1983

The Faculty evaluation system

Peter Weiler

The University of Montana

Let us know how access to this document benefits you.
Follow this and additional works at: https://scholarworks.umt.edu/etd

Recommended Citation
Weiler, Peter, "The Faculty evaluation system" (1983). Graduate Student Theses, Dissertations, & Professional Papers. 4682.
https://scholarworks.umt.edu/etd/4682

This Thesis is brought to you for free and open access by the Graduate School at ScholarWorks at University of Montana. It has been accepted for inclusion in Graduate Student Theses, Dissertations, & Professional Papers by an authorized administrator of ScholarWorks at University of Montana. For more information, please contact scholarworks@mso.umt.edu.
COPYRIGHT ACT OF 1976

THIS IS AN UNPUBLISHED MANUSCRIPT IN WHICH COPYRIGHT SUBSISTS. ANY FURTHER REPRINTING OF ITS CONTENTS MUST BE APPROVED BY THE AUTHOR.

MANSFIELD LIBRARY
UNIVERSITY OF MONTANA
DATE: 1983
The Faculty Evaluation System

by Peter Weiler

Presented in partial fulfillment of the requirements for
the degree of Master of Science

University of Montana

1983

Approved by:

Chairman

Dean, Graduate School

6-10-83

Date
Weiler, Peter P., M.S., June 1983 Computer Science

The Faculty Evaluation System (268 pp.)

Director: Dr. John Barr

The faculty evaluation system was developed to be an automated aid for evaluating faculty. The instructor submits a request for questionnaires. Questionnaires are printed on Scantron readable forms. The forms are read by a Scantron card reader and the results of the questions are interpreted to produce reports.

The evaluation system was designed by using all of the phases of development for a large software system. The specifications and requirements of the system were approved by the thesis committee. A logical design of the system was then developed. The logical design consisted of data flow diagrams and PSL/PSA reports describing the diagrams.

The programs were written according to the specifications and followed very closely the structure set up by the logic design. The programs were tested mostly under actual operation conditions. All of the programs were found to operate according to the specifications set up for them.

In closing problems encountered in developing the system and recommendations for future software projects are discussed.
# Table of Contents

Abstract.................................ii
List of figures.............................iv
Introduction................................1
The Faculty Evaluation System.............3
System Requirements......................9
Developing the Faculty Evaluation System..12
Integration and Testing....................18
Problems and Recommendations............24
Summary and Conclusion...................27
Bibliography................................30
User's Manual............................A-1
System Manual.............................B-1
Logic Design..............................C-1
Program Documentation....................D-1
Programs..................................E-1
List of Figures

Identification card...............................A-12
File Identification card.........................A-13
Logic Diagram 0..................................C-0
Logic Diagram 1..................................C-18A
Logic Diagram 1.1.................................C-29A
Logic Diagram 2..................................C-39A
Logic Diagram 3..................................C-50A
Program Flow Diagrams..........................E-a
Evaluation System................................E-b
Generate Forms...................................E-c
Generate Reports..................................E-d
Question Maintenance............................E-e
Introduction

The faculty evaluation system was developed as an automated aid for evaluating faculty. The system allows the evaluations to be processed at a central location by data entry personnel. The system was set up and documented to allow unexperienced data entry personnel to use the system. The user's manual and the system manual are described in appendix A and appendix B respectively.

The faculty evaluation system is composed of three parts. The first part reads in the information on the instructors that is needed to produce questionnaire forms. The first part of the system produces the questionnaire forms on Scantron readable cards. The second part of the system uses a Scantron card reader to read in the answers to the questions and generate statistics about them. A report for the instructor and the faculty evaluation committee is then produced showing the results of the questionnaires. The instructor receives a report on both the general questions and on the questions that he picked from the database. The faculty evaluation committee receives a report
with only the results to the general questions. The third part of the system is a maintenance program for the question data base and the standard questionnaire form data base.

The requirements for the project were developed by prototyping the system and discussion with members of my thesis committee. The final draft of the requirements and specifications were the bases for all of the future development.

The development of the system is then discussed. This consists of the logical design (appendix C), the program documentation (appendix D), and the programs (appendix E). The logical design consists of data flow diagrams and PSL/PSA formatted problem statements which describe the data flow diagrams in detail.

The faculty evaluation system was tested mostly under actual operation conditions. The programs were tested to see if they matched the requirements and the specifications. The complete system performs all of the items described in the requirements and specifications.

In closing some of the problems that were encountered in developing the system will be discussed, along with a summary of the system and recommendations for future projects.
The Faculty Evaluation System

The faculty evaluation system is being used by the university because a collective bargaining agreement between the University of Montana Teacher's Union and the Board of Regents requires that there be some method of evaluating faculty. Each faculty member is rated as to how well he performs in the following categories:

Research
Public Service
Teaching - The faculty evaluation will measure this.

The faculty evaluation system creates a questionnaire with 10 general questions and up to 23 private questions. The 10 general questions consists of seven demographics questions about the students taking the course. The other three general questions are used for evaluating the instructor. The 23 private questions are chosen by the instructor from a data base of questions.
The faculty evaluation system has some other benefits besides being required. The system is used to aid teachers in the development of their teaching abilities. Each instructor is ensured that only he will see the results of the confidential questions. The confidential results will allow the instructor to ask questions that will provide him with feedback that will benefit him. By selecting appropriate questions from the available data base the instructor can obtain ideas on how to improve his teaching.

The faculty evaluation system will aid the faculty evaluation committee by providing standard reports. The committee will use the global questions to make general personnel decisions, deciding, for example, if a faculty member should be promoted. The global questions will also provide information on what the students think about the course.

The faculty evaluation system is composed of three basic parts: production of instructor questionnaire forms, generation of the reports on the answers to the forms, and maintenance of a data base of evaluation questions and standard questionnaire forms.

The first part of the faculty evaluation system produces the evaluation questionnaires. The various departments on campus receive a list of all the questions in
the data base and a copy of all of the standard forms. The instructor then chooses the questions that he wants on his evaluation form. If a department is using the same questions they can request that a new standard form be created for them. If the instructor wants to have additional questions on his evaluation form he can write the questions on his question request form and they will be added to the question data base.

The department, or instructor, turns the question request forms into a central location where the information on the forms is entered into the instructor data base. The routine to enter the instructor information into the instructor data base allows the data entry personnel to add an instructor record, delete an instructor record, or print a list of all of the instructor records that have been printed to date.

The questionnaire forms are printed at the end of the quarter. The routine that prints the forms only submits a limited number of the questionnaires at a time to keep from overloading the line printer queue on the system. After a questionnaire form is created for an instructor record a flag is set in the instructor's record to ensure that only one set of forms will be printed for any single instructor record each quarter. The program that creates the questionnaire forms from the instructor records also creates
a print command file for printing the questionnaires with the appropriate switches. The main reason for having the print commands file is so that the number of copies of the questionnaires can be set by the spooler queue instead of having to create a very large file with multiple copies. This was set up to save disk space on the system. If we could write directly to the lineprinter with forms control it would be better to create the large files.

The questionnaire forms are a special form which can be read by a Scantron card reader. The ten demographic and global questions that will be used by the faculty evaluation committee are preprinted on the forms. The forms also have preprinted marks on them so the user will mark the correct spots on the card so a Scantron card reader can read in the answers.

The questionnaire forms are passed out in the instructor's class at the end of the quarter. The answered forms are then returned to a central location so they may be interpreted and have reports produced from them.

The answered questionnaires are fed into a Scantron card reader along with a header card identifying the department, course, and section number for each questionnaire set. The card reader will read the answers off of the cards and place the results into a file on the
computer. The data entry person entering the questionnaire answers will then run a program that will compute the frequency, mean, and standard deviation of the questions for the instructor, as well as a brief summary of the level of student consensus on each question. The statistics for each instructor record are placed into separate files that will be used for report production.

After the user has created a number of statistics files he runs a program that will produce the reports for the instructor and for the faculty evaluation committee. The report for the faculty evaluation committee contains only the ten preprinted demographic and global questions. The report for the instructor contains the results of the ten demographic and global questions as well as the results of his picked questions. The report program generates an error message if it cannot find a unique instructor record in the data base or when it cannot match the identification record that is at the beginning of the statistics file. The data entry person then has to correct the errors or run the program and enter the instructor's name as the additional information to make the instructor record lookup find a unique record.
The reports are generally produced at the beginning of the quarter following the evaluation. The reports are passed out so that the material remains confidential.

The third part of the faculty evaluation system has been mentioned briefly already. It is the program that maintains the questions and the standard forms. This program allows the user to add, edit, and delete questions from the data base. The routine will also print out a list of all of the questions in the data base. The program also adds and deletes the standard forms from their file. The program uses the questions in the question data base to create copies of the standard forms when the user chooses the option to print all of the standard forms.

The faculty evaluation system was designed to be easy for the user to operate since people inexperienced in the use of computers would be using it. The programs were set up to trap all of the errors that I could think of and that the user could make. In all of the programs the input is parsed so an invalid entry will not cause the program to fail.
System Requirements

The faculty evaluation system is a means for evaluating faculty. The system will generate questionnaires to be completed by students in each of the instructor's classes. The results of the questionnaires will then be evaluated and copies of the general questions will go to the faculty evaluation committee and the instructor. The results of a list of instructor picked questions will go to the instructor only.

Question Requests:

Each instructor that is using the faculty evaluation system will have the first ten questions of the evaluation questionnaire generated automatically. These questions consist of two types of questions. The first seven questions pertain to demographic questions about the students. The next three questions are to obtain general information about the instructor.
The instructor has the option of adding up to 23 additional questions to the questionnaire by picking questions from a given list of allowed questions. The instructor can also pick a standard form as his additional questions. The standard forms contain questions from the list of allowed questions. The list of allowed questions consists of a set of questions purchased for the reason of evaluating faculty and questions added by departments and instructors.

Questionnaire Generation:

The information about each instructor and the questions that he is using will be used to print out the instructors requested number of forms. The questionnaire forms will be printed on a special form that can also be fed through a Scantron card reader. The ten questions referenced above will be preprinted on these forms. The questionnaires will then be distributed to the instructor's to be passed out to the students at the end of the quarter.

Report Generation:

The questionnaire answers will be entered into the computer by the Scantron card reader along with the general information on the instructor. The questionnaire answers will be converted into statistics to be used in generating the reports. The statistics generated will be a frequency
of the occurrence for each answer to each question, and the mean and standard deviation for each question.

Reports will be generated that will use this information and the information about the instructors. The reports will list the general questions and the questions that the instructor picked. The reports will list the frequency of occurrence for each answer to each question and the mean and standard deviation of the answers for each question. The report on the general information and the demographic questions about the students will go to both the instructor and the faculty evaluation committee.

Program Maintenance:

There will be an automated method for updating the list of questions and the standard forms. There will also be a method used to print out the list of questions and the standard forms.
Developing the Faculty Evaluation System

The faculty evaluation system has undergone considerable changes since I started work on it. The initial faculty evaluation system was purchased from the University of Illinois as the ICES (Instructor Course Evaluation System)[1]. This system was written in COBOL and designed to run in batch mode on an IBM machine. The initial project was to install the system on the University of Montana's DECsystem_20. After considerable effort in trying to modify the programs so that they would run on the DECsystem_20, I came to the conclusion that it would be easier to write a system of my own instead of trying to set up the ICES system.

The first part of the project that was to become my thesis was to model the ICES system. I used data flow diagrams[2] to understand the different parts of the system. The data flow diagrams were described in detail by using PSL/PSA[3,4,5]. By using PSL/PSA I generated reports[6] to describe the different data flow diagrams in detail. The reports produced by PSL/PSA were the basis for the data
structures and program structures that I used in the evaluation system.

The second step in the thesis project was to develop a logical model of the ICES system. By doing the logical model I found that there were a lot of routines that we did not need in the system. A major difference in installing the system on the two computers was that the IBM machine was run in batch mode whereas the DECsystem 20 was run in interactive mode. The logical model gave me an indepth understanding of the components of the evaluation system.

There were some problems in creating the models using PSL/PSA, since nobody had any knowledge about how to use the automated design aid. This problem was overcome by doing a lot of reading and trial and error[7,8].

At this point of development of the thesis a problem arose. The system had to be operational by the end of the quarter which was only two weeks away. At this point the only programs that were working from the ICES system were the ones to create the questionnaire forms and the instructor data base. I set up what was to be a temporary system to read in the Scantron cards and create statistics files. I also wrote a program to create the reports for the instructor and the faculty evaluation system. The report program used the same data structure as the original ICES
system. The Scantron cards that we were using were roughly the same as the cards used by instructors in giving tests, with one difference. The Scantron cards that we used were printed poorly so only one in four cards was being read. A lot of the questionnaires had to be rewritten onto good forms and run through again.

The temporary programs that were developed were not just thrown together. At the time that they were written I had a complete knowledge of how the system worked and what kind of reports should be generated. The programs had a very short design phase and were written using an incremental development mode. Each piece was then tested separately and the routines were set up for a data entry person to use with me doing some heavy consulting in the initial operation of the programs.

Once the pressure was off I started working on the project following the "normal" software life cycle. I spent a considerable amount of time in writing the requirements and specifications for the system. There had been a number of changes in how the complete system should operate and they had to be resolved before the project could continue.

After the requirements for the faculty evaluation system were accepted there were enough changes in the overall system to require that it be redesigned. I used
data flow diagrams and PSL/PSA reports to define the final evaluation system (see appendix c). The logical design for the system that I developed was tailored closer to what our university needed than the original ICES system had been. In the new system the instructor record was defined to include only the information that we needed. The questions were then altered so they could be printed easily on a special Scantron form.

At this point once again programs were needed so that the instructor information could be entered before the end of the quarter. I spent several weeks studying different languages that could be used in developing the system. The original Scantron interpreting routines were written in FORTRAN. Since FORTRAN is a supported language I studied the possibility of writing the programs for the system in FORTRAN. There was one major drawback to using FORTRAN. FORTRAN does not have a dynamic method of allocating storage so a lot of the structures would be very sparsely used most of the time.

Another possible language to use was COBOL. This language was discarded because of the difficulties in file handling and in structuring the code.
The language that I spent the most time studying was 1022 data base programming language. This language would allow me to write trivial programs to generate the reports that I needed. There were several drawbacks to this language. The most notable drawback was that nobody new enough about the language for me to do the file handling that needed to be done. The 1022 data base language is difficult to use when you want to access multiple files.

The language I chose to write the system in was Pascal. The only major drawback to this language was that it was not supported on our computer system. Pascal had all of the data structures that I needed to allow dynamic storage and structured records and programs. Pascal also had all of the file handling operations that I need for the system.

With the logical design of the system I developed the programs to enter in the instructor records and to print the questionnaire forms (appendix E). I had previously developed the program to maintain the question data base and the standard forms (appendix E). In developing the programs the requirements and the logical design aided greatly in understanding what needed to be done and how it should be implemented.
After all of the programs were written I settled down and started writing manuals on how to use them (appendix A and appendix B). The manuals describe in detail how to use the programs and how to use the entire evaluation system. The manuals also discuss the error messages that the programs produce for invalid data and methods of correcting errors.

The last step of my thesis project was writing up a design document for each program in the system (appendix D). This document helped refresh my memory on how I implemented several routines and left me with some possible methods for improving the system. The design document describes in detail all of the procedures in the program and how they interconnect. The document also discusses the flow of data between the procedures. The design document would be an excellent aid for reimplementing the system in some other block structured language.
Integration and Testing

The faculty evaluation system was tested under actual operating conditions. The routines were set up and checked to see if they were free of most of the errors. The data entry personnel were then allowed to use the programs. There were relatively few errors found by the data entry personnel.

The routines were set up in three basic groups: the questionnaire generation routines, the report generation routines and the questionnaire maintenance routines.

The report generation routines were the first set of programs written. The initial versions of these routines were used to print the reports from the data bases created by the original ICES system [1].

The statistics routine in its original version read information from scantron cards that instructor's were using for courses. There was an error in resetting the statistics values that was corrected at a later date. For the second quarter of use a different set of forms were used. There
was some initial problems in changing the program to operate with the new forms. As of the current date the statistics routine is generating all information correctly. Some results were checked by hand to verify this. The statistics routine is also generating all of the statistics that were specified in the system requirements.

The report generating routine was also set up to produce reports using the original ICES files. The report routine prints two reports, one for the instructor of the course and the other for the faculty evaluation committee for the department the instructor is under. This routine had some errors in printing the results from the statistics because the statistics were not being created correctly. Once the statistics routine was corrected the report program ran without errors for the rest of the quarter.

For the second quarter of operation the statistics routine had been creating the statistics incorrectly because of the change in forms. This error caused invalid reports to be printed. Once the statistics file was changed to correct for the new forms the report program was again producing reports correctly. Some of the reports that the report program produced were checked by hand to insure that they were being produced correctly. The reports that were sent to the instructors when all of the routines were running correctly were never reported to be in error.
The complete report generating programs are currently running without any apparent errors at this time. For proof of correctness I can only say that at this time with the current routines that are being executed no error has been found. The routines have run for a complete quarter and there is no evidence that they will fail in future quarters.

The program that was created next was for editing the question data base and for entering standard forms. The program was initially set up to change the questions that came with the original ICES system. The questions were all changed so that they all would use the same responses for the answer to the question. After the program was set up data entry personnel began using it to update the question list. By printing out the updated list of questions and checking them against what was entered no errors were found.

Routines were then added to the program to add, delete, and print the standard forms. These routines were also put into immediate use with data entry personnel operating the program. The program was checked to see if the standard forms were being generated correctly by online use. There were no errors reported from the instructor's that used the standard forms.
The last set of programs installed were the routines for creating the questionnaire forms for the instructor. The first program is used to generate the instructor data base. The second program is used to generate the questionnaire forms and to create a print command file to print the forms.

The program to create the instructor data base was installed for use and the data entry person entered an invalid entry which was not correctly handled by the program. The program was then changed to parse all of the input that was entered so that it would print out an error message and have the user reenter the value. The program was set up to add, delete, and print the instructor records. The program had another error that caused the loss of a lot of instructor records. The routine was writing to the output file and then reading the information back in after ten modifications had been made to the data base. The problem with backing up the data base using this method is that Pascal keeps the file pointer to the input file pointing to the original file not to the updated version. This error caused the system to read in the information from the wrong version of the instructor data base file. This error was corrected late in the last quarter that this routine was used. The program appears to be running correctly as of the current data and allows the user to
perform all of the tasks specified in the requirements document.

The program to generate the questionnaires and create the print command file was set up without any apparent errors. A hidden error was found immediately upon running the print commands file. The print command routine submitted about 550 jobs to the line printer within a matter of minutes. The massive amount of print requests that then occupied the printer queue caused the line printer to freeze up. The operator was getting an extremely slow response when she tried to look at the queue. The print jobs all had to be cancelled before any more problems happened to the system.

The problem that was encountered in submitting all of the questionnaire forms for printing at once was corrected in a later version of the program that has yet to be fully tested. In this version of the program at most 100 print requests can be submitted at any one time. The routine changes a flag in the instructor's record to signify that the instructor's questionnaire forms have been printed. The reason for creating the print commands file was to automatically set the number of copies of forms to be printed instead of creating one large file with multiple copies of the form. The tests that have been run on this routine show that it is running correctly for the given
input.

The complete faculty evaluation system has only been tested in parts up until the current quarter. All of the separate parts appear to be working as specified in the requirements specification.
Problems and Recommendations

In doing my thesis I encountered a number of problems that considerably lengthened the time it took me to complete my thesis. These problems could be eliminated for the most part in future thesis projects. One of the main handicaps that I had was that I was the first one through our program and as such there where no clear guidelines to follow for doing my thesis.

I was also in the dark most of the time as to what steps I had to do to continue in an orderly fashion on the project. In the future I strongly recommend that the person writing the thesis have a good understanding as to the steps to take in developing the software project. There should be some criteria as to the quality and the completeness required for each phase of development of the software project. I spent considerable time rewriting things because I did not understand what was required when I started.
The part of the project that caused most of the set backs was the changing of the specifications of the system. After the system was running for one month we found a lot of things that had to be changed to make the system run a lot smoother. The changes forced the system to be redesigned and some of the old programs partially rewritten. For this project I felt that all of the changes were for good reasons and necessary to the complete system.

The initial set up of the system was a prototype of the complete system. The prototype was developed to determine how the system would work. After the system had been running we knew enough on how to improve the evaluation system for running on the university's computer and for our users.

Another problem that I encountered was using Pascal on the system. Our version of Pascal is undocumented. I had a number of routines that failed because of the way our Pascal compiler worked. If at all possible the next thesis writer should use a language that is both transportable and has documentation written for it.

At this point the reader may wonder why I used Pascal instead of some other documented language. I used Pascal because of the dynamic storage structures that it allows and so I could write structured code. I had studied several
languages that could be used for the project and because of
the structuring and the file handling capabilities I chose
Pascal.

There were also a couple of minor problems that I had
with the system that I never thought about. The
questionnaire generating routine, the statistics routine,
and the report routine all create a large number of files.
There are some limits on the DEC20 as to the number of files
that can be placed in any one directory. Also, when
directories have a large number of files in them, some
commands take a long time to execute.

I solved this problem for the most part by having the
programs write the files that they were creating to
subdirectories. This correction left the main working area
free of all temporary files. The main directory was left
with the executable versions of the programs and the data
files needed to create the reports.

I inadvertently found another problem with our computer
system. The routine that generates the questionnaire forms
also creates a command files to submit the questionnaire
forms for printing. When I executed the command program for
the first time I overloaded the lineprinter queue on the
system. I had to change the program so only a maximum of
100 forms would be sent to the print queue at any one time.
Summary and Conclusions

The faculty evaluation system was designed and implemented to aid in the evaluation of faculty. The system was set up so that the evaluations could be tabulated in a single area instead of having each department form a student committee to tabulate the results. The system was also set up so that instructors could choose a wide variety of questions and improve their teaching by the feedback.

The system was implemented after extensive modeling of the old system was done. The system was then redesigned and implemented after a quarter of testing. The current system is set up so that it can be easily changed as new ideas are presented.

The project was very close to a project done in industry. The only difference being that this system probably has better documentation and a lot more time went into this design than into a normal business system. The project helped me a lot in writing a program system that had to be used by data entry person and which therefore had to be user friendly.
In conclusion, I have several recommendations for future projects of this nature. The project should be set up so the developer does not have to rush the code and has time to go through all of the program development stages before the code is written. The researcher should also be told exactly what was expected in the way of documents for each of the stages of development.

The final program system should have the documentation and quality of a professional software project. In my project I learned a lot about interfacing a program to a specific user. The researcher should make sure the requirements and the logic design are done to the specifications of the customer or project head. This will save a lot of time in redoing code and designs.

The faculty evaluation system could have been written better if I followed all of the steps in developing a program system in the proper order. I would not have had to rewrite the programs for the statistics and the report programs if the data structures had been completely known when I wrote the programs.

The program system uses a lot of what I learned in the graduate level courses on writing programs. All of the programs are very modular and there is a minimum of connectivity between the different procedures in the
programs. The system was developed in an incremental design format. This design format allowed for additional features to be added easily and for the programs to be tested and debugged in easy to manage sections.

The complete project was an application of what I learned in the graduate courses. The courses gave me additional insight on how to go about developing software and produce better quality code.
Bibliography

1. Instructor Course Evaluation System, published by the University of Illinois at Urbana-Champaign, 1977.


APPENDIX A
USER'S MANUAL

Create Questionnaire Forms

The routines to create the instructor questionnaire forms should be run from about the middle of the quarter to the end of the quarter that the forms are for. This will give ample time to produce all of the reports from the previous quarter. If any questions are to be added to the question data base they should be entered before the information on the instructors is entered. The routine to enter questions will be discussed later in this manual.

Before the user starts entering the instructor information for the current quarter the instructor information from the previous quarter should be deleted. The reason for this is a lot of classes have the same course
and section numbers each quarter and this could cause troubles when producing the reports. To delete the previous quarter's information type:

@DEL INST.SRT
@DEL INST.RPT

It is important to note that all of the reports for the previous quarter should have been made before deleting this file or the information needed to create the reports for the previous quarter will have been deleted. If this mistake was made, talk to the computer center about getting the file back.

The faculty evaluation system manager should send out instructor questionnaire request forms at the beginning of the quarter in order for them to be returned by about the middle of the quarter. The data entry person then enters the information into the computer by running the following program.

@TAKE INSTRC

This program modifies the list of instructors using the faculty evaluation system. This program contains the following options:

1) A - add an instructor
2) D - delete an instructor
3) P - creates a printable instructor file
4) ? - prints this menu
Adding an instructor to the data base

The program prints out a number of options for the user. The user chooses an option and then enters the information that the prompts request. To add an instructor to the data base follow the example given below.

Enter option : (A,D,P,?) A

Enter the instructor information.

Enter the instructor's name: JONES
Enter instructor's initial: J
Enter the dept. abr.: 133
Enter the dept. no.: 133
Enter the course number: 322
Enter the section number: 1
Enter number of forms: 22

Is information correct? (Y or N) N
Enter the instructor's name: JONES
Enter instructor's initial: J
Enter the dept. abr.: CS
Enter the dept. no.: 112
Enter the course number: 322
Enter the section number: 1
Enter number of forms: 22

Is information correct? (Y or N) Y

In the above example the user made an error in entering the department abbreviation the first time and had to reenter the record to have the correct information in the record. If the user had not hit a carriage return before the error was noted the user could have corrected the mistake by using the delete key.
After the user has verified that the information is correct the program will continue and let the user enter in either the standard form number or the numbers of the instructor's picked questions. If the user is going to enter the instructor's picked questions then type the following:

Enter form number, (0 if picked questions): 0

Enter the question numbers for the instructor. If you have fewer than 23 questions type a , 0, as the final question number to stop.

Enter the number of question 1 : 1
Enter the number of question 2 : 3
Enter the number of question 3 : 5
Enter the number of question 4 : 8
Enter the number of question 5 : 33
Enter the number of question 6 : 45
Enter the number of question 7 : 67
Enter the number of question 8 : 88
Enter the number of question 9 : 29
Enter the number of question 10 : 167
Enter the number of question 11 : 322
Enter the number of question 12 : 333
Enter the number of question 13 : 451
Enter the number of question 14 : 536
Enter the number of question 15 : 857
Enter the number of question 16 : 0
Is information correct? (Y or N) Y
The user may enter at most 23 instructor picked questions. If fewer questions are to be entered type a zero as the last question number. (See example above.) If the instructor has chosen a standard form then the following information should be entered.

Enter form number, (0 if picked questions):  2

Is information correct? (Y or N) Y

By entering the standard form number all of the questions in the standard form are entered into the instructor's record. The standard forms are entered into a data base by the question program which will be explained later in this manual.

Once an instructor record has been entered the following prompt is given.

Do you want to continue? (Y or N)

If the answer given to the question is ,N, then the program will stop and the user may continue to the next step of creating the questionnaire forms. If the answer to the question is ,Y, then the program will return with the prompt asking the user to enter in an option.
Deleting an instructor record

If an error in an instructor record is not noticed until after the record has been entered, then the user will want to delete the record. To delete a record the user types the following.

Enter option : (A,D,P,?) D

Enter instructor's name being deleted: JONES
Enter instructor's course number: 322

Instructor name: JONES
Instructor initial: J
Department abr: CS
Department no: 0112
Course number: 322
Section number: 001
No. of questions: 022

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0001</td>
<td>13</td>
<td>0451</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0003</td>
<td>14</td>
<td>0536</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0005</td>
<td>15</td>
<td>0857</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0008</td>
<td>16</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0033</td>
<td>17</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0045</td>
<td>18</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0067</td>
<td>19</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0088</td>
<td>20</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0029</td>
<td>21</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0167</td>
<td>22</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0322</td>
<td>23</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0333</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Do you wish to delete this instructor? (Y or N) N

In the above example the user wanted to either delete or view the record. The user entered the instructor's name and the course that he was teaching. The instructor record was then displayed. If the user had chosen to delete the
record he could have typed \textit{Y}, instead of \textit{N}.

The routine will list all of the records that are in the instructor data base that have the same instructor and course names. If an instructor is teaching two sections of the same course both records will be displayed. By displaying all records that have these same fields the user can view records and remove duplicates.

If the user wants to see if a record was entered correctly he can run this routine and not delete the record.

\textbf{Printing the instructor record}

The user has the option of printing all of the instructor records in the data base. To print the records the user types the following:

\textbf{Enter option : (A,D,P,?)} \texttt{P}

Created file instr.rpt

The above file may be printed out on the line printer by typing the following after exiting the program:

\texttt{@PRINT INSTR.RPT}

The user should create the report periodically while entering the instructor records to see if all of the information is correct. A report will also be needed when the user is running the report program to see if any errors
were made in entering any information in that part of the system.

After the user has picked up a copy of the instructor records the file INSTR.RPT should be deleted to save some room. The report can be created again by running the BLDIST program so once a copy is obtained delete the file by typing the following:
@DEL INSTR.RPT

Printing questionnaires

Before the user runs the report to create the instructor questionnaires he should delete the batch file that keeps track of the number of the questionnaire form. To delete the batch file type the following:
@DEL BAT.NUM

The batch file should only be deleted once a quarter. The user should delete it before running the create instructor questionnaires routine for the first time in the current quarter.

To create the instructor questionnaires run the following program:
@RUN PRPT1
2 questionnaire forms created

EXIT

To print the questionnaire forms run the following program:

@TAKE PRCMD

The program **PRPT1** creates questionnaire forms from the instructor records. The program will only create at most 100 forms at a time so as not to overload the line printer queue. The user will then have to wait until the forms have been printed before printing more. Usually it will take about a day to print 100 forms. If the user has entered more than 100 classes the routine will create at most 100 questionnaire forms and the commands to print them. The user will have to wait and rerun the program for the rest of the forms to be printed.

After the user has run the routine **PRCMD** the file **PRCMD.CMD** may be deleted since it is created each time the **PRPT1** program is run. The routine **PRCMD** submits all of the questionnaire forms to be printed. To delete the file type the following:

@DEL PRCMD.CMD
The user may want to run the program PRPT1 after entering a couple of departments so that separation of forms will be made easier. The forms should be put into envelopes and sent to the respective department the week before the final week of regular classes.

To save space the instructor questionnaire forms are deleted after they have been printed. If an instructor's questionnaires are lost, then the instructor's record should be deleted from the data base by using the BLDIST program. The instructor record must then be reentered and printed again.
Manual for printing reports

This section of the manual deals with producing the reports for the instructors and the faculty evaluation committee. The instructors or department turn in the questionnaire answer forms to some central location. The answer forms should for each instructor should be in an envelope with the instructor's name, department number, course and section number written on it.

Running Scantron Cards

A scantron identification card has to be made for each course. The scantron identification card will have the department number, course number, and section number marked on it as in figure (1). On the scantron card the first three fields are the department number, the second three fields are the course number, and the next three fields are the course section number. All three of the fields have to be filled out for the header record to be entered properly. If the number being entered is greater than 5 then the field marked 5 and another field have to be marked so the combined
The sum of the two numbers equals the number desired. For example, by marking fields 5 and field 3 the number 8 will be entered. The identification card should then be placed into the envelope with the questionnaire answers.

**Figure 1**

<table>
<thead>
<tr>
<th>Department number</th>
<th>Course number</th>
<th>Section number</th>
</tr>
</thead>
<tbody>
<tr>
<td>c:2 j:3 c:4 j:5 c:6</td>
<td>c:2 j:3 c:4 j:5 c:6</td>
<td>c:2 j:3 c:4 j:5 c:6</td>
</tr>
<tr>
<td>c:1 j:3 c:4 j:5 c:6</td>
<td>c:2 j:3 c:4 j:5 c:6</td>
<td>c:2 j:3 c:4 j:5 c:6</td>
</tr>
<tr>
<td>c:1 j:3 c:4 j:5 c:6</td>
<td>c:2 j:3 c:4 j:5 c:6</td>
<td>c:1 j:3 c:4 j:5 c:6</td>
</tr>
<tr>
<td>c:1 j:3 c:4 j:5 c:6</td>
<td>c:2 j:3 c:4 j:5 c:6</td>
<td>c:2 j:3 c:4 j:5 c:6</td>
</tr>
</tbody>
</table>

The person entering the questionnaire answer cards should be familiar with the use of the scantron card reader. Documentation on the use of the card reader is available at the computer center.

The faculty evaluation system was set up to run with the scantron card reader on the system. The scantron card reader allows the generation of up to 10 different files at a time. The data entry person has to fill out 10 cards to identify each of the files. An example on how to fill out the cards is shown below (fig.(2)).
In the example the first two fields tell the scantron card reader to expect an identification field. Fields two through eight represent the code number for the faculty evaluation area. This field tells the scantron card reader where to put the results from the cards that it is reading. The identification number is given to the user by the computer system operators. The last field in the header card is to set the file name number. The first header card in the example will create file .FILE1.DTA, the second header card will create the file .FILE7.DTA. In order to run the questionnaire answers the user should create cards
for file numbers zero through nine. The file identification cards may be used multiple times as needed.
Generating Statistics files

The user must delete the batch file for the statistics routine before running the statistics program for the first time in the quarter. To delete the batch file for the statistics routine, type the following:

@DEL BATCH.NUM

The batch file is used to number the statistics files produced by the STATS routine. The user must also delete all of the statistics files that were generated for the previous quarter. These files must be deleted or there will not be enough room to create all of the files for the current quarter. To delete the statistics files the user must type the following:

@DEL STS:.*,*

The above files should only be deleted once a quarter.

