A GRADING REPORT GENERATOR

by

RONALD WING-KEONG LAW

S.B., Massachusetts Institute of Technology
(1973)

SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF

SCIENCE

at the

MASSACHUSETTS INSTITUTE OF

TECHNOLOGY

June, 1976

Signature of Author .....
Alfred P. Sloan School of Management, May 7, 1976

Certified by .....
Thesis Supervisor

Accepted by .....
Chairman, Departmental Committee on Graduate Students

ABSTRACT

Title of thesis : A GRADING REPORT GENERATOR

Name of author : RONALD WING-KEONG LAW

Submitted to the Alfred P. Sloan School of Management on May 14, 1976 in partial fulfillment of the requirements for the degree of Master of Science.

The Grading Report Generator is a computer program designed to automate the process of preparing grading reports and to provide a set of meaningful statistics on the performance of the students in order to aid both the teacher and the students. It allows the user to divide the students in the subject into sections. The subject format is specified by the user as any combination of grades for quizzes, problem sets, machine problems, papers, take-home examinations, etc. Each grade is assigned a weight so that a weighted average grade may be computed for each student. The GRG generates class and section reports which arrange the students in some user specified order, e.g. by their name, by their year, by their latest grade, by their average. Another kind of report shows the overall average for each grade in a class or section, and the percentage of students who participated in it. A count of the number of students who are taking the subject for a letter grade, PASS/FAIL, as listeners or have dropped the subject is also produced.

It is intended that these reports, produced by the GRG on a time-sharing computer system, would provide meaningful and timely feedback to the teacher and the students.

Thesis supervisor : Stuart E. Madnick

Title : Assistant Professor of Management Science

ACKNOWLEDGEMENTS

The author wishes to express his sincere gratitude to his thesis advisor Prof. Stuart Madnick for his guidance and encouragement in making this work possible. Through Stu's constructive criticism and valuable suggestions, the author has gained an enormous amount of experience in the solving of a real life problem. Lastly, the amount of computer time made available by Stu is greatly appreciated.

TABLE OF CONTENTS

CHAPTER ONE : INTRODUCTION

One of the more time-consuming tasks involved in the administration of a subject is the monitoring of the students' performance. For example, it is customary for the person in charge of a subject to calculate after every quiz the mean, the median, the range of grades, and the number of students who took part in it. This information provides useful feedback to both the teacher and students. The teacher may use this information to decide on the way the subject will be conducted in the future. The student may picture where he stands in the subject relative to his classmates and decide whether extra effort will be required.

The objective of this thesis is two folds :

1) to automate the repetitious task of manually preparing grading reports, and
2) to provide a set of meaningful statistics on the performance of the students in order to aid both the teacher and students.

In the past there had been numerous grading programs that were written for these purposes. But they were generally found to be either too restrictive, because they were designed for one kind of subject format, or they provided too little useful information. Furthermore, their documentation either did not exist or was incomplete, thus making it extremely difficult for current users to maintain or update them.

In order to useful to a broad spectrum of users, a grading program must be able to accomodate subjects of different formats. It must also provide the users with a comprehensive users' guide so that users may take

advantage of the various options they may want to exercise. It must also be thoroughly designed and documented, with particular emphasis on maintainability and future extensibility.

The present implementation of the Grading Report Generator is a FORTRAN IV program that runs in a PRIME 300 minicomputer at the Sloan School of Management. The GRG allows the user to define the format of a subject as a collection of scores from quizzes, problem sets, machine problems, take-home exams, etc. Each score carries with it a weight which reflects a percentage of the requirements for the subject. The overall average is a weighted average of the score weight and the score achieved, and it is computed for every student in the subject. The students may be assigned into sections. There are two classes of reports that may be generated, and these apply to the class as a whole and to the individual sections. The first kind of report deals with the ordering of the students by some values, e.g. by the name of the students, by the year of students, by the latest grade, by the student overall average, etc. The second kind of report deals with the aggregate of the student data in a class or section, e.g. the average for a particular grade, the rate of participation, the number of students who registered the subject as a graded student, listener, etc.

Although the present implementation may not contain all the desirable features, it had been designed with future extensions in mind. A well structured framework had been laid out with emphasis on modularity. The coding adhered rigidly to a set of rules and is described in detail in Chapter Three.

CHAPTER TWO : HOW TO USE THE GRG

This chapter is intended as a user's guide for the Grading Report Generator. Basically, the GRG accepts input from an input file, performs the specified processing, and then outputs the requested reports to the output file. The input file may be created in a number of ways. A deck of cards may be keypunched and then read into the PRIME. The basic procedure of accessing and logging into the PRIME minicomputer will not be discussed here. Assuming the user is attached to the directory where the GRG resides and he has the input already punched as a deck of cards. The cards should be read in by issuing the following commands to the operating system :

```
ASSIGN CR1
CRMPC filename                filename is name assigned to file
CLOSE ALL
UNASSIGN CR1
```

The input file may also be created initially by the text editor, or subsequently modified by it. When the user is ready, he invokes the GRG by entering :

```
RESUME  *GRG
```

When the GRG is activated, it issues

```
PLEASE ENTER FILENAME
```

and user responds by :

```
filename
```

The GRG will run to completion and produce an output named REPORT.

Before describing the contents of the input file, it is necessary to understand the functions of the GRG. The GRG is designed to allow for up to 9 sections for the subject, each section not to exceed 80 students. The total number of students in the subject may not exceed 300. For the subject, a maximum of 13 scores or grades may be specified. The range of a score is from 0 to 100. A numerical weight expressed as a percentage must be assigned to each score, and it must also range from 0 to 100. It is recommended that the sum of the weights make up 100 so that a normalized average score would result.

