngConcSubjects[i], ECNames[i], 2);
    sqscs1->including = scs;
    sqscs2->including = scs;
    SetOfSQSCS_insert (ReqCat[RE_EC_LARGE].sosqscs, sqscs1);
    SetOfSQSCS_insert (ReqCat[RE_EC_SMALL].sosqscs, sqscs2);
  }

  /***** Engineering Concentration Electives *****/

  ReqCat[RE_EC_TWO_MORE].sqscs = SubQualSCS_create();
  (ReqCat[RE_EC_TWO_MORE].sqscs)->subcs = SubCS_create(EngConcAll, "Engineering Concentration Electives", 2);

  /***** Engineering Design Points Subjects ***************************************************************************/

  EDPointsSubjects = CrseSet_new(EDSubjects);
  for (i = 0 ; i < EDPointsSubjects->count ; i++) {
    cd = CrseDesc_search (EDPointsSubjects->members[i], ess->cds, 0);
    cd->tag_third  = EDPoints[i];
    cd->tag_total |= cd->tag_third;
  }

  /***** Graduate H-Level Units, Engineering Design Points, and Unrestricted Electives ******************************/

  ReqCat[GRAD_H_UNITS].units            = Units_create(42, "units");
  ReqCat[ED_POINTS].units               = Units_create(48, "points");
  ReqCat[UNRESTRICTED_ELECTIVES].units = Units_create(60, "units");

  return (ERROR_NONE);
}

/*****************************************************************************************************************
 ***** clear_state *************************************************************************************************
 *****************************************************************************************************************/

void clear_state (void)
{
  int i;

  SetOfSQSCS_free_excludings (ReqCat[RE_EC_SMALL].sosqscs);
  CrseSet_free((ReqCat[RE_EC_TWO_MORE].sqscs)->excluding);
  for (i = 0 ; i < NO_REQ_CATS ; i++) ReqCatSatisfied[i] = FALSE;
}

/*****************************************************************************************************************
 ***** unparse_*_X *************************************************************************************************
 *****************************************************************************************************************/
```

```
int unparse_required_X (Env * env)
{
  int  i      = 0;

  Display_psn (env, "The requirements for VI-P are as follows:");
  Display_psn (env, "");
  for (i = 0; i < NO_REQ_CATS; i++) {

    /***** Print requirement category name *****/

    Display_psx (env, ReqCatStr[i], DP_TAB1);

    /***** Handle any 'exceptions' here *****/

    if (i == RE_EC_LARGE) { Display_psnx (env, "Header plus two subjects from one EC", DP_TAB2) ; continue; }
    if (i == RE_EC_SMALL) { Display_psnx (env, "Header plus one subject from two ECs", DP_TAB2) ; continue; }

    /***** The remaining, by datatype *****/

    switch (ReqCatType[i]) {
    case UNDEFINED    : Display_psnx (env, "Undefined", DP_TAB2) ; break;
    case SUB_CS       :
      if (((ReqCat[i].subcs)->set)->count > MNO_CRSES_TO_DISPLAY)
           SubCS_unparse_X (env, ReqCat[i].subcs, DP_TAB2, TRUE);
      else SubCS_unparse_X (env, ReqCat[i].subcs, DP_TAB2, FALSE);
      Display_psn (env, "");
      break;
    case UNITS        : Units_unparse_X      (env, ReqCat[i].units, DP_TAB2) ; Display_psn (env, ""); break;
    case ONE_OF_SCS   : OneOfSCS_unparse_X   (env, ReqCat[i].ooscs, DP_TAB2) ; break;
    case SUB_QUAL_SCS :
      if (i == RE_MATH       ) SubQualSCS_unparse_X (env, ReqCat[i].sqscs, DP_TAB2, FALSE, FALSE);
      if (i == RE_EC_TWO_MORE) SubQualSCS_unparse_X (env, ReqCat[i].sqscs, DP_TAB2,  TRUE, FALSE);
      Display_psn (env, "");
      break;
    case SET_OF_SUB_QUAL_SCS : SetOfSQSCS_unparse_X (env, ReqCat[i].sosqscs, DP_TAB2, TRUE, FALSE); break;
      default : Display_psnx (env, "Not yet implemented", DP_TAB2) ; break;
    }}
  return (ERROR_NONE);

}

/*****************************************************************************************************************
 ***** audit *****************************************************************************************************
 *****************************************************************************************************************/

int audit (VeryShortNum flag, Essence * ess, AuditResult ** ar)
{
  /***** Local Variables for Essence Members ********************************************/

  Env        *  env;
  CrseDesc   ** cds;
  StudStatus *  ss;
  StudPrefs  *  sp;
  Stats      *  st;
  GIR        *  gir;

  /***** General ***********************************************************************/

  CrseSet  * considering = NULL;

  AuditResult * gir_audit              = NULL;
  AuditResult * none_ue_satisfying_gir = NULL;

  VeryShortNum satisfied_count = 0;  /* ??? don't forget to add this to Verbose Mode */

  UShort total_units = 0;

  /***** Specific to Verbose Mode ******************************************************/

  CrseSet  * applicable = NULL;
  CrseSet  * none_ue    = NULL;
  CrseSet  * ue         = NULL;

  UShort dpc_satisfying_gir_units = 0;
  UShort ue_units                 = 0;
  UShort total_ue_units           = 0;
  UShort units_tbc                = 0;  /* units to be compared */
```

```
Boolean encountered_units_arranged = FALSE;

/***** Specific to Silent Mode ******************************************************/

CrseDesc * considering_cds [MNO_APPLICABLE];

EngConcAuditState * EC_seen [7];
EngConcAuditState * key     = NULL;

UShort EDP_earned = 0;
UShort GHU_earned = 0;

VeryShortNum Required_count    = 0;
VeryShortNum REMath_count      = 0;
VeryShortNum considering_count = 0;
VeryShortNum ECElective_count  = 0;
VeryShortNum ECSmallCats_count = 0;
VeryShortNum EC_all_count      = 0;

VeryShortNum EC_last_seen_code  = 0;
VeryShortNum EC_last_seen_index = 0;

Boolean REMath_required_satisfied      = FALSE;
Boolean REMath_count_satisfied         = FALSE;
Boolean GHU_encountered_units_arranged = FALSE;

/***** Temp Variables ***************************************************************/

CrseDesc * cd   = NULL;
CrseSet  * temp = NULL;
char     * c    = NULL;

char buffer [MLO_BUFFER];

int cpx1 = 0;
int cpx2 = 0;

char i, j, k, l, t;

/***** Initialize *******************************************************************************************/

if (dfs[DP_VI_P_TRACE]) printf("dp_VI_P : Setting local variables for Essence members...\n");
env = ess->env;
cds = ess->cds;
ss  = ess->ss;
sp  = ess->sp;
st  = ess->st;
gir = ess->gir;

*ar = AuditResult_create();

/***** Verbose Mode *****************************************************************************************/

if (flag == FLAG_DISPLAY_AUDIT_RESULTS) {

  if (dfs[DP_VI_P_TRACE]) printf("dp_VI_P : Calling dp_GIR with FLAG_NONE...\n");
  dp_GIR(MSG_AUDIT, FLAG_NONE, ess, &gir_audit);

  if (dfs[DP_VI_P_TRACE]) printf("dp_VI_P : Creating none_ue and considering...\n");
  none_ue     = CrseSet_create(MNO_APPLICABLE);
  considering = CrseSet_create(MNO_APPLICABLE);
  clear_state ();

  Display_psx  (env, "REQ. CATEGORY", DP_TAB1);
  Display_psnx (env, "STATUS", DP_TAB2);
  Display_psn  (env, "");

  if (dfs[DP_VI_P_TRACE]) printf("dp_VI_P : Determining considering...\n");
  for (i = 0 ; i < ss->taken_all->count ; i++) {
    c = ss->taken_all->members[i];
    cd = CrseDesc_search (c, cds, 0);
    if (cd->tag_first > 0) CrseSet_insert (considering, c);
  }

  Display_psx (env, "Completed in GIR", DP_TAB1);
  Strings_unparse_X (env, gir_audit->completed, DP_TAB2);
```