To enter the questionnaire answers into the computer the user should log onto the computer and type the following:

@CONN
This will put the user into the area where the questionnaire answer files will be placed. The user then feeds in a file identification card, the course identification card, and the questionnaire answers for that course. This will create a file with the necessary information to produce a report. The user starts by using file identification card 0 and continues until he either has entered file 9, or no longer has any questionnaire answer cards to feed in.

The user must run the scantron card reader, as specified in the scantron manual, so that each of the different sets of instructor questionnaire answers is placed in a different file. To briefly describe what must be done the user hits the rekey button twice, enters a set of cards (the file header card, course header card, and questionnaire answers for that course) then hits the rekey button twice more to close the file.

The user must perform the following commands to create the statistics files after the scantron files zero through nine have been created. The information will be overwritten if more cards are entered before the statistical routines have been executed.
The user will then type the following command:

@COPY FILE*.FAC

This is followed by a carriage return. This will copy the questionnaire data files into an area where they will be processed. The user then types:

@CONN FAC
@RUN STATS

The first command connects the user to the directory where the information will be processed. The second commands takes the raw data and creates a statistics file on the questionnaire answers. The output from the program is as follows:

Input file is FILE0.SCN output file is IN0034.dta
Input file is FILE1.SCN output file is IN0035.DTA
Input file is FILE2.SCN output file is IN0036.DTA
Input file is FILE3.SCN output file is IN0037.DTA
Input file is FILE4.SCN output file is IN0038.DTA
? FILE5.SCN (0) file not found
? FILE6.SCN (0) file not found
? FILE7.SCN (0) file not found
? FILE8.SCN (0) file not found
? FILE9.SCN (0) file not found
EXIT
In the above example statistics files were only generated for files 0-4; the other files were empty, so no information was generated about them.

It is extremely important that the user keep a log of what he is doing. The log should contain the following information for each set of course questionnaire answers: The instructor's name, the scantron file number used, and the statistics file number created. This information is used in the case where more than one teacher is teaching a class.

The following error message may occur:

File FILEN.SCN contains no data, rerun

Where the 'N' in the file name is the number of the file. This error message means that the scantron card reader is probably not reading in information correctly and is missing data. In this case the questionnaire answer cards for this course should be reentered.

The user the types the following commands:

@DEL FILE*.*
@EXP

This will delete the scantron files out of the work area.
The user then connects back the area where the scantron files are initially sent by typing:

@CONN

The user then deletes the scantron files from this area by typing:

@DEL FILE*. *
@EXP

Producing Evaluation Reports

After the user has entered as many questionnaire answers as desired, the report generation program may be run. The user needs to know the file numbers for which reports have to be generated. This information should be in the log that the data entry personal made while entering in the questionnaire forms. If the user knows that number of the file that a report was last generated for, then the file number of the last file entered can be found by typing:

@DIR STS:

The user then runs the following program to produce the report:

@RUN REPORT
The program will then list the following information:

ICES Report generating program. This program will generate reports for both the instructor and the Faculty Evaluation Committee. The instructor's statistics file is used to get information from the database for the reports.

Do you want single or multiple file entry? (S or M) M

Enter the lowest statistics file number: 1

Enter the highest statistics file number: 10

EXIT

In the above example the user typed ,M, to signify the entry of more than one file. The user then typed the number of the file where the reports were to start and then the number of the file where the reports were supposed to stop. The above routine will generate reports for files IN0001.DTA to IN0010.DTA.

If the user wants to generate a single report, he types the following:

@RUN REPORT

The program then prints the following:

ICES Report generating program. This program will generate reports for both the instructor and the Faculty Evaluation Committee. The instructor's statistics file is used to get information from the database for the reports.

Do you want single or multiple file entry? (S or M) S

Single file entry routine:
Enter statistics file number: 63
Enter the Instructor's last name: JONES
If you have another entry type ,c, to continue. C

Single file entry routine: Enter statistics file number: 70
Enter the Instructor's last name: SMITH
If you have another entry type ,c, to continue. N
EXIT

In the above example a report will be produced for the files IN0063.DTA and IN0070.DTA. Note that when the user executes the report program in the single mode the last name of the instructor has to be entered. The reason for this is so that the program can find the correct instructor record in the instructor data base.

Correcting Report Errors

The report program will also produce an error report. If a statistics file has an error in it a report will not be generated for that instructor. The user has to correct the errors and rerun the report program.

The following errors can occur when running the report program.

Duplicate entry in file number: 34 run the file in single mode.
In the above example, more than one instructor was teaching the same course so the program needs additional information to produce the report. When the user runs the report program in single mode he has to type in the instructor's name. This provides the information necessary to produce a report for the instructor. Therefore whenever this error occurs run the report program for the file on single mode. The name of the instructor that corresponds to the file number should be in the log that the data entry personnel is keeping when running the questionnaire answer forms.

File IN0034.dta not found

If this error is encountered there was something wrong with the entry of the information and the data belonging to that file has to be rerun.

If the report program is being run in single mode the following error message may occur:
Instructor's information not in data base

In this case the information about the file number or the instructor's name may have been entered incorrectly. If the information has been entered correctly then the user should check the list of all instructor's in the data base to see if that entry was correct.
The remaining errors fall into roughly the same catagory. These errors have to do with either improperly entered header cards for the questionnaire answers, or the original instructor information was entered incorrectly. The error messages are as follows:

Department name for input file: 112 not found.

Course number for input file: 223 not found.

Section number for input file: 177 not found.

In the above error messages either the department, course, or section number was not found in the data base. In this case most of the errors are caused by incorrect header cards for the instructor's questionnaire answers. To correct this problem the following program must be executed to change the fields that are in error in the header record.

Type the following:

@RUN EDSTAT

This routine allows the user to edit the header record of the statistics file.

Enter the statistics file number: 1

Department number = 155
Course number = 368
Section number = 001

Do you wish to change the information? (Y/N) Y

Enter the dept. no. : 155
Enter the course number : 368
Enter the section number: 10
In the above example there was an error message written saying that the section number was not found for the file. By looking up what the number should have been in the instructor listing the user edited the header record in the statistics file to what it should be. A report will be created for the file by running the report routine in single mode.

Distribution of reports

Once the report program has been run the reports have to be printed. To print the reports type the following:

@TAKE WRITE.CMD

The above routine will print out the reports on the line printer. All of the reports are deleted after they have been printed to save space. To regenerate any of the reports the user runs the report program again with the file numbers desired. The above routine is executed for both the single and multiple entry modes for the report program.
After the reports have been printed they must be distributed to the different departments. The instructor's personal report must go only to the instructor since this is confidential information. The second report must go to the faculty evaluation committee for the instructor's department.
Manual for Question Maintainance

The routine described in this section of the manual is used for adding, deleting, and editing questions in the database. The routine is also used to add and delete standard forms in the database. This routine should be run after the reports have been generated from the previous quarter and before the information for the instructor's is entered for the current quarter.

To run the question maintainance program type the following:

@RUN QUEST

The program will then print out a menu of options for the user. The menu will appear as follows:

This program modifies the list of questions for the ICES system. This program contains the following options:
1) M Multiple question modification
2) S Single question modification
3) D Deletion of a question
4) A Adding a question
5) P Create a printable question list
6) N Add standard form numbers
7) X Delete a standard form
8) L List standard forms
9) ? Prints this menu

The maximum size of a question is 86 characters.
Multiple question modification

The first option listed for the question maintenance routine is for multiple question modification. This option was created to allow the user to change a large number of questions in the question data base. The routine is executed as follows:

Enter option M,S,D,A,P,N,X,L,?) M

Multiple question modification routine
Enter starting question number: 1

Enter end question number: 3

Each question that is entered has to be in two parts of 43 characters or less) each.

0001 THE COURSE OBJECTIVES WERE VERY CLEAR.
Do you want to modify this question? Y or N) N

0002 THE INSTRUCTOR STATED CLEARLY WHAT WAS EXPECTED OF STUDENTS.
Do you want to modify this question? Y or N) N

0003 THE COURSE WAS WELL ORGANIZED.
Do you want to modify this question? Y or N) Y

Enter the first half of the question. ! (43 char)
RATE THE COURSE ORGANIZATION.

Enter the second half of the question. ! (43 char)

In the above example the user entered the option for multiple question modifications. The user then entered the range of questions being modified. The user then answers
the prompt as to whether or not to modify a question. If the question is to be modified the user types in the question in two parts. Note on the prompt above the line where you are entering text there is a vertical bar (\{(43 char)) which tells the user where to stop entering text. The program will only read the first 43 characters in for the question. Any additional characters will be ignored. The vertical bar is displayed to let the user know where the 43 character is. After the user has entered the first string the user is prompted for the second string. If the question has no second string then the user types a couple of carriage returns.

The following prompt is then printed:

Do you want to continue? Y or N) Y

The above prompt allows the user to exit the program or to return to the routine to enter in a new option.

Single question modification

If the user wants to modify only one question in the data base the following information is entered.

Enter option M,S,D,A,P,N,X,L,?) $ Enter question number: 5

The question that is to be entered has to be in two parts of 43 characters or less) each.
0005 THE PROGRESSION OF THE COURSE WAS LOGICAL
AND COHERENT FROM BEGINNING TO END.
Do you want to modify this question?  Y or N) Y

Enter the first half of the question.  1(43 char)
WAS THE PROGRESSION OF THE COURSE LOGICAL?

Enter the second half of the question.  1(43 char)
AN COHERENT FROM THE START?

In the above example the user modified a single
question. The question that initially was question number 5
was overwritten with the new version of the question. After
entering the question the user has the option of returning
to the main routine or exiting the program.

Deleting a question

To delete a question from the data base run the
following routine.
Enter option:  (M,S,D,A,P,N,X,L,?) D
Enter the number of the question to be deleted:  5

0005 WAS THE PROGRESSION OF THE COURSE LOGICAL
AN COHERENT FROM THE START?

Do you wish to delete this question?  (Y or N) Y.

The above routine deletes a question from the data
base. The user enters the question number to be deleted and
verifies that it is the question to be deleted. The routine
will then return with a prompt for the user to tell whether
he wants to return to the main routine or to exit the program.

Adding a question to the data base

To add a question to the data base the user runs the following option.

Enter option: \( (M,S,D,A,P,N,X,L,?) \) &

The question that is to be added has to be in two parts of 43 characters (or less) each.

Enter the first half of the question. 1(43 char)

\textbf{THIS IS A TEST QUESTION.}

Enter the second half of the question. 1(43 char)

Question 0898 created

In the above example the user entered a new question into the data base. The routine will always append the new question to the end of the question data base list. The user may enter the question in either upper or lower case characters.

Printing the question list
The user will want to print out copies of the questions to distribute to the departments using the system. To print a copy of the questions in the database the user runs the following routine.

Enter option: (M,S,D,A,P,N,X,L,?) P

Output file is quest.rpt

The routine has created a printable version of the questions. Assuming the user wants 10 copies he can exit the question routine and type the following to print them out on the line printer.

@PRINT QUEST.RPT/COPIES:10

It will probably take a day or so to print out 10 copies of the questions. Each question list is about 60 pages long therefore, the user should delete the file after it has been printed. To delete the questionnaire list file type the following:

@QUEST.RPT

Entering standard forms

Standard forms are lists of questions from the database. If a department decides to use the same questionnaire form the user can enter in the question numbers for the
standard form. This saves a lot of time in typing question numbers into the instructor record, since you can simply enter the form number instead. This routine is useful to run along with the instructor entry routine. If a number of teachers are using the same questions, the questions may be entered as a standard form and then entered into the instructor's record as a standard form.

To create a new standard form the user types the following:

Enter option : (M,S,D,A,P,N,X,L,?) N

Enter the question numbers for standard form number 19. If you have fewer than 23 questions type a ,0, as the final question number to stop.

Enter the number of question 1 :234
Enter the number of question 2 :235
Enter the number of question 3 :236
Enter the number of question 4 :345
Enter the number of question 5 :366
Enter the number of question 6 :367
Enter the number of question 7 :380
Enter the number of question 8 :401
Enter the number of question 9 :599
Enter the number of question 10 :0
In the above example the user entered a standard form consisting of nine questions. When the user is running the instructor entry routine he would only have to type form number 19 instead of having to enter the numbers of the questions for the instructor if the instructor had chosen these questions.

The user may enter at most 23 question numbers for a given standard form. This limit is set by the number of questions that can be printed on the scantron form. The user may enter fewer than 23 questions if desired.
Deleting standard forms

The user has the option of deleting standard forms since some departments might decide to change their standard form. To reduce cluttering, the user may run the following routine.

Enter option: (M,S,D,A,P,N,X,L,?) X

Enter the number of the form to be deleted: 10

1) 0234
2) 0235
3) 0236
4) 0345
5) 0366
6) 0367
7) 0380
8) 0401
9) 0599
10) 0000
11) 0000
12) 0000
13) 0000
14) 0000
15) 0000
16) 0000
17) 0000
18) 0000
19) 0000
20) 0000
21) 0000
22) 0000
23) 0000

Do you wish to delete this form? (Y or N) Y

In the above example the question numbers in the standard form are displayed and the user has to verify that the form is to be deleted.

Printing standard forms

The last item on the question maintenance menu is to print out the standard forms. This routine will print out the questions that correspond to the question numbers
entered for the standard forms. This is the form that the
standard forms will appear on the instructor's evaluation
form. To create the file of all of the standard forms run
the following routine.

Enter option : (M,S,D,A,P,N,X,L,?)

Output file is stan.rpt
Question 0005 not found

In the above example a printable version of the
standard forms was created. Note that one of the standard
forms used question number five and since that question was
deleted an error message was printed. The routine will put
a blank question into the file for any questions that it can
not find in the data base. To print out the standard
question file exit the question program and type the
following.

@PRINT STAN.RPT

The standard forms should be printed from time to time
to make sure that the correct question numbers where entered
for the standard forms. After the standard forms have been
printed the user should delete the file by typing the
following:

@STAN.RPT
This concludes the question maintenance routines. If the user has any troubles with the question files he should read the faculty evaluation system maintenance manual.
The faculty evaluation system is composed of a number of pascal and command routines. The programs and the files that they need are discussed in this manual.

The following pascal programs are used:
BLDIST - Enter instructor records
EDSTAT - Edit statistics files
PRPT1 - Print questionnaire forms
QUEST - Maintain question data base
REPORT - Generate evaluation reports
STATS - Convert scantron files to stats. files

System Maintainance Manual
The following command and control files are used:

LOGIN - Sets logical names
INSTRC - Runs BLDIST and submits sort
SRT - Sort control file
WRITE - Submits report print request

The only file that is needed to run the system initially is the question file QUEST.DTA. There are a large number of other files that are created and modified by the system. These files have to maintained each quarter for proper operation of the system.

System Installation

The system was set up to operate using a main directory and at least two subdirectories. The programs are installed in the main directory and the reports generated are written to the subdirectories.

Each of the subdirectories should be capable of storing two million permanent characters and double that number in working storage. The main directory should also be of comparable size. The directories should be named as described in the LOGIN command file. The logical names that are defined in the LOGIN file are used by the programs to write temporary files to the appropriate directory.
The SRT control file is submitted to sort the instructor records. This file will need to be updated if the user is running the system on a computer system other than the DEC20/60. The sort that the control file uses is the defined system sort. If one is not available, the user may have to write his own sort.

The WRITE command submits the evaluation reports to the printer. This routine may have to be modified if there are any changes to forms or if the user wants to use different switches.

After the user has created the directories, all of the Pascal programs have to be compiled and loaded. The sources may then be moved to some other storage area. The question file QUEST.DTA must be installed in the main area of the system.

All of the other files that the system uses will be either initialized or updated by the programs. Some of these files have to be updated each quarter.

File Maintainance

The files used by this system are in a variety of categories. Some of the files are used constantly, some are updated each quarter, and some are deleted after they are
The file QUEST.DTA is the data base for all of the questions. This file is never deleted, but may be modified each quarter between the time of running the reports from the previous quarter and entering the information for the instructors for the current quarter.

The file STAN.FRM is created when the user runs the QUEST program and enters a standard form. Once created this file need not be deleted. This file was set up to aid the user in entering information into the instructor record. By using the standard forms the user can enter a standard form number instead of typing in all of the questions that the instructor picked. This file may be modified as the user is using it.

The file INST.SRT is created by the BLDIST program. This file contains all of the instructor records. The file is created after the previous quarter's reports have been completed. This file must be kept until all of the reports have been created for the quarter for which the instructor information was entered. The file must be deleted before any instructor records are entered for the next quarter. In other words, this file should contain instructor information for one quarter only.
The report files that may be generated for the above file all end with the extension RPT. These files should be deleted as soon as the reports have been printed.

The questionnaire forms and the report forms are deleted automatically so the user does not have to worry about deleting them. The user should delete the command file PRCMD.CMD after it has been executed. This file is used to submit the print requests for the questionnaire forms so it is created again each time the routine PRPT1 is executed.

There are some files that must be deleted each quarter. These files are the batch number files and the old statistics files. The file BAT.NUM is used to keep track of the questionnaire file numbers. This file should be deleted at the beginning of each quarter before any questionnaires are generated. The file BATCH.NUM is used to keep track of the statistics files. This file should be deleted each quarter before the questionnaire answers are entered.

The statistics files are all located in one directory. Therefore the user may delete all of the files in this directory to get rid of the statistics files. The statistics files should be deleted after all of the reports have been generated for the instructors.
Other Errors

Most of the errors that the user may encounter are discussed in the User Manual. There are several system error messages that may occur at times.

When some of the programs are run for the first time an error message is printed that a particular file is not found. The user may ignore these messages since the programs will create these files.

The user will get an error if he is in the wrong directory and trying to run a program. The main directory is defined in the LOGIN file as FAC; the user should connect to this directory and try running the routine again.

The user will get an error message if he exceeds permanent storage and the programs will fail if they try to write a file into an area that is exceeding its working storage. If the user is exceeding permanent storage he should delete the report files and other temporary file described above. If a user exceeds working storage then he should either try to get more directory space or delete temporary files. The only time this error will cause major problems is if the user is over working storage in the statistics directory. If this happens the user should try to obtain more storage space for the directory. If the user
cannot obtain any more space then all of the reports should be generated for the existing statistics files and then the user can delete all of the statistics files.

The programs were designed to catch most of the errors that the user can make. If an error occurs that is not discussed in the manuals, write down the exact wording of the error and call somebody who has knowledge about the computer system.
LOGICAL-SYSTEM

Name Selection

Parameters: DB=LOGIC.DBF PRINT PUNCH=PSANAM.TMP EMPTY
SELECTION='TKEY="level-0"' ORDER=BYTYPE

1 course-number ELEMENT
2 department-abbreviation ELEMENT
3 department-number ELEMENT
4 form-number ELEMENT
5 frequency-of-answers ELEMENT
6 instructor-picked-quest-num ELEMENT
7 instructors-first-init ELEMENT
8 instructors-last-name ELEMENT
9 list-of-all-questions ELEMENT
10 mean-of-answers ELEMENT
11 num-quest-copies-request ELEMENT
12 question-answers ELEMENT
13 question-numbers ELEMENT
14 section-number ELEMENT
15 selected-questions ELEMENT
16 standard-dev-of-answers ELEMENT
17 standard-forms-list ELEMENT
18 instructor-information ENTITY
19 question-inst-info ENTITY
20 questions ENTITY
21 valid-idp-updates ENTITY
22 dept-quest-list INPUT
23 question-list-request INPUT
24 questionnaire-answers INPUT
25 questionnaire-request-form INPUT
26 faculty-evaluation-committee INTERFACE
27 instructor INTERFACE
28 students INTERFACE
29 global-questionnaire-report OUTPUT
30 invalid-quest-request OUTPUT
31 personal-questionnaire-report OUTPUT
32 question-list OUTPUT
33 questionnaire-forms OUTPUT
34 database-maintenance PROCESS
35 generate-questionnaire-form PROCESS
36 report-generation PROCESS
37 ices-data-pool SET
Parameters: DB=LOGIC.DBF FILE=PSANAM.TMP INDEX
NOPUNCHED-NAMES PRINT NOPUNCH SMARG=5 NMARG=20 AMARG=10
BMARG=25 RNMARG=60 CMARG=1 HMARG=23 ONE-PER-LINE
COMMENT NONEW-PAGE NONEW-LINE NOALL-STATEMENTS
COMPLEMENTARY-STATEMENTS LINE-NUMBERS PRINTEOF
DLC-COMMENT NOSORT-NAME-LIST

1 DEFINE ELEMENT course-number;
2 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
3 DESCRIPTION;
4 A University course number.
5 TRACE-KEY IS: 'level-0',
6 'level-1',
7 'level-1.1',
8 'level-2';
9 CONTAINED IN: instructor-information,
10 inst-general-info,
11 questionnaire-request-form,
12 global-questionnaire-report,
13 personal-questionnaire-report,
14 questionnaire-forms,
15 questionnaire-answers;
16 VALUES ARE:
17 1 THRU 999;
18
19 DEFINE ELEMENT department-abbreviation;
20 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
21 DESCRIPTION;
22 The department abbreviation is a character string
23 of letters. The length of the string is between 0 and
24 5 characters.
25 TRACE-KEY IS: 'level-0',
26 'level-1',
27 'level-1.1',
28 'level-2';
29 CONTAINED IN: instructor-information,
30 inst-general-info,
31 questionnaire-request-form,
32 global-questionnaire-report,
33 personal-questionnaire-report,
34 questionnaire-forms;
35
36 DEFINE ELEMENT department-number;
37 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
38 DESCRIPTION;
39 The department number is a number used to identify
40 a particular university department;
LOGICAL-SYSTEM

Formatted Problem Statement

43 TRACE-KEY IS: 'level-0',
44  'level-1',
45  'level-1.1',
46  'level-2';
47 CONTAINED IN: instructor-information,
48  inst-general-info,
49  questionnaire-request-form,
50  global-questionnaire-report,
51  personal-questionnaire-report,
52  questionnaire-forms,
53  questionnaire-answers;
55 VALUES ARE:
56  1 THRU 9999;
57
58 DEFINE ELEMENT form-number;
59 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
60 DESCRIPTION;
61 The form number is used to obtain the question numbers for questions in a given standard form. The standard for...
64 TRACE-KEY IS: 'level-0',
65  'level-1',
66  'level-1.1';
67 CONTAINED IN: inst-general-info,
68  questionnaire-request-form,
69  valid-idp-updates,
70  standard-form-quest-numbers;
71 VALUES ARE:
72  1 THRU 999;
73
74 DEFINE ELEMENT frequency-of-answers;
75 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
76 DESCRIPTION;
77 The frequency of the answers is a list of the number of occurrences for each answer to a given question.;
79 TRACE-KEY IS: 'level-0',
80  'level-2';
81 CONTAINED IN: questionnaire-answer-stats,
82  global-questionnaire-report,
83  personal-questionnaire-report;
84
85 DEFINE ELEMENT instructor-picked-quest-num;
86 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
87 DESCRIPTION;
88 The instructor picked question numbers are a list of numbers to questions in the question data base. The inst...
90 chooses up to 23 questions from the data base and enters ... 91 numbers on his request form.;
LOGICAL-SYSTEM

Formatted Problem Statement

92 TRACE-KEY IS: 'level-0',
93  'level-1',
94  'level-1.1',
95  'level-2';
96 CONTAINED IN: instructor-information,
97  question-inst-info,
98  questionnaire-request-form;
99
100 DEFINE ELEMENT instructors-first-init;
101 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
102 TRACE-KEY IS: 'level-0',
103  'level-1',
104  'level-1.1',
105  'level-2';
106 CONTAINED IN: instructor-information,
107  inst-general-info,
108  question-inst-info,
109  questionnaire-request-form,
110  global-questionnaire-report,
111  personal-questionnaire-report,
112  questionnaire-forms;
113
114 DEFINE ELEMENT instructors-last-name;
115 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
116 DESCRIPTION;
117 The instructor's name is the first 14 characters in
118 his last name.;
119 TRACE-KEY IS: 'level-0',
120  'level-1',
121  'level-1.1',
122  'level-2';
123 CONTAINED IN: instructor-information,
124  inst-general-info,
125  question-inst-info,
126  questionnaire-request-form,
127  global-questionnaire-report,
128  personal-questionnaire-report,
129  questionnaire-forms;
130
131 DEFINE ELEMENT list-of-all-questions;
132 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
133 DESCRIPTION;
134 This is a list of all of the questions that are
135 kept in the data base. The questions are in three parts,
136 a question number, and two string parts each containing
137 half of a question.;
138 TRACE-KEY IS: 'level-0',
139  'level-3';
140 CONTAINED IN: questions,
141 question-list;

143 DEFINE ELEMENT mean-of-answers;
144 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
145 DESCRIPTION;
146 The mean of answers will consist of two parts.
147 The first part is the mean of the answers to a particular
148 question. The second part is a written description telling
149 whether the mean is high, high average, average, low
150 average, or low. The second part was added to help
151 instructors interpret the results.;
152 TRACE-KEY IS: 'level-0',
153 'level-2';
154 CONTAINED IN: questionnaire-answer-stats,
155 global-questionnaire-report,
156 personal-questionnaire-report;

158 DEFINE ELEMENT num-quest-copies-request;
159 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
160 DESCRIPTION;
161 The number of questionnaire copies are the number of
162 forms that are requested to be printed.;
163 TRACE-KEY IS: 'level-0',
164 'level-1',
165 'level-1.1';
166 CONTAINED IN: instructor-information,
167 inst-general-info,
168 question-inst-info,
169 questionnaire-request-form;

171 DEFINE ELEMENT question-answers;
172 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
173 DESCRIPTION;
174 This is a list of symbols telling which answers
175 were picked for the questions in the questionnaire.;
176 TRACE-KEY IS: 'level-0',
177 'level-2';
178 CONTAINED IN: questionnaire-answers;

180 DEFINE ELEMENT question-numbers;
181 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
182 DESCRIPTION;
183 The question numbers are the numbers for the questions
184 in the data base.;
185 TRACE-KEY IS: 'level-0',
186 'level-1.1',
187 'level-3';
188 CONTAINED IN: dept-quest-list,
189 valid-idp-updates,
Formatted Problem Statement

190  VALUES ARE: standard-form-quest-numbers;
191            1 THRU 9999;
193
194  DEFINE ELEMENT section-number;
196  DESCRIPTION;
197  The section number is the section number for a given
198  university course.;
199  TRACE-KEY IS: 'level-0',
200        'level-1',
201        'level-1.1',
202        'level-2';
203  CONTAINED IN: instructor-information,
204       inst-general-info,
205       question-inst-info,
206       questionnaire-request-form,
207       global-questionnaire-report,
208       personal-questionnaire-report,
209       questionnaire-forms,
210       questionnaire-answers;
211  VALUES ARE:
212            1 THRU 999;
213
214  DEFINE ELEMENT selected-questions;
216  DESCRIPTION;
217  The selected questions are a list of up to
218  23 question numbers. These will be used to obtain the
219  corresponding questions from the data base. The questions
220  are either instructor picked or standard form questions.;
221  TRACE-KEY IS: 'level-0',
222        'level-1',
223        'level-2';
224  CONTAINED IN: question-inst-info,
225       questionnaire-forms,
226       personal-questionnaire-report;
227
228  DEFINE ELEMENT standard-dev-of-answers;
230  DESCRIPTION;
231  The standard deviation of the answers is in two
232  parts. The first part lists the standard deviation of
233  a particular question. The second part uses the standard
234  deviation to list weather the consensus on the answer was,
235  high, average, or low.;
236  TRACE-KEY IS: 'level-0',
237        'level-2';
238  CONTAINED IN: questionnaire-answer-stats,
Formatted Problem Statement

239  global-questionnaire-report,
240  personal-questionnaire-report;
241
242 DEFINE ELEMENT standard-forms-list;
244 DESCRIPTION;
245  This form will be a list of all standard forms
246  currently in the data base. It will list out the questions
247  in each of the standard forms.;
248  TRACE-KEY IS: 'level-0',
249  'level-3';
250  CONTAINED IN: questions,
251  question-list;
252
253 DEFINE ENTITY instructor-information;
254  /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
255  TRACE-KEY IS: 'level-0',
256  'level-1',
257  'level-1.1';
258  CONSISTS OF:
259  course-number,
260  department-abbreviation,
261  department-number,
262  instructor-picked-quest-num,
263  instructors-first-init,
264  instructors-last-name,
265  num-quest-copies-request,
266  section-number;
267  DERIVED BY: generate-questionnaire-form
268  USING: question-inst-info,
269  questionnaire-request-form;
270  DERIVED BY: get-standard-form
271  USING: standard-form-quest-numbers,
272  inst-general-info;
273  DERIVED BY: create-instructor-database
274  USING: questionnaire-request-form;
275  DERIVED BY: get-inst-picked-quests
276  USING: inst-general-info,
277  questionnaire-request-form;
278  USED BY:
279  generate-questionnaire-form
280  TO UPDATE ices-data-pool;
281
282 DEFINE ENTITY question-inst-info;
283  /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
284  TRACE-KEY IS: 'level-0',
285  'level-1',
286  'level-2';
287  COLLECTED IN: ices-data-pool;
LOGICAL-SYSTEM

Formatted Problem Statement

CONSISTS OF:

course-number,
department-abbreviation,
department-number,
instructor-picked-quest-num,
instructors-first-init,
instructors-last-name,
um-quest-copies-request,
section-number,
selected-questions;

USED BY:
generate-questionnaire-form
TO DERIVE instructor-information;

USED BY:
generate-questionnaire-form
TO DERIVE invalid-quest-request;

USED BY:
generate-questionnaire-form
TO DERIVE questionnaire-forms;

USED BY:
report-generation
TO DERIVE personal-questionnaire-report;

USED BY:
report-generation
TO DERIVE global-questionnaire-report;

USED BY:
questionnaire-print-routine
TO DERIVE questionnaire-forms;

USED BY:
print-reports
TO DERIVE personal-questionnaire-report;

USED BY:
print-reports
TO DERIVE global-questionnaire-report;

DEFINE ENTITY questions;

/* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */

TRACE-KEY IS: 'level-0',
'level-3';

COLLECTED IN: ices-data-pool;

CONSISTS OF:
list-of-all-questions,
standard-forms-list;

USED BY:
database-maintainance
TO DERIVE question-list;

USED BY:
print-questions
TO DERIVE question-list;
LOGICAL-SYSTEM

Formatted Problem Statement

337 DEFINE ENTITY valid-idp-updates;
338 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
339 TRACE-KEY IS: 'level-0',
340 'level-3';
341 CONSISTS OF:
342 form-number,
343 question-numbers;
344 DERIVED BY: database-maintainance
345 USING: dept-quest-list;
346 DERIVED BY: update-questions
347 USING: dept-quest-list;
348 USED BY:
349 database-maintainance
350 TO UPDATE ices-data-pool;
351
352 DEFINE INPUT dept-quest-list;
353 /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
354 TRACE-KEY IS: 'level-0',
355 'level-3';
356 GENERATED: BY faculty-evaluation-committee;
357 CONSISTS OF:
358 question-numbers;
359 USED BY:
360 database-maintainance
361 TO DERIVE valid-idp-updates;
362 USED BY:
363 update-questions
364 TO DERIVE valid-idp-updates;
365
366 DEFINE INPUT question-list-request;
367 /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
368 TRACE-KEY IS: 'level-0',
369 'level-3';
370 GENERATED: BY instructor;
371 USED BY:
372 database-maintainance
373 TO DERIVE question-list;
374 USED BY:
375 print-questions
376 TO DERIVE question-list;
377
378 DEFINE INPUT questionnaire-answers;
379 /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
380 TRACE-KEY IS: 'level-0',
381 'level-2';
382 GENERATED: BY students;
383 CONSISTS OF:
384 course-number,
LOGICAL-SYSTEM

Formatted Problem Statement

department-number,
question-answers,
section-number;
USED BY:
generate-statistics
TO DERIVE questionnaire-answer-stats;
USED BY:
report-generation
TO DERIVE personal-questionnaire-report;
USED BY:
report-generation
TO DERIVE global-questionnaire-report;
DEFINE INPUT questionnaire-request-form;

/* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
TRACE-KEY IS: 'level-0',
'level-1',
'level-1.1';
GENERATED: BY instructor;
CONSISTS OF:
course-number,
department-abbreviation,
department-number,
form-number,
instructor-picked-quest-num,
instructors-first-init,
instructors-last-name,
um-num-quest-copies-request,
section-number;
USED BY:
generate-questionnaire-form
TO DERIVE questionnaire-forms;
USED BY:
create-instructor-database
TO DERIVE invalid-quest-request;
USED BY:
create-instructor-database
TO DERIVE instructor-information;
USED BY:
generate-questionnaire-form
TO DERIVE instructor-information;
USED BY:
generate-questionnaire-form
TO DERIVE invalid-quest-request;
USED BY:
get-inst-general-info
TO DERIVE inst-general-info;
USED BY:
get-inst-general-info
LOGICAL-SYSTEM