Input File

The input file consist of two groups of statements

1) control statements
2) data statements

Control statements are variable in number and they are further divided into 4 groups. They are

1) subject statement
2) section statements
3) grade statements
4) listing statements

Data statements are usually fixed in number because each one corresponds to a student. When students have dropped the subject, their data statements may remain in the file since they do not interfere with the data of others. It is recommended that the data statements be maintained in alphabetical order of the student name for ease of handling,

although the GRG has no requirement for them to be in any order.

Subject Statement

This is a single statement (one line or one card) of text containing up to 80 characters. This is also the first statement of the input file. The user may use this statement to identify the subject, the lecturer and any other information pertinent to the subject. This statement appears as the heading for reports which involves the whole class.

Section Statements

The section statements are similar to the subject statement in that the GRG treats them as text. The user must include one section statement for every section the class is divided into. A section statement will appear as the heading for every report of that section. The collection of section statements must be preceeded by a line which states the number of section statements that are present. This number is read by the GRG in I2 format in columns 1 and 2. The maximum allowable number of sections is 9.

Grade Statements

The group of grade statements are preceeded by a line which declares how many grade statements are present. This number is also read in from columns 1 and 2 in I2 format. The maximum number of grade statements is 13. For each grade statement the following format applies :

column 1, 2, 3	any three characters
column 4	BLANK
column 5, 6, 7	a weight expressed as a percentage (1 to 100)

It is suggested that the user adapt a convention regarding the three character assignment, e.g. QZ1, QZ2, ... for quiz 1, quiz 2, ... PS1, PS2 ... for problem sets etc. The first grade statement will correspond to the first grade columns (41,42,43) on the data statement, the second grade statement with the second grade columns (44,45,46) etc.

Listing Statements

There are two listing statements, each containing different logical flags. Each logical flag controls whether a particular report is generated or not. A T, meaning true, will enable a report to be generated, while a F, meaning false, will suppress that report. The first listing statement contains 8 logical flags which control the overall or class reports.

Flag 1	Generates a class report ordered by the student's sequence number.
Flag 2	Generates a class report ordered by the year or class of the student.
Flag 3	Generates a class report ordered by the course (department) the student belongs to.
Flag 4	Generates a class report ordered by the names of the students in alphabetical order.
Flag 5	Generates a class report ordered by the registration status of the students.
Flag 6	Generates a class report ordered by the section the student belongs to.
Flag 7	Generates a class report ordered by the last or latest score the user has defined.
Flag 8	Generates a class report ordered by the weighted average score of the student.

Columns 1 through 8

The second listing control statement contains 3 logical flags.

Flag 1	Generates a section report for each section and orders the students by their names.
Flag 2	Generates a section report for each section and orders it by the latest score.
Flag 3	Generates a section report for each section and orders it by the overall average.

columns 1
through 3

Data Statements

Each data statement contains the background information and the scores for one student. There may be up to 300 data statements to represent 300 students.

COLUMNS	INFORMATION
1 to 5	Sequence number
6	BLANK
7 - 8	Year, e.g. G, 1, 2, 3, 4, SG
9	BLANK
10 to 12	Course, e.g. 2, 2A, 15, 63
13	BLANK
14 to 36	Student Name
37	Student Status, e.g. BLANK means the student is taking the subject for a letter grade N means he is taking the subject for a PASS/FAIL grade L means he is a listener C means he has dropped the subject
38	BLANK
39 to 40	Section
41 to 43	Score 1
44 to 46	Score 2
⋮	⋮
77 to 79	Score 13

CHAPTER THREE: INTERNAL DESIGN OF THE GRG

Choice of Computer

It is highly desirable to be able to execute the GRG online because of delays and other inconveniences involved with a batch system. There are 3 general-purpose time-sharing computer systems at MIT that are suitable for this application. The Information Processing Center operate an IBM S/370-168 and a HIS 6180, both being large scale computer systems. The Sloan School of Management operates a PRIME 300 minicomputer in the basement of the Sloan Building.

Although the execution speed of the PRIME is slow by large computer standards, it is quite adequate for this application because of the small amount of processing activity involved. Also the memory and storage capacity of the PRIME is small compared to the IPC's two large systems, but again this application does not require an extraordinary amount of space. Besides, the operating system on the PRIME is a truly general-purpose time-sharing system which provides essentially the same services as the large computers but without many of the costly overheads. The PRIME currently supports as many as 7 terminals simultaneously, thus providing the users with good accessibility to a terminal. Therefore the PRIME was selected as the computer to develop and run the GRG. This is also consistent with current trends towards decentralizing of processing functions from large central facilities to local, distributed sites where smaller machines takeover the less demanding chores of computation and provide more timely turnaround of results.

Choice of Language

There are currently 3 suitable language for implementing the GPG on the PRIME: FORTRAN IV, BASIC and Macro Assembler. Macro Assembler was the first to be eliminated in the choice because it was not a higher order language. A higher order language program usually requires less time to develop, costs less to maintain, and may be transported to another machine when the compiler exists for it. The author selected FORTRAN IV as the implementation language, not because there were any short comings with BASIC, but rather the author is more familiar with FORTRAN.

Although FORTRAN is termed a higher order language, there are many desirable features of HOL's that it lacked, e.g. nested procedures, data-structures and the if-then-else construct, etc. Therefore it would be particularly difficult to apply the principles of structured programming to program development unless some programming rules or restrictions were introduced and strictly followed. They include

1) Restrict program modules to under 2 pages in length.
2) Declare all the major variables in separate files. Then include them into program modules by the $INSERT preprocessor facility.
3) Restrict the scope of GOTO statements.
4) Extensive use of CONTINUE statements to serve as the bounds of DO-loops and as program branch destinations.
5) Assign statement numbers such that they appear in ascending order on the left margin.

Rule 1 essentially prohibits program modules to become too long and

thus difficult to follow. It forces the designer to break down the whole system into smaller modules and have each module perform a single or a small set of functions.