```
Display_psn (env, "");
Display_psn (env, "");

for (i = 0 ; i < NO_REQ_CATS ; i++) {

  if (dfs[DP_VI_P_TRACE_INDEX]) printf("dp_VI_P : Auditing %s...\n", ReqCatStr[i]);
  if (dfs[DP_VI_P_WAIT_FOR_CLICK]) wait_for_left_click (env);

  /***** Handle Unrestricted Electives as an exception. **********************************************************/

  if (i == UNRESTRICTED_ELECTIVES) {

    Display_psn (env, "");

    /***** Determine departmental program courses (other than unrestricted electives) that also satisfy GIRs. *****/

    if (StudStatus_save_state (ss) != ERROR_NONE) exit (-1);

    ss->taken = none_ue;
    StudStatus_compute_taken_all (cds, ss, st);
    dp_GIR(MSG_AUDIT, FLAG_NONE, ess, &none_ue_satisfying_gir);

    Display_psnx (env, "Departmental program courses that also satisfy the GIRs :", DP_TAB1);
    CrseSet_unparse_X (env, none_ue_satisfying_gir->applicable, DP_TAB2);

    dpc_satisfying_gir_units = CrseSet_get_units (none_ue_satisfying_gir->applicable, cds, st->listed_count,
                               &encountered_units_arranged);

    sprintf(buffer, ": Total of %d units. ", dpc_satisfying_gir_units);
    Display_psn (env, buffer);
    if (encountered_units_arranged) Display_psn (env, "'Units arranged' subjects were not counted.");

    CrseSet_free     (ss->taken_all);
    AuditResult_free (none_ue_satisfying_gir);

    if (StudStatus_restore_state (ss) != ERROR_NONE) exit (-1);

    /***** Determine unrestricted electives. ***********************************************************************/

    CrseSet_augment_with_synonyms (none_ue, cds, st->listed_count);
    ue = CrseSet_subtract (ss->taken_all, none_ue);

    CrseSet_augment_with_synonyms (gir_audit->applicable, cds, st->listed_count);
    temp = ue;
    ue = CrseSet_subtract (ue, gir_audit->applicable);
    CrseSet_free (temp);

    temp = ue;
    ue = CrseSet_remove_similar (ue, cds, st->listed_count);
    CrseSet_free (temp);

    Display_psnx (env, "Unrestricted Electives :", DP_TAB1);
    CrseSet_unparse_X (env, ue, DP_TAB2);

    ue_units = CrseSet_get_units (ue, cds, st->listed_count, &encountered_units_arranged);

    sprintf(buffer, ": Subtotal of %d units. ", ue_units);
    Display_psn (env, buffer);
    if (encountered_units_arranged) Display_psn (env, "'Units arranged' subjects were not counted.");

    total_ue_units = ue_units + ss->other_units;

    sprintf(buffer, "Other units received = %d. Total is %d units.", ss->other_units, total_ue_units);
    Display_psnx (env, buffer, DP_TAB2);

    if (Units_satisfied (ReqCat[i].units, total_ue_units, FALSE, FALSE, NULL, 0)) {
      ReqCatSatisfied[i] = TRUE;
      Display_psnx (env, "Completed.", DP_TAB2);
    } else Display_psnx (env, "Not yet completed.", DP_TAB2);

    continue;
  }

  Display_psx (env, ReqCatStr[i], DP_TAB1);