Formatted Problem Statement

435 TO DERIVE invalid-quest-request;
436 USED BY:
437 get-inst-picked-quests
438 TO DERIVE instructor-information;
439
440 DEFINE INTERFACE faculty-evaluation-committee;
441 /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
442 TRACE-KEY IS: 'level-0';
443 GENERATES:
444 dept-quest-list;
445 RECEIVES:
446 global-questionnaire-report;
447
448 DEFINE INTERFACE instructor;
449 /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
450 TRACE-KEY IS: 'level-0';
451 GENERATES:
452 question-list-request,
453 questionnaire-request-form;
454 RECEIVES:
455 question-list,
456 invalid-quest-request,
457 personal-questionnaire-report;
458
459 DEFINE INTERFACE students;
460 /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
461 TRACE-KEY IS: 'level-0';
462 GENERATES:
463 questionnaire-answers;
464 RECEIVES:
465 questionnaire-forms;
466
467 DEFINE OUTPUT global-questionnaire-report;
468 /* DATE OF LAST CHANGE - Apr 21, 1983, 22:59:46 */
469 DESCRIPTION;
470 The global questionnaire report will contain
471 the general information about the instructor. The questions
472 that will be in the report will be the demographic questions
473 about the students and the general questions about the
474 instructor. The report will also list the frequency of
475 the question answers, and the mean and standard deviation
476 for each question;
477 TRACE-KEY IS: 'level-0',
478 'level-2';
479 RECEIVED: BY faculty-evaluation-committee;
480 CONSISTS OF:
481 course-number,
482 department-abbreviation,
483 department-number,
LOGICAL-SYSTEM

Formatted Problem Statement

frequency-of-answers,
instructors-first-init,
instructors-last-name,
mean-of-answers,
section-number,
standard-dev-of-answers;

DERIVED BY: report-generation
USING: question-inst-info,
questionnaire-answers;

DERIVED BY: print-reports
USING: question-inst-info;

DERIVED BY: print_reports
USING: questionnaire-answer-stats;

DEFINE OUTPUT invalid-quest-request;

/* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
TRACE-KEY IS: 'level-0',
'level-1',
'level-1.1';
RECEIVED: BY instructor;
DERIVED BY: generate-questionnaire-form
USING: question-inst-info,
questionnaire-request-form;
DERIVED BY: create-instructor-database
USING: questionnaire-request-form;
DERIVED BY: get-inst-general-info
USING: questionnaire-request-form;

DEFINE OUTPUT personal-questionnaire-report;

/* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
DESCRIPTION;
The personal questionnaire report will contain the
instructor's general information. The questions that will
be listed in the report are the demographic questions about
the students, the general questions about the instructor,
and the instructor's picked questions. For each question
a frequency of the occurrence of the answers will be given,
along with the mean and standard deviation of the question.
A written description of the mean and standard deviation ...
be given to help the instructor interpret the results.;
TRACE-KEY IS: 'level-0',
'level-2';
RECEIVED: BY instructor;
CONSISTS OF:
course-number,
department-abbreviation,
department-number,
frequency-of-answers,
instructors-first-init,
Formulated Problem Statement

instructors-last-name,
mean-of-answers,
section-number,
selected-questions,
standard-dev-of-answers;

DERIVED BY: report-generation
USING: question-inst-info,
questionnaire-answers;

DERIVED BY: print-reports
USING: question-inst-info;

DEFINE OUTPUT question-list;
/* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
TRACE-KEY IS: "level-0",
"level-3";
RECEIVED: BY instructor;
CONSISTS OF:
list-of-all-questions,
standard-forms-list;

DERIVED BY: database-maintainance
USING: questions,
question-list-request;

DERIVED BY: print-questions
USING: questions,
question-list-request;

DEFINE OUTPUT questionnaire-forms;
/* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
DESCRIPTION;
The questionnaire-forms will have the demographic
questions about the students and the instructor's general
questions preprinted on them. The forms will also have
a scantron readable section along the right boarder. The
general information about the instructor and his personal
questions will be printed on the form.;

TRACE-KEY IS: "level-0",
"level-1";
RECEIVED: BY students;
CONSISTS OF:
course-number,
department-abbreviation,
department-number,
instructors-first-init,
instructors-last-name,
section-number,
selected-questions;

DERIVED BY: generate-questionnaire-form
USING: question-inst-info,
questionnaire-request-form;
LOGICAL-SYSTEM

Formatted Problem Statement

582  DERIVED BY:  questionnaire-print-routine
583  USING:  question-inst-info;
584  
585  DEFINE PROCESS  database-maintainance;
586  /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
587  SYNONYMS ARE:  dfd3;
588  TRACE-KEY IS:  'level-0';
589  SUBPARTS ARE:  print-questions,
590  update-questions;
591  DERIVES:  question-list
592  USING:  questions,
593  question-list-request;
594  DERIVES:  valid-idp-updates
595  USING:  dept-quest-list;
596  UPDATES:  ices-data-pool
597  USING:  valid-idp-updates;
598  
599  DEFINE PROCESS  generate-questionnaire-form;
600  /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
601  SYNONYMS ARE:  dfd1;
602  TRACE-KEY IS:  'level-0';
603  SUBPARTS ARE:  create-instructor-database,
604  questionnaire-print-routine;
605  DERIVES:  instructor-information
606  USING:  question-inst-info,
607  questionnaire-request-form;
608  DERIVES:  invalid-question-request
609  USING:  question-inst-info,
610  questionnaire-request-form;
611  DERIVES:  questionnaire-forms
612  USING:  question-inst-info,
613  questionnaire-request-form;
614  UPDATES:  ices-data-pool
615  USING:  instructor-information;
616  
617  DEFINE PROCESS  report-generation;
618  /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
619  SYNONYMS ARE:  dfd2;
620  TRACE-KEY IS:  'level-0';
621  SUBPARTS ARE:  generate-statistics,
622  print-reports;
623  DERIVES:  personal-questionnaire-report
624  USING:  question-inst-info,
625  questionnaire-answers;
626  DERIVES:  global-questionnaire-report
627  USING:  question-inst-info,
628  questionnaire-answers;
629  
630  DEFINE SET  ices-data-pool;
logical-system

formatted problem statement

631 /* date of last change - apr 21, 1983, 19:37:15 */
632 trace-key is: 'level-0';
633 collection of:
634 question-inst-info,
635 questions,
636 standard-form-quest-numbers;
637 updated by: database-maintainance
638 using: valid-idp-updates;
639 updated by: generate-questionnaire-form
640 using: instructor-information;
641
642 eof eof eof eof eof
Index

1 course-number
2 create-instructor-database
3 database-maintenance
4 department-abbreviation
5 department-number
6 dept-quest-list
7 dfd1
8 dfd2
9 dfd3
10 faculty-evaluation-committee
11 form-number
12 frequency-of-answers
13 generate-questionnaire-form
14 generate-statistics
15 get-inst-general-info
16 get-inst-picked-quests
17 get-standard-form
18 global-questionnaire-report
19 ices-data-pool
<table>
<thead>
<tr>
<th>Index Code</th>
<th>Description</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>inst-general-info</td>
<td>2( 2), 3( 2), 4( 2), 5, 6, 7( 2), 10</td>
</tr>
<tr>
<td>21</td>
<td>instructor</td>
<td>9, 10, 11, 12( 2), 13</td>
</tr>
<tr>
<td>22</td>
<td>instructor-information</td>
<td>2( 2), 3, 4( 3), 5, 6, 7, 8, 10( 2), 11, 14( 2), 15</td>
</tr>
<tr>
<td>23</td>
<td>instructor-picked-quest-num</td>
<td>3, 7, 8, 10</td>
</tr>
<tr>
<td>24</td>
<td>instructors-first-init</td>
<td>4, 7, 8, 10, 12( 2), 13</td>
</tr>
<tr>
<td>25</td>
<td>instructors-last-name</td>
<td>4, 7, 8, 10, 12, 13( 2)</td>
</tr>
<tr>
<td>26</td>
<td>invalid-quest-request</td>
<td>8, 10( 2), 11( 2), 12, 14</td>
</tr>
<tr>
<td>27</td>
<td>list-of-all-questions</td>
<td>4, 8, 13</td>
</tr>
<tr>
<td>28</td>
<td>mean-of-answers</td>
<td>5, 12, 13</td>
</tr>
<tr>
<td>29</td>
<td>num-quest-copies-request</td>
<td>5, 7, 8, 10</td>
</tr>
<tr>
<td>30</td>
<td>personal-questionnaire-report</td>
<td>2( 2), 3( 2), 4( 2), 5, 6( 2), 7, 8( 2), 10, 11, 12, 13, 14</td>
</tr>
<tr>
<td>31</td>
<td>print-questions</td>
<td>8, 9, 13, 14</td>
</tr>
<tr>
<td>32</td>
<td>print-reports</td>
<td>8( 2), 12, 13, 14</td>
</tr>
<tr>
<td>33</td>
<td>print_reports</td>
<td>12</td>
</tr>
<tr>
<td>34</td>
<td>question-answers</td>
<td>5, 10</td>
</tr>
<tr>
<td>35</td>
<td>question-inst-info</td>
<td>2( 2), 3, 4( 3), 5, 6( 2), 7( 2), 12( 3), 13( 3), 14( 6), 15</td>
</tr>
<tr>
<td>36</td>
<td>question-list</td>
<td>5, 7, 8( 2), 10</td>
</tr>
</tbody>
</table>
Index

37 question-list-request 9, 11, 13( 2), 14
38 question-numbers 5, 9( 2)
39 questionnaire-answer-stats 3, 5, 6, 10, 12
40 questionnaire-answers 2, 3, 5, 6, 9, 11, 12, 13, 14( 2)
41 questionnaire-forms 2( 2), 3, 4( 2), 6( 2), 8( 2), 10, 11, 13, 14
42 questionnaire-print-routine 8, 14( 2)
43 questionnaire-request-form 2( 2), 3( 2), 4( 3), 5, 6, 7( 3), 10, 11, 12( 3), 13, 14( 3)
44 questions 4, 7, 8, 13( 2), 14, 15
45 report-generation 8( 2), 10( 2), 12, 13, 14
46 section-number 6, 7, 8, 10( 2), 12, 13( 2)
47 selected-questions 6, 8, 13( 2)
48 standard-dev-of-answers 6, 12, 13
49 standard-form-quest-numbers 3, 6, 7, 15
50 standard-forms-list 7, 8, 13
51 students 9, 11, 13
52 update-questions 9( 2), 14
53 valid-idp-updates 3, 5, 9( 3), 14( 2), 15
LOGICAL-SYSTEM

Name Selection

Parameters: DB=LOGIC.DBF PRINT PUNCH=PSANAM.TMP EMPTY
SELECTION='TKEY="level-1"' ORDER=BYTYPE

1 course-number ELEMENT
2 department-abbreviation ELEMENT
3 department-number ELEMENT
4 form-number ELEMENT
5 instructor-picked-quest-num ELEMENT
6 instructors-first-init ELEMENT
7 instructors-last-name ELEMENT
8 num-quest-copies-request ELEMENT
9 section-number ELEMENT
10 selected-questions ELEMENT
11 instructor-information ENTITY
12 question-inst-info ENTITY
13 questionnaire-request-form INPUT
14 invalid-quest-request OUTPUT
15 questionnaire-forms OUTPUT
16 create-instructor-database PROCESS
17 questionnaire-print-routine PROCESS
Diagram 1

Create-instructor-database (1.1)

Invalid-quesrequest → questionnaire-request-form

questionnaire-print-routine (1.2)

questionnaire-form

instructor-information

questionnaire-install-info
Formed Problem Statement

Parameters: DB=LOGIC.DBF  FILE=PSANAM.TMP  INDEX
           NOPUNCHED-NAMES  PRINT  NOPUNCH  SMARG=5  NMARG=20  AMARG=10
           BMARG=25  RNMARG=60  CMARG=1  HMARG=23  ONE-PER-LINE
           COMMENT  NONEW-PAGE  NONEW-LINE  NOALL-STATEMENTS
           COMPLEMENTARY-STATEMENTS  LINE-NUMBERS  PRINTEOF
           DLC-COMMENT  NOSORT-NAME-LIST

1  DEFINE ELEMENT  course-number;
   /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
2   DESCRIPTION;
3   A University course number.;
4   TRACE-KEY IS: 'level-0',
5      'level-1',
6      'level-1.1',
7      'level-2';
8   CONTAINED IN: instructor-information,
9      inst-general-info,
10     question-inst-info,
11     questionnaire-request-form,
12     global-questionnaire-report,
13     personal-questionnaire-report,
14     questionnaire-forms,
15     questionnaire-answers;
16   VALUES ARE:
17      1 THRU 999;
18
19  DEFINE ELEMENT  department-abbreviation;
20   /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
21   DESCRIPTION;
22   The department abbreviation is a character string
23      of letters. The length of the string is between 0 and
24      5 characters.;
25   TRACE-KEY IS: 'level-0',
26      'level-1',
27      'level-1.1',
28      'level-2';
29   CONTAINED IN: instructor-information,
30      inst-general-info,
31      question-inst-info,
32      questionnaire-request-form,
33      global-questionnaire-report,
34      personal-questionnaire-report,
35      questionnaire-forms;
36
37  DEFINE ELEMENT  department-number;
38   /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
39   DESCRIPTION;
40   The department number is a number used to identify
41   a particular university department.;
FORMATTED PROBLEM STATEMENT

43 TRACE-KEY IS: 'level-0',
44 'level-1',
45 'level-1.1',
46 'level-2';
47 CONTAINED IN: instructor-information,
48 inst-general-info,
49 question-inst-info,
50 questionnaire-request-form,
51 global-questionnaire-report,
52 personal-questionnaire-report,
53 questionnaire-forms,
54 questionnaire-answers;
55 VALUES ARE:
56 1 THRU 9999;
57
58 DEFINE ELEMENT form-number;
59 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
60 DESCRIPTION;
61 The form number is used to obtain the question numbers
62 for questions in a given standard form. The standard form
63 be a list of up to 23 question numbers.;
64 TRACE-KEY IS: 'level-0',
65 'level-1',
66 'level-1.1';
67 CONTAINED IN: inst-general-info,
68 questionnaire-request-form,
69 valid-idp-updates,
70 standard-form-quest-numbers;
71 VALUES ARE:
72 1 THRU 999;
73
74 DEFINE ELEMENT instructor-picked-quest-num;
75 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
76 DESCRIPTION;
77 The instructor picked question numbers are a list of
78 numbers to questions in the question data base. The inst...
79 chooses up to 23 questions from the data base and enters ... 
80 numbers on his request form.;
81 TRACE-KEY IS: 'level-0',
82 'level-1',
83 'level-1.1',
84 'level-2';
85 CONTAINED IN: instructor-information,
86 question-inst-info,
87 questionnaire-request-form;
88
89 DEFINE ELEMENT instructors-first-init;
90 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
91 TRACE-KEY IS: 'level-0',
LOGICAL-SYSTEM

Formatted Problem Statement

92 'level-1',
93 'level-1.1',
94 'level-2';
95 CONTAINED IN: instructor-information,
96 inst-general-info,
97 question-inst-info,
98 questionnaire-request-form,
99 global-questionnaire-report,
100 personal-questionnaire-report,
101 questionnaire-forms;

102 DEFINE ELEMENT instructors-last-name;
103 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
104 DESCRIPTION;
105 The instructor's name is the first 14 characters in
106 his last name.;
107 TRACE-KEY IS: 'level-0',
108 'level-1',
109 'level-1.1',
110 'level-2';
111 CONTAINED IN: instructor-information,
112 inst-general-info,
113 question-inst-info,
114 questionnaire-request-form,
115 global-questionnaire-report,
116 personal-questionnaire-report,
117 questionnaire-forms;

119 DEFINE ELEMENT num-quest-copies-request;
120 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
121 DESCRIPTION;
122 The number of questionnaire copies are the number of
123 forms that are requested to be printed.;
124 TRACE-KEY IS: 'level-0',
125 'level-1',
126 'level-1.1';
127 CONTAINED IN: instructor-information,
128 inst-general-info,
129 question-inst-info,
130 questionnaire-request-form;

132 DEFINE ELEMENT section-number;
134 DESCRIPTION;
135 The section number is the section number for a given
136 university course.;
137 TRACE-KEY IS: 'level-0',
138 'level-1',
139 'level-1.1',
LOGICAL-SYSTEM

Formatted Problem Statement

141 'level-2';
142 CONTAINED IN: instructor-information,
143 inst-general-info,
144 question-inst-info,
145 questionnaire-request-form,
146 global-questionnaire-report,
147 personal-questionnaire-report,
148 questionnaire-forms,
149 questionnaire-answers;
150 VALUES ARE:
151 1 THRU 999;
152
153 DEFINE ELEMENT selected-questions;
155 DESCRIPTION;
156 The selected questions are a list of up to
157 23 question numbers. These will be used to obtain the
158 corresponding questions from the data base. The questions
159 are either instructor picked or standard form questions.;
160 TRACE-KEY IS: 'level-0',
161 'level-1',
162 'level-2';
163 CONTAINED IN: question-inst-info,
164 questionnaire-forms,
165 personal-questionnaire-report;
166
167 DEFINE ENTITY instructor-information;
168 /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
169 TRACE-KEY IS: 'level-0',
170 'level-1',
171 'level-1.1';
172 CONSISTS OF:
173 course-number,
174 department-abbreviation,
175 department-number,
176 instructor-picked.quest-num,
177 instructors-first-init,
178 instructors-last-name,
179 num-quest-copies-request,
180 section-number;
181 DERIVED BY: generate-questionnaire-form
182 USING: question-inst-info,
183 questionnaire-request-form;
184 DERIVED BY: get-standard-form
185 USING: standard-form-quest-numbers,
186 inst-general-info;
187 DERIVED BY: create-instructor-database
188 USING: questionnaire-request-form;
189 DERIVED BY: get-inst-picked-quests
FORMATTED PROBLEM STATEMENT

190 USING: inst-general-info,
191 questionnaire-request-form;
192 USED BY:
193 generate-questionnaire-form
194 TO UPDATE ices-data-pool;
196 DEFINE ENTITY question-inst-info;
197 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
198 TRACE-KEY IS: ‘level-0’,
199 ‘level-1’,
200 ‘level-2’;
201 COLLECTED IN: ices-data-pool;
202 CONSISTS OF:
203 course-number,
204 department-abbreviation,
205 department-number,
206 instructor-picked-quest-num,
207 instructors-first-init,
208 instructors-last-name,
209 num-quest-copies-request,
210 section-number,
211 selected-questions;
212 USED BY:
213 generate-questionnaire-form
214 TO DERIVE instructor-information;
215 USED BY:
216 generate-questionnaire-form
217 TO DERIVE invalid-quest-request;
218 USED BY:
219 generate-questionnaire-form
220 TO DERIVE questionnaire-forms;
221 USED BY:
222 report-generation
223 TO DERIVE personal-questionnaire-report;
225 USED BY:
226 report-generation
227 TO DERIVE global-questionnaire-report;
228 USED BY:
229 questionnaire-print-routine
230 TO DERIVE questionnaire-forms;
231 print-reports
232 TO DERIVE personal-questionnaire-report;
233 USED BY:
234 print-reports
235 TO DERIVE global-questionnaire-report;
237 DEFINE INPUT questionnaire-request-form;
238 /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
LOGICAL-SYSTEM

Formatted Problem Statement

TRACE-KEY IS: 'level-0',
'level-1',
'level-1.1';
GENERATED: BY instructor;
CONSISTS OF:
course-number,
department-abbreviation,
department-number,
form-number,
instructor-picked-quest-num,
instructors-first-init,
instructors-last-name,
num-quest-copies-request,
section-number;
USED BY:
generate-questionnaire-form
TO DERIVE questionnaire-forms;
create-instructor-database
TO DERIVE invalid-quest-request;
create-instructor-database
TO DERIVE instructor-information;
generate-questionnaire-form
TO DERIVE instructor-information;
generate-questionnaire-form
TO DERIVE invalid-quest-request;
get-inst-general-info
TO DERIVE inst-general-info;
get-inst-general-info
TO DERIVE invalid-quest-request;
get-inst-picked-quests
TO DERIVE instructor-information;
DEFINE OUTPUT invalid-quest-request;

/ * DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
TRACE-KEY IS: 'level-0',
'level-1',
'level-1.1';
RECEIVED: BY instructor;
DERIVED BY: generate-questionnaire-form
USING: question-inst-info,
questionnaire-request-form;
DERIVED BY: create-instructor-database
LOGICAL-SYSTEM

Formatted Problem Statement

288 USING: questionnaire-request-form;
289 DERIVED BY: get-inst-general-info
290 USING: questionnaire-request-form;
291
292 DEFINE OUTPUT questionnaire-forms;
293 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
294 DESCRIPTION;
295 The questionnaire-forms will have the demographic
296 questions about the students and the instructor's general
297 questions preprinted on them. The forms will also have
298 a scantron readable section along the right boarder. The
299 general information about the instructor and his personal
300 questions will be printed on the form.;
301 TRACE-KEY IS: 'level-0',
302 'level-1';
303 RECEIVED: BY students;
304 CONSISTS OF:
305 course-number,
306 department-abbreviation,
307 department-number,
308 instructors-first-init,
309 instructors-last-name,
310 section-number,
311 selected-questions;
312 DERIVED BY: generate-questionnaire-form
313 USING: question-inst-info,
314 questionnaire-request-form;
315 DERIVED BY: questionnaire-print-routine
316 USING: question-inst-info;
317
318 DEFINE PROCESS create-instructor-database;
319 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
320 SYNONYMS ARE: dfdl.1;
321 TRACE-KEY IS: 'level-1';
322 SUBPARTS ARE: get-inst-general-info,
323 get-standard-form,
324 get-inst-picked-quests;
325 PART OF: generate-questionnaire-form;
326 DERIVES: invalid-quest-request
327 USING: questionnaire-request-form;
328 DERIVES: instructor-information
329 USING: questionnaire-request-form;
330
331 DEFINE PROCESS questionnaire-print-routine;
332 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
333 SYNONYMS ARE: dfdl.2;
334 DESCRIPTION;
335 This process will use the information given it
336 to produce the requested number of copies of the
Formatted Problem Statement

337 questionnaire forms.;
338 TRACE-KEY IS: 'level-1';
339 PART OF: generate-questionnaire-form;
340 DERIVES: questionnaire-forms
341 USING: question-inst-info;
342
343 EOF EOF EOF EOF EOF
### Index

<table>
<thead>
<tr>
<th>Course/Module</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 course-number</td>
<td>20, 23, 24, 25, 26</td>
</tr>
<tr>
<td>2 create-instructor-database</td>
<td>23, 25(3), 26</td>
</tr>
<tr>
<td>3 department-abbreviation</td>
<td>20, 23, 24, 25, 26</td>
</tr>
<tr>
<td>4 department-number</td>
<td>20, 23, 24, 25, 26</td>
</tr>
<tr>
<td>5 dfdl.1</td>
<td>26</td>
</tr>
<tr>
<td>6 dfdl.2</td>
<td>26</td>
</tr>
<tr>
<td>7 form-number</td>
<td>21, 25</td>
</tr>
<tr>
<td>8 generate-questionnaire-form</td>
<td>23, 24(4), 25(4), 26(2), 27</td>
</tr>
<tr>
<td>9 get-inst-general-info</td>
<td>25(2), 26(2)</td>
</tr>
<tr>
<td>10 get-inst-picked-quests</td>
<td>23, 25, 26</td>
</tr>
<tr>
<td>11 get-standard-form</td>
<td>23, 26</td>
</tr>
<tr>
<td>12 global-questionnaire-report</td>
<td>20(2), 21, 22(2), 23, 24(2)</td>
</tr>
<tr>
<td>13 ices-data-pool</td>
<td>24(2)</td>
</tr>
<tr>
<td>14 inst-general-info</td>
<td>20(2), 21(2), 22(3), 23(2), 24, 25</td>
</tr>
<tr>
<td>15 instructor</td>
<td>25(2)</td>
</tr>
<tr>
<td>16 instructor-information</td>
<td>20(2), 21(2), 22(3), 23(2), 24, 25(3), 26</td>
</tr>
<tr>
<td>17 instructor-picked-quest-num</td>
<td>21, 23, 24, 25</td>
</tr>
<tr>
<td>18 instructors-first-init</td>
<td>21, 23, 24, 25, 26</td>
</tr>
<tr>
<td>19 instructors-last-name</td>
<td>22, 23, 24, 25, 26</td>
</tr>
</tbody>
</table>
## Index

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>invalid-quest-request</td>
<td>24, 25(4), 26</td>
</tr>
<tr>
<td>21</td>
<td>num-quest-copies-request</td>
<td>22, 23, 24, 25</td>
</tr>
<tr>
<td>22</td>
<td>personal-questionnaire-report</td>
<td>20(2), 21, 22(2), 23(2), 24(2)</td>
</tr>
<tr>
<td>23</td>
<td>print-reports</td>
<td>24(2)</td>
</tr>
<tr>
<td>24</td>
<td>question-inst-info</td>
<td>20(2), 21(2), 22(3), 23(3), 24, 25, 26(2), 27</td>
</tr>
<tr>
<td>25</td>
<td>questionnaire-answers</td>
<td>20, 21, 23</td>
</tr>
<tr>
<td>26</td>
<td>questionnaire-forms</td>
<td>20(2), 21, 22(2), 23(2), 24(2), 25, 26, 27</td>
</tr>
<tr>
<td>27</td>
<td>questionnaire-print-routine</td>
<td>24, 26(2)</td>
</tr>
<tr>
<td>28</td>
<td>questionnaire-request-form</td>
<td>20(2), 21(3), 22(3), 23(3), 24(2), 25, 26(5)</td>
</tr>
<tr>
<td>29</td>
<td>report-generation</td>
<td>24(2)</td>
</tr>
<tr>
<td>30</td>
<td>section-number</td>
<td>22, 23, 24, 25, 26</td>
</tr>
<tr>
<td>31</td>
<td>selected-questions</td>
<td>23, 24, 26</td>
</tr>
<tr>
<td>32</td>
<td>standard-form-quest-numbers</td>
<td>21, 23</td>
</tr>
<tr>
<td>33</td>
<td>students</td>
<td>26</td>
</tr>
<tr>
<td>34</td>
<td>valid-idp-updates</td>
<td>21</td>
</tr>
</tbody>
</table>
Name Selection

Parameters:  DB=LOGIC.DBF  PRINT PUNCH=PSANAM.TMP EMPTY
SELECTION="TKEY="level-1.1""  ORDER=BYTYPE

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>course-number</td>
<td>ELEMENT</td>
</tr>
<tr>
<td>2</td>
<td>department-abbreviation</td>
<td>ELEMENT</td>
</tr>
<tr>
<td>3</td>
<td>department-number</td>
<td>ELEMENT</td>
</tr>
<tr>
<td>4</td>
<td>form-number</td>
<td>ELEMENT</td>
</tr>
<tr>
<td>5</td>
<td>instructor-picked-quest-num</td>
<td>ELEMENT</td>
</tr>
<tr>
<td>6</td>
<td>instructors-first-init</td>
<td>ELEMENT</td>
</tr>
<tr>
<td>7</td>
<td>instructors-last-name</td>
<td>ELEMENT</td>
</tr>
<tr>
<td>8</td>
<td>num-quest-copies-request</td>
<td>ELEMENT</td>
</tr>
<tr>
<td>9</td>
<td>question-numbers</td>
<td>ELEMENT</td>
</tr>
<tr>
<td>10</td>
<td>section-number</td>
<td>ELEMENT</td>
</tr>
<tr>
<td>11</td>
<td>inst-general-info</td>
<td>ENTITY</td>
</tr>
<tr>
<td>12</td>
<td>instructor-information</td>
<td>ENTITY</td>
</tr>
<tr>
<td>13</td>
<td>standard-form-quest-numbers</td>
<td>ENTITY</td>
</tr>
<tr>
<td>14</td>
<td>questionnaire-request-form</td>
<td>INPUT</td>
</tr>
<tr>
<td>15</td>
<td>invalid-quest-request</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>16</td>
<td>get-inst-general-info</td>
<td>PROCESS</td>
</tr>
<tr>
<td>17</td>
<td>get-inst-picked-quests</td>
<td>PROCESS</td>
</tr>
<tr>
<td>18</td>
<td>get-standard-form</td>
<td>PROCESS</td>
</tr>
</tbody>
</table>
LOGICAL-SYSTEM

Formatted Problem Statement

Parameters: DB=LOGIC.DBF FILE=PSANAM.TMP INDEX
NOPUNCED-NAMES PRINT NOPUNCH SMARG=5 NMARG=20 AMARG=10
BMARG=25 RNMRG=60 CMARG=1 HMARG=23 ONE-PER-LINE
COMMENT NONEW-PAGE NONEW-LINE NOALL-STATEMENTS
COMPLEMENTARY-STATEMENTS LINE-NUMBERS PRINTEOF
DLC-COMMENT NOSORT-NAME-LIST

1 DEFINE ELEMENT course-number;
2 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
3 DESCRIPTION;
4 A University course number.;
5 TRACE-KEY IS: 'level-0',
6 'level-1',
7 'level-1.1',
8 'level-2';
9 CONTAINED IN: instructor-information,
10 inst-general-info,
11 questionnaire-request-form,
12 global-questionnaire-report,
13 personal-questionnaire-report,
14 questionnaire-forms,
15 questionnaire-answers;
16 VALUES ARE:
17 1 THRU 999;
18
19 DEFINE ELEMENT department-abbreviation;
20 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
21 DESCRIPTION;
22 The department abbreviation is a character string
23 of letters. The length of the string is between 0 and
24 5 characters.;
25 TRACE-KEY IS: 'level-0',
26 'level-1',
27 'level-1.1',
28 'level-2';
29 CONTAINED IN: instructor-information,
30 inst-general-info,
31 questionnaire-request-form,
32 global-questionnaire-report,
33 personal-questionnaire-report,
34 questionnaire-forms;
35
36 DEFINE ELEMENT department-number;
37 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
38 DESCRIPTION;
39 The department number is a number used to identify
40 a particular university department.;
LOGICAL-SYSTEM

Formatted Problem Statement

43 TRACE-KEY IS: 'level-0',
44 'level-1',
45 'level-1.1',
46 'level-2';
47 CONTAINED IN: instructor-information,
48 inst-general-info,
49 question-inst-info,
50 questionnaire-request-form,
51 global-questionnaire-report,
52 personal-questionnaire-report,
53 questionnaire-forms,
54 questionnaire-answers;
55 VALUES ARE:
56 1 THRU 9999;
57
58 DEFINE ELEMENT form-number;
59 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
60 DESCRIPTION;
61 The form number is used to obtain the question numbers
62 for questions in a given standard form. The standard for...
63 be a list of up to 23 question numbers.;
64 TRACE-KEY IS: 'level-0',
65 'level-1',
66 'level-1.1';
67 CONTAINED IN: inst-general-info,
68 questionnaire-request-form,
69 valid-idp-updates,
70 standard-form-quest-numbers;
71 VALUES ARE:
72 1 THRU 999;
73
74 DEFINE ELEMENT instructor-picked-quest-num;
75 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
76 DESCRIPTION;
77 The instructor picked question numbers are a list of
78 numbers to questions in the question data base. The inst...
79 chooses up to 23 questions from the data base and enters ...
80 numbers on his request form.;
81 TRACE-KEY IS: 'level-0',
82 'level-1',
83 'level-1.1',
84 'level-2';
85 CONTAINED IN: instructor-information,
86 question-inst-info,
87 questionnaire-request-form;
88
89 DEFINE ELEMENT instructors-first-init;
90 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
91 TRACE-KEY IS: 'level-0',
LOGICAL-SYSTEM

Formatted Problem Statement

DEFINE ELEMENT instructors-last-name;
/* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
DESCRIPTION;
The instructor's name is the first 14 characters in his last name.;
TRACE-KEY IS: 'level-0', 'level-1', 'level-1.1', 'level-2';
CONTAINED IN: instructor-information, inst-general-info, question-inst-info, questionnaire-request-form, global-questionnaire-report, personal-questionnaire-report, questionnaire-forms;

DEFINE ELEMENT num-quest-copies-request;
/* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
DESCRIPTION;
The number of questionnaire copies are the number of forms that are requested to be printed.;
TRACE-KEY IS: 'level-0', 'level-1', 'level-1.1';
CONTAINED IN: instructor-information, inst-general-info, question-inst-info, questionnaire-request-form, global-questionnaire-report, personal-questionnaire-report, questionnaire-forms;

DEFINE ELEMENT question-numbers;
/* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
DESCRIPTION;
The question numbers are the numbers for the questions in the data base.;
TRACE-KEY IS: 'level-0', 'level-1.1', 'level-3';
LOGICAL-SYSTEM

Formatted Problem Statement

141 CONTAINED IN: dept-quest-list,
142 valid-idp-updates,
143 standard-form-quest-numbers;
144 VALUES ARE:
145 1 THRU 999;
146
147 DEFINE ELEMENT section-number;
149 DESCRIPTION;
150 The section number is the section number for a given
151 university course;
152 TRACE-KEY IS: `level-0`,
153 `level-1`,
154 `level-1.1`,
155 `level-2`;
156 CONTAINED IN: instructor-information,
157 inst-general-info,
158 question-inst-info,
159 questionnaire-request-form,
160 global-questionnaire-report,
161 personal-questionnaire-report,
162 questionnaire-forms,
163 questionnaire-answers;
164 VALUES ARE:
165 1 THRU 999;
166
167 DEFINE ENTITY inst-general-info;
168 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
169 TRACE-KEY IS: `level-1.1`;
170 CONSISTS OF:
171 course-number,
172 department-abbreviation,
173 department-number,
174 form-number,
175 instructors-first-init,
176 instructors-last-name,
177 num-quest-copies-request,
178 section-number;
179 DERIVED BY: get-inst-general-info
180 USING: questionnaire-request-form;
181 USED BY:
182 get-inst-picked-quests
183 TO DERIVE instructor-information;
184 USED BY:
185 get-standard-form
186 TO DERIVE instructor-information;
187
188 DEFINE ENTITY instructor-information;
189 /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
LOGICAL-SYSTEM