Rule 2 guarantees the integrity of data declarations over the various program modules by allowing the compiler to merge the declarations during compile time. When modifications to the system is required, only the declaration files need be modified. The changes would be picked up when the program modules are later recompiled.

Since it is not possible to eliminate the use of GOTO statements in a FORTRAN program because of the basic limitations of the language, the use of GOTO statements should be carefully planned. For example, a branch out of a DO-loop should be to the statement immediately following the end of the loop. The program should also be structured such that backward branching with GOTO statements be replaced by DO-loops whenever possible.

Rule 4 and Rule 5 combined would make the effort required to follow the program flow much easier because ascending statement numbers suggest the sequential order of program steps.

The hierachy diagram of the GRG is depicted in Figure 1. In the diagram, each program module is represented by a box. The diagram does not show the order of execution of the modules. It only shows the logical level of processing each module is responsible for. The following is a description of each module.

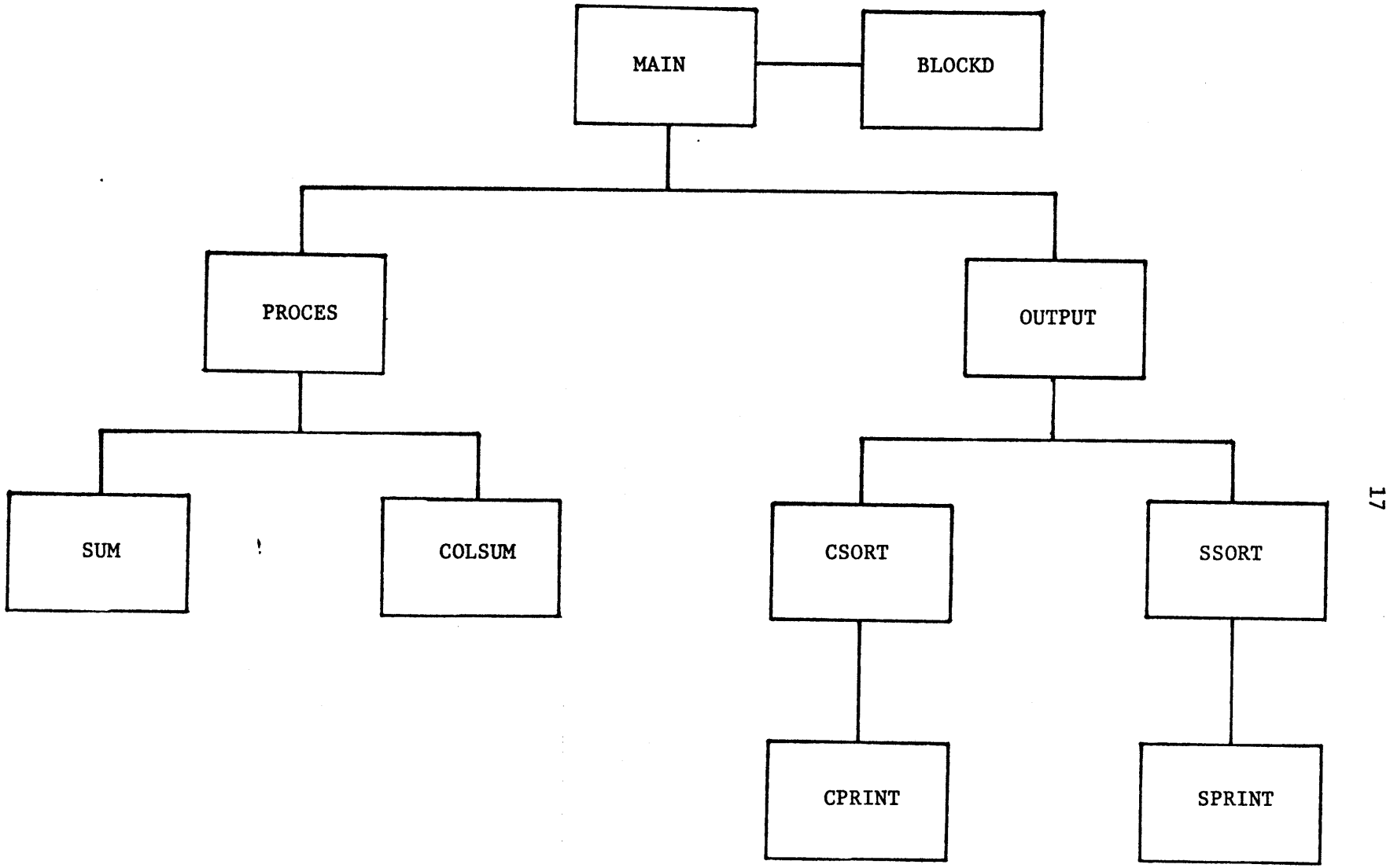

Figure 1. Program Hierachy

Program Modules

MAIN

This is the main program of the GRG. It obtains the input file name from the user, opens the input and output files, and then reads in the control statements and data statements from the input file. Control is then passed first to module PROCES and then to module OUTPUT. After returning from OUTPUT, all functions of the GRG are complete. The input and output files are closed and the GRG returns control to the operating system.

PROCES

This module performs most of the arithmetic of the GRG. The weighted average of each student is computed. Students are also assigned into the different sections that they belong to by taking up a section pointer. They are further classified by their status. Student scores are also being accumulated in order to compute section and class averages for each grade. In the computation of the average, only the grades for credit students, i.e. students who are getting a letter grade or a grade of PASS/FAIL, are being used. Moreover, exceptionally low grades (under 20) are also not included because of the hugh negative bias they would introduce. Another statistic known as the participation rate is computed on credit student by sections and as a class. This rate shows the percentage of the credit students who took part in a certain quiz or turned in their problem sets etc.