  switch (ReqCatType[i]) {
```

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

case UNDEFINED    : Display_psx (env, "Undefined", DP_TAB2) ; break;

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

case SUB_CS       :

  if (SubCS_satisfied (ReqCat[i].subcs, considering, &applicable, TRUE, TRUE, env, DP_TAB2))
    ReqCatSatisfied[i] = TRUE;
  CrseSet_append (none_ue, applicable);
  CrseSet_free (applicable);
  break;

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

case SUB_QUAL_SCS :

  if (SubQualSCS_satisfied (ReqCat[i].sqscs, considering, &applicable, TRUE, TRUE, env, DP_TAB2))
    ReqCatSatisfied[i] = TRUE;
  CrseSet_append (none_ue, applicable);
  CrseSet_free (applicable);
  break;

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

case SET_OF_SUB_QUAL_SCS :

  if (SetOfSQSCS_satisfied (ReqCat[i].sosqscs, considering, &applicable, TRUE, TRUE, env, DP_TAB2))
    ReqCatSatisfied[i] = TRUE;
  CrseSet_append (none_ue, applicable);
  if (i == RE_EC_LARGE) {
    SetOfSQSCS_set_excludings(ReqCat[RE_EC_SMALL].sosqscs, applicable);
    (ReqCat[RE_EC_TWO_MORE].sqscs)->excluding = CrseSet_copy (applicable);
  }
  if (i == RE_EC_SMALL)
    CrseSet_append ((ReqCat[RE_EC_TWO_MORE].sqscs)->excluding, applicable);
  CrseSet_free (applicable);
  break;

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

case UNITS        :

  units_tbc = 0;

  if (i == THESIS) {
    units_tbc = ss->thesis_units;
  }
  if (i == GRAD_H_UNITS) {
    encountered_units_arranged = FALSE;
    for (j = 0 ; j < (ss->taken)->count ; j++) {
      cd = CrseDesc_search ((ss->taken)->members[j], cds, st->listed_count);
      if ((cd->h_level != H_LEVEL_NONE) &&
    (cd->h_level != H_LEVEL_EXCEPT_2_6_8_12_13_16_18_22) &&
    (cd->h_level != H_LEVEL_EXCEPT_2_6_16_18_22)) {
        if (cd->units_arranged) encountered_units_arranged = TRUE;
        else units_tbc += cd->total_units;
      }}
  }
  if (i == ED_POINTS) {
    for (j = 0 ; j < (ss->taken)->count ; j++)
      if ((t = CrseSet_index(EDPointsSubjects, (ss->taken)->members[j])) > -1)
        units_tbc += EDPoints[t];
  }
  if (Units_satisfied (ReqCat[i].units, units_tbc, TRUE, TRUE, env, DP_TAB2))
    ReqCatSatisfied[i] = TRUE;
  if (((i == GRAD_H_UNITS) || (i == UNRESTRICTED_ELECTIVES)) && encountered_units_arranged) {
    Display_psn (env, "");
    Display_psx (env, "'Units arranged' subjects were not counted.", DP_TAB2);
  }

  break;

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

  default         : Display_psx (env, "*****ERROR*****", DP_TAB2) ; break;
```

```
    }
    Display_psn  (env, "");

  }
  /*** ??? The Verbatim Mode does not return anything in *ar. ***/
}

/***** Silent Mode ************************************************************************************/

else {

  /***** Initialize *************************************************************************************/

  for (i = 0 ; i < NO_REQ_CATS ; i++) ReqCatSatisfied[i] = FALSE;

  for (i = 0 ; i < 7 ; i++)
    EC_seen[i] = (EngConcAuditState *) calloc (1, sizeof(EngConcAuditState));

  for (i = 0 ; i < ss->taken_all->count ; i++) {
    c = ss->taken_all->members[i];
    if ((cd = CrseDesc_search (c, cds, st->listed_count)) == NULL) printf ("dp_GIR:audit : Could not find %s.\n", c);
    else {
      if (cd->tag_first > 0) {
        considering_cds[considering_count++] = cd;
        if ((cd->h_level != H_LEVEL_NONE) &&
            (cd->h_level != H_LEVEL_EXCEPT_2_6_8_12_13_16_18_22) &&
            (cd->h_level != H_LEVEL_EXCEPT_2_6_16_18_22)) {
          if (cd->units_arranged) GHU_encountered_units_arranged = TRUE;
          else                    GHU_earned += cd->total_units;
        }
        EDP_earned += cd->tag_third;
      }}}
  CrseDesc_sort_on_tag (considering_cds, considering_count);

  /***** Let's get the easy ones first. *****************************************************************/

  if (ss->thesis_units >= 24) {
    ReqCatSatisfied[THESIS] = TRUE;
    satisfied_count++;
  }

  if (GHU_earned >= 42) {
    ReqCatSatisfied[GRAD_H_UNITS] = TRUE;
    satisfied_count++;
  }

  if (EDP_earned >= 48) {
    ReqCatSatisfied[ED_POINTS] = TRUE;
    satisfied_count++;
  }

  j = 0;
  cd = considering_cds[j];

  while (TRUE) {

    if (j == considering_count) break;

    /***** Required **************************************************************************************/

    if (dfs[DP_VI_P_TRACE_INDEX]) printf("dp_VI_P : Auditing the Required...\n");
    while ((cd->tag_first & VI_P_T1_REQUIRED) == VI_P_T1_REQUIRED) {
      if (!ReqCatSatisfied[REQUIRED]) {
        if (++Required_count == 5) {
          ReqCatSatisfied[REQUIRED] = TRUE;
          satisfied_count++;
        }
        CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
        (*ar)->appl_units += cd->total_units;
      }
      cd = considering_cds[++j];
      if (j == considering_count) break;
    }
    if (j == considering_count) break;

    /***** RE_MATH ***************************************************************************************/
    if (dfs[DP_VI_P_TRACE_INDEX]) printf("dp_VI_P : Auditing the Restricted Math...\n");
    while ((cd->tag_first & VI_P_T1_RE_MATH) == VI_P_T1_RE_MATH) {
      if (!REMath_required_satisfied) {
        if ((cd->tag_second & VI_P_T2_RE_MATH_REQUIRED) == VI_P_T2_RE_MATH_REQUIRED) {
          REMath_required_satisfied = TRUE;
          if (++REMath_count == 3) REMath_count_satisfied = TRUE;
          CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
          (*ar)->appl_units += cd->total_units;
          cd = considering_cds[++j];
          if (j == considering_count) break;
          continue;
        }}
      if (!REMath_count_satisfied) {
        if (++REMath_count == 3) REMath_count_satisfied = TRUE;
        CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
        (*ar)->appl_units += cd->total_units;
      }
      cd = considering_cds[++j];
      if (j == considering_count) break;
    }
    if (j == considering_count) break;
    if (REMath_required_satisfied & REMath_count_satisfied) {
      ReqCatSatisfied[RE_MATH] = TRUE;
      satisfied_count++;
    }

    /***** RE_LAB *********************************************************************************************/

    if (dfs[DP_VI_P_TRACE_INDEX]) printf("dp_VI_P : Auditing the Lab...\n");
    while ((cd->tag_first & VI_P_T1_LAB) == VI_P_T1_LAB) {
      if (!ReqCatSatisfied[RE_LAB]) {
        ReqCatSatisfied[RE_LAB] = TRUE;
        satisfied_count++;
        CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
        (*ar)->appl_units += cd->total_units;
      }
      cd = considering_cds[++j];
      if (j == considering_count) break;
    }
    if (j == considering_count) break;

    /***** EC *************************************************************************************************/

    if (dfs[DP_VI_P_TRACE_INDEX]) printf("dp_VI_P : Auditing the EC Headers...\n");
    EC_last_seen_code  = 1;
    EC_last_seen_index = 0;
    while ((cd->tag_first & VI_P_T1_EC_HEADER) == VI_P_T1_EC_HEADER) {
      while (TRUE) {
        if ((cd->tag_second & EC_last_seen_code) == EC_last_seen_code) break;
        EC_last_seen_code <<= 1;
        EC_last_seen_index++;
      }
      EC_seen[EC_last_seen_index]->header_seen = TRUE;
      EC_seen[EC_last_seen_index]->members[EC_seen[EC_last_seen_index]->count++] = cd;
      cd = considering_cds[++j];
      if (j == considering_count) break;
    }
    if (j == considering_count) break;

    if (dfs[DP_VI_P_TRACE_INDEX]) printf("dp_VI_P : Auditing the EC NonHeaders...\n");
    EC_last_seen_code  = 1;
    EC_last_seen_index = 0;
    while ((cd->tag_first & VI_P_T1_EC_NONHEADER) == VI_P_T1_EC_NONHEADER) {
      while (TRUE) {
        if ((cd->tag_second & EC_last_seen_code) == EC_last_seen_code) break;
        EC_last_seen_code <<= 1;
        EC_last_seen_index++;
      }
      EC_seen[EC_last_seen_index]->members[EC_seen[EC_last_seen_index]->count++] = cd;
      cd = considering_cds[++j];
      if (j == considering_count) break;
    }
    if (j == considering_count) {
      break;
    }
    break;
  }
```

```
    /***** The above code categorized all the ECs, so now let's check to see if we have fulfilled anything. ***********/

    for (i = 0 ; i < 7 ; i++) {
      if (EC_seen[i]->header_seen) {
        if (EC_seen[i]->count < 3) EC_seen[i]->dfc = 3 - EC_seen[i]->count;
        else                       EC_seen[i]->dfc = 0;
      } else {
        if (EC_seen[i]->count < 3) EC_seen[i]->dfc = 3 - EC_seen[i]->count;
        else                       EC_seen[i]->dfc = 1;
      }}

    if (dfs[DP_VI_P_TRACE_INDEX]) printf("dp_VI_P : Sorting the EC_seen...\n");
    for (j = 1 ; j < 7 ; j++) {
      key = EC_seen[j];
      for (k = j - 1 ; k >= 0 ; k--) {
        if (EC_seen[k]->dfc <= key->dfc) break;
        else EC_seen[k+1] = EC_seen[k];
      }
      EC_seen[k+1] = key;
    }

    if (dfs[DP_VI_P_TRACE_INDEX]) {
      for (j = 0 ; j < 7 ; j++) {
        printf("%s : ", ECNames[j]);
        for (k = 0 ; k < EC_seen[j]->count ; k++) {
          cd = EC_seen[j]->members[k];
          printf("%s ", cd->number->members[0]->number);
        }
        printf("\n");
      }
    }

    if (dfs[DP_VI_P_TRACE_INDEX]) printf("dp_VI_P : Checking for completion of categories...\n");

    for (j = 0 ; j < 7 ; j++) {
      if (!ReqCatSatisfied[RE_EC_LARGE]) {
        if (EC_seen[j]->dfc == 0) {
          ReqCatSatisfied[RE_EC_LARGE] = TRUE;
          satisfied_count++;
        }}
      if (!ReqCatSatisfied[RE_EC_SMALL]) {
        if (EC_seen[j]->dfc <= 1) {
          if (++ECSmallCats_count == 2) {
            ReqCatSatisfied[RE_EC_SMALL] = TRUE;
            satisfied_count++;
          }}}
      if ((EC_all_count += EC_seen[j]->count) > 9) break;
    }
    if (EC_all_count > 9) EC_all_count = 9;

    for (j = 0 ; j < 7 ; j++) {
      for (k = 0 ; k < EC_seen[j]->count ; k++) {
        cd = EC_seen[j]->members[k];
        CrseSet_insert ((*ar)->applicable, cd->number->members[0]->number);
        (*ar)->appl_units += cd->total_units;
        if (--EC_all_count == 0) break;
      }
      if (EC_all_count == 0) break;
    }

    /***** Fill in AuditResults ****************************************************************************************/

    (*ar)->completed     = Strings_create (satisfied_count);
    (*ar)->not_completed = Strings_create (NO_REQ_CATS - satisfied_count);

    for (i = 0 ; i < NO_REQ_CATS ; i++) {
      if (ReqCatSatisfied[i]) Strings_insert ((*ar)->completed, ReqCatStr[i]);
      else                    Strings_insert ((*ar)->not_completed, ReqCatStr[i]);
    }
  }

  if (flag == FLAG_DISPLAY_AUDIT_RESULTS) {
    total_units = CrseSet_get_units (ss->taken, cds, st->listed_count, &encountered_units_arranged);
    total_units += ss->other_units;
    total_units += ss->thesis_units;
    total_units -= CrseSet_get_units (gir_audit->applicable,
                          cds, st->listed_count, &encountered_units_arranged);
    Display_psn (env, "");
    Display_ps  (env, "Remaining to be completed in GIR");
    cpx1 = env->posx + 30;
    Display_psx (env, ": ", cpx1);
    cpx2 = env->posx;
    Strings_unparse_X (env, gir_audit->not_completed, cpx2);
    Display_psn (env, "");
    Display_ps  (env, "Not audited in GIR");
    Display_psx (env, ": ", cpx1);
    Strings_unparse_X (env, gir_audit->not_audited, cpx2);
    Display_psn (env, "");
    Display_ps  (env, "Remaining to be completed in VI-P");
    Display_psx (env, ": ", cpx1);
    env->posx = cpx2;
    for (i = 0 ; i < NO_REQ_CATS ; i++) {
      if (!(ReqCatSatisfied[i]) && (ReqCatType[i] != UNDEFINED)) {
        sprintf(buffer, "%s; ", ReqCatStr[i]);
        if (Display_ps (env, buffer) == ERROR_OUT_OF_BOUNDS) {
          Display_psn (env, "");
          Display_psx (env, buffer, cpx2);
        }}}
    Display_psn (env, "");
    Display_psn (env, "");
    Display_ps  (env, "Total units completed beyond GIRs");
    Display_psx (env, ": ", cpx1);
    sprintf(buffer, "%d", total_units);
    Display_psnx (env, buffer, cpx2);
    if (encountered_units_arranged) Display_psx (env, "'Units arranged' subjects were not counted.", cpx2);
  }

  CrseSet_free     (none_ue);
  CrseSet_free     (ue);
  CrseSet_free     (considering);
  AuditResult_free (gir_audit);

  return (ERROR_NONE);
}

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

III.C. main.c