Formatted Problem Statement

TRACE-KEY IS: 'level-0',
TRACE-KEY IS: 'level-1',
TRACE-KEY IS: 'level-1.1';
CONSISTS OF:
course-number,
department-abbreviation,
department-number,
instructor-picked-quest-num,
instructors-first-init,
instructors-last-name,
um-quest-copies-request,
section-number;
DERIVED BY: generate-questionnaire-form
USING: question-inst-info,
DERIVED BY: get-standard-form
USING: standard-form-quest-numbers,
instr-general-info;
DERIVED BY: create-instructor-database
USING: questionnaire-request-form;
DERIVED BY: get-inst-picked-quests
USING: instr-general-info,
questionnaire-request-form;
USED BY:
generate-questionnaire-form
TO UPDATE ices-data-pool;

DEFINE ENTITY standard-form-quest-numbers;
/* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
DESCRIPTION;
The standard form question numbers consist of a
table of all possible forms. Each form has a form number
and a list of up to 23 numbers of questions from the data
base of questions;
TRACE-KEY IS: 'level-1.1';
COLLECTED IN: ices-data-pool;
CONSISTS OF:
form-number,
question-numbers;
USED BY:
get-standard-form
TO DERIVE instructor-information;

DEFINE INPUT questionnaire-request-form;
/* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
TRACE-KEY IS: 'level-0',
TRACE-KEY IS: 'level-1',
TRACE-KEY IS: 'level-1.1';
GENERATED: BY instructor;
Formatted Problem Statement

239 CONSISTS OF:
240 course-number,
241 department-abbreviation,
242 department-number,
243 form-number,
244 instructor-picked-quest-num,
245 instructors-first-init,
246 instructors-last-name,
247 num-quest-copies-request,
248 section-number;
249 USED BY:
250 generate-questionnaire-form
251 TO DERIVE questionnaire-forms;
252 USED BY:
253 create-instructor-database
254 TO DERIVE invalid-quest-request;
255 USED BY:
256 create-instructor-database
257 TO DERIVE instructor-information;
258 USED BY:
259 generate-questionnaire-form
260 TO DERIVE instructor-information;
261 USED BY:
262 generate-questionnaire-form
263 TO DERIVE invalid-quest-request;
264 USED BY:
265 get-inst-general-info
266 TO DERIVE inst-general-info;
267 USED BY:
268 get-inst-general-info
269 TO DERIVE invalid-quest-request;
270 USED BY:
271 get-inst-picked-quests
272 TO DERIVE instructor-information;
273
274 DEFINE OUTPUT invalid-quest-request;
275 /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
276 TRACE-KEY IS: 'level-0',
277 'level-1',
278 'level-1.1';
279 RECEIVED: BY instructor;
280 DERIVED BY: generate-questionnaire-form
281 USING: question-inst-info,
282 questionnaire-request-form;
283 DERIVED BY: create-instructor-database
284 USING: questionnaire-request-form;
285 DERIVED BY: get-inst-general-info
286 USING: questionnaire-request-form;
LOGICAL-SYSTEM

Formatted Problem Statement

288 DEFINE PROCESS get-inst-general-info;
289 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
290 SYNONYMS ARE: dfdl.1.1;
291 DESCRIPTION;
292 This process will accept the information about the
293 instructor. This is all of the information except the
294 information about the questions.;
295 TRACE-KEY IS: 'level-1.1';
296 PART OF: create-instructor-database;
297 DERIVES: inst-general-info
298 USING: questionnaire-request-form;
299 DERIVES: invalid-quest-request
300 USING: questionnaire-request-form;
301
302 DEFINE PROCESS get-inst-picked-quests;
303 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
304 SYNONYMS ARE: dfdl.1.3;
305 DESCRIPTION;
306 This routine will read in the questions that the
307 instructor put on the questionnaire request form.;
308 TRACE-KEY IS: 'level-1.1';
309 PART OF: create-instructor-database;
310 DERIVES: instructor-information
311 USING: inst-general-info,
312 questionnaire-request-form;
313
314 DEFINE PROCESS get-standard-form;
315 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
316 SYNONYMS ARE: dfdl.1.2;
317 DESCRIPTION;
318 This process will convert the given standard form
319 number into a list of question numbers to be put into the
320 instructor's information.;
321 TRACE-KEY IS: 'level-1.1';
322 PART OF: create-instructor-database;
323 DERIVES: instructor-information
324 USING: standard-form-quest-numbers,
325 inst-general-info;
326
327 EOF EOF EOF EOF EOF
## LOGICAL-SYSTEM

### Index

1. **course-number** 31, 34, 35, 36
2. **create-instructor-database** 35, 36(3), 37(3)
3. **department-abbreviation** 31, 34, 35, 36
4. **department-number** 31, 34, 35, 36
5. **dept-quest-list** 34
6. **dfd1.1.1** 37
7. **dfd1.1.2** 37
8. **dfd1.1.3** 37
9. **form-number** 32, 34, 35, 36
10. **generate-questionnaire-form** 35(2), 36(4)
11. **get-inst-general-info** 34, 36(3), 37
12. **get-inst-picked-quests** 34, 35, 36, 37
13. **get-standard-form** 34, 35(2), 37
14. **global-questionnaire-report** 31(2), 32, 33(2), 34
15. **ices-data-pool** 35(2)
16. **inst-general-info** 31(2), 32(2), 33(3), 34(2), 35(2), 36, 37(3)
17. **instructor** 35, 36
18. **instructor-information** 31(2), 32(2), 33(3), 34(4), 35, 36(3), 37(2)
19. **instructor-picked-quest-num** 32, 35, 36
20. **instructors-first-init** 32, 34, 35, 36
21. **instructors-last-name** 33, 34, 35, 36
22. **invalid-quest-request** 36(4), 37
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>num-quest-copies-request</td>
<td>33, 34, 35, 36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>personal-questionnaire-report</td>
<td>31(2), 32, 33(2), 34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>question-inst-info</td>
<td>31(2), 32(2), 33(3), 34, 35, 36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>question-numbers</td>
<td>33, 35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>questionnaire-answers</td>
<td>31, 32, 34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>questionnaire-forms</td>
<td>31(2), 32, 33(2), 34, 36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>questionnaire-request-form</td>
<td>31(2), 32(3), 33(3), 34(2), 35(4), 36(3), 37(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>section-number</td>
<td>34(2), 35, 36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>standard-form-quest-numbers</td>
<td>32, 34, 35(2), 37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>valid-idp-updates</td>
<td>32, 34</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Diagram 2

questionnaire-answers

generate-statistics

questionnaire-answer-stats

Print-reports

2.1

question-inst-info

2.2

personal-questionnaire-report

global-questionnaire-report
LOGICAL-SYSTEM

Name Selection

Parameters:  DB=LOGIC.DBF  PRINT PUNCH=PSANAM.TMP  EMPTY

SELECTION='"TKEY="level-2""  ORDER=BYTYPE

1 course-number ELEMENT
2 department-abbreviation ELEMENT
3 department-number ELEMENT
4 frequency-of-answers ELEMENT
5 instructor-picked-quest-num ELEMENT
6 instructors-first-init ELEMENT
7 instructors-last-name ELEMENT
8 mean-of-answers ELEMENT
9 question-answers ELEMENT
10 section-number ELEMENT
11 selected-questions ELEMENT
12 standard-dev-of-answers ELEMENT
13 question-inst-info ENTITY
14 questionnaire-answer-stats ENTITY
15 questionnaire-answers INPUT
16 global-questionnaire-report OUTPUT
17 personal-questionnaire-report OUTPUT
18 generate-statistics PROCESS
19 print-reports PROCESS
LOGICAL-SYSTEM

Formatted Problem Statement

Parameters:  DB=LOGIC.DBF  FILE=PSANAM.TMP  INDEX
NOUNCH-NAMES=PRINT  NOPUNCH  SMARG=5  NMARG=20  AMARG=10
BMARG=25  RNMARG=60  CMARG=1  HMARG=23  ONE-PER-LINE
COMMENT  NONEW-PAGE  NONEW-LINE  NOALL-STATEMENTS
COMPLEMENTARY-STATEMENTS  LINE-NUMBERS  PRINTEOF
DLC-COMMENT  NOSORT-NAME-LIST

1 DEFINE ELEMENT course-number;
2   /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
3   DESCRIPTION;
4   A University course number.;
5   TRACE-KEY IS: 'level-0',
6       'level-1',
7       'level-1.1',
8       'level-2';
9   CONTAINED IN: instructor-information,
10       inst-general-info,
11       questionnaire-request-form,
12       global-questionnaire-report,
13       personal-questionnaire-report,
14       questionnaire-forms,
15       questionnaire-answers;
16   VALUES ARE:
17       1 THRU 999;
18
19 DEFINE ELEMENT department-abbreviation;
20   /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
21   DESCRIPTION;
22   The department abbreviation is a character string
23       of letters. The length of the string is between 0 and
24       5 characters.;
25   TRACE-KEY IS: 'level-0',
26       'level-1',
27       'level-1.1',
28       'level-2';
29   CONTAINED IN: instructor-information,
30       inst-general-info,
31       question-inst-info,
32       questionnaire-request-form,
33       global-questionnaire-report,
34       personal-questionnaire-report,
35       questionnaire-forms;
36
37 DEFINE ELEMENT department-number;
38   /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
39   DESCRIPTION;
40   The department number is a number used to identify
41       a particular university department.;
LOGICAL-SYSTEM

Formatted Problem Statement

43   TRACE-KEY IS: 'level-0',
44       'level-1',
45       'level-1.1',
46       'level-2';
47   CONTAINED IN: instructor-information,
48       inst-general-info,
49       question-inst-info,
50       questionnaire-request-form,
51       global-questionnaire-report,
52       personal-questionnaire-report,
53       questionnaire-forms,
54       questionnaire-answers;
55   VALUES ARE:
56       1 THRU 9999;
57
58 DEFINE ELEMENT frequency-of-answers;
59   /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
60   DESCRIPTION;
61   The frequency of the answers is a list of the number
62   of occurrences for each answer to a given question.;
63   TRACE-KEY IS: 'level-0',
64       'level-2';
65   CONTAINED IN: questionnaire-answer-stats,
66       global-questionnaire-report,
67       personal-questionnaire-report;
68
69 DEFINE ELEMENT instructor-picked-quest-num;
70   /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
71   DESCRIPTION;
72   The instructor picked question numbers are a list of
73   numbers to questions in the question data base. The inst...
74   chooses up to 23 questions from the data base and enters ...
75   numbers on his request form.;
76   TRACE-KEY IS: 'level-0',
77       'level-1',
78       'level-1.1',
79       'level-2';
80   CONTAINED IN: instructor-information,
81       question-inst-info,
82       questionnaire-request-form;
83
84 DEFINE ELEMENT instructors-first-init;
85   /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
86   TRACE-KEY IS: 'level-0',
87       'level-1',
88       'level-1.1',
89       'level-2';
90   CONTAINED IN: instructor-information,
91       inst-general-info,
LOGICAL-SYSTEM

Formatted Problem Statement

92 question-inst-info,
93 questionnaire-request-form,
94 global-questionnaire-report,
95 personal-questionnaire-report,
96 questionnaire-forms;
97
98 DEFINE ELEMENT instructors-last-name;
100 DESCRIPTION;
101 The instructor's name is the first 14 characters in
102 his last name.;
103 TRACE-KEY IS: 'level-0',
104 'level-1',
105 'level-1.1',
106 'level-2';
107 CONTAINED IN: instructor-information,
108 inst-general-info,
109 question-inst-info,
110 questionnaire-request-form,
111 global-questionnaire-report,
112 personal-questionnaire-report,
113 questionnaire-forms;
114
115 DEFINE ELEMENT mean-of-answers;
117 DESCRIPTION;
118 The mean of answers will consist of two parts.
119 The first part is the mean of the answers to a particular
120 question. The second part is a written description telling
121 weather the mean is high, high average, average, low
122 average, or low. The second part was added to help
123 instructors interprete the results.;
124 TRACE-KEY IS: 'level-0',
125 'level-2';
126 CONTAINED IN: questionnaire-answer-stats,
127 global-questionnaire-report,
128 personal-questionnaire-report;
129
130 DEFINE ELEMENT question-answers;
131 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
132 DESCRIPTION;
133 This is a list of symbols telling which answers
134 were picked for the questions in the questionnaire.;
135 TRACE-KEY IS: 'level-0',
136 'level-2';
137 CONTAINED IN: questionnaire-answers;
138
139 DEFINE ELEMENT section-number;
DESCRIPTION;
The section number is the section number for a given university course.

TRACE-KEY IS: 'level-0', 'level-1', 'level-1.1', 'level-2';

CONTAINED IN: instructor-information, inst-general-info, question-inst-info, questionnaire-request-form, global-questionnaire-report, personal-questionnaire-report, questionnaire-forms, questionnaire-answers;

VALUES ARE:
1 THRU 999;

DEFINE ELEMENT selected-questions;

DESCRIPTION;
The selected questions are a list of up to 23 question numbers. These will be used to obtain the corresponding questions from the data base. The questions are either instructor picked or standard form questions.

TRACE-KEY IS: 'level-0', 'level-1', 'level-2';

CONTAINED IN: question-inst-info, questionnaire-forms, personal-questionnaire-report;

DEFINE ELEMENT standard-dev-of-answers;

DESCRIPTION;
The standard deviation of the answers is in two parts. The first part lists the standard deviation of a particular question. The second part uses the standard deviation to list weather the consensus on the answer was, high, average, or low.

TRACE-KEY IS: 'level-0', 'level-2';

CONTAINED IN: questionnaire-answer-stats, global-questionnaire-report, personal-questionnaire-report;

DEFINE ENTITY question-inst-info;

DESCRIPTION;

TRACE-KEY IS: 'level-0',
LOGICAL-SYSTEM

Formatted Problem Statement

'level-1',
'level-2';
COLLECTED IN: ices-data-pool;
CONSISTS OF:
course-number,
department-abbreviation,
department-number,
instructor-picked-quest-num,
instructors-first-init,
instructors-last-name,
um-quest-copies-request,
section-number,
selected-questions;

USED BY:
generate-questionnaire-form
TO DERIVE instructor-information;

USED BY:
generate-questionnaire-form
TO DERIVE invalid-quest-request;

USED BY:
generate-questionnaire-form
TO DERIVE questionnaire-forms;

USED BY:
report-generation
TO DERIVE personal-questionnaire-report;

USED BY:
report-generation
TO DERIVE global-questionnaire-report;

USED BY:
questionnaire-print-routine
TO DERIVE questionnaire-forms;

USED BY:
print-reports
TO DERIVE personal-questionnaire-report;

USED BY:
print-reports
TO DERIVE global-questionnaire-report;

DEFINE ENTITY questionnaire-answer-stats;

/ * DATE OF LAST CHANGE - Apr 21, 1983, 22:59:46 */
TRACE KEY IS: 'level-2';
CONSISTS OF:
frequency-of-answers,
mean-of-answers,
standard-dev-of-answers;
DERIVED BY: generate-statistics
USING: questionnaire-answers;
USED BY:
print_reports
LOGICAL-SYSTEM

Formatted Problem Statement

239 TO DERIVE personnal-questionnaire-report;
240 USED BY:
241 print_reports
242 TO DERIVE global-questionnaire-report;
243
244 DEFINE INPUT questionnaire-answers;
245 /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
246 TRACE-KEY IS: 'level-0',
247 'level-2';
248 GENERATED: BY students;
249 CONSISTS OF:
250 course-number,
251 department-number,
252 question-answers,
253 section-number;
254 USED BY:
255 generate-statistics
256 TO DERIVE questionnaire-answer-stats;
257 USED BY:
258 report-generation
259 TO DERIVE personal-questionnaire-report;
260 USED BY:
261 report-generation
262 TO DERIVE global-questionnaire-report;
263
264 DEFINE OUTPUT global-questionnaire-report;
265 /* DATE OF LAST CHANGE - Apr 21, 1983, 22:59:46 */
266 DESCRIPTION;
267 The global questionnaire report will contain
268 the general information about the instructor. The questions
269 that will be in the report will be the demographic questions
270 about the students and the general questions about the
271 instructor. The report will also list the frequency of
272 the question answers, and the mean and standard deviation
273 for each question.;
274 TRACE-KEY IS: 'level-0',
275 'level-2';
276 RECEIVED: BY faculty-evaluation-committee;
277 CONSISTS OF:
278 course-number,
279 department-abbreviation,
280 department-number,
281 frequency-of-answers,
282 instructors-first-init,
283 instructors-last-name,
284 mean-of-answers,
285 section-number,
286 standard-dev-of-answers;
287 DERIVED BY: report-generation
LOGICAL-SYSTEM

Formatted Problem Statement

288 USING: question-inst-info, 
289 questionnaire-answers; 
290 DERIVED BY: print-reports 
291 USING: question-inst-info; 
292 DERIVED BY: print_reports 
293 USING: questionnaire-answer-stats; 
294
295 DEFINE OUTPUT personal-questionnaire-report; 
296 /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */ 
297 DESCRIPTION; 
298 The personal questionnaire report will contain the 
299 instructor's general information. The questions that will 
300 be listed in the report are the demographic questions about 
301 the students, the general questions about the instructor, 
302 and the instructor's picked questions. For each question 
303 a frequency of the occurrence of the answers will be given, 
304 along with the mean and standard deviation of the question. 
305 A written description of the mean and standard deviation ... 
306 be given to help the instructor interpret the results.; 
307 TRACE-KEY IS: 'level-0', 
308 'level-2'; 
309 RECEIVED: BY instructor; 
310 CONSISTS OF: 
311 course-number, 
312 department-abbreviation, 
313 department-number, 
314 frequency-of-answers, 
315 instructors-first-init, 
316 instructors-last-name, 
317 mean-of-answers, 
318 section-number, 
319 selected-questions, 
320 standard-dev-of-answers; 
321 DERIVED BY: report-generation 
322 USING: question-inst-info, 
323 questionnaire-answers; 
324 DERIVED BY: print-reports 
325 USING: question-inst-info; 
326
327 DEFINE PROCESS generate-statistics; 
328 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */ 
329 SYNONYMS ARE: dfd2.1; 
330 DESCRIPTION; 
331 This routine will generate the frequency, mean, and 
332 standard deviation of the answers to the questions.; 
333 TRACE-KEY IS: 'level-2'; 
334 PART OF: report-generation; 
335 DERIVES: questionnaire-answer-stats 
336 USING: questionnaire-answers;
LOGICAL-SYSTEM

Formatted Problem Statement

337
338 DEFINE PROCESS print-reports;
340 SYNONYMS ARE: dfd2.2;
341 DESCRIPTION;
342 This routine will generate the reports for the
343 instructor and the faculty evaluation committee.;
344 TRACE-KEY IS: 'level-2';
345 PART OF: report-generation;
346 DERIVES: personal-questionnaire-report
347 USING: question-inst-info;
348 DERIVES: global-questionnaire-report
349 USING: question-inst-info;
350
351 EOF EOF EOF EOF EOF
Index

1 course-number 41, 45, 46( 2), 47
2 department-abbreviation 41, 45, 46, 47
3 department-number 41, 45, 46( 2), 47
4 dfd2.1 47
5 dfd2.2 48
6 faculty-evaluation-committee 46
7 frequency-of-answers 42, 45, 46, 47
8 generate-questionnaire-form 45( 3)
9 generate-statistics 45, 46, 47
10 global-questionnaire-report 41( 2), 42( 2), 43( 3), 44( 2), 45( 2), 46( 3), 48
11 ices-data-pool 45
12 inst-general-info 41( 2), 42( 2), 43, 44
13 instructor 47
14 instructor-information 41( 2), 42( 3), 43, 44, 45
15 instructor-picked-quest-num 42, 45
16 instructors-first-init 42, 45, 46, 47
17 instructors-last-name 43, 45, 46, 47
18 invalid-quest-request 45
19 mean-of-answers 43, 45, 46, 47
20 num-quest-copies-request 45
21 personal-questionnaire-report 41( 2), 42( 2), 43( 3), 44( 3),
Index

22 personnal-questionnaire-report 46
23 print-reports 45, 46, 47
24 print_reports 45, 46, 47
25 question-answers 43, 46
26 question-inst-info 41, 43, 44, 45, 46, 47(2)
27 questionnaire-answer-stats 42, 43, 44, 45, 46, 47(2)
28 questionnaire-answers 41, 42, 43, 44, 45, 46, 47(3)
29 questionnaire-forms 41(2), 42, 43(2), 44(2), 45(2)
30 questionnaire-print-routine 45
31 questionnaire-request-form 41, 42(2), 43(2), 44
32 report-generation 45(2), 46(3), 47(2), 48
33 section-number 43, 45, 46(2), 47
34 selected-questions 44, 45, 47
35 standard-dev-of-answers 44, 45, 46, 47
36 students 46
Diagram 3

Print-questions

3.1

question-list-request

question-list

questions

update-questions

dept-question-list

valid-idp-updates
Name Selection

Parameters:  DB=LOGIC.DBF  PRINT  PUNCH=PSANAM.TMP  EMPTY  
SELECTION="TKEY="level-3""  ORDER=BYTYPE

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>list-of-all-questions</td>
<td>ELEMENT</td>
</tr>
<tr>
<td>2</td>
<td>question-numbers</td>
<td>ELEMENT</td>
</tr>
<tr>
<td>3</td>
<td>standard-forms-list</td>
<td>ELEMENT</td>
</tr>
<tr>
<td>4</td>
<td>questions</td>
<td>ENTITY</td>
</tr>
<tr>
<td>5</td>
<td>valid-idp-updates</td>
<td>ENTITY</td>
</tr>
<tr>
<td>6</td>
<td>dept-quest-list</td>
<td>INPUT</td>
</tr>
<tr>
<td>7</td>
<td>question-list-request</td>
<td>INPUT</td>
</tr>
<tr>
<td>8</td>
<td>question-list</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>9</td>
<td>print-questions</td>
<td>PROCESS</td>
</tr>
<tr>
<td>10</td>
<td>update-questions</td>
<td>PROCESS</td>
</tr>
</tbody>
</table>
Formated Problem Statement

Parameters: DB=LOGIC.DBF FILE=PSANAM.TMP INDEX NOUNPUNCHED=NAMES PRINT NOPUNCH SMARG=5 NMARG=20 AMARG=10 BMARG=25 RNMARG=60 CMARG=1 HMARG=23 ONE-PER-LINE COMMENT NONEW-PAGE NONEW-LINE NOALL-STATEMENTS COMPLEMENTARY-STATEMENTS LINE-NUMBERS PRINTEOF DLC-COMMENT NOSORT-NAME-LIST

1 DEFINE ELEMENT list-of-all-questions;
2 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
3 DESCRIPTION;
4 This is a list of all of the questions that are
5 kept in the data base. The questions are in three parts,
6 a question number, and two string parts each containing
7 half of a question.;
8 TRACE-KEY IS: ’level-0’,
9 ’level-3’;
10 CONTAINED IN: questions,
11 question-list;
12
13 DEFINE ELEMENT question-numbers;
14 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
15 DESCRIPTION;
16 The question numbers are the numbers for the questions
17 in the data base.;
18 TRACE-KEY IS: ’level-0’,
19 ’level-1.1’,
20 ’level-3’;
21 CONTAINED IN: dept-quest-list,
22 valid-idp-updates,
23 standard-form-quest-numbers;
24 VALUES ARE:
25 1 THRU 9999;
26
27 DEFINE ELEMENT standard-forms-list;
29 DESCRIPTION;
30 This form will be a list of all standard forms
31 currently in the data base. It will list out the questions
32 in each of the standard forms.;
33 TRACE-KEY IS: ’level-0’,
34 ’level-3’;
35 CONTAINED IN: questions,
36 question-list;
37
38 DEFINE ENTITY questions;
40 TRACE-KEY IS: ’level-0’,
41 ’level-3’;
42 COLLECTED IN: ices-data-pool;
Formatted Problem Statement

43 CONSISTS OF:
44    list-of-all-questions,
45    standard-forms-list;
46 USED BY:
47    database-maintainance
48       TO DERIVE question-list;
49 USED BY:
50    print-questions
51       TO DERIVE question-list;
52
53 DEFINE ENTITY  valid-idp-updates;
54  /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
55  TRACE-KEY IS: 'level-0',
56    'level-3';
57 CONSISTS OF:
58    form-number,
59    question-numbers;
60 DERIVED BY:  database-maintainance
61      USING: dept-quest-list;
62 DERIVED BY:  update-questions
63      USING: dept-quest-list;
64 USED BY:
65    database-maintainance
66      TO UPDATE ices-data-pool;
67
68 DEFINE INPUT  dept-quest-list;
69  /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
70  TRACE-KEY IS: 'level-0',
71    'level-3';
72 GENERATED:  BY faculty-evaluation-committee;
73 CONSISTS OF:
74    question-numbers;
75 USED BY:
76    database-maintainance
77       TO DERIVE valid-idp-updates;
78 USED BY:
79    update-questions
80       TO DERIVE valid-idp-updates;
81
82 DEFINE INPUT  question-list-request;
83  /* DATE OF LAST CHANGE - Apr 21, 1983, 20:26:30 */
84  TRACE-KEY IS: 'level-0',
85    'level-3';
86 GENERATED:  BY instructor;
87 USED BY:
88    database-maintainance
89       TO DERIVE question-list;
90 USED BY:
91    print-questions
FORMATTED PROBLEM STATEMENT

92 TO DERIVE question-list;
93
94 DEFINE OUTPUT question-list;
95 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
96 TRACE-KEY IS: 'level-0',
97 'level-3';
98 RECEIVED: BY instructor;
99 CONSISTS OF:
100 list-of-all-questions,
101 standard-forms-list;
102 DERIVED BY: database-maintainance
103 USING: questions,
104 question-list-request;
105 DERIVED BY: print-questions
106 USING: questions,
107 question-list-request;
108
109 DEFINE PROCESS print-questions;
110 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
111 SYNONYMS ARE: dfd3.1;
112 DESCRIPTION;
113 This routine will print out a list of all the questions
114 in the question database;
115 TRACE-KEY IS: 'level-3';
116 PART OF: database-maintainance;
117 DERIVES: question-list
118 USING: questions,
119 question-list-request;
120
121 DEFINE PROCESS update-questions;
122 /* DATE OF LAST CHANGE - Apr 21, 1983, 19:37:15 */
123 SYNONYMS ARE: dfd3.2;
124 DESCRIPTION;
125 This routine will create new standard forms for the
126 departments requesting them;
127 TRACE-KEY IS: 'level-3';
128 PART OF: database-maintainance;
129 DERIVES: valid-idp-updates
130 USING: dept-quest-list;
131
132 EOF EOF EOF EOF EOF
LOGICAL-SYSTEM

Index

1 database-maintainance 53( 5), 54( 3)
2 dept-quest-list 52, 53( 3), 54
3 dfd3.1 54
4 dfd3.2 54
5 faculty-evaluation-committee 53
6 form-number 53
7 ices-data-pool 52, 53
8 instructor 53, 54
9 list-of-all-questions 52, 53, 54
10 print-questions 53( 2), 54( 2)
11 question-list 52( 2), 53( 3), 54( 3)
12 question-list-request 53, 54( 3)
13 question-numbers 52, 53( 2)
14 questions 52( 3), 54( 3)
15 standard-form-quest-numbers 52
16 standard-forms-list 52, 53, 54
17 update-questions 53( 2), 54
18 valid-idp-updates 52, 53( 3), 54
Design document for the first part of the Faculty Evaluation System.

1) Generate questionnaire forms

Description:
The first part of the faculty evaluation system generates the questionnaire forms for the instructor. The instructor picks the questions from a given data base of questions.

Input:
A list of the questions in the data base along with the standard forms are distributed to each department. The instructor returns a form with his general information, and a list of question numbers which reference the question data base, to data entry personnel who enter the information into the data base containing instructor's information.

Output:
The number of copies of the questions that were requested. Global questions are preprinted on the forms. The output format is set to print on the scantron faculty evaluation forms. A command file is created to print out the questionnaires for each instructor.

Algorithm:
1) build instructor data base
2) print evaluation questionnaires

1.1) Create Instructor Data Base

Description:
This process will display a menu of options for modifications to the instructor data base. The process will then call the appropriate procedure to perform the task. This routine allows the user to add, delete, and print records from the instructor data base. The routine will also automatically back up the instructor data base after every ten changes made to it.

Input:

Instructor request form:
   Instructor's name : a string of up to 14 characters
                       first initial : character
   Department abbreviation : string of up to 5 characters
   Department number : integer number (0<=n<=9999)
   Course number : integer number (0<=n<=999)
   Course section number : integer number (0<=n<=999)
   Print flag : char
   Number of questionnaires requested : integer number (0<=n<=999)
   Form number : integer number (0<=n<=9999)
   instructor picked questions : list of question numbers (0<=n<=23)
   question numbers : integer number (0<=n<=9999)

Output:

   Instructor's name : array [1..14] of char;
   first initial : char;
   Department abbreviation : array [1..5] of char;
   Department number : array [1..4] of char;
   Course number : array [1..3] of char;
   Course section number : array [1..3] of char;
   Print flag : char
   Number of questionnaires requested : array [1..3] of char;
   Form number : array [1..4] of char;
   instructor picked questions : array [1..23] of question numbers;
   question numbers : array [1..4] of char;

The following subroutines are called by this procedure:
   get_forms - gets standard forms
   get_instruct_info - reads in the instructor data base
   heading - print program heading
   reset_files - updates the instructor file
   add_instruct - adds and instructor record
   delete_instruct - deletes an instructor record
   print_instruct - prints a list of all instructors in the data base.
   dump_instruct - writes the instructor information to a file

1.1.1) get_forms
Description:
This routine reads in the standard questionnaire records into a linked list. Each standard questionnaire record consists of a form number and 23 question numbers.

input:
a file of standard forms

output:
a linked list of standard forms.
each record consists of the following:
  form_num : packed array [1..4] of char;
  quest_nums : array [1..23] of key_type
where key_type is a packed array [1..4] of char.

This routine is called by:
  the main routine

1.1.2) get_instruct_info

Description:
This routine reads in the instructor database into a linked list from a file.

Input:
  head_inst_list : a pointer to the beginning of the list.

Output:
  Head_inst_list : a pointer to the head of the list

This routine is called by:
  the main routine
This routine call:
  get_rec

1.1.2.1) get_rec

Description:
This routine reads in an instructor record for the instructor database file.

Input:
  Instructor's name : a string of up to 14 characters
  first initial : character
  Department abbreviation : string of up to 5 characters
  Department number : integer number (0<=n<=9999)
  Course number : integer number (0<=n<=999)
  Course section number : integer number (0<=n<=99)
  Print flag : char
  Number of questionnaires requested : integer number
Form number : integer number (0<=n<=9999)
instructor picked questions : list of question numbers (0<=n<=23)
question numbers : integer number (0<=n<=9999)

Output:
The output is the same as the input except the information is in a linked list.

This routine is called by:
   get_instruct_info

1.1.3) heading

Description:
This routine prints the program heading and the menu of options onto the screen.

1.1.4) reset_files

Description:
This routine is called after 10 changes have been made to the instructor data base. The routine calls dump_inst to write the instructor records to a file.

Input:
head_inst : a pointer to the head of the instructor list

Output:
none

This routine is called by:
   the main routine
This routine calls:
   dump_inst

1.1.4.1) dump_inst

Description:
This routine writes the updated instructor records to a file.

Input:
head_inst : a pointer to the beginning of the instructor list

Output:
A file containing the instructor records

The routine is called by:
   reset_files
   the main routine
1.1.5) add_inst

Description:
This routine will read in the information about the instructor and read in either the form or question numbers that he selected. It will also allow the information to be reviewed before accepting it. This routine calls the appropriate subroutine to read in each of the fields in the instructor record so that all input is parsed.

Input:
Instructor request form:
- Instructor's name: a string of up to 14 characters
- First initial: character
- Department abbreviation: string of up to 5 characters
- Department number: integer number (0<=n<=9999)
- Course number: integer number (0<=n<=999)
- Course section number: integer number (0<=n<=999)
- Number of questionnaires requested: integer number (0<=n<=999)
- Form number: integer number (0<=n<=9999)
- Instructor picked questions: list of question numbers (0<=n<=23)
- Question numbers: integer number (0<=n<=9999)

Output:
- Instructor's name: array [1..14] of char
- First initial: char
- Department abbreviation: array [1..5] of char
- Department number: array [1..4] of char
- Course number: array [1..3] of char
- Course section number: array [1..3] of char
- Number of questionnaires requested: array [1..3] of char
- Form number: array [1..4] of char
- Instructor picked questions: array [1..23] of question numbers

This routine is called by:
- The main routine

The routine calls:
- get_name - read an instructor's name
- get_dept_abr - read in the department abbreviation
- get_num - read in an integer number
- convert4 - convert a number to a [1..4] array
- convert3 - convert a number to a [1..3] array
- get_picked - read instructor picked questions
- forms_quest - read inn questions for a form number

1.1.5.1) get_name
Description:
This routine reads in an instructor name from the terminal. The routine maps all lower case letters to upper case. The routine will only read in characters up to the maximum length of a name.