OUTPUT

This module calls subroutines CSORT and SSORT to perform the various sorts requested by the user and outputs the result by calling the print

subroutines CPRINT and SPRINT. In addition, a table which contains class averages and the participation rate for every grade is provided.

SUM

This is a function which returns the sum of an array of integers.

COLSUM

This is a function which returns the sum of a column of a two-dimensional integer array.

CSORT

This is a subroutine which performs an interchange sort on an array of pointers for the whole class of students. The sort argument may be specified as a field of the student character data such as the student name, the section he belongs to, etc. Alternatively, the sort argument may be one of the students' grades or their average score.

SSORT

This subroutine is similar to CSORT in function but it operates only on the pointers for the section of students which has been specified.

CPRINT

This subroutine generates the class reports by putting out the appropriate headings to indicate what type of report it is. The students are presented in the report in the order specified by the class pointers.

SPRINT

This subroutine is similar to CPRINT except that it generates a report for the section that has been specified instead of the whole class, and the ordering is by the section pointers.

BLOCKD

This is the BLOCK DATA subprogram where all the initialization for the variables in labelled COMMON are specified.

The Unassigned Section

At the beginning of a term, a student may indicate that he would register in the subject, but he could not select which section to attend because of the scheduling for other subjects. This situation implies that the GRG must handle these unassigned students as a special section. When the student's data statement does not specify which section he belongs to, he is automatically assigned to the unspecified section. Later in the term, when his section assignment is confirmed, his data statement may be updated to reflect the change.

Data Organization

In the interest of structured and modular programming, program modules are deliberately kept to a small, readable size. As a result, most of the program variables would be referenced by more than one program module. In order to minimize the passing of these variables as arguments in subroutine calls and function references, they are declared as global or COMMON variables. This means the COMMON variables must be declared in every module where they are referenced. The PRIME FORTRAN IV compiler has a very useful feature in the $INSERT compile-time facility. It allows the merging of statements from another file into the current file being compiled as if those statements were part of the current file.

In the GRG, the COMMON variables are separated into 5 LABELLED COMMON areas COM1, COM2, COM3, COM4, and COM5 according to their data type. These are stored on the system as files COM1 through COM5 respectively.

COM1

COMMON COM1 consists of INTEGER variables which are used to store packed character strings two to a storage word.

INFILE(3) stores the input filename of up to 6 characters long.

SBHEAD(40) stores a string of 80 characters which serves as the subject heading in class reports.

SCHEAD(40, 10) stores a string of 80 characters for each of the 9 assignable sections plus the unassigned section. This string is used to identify the sections in section reports.

GRHEAD(2, 14) stores the 3 character grade or score heading for each of the 13 allowable grades plus one for the column which contains the average.

STUDC(20,300) stores the 40 character information for each of the 300 allowable students in the class. The fields which comprise the 40 characters are describled in Chapter Two.

NUMBER(10) is a constant string of ASCII character initialized from 1 to 10. It is used to identify the section the student is assigned to.

PRFILE(3) is initialize to REPORT. This is the name of the output file.

CRHEAD(10) is another 20 character heading used in the reports. It reads 'STUDENTS ORDERED BY'.

RPHEAD(5,8) is an array of 10 character strings used in the reports in conjunction with CRHEAD to describe the type of report.

COM2

COM2 consists of all the numeric variables used in the program and they are all typed INTEGER.

ISEC is the number of sections in the class.

IGRADE is the number of grades or scores presented.

STUDN(14,300) stores the 13 allowable scores plus the average for each of the up to 300 students.

WEIGHT(13) stores the weighing factor for each grade or score.

NCLASS is the total number of students in the class.

NSEC(10) is the number of students in each of the 9 assignable sections plus the unassigned section.

AVGCOL is the number of the column in STUDN which is used to hold the average. It has a value of IGRADE + 1.

WTOTAL is the sum of the specified weights for the grades that are present.

TSEC is the number assigned to the unspecified section and has a value of ISEC + 1.

CSEC is the number of the section which contains the total for the

individual sections.

GRADED(11), PSFAIL(11), LISTEN(11), DROP(11) contain the number of students who select a letter grade, a grade of PASS/FAIL, select the subject as a listener, or who dropped the subject, respectively. They allow for up to 9 sections plus an unspecified section, and the last element contains a sum of each classification.

GSECOL(10,14) contains the sum of grades in each column of the 9 allowable sections plus the unassigned section.

NSECOL(10,14) contains the number of students who figured in the above count.

MASAVG(11,14) contains the master average of grades by column for the 9 allowable sections, the unassigned section, and for the whole class.

MASPCT(11,14) is similar to MASAVG but contains the participation rate.

COM3

COM3 contains the only REAL variable used in the program, RTEMP.

COM4

COM4 declares the global logical variables in the program.

OLIST(8) contains 8 logical flags used to control the overall class listing reports.

SLIST(3) contains 3 logical flags for the sectional listing reports.

These flags are described in detail in Chapter Two.

COM5

COM5 contains variables used as pointers in the program and are all typed INTEGER.

CLASS(300) are pointers to the whole class of up to 300 students.

SEC(80,10) are section pointers for 10 sections, each having up to 80 students.

CHAPTER FOUR : EXTENSIONS AND IMPROVEMENTS

There are 3 areas in the GRG which may be considered for extensions or improvements. They are

1) increase the capacity of the GRG to handle more students, more sections, and the number of scores,
2) generate additional reports of interest, and
3) provide an online data-entry, enquiry and edit capability.

Increasing the Capacity

The current size of the GRG is less than 20K words (1K = 1024). This includes about 8K words of instructions and 12K words of data. As the maximum allowable size for a program for the PRIME minicomputer is 64K words, there is ample space to expand the GRG to handle more students than the present maximum of 300. Also the number of students per section and the number of sections allowed may also be increased. To carry out these changes, the data declarations which are contained in the five COMMON areas may easily be modified. For example, to increase the number of students say from 300 to 500, the dimensions of 3 variables should be changed. In COM1, STUDC (20,300) becomes STUDC (20,500). In COM2, STUDN (14,300) becomes STUDN (14,500). In COM5, CLASS (300) becomes CLASS (500). In addition, a constant used as a loop count upper bound for reading in student data statements in program module MAIN should be changed from 300 to 500.