```
/***********************************************************************************************************************
 ***** General Info ****************************************************************************************************
 ***********************************************************************************************************************
 *****
 ***** File          : main.c
 *****
 ***** Author        : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *****                                MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *****
 ***** Background    : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *****                 designed and implemented in the spring of 1995 as the author's Master of Engineering thesis,
 *****                 under the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.
 *****
 *****                 Complete description of AAA ???
 *****
 ***** Compiling Env : gcc -ansi -pedantic -c main.c
 *****
 ***** Code Status                   : 1.0.0 WIP
 ***** Last Modification Date & Time : April 27, 1995
 *****
 ***********************************************************************************************************************
 ***********************************************************************************************************************
 ***********************************************************************************************************************/

/***********************************************************************************************************************
 ***** Header Files ****************************************************************************************************
 ***********************************************************************************************************************/

#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"                      /* MSG_ */
#include "primitive_datatype_reps_constants.h"
#include "datatype_reps_constants.h"        /* FALL */
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "prototypes.h"
#include "tables.h"                         /* global tables - This *must* be included here */

/*******************************************************************************************************
 ***** Body ********************************************************************************************
 *******************************************************************************************************/

main ()
{
  /****************************************************************************************************
   **** Local Variables *******************************************************************************
   ****************************************************************************************************/

  /***** The Essence *****/

  Essence essence;

  /***** Variables pertaining to course descriptions. *****/

  char          ** crse_desc_filenames    = NULL;  /* names of course description files  */
  VeryShortNum     num_of_crse_desc_files = 0;     /* number of course description files */
  CrseDesc      ** crse_descs         = NULL;  /* course descriptions           */
  GIR              gir;

  /***** Variables pertaining to other parsed information. *****/

  AcadProgs   acad_progs;
  StudStatus  stud_stat;
  StudPrefs   stud_prefs;

  AuditResult * ar = NULL;

  /***** Variables pertaining to generation of possible courses. *****/

  Stats     stats;

  RankedCrse * ranked = NULL;

  /***** Variables pertaining to the display. *****/

  Env     env;
  Buttons curr_buttons;

  /***** Variables pertaining to the driving of the user interface. *****/

  int     option       = 0;
  Boolean quit_me_baby = FALSE;
  Boolean quit_audit   = FALSE;

  /***** Misc *****/

  int       status = 0;  /* status */
  int       i      = 0;  /* lcv    */

  /****************************************************************************************************
   ***** Function Body ********************************************************************************
   ****************************************************************************************************/

  /***** Initialize ***********************************************************************************/

  if (print_welcome_message() != ERROR_NONE) handle_error (status, "executing print_welcome_message");
  if (read_debugging_flags () != ERROR_NONE) handle_error (status, "executing read_debugging_flags");

  StudStatus_initialize (&stud_stat);
  StudPrefs_initialize  (&stud_prefs);
  GIR_initialize        (&gir);
```

```
/***** Determine the names of the files containing course descriptions. *******************************************/

if (dfs[MAIN_TRACE]) printf("Determining filenames of course descriptions...\n");
if ((status = determine_filenames(&crse_desc_filenames, &num_of_crse_desc_files)) != ERROR_NONE)
  handle_error (status, "executing determine_filenames");
if (dfs[MAIN_TRACE]) printf("\tDone.\n");

/***** Parse those files to create a set of CrseDesc structures. ***************************************************/

if (dfs[MAIN_TRACE]) printf("Parsing course description files...\n");
if ((status = CrseDesc_parse(crse_desc_filenames, num_of_crse_desc_files, &crse_descs, &(stats.listed_count), &gir))
    != ERROR_NONE) handle_error (status, "executing CrseDesc_parse");
if (dfs[MAIN_TRACE]) printf("\tDone.\n");

/***** Compute remaining members of gir. ***************************************************************************/

for (i = 0 ; i < NO_HASS_FULFILLMENT_CODES ; i++)
  CrseSet_append (gir.all_hass, gir.categories[HASSFulfillmentCodes[i]]);
for (i = 0 ; i < NO_PHASE_I_FULFILLMENT_CODES ; i++)
  CrseSet_append (gir.all_phaseI, gir.categories[PhaseIFulfillmentCodes[i]]);
for (i = 0 ; i < NO_PHASE_II_FULFILLMENT_CODES ; i++)
  CrseSet_append(gir.all_phaseII, gir.categories[PhaseIIFulfillmentCodes[i]]);

/***** Sort course descriptions and gir members. *******************************************************************/

if (dfs[MAIN_TRACE]) printf("Sorting course descriptions and GIR members...\n");
CrseDesc_sort (crse_descs, stats.listed_count);
GIR_sort (&gir);
if (dfs[MAIN_TRACE]) printf("\tDone.\n");

/***** Build a hash table for CrseDescs. ***************************************************************************/

if (dfs[MAIN_TRACE]) printf("Building a hash table for course descriptions...\n");
CrseDesc_build_ht (crse_descs, stats.listed_count);
if (dfs[MAIN_TRACE]) printf("\tDone.\n");

/***** Parse academic programs. ************************************************************************************/

if (dfs[MAIN_TRACE]) printf("Parsing academic programs...\n");
if ((status == AcadProgs_parse (HOMEDIR ACADEMIC_PROGRAMS_FILE, &acad_progs)) != ERROR_NONE)
  handle_error (status, "executing AcadProgs_parse");
if (dfs[MAIN_TRACE]) printf("\tDone.\n");

/***** Parse the student status file. ******************************************************************************/

if (dfs[MAIN_TRACE]) printf("Parsing student status file...\n");
status = StudStatus_parse (crse_descs, &stats, HOMEDIR STUDENT_STATUS_FILE, &stud_stat);
if (status != ERROR_NONE) handle_error (status, "executing StudStatus_parse");
if (dfs[MAIN_TRACE]) printf("\tDone.\n");

/***** Parse the student prefs file. *******************************************************************************/

if (dfs[MAIN_TRACE]) printf("Parsing student prefs file...\n");
status = StudPrefs_parse (HOMEDIR STUDENT_PREFS_FILE, &acad_progs, &stud_prefs);
if (status != ERROR_NONE) handle_error (status, "executing StudPrefs_parse");
if (dfs[MAIN_TRACE]) printf("\tDone.\n");

/***** Pop up the beautiful UI. ************************************************************************************/

if (dfs[MAIN_TRACE]) printf("Creating and setting up window...\n");
Display_create     (&env);
Display_initialize (&env);
show_startup_info  (&env);
if (dfs[MAIN_TRACE]) printf("\tDone.\n");

/***** Capture the essence. ****************************************************************************************/

essence.env = &env;
essence.cds = crse_descs;
essence.ap  = &acad_progs;
essence.ss  = &stud_stat;
essence.sp  = &stud_prefs;
essence.st  = &stats;
essence.gir = &gir;

/***** Initialize degree program modules. **************************************************************************/
```

```
if (dfs[MAIN_TRACE]) printf("Initializing degree program GIR...\n");
status = dp_GIR(MSG_INITIALIZE, FLAG_NONE, &essence, NULL);
if (status != ERROR_NONE) handle_error (status, "sending MSG_INITIALIZE to dp_GIR");
if (dfs[MAIN_TRACE]) printf("\tDone.\n");
if (dfs[MAIN_TRACE]) printf("Initializing degree program VI-P...\n");
status = dp_VI_P(MSG_INITIALIZE, FLAG_NONE, &essence, NULL);
if (status != ERROR_NONE) handle_error (status, "sending MSG_INITIALIZE to dp_VI_P");
if (dfs[MAIN_TRACE]) printf("\tDone.\n");