Input:
characters from the terminal

Output:
name : packed array [1..14] of char;

This routine is called by:
add_inst - add an instructor record
delete_inst - delete an instructor record

1.1.5.2) get_dept_abr

Description:
This routine reads in the department abbreviation from the terminal. The routine maps all lower case letters to upper case letters. The routine will only read in characters up to the maximum length of the department abbreviation.

Input:
characters from the terminal

Output:
dept_abr : packed array [1..5] of char

This routine is called by:
add_inst - add an instructor record

1.1.5.3) get_num

Description:
This routine reads in an integer number from the terminal. The number is read in a character at a time and the number is computed. The routine will only except valid integer numbers. An error message will be printed for invalid entries and the user will have to enter a new number.

Input:
characters from the terminal

Output:
An integer number

This routine is called by
add_inst - add an instructor record
get_picked - get an instructor picked question number
delete_inst - delete an instructor record

1.1.5.4) convert4

Description:
This routine converts an integer number into an array with a range of four.

Input:
an integer number

Output:
key : packed array [1..4] of char;

This routine is called by:
add_inst - add an instructor record
get_picked - get an instructor picked question number
forms_quest - get standard form question numbers

1.1.5.5) convert3

Description:
This routine converts an integer number into an array with a range of three.

Input:
an integer number

Output:
key : packed array [1..3] of char;

This routine is called by:
add_inst - add an instructor record
delete_inst - delete an instructor record

1.1.5.6) get_picked

Description:
This routine will read in the instructor picked questions. The numbers of the questions are entered. The routine will allow the user to enter a maximum of 23 questions which is the number of questions set for the size of the forms that the questions are printed out on. If the instructor chooses less than 23 questions then the data entry personal can enter a zero as the last question number and the routine will stop.

Input:
question numbers

Output:
A list of question numbers

This routine is called by:
   add_inst - add an instructor record
The routine calls:
   init_quest - initialize the question list
   get_num - get an integer number
   convert4 - convert an integer to an array of characters
   char_to_dig - convert a array to an integer

1.1.5.6.1) init_quest

Description:
This routine initializes the list of instructor picked questions to be all zeros.

Input:
quests : array [1..23] of question numbers

Output:
quests : array [1..23] of question numbers

This routine is called by:
   get_picked_quest - get instructor picked questions

1.1.5.6.2) char_to_dig

Description:
This routine converts an array [1..4] of digits to a integer number.

Input:
key : packed array [1..4] of char

Output:
an integer number

This routine is called by:
   get_picked_quest - get instructor picked questions

1.1.5.7) forms_quest

Description:
This routine receives a form number. It looks through the list of forms until it finds the form number. The routine then writes the question numbers from the standard form into the question list in the instructor record.

Input:
Form number : array [1..4] of char;
Output:

standard form question numbers : array [1..23] of
question numbers

This routine is called by:
add_inst - add an instructor record

The routine calls:
convert4 - convert an integer number to an array of
characters

1.1.6) delete_inst

Description:
This routine allows the user to delete an instructor record.
The routine has the user enter the instructor's name and the
course that he is teaching. The routine will then display all
records that have the information that was given. This
routine is good for deleting duplicate records since the user
can see both records.

Input:
the instructor's name
the instructor's course number

Output:
updated instructor data base

This routine is called by:
the main routine

This routine calls:
get_name - get a instructor's name
get_num - get a integer number
convert3 - covert a integer number to an array of
characters
d_inst - delete an instructor record

1.1.6.1) d_inst

Description:
This routine displays the complete instructor record so that
the user can see if the record is to be deleted. If the user
chooses to delete the record then the record is removed from
the data base.

Input:
inst : the instructor record
character from the terminal

Output:
the updated instructor data base
This routine is called by:
    delete_inst - delete an instructor record

1.1.7) print_inst

Description:
This routine will create a printable version of the instructor records. It will write headings for the fields at the top and write the records into a formatted output.

Input:
The instructor record list

Output:
A file of instructor records

This routine is called by:
    the main routine
This routine calls:
    print_heading - prints the page heading
    write_line - writes an instructor record

1.1.7.1) print_heading

Description:
This routine writes out the heading for the instructor report file.

Input
page number : integer

Output:
Heading to report file

This routine is called by:
    print_inst - print the instructor records

1.1.7.2) write_line

Description:
This routine writes out the complete instructor record to the instructor report file.

Input:
An instructor record

Output:
The instructor record written to the report file

This routine is called by:
    print_inst - print the instructor records
The routine calls:
  line - write a line of characters

1.1.7.2.1) line

Description:
This routine writes out a string of characters for the specified length.

Input:
  ch : char
  n : integer

Output:
A string of length n of char ch

This routine is called by:
  write_line - write an instructor reco
1.2) Questionnaire Print Routine

The second part of the questionnaire generation routine deals with printing out the questionnaire forms. The questionnaire forms are all special forms. The global information questions and the scantron boxes (for answers) are all printed on the form. This routine will print out the instructor's information and the questions that he picked.

Input:
Data base of instructor information

Output:
The questionnaire form for each instructor in the data base for whom one has not yet been printed.
A control file that will submit a print request for each instructor's questionnaires with the number of copies supplied by the data base and with the appropriate control switches.

The main routine calls:

- `init_table` - initialize the questions table
- `build_question_list` - put questions into the table
- `get_instruct_info` - read instructor information in a linked list
- `get_commands` - opens the print command file
- `print_info` - print the questionnaire forms
- `dump_inst` - write the updated instructor information back to a file.

1.2.1) init_table

Description:
This routine initializes the question table file to be an array of pointers all equal to nil.

Input:
none

Output:
q_tab : array [1.9999] of pointers

This routine is called by:
the main routine

1.2.2) build_question_list

Description:
This routine reads in the questions from the question data base. The questions are placed into the question table according to there number. All empty spots in the table will have a nil value.
Input:
A file of questions

Output:
q_tab : array [1..9999] of questions

where questions consist of the following
    question number : packed array [1..4] of char;
    question string 1 : packed array [1..43] of char;
    question string 2 : packed array [1..43] of char;

This routine is called by:
    the main routine
This routine calls:
    get_quest - reads in a question
    get_num - converts an array to an integer

1.2.2.1) get_quest

Description:
This routine reads in a question from the question data base file.

Input:
A question record

Output:
A pointer to the question record

This routine is called by:
    build_question_list - builds the question table

1.2.2.2) get_num

Description:
This routine receives an array of digits and converts them into an integer number:

Input:
key : packed array [1..4] of char;

Output:
num : integer

This routine is called by:
    build_question_list - create the question table
    print_questionnaires - prints the questionnaire forms
    print_cmd - creates the print command file

1.2.3)get_instruct_info
Description:
This routine reads in the instructor data base into a linked list from a file.

Input:
head_inst_list : a pointer to the beginning of the list.

Output:
Head_inst_list : a pointer to the head of the list

This routine is called by:
the main routine
This routine call:
get_rec

1.1.2.1) get_rec

Description:
This routine reads in an instructor record for the instructor data base file.

Input:
Instructor's name : a string of up to 14 characters
first initial : character
Department abbreviation : string of up to 5 characters
Department number : integer number (0<=n<=9999)
Course number : integer number (0<=n<=999)
Course section number : integer number (0<=n<=999)
Print flag : char
Number of questionnaires requested : integer number (0<=n<=999)
Form number : integer number (0<=n<=9999)
instructor picked questions : list of question numbers (0<=n<=23)
question numbers : integer number (0<=n<=9999)

Output:
The output is the same as the input except the information is in a linked list

This routine is called by:
get_instruct_info

1.2.4) get_commands

Description:
This routine opens the print commands file.

Input:
none
1.2.5) Print_info

Description:
The instructor data base is checked to see what needs to be printed. A key will be unset if the file has not been printed. General information about the instructor is placed into a file. The appropriate number of blank lines are then printed to skip over the global questions that are preprinted at the top of the form. The questions that the instructor picked are then used to into the question data base to obtain the questions to be printed. The questions are printed into the same file as the instructor's information. Once the form is created, a key is set in the instructor's information record signifying that the instructor's form has been printed.

Input:
Data base of instructor information
The table of questions

Output:
Files containing the instructor's questionnaire form

This routine is called by:
the main routine

This routine calls:
get_out_file - Opens the questionnaire output file
print_heading - Prints the questionnaire heading
print_quest - Prints the questions for the form
print_cmd - Prints the print command file

1.2.5.1) get_out_file

Description:
This routine opens the output files for the questionnaire forms. The routine keeps track of the questionnaire file number by using a batch file which contains the number of the file. The routine updates the batch file number each time it is called.

Input:
none

Output:
opens a text file
This routine is called by:
    print_info - prints the questionnaire forms
The routine calls:
    init_out_string - initialize file name

1.2.5.1.1) init_out_string

Description:
This routine initializes the string that will be the file name.

Input:
none

Output:
out_string : packed array [1..14] of char;

This routine is called by:
    get_out_string - opens the output file

1.2.5.2) print_heading

Description:
This routine writes the instructor's name, department abbreviation, department number, course number, and section number onto the top of the questionnaire form. The routine then writes the blank lines needed to start the questions in the right spot.

Input:
an instructor record

Output:
Heading for the file

This routine is called by:
    print_info - print questionnaire form

1.2.5.3) print_quest

Description:
This routine uses the question numbers in the instructor record to obtain the questions from the question table. The routine uses the question numbers as an index to read the questions out of the table. If the question is not found in the table then a blank line is printed. The routine will stop when it reaches 23 questions or until a question number zero is found in the instructor record.

Input:
A pointer to the instructor record
The question table

Output:
The questions for the questionnaire

This routine is called by:
  print_info - prints a questionnaire form
This routine calls:
  get_num - Convert an array of digits to an integer number

1.2.5.5) print_cmd

Description:
This routine will create a command file of print requests. For each instructor's questionnaire form a print command will be given. The print command will give the name of the instructor's file to be printed, the number of copies to print, the type of forms to use and any other appropriate specifications. The number of forms to be printed will be obtained from the instructor's information. If more than 300 forms are to be printed the routine will write multiple print requests.

Input:
A questionnaire form number from a file
A list of instructor requests that need to be printed

Output:
A command file of print requests

This routine is called by:
  print_info - prints the questionnaire forms
This routine calls:
  get_num - convert an array of digits to a integer number
  print_line - prints a command line

1.2.5.5.1) print_line

Description:
This routine writes out the command line to print the questionnaire form with the appropriate switches.

Input:
output file name
number of copies to print

Output
A command string to print the form
This routine is called by:
    print_cmd - prints the command lines

1.1.6) dump_inst

Description:
This routine writes the updated instructor records to a file.

Input:
head_inst: a pointer to the beginning of the instructor list.

Output:
A file containing the instructor records

The routine is called by:
    the main routi
Design document for the report generation routines for the Faculty Evaluation System

2) Generating evaluation report

Description:
The report generation routines are composed of two parts. The first part computes the statistics files from the questionnaire answers. The second part of the routine will generate the reports for the instructor and for the faculty evaluation committee.

Input:
questionnaire answers from a scantron card reader

Output:
report for the instructor
report for the faculty evaluation committee

The following routines are used by the system:
  stats - Computes the statistics from the questionnaire answers
  report - Produces the reports for the instructor and the faculty evaluation committee

2.1) stats

Description:
The statistics routine reads in files that were generated by the scantron card reader. The routine converts the scantron codes into numbers. The routine then computes the following statistics from the questionnaire answers: the frequency, the mean, and the standard deviation. The first card that the routine reads contains the department number, course number, and section number for the questionnaire answers. The header record will be used in the report program to find the correct instructor record. Each set of questionnaire statistics is placed into a separate file that will be accessed to create the report. If the scantron file is empty an error message is printed and no file is generated.

Input:
header record
raw scantron data

Output:
Statistics files

The following subroutines are called by this routine:
  get_files - get a scantron produced file
get_info - read in scantron information  
compute_stats - computes the statistics on the information  
print_results - prints the statistics information into a file

2.1.1) get_files

Description:
This routine will open a scantron generated file if it exists. If the file is not there then an error messages is returned. The routine will check for scantron file 0-9. If the file exists then the result file is opened.

Input:
scantron file number

Output:
error flag
the opened scantron file
the opened statistics file

This routine is called by:
the main routine
The routine calls:
    get_out_file - open the output file

2.1.1.1) get_out_file

Description:
This routine will create the output files for the statistics. It will also increment the batch number in the batch file so that each call to this routine will increment the number so the output files will all be named differently.

Input:
none

Output:
the opened output file

This routine is called by:
get_files
The routine calls:
    init_out_string

2.1.1.1.1) init_out_string

Description:
This routine initializes the string that will contain the
output file name.

Input:
none

Output:
out_string : packed array [1..14] of char

This routine is called by;
    get_out_file

2.1.2) get_info

Description:
This routine will read in all of the information that is in
the scantron questionnaire answer file. Each answer record is
the answers for one person's evaluation of the instructor.
The records are put into a linked list so the statistics can
be computed. The routine will write an error message if there
is no data in the file.

Input:
Raw scantron data for an instructor

Output:
A linked list of answers and a header record with the
department number, course number, and section number.

This routine is called by:
    the main routine
The routine calls:
    init_head_conv_table - initialize the header conversion
table
    get_header - read in and convert the header record
    init_array - initialize the array to hold the answers
    get_card - read in a answer record

2.1.2.1) init_head_conv_table

Description:
This routine sets up a table that contains all of the possible
values for a header card record.

Input:
none

Output:
head_conv_table

This routine is called by:
    get_info - reads in a scantron file
2.1.2.2) get_header

Description:
This routine uses the header conversion table to convert the scantron code to the numbers representing the department, course, and section numbers.

Input:
scantron header record

Output:
header : packed array [1..11] of char;

This routine is called by:
get_info - reads in a scantron file

2.1.2.3) init_array

Description:
This routine initializes the array that will contain a questionnaire answer record. That is the answers to one questionnaire form.

Input:
none

Output:
l_array : array [1..33] of integer;

This routine is called by:
get_info - reads in a scantron file

2.1.2.4) get_card

Description:
This routine converts the scantron code for a questionnaire answer into integer numbers.

Input:
A scantron answer record

Output:
The integer numbers for the answers

This routine is called by:
get_info - reads in a scantron file

2.1.3) compute_stats

Description:
This routine computes the frequency, mean, and standard deviation for each of the questions. All zero values in the records are discarded.

Input:
The list of all question answers

Output:
The frequency, mean, and standard deviation of the questions

This routine is called by:
    the main routine
The routine calls:
    init_results - initialize the results to zero

2.1.3.1) init_results

Description:
This routine initializes the statistics variables to start at zero.

Input:
none

Output:
tot_ans_line : array [1..33] of real;
mean : array [1..33] of real;
stand_dev : array [1..33] of real;
frequency : array [1..33,1..5] of real;

This routine is called by:
    compute_stats : computes the questionnaire answers
                    statistics

2.1.4) print_results

Description:
The routine will print the statistics information to the appropriate file. The output file name will be changed by the time this routine is executed again.

Input:
tot_ans_line : array [1..33] of real;
mean : array [1..33] of real;
stand_dev : array [1..33] of real;
frequency : array [1..33,1..5] of real;

Output:
The above input information is written to a file

This routine is called by:
the main routine
2.2) report

Description:
This routine will produce the reports for the answers given by the students to the evaluation questions. The report for the instructor consists of questions that the instructor picked and the results to the questions. The routine also produces a report for the faculty evaluation committee. This report lists the global questions and the results for the questions. The routine allows the user to create multiple reports or to create a single report.

Input:
The instructor data base
The question data base
The statistics files for the instructors

Output:
The instructor's personal report
The faculty evaluation report

This routine calls the following subroutines:
get_instruct_info - read in the instructor data base
build_question_list - create the question table
single_entry - create a single report
mult_entry - create multiple reports

2.2.1) get_instruct_info

Description:
This routine builds a tree containing the instructor information. The tree contains three branches. In the top branch of the tree is a linked list of departments. On each of the nodes in the department linked list is a pointer to a list of all of the courses in the department. On each of the nodes for a course is a pointer to a list of sections that belong to a course. The tree structure allows for rapid lookup of instructor records as well as finding errors if the information is not in the tree.

Input:
head_inst_list : a pointer to the beginning of the list.

Output:
Head_inst_list : a pointer to the head of the list
the tree containing the instructor records

This routine is called by:
the main routine
This routine call:
get_rec - get an instructor record

2.2.1.1) get_rec

Description:
This routine reads in an instructor record for the instructor data base file.

Input:
Instructor's name : a string of up to 14 characters
first initial : character
Department abbreviation : string of up to 5 characters
Department number : integer number (0<=n<=9999)
Course number : integer number (0<=n<=999)
Course section number : integer number (0<=n<=999)
Print flag : char
Number of questionnaires requested : integer number (0<=n<=999)
Form number : integer number (0<=n<=9999)
instructor picked questions : list of question numbers (0<=n<=23)
question numbers : integer number (0<=n<=9999)

Output:
The output is the same as the input except the information is in a linked list

This routine is called by:
    get_instruct_info

2.2.2) build_question_list

Description:
This routine reads in the questions from the question data base. The questions are placed into the question table according to their number. All empty spots in the table will have a nil value.

Input:
A file of questions

Output:
q_tab : array [1..9999] of questions

where questions consist of the following
question number : packed array [1..4] of char;
question string 1 : packed array [1..43] of char;
question string 2 : packed array [1..43] of char;

This routine is called by:
    the main routine
This routine calls:
  init_table - initialize the question table to all nils
  get_quest - reads in a question
  get_num - converts an array to an integer

2.2.2.1) init_table

Description:
This routine initializes the question table file to be an array of pointers all equal to nil.

Input:
none

Output:
q_tab : array [1..9999] of pointers

This routine is called by:
  build_question_list - creates the question table

2.2.2.2) get_quest

Description:
This routine reads in a question from the question data base file.

Input:
A question record

Output:
A pointer to the question record

This routine is called by:
  build_question_list - builds the question table

2.2.2.3) get_num

Description:
This routine receives an array of digits and converts them into an integer number:

Input:
key : packed array [1..4] of char;

Output:
num : integer

This routine is called by:
  build_question_list - create the question table
  print_instructor - print the instructor information
2.2.3) Single_mode

Description:
This routine will produce a report for one instructor. The correct department, course, and section number are first found and then a check to see if the instructor taught the class is made. This routine is used mainly for the case where more than one teacher taught a class and the instructor's name has to be entered to make the record unique. This routine will write all error messages to the terminal.

Input:
The statistics file number
The instructor's name

Output:
A report for the instructor
A report for the faculty evaluation committee

This routine is called by:
the main routine
The routine calls:
get_number - reads in an integer number
get_file - get the appropriate output file
get_header - read in the header record
find_rec - find the instructor record in the database
print_report - print the reports

2.2.3.1) get_numbers

Description:
This routine reads in an integer number from the terminal. The number is read in a character at a time and the number is computed. The routine will only except valid integer numbers. An error message will be printed for invalid entries and the user will have to enter a new number.

Input:
characters from the terminal

Output:
An integer number

This routine is called by
single_entry - create a single report
mult_entry - create multiple reports

2.2.3.2) get_file

Description:
This routine will get an input statistics file. The number
that is passed to it is the number of the file that is to be opened.

Input:
file_number : integer

Output:
An opened statistics file

This routine is called by:
  single_entry - create a single report
  mult_entry - create multiple reports

2.2.3.3) get_header

Description:
This routine will read in the header card from the statistics file. It fills in the extra characters needed to match the fields in the instructor record.

Input:
Header record

Output:
The department, course and section number

This routine is called by:
  single_entry - create a single report
  mult_entry - create multiple reports

2.2.3.4) find_rec

Description:
This routine will find an instructor's record in the list of instructor records. If the record is not found then an error message is printed to the error file and a nil values is returned as a pointer. If the record is found then a pointer to the record is returned.

Input:
The header record (dept, course, sect. numbers)

Output:
A pointer to the instructor record

This routine is called by:
  single_entry - create a single report
  mult_entry - create multiple reports

2.2.3.5) print_report
Description:
This routine calls the appropriate subroutines to print out the reports for the instructor's report and for the faculty evaluation report.

Input:
A pointer to the instructor record
The opened statistics file
The file number (for an output file number)

Output:
The instructor's report
The faculty evaluation committee's report

This routine is called by:
  single_entry - create a single report
  mult_entry - create multiple reports

The routine calls:
  get_out_file - opens the output file
  get_global_stats - gets the global question results
  line - draws a line
  print_global - prints the global questions
  print_inst - prints the instructor's picked questions

2.2.3.5.1) get_out_file

Description:
This routine will open the output file. The file number that is passed to it is the same as the number of the statistics file that the report is for.

Input:
file_number : integer

Output:
the opened output file

This routine is called by:
  print_report - prints the reports

2.2.3.5.2) get_global_stats

Description:
This routine will read in the statistics for the global questions from the statistics file.

Input:
frequency, mean, and standard deviation for the global questions

Output:
globals : array [1..10] of stats type;

This routine is called by:
   print_report - prints the reports

2.2.3.5.3) line

Description:
This routine will write a line with the given character for the
length passed to it.

Input:
ch : char
num : integer

Output:
A line of length num using character ch

This routine is called by:
   print_report - prints the reports
   print_quest - prints the instructor's questions
   print_student - prints the global questions

2.2.3.5.4) print_global

Description:
This routine calls the routines to print out the report for
the global question.

Input:
A pointer to the instructor record
the global statistics record
the opened output file

Output:
The global questions report

This routine is called by:
   print_report - prints the reports
The routine calls:
   print_student - prints students general information
   print_eval - print the evaluation questions

2.2.3.5.4.1) print_student

Description:
This routine will print the global questions on the students.
A pointer to the instructor's information is passed to print
out the heading. The routine prints out the first seven
questions in the questionnaire form. These questions are
general questions about the students taking the class.
Input:
A pointer to the instructor record
the global statistics record
the opened output file

Output:
The first seven questions of the report

This routine is called by:
  print_global - prints the global questions
The routine calls:
  line - writes a line with the given character

2.2.3.5.4.2) print_eval

Description:
This routine will create the questions reports for the evaluation questions in the global questions. The routine calls a routine to print the questions. The question is changed each time the routine is called.

Input:
the global statistics record
the opened output file

Output:
the evaluation questions of the report (8-10)

This routine is called by:
  print_global - prints the global questions report
The routine calls:
  print_quest - prints a question

2.2.3.5.4.2.1) print_quest

Description:
This routine will print the standard evaluation questions and the instructor picked questions. The question to be printed is passed as a parameter along with the number of the question to be printed and the statistics for the question. The routine displays a different heading for the standard evaluation questions and the instructor picked questions.

Input:
quest_num : integer
quest : the question to be printed
response : type of response for a heading
q_stats : the statistics for the question
outfile : the opened output file
Output:
the question is written to the output file with the above information.

This routine is called by:
   print_eval - print evaluation questions
   print_instructor - prints the instructor picked questions

The routine calls:
   print_mean - prints the mean for the question
   print_con - prints the consensus for the question

2.2.3.5.4.2.1.1) print_mean

Description:
This routine will write an interpretation of the value for the mean.

Input:
mean : real

Output:
A written form of the mean

This routine is called by:
   print_quest - prints and instructor question

2.2.3.5.4.2.1.2) print_con

Description:
This routine will write an interpretation of the value for the standard deviation.

Input:
stand : real

Output:
A written form of the standard deviation

This routine is called by:
   print_quest - prints and instructor question

2.2.3.5.5) print_instructor

Description:
This routine receives a pointer to the instructor's information as input, and has the output files passed to it. This routine will go through the list of the instructor's picked questions and finds them in the question list. The procedure print_quest is then called to print out the question.
Input:
A pointer to the instructor record
the question table
the output file number
the stats and output files

Output:
the instructor's picked questions are written to the output file.

The routine is called by:
    print_report - prints the reports
The routine calls:
    get_num - converts an array to an integer number
    print_ques - prints a question

2.2.4) Mult_entry

Description:
This routine will generate the report files for any number of files. The number of the low and high files are entered. These numbers are used to obtain the statistics files and to create the output files. All error messages are written to the file error.fil

Input:
The low statistics file number
The high statistics file number

Output:
The reports for all of the files

This routine is called by
    the main routine
The routine calls:
    get_number - reads in an integer number
    get_file - get the appropriate output file
    get_header - read in the header record
    find_rec - find the instructor record in the dat base
    print_report - print the reports
Design document for the third part of the faculty evaluation system.

3) quest

Description:
This routine is used to maintain the question data base and the standard forms data base. The user has the options to modify multiple questions of single questions. The user can also delete, add, and print questions. For the standard forms the user can add, delete and print the standard forms. The main routine allows the user to enter the modification option for the file and then calls the appropriate subroutine to perform the task.

Input:
The question data base
The standard forms data base

Output:
The updated question data base
The updated standard form data base

This routine calls the following subroutines:
- build_question_list - puts the questions into a linked list
- get_forms - reads in the standard forms
- heading - prints the heading and menu
- mult_quest - lets the user modify multiple questions
- single_quest - lets the user modify a single question
- delete_quest - deletes a question
- add_quest - adds a question
- print_quest - prints the question data base
- add_stan_form - adds a standard form
- delete_stan_form - deletes a standard form
- print_stan_form - prints the standard forms
- dump_question_list - writes the question list to an output file
- dump_standard_forms - writes the standard forms to an output file

3.1) build_question_list

Description:
This routine reads in the questions from the question data base. The questions are placed into a linked list.

Input:
A file of questions
Output:
head : a pointer to the head of the question list

where questions consist of the following
  question number : packed array [1..4] of char;
  question string 1 : packed array [1..43] of char;
  question string 2 : packed array [1..43] of char;

This routine is called by:
  the main routine
This routine calls:
  get_quest - reads in a question

3.1.1) get_quest

Description:
This routine reads in a question from the question database.

Input:
A question record

Output:
A pointer to the question record

This routine is called by:
  build_question_list - builds the question list

3.2) get_form

Description:
This routine reads in the standard forms into a linked list.
The standard forms are read in from a file "stan.frm".

Input:
A file of standard forms

Output:
A linked list of standard forms

where a standard form consists of:
  A form number : packed array [1..4] of char
  question_number : array [1..23] of question numbers

This routine is called by:
  the main routine

3.3) heading

Description:
This routine writes the heading message for the question
maintainance routine. It also displays a list of options for modifying the questions and the standard forms.

Input:
none

Output:
The program heading and menu

This routine is called by:
the main routine

3.4) mult_quest

Description:
This routine allows the user to modify the existing questions in the data base. The user will type in a replacement for the questions. The old question is overwritten.

Input:
start question number
end question number

Output:
The updated question list

This routine is called by:
the main routine
The routine calls:
convert - converts an integer to an array of characters
mod_question - lets the user modify the question

3.4.1) convert

Description:
This routine receives an integer number as input and converts it to an array [1..4] of characters.

Input:
num : integer

Output:
key : packed array [1..4] of char

This routine is called by:
mult_quest - lets the user modify multiple questions
single_quest - lets the user modify a single question
delete_quest - deletes a question
add_quest - adds a question
add_stan_form - adds a standard form
deleteStan_form - deletes a standard form

3.4.2) mod_question

Description:
This routine allows the user to modify a question. The routine
is passed a pointer to the question that is to be modified.
The routine displays the question an asks the user if it is to
be modified. If the user chooses to modify the question he
enters in the new question.

Input:
A pointer to the question

Output:
The updated question

This routine is called by:
mult_quest - lets the user modify multiple questions
single_quest - lets the user modify a single question

The routine calls:
get_string - reads in a question string

3.4.2.1) get_string

Description:
This routine reads in a string for the question. The maximum
length of the string is set at 43 characters. The routine
will ignore all input past 43 characters. The questions in
the data base are composed of two strings.

Input:
text from the terminal

Output:
string : packed array [1..43] of char

This routine is called by:
mod_question - modify a question
add_quest - adds a question

3.5) single_quest

Description:
This routine allows the user to modify a single question. The
user enters the question that is to be modified. The old
question will be overwritten by the new question.

Input:
The head of the question list
3.6) delete_quest

Description:
This routine allows the user to delete a question from the data base. The user enters in the number of the question to be deleted. The routine then finds the question in the data base and calls delete to display the question.

Input:
A pointer to the beginning of the question list

Output:
The updated question list

This routine is called by:
the main routine
The routine calls:
convert - converts an integer number to an array of characters
delete - displays and deletes the question

3.6.1) delete

Description:
This routine displays the question to be deleted. The routine gives the user the option of deleting the question or returning to the main routine.

Input:
A pointer to the question to be deleted

Output:
The updated question list

This routine is called by:
delete_quest - delete a question

3.7) add_quest

Description:
This routine allows the user to add a question to the question
list. The routine adds the question number onto the end of the question list.

Input:
A pointer to the beginning of the question list

Output:
The updated question list

This routine is called by:
the main routine

The routine calls:
get_num - convert a character array to an integer
convert - converts an integer number to a character array
get_string - gets a string for the question

3.7.1) get_num

Description:
This routine receives an array of digits and converts them into an integer number:

Input:
key : packed array [1..4] of char;

Output:
num : integer

This routine is called by:

3.8) print_quest

Description:
This routine writes all of the questions in the data base to a file "quest.rpt". The routine writes headers on the top of a page an draws a line of stars between each question.

Input:
head_list : a pointer to the beginning of the question list

Output:
A file containing the questions

This routine is called by;
the main routine

3.9) add_stan_forms

Description:
This routine allows the user to add a new standard form to the
standard form data base. The routine displays the new standard form number and the prompts the user for entry of the question numbers for the form. The user may type at most 23 questions for a standard form. If the standard form is to contain less than 23 questions then the user may type a zero for the last question number and the rest of the standard form will be filled in with zeros.

Input:
The head of the standard form list
the new standard form

Output:
The updated standard form list

This routine is called by:
the main routine
The routine calls:
    char_to_dig - character to digit conversion
    convert - convert an integer number to an array of characters
    init_quest_array - initialize the question array to all zeros

3.9.1) char_to_dig

Description:
This routine receives an array of characters and converts it into an integer number.

Input:
key : packed array [1..4] of char

Output:
num : integer

This routine is called by:
    add_stan_form - adds a standard form

3.9.2) init_quest_array

Description:
This routine initializes the array of standard form numbers to be all zeros.

Input:
A pointer to the standard form

Output:
The updated standard form
3.10) delete_stan_form

Description:
This routine allows the user to delete a standard form. The routine is passed a pointer to the beginning of the standard form list. The user then enters the standard form to be deleted. The user has the option of deleting the standard form or returning to the main routine.

Input:
A pointer to the beginning of the standard form list

Output:
The updated standard form list

This routine is called by:
the main routine
The routine calls:
convert - converts and integer number to an array of characters
d_form - displays and deletes a standard form

3.10.1) d_form

Description:
This routine will display the question numbers in the standard form to be deleted. The user then has the option of deleting the form or returning to the main routine.

Input:
A pointer to the standard form to be deleted

Output:
The updated standard form list

This routine is called by:
delete_stan_form - delete a standard form

3.11) print_stan_form

Description:
This routine uses the question numbers in the standard forms to locate the questions in the question list. The routine then prints the questions that are in the standard forms.

Input:
A pointer to the beginning of the standard form list
Output:
A text file containing all of the standard forms

This routine is called by:
    the main routine
The routine calls:
    line - draws a line of characters

3.11.1) line

Description:
This routine draws a line of characters for the length of the
number passed to it with the letter passed to it.

Input:
ch : char
num : integer

Output:
A line drawn with character ch of length num

This routine is called by;
    print_stan_forms - print the standard forms

3.12) dump_question_list

Description:
This routine writes the updated version of the question list
to a file. The file that it is written to will overwrite the
file that the questions were originally read in from.

Input:
A pointer to the beginning of the question list

Output:
A text file containing the questions

This routine is called by:
    the main routine

3.12) dump_standard_forms

Description:
This routine writes the updated version of the standard forms
to a file. The file that it is written to will overwrite the
file that the standard forms were originally read in from.

Input:
A pointer to the beginning of the standard form list

Output:
A text file containing the standard forms

This routine is called by:
the main routine
This routine is an addition to the original design of the faculty evaluation system.

4) edstat

Description:
This routine allows the user to edit the header card on the statistics file. The statistics files had a large number of errors in the header record this routine was written so the user did not have to enter in the information again.

Input:
A statistics file

Output:
The updated statistics file

The routine calls the following subroutines:
  get_num - reads in an integer number
  get_file - opens a statistics file
  read_stats - reads in the statistics file
  edit_stats - allows the user to alter the header record
  write_stats - writes out the updated statistics file

4.1) get_num

Description:
This routine reads in an integer number from the terminal. If an invalid character is entered the user has to reenter the number.

Input:
Text from the terminal

Output:
An integer number

This routine is called by:
  the main routine
  edit_stats - allows the user to alter the header record

4.2) get_file

Description:
This routine will open a statistics file. The number that is passed to it is the number of the statistics file to be opened.
Input:
num : integer

Output:
The opened statistics file

This routine is called by:
the main routine

4.3) read_stats

Description:
This routine will read in the information from the statistics file. The header record is read into a record. The rest of the file is then read into a linked list.