Generate Additional Reports

There are many other statistics on the class or sections that a teacher may want to collect. For example, an average score may be computed for students belonging to each undergraduate class and for graduate students. The average score for students belonging to different courses may also be collected. In general there are numerous such cross-tabulating statistics which may be gathered on the different student attributes.

Additional Online Capability

This is perhaps the most challenging of the 3 areas for extensions and improvements. At present, a user who wishes to carry out the functions online data entry, enquiry and editing must use the TEXT EDITOR of the PRIME system. The TEXT EDITOR allows the searching of fully specified text strings as well as partially specified text strings. However, when it comes to entering data in specific columns, the TEXT EDITOR becomes very awkward because there is no command to allow easy column justification in edit mode.

It would be very helpful to the user if a set of powerful editing commands is available within the GRG to allow him to inquire and modify the database in core and to produce an updated version of the INPUT file. This would entirely eliminate the need for keypunching or use of the TEXT EDITOR. The user may then created the INPUT file online, edit the file online and produce reports online.

C MAIN PROGRAM

```
C MAIN PROGRAM
      INTEGER
$INSERT COM1
$INSERT COM2
$INSERT COM3
$INSERT COM4
$INSERT COM5
      WRITE(1,10)
10    FORMAT('PLEASE ENTER FILENAME')
      READ(1,20) INFILE
20    FORMAT(3A2)
      CALL SEARCH(1,INFILE,1)     /* OPEN INFILE ON FUNIT1 */
      CALL SEARCH(2,PRFILE,2)     /* OPEN PRINT FILE ON FUNIT2 */
      READ(5,30) SBHEAD
30    FORMAT(40A2)
      READ(5,40) ISEC
40    FORMAT(I2)
      DO 50 I = 1,ISEC
      READ(5,30) (SCHEAD(J,I),J = 1,40)
50    CONTINUE
      READ(5,40) IGRADE
      DO 70 I = 1,IGRADE
      READ(5,60) GRHEAD(1,I),GRHEAD(2,I),WEIGHT(I)
60    FORMAT(2A2,I3)
70    CONTINUE
```

```
C     NOW READ IN LOGICAL FLAGS
      READ(5,80) (OLIST(I),I = 1,8)
80    FORMAT(8L1)
      READ(5,80) (SLIST(I),I = 1,3)
      DO 100 I = 1,300
      READ(5,90,END=110) (STUDC(J,I),J = 1,20),
     1                   (STUDN(J,I),J = 1,IGRADE)
90    FORMAT(20A2,13I3)
100   CONTINUE
110   CONTINUE
      NCLASS = I-1
      AVGCOL = IGRADE + 1
      GRHEAD(1,AVGCOL) = 'AV' /* ASSIGN AVG INTO GRHEAD */
      GRHEAD(2,AVGCOL) = 'G '
      TSEC = ISEC + 1  /* THE UNASSIGNED SECTION */
      CSEC = ISEC + 2  /* CLASS AVG AND PCT REF */
      DO 120 I = 1,40  /* COPY UNASSIGNED SECTION HEADING */
      SCHEAD(I,TSEC) = SCHEAD(I,10)
120   CONTINUE
      WTOTAL = SUM(WEIGHT,IGRADE)
C
C
      CALL PROCES
      CALL OUTPUT
      CALL SEARCH(4,0,1) /* CLOSE FUNIT 1 */
      CALL SEARCH(4,0,2) /* CLOSE FUNIT 2 */
      CALL EXIT
      END
```