/***** Calculate inverse prerequisite counts. *********************************************************************/

if (dfs[MAIN_TRACE]) printf("Determining inverse prerequisite counts...\n");
CrseDesc_determine_inv_prereqs (&essence);
if (dfs[MAIN_TRACE]) printf("\tDone.\n");

/***** Main Loop ********************************************************************************************************/

Buttons_display    (&env, MainScreenOptions, &curr_buttons);

while (!quit_me_baby) {
  option = Buttons_get_press (&env, &curr_buttons);
  switch (option) {

  case MS_OPT_PARSE_SSF :
    Display_clear_text_area (&env);
    Display_print_string (&env, "Parsing the student status file ");
    Display_print_string (&env, STUDENT_STATUS_FILE "... ");
    status = StudStatus_parse (crse_descs, &stats, HOMEDIR STUDENT_STATUS_FILE, &stud_stat);
    Display_print_string_and_nl (&env, "Done.");
    break;

  case MS_OPT_PARSE_SPF :
    Display_clear_text_area (&env);
    Display_print_string (&env, "Parsing the student preferences file ");
    Display_print_string (&env, STUDENT_PREFS_FILE "... ");
    status = StudPrefs_parse (HOMEDIR STUDENT_PREFS_FILE, &acad_progs, &stud_prefs);
    Display_print_string_and_nl (&env, "Done.");
    break;

  case MS_OPT_GEN_SCH :
    Display_clear_text_area (&env);
    RankedCrse_initialize (&ranked, &essence);
    RankedCrse_sort (ranked, stats.takable_count, INV_PREREQS);
    for (i = 0 ; i < 10 ; i++) RankedCrse_unparse (&ranked[i]);
    RankedCrse_determine_ff (ranked, &essence);
    break;

  case MS_OPT_SHOW_GIR :
    Display_clear_text_area (&env);
    status = dp_GIR(MSG_UNPARSE_REQUIRED_X, FLAG_NONE, &essence, NULL);
    if (status != ERROR_NONE) handle_error (status, "sending MSG_UNPARSE_REQUIRED_X to dp_GIR");
    break;

  case MS_OPT_SHOW_VI_P :
    Display_clear_text_area (&env);
    status = dp_VI_P(MSG_UNPARSE_REQUIRED_X, FLAG_NONE, &essence, NULL);
    if (status != ERROR_NONE) handle_error (status, "sending MSG_UNPARSE_REQUIRED_X to dp_VI_P");
    break;

  case MS_OPT_AUDIT_GIR :
    Display_clear_text_area (&env);
    status = dp_GIR(MSG_AUDIT, FLAG_DISPLAY_AUDIT_RESULTS, &essence, &ar);
    if (status != ERROR_NONE) handle_error (status, "sending MSG_AUDIT to dp_GIR");
    AuditResult_free    (ar);
    break;

  case MS_OPT_AUDIT_VI_P :
    Display_clear_text_area (&env);
    status = dp_VI_P(MSG_AUDIT, FLAG_DISPLAY_AUDIT_RESULTS, &essence, &ar);
    if (status != ERROR_NONE) handle_error (status, "sending MSG_AUDIT to dp_VI_P");
    AuditResult_free    (ar);
    break;

  case MS_OPT_QUIT :
    quit_me_baby = TRUE;
    break;
    case MS_OPT_DISP_SS :
      Display_clear_text_area (&env);
      StudStatus_unparse_X (&env, &stud_stat);
      break;

    case MS_OPT_DISP_SP:
      Display_clear_text_area (&env);
      StudPrefs_unparse_X (&env, &stud_prefs, &acad_progs);
      break;

    case MS_OPT_DISP_INFO :
      read_debugging_flags ();
      Display_clear_text_area (&env);
      Display_print_string (&env, "Generating statistics...");
      RankedCrse_initialize (&ranked, &essence);
      Display_print_string_and_nl (&env, " Done.");
      Display_print_string_and_nl (&env, "");
      show_interesting_info (&env, &stud_prefs, &stats);
      break;
    }
  }

  printf("Thank you for using the Academic Advisor Assistant.  Have a nice day!\n\n");

  /***** ??? close up better *****/

  return(0);
}

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

III.D. proc.c

```
/**********************************************************************************************************************
 ***** General Info ***************************************************************************************************
 **********************************************************************************************************************
 *****
 ***** File         : proc.c
 *****
 ***** Author       : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *****                              MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *****
 ***** Background   : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *****                designed and implemented in the spring of 1995 as the author's Master of Engineering thesis,
 *****                under the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.
 *****                See main.c for a more complete description of AAA.
 *****
 *****                General info???
 *****
 *****                     determine_filenames
 *****                     handle_error
 *****                     print_welcome_message
 *****                     read_debugging_flags
 *****
 ***** Compiling Env : gcc -ansi -pedantic -c proc.c
 *****
 ***** Code Status                  : 0.5.0 WIP
 ***** Last Modification Date & Time : April 20, 1995
 *****
 **********************************************************************************************************************
 ***** Functions Info *************************************************************************************************
 **********************************************************************************************************************
 *****
 ***** int determine_filenames (char *** crse_desc_filenames, VeryShortNum * count)
 *****
 ***** Requires : - none
 ***** Modifies : - crse_desc_filenames, count
 ***** Effects  : - Parses the file DESC_FILES_LISTING in the CRSE_DESC_DIR directory by reading its lines into the
 *****                array of strings crse_desc_filenames.
 *****              - Sets count to the number of filenames.
 *****              - Returns any errors.  If none, returns ERROR_NONE.
 *****
 **********************************************************************************************************************
 *****
 ***** void handle_error (int error, char * msg)
 *****
 ***** Requires : - none
 ***** Modifies : - stderr
 ***** Effects  : - Prints a simple error message that includes both the text form of error and msg.
 *****            - Exits the program by returning -1.
 *****
 ************************************************************************************************************************
 *****
 ***** int print_welcome_message (void)
 *****
 ***** Requires : - none
 ***** Modifies : - stdout
 ***** Effects  : - Prints a welcome message that's both preceded and followed by a newline.
 *****            - Returns any errors.
 *****
 ************************************************************************************************************************
 *****
 ***** int read_debugging_flags (void)
 *****
 ***** Requires : - dfs is a globally visible array of Booleans.
 ***** Modifies : - dfs
 ***** Effects  : - Parses the debugging flags file, whose name is DEBUGGING_FLAGS_FILE and located in HOMEDIR, into
 *****              dfs as follows:
 *****
 *****                   an entry in the file : alpha beta  ===> dfs[alpha] = beta
 *****
 *****              Note that alpha is in hex, and that beta must be either TRUE or FALSE.
 *****            - Sets count to the number of debugging flags read.
 *****            - Returns any errors.
 *****
 ************************************************************************************************************************
 ************************************************************************************************************************
 ************************************************************************************************************************/

/************************************************************************************************************************
 ***** Header Files *****************************************************************************************************
 ************************************************************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"               /* HOMEDIR, MLO_FILENAME, etc */
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"           /* VeryShortNum               */
#include "prototypes.h"              /* function prototypes        */
#include "externs.h"                 /* dfs                        */

/************************************************************************************************************************
 ***** determine_filenames **********************************************************************************************
 ************************************************************************************************************************/

int determine_filenames (char *** crse_desc_filenames, VeryShortNum * count)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - crse_desc_filenames, count
   ***** Effects  : - Parses the file DESC_FILES_LISTING in the CRSE_DESC_DIR directory by reading its lines into the
   *****              array of strings crse_desc_filenames.
   *****            - Sets count to the number of filenames.
   *****            - Returns any errors.  If none, returns ERROR_NONE.
   *****
   ***** Note     : - The DESC_FILES_LISTING should be of the following form:
   *****
   *****                 file_name_1
   *****                 file_name_2
   *****                 ...
   *****                 file_name_n
   *****                 DESC_FILES_LISTING_END_MARKER
   *****
   *****              For example, it might appear as follows, with DESC_FILES_LISTING_END_MARKER = "END"
   *****
   *****                   Course_1
   *****                   Course_2
   *****                   END
   *****/

  FILE * listing_file                   = NULL;
  char   listing_file_name[MLO_FILENAME];
  char   one_filename[MLO_FILENAME];
  int    index                          = 0;

  if ((*crse_desc_filenames = (char **) malloc(MNO_CRSE_DESC_FILES * sizeof(char *))) == NULL)
    return (ERROR_ALLOC_FAIL);
  if (sprintf(listing_file_name, "%s%s%s", HOMEDIR, CRSE_DESC_DIR, DESC_FILES_LISTING) < 0)
    return (ERROR_SPRINTF_FAIL);
  if ((listing_file = fopen(listing_file_name, "r")) == NULL) return (ERROR_BAD_DESC_FILES_LISTING);

  while (fscanf(listing_file, "%s", one_filename) != EOF) {
    if (!strcmp(one_filename, DESC_FILES_LISTING_END_MARKER)) break;
    if ((*(*crse_desc_filenames + index) = (char *) malloc(sizeof(char) * (strlen(one_filename) + 1))) == NULL)
      return (ERROR_ALLOC_FAIL);
    strcpy(*(*crse_desc_filenames + index++), one_filename);
  }

  if ((*crse_desc_filenames = (char **) realloc(*crse_desc_filenames, index * sizeof(char *))) == NULL)
      return (ERROR_ALLOC_FAIL);

  *count = index;
  return (fclose (listing_file));
}

/*****************************************************************************************************************
 ***** handle_error **********************************************************************************************
 *****************************************************************************************************************/

void handle_error (int error, char * msg)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - stderr
   ***** Effects  : - Prints a simple error message that includes both the text form of error and msg.
   *****            - Exits the program by returning -1.
   *****/

  fprintf(stderr, "\nSorry! An error occurred : %s\n", ErrorStrings[error]);
  fprintf(stderr, "The error had something to do with %s.\n", msg);
  exit(EXIT_FAILURE);
}

void handle_warning (int error, char * msg)
{
  fprintf(stderr, "***** Warning %d\n", error);
  return;
}

/*****************************************************************************************************************
 ***** initialize_stats ******************************************************************************************
 *****************************************************************************************************************/

void initialize_stats (Stats * st)
{
  /*****
   ***** Requires : - st is a non-NULL Stats.
   ***** Modifies : - st
   ***** Effects  : - Sets all of st's members, except listed_count, to zero.
   *****/

  st->offered_count           = 0;
  st->undergrad_count         = 0;
  st->grad_count              = 0;
  st->considered_count        = 0;
  st->no_prereqs_count        = 0;
  st->prereqs_satisfied_count = 0;
  st->taken_already_count     = 0;
  st->takable_count           = 0;
}

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