Input:
A statistics file

Output:
The header record
The statistics information

This routine is called by:
the main routine

4.4) edit_stats

Description:
This routine displays the header record and asks the user if it is to be changed. The user has the option of changing the header record or returning to the main routine. The header record is displayed again after the change has been made.

Input:
The header record

Output:
The updated header record

This routine is called by:
the main routine
This routine calls:
get_num - read in an integer number
convert3 - convert an integer number into an array of characters

4.4.1) convert3

Description:
This routine converts the integer number passed to it into an
array [1..3] of characters.

Input:
num : integer

Output:
letters : packed array [1..3] of char;

This routine is called by:
   edit_stats - alters the header record for a statistics file

4.5) write_stats

Description:
This routine writes the updated information back to the original statistics file. The old statistics file is deleted in the process.

Input:
header - statistics header record
head_stats - a pointer to the statistics

Output:
The updated statistics file

This routine is called by:
   the main routine
Program Flow Diagrams

Notation

A process

A data flow

A file

A local file

PROG

A program
Generate Forms

Diagram:

- **Terminal Entry**
- **SRT**
- **BLDIST**
- **Instr.rpt**
- **Instr.srt**
- **Quest.dt9**
- **Bat.num**
- **Pr cmd .cmd**
- **Qu0000.dt9**
- **Qu9999.dt9**
Generate Reports

File0.scn ... File9.scn (Scantron Files)

STATS

IN0000.dta ... IN9999.dta

EDSTAT

Inst.srt

REPORT

O0000.dta ... O9999.dta
Question Maintenance

Terminal Entry

QUEST

Quest.dta

Quest.rpt

Stan.Frm

Stan.rpt
APPENDIX E

PROGRAMS

; The login command file sets up all of the logical names that
; will be needed in the programs for the faculty evaluation system.
; The command file also connects the user to psl: where all of
; the programs and data areas are located.
;
def sys: psl:<faculty-evaluation>, psl:<faculty-evaluation.source>, sys:
def man: psl:<faculty-evaluation.manuals>
def fac: psl:<faculty-evaluation>
def src: psl:<faculty-evaluation.source>
def sts: psl:<faculty-evaluation.stats>
def rpt: psl:<faculty-evaluation.report>
def cs: ps:<cs.grad.weiler>

ter no ra
ter no pau en
cookie
access psl:<faculty-evaluation>
i mail
connect psl:<faculty-evaluation>
;This commands file runs the program to enter instructor
;information into the instructor data base. After the
;program has been exited the routine submits a batch job
;to sort the instructor records according to the department,
course, and section number.
;
run bldist
submit srt.ctl
;This routine will print out all of the evaluation reports that are currently in the report area. The routine submits the reports for printing with all of the appropriate switches.

; print /note:"hold&B&c"/delete/forms:bk9713 rpt:ot*.*
print /note:"hold" psi:<faculty_evaluation.report>error.fil
program build_instructor_database;
{
BLDIST.PAS - Maintain instructor data base

Written by: Peter Weiler
Final modification date: May 20, 1983

This program is used to maintain the instructor data base. The program will display a menu of options for modifying the instructor data base. The program will then call the appropriate procedure to perform the task. This routine allows the user to add, delete, and print records in the instructor data base. The routine will also automatically back up the instructor data base after every ten changes have been made to it.

The information for the instructor data base is stored in the file "INST.SRT". This file should be sorted when the program reads it in. All new instructor records are added to the bottom of the file. The file will be resorted after this program is exited. The input data file is in the following format:

department_code : field (1 - 4)
course_number : field (5 - 7)
section_number : field (8 - 11)
initial : field (12)
name : field (13 - 26)
department_abbreviation : field (27 - 31)
print_flag : field (32 - 36)
number_of_questions : field (37 - 39)
question_numbers : field (40 - 131)

The instructor maintainance program also uses a file that contains the standard forms. This file is called "STAN.FRM". The format for the file is as follows:

form_number : fields(1 - 4)
question_numbers : fields(5 - 96)

where the question number field contains 23 question numbers with each question number taking up four characters.

}
const
key_length = 4;
um_length = 3;
max_quest_nums = 23;
name_length = 14;
d_abr_length = 5;

type
key_type = packed array [1..key_length] of char;

num_type = packed array [1..num_length] of char;

quest_list = array [1..max_quest_nums] of key_type;

form_ptr = ^form_type;

form_type = record
    form_num : key_type;
    quest_nums : quest_list;
    next : form_ptr
end;

name_type = packed array [1..name_length] of char;

d_abr_type = packed array [1..d_abr_length] of char;

inst_list = ^inst_rec;

(* The instructor record is read in from the file inst.srt *)
inst_rec = record
    dept_code : packed array [1..4] of char;
    course_num : num_type;
    sect_num : num_type;
    inst_init : char;
    inst_name : name_type;
    dept_abr : d_abr_type;
    print_flag : char;
    num_of_quest : num_type;
    quest_nums : quest_list;
    next : inst_list
end;

var
form_file, in_inst, out_inst : text;
head_form : form_ptr;
head_inst : inst_list;
option : char;
eof_flag : boolean;
changes : integer;

procedure line (length : integer; ch : char; var outfile : text);

(* The procedure will write a line for the given character ,ch, and length ,length, for file ,outfile *)

var
i : integer;

begin
for i := 1 to length do
  write (outfile,ch)
end; (* line *)

procedure get_forms (var stan_forms : form_ptr);

{ This routine reads in the standard questionnaire records
  into a linked list. Each standard questionnaire consists of
  a form number and 23 question numbers }

var
  i : integer;
  temp_f_ptr : form_ptr;

begin { get_forms }
  reset (form_file,'stan.frm');
  new (stan_forms); { set head of form list }
  temp_f_ptr := stan_forms;
  temp_f_ptr^.form_num := '0000';
  temp_f_ptr^.next := nil;
  while not eof (form_file) do
    begin { read in a standard form }
      read (form_file,temp_f_ptr^.form_num);
      for i := 1 to max_quest_nums do
        read (form_file,temp_f_ptr^.quest_nums[i]);
      readln (form_file);
      if not eof (form_file)
        then
          begin
            new (temp_f_ptr^.next);
            temp_f_ptr := temp_f_ptr^.next
          end
        else
          temp_f_ptr^.next := nil { set end of list }
    end
  end; { get_forms }

procedure convert4 (num : integer; var key : key_type);

{ This routine converts an integer number ,num, to an
  array of characters ,key, The array has a range of [1..4].
}

var
  i : integer;

begin { convert 4 }
  for i := key_length downto 1 do
    begin
      key[i] := chr((num mod 10) + ord(′0′));
num := num div 10
end
end; (* convert4 *)

procedure convert3 (num : integer; var letters : num_type);
{
This routine converts an integer number, num, to an array of
characters, letters. The array has a range of [1..3].
}
var
i : integer;
begin { convert3 }
for i := num_length downto 1 do
begin
letters[i] := chr((num mod 10) + ord('0'));
num := num div 10
end
end; (* convert3 *)

procedure get_name (var name : name_type);
{
This routine reads in the instructor's name and puts it into an
array [1..14] the routine will fill in the right side of the array with
blanks. The routine also converts all lower case characters to upper
case characters.
}
var
ch : char;
count : integer;
begin { get_name }
name := '  ';
count := 1;
ch := input
get (input);
if not eoln
then
name[1] := ch;
while not eoln and (count < name_length) do
begin
count := count + 1;
ch := input
get (input);
if (ch >= 'a') and (ch <= 'z')
then
ch := chr(ord(ch) - ord('a') + ord('A'));
name[count] := ch;
end
end;
readln}
end; { get_name }

procedure get_dept_abr (var dept_abr : d_abr_type);

{ This routine will read in the department name into an array [1..5]. The routine will fill in the right side of the department abbreviation with blanks. The routine also converts lower case characters to uppercase. }

var
  ch : char;
  count : integer;

begin { get_dept_abr }
  dept_abr := ' ';
  count := 1;
  ch := input^;
  get (input);
  if not eoln
    then
      dept_abr[1] := ch;
  while not eoln and (count < d_abr_length) do
    begin
      count := count + 1;
      ch := input^;
      if (ch >= 'a') and (ch <= 'z')
        then
          ch := chr(ord(ch) - ord('a') + ord('A'));
      dept_abr[count] := ch;
      get (input)
    end;
  readln
end; { get_dept_abr }

procedure get_rec (var rec : inst_list;var ins : text);

(* This procedure will read in the information on one instructor. *)

var
  i : integer;

begin (* get_rec *)
  new (rec);
  with rec^ do
    begin
      read (ins,dept_code,course_num,sect_num);
      read (ins,inst_init,inst_name,dept_abr);
      read (ins,print_flag,num_of_quest);
    end;

end;
for i := 1 to max_quest_nums do
  (* get instructor picked questions *)
  read (ins,quest_nums[i]);
  readln (ins);
  next := nil
end
end; (* get_rec *)

procedure get_instruct_info(var head_inst_list : inst_list);

(* This procedure builds the linked list of instructor
  information. The input file 'INST.SRT' should be a file that
  has been sorted according to the department, course, and
  section number. *)

var
  temp,rec : inst_list;

begin (* get_instruct_info *)
  (* sorted instructor information file *)
  reset (in^inst,'inst.srt');
  if not eof (in_inst)
  then
    begin
      get_rec (head_inst_list, in_inst);
      temp := head_inst_list;
      (* get all instructor information *)
      while not eof (in_inst) do
        begin
          get_rec (rec,in_inst);
          temp^.next := rec;
          temp := temp^.next
        end;
      eof_flag := false
    end
  else
    eof_flag := true { instructor file is empty }
end; (* get_instruct_info *)

procedure get_num (var num : integer);

(* This procedure reads in a string of digits from the
  terminal and converts them to an integer number. The routine
  checks to see if the input is only the characters 0..9 if
  the user types in a different character the routine will
  print an error message and have the user reenter the
  information. *)

var
correct : boolean;
ch : char;

begin { get_num }
   repeat { until correct }
      correct := true;
      num := 0;
      repeat { until wrong or eoln }
         ch := input;
         get (input);
         if (ch >= '0') and (ch <= '9')
            then
               num := num * 10 + (ord(ch) - ord('0'))
            else
               begin
                  correct := false;
                  readln;
                  write ('invalid number reenter: ')
               end
            until (eoln or not correct);
      if correct
         then
            readln
      until correct
end; { get_num }

procedure get_picked_quests (var quests : quest_list);

   {  This procedure will read in the instructor's picked questions. The numbers for the questions are entered from the terminal. The routine allows for a maximum of 23 characters to be entered. If the user has fewer questions to enter he types a zero for the last question and the routine will exit.  }

   var
      temp_f_ptr : form_ptr;
      i, quest_num, form_num, num_of_quest : integer;
      answer : char;

   procedure init_quest_array (var quests : quest_list);

      {  This procedure initializes the list of instructor picked questions to be all zeros.  }

      var
      i : integer;

      begin { init_quest_array }
      for i := 1 to max_quest_nums do
quests[i] := '0000'
end; { init_quest_array}

procedure char_to_dig (var num : integer; var key : key_type);
{ This procedure will convert an array [1..4] of digits to a integer number. }
var
  i : integer;
begin { char_to_dig }
  num := 0;
  for i := 1 to key_length do
    num := num * 10 + (ord(key[i]) - ord('0'))
  end; { char_to_dig }
begin { add_stan_forms }
  writeln;
  writeln ('Enter the question numbers for the instructor');
  writeln ('If you have fewer than 23 questions type a ,0, as the final question number to stop.');
  writeln;
  num_of_quest := 1;
  quest_num := 1;
in init_quest_array (quests);
  { exit the while loop if a question number is zero or if the number of questions is greater than 23 }
while (quest_num <> 0) and (num_of_quest <= max_quest_nums) do
begin
  write ('Enter the number of question',num_of_quest:3,' :');
  get_num (quest_num);
  convert4 (quest_num,quests[num_of_quest]);
  writeln;
  num_of_quest := num_of_quest + 1
end;
end; { add_stan_forms }

procedure forms_quest (var quests : quest_list;form : integer);
{ This procedure receives a form number an looks through the list of forms until it finds the form number. The routine then writes the question numbers from the standard form into the question list in the instructor record. }
var
  temp_form : form_ptr;
i : integer;
form_num : key_type;

begin { forms_quest }
temp_form := head_form;
convert4 (form,form_num);

{ find the correct form number }

while (temp_form^.form_num <> form_num) and
(temp_form^.next <> nil) do
  temp_form := temp_form^.next;
if (temp_form^.form_num <> form_num)
  then
    writeln ('Form number not found')
  else
    for i := 1 to max_quest_num do
      quests[i] := temp_form^.quest_nums[i]
end; { forms_quest }

procedure add_inst (var head_inst : inst_list);

{ This procedure will read in the information about the
  instructor and read in either the form or question numbers
  that he selected. It will also allow the information to be
  reviewed before accepting it. This routine calls the
  appropriate subroutine to read in each of the fields in the
  instructor record so that all input is parsed. }

var
  temp_i_ptr : inst_list;
  num : integer;
  answer : char;

begin { add_inst }
  writeln;
  writeln ('Enter the instructor information.');
  writeln;
  if not eof_flag
  then
    begin
      temp_i_ptr := head_inst;

      { find last place }
      while (temp_i_ptr^.next <> nil) do
        temp_i_ptr := temp_i_ptr^.next;
      new (temp_i_ptr^.next);
      temp_i_ptr := temp_i_ptr^.next
    end
  else
begin { file is empty }
eof_flag := false;
new (temp_i_ptr);
head_inst := temp_i_ptr
end;
temp_i_ptr^ . next := nil;

repeat { until correct }
{ get information }
temp_i_ptr^ . print_flag := '0';
write ('Enter the instructor's name: '); get_name (temp_i_ptr^ . inst_name);
write ('Enter instructor's initial: '); readln (temp_i_ptr^ . inst_init);
write ('Enter the dept. abr.: '); get_dept_abr (temp_i_ptr^ . dept_abr);
write ('Enter the dept. no.: '); get_num (num);
convert4 (num,temp_i_ptr^ . dept_code);
write ('Enter the course number: '); get_num (num);
convert3 (num,temp_i_ptr^ . course_num);
write ('Enter the section number: '); get_num (num);
convert3 (num,temp_i_ptr^ . sect_num);
write ('Enter number of forms: '); get_num (num);
convert3 (num,temp_i_ptr^ . num_of_quest);
writeln;
write ('Is information correct? (Y or N) '); readln (answer)
until (answer = 'y') or (answer = 'Y');

repeat { until questions correct }
writeln;
write ('Enter form number, (0 if picked questions): '); get_num (num);
if (num = 0) then get_picked_quest (temp_i_ptr^.quest_nums) 
else forms_quest (temp_i_ptr^.quest_nums,num);
writeln;
write ('Is information correct? (Y or N) '); readln (answer)
until (answer = 'y') or (answer = 'Y');
end; { add_inst }

procedure delete_inst (var head_inst : inst_list);

{ This procedure allows the user to delete an instructor
record. The routine has the user enter the instructor's name and the course that he is teaching. The procedure will then display all records that have the information in them. The procedure will allow the user to view duplicate records in this manner.

```pascal
var
temp_i_ptr : inst_list;
c_num : integer;
found,done : boolean;
course : num_type;
name : name_type;

procedure d_inst(var inst : inst_list);
{
This procedure displays the complete instructor record so that the user can see if the record is to be deleted. If the user chooses to delete the record then the record is removed from the linked list.
}

var
answer : char;
i,offset,half_quests : integer;

begin (* d_inst *)
writeln;
writeln ('Instructor name: ',inst^.inst_name);
writeln ('Instructor initial: ',inst^.inst_init);
writeln ('Department abr: ',inst^.dept_abr);
writeln ('Department no: ',inst^.dept_code);
writeln ('Course number: ',inst^.course_num);
writeln ('Section number: ',inst^.sect_num);
writeln ('No. of questions: ',inst^.num_of_quest);
offset := (max_quest_nums div 2) + max_quest_nums mod 2;
half_quests := max_quest_nums div 2;
with inst^ do
begin
for i := 1 to half_quests do
begin
write (' ',i:2,') ',quest_nums[i]);
writeln (' ':10,i+offset:2,') ',quest_nums[i+offset]);
end;
if odd(max_quest_nums)
then
writeln (' ',half_quests+1:2,') ',quest_nums[half_quests+1])
end;
writeln;
write ('Do you wish to delete this instructor? (Y or N) '); readln (answer);
```
if (answer = 'y') or (answer = 'Y')
    then
        inst := inst^.next
    end; (* d_inst *)

begin (* delete instructor *)
    done := false;
    writeln;
    write ('Enter instructor's name being deleted: ');
    get_name (name);
    write ('Enter instructor's course number: ');
    get_num (c_num);
    temp_i_ptr := head_inst;
    convert3 (c_num,course);
    (* check if first instructor *)
    if (course = temp_i_ptr' . course_num) and
        (name = temp_i_ptr' . inst_name)
    then
        d_inst (head_inst)
    else
        found := false;
        begin { until end of list }
            repeat { find the instructor record }
                while (temp_i_ptr' . next' . next <> nil) and
                    (course <> temp_i_ptr' . next' . course_num) and
                    (name <> temp_i_ptr' . next' . inst_name) do
                    temp_i_ptr := temp_i_ptr' . next;
                    if (course = temp_i_ptr' . next' . course_num) and
                        (name = temp_i_ptr' . next' . inst_name)
                    then
                        begin
                            found := true;
                            d_inst (temp_i_ptr' . next)
                        end;
                        if (temp_i_ptr' . next = nil)
                    then
                        begin
                            temp_i_ptr := head_inst;
                            done := true
                        end
                    end
                else
                    if (temp_i_ptr' . next' . next <> nil)
                then
                    temp_i_ptr := temp_i_ptr' . next
                    until ((temp_i_ptr' . next' . next = nil) or done);
                    if not found
                then
                    writeln ('Instructor ',name,' not found.')
procedure print_inst (var instructor : inst_list);

(* This procedure was written to help in making the header cards that go on the top of the student's answer cards. The header cards consist of the department number, the course number, and the section number. This card is then used in the report file to find the instructors picked questions. This program will print out the department number, the course number, the section number, the instructor's name, the department and the batch number. The output of this program is formatted so that corrections and deletions can be made to the instructor data file.

The input file to this program is the sorted instructor data file. This file should be sorted on the following keys, the department number, the course number, and the section number.

The output file is a temporary file that is handy to have for the data entry personal to check there scantron header cards. *)

var
  linecount, pagenum : integer;
  temp_ptr : inst_list;

procedure print_heading (page_num : integer);

{ This procedure will print the heading for the instructor report file.
}

var
  i : integer;

begin
  writeln (out_inst);
  writeln (out_inst,' ':60,'page ' ,pagenum:2);
  write (out_inst,' ':3,'Dept. no ','course no ',' sect. no '); 
  write (out_inst,' instructor ','forms',' d. abr');
  writeln (out_inst,' ':2,'prtd');
  for i := 1 to 70 do
    write (out_inst,'*');
  writeln (out_inst)
end;

procedure write_line (var instrt : inst_list);
{ This procedure writes a complete instructor record to the instructor report file. }

var
 i : integer;

begin
 with instrt do
 begin
   begin
     write (out_inst,' ':6,dept_code,' ':6,course_num);
     write (out_inst,' ':7,sect_num);
     write (out_inst,' ':5,inst_init,' ',inst_name,' ');
     write (out_inst,num_pf_quest,' ':3,dept_abr);
     write (out_inst,' ':4,print_flag,chr(13));
     line (70,' ',out_inst);
     writeln (out_inst);
     for i := 1 to (max_quest_nums div 2) do
       write (out_inst,quest_nums[i],' ');
     writeln (out_inst);
     for i := (max_quest_nums div 2 + 1) to max_quest_nums do
       write (out_inst,quest_nums[i],' ');
     writeln (out_inst);
   end;
 end; { print_heading }

begin (* print_inst *)
 rewrite (out_inst,'instr.rpt');
 writeln;
 writeln ('Created file instr.rpt');
 writeln;
 temp_ptr := instructor;
 pagenum := 1;

{ while the instructor list is not empty perform the following loop. }

while (temp_ptr <> nil) do
 begin
   linecount := 0;
   print_heading (pagenum);
   while (linecount <= 50) and (temp_ptr <> nil) do
     begin
       write_line (temp_ptr);
       linecount := linecount + 4;
       temp_ptr := temp_ptr^.next
     end;
   page (out_inst);
   pagenum := pagenum + 1
procedure dump_inst (var head_inst : inst_list);

{ This procedure writes the updated record to the instructor database file. }

var
temp_i_ptr : inst_list;
i : integer;

begin { dump_inst }
{ rewrite the instructor file }
rewrite (out_inst,'inst.srt');
temp_i_ptr := head_inst;
while (temp_i_ptr <> nil) do
begin
with temp_i_ptr do

begin
write (out_inst,dept_code,course_num,sect_num);
write (out_inst,inst_init,inst_name,dept_abr);
write (out_inst,print_flag,num_of_quest);
for i := 1 to max_quest_nums do
(* write instructor picked questions *)
write (out_inst,quest_nums[i]);
writeln (out_inst)
end;

end; { dump_inst }

procedure reset_files (var head_list : inst_list);

begin { reset_files }
dump_inst (head_list);
end; { reset_files }

procedure heading;

{ This procedure prints out the heading and the menu of options for this program. }

begin { heading }
writeln;
writeln ('This program modifies the list of instructors');
writeln ('using the faculty evaluation system. This program');
writeln ('contains the following options: ');}
writeln (′ 1) A - add an instructor′);
writeln (′ 2) D - delete an instructor′);
writeln (′ 3) P - creates a printable instructor file′);
writeln (′ 4) ? - prints this menu′);
writeln
end; (* heading *)

begin (* main *)

g et_forms (head_form);
g et_instruct_info (head_inst);

heading;

changes := 0;
repeat (* until you want to stop *)
  if (changes = 10)
    then
    begin
      writeln (′Wait resetting file′);
      changes := 0;
      reset_files (head_inst)
    end
  else
  changes := changes + 1;
write (′Enter option : (A,D,P,?) ′);
readln (option);
case option of
  ′a′,′A′ :
    add_inst (head_inst);
  ′d′,′D′ :
    delete_inst (head_inst);
  ′p′,′P′ :
    print_inst (head_inst);
  ′?′ :
    heading;
  others :
    begin
      writeln;
      writeln (′Invalid command′)
    end
  end; (* case *)
writeln;
write (′Do you want to continue? (Y or N) ′);
readln (option)
until (option = ′n′) or (option = ′N′);
dump_inst (head_inst)
end.
(* main *)
program print_questions;
{
    Written by: Peter Weiler
    last revised: May, 1983

    This program creates the instructor's evaluation form.
The routine will also create a file for printing out the
questionnaire forms.

    The information for the instructor data base is stored in the
file 'INST.SRT' this file should be sorted when the program reads
it in. The input data file is in the following format:

department code : field (1 - 4)
course number : field (5 - 7)
section number : field (8 - 11)
instructor initial : field (12)
instructor name : field (13 - 26)
department abbreviation : field (27 - 31)
print flag : field (32 - 36)
number of questions : field (37 - 39)
question numbers : field (40 - 131)

    The routine reads in the questions for the questionnaires
from a file 'QUEST.DTA'. This file is set up in the following
format:

    key : field (1 - 4) the question number
    string1 : field (5 - 47) first half of the question
    string2 ; field (47 - 90) second half of the question

    The question data base will be read into a hash table so
that the questions can be accessed rapidly.

    The program will create a file call 'PRCMD.CMD' this file
will be a print commands file. The file will be used to
submit the print requests with all of the appropriate
switches.

    This routine will create a file for each questionnaire for
a file, 'BAT.NUM', will be used to keep track of the
questionnaire file number. The questionnaire forms will be
written to a subdirectory with the logical name of 'RPT:'.
}

const
    key_length = 4;
    num_length = 3;
    max_quest_nums = 23;
    name_length = 14;
    d_abr_length = 5;
string_length = 43;
max_tot_quest = 9999;

type

key_type = packed array [1..key_length] of char;
num_type = packed array [1..num_length] of char;
string_type = packed array [1..string_length] of char;
questions = array [1..max_quest_nums] of key_type;
quest_list = ^quest_type;

quest_type = record
  key : key_type;
  quest1_string : string_type;
  quest2_string : string_type
end;

{ question list table }
q_tab_type = array [1..max_tot_quest] of quest_list;

name_type = packed array [1..name_length] of char;
d_abr_type = packed array [1..d_abr_length] of char;

inst_list = ^inst_rec;

(* The instructor record is read in from the file inst.srt *)
inst_rec = record
  dept_code : packed array [1..4] of char;
  course_num : num_type;
  sect_num : num_type;
  inst_init : char;
  inst_name : name_type;
  dept_abr : d_abr_type;
  print_flag : char;
  num_of_quest : num_type;
  quest_nums : questions;
  next : inst_list
end;

file_type_string = packed array [1..14] of char;

var

quest_file, prcmd_fil, inprcmd : text;
instr_file, out_inst, out_q, batch : text;
inst_head : inst_list;
quest_table : q_tab_type;
out_string : file_type_string;

procedure init_table (var q_tab : q_tab_type);
{
  This procedure initializes the pointers in the question table to be nil.
}
var
  i : integer;
begin { init_table }
  for i := 1 to max_tot_quest do
    q_tab[i] := nil
end; { init_table }

procedure get_num(var key : key_type; var num : integer);
{
  This procedure will convert an array [1..4] of digits to an integer number.
}
var
  i : integer;
begin { get_num }
  num := 0;
  for i := 1 to key_length do
    num := num * 10 + (ord(key[i]) - ord('0'))
end; { get_num }

procedure get_out_file (var outfile : text);
{
  This procedure will create the output files for the questionnaire forms, it will also increment the batch number in the batch file so that each call to this program increments this number. *)
}
var
  batch_num : integer;
begin { init_out_string }
  out_string := "RPT:QU0000.DTA"
end; { init_out_string }
begin (* get_out_file *)
(* set batch number *)
(* make sure batch is initialized to zero *)

reset (batch,"bat.num");
if eof(batch)
  then
  batch_num := 0 { if the file is not there initialize }
else
  read (batch,batch_num);
  batch_num := batch_num + 1;
  rewrite (batch,"bat.num");
  writeln (batch,batch_num);

(* create a file name for the output file *)

init_out_string (out_string);  
out_string[7] := chr(batch_num div 1000 + ord('0'));  
{ insert file number }

out_string[8] := chr((batch_num div 100) mod 10 + ord('0'));  
out_string[9] := chr((batch_num div 10) mod 10 + ord('0'));  
out_string[10] := chr(batch_num mod 10 + ord('0'));  
rewrite (outfile,out_string)
end; (* get_out_file *)

procedure build_question_list(var q_tab : q_tab_type);

{ This procedure will create the table of questions that 
  will be used in the questionnaire forms. The routine will 
  set up the table to be indexed by the question's number. }

var
  q_rec : quest_list;
  index : integer;

procedure get_quest (var q_rec : quest_list;var qfile : text);

(* This procedure will read in one question from the question 
  file. *)

begin { get_quest }
  with q_rec do
  begin
    read (quest_file,key);
    readln (quest_file,quest1_string,quest2_string)
  end
end; (* get quest *)
begin (* build question list *)
(* get all of the questions in the data pool *)
reset (quest_file,'quest.dta');
while not eof (quest_file) do
begin
new (q_rec);
get_quest (q_rec,quest_file);
get_num (q_rec~.key,index);
q_tab[index] := q_rec
end
end; (* build_data_base *)

procedure get_rec (var rec : inst_list;var ins : text);

(* This procedure will read in the information on
one instructor. *)
var
i : integer;

begin (* get_rec *)
new (rec);
with rec~ do
begin
read (ins,dept_code,course_num,sect_num);
read (ins,inst_init,inst_name,dept_abr);
read (ins,print_flag,num_of_quest);
for i := 1 to max_quest_nums do
(* get instructor picked questions *)
read (ins,quest_nums[i]);
readln (ins);
next := nil
end
end; (* get_rec *)

procedure get_instruct_info(var head_inst_list : inst_list);

(* This procedure builds the linked list of instructor
information. The instructor file is assumed to be sorted
according to department, course, and section number. *)
var
temp,rec : inst_list;
eof_flag : boolean;

begin (* get_instruct_info *)
(* sorted instructor information file *)
reset (instr_file,'inst.srt');
if not eof (instr_file) then
begin
get_rec (head_inst_list, instr_file);
temp := head_inst_list;
(* get all instructor information *)
while not eof (instr_file) do
  begin
    get_rec (rec, instr_file);
temp^.next := rec;
temp := temp^.next
  end;
eof_flag := false
end
else
  eof_flag := true
end; (* get_instruct_info *)

procedure dump_inst (var head_inst : inst_list);
{ This procedure will write the updated instructor list
  back to the instructor file INST.SRT. The file
  has been updated with the print flag.
}

var
  temp_i_ptr : inst_list;
i : integer;

begin { dump_inst }
  { rewrite the instructor file }
rewrite (out_inst, 'inst.srt');
temp_i_ptr := head_inst;
while (temp_i_ptr <> nil) do
  begin
    with temp_i_ptr^ do
      begin
        write (out_inst, dept_code, course_num, sect_num);
        write (out_inst, inst_init, inst_name, dept_abr);
        write (out_inst, print_flag, num_of_quest);
        for i := 1 to max_quest_nums do
          (* write instructor picked questions *)
          write (out_inst, quest_nums[i]);
          writeln (out_inst)
      end;
temp_i_ptr := temp_i_ptr^.next
  end;
end; { dump_inst }

procedure get_commands;
{ This procedure opens the file that will contain all
  of the print commands
}
var
ch : char;

begin
rewrite (prcmd_fil, 'prcmd.cmd');
end; { get_commands }

procedure print_heading (var inst_head : inst_list;  
var out_head : text);

{ This procedure will print the heading that will go on the 
top of the questionnaire form. The routine will then write 
the appropriate number of blank lines so that when the 
questions are printed to the file they will appear in the 
correct place on the questionnaire form. }

var
i : integer;

begin { print_heading }
with inst_head do 
begin
{ print down to position }
writeln (out_head);
writeln (out_head);
writeln (out_head);
write (out_head, ' ', inst_init, ' ', inst_name, ' ');
write (out_head, dept_abr, ' ', dept_code, ' ');
write (out_head, course_num, ' ', sect_num);
writeln (out_head);

{ print blank lines down to question section }
for i := 1 to 10 do 
writeln (out_head)
end
end; { print_heading }

procedure print_quest (var inst_head : inst_list;  
var q_tab : q_tab_type;  
var out_quest : text);

{ This procedure will print the questions that are in the 
instructor's record onto the file that will be printed for 
his questionnaire form. }

}
var
  index,q_num : integer;
  done : boolean;

begin
  done := false;
  index := 1;
  while (not done) and
       (inst_head^.quest_nums[index] <> '0000') do
    begin
      get_num (inst_head^.quest_nums[index],q_num);

      { If the question is not in the table print a blank line }
      if (q_tab[q_num] = nil)
        then
          begin
            writeln (out_quest);
            writeln (out_quest)
          end
        else
          begin
            writeln (out_quest,' ':6,q_tab[q_num]^.quest1_string);
            writeln (out_quest,' ':6,q_tab[q_num]^.quest2_string)
          end;
      index := index + 1;
      if (index = 24)
        then
          begin
            index := 23;
            done := true
          end
    end;
  page (out_quest)
end; { print_questions }

procedure print_cmd (var inst_head : inst_list;
                      var prcmd : text);

{ This procedure will create the print commands file for
  the questionnaire form. The procedure will create multiple
  print requests for instructor's who have more than 300
  questionnaire forms. The maximum number of copies that the
  procedure will allow in any print request is 300 copies. }

var
  copies : integer;
  cp_flag : boolean;
procedure print_line (var inst_head : inst_list;
    number : integer;
    var prcmd : text);

{ This procedure will print the command line to print
out a questionnaire form. The routine receives the
name of the file to be printed,'out_string'. }

begin { print_line }
    write (prcmd,'print ',out_string,'/copies:',number:4);
    write (prcmd,'/noheader /forms:eval /note:"burst&hold"');
    if cp_flag
        then
            writeln(prcmd)
        else
            writeln (prcmd,'/delete')
end; { print_line }

procedure get_num (var num_string : num_type;
    var number : integer);

{ This procedure will convert an array of digits [1..4]
to an integer number. }

var
    i : integer;

begin { get_num }
    number := 0;
    for i := 1 to num_length do
        number := number * 10 + (ord(num_string[i]) - ord('0'))
    end; { get_num }

begin { print_cmd }
    get_num (inst_head^.num_of_quest,copies);
    cp_flag := false;
    while (copies > 0) do
        begin
            if (copies <= 300)
                then
                    begin
                        print_line(inst_head,copies,prcmd);
                        copies := 0
                    end
            else
                begin
                    print_line(inst_head,300,prcmd);
                    copies := copies - 300
                end;
        end;
cp_flag := true
end
end; { print_cmd }

procedure print_info (var inst_head : inst_list;
var q_tab : q_tab_type);

{ This procedure will call the appropriate subroutines
to print both the questionnaire form and the print command
file. The routine will create a maximum of 100 forms
at a time so the print queue will not be overloaded. }

var
temp_i_ptr : inst_list;
i : integer;

begin { print_info }
i := 0;
temp_i_ptr := inst_head;
while (temp_i_ptr <> nil) and (i < 100) do
begin
if (temp_i_ptr^.print_flag = '0')
then
begin
i := i + 1;
get_out_file (out_q);
print_heading (temp_i_ptr ,out_q);
print_quest (temp_i_ptr,q_tab ,out_q);
print_cmd (temp_i_ptr ,prcmd_fil);
temp_i_ptr^.print_flag := '1';
end;
end;
temp_i_ptr := temp_i_ptr^.next
end;
if (i = 100)
then
begin
writeln ('100 questionnaire forms created. Rerun the');
writeln ('program after the files have been printed.');
end
else
writeln (i:4,' questionnaire forms created.');
end; { print_info }

begin { main }
init_table (quest_table);
build_question_list (quest_table);
get_instruct_info(inst_head);
get_commands;
print_info (inst_head, quest_table);
dump_inst (inst_head)
end.
program calculate;

(* Written by Peter Weiler
date last revised: May 1983

This program will compute statistics for the faculty evaluation scantron cards. It will search for the scantron files and generate the statistics for the files that it finds. If a file is not found then it will go on to the next file.