SUBROUTINE PROCES

```
      SUBROUTINE PROCES
      INTEGER
      LOGICAL CREDIT
$INSERT COM1
$INSERT COM2 !
$INSERT COM3
$INSERT COM4
$INSERT COM5
      /* PROCESS STUDENTS INDIVIDUALLY
      DO 400 I = 1,NCLASS
      CLASS(I) = I /* ASSIGN CLASS PTR */


      /* COMPUTE INDIVIDUAL AVERAGE */
      RTEMP = 0.0
      DO 100 J = 1,IGRADE
      RTEMP = RTEMP + WEIGHT(J)* STUDN(J,I)
100   CONTINUE
      STUDN(AVGCOL,I) = IFIX(RTEMP/WTOTAL)
```

```
      /* ASSIGN SECTION PTR AND COUNT STUDENTS IN EACH SECTION
      DO 110 J = 1,ISEC /* TRY MATCH WITH A DECLARED SECTION */
      IF (STUDC(20,I).NE.NUMBER(J)) GOTO 110 /* LOOP IF DONT MATCH */
      GOTO 200 /* MATCHED,EXIT WITH THIS J */
110   CONTINUE
      J = TSEC /* SET J TO THE ANASSIGNED SECTION */
200   CONTINUE

      NSEC(J) = NSEC(J) + 1 /* COUNT UP ONE */
      II = NSEC(J)
      SEC(II,J) = I /* SECTION J PTR */


      /* DETERMINE STATUS OF STUDENT */
      IF (STUDC(19,I).NE.2H  ) GOTO 210
      GRADED(J) = GRADED(J) + 1
      CREDIT = .TRUE.
      GOTO 300 /* PROCEED TO NEXT STEP OF PROCESSING */
210   CONTINUE
      IF (STUDC(19,I).NE.2HN ) GOTO 220
      PSFAIL(J) = PSFAIL(J) + 1
      CREDIT = .TRUE.
      GOTO 300 /* PROCEED TO NEXT STEP OF PROCESSING */
220   CONTINUE
      CREDIT = .FALSE.  /* MUST BE NON-CREDIT STUDENT */
      IF (STUDC(19,I).NE.2HL ) GOTO 230
      LISTEN(J) = LISTEN(J) + 1
      GOTO 300 /* PROCEED TO NEXT STEP OF PROCESSING */
230   CONTINUE
      DROP(J) = DROP(J) + 1
```

```
300   CONTINUE
      /* ACCUMULATE GRADES OF CREDIT STUDENTS BY SECTION AND COLUMN */
      IF (.NOT.CREDIT) GOTO 400 /* OMIT THIS STEP FOR NON-CREDITS
      DO 310 K = 1,AVGCOL
      IF (STUDN(K,I).LT.20) GOTO 310 /* FILTER OUT NOISE */
      NSECOL(J,K) = NSECOL(J,K) + 1
      GSECOL(J,K) = GSECOL(J,K) + STUDN(K,I)
310   CONTINUE
400   CONTINUE
      /* END OF INDIVIDUAL STUDENT PROCESSING */


      /* COMPUTE THE TOTAL NUMBER OF GRADED, PASS-FAIL, */
      /* LISTEN AND DROPPED STUDENTS IN CLASS */
      GRADED(CSEC) = SUM(GRADED,TSEC)
      PSFAIL(CSEC) = SUM(PSFAIL,TSEC)
      LISTEN(CSEC) = SUM(LISTEN,TSEC)
      DROP(CSEC) = SUM(DROP,TSEC)
```

```
      /* COMPUTE MASTER AVERAGE AND PERCENTAGE BY SECTION AND COLUMN */
      DO 600 J = 1,TSEC
      ITEMP = GRADED(J) + PSFAIL(J) /* NUMBER OF CREDIT STUDENTS IN SEC
      DO 500 K = 1,AVGCOL
      MASAVG(J,K) = GSECOL(J,K)/NSECOL(J,K)
      MASPCT(J,K) = (NSECOL(J,K)*100)/ITEMP
500   CONTINUE
600   CONTINUE



      /* COMPUTE CLASS AVERAGE AND PERCENTAGE OF CREDIT STUDENTS WHO DID
      ITEMP = GRADED(CSEC) + PSFAIL(CSEC) /* # OF CREDIT STUDENTS */
      DO 700 K = 1,AVGCOL
      ITEMP2 = COLSUM(NSECOL,TSEC,K)
      MASAVG(CSEC,K) = COLSUM(GSECOL,TSEC,K)/ITEMP2
      MASPCT(CSEC,K) = (ITEMP2*100.0)/ITEMP
700   CONTINUE

      RETURN
      END
```

SUBROUTINE OUTPUT

```
      SUBROUTINE OUTPUT
      INTEGER
      INTEGER LINE1(4),LINE2(12),LINE3(12)
$INSERT COM1
$INSERT COM2
$INSERT COM3
$INSERT COM4
$INSERT COM5

      DATA LINE1/'SECTION'/
      DATA LINE2/'CREDIT STUDENT AVERAGE  '/
      DATA LINE3/'THEIR PARTICIPATION RATE'/
      /* GENERATE CLASS OVERALL SORTED LISTINGS IF SELECTED */
      IF (.NOT.OLIST(1)) GOTO 20
      CALL CSORT(1,6,0) /* SORT BY SEQ# */
      CALL CPRINT(1)

20    CONTINUE
      IF (.NOT.OLIST(2)) GOTO 30
      CALL CSORT(7,8,0) /* SORT BY YEAR */
      CALL CPRINT(2)

30    CONTINUE
      IF (.NOT.OLIST(3)) GOTO 40
      CALL CSORT(9,12,0) /* SORT BY COURSE */
      CALL CPRINT(3)
```

```
40    CONTINUE
      IF (.NOT.OLIST(4)) GOTO 50
      CALL CSORT(13,36,0) /* SORT BY NAME */
      CALL CPRINT(4)

50    CONTINUE
      IF (.NOT.OLIST(5)) GOTO 60
      CALL CSORT(37,38,0) /* SORT BY STATUS */
      CALL CPRINT(5)

60    CONTINUE
      IF (.NOT.OLIST(6)) GOTO 70
      CALL CSORT(39,40,0) /* SORT BY SECTION */
      CALL CPRINT(6)

70    CONTINUE
      IF (.NOT.OLIST(7)) GOTO 80
      CALL CSORT(0,0,IGRADE) /* SORT BY LAST GRADE */
      CALL CPRINT(7)

80    CONTINUE
      IF (.NOT.OLIST(8)) GOTO 100
      CALL CSORT(0,0,AVGCOL) /* SORT BY AVERAGE */
      CALL CPRINT(8)

100   CONTINUE
```

```
      /* PRINT OVERALL STATISTICS */
      WRITE(6,110)(GRHEAD(1,K),GRHEAD(2,K),K=1,AVGCOL)
110   FORMAT(1H1,40X,'COMPREHENSIVE STATISTICS',///,47X,14(1X,2A2))


      DO 300 I = 1,ISEC
      IF (NSEC(I).EQ.0) GOTO 300
      WRITE(6,210) LINE1,I,LINE2,(MASAVG(I,K),K=1,AVGCOL)
210   FORMAT(/,11X,4A2,I1,2X,12A2,14I5)
      WRITE(6,220) LINE3,(MASPCT(I,K),K=1,AVGCOL)
220   FORMAT(/,22X,12A2,14I5)
300   CONTINUE


      /* THIS HANDLES THE UNASSIGNED SECTION */
      IF (NSEC(TSEC).EQ.0) GOTO 400
      WRITE(6,310) LINE2,(MASAVG(TSEC,K),K=1,AVGCOL)
310   FORMAT(/,11X,'UNASSIGNED ',12A2,14I5)
      WRITE(6,220) LINE3,(MASPCT(TSEC,K),K=1,AVGCOL)


400   CONTINUE
      WRITE(6,410) LINE2,(MASAVG(CSEC,K),K=1,AVGCOL)
410   FORMAT(///,11X,'OVERALL     ',12A2,14I5)
      WRITE(6,220) LINE3,(MASPCT(CSEC,K),K=1,AVGCOL)


      WRITE(6,420) GRADED(CSEC),PSFAIL(CSEC),
     *             LISTEN(CSEC),DROP(CSEC)
420   FORMAT(/////,21X,'STATUS',10X,'STUDENT COUNT',///
     * 21X,'LETTER GRADE',  9X,I3,//
     * 21X,'PASS-FAIL',    12X,I3,//
     * 21X,'LISTENER',     13X,I3,//
     * 21X,'DROPPED',      14X,I3)
```

```
      DO 800 I = 1,TSEC

      IF (NSEC(I).EQ.0) GOTO 800 /* SKIP IF NO STUDENTS */

      IF (.NOT.SLIST(1)) GOTO 500
      CALL SSORT(13,36,0,I) /* SORT BY NAME */
      CALL SPRINT(4,I)

500   CONTINUE
      IF (.NOT.SLIST(2)) GOTO 600
      CALL SSORT(0,0,IGRADE,I) /* SORT BY LATEST GRADE */
      CALL SPRINT(7,I)

600   CONTINUE
      IF (.NOT.SLIST(3)) GOTO 700
      CALL SSORT(0,0,AVGCOL,I) /* SORT BY AVERAGE */
      CALL SPRINT(8,I)

700   CONTINUE
      WRITE(6,710)(GRHEAD(1,K),GRHEAD(2,K),K=1,AVGCOL)
710   FORMAT(1H1,40X,'SECTION STATISTICS',///,47X,14(1X,2A2))
      WRITE(6,210) LINE1,I,LINE2,(MASAVG(I,K),K=1,AVGCOL)
      WRITE(6,220) LINE3,  (MASPCT(I,K),K=1,AVGCOL)


      WRITE(6,420) GRADED(I),PSFAIL(I),
     *                LISTEN(I),DROP(I)


800   CONTINUE
      RETURN
      END
```