```
 ***** print_welcome_message ****************************************************************************************
 *********************************************************************************************************************/

int print_welcome_message (void)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - stdout
   ***** Effects  : - Prints a welcome message that's both preceded and followed by a newline.
   *****            - Returns any errors.
   *****/

  int status = ERROR_NONE;

  status = printf("\nAcademic Advisor Assistant, %s\n", VERSION);
  status = printf("by Dae-Chul Sohn, MIT M.Eng. in EECS, May 1995\n\n");
  if (status < 0) return (ERROR_PRINTF_FAIL);
  else            return (ERROR_NONE);
}

/*********************************************************************************************************************
 ***** read_debugging_flags *****************************************************************************************
 *********************************************************************************************************************/

int read_debugging_flags (void)
{
  /*****
   ***** Requires : - dfs is a globally visible array of Booleans.
   ***** Modifies : - dfs
   ***** Effects  : - Parses the debugging flags file, whose name is DEBUGGING_FLAGS_FILE and located in HOMEDIR, into
   *****              dfs as follows:
   *****
   *****                   an entry in the file : alpha beta  ===> dfs[alpha] = beta
   *****
   *****              Note that alpha is in hex, and that beta must be either TRUE or FALSE.
   *****            - Sets count to the number of debugging flags read.
   *****            - Returns any errors.
   *****/

  FILE * fp     = NULL;
  char * fn     = NULL;
  int    index  = 0;
  int    flag   = 0;

  if ((fn = (char *) malloc((strlen(HOMEDIR) + strlen(DEBUGGING_FLAGS_FILE) + 1) * sizeof(char))) == NULL)
    return (ERROR_ALLOC_FAIL);
  if (sprintf(fn, "%s%s", HOMEDIR, DEBUGGING_FLAGS_FILE) < 0) return (ERROR_SPRINTF_FAIL);

  if ((fp = fopen(fn, "r")) == NULL) return (ERROR_FOPEN_FAIL);
  while ((fscanf(fp, "%x", &index)) != EOF) {
    if (fscanf(fp, "%d", &flag) == EOF) return (ERROR_BAD_DEBUGGING_FLAGS_FILE);
    dfs[index] = flag;
  }
  free (fn);
  return(fclose(fp));
}

/*********************************************************************************************************************
 ***** show_interesting_info ****************************************************************************************
 *********************************************************************************************************************/

void show_interesting_info (Env * env, StudPrefs * sp, Stats * st)
{
  char buffer[MLO_BUFFER];

  sprintf      (buffer, "%d", st->listed_count);
  Display_ps   (env, buffer);
  sprintf      (buffer, "courses are listed in the %s version of the Bulletin.", BULLETIN_VERSION);
  Display_psnx (env, buffer, TAB);

  sprintf      (buffer, "%d", st->offered_count);
  Display_psx  (env, buffer, TAB);
  Display_psnx (env, "of those are going to be offered next term.", 2*TAB);

  sprintf      (buffer, "%d", st->undergrad_count);
  Display_psx  (env, buffer, 2*TAB);
  Display_psnx (env, "of those are undergraduate courses.", 3*TAB);
  sprintf      (buffer, "%d", st->grad_count);
  Display_psx  (env, buffer, 2*TAB);
  Display_psnx (env, "of those are graduate courses.", 3*TAB);

  Display_psn (env, "");
  Display_psn (env, "As per your preferences:");
  if (sp->consider_undergrad) {
    if (sp->consider_grad)
      Display_psnx (env, "Both undergraduate and graduate courses were considered.", TAB);
    else
      Display_psnx (env, "Only undergraduate courses were considered.", TAB);
  } else {
    if (sp->consider_undergrad)
      Display_psnx (env, "Only graduate courses were considered.", TAB);
    else
      Display_psnx (env, "Neither undergraduate nor graduate courses were considered.", TAB);
  }
  if (sp->override_poi)
    Display_psnx (env, "'Permission of instructor' prerequisites were regarded as fulfilled.", TAB);
  else
    Display_psnx (env, "'Permission of instructor' prerequisites were regarded as unfulfilled.", TAB);

  Display_psn (env, "");
  sprintf     (buffer, "Of the %d courses considered, then:", st->considered_count);
  Display_psn (env, buffer);

  sprintf      (buffer, "%d", st->taken_already_count);
  Display_psx  (env, buffer, TAB);
  Display_psnx (env, "have been taken already by you.", 2*TAB);
  sprintf      (buffer, "%d", st->no_prereqs_count);
  Display_psx  (env, buffer, TAB);
  Display_psnx (env, "have no prerequisites.", 2*TAB);
  sprintf      (buffer, "%d", st->prereqs_satisfied_count);
  Display_psx  (env, buffer, TAB);
  Display_psnx (env, "have prerequisites satisfied by you.", 2*TAB);
}

/*************************************************************************************************************
 ***** show_startup_info ***************************************************************************************
 *************************************************************************************************************/

void show_startup_info (Env * env)
{
  Display_psn  (env, "Welcome to the Academic Advisor Assistant, Version 1.0.0.");
  Display_psn  (env, "This program was written by Dae-Chul Sohn, MIT M.Eng. in EECS, 1995.");
  Display_psn  (env, "This release was completed on May 15, 1995.");
  Display_psn  (env, "");
  Display_psn  (env, "Please note the following restrictions in this release:");
  Display_psnx (env, "- AAA is hard-coded to handle VI-P majors only.", TAB);
  Display_psnx (env, "- AAA does *not* check for time conflicts.", TAB);
  Display_psnx (env, "- AAA does *not* audit minors.", TAB);
  Display_psnx (env, "- AAA does *not* audit concentrations.", TAB);
  Display_psn  (env, "");
  Display_psn  (env, "Please send bugs and comments to marlboro@mit.edu.");
  Display_psn  (env, "Thank you for using this program.");
}