The scantron files will be named 'FILEn.SCN' where n is a number from (0..9). The scantron files will contain the codes for the different positions that were marked on the scantron card. The program will interpret the codes so that it can compute the statistics files.

The output files will be written to a subdirectory 'STS:' A file containing the file number will be used to keep track of the different statistics files created. The statistics files will be named 'INnnnn.dta' where the field 'nnnn' will be a number from '0001' to '9999'. *)

const
  max_quest = 33;
  min_quest = 10;
  head_length = 11;
  min_file_num = 0;
  max_file_num = 9;

type
  { converted questionnaire answers }
  line_type = array [1..max_quest] of integer;

  { A list of all the questionnaire results for an instructor }
  list_type = ^info_type;

  info_type = record
    line : line_type;
    next : list_type
  end;

  result = array [1..max_quest] of real;

  { file identification record (department, course, and section number. }
  head_type = packed array [1..head_length] of char;

  { name of input file }
  file_type_string = packed array [1..10] of char;
outfile_type_string = packed array [1..14] of char;

var
  batch, infile, outfile : text;
  header : head_type;
  head_conv_tab : array [0..31] of char;
  head_list : list_type;
  freq : array [1..max_quest,1..5] of integer;
  means, standard_dev : result;
  tot_ans_line : line_type;
  file_count : integer;
  file_there : boolean;
  file_string : file_type_string;
  out_string : outfile_type_string;
  error_flag : boolean;

procedure init_head_conv_tab;

(* This procedure sets up a table that contains the possible
  values for a header card line. The values were obtained
  from the manual on the scantron card reader.
  *)

begin
  head_conv_tab[0] := ' 0 ';
  head_conv_tab[1] := ' 1 ';
  head_conv_tab[2] := ' 2 ';
  head_conv_tab[3] := ' 3 ';
  head_conv_tab[4] := ' 3 ';
  head_conv_tab[5] := ' 3 ';
  head_conv_tab[6] := ' 3 ';
  head_conv_tab[7] := ' 4 ';
  head_conv_tab[8] := ' 4 ';
  head_conv_tab[9] := ' 5 ';
  head_conv_tab[10] := ' 5 ';
  head_conv_tab[12] := ' 7 ';
  head_conv_tab[13] := ' 8 ';
  head_conv_tab[14] := ' 9 ';
  head_conv_tab[15] := '@'
  head_conv_tab[16] := ' 5 ';
  head_conv_tab[17] := ' 6 ';
  head_conv_tab[18] := ' 7 ';
  head_conv_tab[19] := ' 8 ';
  head_conv_tab[20] := ' 8 ';
  head_conv_tab[21] := ' 9 ';
  head_conv_tab[22] := '@'
  head_conv_tab[23] := '@';
  head_conv_tab[24] := ' 9 ';
  head_conv_tab[25] := '@';
  head_conv_tab[26] := '@';
  head_conv_tab[27] := '@';
  head_conv_tab[28] := '@';
  head_conv_tab[29] := '@';
  head_conv_tab[30] := '@';
  head_conv_tab[31] := '@';
end; (* init head conv table *)

procedure init_array (var l_array : line_type);

(* initialize array to hold answers to questions *)

var
  i : integer;
begin
for i := 1 to max_quest do
l_array[i] := 0
end; (* init array *)

procedure init_results;

(* initialize statistics arrays *)

var
i,j : integer;

begin
for i := 1 to max_quest do
begin
  tot_ans_line[i] := 0;
  means[i] := 0.0;
  standard_dev[i] := 0.0;
  for j := 1 to 5 do
    freq[i,j] := 0
end
end; (* init results *)

procedure get_out_file (var outfile : text);

(* This procedure will create the output files for the statistics, it will also increment the batch number in the batch file so that each call to this program increments this number. *)

var
batch_num : integer;

procedure init_out_string (var out_string :(outfile_type_string));

{ This procedure will initialize the string that will contain the name for the output file. }

begin { init_out_string }
  out_string := "sts:in0000.dta";
end; { init_out_string }

begin (* get_out_file *)
(* set batch number *)
(* make sure batch is initialized to zero *)
reset (batch,"batch.num");
if eof (batch)
  then
    batch_num := 0
else
read (batch,batch_num);
batch_num := batch_num + 1;
rewrite (batch,'batch.num');
writeln (batch,batch_num);
(* create a file name for the output file *)
init_out_string (out_string);

{ set the number of the file }

out_string[7] := chr(batch_num div 1000 + ord('0'));
out_string[8] := chr((batch_num div 100) mod 10 + ord('0'));
out_string[9] := chr((batch_num div 10) mod 10 + ord('0'));
out_string[10] := chr(batch_num mod 10 + ord('0'));
rewrite (outfile,out_string)
end; (* get_out_file *)

procedure get_files (file_count : integer;
var file_there : boolean;
var infile,outfile : text);

(* This procedure will get the scantron file with the number
that was passed to it. If the file was found then a true
value is returned. *)

var
  i : integer;
begin (* get_files *)
  file_string := 'FILE0.SCN ';
  file_string[5] := chr(file_count + ord('0'));
  file_there := false;
  reset (infile,file_string);
  if not eof (infile)
  then
  begin
    get_out_file (outfile);
    file_there := true
  end
end; (* get files *)

procedure get_header (var top : head_type);

(* This program converts then header card into integer numbers
from the scantron code. *)

var
  i : integer;
begin { get_header }
  readln (infile,top);
  for i := 1 to head_length do
if ((ord(top[i]) - ord(′@′)) > 31)
then
top[i] := head_conv_tab[ord(top[i]) - ord(′@′) - 32]
else
top[i] := head_conv_tab[ord(top[i]) - ord(′@′)]
end; (* get_header *)

procedure get_card (var line : line_type);

(* This procedure will convert the scantron code on a card into integer numbers. A card is one student's answers to the questionnaire form. *)

var
i : integer;
header : head_type;
card : array [1..max_quest] of char;

begin { get_card }

{ Transpose input file since cards are read backward }
for i := max_quest downto 1 do
read (infile,card[i]);
readln (infile);

{ The values for conversion are the reverse of those stated in the scantron manual because of the way they are read in. }
for i := 1 to max_quest do
  case card[i] of
  ′@′ :
    line[i] := 0;
  ′A′ :
    line[i] := 5;
  ′B′ :
    line[i] := 4;
  ′D′ :
    line[i] := 3;
  ′H′ :
    line[i] := 2;
  ′P′ :
    line[i] := 1;
  others :
    line[i] := -1
  end
end; (* get_card *)

procedure get_info;

(* This procedure will get all of the answer cards that are in a given file and put them into a linked list. *)
var
  quest_number : integer;
  list : list_type;

begin { get_info }
  init_head_conv_tab;
  head_list := nil;
  error_flag := false;
  get_header (header);

  { if the file contains only a header print an error }

  if eof (infile)
    then
      begin
        error_flag := true;
        writeln;
        writeln ('File ',file_string,' contains no data, rerun')
      end
    else
      while not eof(infile) do
        begin
          new (list);
          init_array ( list^.line );
          get_card (list^.line);
          list^.next := head_list;
          head_list := list;
        end
      end; (* get_info *)

procedure compute_stats;

  (* This procedure computes the frequency, mean, and standard
     deviation for each of the questions. All zero values are
     discarded. *)

var
  temp_list : list_type;
  quest_number,sum,sumsqr : integer;

begin { compute_stats }
  init_results;

  { compute stats for all questions }

  for quest_number := 1 to max_quest do
    begin
      temp_list := head_list;
      sum := 0;
      sumsqr := 0;
{ compute stats for a single question }

while temp_list <> nil do
begin

{ if the answer is valid compute stats }

if (temp_list^.line[quest_number] > 0) and
(temp_list^.line[quest_number] < 6)
then
begin
  freq[quest_number,temp_list^.line[quest_number]] :=
  freq[quest_number,temp_list^.line[quest_number]] + 1;
  sum := sum + temp_list^.line[quest_number];
  sumsqr := sumsqr + sqr(temp_list^.line[quest_number]);
  tot_ans_line[quest_number] := tot_ans_line[quest_number]+1
end;
  temp_list := temp_list^.next
end; (* while *)
if (sum <> 0.0) and (sumsqr <> 0.0) and
(tot_ans_line[quest_number] <> 0) and
(tot_ans_line[quest_number] <> 1)
then
begin
  means[quest_number] := sum / tot_ans_line[quest_number];
  standard_dev[quest_number] := sqrt((sumsqr - sqr(sum)/
                                      (tot_ans_line[quest_number] - 1))
                              / (tot_ans_line[quest_number] - 1))
end
end; (* compute_stats *).

procedure print_results;

(* This procedure will print the statistics.out to the
  appropriate file. The file number is changed for each given
  run of print results. *)

var
  quest_number,num_quest,i : integer;

begin
(* determine the number of questions in the file *)
  quest_number := max_quest;
  num_quest := min_quest;
  writeln (outfile,header);

  { compute number of results to print out. }
  while (quest_number > min_quest) and (num_quest = min_quest) do
if tot_ans_line[quest_number] <> 0
then
num_quest := quest_number
else
quest_number := quest_number - 1;

{ write the results out }

for quest_number := 1 to num_quest do
begin
for i := 1 to 5 do
write (outfile,freq[quest_number,i]:4);
write (outfile,means[quest_number]:10:5);
write (outfile,' ':5,standard_dev[quest_number]:10:3);
writeln (outfile)
end (* for *)
end; (* print_results *)

begin (* main *)

{ For all possible scantron files generate statistics files for them. }

for file_count := min_file_num to max_file_num do
begin
get_files (file_count,file_there,infile,outfile);
if file_there
then
begin
get_info;
writeln;
write ('Input file is ',file_string,' output file is ');
writeln (out_string);
if not error_flag
then
begin
compute_stats;
print_results
end
end
end
end.
program report;

(* Program report generator
Written by Peter Weiler
for: faculty evaluation
University of Montana

This program will produce the reports to the answers given by
the students to the faculty evaluation questions. The report
consists of the questions that the instructor picked and the
results to them. *)

(* The following files are needed to run this program:

The report generator will read in the following files:

inst.srt : This file contains all of the instructor information.
The file has to be sorted according to the department
number, the course number, and the section number
before it can be used. The format of the files is as follows:

dept_code : field (1 - 4)
course_num : field (5 - 7)
sect_num : field (8 - 11)
inst_init : field (12)
inst_name : field (13 - 26)
dept_abr : field (27 - 31)
print_flag : field (32 - 36)
um_of_quest : field (37 - 39)
quest_nums : field (40 - 131)

The information will be put into a tree with the
following pointer fields:

department_ptr : will point to the next department
class_ptr : will point to the next class of that
particular department.
sect_ptr : will point to the next section of the class
Each of the sections are ordered according to the
teachers name. This will be used when the files are
being run separately, (which will be the case were
two people teach the same class.

The routine reads in the questions for the questionnaires
from a file "QUEST.DTA". This file is set up in the following
format:

key : field(1 - 4) the question number
string1 : field(5 - 47) first half of the question
string2 ; field(47 - 90) second half of the question
The question data base will be read into a hash table so that the questions can be accessed rapidly.

INnnnn.dta: This file is the statistics about each question that the instructor asked. The first record in this file is a record which tells the department number, class number and section number of the data. The statistics file will contain have seven real numbers for the statistics to each question. The first five real numbers are the frequency of each answer to the question, the sixth number is the mean and the last number is the standard deviation of the answers to the question.

The following output files will be generated:

OTnnnn.dta: The report for the instructor and the report for the Faculty Evaluation Committee.

ERROR.FIL: this file will contain a list of the files where there was more than one instructor teaching the class.

The error file and the report files will be printed in a subdirectory with the logical name 'RPT:'.

*)

const
star = ' *';
under = '_';
key_length = 4;
num_length = 3;
max_quest_nums = 23;
name_length = 14;
d_abr_length = 5;
string_length = 43;
max_tot_quest = 9999;
max_global = 10;

type
key_type = packed array[1..key_length] of char;
q_num_type = key_type;
um_type = packed array[1..num_length] of char;
string_type = packed array[1..string_length] of char;
questions = array[1..max_quest_nums] of key_type;
name_type = packed array[1..name_length] of char;
d_abr_type = packed array [1..d_abr_length] of char;

res_type = (ex_poor,agree);

inst_list = ^inst_rec;

(* The instructor record is read in from the file inst.srt *)
inst_rec = record
  dept_code : packed array [1..4] of char;
course_num : num_type;
sect_num : num_type;
inst_init : char;
inst_name : name_type;
department : d_abr_type;
print_flag : char;
num_of_quest : num_type;
quest_nums : questions;
deptr_ptr : inst_list;
course_ptr : inst_list;
sect_ptr : inst_list
end;

quest_list = ^quest_type;

(* The question type is read in in half fields so that the different parts can be listed on different lines. *)
quest_type = record
  key : key_type;
quest1_string : string_type;
quest2_string : string_type
end;

{ the question table type }
q_tab_type = array [1..max_tot_quest] of quest_list;

(* The header card is used to find the instructor's information in the list of instructor records. This is the first card in the stats file. *)
header_type = record
  dept_code : packed array [1..4] of char;
  course_num : packed array [1..3] of char;
  sect_num : packed array [1..3] of char
end;

(* The stats record is the structure generated by the stats program. This is the structure of the information that follows the header card. *)
stats_type = record
  freq1,freq2,freq3,freq4,freq5 : integer;
  mean,stand_dev : real
end;

global_stats = array [1..max_global] of stats_type;

var
  err_file : text;
  stats : text;
  quest_file : text;
  instrt : text;
  outfile : text;
  head_inst_list : inst_list;
  quest_table : q_tab_type; (* question table *)
  quest : char;

procedure get_num(var key : key_type; var num : integer);

  { This procedure will convert an array [1..4] of digits
    into an integer number.
  }

var
  i : integer;

begin { get_num }
  num := 0;
  for i := 1 to key_length do
    num := num * 10 + (ord(key[i]) - ord(\'0\'))
  end; { get_num }

procedure get_number (var num : integer);

  { This procedure will read in digits from the terminal
    the routine will then convert the digits into an integer
    number. The routine checks to see that only valid characters
    are enter. If the user types a character other than (0..9)
    the routine will have the user reenter the number.
  }

var
  correct : boolean;
  ch : char;

begin { get_number }
  repeat { until correct }
    correct := true;
    num := 0;
repeat { until wrong or eoln }
ch := input'';
get (input);
if (ch >= '0') and (ch <= '9')
then
num := num * 10 + (ord(ch) - ord('0'))
else
begin
  correct := false;
  readln;
  write ('invalid number reenter: ')
end
until (eoln or not correct);
if correct
then
readln
until correct
end; { get_number }

procedure get_rec (var rec : inst_list;var ins : text);

(* This procedure will read in the information on one instructor. *)
var
i : integer;

begin (* get_rec *)
new (rec);
with rec'' do
begin
  read (ins,dept_code,course_num,sect_num);
  read (ins,inst_init,inst__name,dept_abr);
  read (ins,print_flag,num_of_quest);
  for i := 1 to max_quest_nums do
    (* get instructor picked questions *)
    read (ins,quest_nums[i]);
    readln (ins);
    dept_ptr := nil;
    course_ptr := nil;
    sect_ptr := nil
  end
end; (* get_rec *)

procedure get_instruct_info (var head_inst_list : inst_list);

(* This procedure builds the linked list of instructor information. The list is in the order of a three branched tree. In the top linked list is the list of department. Pointing off of each department is a linked list of each course in that department. Pointing off of each course is a list of sections for that course. *)
var
  temp, rec, dept_temp, course_temp : inst_list;

begin (* get_instruct_info *)
  (* sorted instructor information file *)
  reset (instrt,'inst.srt');
  get_rec (head_inst_list,instrt);
  temp := head_inst_list;
  get_rec (rec,instrt);

  while not eof (instrt) do (* get all instructor information *)
    begin
      dept_temp := temp; (* head of a department list *)

      while not eof (instrt) and
        (temp^.dept_code = rec^.dept_code) do
        begin
          course_temp := temp; (* head of a course list *)

          while not eof (instrt) and
            (temp^.course_num = rec^.course_num)
            and (temp^.dept_code = rec^.dept_code) do
            begin (* list of course sections *)
              temp^.sect_ptr := rec;
              temp := rec;
              get_rec (rec,instrt)
            end;
          if not eof (instrt) and (temp^.dept_code = rec^.dept_code)
            then
            begin (* list of courses *)
              course_temp^.course_ptr := rec;
              temp := rec;
              get_rec (rec,instrt)
            end;
          end; (* course while *)
        if not eof (instrt)
          then
          begin (* list of departments *)
            dept_temp^.dept_ptr := rec;
            temp := rec;
            get_rec (rec,instrt)
          end
        end; (* department while *)
    end; (* get_instruct_info *)
end;

procedure build_question_list(var q_tab : q_tab_type);

{ This procedure will create the table of questions that will be used in the reports. The routine will set up the
table to be indexed by the question's number.
}

var
q_rec : quest_list;
index : integer;

procedure init_table (var q_tab : q_tab_type);

{ This procedure will initialize the pointers in the
question table to be nil.
}

var
i : integer;

begin { init_tab }
  for i := 1 to max_tot_quest do
    q_tab[i] := nil
  end; { init_tab }

procedure get_quest (var q_rec : quest_list;var qfile : text);

(* This procedure will read in one question from the question
file. *)

begin { get_quest }
  with q_recA do
    begin
      read (quest_file,key);
      readln (quest_file,quest1_string,quest2_string)
    end
  end; (* get_quest *)

begin (* build question list *)
(* get all of the questions in the data pool *)
reset (quest_file,'quest.dta');
init_table(q_tab);
while not eof (quest_file) do
  begin
    new (q_rec);
    get_quest (q_rec,quest_file);
    get_num (q_rec^.key,index);
    q_tab[index] := q_rec
  end
end; (* build_data_base *)

procedure get_file (file_number : integer;var temp_file : text);

(* This procedure will get an input stats file. The number
that is passed to it is the number of the file. *)
var
in_string : packed array [1..14] of char;

begin { get_file }
  in_string := 'sts:IN0000.DTA';
in_string[7] := chr(file_number div 1000 + ord('0'));
in_string[8] := chr((file_number div 100) mod 10 + ord('0'));
in_string[9] := chr((file_number div 10) mod 10 + ord('0'));
in_string[10] := chr(file_number mod 10 + ord('0'));
reset (temp_file,in_string)
end; (* get file *)

procedure get_header (var head : header_type; var stat:text);

(* This procedure will read in the header card from the stats file. It fills in the extra characters needed to match the fields with those in the instructor information records. *)

var
  chl, ch2, ch3 : char;
begi { get_header }
  with head do
    begin
      read (stats, chl, ch2, ch3);
      dept_code[1] := '0';
      dept_code[3] := ch2;
      dept_code[4] := ch3;
      read (stat, course_num);
      readln (stat, sect_num)
    end
  end; (* get header *)

procedure find_rec (var temp : inst_list; var header : header_type;
  var err_file : text; file_num : integer);

(* This procedure will find an instructor's record in the list of records. If the record is not found then an error message is printed to the error file and a nil value is returned in the pointer to the record. *)

begin { find_rec }
  temp := head_inst_list;

  { find the list with the same department number as the department number in the header record. }
  while (temp^.dept_code <> header.dept_code) and (temp^.dept_ptr <> nil) do
temp := temp^.dept_ptr;
if (temp^.dept_code <> header.dept_code)
then
  begin (* if error return a nil *)
    temp := nil;
    writeln (err_file);
    write (err_file,'Department name for input file: ');
    writeln (err_file,file_num:4,' not found')
  end
else
  begin

    { Find the course list that has the same course number as the course number in the header record. }

    while (temp^.course_num <> header.course_num) and
    (temp^.course_ptr <> nil) do
      temp := temp^.course_ptr;
    if (temp^.course_num <> header.course_num)
      then
        begin (* if error return nil *)
          temp := nil;
          writeln (err_file);
          write (err_file,'Course number for input file: ');
          writeln (err_file,file_num:4,' not found')
        end
      end else

      begin

        { Find the record in the section linked list that has the same section number as the number in the header record. This record will be unique unless more than one instructor is teaching the class. }

        while (temp^.sect_num <> header.sect_num) and
        (temp^.sect_ptr <> nil) do
          temp := temp^.sect_ptr;
        if (temp^.sect_num <> header.sect_num)
          then
            begin (* if error return nil *)
              temp := nil;
              writeln (err_file);
              write (err_file,'Section number for input file: ');
              writeln (err_file,file_num:4,' not found')
            end
          end

      end; (* find rec *)

procedure get_out_file (file_number : integer;
                      var temp_file : text);
(*) This procedure will create an output file. The file number will be the same as the number for the input stats file *)

var
  out_string : packed array [1..14] of char;

begin { get_out_file }
  out_string := 'rpt:OT0000.DTA';
  out_string[7] := chr(file_number div 1000 + ord('O'));
  out_string[8] := chr((file_number div 100) mod 10 + ord('O'));
  out_string[9] := chr((file_number div 10) mod 10 + ord('O'));
  out_string[10] := chr(file_number mod 10 + ord('O'));
  rewrite (temp_file,out_string)
end; (* get out file *)

procedure get_global_stats ( var globals : global_stats;
  var stats : text);

  (* This procedure will read in the statistics to the global questions from the statistics file *)

var
  i : integer;

begin { get_global_stats }
  for i := 1 to max_globals do
    with globals[i] do
      begin
        read (stats,freq1,freq2,freq3,freq4,freq5);
        readln (stats,mean,stand_dev)
      end
  end; (* get global stats *)

procedure line (length : integer; ch : char; var outfile : text);

  (* The procedure will write a line for the given character,ch, and length, length, and write the line to the file ,outfile. *)

var
  i : integer;

begin { line }
  for i := 1 to length do
    write (outfile,ch)
end; (* line *)

procedure print_mean (mean : real; var outfile : text);

  (* This procedure will write an interpretation of the value for the mean. The interpretation is written out
so the reports may be interpreted easier.
*)

begin { print_mean }
if (mean <= 1.4)
then
write (outfile,'mean is high 
')
else
if (mean > 1.4) and (mean <= 2.2)
then
write (outfile,'mean is high ave 
')
else
if (mean > 2.2) and (mean <= 3.8)
then
write (outfile,'mean is average 
')
else
if (mean > 3.8) and (mean < 4.6)
then
write (outfile,'mean is low ave 
')
else
write (outfile,'mean is low 
')
end; (* print mean *)

procedure print_con (stand : real; var outfile : text);
(* This procedure will write an interpretation of the standard
 deviation. The consensus means how many students agreed on a
 given answer to a question. *)

begin { print_con }
if (stand <= 0.57)
then
write (outfile,'consensus is high 
')
else
if (stand > 0.57) and (stand <= 1.1)
then
write (outfile,'consensus is average 
')
else
write (outfile,'consensus is low 
')
end; (* print con *)

procedure print_quest (quest_num : integer;
    var quest : quest_type;
    response : res_type;
    var q_stats : stats_type;
    var outfile : text);
(* This procedure will print the standard evaluation questions
 and the instructor picked questions. The question to be printed
 is passed as a parameter along with the question number and
 the statistic for that question. The procedure also receives

a flag telling it which response to print out for the question. *)

```pascal
type
  res_string = packed array [1..11] of char;

var
  denom : real;
  highl_response, high2_response : res_string;
  lowl_response, low2_response : res_string;

begin { print_quest }

{ Set responses to the questions }

if (response = ex_poor) then
begin
  highl_response := 'Excellent ';
  high2_response := '  ';
  lowl_response := 'Very poor ';
  low2_response := '  ';
end
else
begin
  highl_response := ' strongly ';
  high2_response := ' agree ';
  lowl_response := ' strongly ';
  low2_response := ' disagree ';
end;

line (72, star, outfile);
writeln (outfile);
with quest do
begin
{ print the questions and the means }

write (outfile, ' ', quest_num:4, ' . ', quest1_string, '*' );
writeln (outfile, ' mean stah dev *');
write (outfile, ' ':7, quest2_string,'* ' , q_stats.mean:5:2);
write (outfile, ' ' , q_stats, stand_dev: 5:2,' *');
write (outfile, chr(13),' ':5);
line (67, under, outfile);
writeln (outfile);
with q_stat8 do
begin
{ print the frequency of responses }

write (outfile, ' ' :7, highl_response,freq1:4,freq2:4,freq3:4);
write (outfile,freq4:4,freq5:4, ' ', lowl_response, '*' );
```
print_mean (mean, outfile);
writeln (outfile, '*');
write (outfile, ':7, high2_response);
denom := (freq1 + freq2 + freq3 + freq4 + freq5) / 100;
if (denom <= 0.0)
then
denom := 0.01;

{ print percentages of for each response }
write (outfile, trunc(freq1/denom):3, '%');
write (outfile, trunc(freq2/denom):3, '%');
write (outfile, trunc(freq3/denom):3, '%', trunc(freq4/denom):3);
write (outfile, '%', trunc(freq5/denom):3, '%');
write (outfile, low2_response, '*');
print_con (stand_dev, outfile);
writeln (outfile, '*')
end (* with stats *)
end (* with question *)
end; (* print_quest *)

procedure print_student (var ptr : inst_list;
var globe_stats : global_stats;
var outfile : text);

(* This procedure will print out the general questions on the
students. A pointer to the instructor's information is passed
 to print out the heading, the answers, and the statistics for
the report. The routine has all of the global questions
hard coded into itself. *)

var
total : real;
begin (* print_student *)

{ Print the heading for the report }

writeln (outfile);
writeln (outfile, ':5,' 'Instructor     Dept     Course Sec');
with ptr do
begin
  write (outfile, ':5, inst_init, inst_name, ', dept_code);
  writeln (outfile, ', dept_abr, ', course_num, ', sect_num)
end;
writeln (outfile);
write (outfile, ':8,' 'The evaluation report consists of two');
writeln (outfile, ' parts, the first seven');
write (outfile, ':5,' 'questions give information about the');
writeln (outfile, ' students. The remaining');
write (outfile, ':5,' 'questions evaluate the instructor and');
writeln (outfile, ' course. In the');
write (outfile, ':5,' 'evaluation questions a high rating');
writeln (outfile, ' will have a low number for');
write (outfile, ':5,' 'the mean (ex. high=1 low=5). The');
writeln (outfile, ' consensus tells how much');
write (outfile, ':5,' 'agreement the students had about the');
writeln (outfile, ' question. For each');
write (outfile, ':5, ' 'question there is a frequency');
writeln (outfile, ' distribution of the answers and');
write (outfile, ':5, ' 'the percentages of students marking each');
writeln (outfile, ' answer.\');

(***** Question one *****)
line (72, star, outfile);
writeln (outfile);
write (outfile, ' 1. Pre-course opinion toward instructor');
write (outfile, chr(13), ':5);
line (67, under, outfile);
writeln (outfile);
writeln (outfile, ' positive no opinion negative');
with globe_stats[1] do
begin
writeln (outfile, ':7, freq1:4, ':7, freq3:4, ':7, freq5:4);
total := freq1 + freq3 + freq5;
if (total <= 0.0)
then
  total := 1.0;
write (outfile, ':7, trunc(freq1/total*100):3,\%');
write (outfile, trunc(freq3/total*100):3,\%);
writeln (outfile, trunc(freq5/total*100):4,\%)
end;

(***** Question two *****)
line (72, star, outfile);
writeln (outfile);
write (outfile, ' 2. Pre-course opinion toward course');
write (outfile, chr(13), ':5);
line (67, under, outfile);
writeln (outfile);
writeln (outfile, ' positive no opinion negative');
with globe_stats[2] do
begin
writeln (outfile, ':7, freq1:4, ':7, freq3:4, ':7, freq5:4);
total := freq1 + freq3 + freq5;
if (total <= 0.0)
then
  total := 1.0;
write (outfile, ':7, trunc(freq1/total*100):3,\%');
write (outfile, trunc(freq3/total*100):3,\%);
writeln (outfile, trunc(freq5/total*100):4,\%)
end;
end;

(***** Question three *****)

line (72,star,outfile);
writeln (outfile);
write (outfile, ' 3. Course in:');
write (outfile, chr(13), ':5);
line (67, under, outfile);
writeln (outfile);
write (outfile, ' ':5, 'major minor other');
with globe_stats[3] do
begin
    writeln (outfile, ':6, freq1:4, ':5, freq3:4, ':5, freq5:4);
    total := freq1 + freq3 + freq5;
    if (total <= 0.0) then
        total := 1.0;
    write (outfile, ':6, trunc(freq1/total*100):3,'\% ',
        ':13);
    write (outfile, ':5, trunc(freq3/total*100):3,'\% ',
        ':13);
    writeln (outfile, trunc(freq5/total*100):3,'\% ')
end;

(***** Question four *****)

line (72,star,outfile);
writeln (outfile);
write (outfile, ' 4. This course was:');
write (outfile, chr(13), ':5);
line (67, under, outfile);
writeln (outfile);
write (outfile, ' ':5, 'Required', ':17, 'Elective');
with globe_stats[4] do
begin
    writeln (outfile, ':7, freq2:4, ':17, ':13, freq4:4);
    total := freq2 + freq4;
    if (total <= 0.0) then
        total := 1.0;
    write (outfile, ':7, trunc(freq2/total*100):3,'\% ',
        ':13);
    write (outfile, ':17);
    writeln (outfile, trunc(freq4/total*100):3,'\% ')
end;

(***** Question five *****)

line (72,star,outfile);
writeln (outfile);
write (outfile, ' 5. Sex: Male Female');
write (outfile, chr(13), ':5);
line (67,under,outfile);
writeln (outfile);
with globe_stats[5] do begin
  writeln (outfile, ':16,freq2:4, ':9,freq4:4);
  total := freq2 + freq4;
  if (total <= 0.0) then total := 1.0;
  write (outfile, ':16,trunc(freq2/total*100):3,%'(':9);
  writeln (outfile,trunc(freq4/total*100):3,'%')
end;

(***** Question six *****)

line (72,star,outfile);
writeln (outfile);
write (outfile,' ':4,'6. ');write (outfile,'Class: fresh soph junior senior other');write (outfile,chr(13),' ':5);
line (67,under,outfile);
writeln (outfile);
with globe_stats[6] do begin
  write (outfile,' ':15,freql:4,' ',freq2:4,' ',freq3:4);
  writeln (outfile,' ',freq4:4,' ',freq5:4);
  total := freql + freq2 + freq3 + freq4 + freq5;
  if (total <= 0.0) then total := 1.0;
  write (outfile,' ':15,trunc(freql/total*100):3,'% ');
  write (outfile,trunc(freq2/total*100):3,'% ');
  write (outfile,trunc(freq3/total*100):3,'% ');
  write (outfile,trunc(freq4/total*100):3,'% ');
  writeln (outfile,trunc(freq5/total*100):3,'%')
end;

(***** Question seven *****)

line (72,star,outfile);
writeln (outfile);
write (outfile,' 7. ');write (outfile,'Expected grade :  A B C D F');write (outfile,chr(13),' ':5);
line (67,under,outfile);
writeln (outfile);
with globe_stats[7] do begin
  write (outfile,' ':23,freql:4,' ',freq2:4,' ',freq3:4,' ');
  writeln (outfile,freq4:4,' ',freq5:4);
  total := freql + freq2 + freq3 + freq4 + freq5;
  if (total <= 0.0)
then
  total := 1.0;
  write (outfile,':23,trunc(freq1/total*100):3,'%')
  write (outfile,trunc(freq2/total*100):3,'%')
  write (outfile,trunc(freq3/total*100):3,'%')
  write (outfile,trunc(freq4/total*100):3,'%')
  writeln (outfile,trunc(freq5/total*100):3,'%')
end;
end; (* print student *)

procedure print_eval (var globe_stats : global_stats;
  var outfile : text);

  (* This procedure will create the question records for the
     evaluation questions and call the procedure to print them
     out. These question are questions 8 - 10 on the questionnaire
     form. *)

var
  temp_quest : quest_type;
  response : res_type;

begin { prin_eval }
  (* begin evaluation questions *)
  response := ex_poor;

  (***** Question eight *****)
  with temp_quest do
    begin
      quest1_string := 'Rate the course content';
      quest2_string :=
    end;
  print_quest(8,temp_quest,response,globe_stats[8],outfile);

  (***** Question nine *****)
  with temp_quest do
    begin
      quest1_string := 'Rate the instructor';
      quest2_string :=
    end;
  print_quest(9,temp_quest,response,globe_stats[9],outfile);