INTEGER FUNCTION SUM(ARRAY,DIMEN)

```
      INTEGER FUNCTION SUM(ARRAY,DIMEN)
      INTEGER ARRAY,DIMEN,TOTAL
      DIMENSION ARRAY(DIMEN)


      TOTAL = 0
      DO 100, I = 1,DIMEN
      TOTAL = TOTAL + ARRAY(I)
100   CONTINUE

      SUM = TOTAL

      RETURN
      END
```

INTEGER FUNCTION COLSUM(ARRAY,ROW

```
      INTEGER FUNCTION COLSUM(ARRAY,ROWS,COL)
      INTEGER ARRAY(10,14),ROWS,COL,TOTAL
      TOTAL = 0
      DO 100 I = 1,ROWS
      TOTAL = TOTAL + ARRAY(I,COL)
100   CONTINUE
      COLSUM =TOTAL
      RETURN
      END
```

SUBROUTINE CSORT(BEGIN,ENDING,GRCOL)

```
      SUBROUTINE CSORT(BEGIN,ENDING,GRCOL)
      INTEGER
      INTEGER BEGIN,ENDING,GRCOL
      LOGICAL INCHNG
$INSERT COM1
$INSERT COM2
$INSERT COM3
$INSERT COM4
$INSERT COM5 !
      INCHNG = .TRUE.  /* INITIALIZE TO TRUE FOR FIRST PASS */
      LEN = (ENDING-BEGIN+1)/2
      WORD = (BEGIN-1)/2
      MAX = NCLASS-1
      DO 500 I = 1,MAX
      IF (.NOT.INCHNG) GOTO 600 /* SORT COMPLETE */
      INCHNG = .FALSE.
      MAX1 = NCLASS-I
      DO 400 J = 1,MAX1
      JJ = J+1
      TEMP1 = CLASS(J)
      TEMP2 = CLASS(JJ)
      IF (BEGIN.EQ.0) GOTO 200  /* NUMERIC SORT */
      DO 100 K = 1,LEN
      KK = WORD+K
      IF (STUDC(KK,TEMP1)-STUDC(KK,TEMP2)) 400,100,300
100   CONTINUE
```

```
      GOTO 400 /* NO INTERCHANGE, TRY NEXT ITEM */
200   CONTINUE
      IF (STUDN(GRCOL, TEMP1). LT. STUDN(GRCOL, TEMP2)) GOTO 300
      GOTO 400 /* NO INTERCHANGE, TRY NEXT ITEM */
300   CONTINUE
C     PERFORM INTERCHANGE OF POINTERS
      CLASS(J) = TEMP2
      CLASS(JJ) = TEMP1
      INCHNG = .TRUE.
400   CONTINUE
500   CONTINUE
600   CONTINUE
      RETURN
      END
```