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

III.E. util.c

```
/*************************************************************************************************************
 ***** General Info ********************************************************************************************
 *************************************************************************************************************
 *****
 ***** File          : util.c
 *****
 ***** Author        : Dae-Chul Sohn, MIT Bachelor of Science in Computer Science, '95
 *****                                MIT Master of Engineering in Electrical Engineering and Computer Science '95
 *****
 ***** Background    : This file contains a part of the C source code for the Academic Advisor Assistant (AAA) system,
 *****                 designed and implemented in the spring of 1995 as the author's Master of Engineering thesis,
 *****              under the supervision of Prof. Gill Pratt of the MIT Laboratory for Computer Science.
 *****              See main.c for a more complete description of AAA.
 *****
 *****              General info ???
 *****
 *****                  parse_term_offered
 *****
 *****                  unparse_GIR_fulfillment
 *****                  unparse_h_level
 *****                  unparse_pass_fail
 *****                  unparse_repeatable_for_credit
 *****                  unparse_subj_level
 *****                  unparse_units
 *****                  unparse_term_offered
 *****                  Boolean_unparse
 *****
 *****                  convert_to_upper
 *****                  get_mouse_event
 *****                  strptr_cmp
 *****                  wait_for_left_click
 *****
 ***** Compiling Env : gcc -ansi -pedantic -c util.c
 *****
 ***** Code Status                    : 0.5.0 WIP
 ***** Last Modification Date & Time : April 20, 1995
 *****
 *******************************************************************************************************
 ***** Parsing Utilities *******************************************************************************
 *******************************************************************************************************
 *****
 ***** VeryShortNum parse_term_offered (char * buffer, VeryShortNum b, VeryShortNum year, VeryShortNum * subject_level)
 *****
 ***** Requires : - buffer is a string of the form X (alpha+), where alpha+ means one or more occurrences of alpha,
 *****              and where X is one of {U, G} and alpha is one of {1, 2, IAP, S}.  Example: U (1, 2).
 *****              X is optional.
 ***** Modifies : - subject_level
 ***** Effects  : - Parses buffer and returns the results by returning a new b and setting subject_level.
 *****              If X is missing, then subject_level is left untouched.
 *****
 *******************************************************************************************************
 ***** Unparsing Utilities *****************************************************************************
 *******************************************************************************************************
 *****
 ***** ***** The Simple Ones ***************************************************************************
 *****
 ***** char * unparse_GIR_fulfillment       (CrseDesc * cd, Boolean should_return_result)
 ***** char * unparse_h_level               (CrseDesc * cd, Boolean should_return_result)
 ***** char * unparse_pass_fail             (CrseDesc * cd, Boolean should_return_result)
 ***** char * unparse_repeatable_for_credit (CrseDesc * cd, Boolean should_return_result)
 ***** char * unparse_subj_level            (CrseDesc * cd, Boolean should_return_result)
 ***** char * unparse_units                 (CrseDesc * cd, Boolean should_return_result)
 *****
 ***** ***** The Slightly Sophisticated ****************************************************************
 *****
 ***** char * unparse_term_offered (CrseDesc * cd, Boolean should_return_result)
 ***** char * Boolean_unparse      (Boolean     b, Boolean should_return_result)
 *****
 *******************************************************************************************************
 ***** Misc Utilities **********************************************************************************
 *******************************************************************************************************
 *****
 ***** void convert_to_upper (char * str)
 *****
 *******************************************************************************************************
 *****
 ***** void get_mouse_event (Env *, XButtonEvent *)
 *****
 *******************************************************************************************************
 *******************************************************************************************************
 *******************************************************************************************************/

/*******************************************************************************************************
 ***** Header Files ************************************************************************************
 *******************************************************************************************************/

#include <stdio.h>
#include <string.h>
```

```
#include <ctype.h>
#include <time.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "constants.h"                /* TERM_OFFERED_NOT_OFFERED_STR */
#include "primitive_datatype_reps_constants.h"
#include "datatype_reps_constants.h"  /* TERM_OFFERED_                */
#include "table_constants.h"
#include "error_and_trace.h"
#include "primitive_datatype_reps.h"
#include "datatype_reps.h"
#include "prototypes.h"
#include "externs.h"                          /* GIRFulfillmentStrings, etc */

/*****************************************************************************************************
 ***** Parsing Utilities *****************************************************************************
 *****************************************************************************************************/

/***** parse_term_offered ***************************************************************************/

VeryShortNum parse_term_offered (char * buffer, VeryShortNum b, VeryShortNum year, VeryShortNum * subject_level)
{
  /*****
   ***** Requires : - buffer is a string of the form X (alpha+), where alpha+ means one or more occurrences of alpha,
   *****              and where X is one of (U, G) and alpha is one of (1, 2, IAP, S).  Example: U (1, 2).
   *****              X is optional.
   ***** Modifies : - subject_level
   ***** Effects  : - Parses buffer and returns the results by returning a new b and setting subject_level.
   *****              If X is missing, then subject_level is left untouched.
   *****/

  if (!strcmp(buffer, TERM_OFFERED_NOT_OFFERED_STR)) return (b);

  if (strchr(buffer,'U') != NULL) *subject_level = SUBJECT_LEVEL_U;
  if (strchr(buffer,'G') != NULL) *subject_level = SUBJECT_LEVEL_G;
  if (strchr(buffer,'1') != NULL) {
    if ((year & THIS_YEAR) == THIS_YEAR) b |= TERM_OFFERED_THIS_FIRST;
    if ((year & NEXT_YEAR) == NEXT_YEAR) b |= TERM_OFFERED_NEXT_FIRST;
  }
  if (strchr(buffer,'2') != NULL) {
    if ((year & THIS_YEAR) == THIS_YEAR) b |= TERM_OFFERED_THIS_SECOND;
    if ((year & NEXT_YEAR) == NEXT_YEAR) b |= TERM_OFFERED_NEXT_SECOND;
  }
  if (strchr(buffer,'I') != NULL) {
    if ((year & THIS_YEAR) == THIS_YEAR) b |= TERM_OFFERED_THIS_IAP;
    if ((year & NEXT_YEAR) == NEXT_YEAR) b |= TERM_OFFERED_NEXT_IAP;
  }
  if (strchr(buffer,'S') != NULL) {
    if ((year & THIS_YEAR) == THIS_YEAR) b |= TERM_OFFERED_THIS_SUMMER;
    if ((year & NEXT_YEAR) == NEXT_YEAR) b |= TERM_OFFERED_NEXT_SUMMER;
  }
  return (b);
}

/*****************************************************************************************************
 ***** Unparsing Utilities ***************************************************************************
 *****************************************************************************************************/

/***** unparse_GIR_fulfillment **********************************************************************/

char * unparse_GIR_fulfillment (CrseDesc * cd, Boolean should_return_result)
{
  char * c = NULL;

  if (should_return_result) {
    if ((c = (char *) malloc (sizeof(char) * (strlen(GIRFulfillmentStrings[cd->GIR_fulfillment]) + 1))) == NULL)
      handle_error (ERROR_ALLOC_FAIL, "executing unparse_GIR_fulfillment");
    strcpy(c, GIRFulfillmentStrings[cd->GIR_fulfillment]);