  (***** Question ten *****)
  with temp_quest do
    begin
      quest1_string := 'Rate the course in general';
      quest2_string :=
    end;
print_quest(10,temp_quest,response,globe_stats[10],outfile);
end; (* print eval *)

procedure print_global (var ptr : inst_list;
   var globe_stats : global_stats;
   var outfile : text);
(* This procedure will call the routines to print out the results to the global questions. *)
begin { print_global }
   print_student (ptr,globe_stats,outfile);
   print_eval (globe_stats,outfile)
end; (* print global *)

procedure print_instructor (var ptr : inst_list;
   var q_tab : q_tab_type;
   file_num : integer;
   var stats,outfile : text);
(* This procedure has a pointer to the instructor's information as input, and has the stats and output files passed to it. This procedure will go through the list of the instructor's picked questions and find them in the question list. The procedure print_quest is then called to print out the question *)

var
   i,q_num : integer;
   done : boolean;
   q_stats : stats_type;
   response : res_type;
begin { print_instructor }
i := 1;
done := false;
response := agree;
while not done and (ptr^.quest_nums[i] <> '0000') do
   begin
      if eof (stats)
      then begin
         writeln (err_file);
         writeln (err_file,'Number of questions do not match ');
         write (err_file,'the number of answers in file number: ');
         writeln (err_file,file_num:4);
         done := true
      end { Note a report is produced even if the questions don't match. The report will halt where the statistics ran out. }

end
else
begin
get_num (ptr^quest_nums[i], q_num);
(* get statistics *)
with q_stats do
begin
read (stats, freq1, freq2, freq3, freq4, freq5);
readln (stats, mean, stand_dev)
end;
print_quest (i+max_global,q_tab[i],response,
q_stats,outfile);
i := i + 1;
if i = 13
then
page (outfile);
if i = 24
then
begin
i := 23;
done := true
end
end
end; (* print instructor *)

procedure print_report ( var pointer : inst_list; var stats:text;
file_number : integer);

(* This procedure will print out the reports that are to go
to both the Faculty Evaluation Committee and the instructor. *)

var
  globe_stats : global_stats;

begin
  { Print the report for the faculty evaluation committee. }
  get_out_file (file_number,outfile);
  get_global_stats (globe_stats,stats);
  write (outfile,"":23,"Faculty Evaluation Report");
  write (outfile,chr(13),"":23);
  line (25,under,outfile);
  writeln (outfile);
  print_global (pointer,globe_stats,outfile);
  { print the report for the instructor. }
  page (outfile);

write (outfile,' ',26,'Instructor's Report');
write (outfile,chr(13),' ',26);
line (19,under,outfile);
writeln (outfile);
print_global (pointer,globe_stats,outfile);
page (outfile);
print_instructor (pointer,quest_table,file_number,stats,outfile)
end; (* print report *)

procedure single_entry;

(* This procedure will produce a report for one instructor.
The correct department, course, and section is first found and
then a check to see which instructor taught the class is then
done. This routine is used for classes where more than one
instructor taught the class. Error messages are sent to the
terminal when this procedure is used instead of being sent
to a file. The routine will print an error message if it
can not find the instructor's record. *)

var
  name : packed array[1..14] of char;
  temp : inst_list;
  header : header_type;
  file_number,i : integer;
  question,ch : char;
begin { single_entry }
  rewrite (err_file,'tty:');
  repeat (* until you want to quit *)
    name := ' '; writeln;
    writeln ('Single file entry routine:');
    write ('Enter statistics file number: ');
    (* This will be the number for the stats file *)
    get_number (file_number);
    get_file (file_number,stats);
    get_header (header,stats);
    writeln;
    write ('Enter the Instructor's last name: '); ch := input; get (input);
    while not eoln and not (ch in ['a'..'z','A'..'Z']) do begin
      ch := input; get (input)
end;
name[1] := ch;
i := 2;
while not eoln and (i <= 14) do 
begin
    name[i] := input'';
    get (input);
    i := i + 1
end; (* get instructor name *)
readln;

find_rec (temp, header, err_file, file_number);
if (temp <> nil)
then
    begin

        { After finding the correct department, course, and section number find the record with the correct instructor's name. }

        while (temp^.inst_name <> name) and
            (temp^.sect_ptr <> nil) do
            temp := temp^.sect_ptr;
        if (temp^.inst_name = name) and
            (temp^.dept_code = header.dept_code) and
            (temp^.course_num = header.course_num) and
            (temp^.sect_num = header.sect_num)
            then
                print_report (temp, stats, file_number)
            else
                writeln ('Instructor's information not in data base.' )
    end;
    writeln;
    write ('If you have another entry type ,c, to continue. ');
    readln (question);
    until (question <> 'c') and (question <> 'C')
end; (* single entry *)

procedure mult_entry;

(* This procedure will generate the report files for any number of files. The number of the low and high files are given. These numbers are used to get the input file name and create the output file. All error messages are sent to the file error.fil *)

var
    header : header_type;
    temp : inst_list;
    file_num, low_file, high_file : integer;

begin { mult_entry }
rewrite (err_file,'rpt:ERROR.FIL');
writeln;
write ('Enter the lowest statistics file number: ');
get_number (low_file);
writeln;
write ('Enter the highest statistics file number: ');
get_number (high_file);

{ Generate reports for all of the files in the range of low_file : high_file. }
for file_num := low_file to high_file do
begin
get_file (file_num,stats);
if eof (stats)
then
begin
    writeln (err_file);
    writeln (err_file,'File IN',file_num:4,' .dta not found')
end
else
begin
    get_header (header,stats);
    find_rec (temp,header,err_file,file_num);
    if (temp <> nil)
then
    if (temp^.sect_ptr <> nil)
then
        if (temp^.dept_code = temp^.sect_ptr^.dept_code) and
            (temp^.course_num = temp^.sect_ptr^.course_num) and
            (temp^.sect_num = temp^.sect_ptr^.sect_num)
then
            begin
                writeln (err_file);
                write (err_file, 'Duplicate entry in file number: ');
                writeln (err_file,file_num:4);
                writeln (err_file,'run the file in single mode.')
            end
        else
            print_report(temp,stats,file_num)
    else
        print_report(temp,stats,file_num)
end
end; (* mult entry *)

begin (* main *)
get_instruct_info(head_inst_list);
build_question_list(quest_table);
writeln ("ICES Report generating program. This program will"); writeln ("generate reports for both the instructor and the"); writeln ("Faculty Evaluation Committee. The instructor's"); writeln ("statistics file is used to get information from the"); writeln ("data base for the reports."); writeln;

write ("Do you want single or multiple file entry? (S or M) ");
quest := input
readln;
while (quest <> 's') and (quest <> 'S') and
(quest <> 'm') and (quest <> 'M') do
begin
  write ("Type S of M ");
  quest := input
  readln
end;
if (quest = 's') or (quest = 'S')
then
  single_entry
else
  mult_entry
end.
program quest;
{
    Written by: Peter Weiler
    Date last revised: May, 1983
    For: Faculty Evaluation System

    This program maintains the questionnaire file and the standard
    form file. The program allows the user to add, delete, and
    print the questions. The user can also modify the
    questions that are in the data base in either a single
    or multiple question mode.
    The routine allows the user to add, delete, and print
    the standard questionnaire forms.

    The routine reads in the questions for the questionnaires
    from a file "QUEST.DTA". This file is set up in the following
    format:

    key : field(1 - 4) the question number
    string1 : field(5 - 47) first half of the question
    string2 : field(47 - 90) second half of the question

    The question file will be put into a linked list since
    there is not much of a speed requirement for this program.
    All questions that are added to the data base are added
    at the end of the question list.

    The instructor maintenance program also uses a file that contains
    the standard forms. This file is called "STAN.FRM". The format
    for the file is as follows:

    form number : fields(1 - 4)
    quest numbers : fields(5 - 96)

    where the question number field contains 23 question numbers with
    each question number taking up four characters.

}\
const
    string_length = 43;
    key_length = 4;
    max_quest_nums = 23;

type
    key_type = packed array [1..key_length] of char;
    string_type = packed array [1..string_length] of char;
PROGRAMS

{ question linked list }
quest_list = "quest_type;

quest_type = record
  key : key_type;
  quest1_string : string_type;
  quest2_string : string_type;
  next : quest_list
end;

{ form linked list }
form_ptr = "form_type;

form_type = record
  form_num : key_type;
  quest_nums : array [1..max_quest_nums] of key_type;
  next : form_ptr
end;

var
quest_file,quest_outfile,outfile : text;
form_file,forms_output : text;
option : char;
head_list : quest_list;
stan_forms : form_ptr;

procedure line (length : integer; ch : char; var outfile : text);

("The procedure will write a line for the given character, ch,
 and length, length. The line will be written to outfile. *

var
  i : integer;

begin { line } 
for i := 1 to length do 
  write (outfile,ch)
end; (* line *)

procedure build_question_list(var head : quest_list);

{ This procedure will create a linked list of all of the
  questions in the data base. The questions will be in order
  according to the question number. }

var
  temp_quest : quest_list;

procedure get_quest (var quest : quest_list;var qfile : text);
(* This procedure will read in one question from the question file. *)

begin { get_quest }
  with quest do
    begin
      read (quest_file,key);
      readln (quest_file,quest1_string,quest2_string);
      next := nil
    end
  end; (* get quest *)

begin (* build question list *)
  (* get all of the questions in the data pool *)
  reset (quest_file,'quest.dta');
  new (head);
  temp_quest := head;
  get_quest (head,quest_file);
  while not eof (quest_file) do
    begin
      new (temp_quest^.next);
      temp_quest := temp_quest^.next;
      get_quest (temp_quest,quest_file)
    end
  end; (* build_data_base *)

procedure get_forms (var stan_forms : form_ptr);

{ This procedure will read in the standard questionnaire form records into a linked list. Each standard form consists of a form number and 23 question numbers. }

var
  i : integer;
  temp_f_ptr : form_ptr;

begin { get_forms }
  reset (form_file,'stan.frm');
  new (stan_forms);
  temp_f_ptr := stan_forms;
  temp_f_ptr^.form_num := '0000';
  temp_f_ptr^.next := nil;
  while not eof (form_file) do
    begin
      read (form_file,temp_f_ptr^.form_num);
      for i := 1 to max_quest_nums do
        read (form_file,temp_f_ptr^.quest_nums[i]);
      readln (form_file);
      if not eof (form_file) then
begin
    new (temp_f_ptr^.next);
    temp_f_ptr := temp_f_ptr^.next
end
else
    temp_f_ptr^.next := nil
end; { get_forms }

procedure convert (num : integer; var key : key_type);
{ This procedure will convert an integer number into and array [1..4] of digits. }

var
    i : integer;

begin { convert }
    for i := key_length downto 1 do
        begin
            key[i] := chr((num mod 10) + ord('0'));
            num := num div 10
        end
    end; (* convert *)

procedure get_string (var string : string_type);
{ This procedure will read in one of the strings that compose a question. A question consists of two strings of the type read in in this routine. The procedure will ignore all characters after the string length has been exceeded. }

var
    ch : char;
    count : integer;

begin { get_string }
    string := "";
    count := 1;
    ch := input^;
    get (input);
    if not eoln
    then
        string[1] := ch;
    while not eoln and (count < string_length) do
        begin
            count := count + 1;
            ch := input^;
        end
    end;
string[count] := ch;
get (input)
end;
readln
end; { get_string }

procedure mod_question (var q_node : quest_list);

{ This procedure allows the user to modify a question. The procedure is passed a pointer to the question that is to be modified. The routine displays the question and asks the user if it is to be modified. If the user chooses to modify the question he enters in the new question. }

var
  answer : char;

begin { mod_question }
  writeln;
  with q_node do
    begin
      writeln (' ',key,' ',quest1_string);
      writeln (' ':6,quest2_string);
      write ('Do you want to modify this question? (Y or N) ');
      readln (answer);
      if (answer = 'y') or (answer = 'Y')
      then
        begin
          writeln;
          write ('Enter the first half of the question. ');
          writeln ('|((43 char)')
          get_string(quest1_string);
          writeln;
          writeln ('Enter the second half of the question. ');
          writeln ('|((43 char)')
          get_string (quest2_string);
          writeln;
        end
    end
  end
end; { mod_question }

procedure mult_quest (var head_list : quest_list);

{ This procedure allows the user to modify a number of the existing questions in the data base. The user enters the start and end question numbers for the range of questions that he is modifying. All of the old questions will be overwritten. }
var
    q_num, start_num, stop_num : integer;
    stop_key, q_key : key_type;
    temp_q : quest_list;

begin { mult_quest }
    writeln;
    writeln ('Multiple question modification routine');
    write ('Enter starting question number: '); readln (start_num);
    writeln;
    write ('Enter end question number: '); readln (stop_num);
    writeln;
    writeln ('Each question that is entered has to be in two');
    writeln ('parts of 43 characters (or less) each.');
    temp_q := head_list;
    q_num := start_num;
    convert (q_num, q_key);
    convert (stop_num, stop_key);

(* find the first question to be modified *)

    while (temp_q^.next <> nil) and (q_key <> temp_q^.key) do
        temp_q := temp_q^.next;
    while (q_key <= stop_key) and (temp_q^.next <> nil) do
    begin
        if (q_key > temp_q^.key)
            then
                if (temp_q^.next <> nil)
                    then
                        temp_q := temp_q^.next;

        if (q_key < temp_q^.key)
            then
                begin
                    q_num := q_num + 1;
                    convert (q_num, q_key)
                end;
        if (q_key = temp_q^.key)
            then
                begin
                    mod_question (temp_q);
                    q_num := q_num + 1;
                    convert (q_num, q_key)
                end
    end (* while *)
end; { mult_quest }
procedure single_quest (var head_list : quest_list);

{ This procedure allows the user to modify a single question. The user enters the question number of the question that is to be modified. The old question will be overwritten by the new question. }

var
q_num : integer;
q_key : key_type;
temp_q : quest_list;

begin { single_quest }
writeln;
write ('Enter question number: '); readln (q_num);
writeln ('The question that is to be entered has to be'); writeln ('in two parts of 43 characters (or less) each.);
(* find question *)
convert (q_num,q_key);
temp_q := head_list;
{ find the question that is to be modified. }

while (temp_q^next <> nil) and
(q_key <> temp_q^key) do
  temp_q := temp_q^next;
if (q_key = temp_q^key)
  then
    mod_question (temp_q)
  else
    writeln ('Question ',q_key,' not found."
end; (* single_quest *)

procedure delete_quest (var head_list : quest_list);

{ This procedure allows the user to delete a question from the data base. The user enters the number of the question to be deleted. The procedure finds the question and calls a subprocedure to display and delete it. }

var
  temp_q : quest_list;
  q_num : integer;
  q_key : key_type;

procedure delete (var quest : quest_list);
{ This procedure will display the question to be deleted. The procedure gives the user the option of deleting the question of returning to the main routine. }

var
  answer : char;

begin { delete }
  writeln;
  with quest^ do
  begin
    writeln (' ',key,' ',quest1_string);
    writeln (' ':6,quest2_string)
  end;
  writeln;
  write ('Do you wish to delete this question? (Y or N) '); readln (answer);
  if (answer = 'y' ) or (answer = 'Y' )
  then
    quest := quest^.next
  end; { delete }

begin (* delete quest *)
  writeln;
  write ('Enter the number of the question to be deleted: '); readln (q_num);
  temp_q := head_list;
  convert (q_num,q_key);
  (* check if first question *)
  if (q_key = temp_q^.key)
  then
    delete (head_list)
  else
    begin
      { Find the question in the linked list }

      while (temp_q^.next^.next <> nil) and
        (q_key <> temp_q^.next^.key) do
        temp_q := temp_q^.next;
      if (q_key = temp_q^.next^.key)
      then
        delete (temp_q^.next)
      else
        writeln ('Question ',q_key,' not found.' )
    end
  end; (* delete quest *)

procedure get_num(var key : key_type;var num : integer);
{ This routine receives an array [1..4] of digits and converts it into an integer number. }

var
  i : integer;

begin { get_num }
  num := 0;
  for i := 1 to key_length do
    num := num * 10 + (ord(key[i]) - ord('0'))
  end; { get_num }

procedure add_quest (var head_list : quest_list);

  { This procedure allows the user to add a question to the question list. The routine will add the question to the end of the question list. }

var
  temp_q : quest_list;
  q_num : integer;

begin { add_quest }
  writeln;
  writeln ('The question that is to be added has to be in');
  writeln ('two parts of 43 characters (or less) each.');
  writeln;
  (* find the end of the list *)
  temp_q := head_list;
  while (temp_q^.next <> nil) do
    temp_q := temp_q^.next;
  { get last question number from list }
  get_num (temp_q^.key,q_num);
  q_num := q_num + 1;

  { Enter in new question }
  new (temp_q^.next);
  temp_q := temp_q^.next;
  temp_q^.next := nil;
  convert (q_num,temp_q^.key);
  writeln ('Enter the first half of the question. |(43 char)');
  get_string(temp_q^.quest1_string);
  writeln;
  writeln ('Enter the second half of the question. |(43 char)');
get_string (temp_q^.quest2_string);
writeln;
writeln ('Question ' , temp_q^.key, ' created')
end; (* add quest *)

procedure print_quest (var head_list : quest_list);
{
This procedure wrote all of the questions in the data
base to a file `QUEST.DTA`. The routine writes a header at
the top of the page and draws lines inbetween questions.
}

var
  temp_q : quest_list;
  line_count : integer;

begin { print_quest }
  writeln;
  writeln ('Output file is quest.rpt');
  rewrite ( outfile,'quest.rpt' );
  temp_q := head_list;
  line_count := 0;
  while (temp_q <> nil) do
    begin
      { check to see if a new page is needed }
      if (line_count mod 28) = 0 then
        begin
          if (line_count <> 0) then page (outfile);
          writeln (outfile);
          writeln (outfile,' key',' ':30,'Question');
          line(91,'*',outfile);
          writeln (outfile);
        end;
      with temp_q.A do
        begin
          write (outfile,' ',key,' ',questL_string);
          writeln (outfile,quest2_string);
          writeln (outfile)
        end;
      line_count := line_count + 1;
      tenqx_q := temp_q^.next
    end
  end; (* print quest *)

procedure add_stan_forms (var head_list : form_ptr);
This procedure allows the user to add a new standard form to the standard form database. The routine displays the new standard form number and prompts the user for entry of the question numbers for a standard form. The user may type at most 23 questions for a standard form. If the standard form is to contain less than 23 questions the user may type a zero for the last question number and the rest of the standard form question numbers will be filled in with zeros.

```
var
temp_f_ptr : form_ptr;
i, quest_num, form_num, num_of_quest : integer;

procedure init_quest_array (var quest : form_ptr);
{ This procedure initializes the array of standard form numbers to be all zeros.
}
var
i : integer;
begin { init_quest_array }
for i := 1 to max_quest_nums do
quest^.quest_nums[i] := '0000'
end; { init_quest_array }

procedure char_to_dig (var num : integer; var key : key_type);
{ This procedure receives an array [1..4] of digits and converts them into an integer number.
}
var
i : integer;
begin { char_to_dig }
num := 0;
for i := 1 to key_length do
num := num * 10 + (ord(key[i]) - ord('0'))
end; { char_to_dig }

begin { add_stan_forms }
temp_f_ptr := head_list;
{ get last form number }
if (temp_f_ptr^.form_num <> '0000') then
begin
while (temp_f_ptr^.next <> nil) do
  temp_f_ptr := temp_f_ptr^.next;
```
char_to_dig (form_num, temp_f_ptr^.form_num);

{ Get new form number }

form_num := form_num + 1;
new (temp_f_ptr^.next);
temp_f_ptr := temp_f_ptr^.next;
temp_f_ptr^.next := nil;
convert (form_num, temp_f_ptr^.form_num)
end
else
begin { There are no standard forms }
form_num := 1;
convert (form_num, temp_f_ptr^.form_num)
end;
writeln;
writeln ("Enter the question numbers for standard form");
writeln ("number",form_num:5,". If you have fewer than");
writeln ("23 questions type a ,0, as the final question ");
writeln ("number to stop.");
writeln;
um_of_quest := 1;
quest_num := 1;
init_quest_array (temp_f_ptr);

{ Read in the questions for the new standard form }

while (quest_num <> 0) and (num_of_quest <= max_quest_nums) do
begin
write ("Enter the number of question",num_of_quest:3," : ");
readln (quest_num);
convert (quest_num, temp_f_ptr^.quest_nums[num_of_quest]);
writeln;
num_of_quest := num_of_quest + 1
end;
end; { add_stan_forms }

procedure print_stand_forms (var stan_form : form_ptr);

{ This procedure uses the question numbers in the standard
forms to obtain the questions from the question list. The
routine will write out the questions for the standard
questionnaire forms. }

var
  temp_f_ptr : form_ptr;
temp_q : quest_list;
  key : key_type;
i : integer;
begin { print_stand_forms }
  writeln;
  writeln ("Output file is stan.rpt");
  rewrite (forms_output, 'stan.rpt');
  temp_f_ptr := stan_form;
  while (temp_f_ptr <> nil) do
  begin
    { print form heading }
    writeln (forms_output);
    writeln (forms_output);
    write (forms_output, 'Standard form number: ');
    writeln (forms_output, temp_f_ptr.form_num);
    writeln (forms_output);
    write (forms_output, 'This form will contain the general');
    writeln (forms_output, ' questions and the following ');
    writeln (forms_output, 'questions.');
    writeln (forms_output);
    writeln (forms_output);
    writeln (forms_output);
    { print the questions for the given form }
    for i := 1 to max_quest_nums do
    begin
      temp_q := head_list;
      key := temp_f_ptr.QUEST_nums[i];
      if (key <> "0000") then
      begin
        { find the question }
        while (key <> temp_q.key) and
        (temp_q.next <> nil) do
          temp_q := temp_q.next;
        if (temp_q.key <> key) and (temp_q.next = nil) then
          writeln ("Question ",key," not found")
        else
          begin
            writeln (forms_output, i:3, ": ", temp_q.quest1_string);
            write (forms_output, ":5", temp_q.quest2_string);
            write (forms_output, chr(13));
            line (75, ":", forms_output);
            writeln (forms_output)
          end
      end
    end;
    { print a single form }
    page (forms_output);
    temp_f_ptr := temp_f_ptr.next
  end
end; { print_standard_form }
procedure delete_stan_forms(var head_list: form_ptr);

{ This procedure allows the user to delete a standard form. The routine is passed a pointer to the beginning of the standard form list. The user then enters the number of the standard form to be deleted. The user has the option of deleting the standard form or returning to the main routine. }

var
    temp_f_ptr: form_ptr;
    f_num: integer;
    f_key: key_type;

procedure d_form(var form: form_ptr);

{ This procedure will display the question numbers in the standard form to be deleted. The user then has the option of deleting the form or returning to the main routine. }

var
    answer: char;
    i, offset, half_quests: integer;

begin (* d_form *)
    writeln;

    { Compute the first half of question numbers to be displayed. }
    offset := (max_quest_nums div 2) + max_quest_nums mod 2;
    half_quests := max_quest_nums div 2;

    { Display the question numbers in two columns. }
    with form^ do
        begin
            for i := 1 to half_quests do
                begin
                    write (' ', quest_nums[i]);
                    writeln (' ', i + offset:2, ' ', quest_nums[i + offset]);
                end;
                if odd(max_quest_nums) then
                    writeln (' ', half_quests + 1:2, ' ', quest_nums[half_quests + 1]);
        end;
    writeln;
    write ('Do you wish to delete this form? (Y or N) ');
    readln (answer);
if (answer = 'y') or (answer = 'Y')
then
form := form^.next
end; (* d_form *)

begin (* delete form *)
writeln;
write ('Enter the number of the form to be deleted: '); 
readln (f^.num);
temp_f_ptr := head_list;
convert (f^.num,f^.key);
(* check if first form *)
if (f^.key = temp_f_ptr^.form_num)
then
d_form (head_list)
else
begin

   { Find the form that is to be deleted. }

   while (temp_f_ptr^.next^.next <> nil) and
   (f^.key <> temp_f_ptr^.next^.form_num) do
   temp_f_ptr := temp_f_ptr^.next;
   if (f^.key = temp_f_ptr^.next^.form_num)
then
d_form (temp_f_ptr^.next)
else
writeln ('form ',f^.key,' not found.');
end
end; (* delete form *)

procedure dump_question_list (var head_list : quest_list);

{ This procedure writes the updated version of the question
list to a file 'QUEST.DTA'. The file will overwrite the
old version of the question file. }

var
  temp_q : quest_list;
begin { dump_question_list }
rewrite (quest_outfile,'quest.dta');
temp_q := head_list;
while (temp_q <> nil) do
begin
write (quest_outfile,temp_q^.key,temp_q^.quest1_string);
writeln (quest_outfile,temp_q^.quest2_string);
temp_q := temp_q^.next
end
end; { dump_question_list }
procedure dump_standard_forms (var head_form : form_ptr);

{ This procedure will write the updated version of the standard forms to a file `STAN.FRM'. The procedure will overwrite the old version of the file. }

var
    temp_f_ptr : form_ptr;
    i : integer;

begin { dump_standard_forms }
{ rewrite the standard form file }
rewrite (form_file, 'stan.frm');
temp_f_ptr := head_form;
while (temp_f_ptr <> nil) do
begin
    write (form_file, temp_f_ptr^.form_num);
    for i := 1 to max_quest_nums do
        write (form_file, temp_f_ptr^.quest_nums[i]);
    writeln (form_file);
    temp_f_ptr := temp_f_ptr^.next
end
end; { dump_standard_forms }

procedure heading;

{ This procedure will display the heading to the question maintenance program and the menu of options that can be performed on the question and standard forms file. }

begin { heading }
write;
writeln ('This program modifies the list of questions for the ICES system. This program contains the following options:');
writeln ('1) M - Multiple question modification');
writeln ('2) S - Single question modification');
writeln ('3) D - Deletion of a question');
writeln ('4) A - Adding a question');
writeln ('5) P - Create a printable question list');
writeln ('6) N - Add standard form numbers');
writeln ('7) X - Delete a standard form');
writeln ('8) L - List standard forms');
writeln ('9) ? - Prints this menu');
writeln
writeln ('The maximum size of a question is 86 characters.');
writeln
end; (* heading *)

begin (* main *)

build_question_list (head_list);
get_forms (stan_forms);
heading;

repeat (* until you want to stop *)
write ('Enter option : (M,S,D,A,P,N,X,L,?) ');
readln (option);
case option of
  'm', 'M' :
mult_quest (head_list);
  's', 'S' :
single_quest (head_list);
  'd', 'D' :
delete_quest (head_list);
  'a', 'A' :
add_quest (head_list);
  'p', 'P' :
print_quest (head_list);
  'n', 'N' :
add_stan_forms (stan_forms);
  'x', 'X' :
delete_stan_forms (stan_forms);
  'l', 'L' :
print_stand_forms (stan_forms);
  '?' :
heading;
others :
begin
  writeln;
  writeln ('Invalid command')
end
end; (* case *)
writeln;
write ('Do you want to continue? (Y or N) ');
readln (option)
until (option = 'n') or (option = 'N');
dump_question_list (head_list);
dump_standard_forms (stan_forms)
end.
(* main *)
program edit_stats;

{ Written by: Peter Weiler
For: Faculty Evaluation System
Date last revised: May, 1983

This program was added to the system to allow the user to edit the header records in the statistics files. The data entry personal were creating a large number of errors in producing header cards. This program allows the user to change the header card values. Previously the user would have to reenter all of the cards for the instructor.

The statistics file is in the following format:

A header record containing:
Department number : field (1 – 3)
Course number : field (4 – 6)
Section number : field (7 – 9)

The data records contain seven real numbers. The first five are the frequency of question answers. The sixth number is the mean of the answers. The last position is the standard deviation of the question answers.

All of the statistics files are in a subdirectory with the logical name 'STS:'. The file names are 'INnnnn.DTA' where ,nnnn, is a number in the range (0000 – 9999).
}

const
  num_length = 3;

type
  num_type = packed array [1..3] of char;

header_type = record
  dept_code : num_type;
  course_num : num_type;
  sect_num : num_type
end;

(* The stats record is the structure generated by the stats program. This is the structure of the information that follows the header card. *)

stat_ptr = ^stats_type;
\text{stats_type = record}
\begin{verbatim}
  freq1,freq2,freq3,freq4,freq5 : integer;
  mean, stand_dev : real;
  next : stat_ptr
end;
\end{verbatim}

\text{var}
\begin{verbatim}
  stats : text;
  header : header_type;
  head_stat : stat_ptr;
  in_string : packed array [1..14] of char;
  num : integer;
  question : char;
\end{verbatim}

\text{procedure read_stats (var header : header_type; var head_stat : stat_ptr);}
\begin{verbatim}
  { This procedure will read in the information from the 
    statistics file. The header record is read into a record 
    and the statistics are read into a linked list. }
\end{verbatim}

\text{var}
\begin{verbatim}
  temp_stat : stat_ptr;
\end{verbatim}

begin \{read_stats\}
  with header do
    readln (stats,dept_code,course_num,sect_num);
  new (head_stat);
  temp_stat := head_stat;
  with head_stat' do
    begin
      read (stats,freq1,freq2,freq3,freq4,freq5);
      readln (stats,mean,stand_dev)
    end;
  while not eof (stats) do
    begin
      new (temp_stat'.next);
      temp_stat := temp_stat'.next;
      with temp_stat' do
        begin
          read (stats,freq1,freq2,freq3,freq4,freq5);
          readln (stats,mean,stand_dev)
        end
    end;
  temp_stat'.next := nil
end; \{read_stats\}

\text{procedure convert3 (num : integer; var letters : num_type);}
{ This procedure will convert an integer number into an array [1..3] of digits.
}

var
  i : integer;

begin { convert3 }
  for i := num_length downto 1 do
    begin
      letters[i] := chr((num mod 10) + ord('0'));
      num := num div 10
    end
  end; (* convert3 *)

procedure get_num (var num : integer);

{ This procedure will read in digits from a terminal and create an integer number. The routine checks to see that only valid digits are entered (0 - 9). If an invalid character is entered the procedure will print an error message and have the user reenter the number. }

var
  correct : boolean;
  ch : char;

begin { get_num }
  repeat { until correct }
    correct := true;
    num := 0;
    repeat { until wrong or eoln }
      ch := input"";
      get (input);
      if (ch >= '0') and (ch <= '9')
        then
          num := num * 10 + (ord(ch) - ord('0'))
        else
          begin
            correct := false;
            readln;
            write ('invalid number reenter: ')
          end
    until (eoln or not correct);
  if correct
    then
      readln
  until correct
end; { get_num }
procedure get_file (file_number : integer; var temp_file : text);

(* This procedure will get an input stats file. The number that is passed to it is the number of the file. *)

begin { get_file }
  in_string := 'sts:IN0000.DTA';
  in_string[7] := chr(file_number div 1000 + ord('0'));
  in_string[8] := chr((file_number div 100) mod 10 + ord('0'));
  in_string[9] := chr((file_number div 10) mod 10 + ord('0'));
  in_string[10] := chr(file_number mod 10 + ord('0'));
  reset (temp_file,in_string)
end; (* get file *)

procedure write_stats ( var header : header_type;
                        var head_stat : stat_ptr);

{ This procedure will write the updated information back to the original statistics file. The old statistics file is deleted in the process. }

var
  temp_stats : stat_ptr;

begin { write_stats }
  rewrite (stats,in_string);
  with header do
    writeln (stats,dept_code,course_num,sect_num);
  temp_stats := head_stat;
  while (temp_stats <> nil) do
    begin
      with temp_stats^ do
        begin
          write ( stats,freq1:4,freq2:4,freq3:4,freq4:4,freq5:4);
          writeln (stats,mean:10:5,stand_dev:10:3)
          end;
          temp_stats := temp_stats^.next
        end;
  end; { write_stats }

procedure edit_stats ( var header : header_type);

{ This procedure displays the header record and asks the user if it is to be changed. The user has the option of changing the header record or returning to the main routine. The header record is displayed again after the change has been made. }

var
    answer : char;
    num : integer;

begin { edit_stats }
    repeat { until info correct }
        writeln;
        writeln ('Department number = ',header.dept_code);
        writeln ('Course number = ',header.course_num);
        writeln ('Section number = ',header.sect_num);
        writeln;
        write ('Do you wish to change the information? (Y/N) '); readln (answer);
        writeln;
        if (answer = 'Y') or (answer = 'y')
            then
                begin
                    write ('Enter the dept. no. : '); get_num (num);
                    convert3 (num,header.dept_code);
                    write ('Enter the course number : '); get_num (num);
                    convert3 (num,header.course_num);
                    write ('Enter the section number: '); get_num (num);
                    convert3 (num,header.sect_num);
                end
            until (answer <> 'y') and (answer <> 'Y')
    end; { edit_stats }

begin { main }
    writeln;
    writeln ('This routine allows the user to edit the header record of the statistics file.');
    repeat { until done }
        writeln;
        write ('Enter the statistics file number: '); get_num (num);
        get_file (num,stats);
        if eof (stats)
            then
                writeln ('The file is empty enter a different number.');
        else
            begin
                read_stats (header,head_stat);
                edit_stats (header);
                write_stats (header,head_stat); end;
        write ('If you have another entry type ,c, to continue. '); readln (question)
until (question <> 'c') and (question <> 'C')
end.

{ main }