    return (c);
  } else {
    if (printf("%s",GIRFulfillmentStrings[cd->GIR_fulfillment]) < 0)
      handle_error(ERROR_PRINTF_FAIL, "unparse_GIR_fulfillment");
    return (NULL);
  }
}

/***** unparse_h_level ******************************************************************************/
```

```
char * unparse_h_level (CrseDesc * cd, Boolean should_return_result)
{
  char * c = NULL;

  if (should_return_result) {
    if ((c = (char *) malloc (sizeof(char) * (strlen(HLevelExceptStrings[cd->h_level]) + 1))) == NULL)
      handle_error (ERROR_ALLOC_FAIL, "executing unparse_h_level");
    strcpy(c, HLevelExceptStrings[cd->h_level]);
    return (c);
  } else {
    if (printf("%s",HLevelExceptStrings[cd->h_level]) < 0)
      handle_error(ERROR_PRINTF_FAIL, "executing unparse_h_level");
    return (NULL);
  }
}

/***** unparse_pass_fail *********************************************************************************/

char * unparse_pass_fail (CrseDesc * cd, Boolean should_return_result)
{
  if (should_return_result) return (Boolean_unparse(cd->pass_fail,TRUE));
  else {
    if (printf("%s",Boolean_unparse(cd->pass_fail,TRUE)))
      handle_error(ERROR_PRINTF_FAIL,"executing unparse_pass_fail");
    return (NULL);
  }
}

/***** unparse_repeatable_for_credit *********************************************************************/

char * unparse_repeatable_for_credit (CrseDesc * cd, Boolean should_return_result)
{
  if (should_return_result) return (Boolean_unparse(cd->repeatable_for_credit,TRUE));
  else {
    if (printf("%s",Boolean_unparse(cd->repeatable_for_credit,TRUE)))
      handle_error(ERROR_PRINTF_FAIL,"executing unparse_pass_fail");
    return (NULL);
  }
}

/***** unparse_subj_level ********************************************************************************/

char * unparse_subj_level (CrseDesc * cd, Boolean should_return_result)
{
  int status = 0;

  if (should_return_result) {
    switch (cd->subject_level) {
    case SUBJECT_LEVEL_U : return ("Undergraduate");
    case SUBJECT_LEVEL_G : return ("Graduate");
    }
  } else {
    switch (cd->subject_level) {
    case SUBJECT_LEVEL_U : status = printf("Undergraduate"); break;
    case SUBJECT_LEVEL_G : status = printf("Graduate")     ; break;
    }
    if (status < 0) handle_error(ERROR_PRINTF_FAIL, "executing unparse_subj_level");
    return (NULL);
  }
}

/***** unparse_units *************************************************************************************/

char * unparse_units (CrseDesc * cd, Boolean should_return_result)
{
  char * c      = NULL;
  int    status = 0;

  if (should_return_result) {
    if ((c = (char *) malloc (sizeof(char) * MLO_UNITS)) == NULL)
      handle_error (ERROR_ALLOC_FAIL, "executing unparse_units");
    if (cd->units_arranged) status = sprintf(c, "Arranged");
    else                    status = sprintf(c, "%d-%d-%d", cd->class_hours, cd->lab_hours, cd->prep_hours);
    if (status < 0) handle_error(ERROR_SPRINTF_FAIL, "executing unparse_units");
    else return (c);
  } else {
    if (cd->units_arranged) status = printf("Arranged");
    else                    status = printf("%d-%d-%d", cd->class_hours, cd->lab_hours, cd->prep_hours);
    if (status < 0) handle_error(ERROR_PRINTF_FAIL, "executing unparse_units");
    return (NULL);
  }
}

/***** unparse_term_offered *************************************************************************************/

char * unparse_term_offered (CrseDesc * cd, Boolean should_return_result)
{
  char        * c      = NULL;
  OneNumCode    to     = 0;
  int           status = 0;

  to = cd->term_offered;
  if (should_return_result) {
    if ((c = (char *) malloc (sizeof(char) * MLO_TERM_OFFERED)) == NULL)
      handle_error (ERROR_ALLOC_FAIL, "executing unparse_term_offered");
    strcpy(c, "This Year : ( ");
    if ((to & TERM_OFFERED_THIS_FIRST)  == TERM_OFFERED_THIS_FIRST)  strcat(c, "1 "  );
    if ((to & TERM_OFFERED_THIS_IAP)    == TERM_OFFERED_THIS_IAP)    strcat(c, "IAP ");
    if ((to & TERM_OFFERED_THIS_SECOND) == TERM_OFFERED_THIS_SECOND) strcat(c, "2 ");
    if ((to & TERM_OFFERED_THIS_SUMMER) == TERM_OFFERED_THIS_SUMMER) strcat(c, "SUMMER ");
    strcat(c, ")   Next Year : ( ");
    if ((to & TERM_OFFERED_NEXT_FIRST)  == TERM_OFFERED_NEXT_FIRST)  strcat(c, "1 "  );
    if ((to & TERM_OFFERED_NEXT_IAP)    == TERM_OFFERED_NEXT_IAP)    strcat(c, "IAP ");
    if ((to & TERM_OFFERED_NEXT_SECOND) == TERM_OFFERED_NEXT_SECOND) strcat(c, "2 ");
    if ((to & TERM_OFFERED_NEXT_SUMMER) == TERM_OFFERED_NEXT_SUMMER) strcat(c, "SUMMER ");
    strcat(c, ")");
    return (c);
  } else {
    status = printf("This Year : ( ");
    if ((to & TERM_OFFERED_THIS_FIRST)  == TERM_OFFERED_THIS_FIRST)  status = printf("1 "  );
    if ((to & TERM_OFFERED_THIS_IAP)    == TERM_OFFERED_THIS_IAP)    status = printf("IAP ");
    if ((to & TERM_OFFERED_THIS_SECOND) == TERM_OFFERED_THIS_SECOND) status = printf("2 ");
    if ((to & TERM_OFFERED_THIS_SUMMER) == TERM_OFFERED_THIS_SUMMER) status = printf("SUMMER ");
    status = printf(")   Next Year : ( ");
    if ((to & TERM_OFFERED_NEXT_FIRST)  == TERM_OFFERED_NEXT_FIRST)  status = printf("1 "  );
    if ((to & TERM_OFFERED_NEXT_IAP)    == TERM_OFFERED_NEXT_IAP)    status = printf("IAP ");
    if ((to & TERM_OFFERED_NEXT_SECOND) == TERM_OFFERED_NEXT_SECOND) status = printf("2 ");
    if ((to & TERM_OFFERED_NEXT_SUMMER) == TERM_OFFERED_NEXT_SUMMER) status = printf("SUMMER ");
    status = printf(")");
    if (status < 0) handle_error(ERROR_PRINTF_FAIL, "executing unparse_term_offered");
    return (NULL);
  }
}

/***** Boolean_unparse ******************************************************************************************/

char * Boolean_unparse (Boolean b, Boolean should_return_result)
{
  /*****
   ***** Requires : - none
   ***** Modifies : - stdout
   ***** Effects  : - If should_return_result is TRUE, returns a string representation of b.
   *****            - Else, prints a string representation of b to stdout.  Returns NULL.
   *****            - Exits via handle_error if error is incurred.
   *****/

  int status = 0;

  if (should_return_result) {
    if (b) return ("TRUE");
    else   return ("FALSE");
  } else {
    if (b) status = printf("TRUE");
    else   status = printf("FALSE");
    if (status < 0) handle_error(ERROR_PRINTF_FAIL, "executing Boolean_unparse");
    return (NULL);
  }
}

/**************************************************************************************************************
 ***** Misc Utilities *****************************************************************************************
 *************************************************************************************************************/
```

```
/***** convert_to_upper *********************************************************************************/

void convert_to_upper (char * str)
{
  char c;
  while ((c = *(str++)) != '\0')
    if islower(c) *(str - 1) = toupper(c);
}

/***** get_mouse_event **********************************************************************************/

void get_mouse_event (Env * env, XButtonEvent * event)
{
  XEvent  an_event;

  while(TRUE) {
    XNextEvent(env->the_display, &an_event);
    switch (an_event.type) {
    case ButtonPress   : *event = an_event.xbutton; break;
    case ButtonRelease : *event = an_event.xbutton; break;
    case Expose        : Display_update_init_dim (env, 0, 0, WINDOW_WIDTH, WINDOW_HEIGHT); break;
    default            : break;
    }
    if ((an_event.type == ButtonRelease) || (an_event.type == ButtonPress)) break;
  }
}

/***** now **********************************************************************************************/

char * now (void)
{
  time_t tt;
  tt = time(NULL);
  return (asctime(localtime(&tt)));  /* does this need to be freed ??? */
}

/***** strptr_cmp ***************************************************************************************/

int strptr_cmp (char ** sp1, char ** sp2)
{
  return (strcmp(*sp1, *sp2));
}

/***** wait_for_left_click ******************************************************************************/

void wait_for_left_click (Env * env)
{
  XButtonEvent xbe;
  Boolean      pressed = FALSE;

  while (!pressed) {
    get_mouse_event (env, &xbe);
    if (xbe.button == 1) {
      if (xbe.type == ButtonPress) {
        pressed = TRUE;
        while (TRUE) {
          get_mouse_event (env, &xbe);
          if ((xbe.button == 1) && (xbe.type == ButtonRelease)) break;
        }}}}
}

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