SUBROUTINE SSORT(BEGIN,ENDING,GRCOL

```
      SUBROUTINE SSORT(BEGIN,ENDING,GRCOL,SECTON)
      INTEGER
      INTEGER BEGIN,ENDING,GRCOL,SECTON
      LOGICAL INCHNG
$INSERT COM1
$INSERT COM2
$INSERT COM3
$INSERT COM4
$INSERT COM5 !
      INCHNG = .TRUE.  /* INITIALIZE TO TRUE FOR FIRST PASS */
      LEN = (ENDING-BEGIN+1)/2
      WORD = (BEGIN-1)/2
      MAX = NSEC(SECTON)-1
      DO 500 I = 1,MAX
      IF (.NOT.INCHNG) GOTO 600 /* SORT COMPLETE */
      INCHNG = .FALSE.
      MAX1 = NSEC(SECTON)-I
      DO 400 J = 1,MAX1
      JJ = J+1
      TEMP1 = SEC(J,SECTON)
      TEMP2 = SEC(JJ,SECTON)
      IF (BEGIN.EQ.0) GOTO 200  /* NUMERIC SORT */
      DO 100 K = 1,LEN
      KK = WORD+K
      IF (STUDC(KK,TEMP1)-STUDC(KK,TEMP2)) 400,100,300
100   CONTINUE
```

```
      GOTO 400 /* NO INTERCHANGE, TRY NEXT ITEM */
200   CONTINUE
      IF (STUDN(GRCOL, TEMP1). LT. STUDN(GRCOL, TEMP2)) GOTO 300
      GOTO 400 /* NO INTERCHANGE, TRY NEXT ITEM */
300   CONTINUE
C     PERFORM INTERCHANGE OF POINTERS
      SEC(J, SECTON) = TEMP2
      SEC(JJ, SECTON) = TEMP1
      INCHNG = . TRUE.
400   CONTINUE
500   CONTINUE
600   CONTINUE
      RETURN !
      END
```

SUBROUTINE CPRINT(TYPE)

```
      SUBROUTINE CPRINT(TYPE)
      INTEGER
      INTEGER TYPE
$INSERT COM1 !
$INSERT COM2
$INSERT COM3
$INSERT COM4
$INSERT COM5
      WRITE(6,10)
10    FORMAT(1H1)
      WRITE(6,20) SBHEAD /* SUBJECT HEADING */
20    FORMAT(1H ,40A2//)
      WRITE(6,30) CRHEAD,(RPHEAD(J,TYPE),J=1,5)
30    FORMAT(1H ,15A2//)
      WRITE(6,40) CGHEAD,(GRHEAD(1,J),GRHEAD(2,J),J=1,AVGCOL)
40    FORMAT(7X,20A2,14(1X,2A2))
      WRITE(6,50)
50    FORMAT(/)
      LINES = 7
C
C
```

```
C       /* NOW BEGINS THE PRINT LOOP
        DO 500 I = 1,NCLASS
        IF (LINES.LE.54) GOTO 100
        LINES = 3
        WRITE(6,10)     /* FORCE NEW PAGE */
        WRITE(6,40) CGHEAD,(GRHEAD(1,J),GRHEAD(2,J),J=1,AVGCOL)
        WRITE(6,50)     /* SKIP ONE LINE */
100     CONTINUE
        II = CLASS(I)
        WRITE(6,200) I,(STUDC(J,II),J=1,20),(STUDN(J,II),J=1,AVGCOL)
200     FORMAT(I5,2X,20A2,I4,13I5)
        LINES = LINES + 1
500     CONTINUE
C
C
        RETURN
        END
```

SUBROUTINE SPRINT(TYPE,SECTON)

```
      SUBROUTINE SPRINT(TYPE,SECTON)
      INTEGER
      INTEGER TYPE,SECTON
$INSERT COM1.!
$INSERT COM2
$INSERT COM3
$INSERT COM4
$INSERT COM5
      WRITE(6,10)
10    FORMAT(1H1)
      WRITE(6,20) (SCHEAD(J,SECTON), J=1,40)
20    FORMAT(1H ,40A2//)
      WRITE(6,30) CRHEAD,(RPHEAD(J,TYPE),J=1,5)
30    FORMAT(1H ,15A2//)
      WRITE(6,40) CGHEAD,(GRHEAD(1,J),GRHEAD(2,J),J=1,AVGCOL)
40    FORMAT(7X,20A2,14(1X,2A2))
      WRITE(6,50)
50    FORMAT(/)
      LINES = 7
C
C
```

```
C       /* NOW BEGINS THE PRINT LOOP
        K = NSEC(SECTON)
        DO 500 I = 1,K
        IF (LINES.LE.54) GOTO 100
        LINES = 3
        WRITE(6,10)     /* FORCE NEW PAGE */
        WRITE(6,40) CGHEAD,(GRHEAD(1,J),GRHEAD(2,J),J=1,AVGCOL)
        WRITE(6,50)     /* SKIP ONE LINE */
100     CONTINUE
        II = SEC(I,SECTON)
        WRITE(6,200) I,(STUDC(J,II),J=1,20),(STUDN(J,II),J=1,AVGCOL)
200     FORMAT(I5,2X,20A2,I4,13I5)
        LINES = LINES + 1
500     CONTINUE
C
C
        RETURN
        END
```

BLOCK DATA

```
      BLOCK DATA
      INTEGER
$INSERT COM1
$INSERT COM2
$INSERT COM3
$INSERT COM4
$INSERT COM5
      DATA
     1 INFILE/3*'   '/,
     2 SBHEAD/40*'   '/,
     3 SCHEAD/360*'   ',
     * 'THESE STUDENTS HAVE NOT BEEN ASSIGNED TO ANY SECTION',
     * 19*'   '/,
     4 GRHEAD/28*'   '/,
     5 STUDC/6000*'   '/,
     6 NUMBER/20H 1 2 3 4 5 6 7 8 910/,
     7 PRFILE/'REPORT'/,
     8 CGHEAD/'SEQ#  YR CS  STUDENT                    S SC'/
     9, CRHEAD/'STUDENTS ORDERED BY '/
     *, RPHEAD/
     *  'SEQUENCE #'
     *, 'YEAR       '
     *, 'COURSE     '
     *, 'NAME       '
     *, 'STATUS     '
     *, 'SECTION    '
     *, 'LAST GRADE'
     *, 'AVERAGE    ' /
```

```
 /* INITIALIZE COM2 VARIABLES */
 DATA
1   ISEC/0/
2,  IGRADE/0/
3,  STUDN/4200*0/
4,  WEIGHT/13*0/
5,  NCLASS/0/
6,  NSEC/10*0/
7,  AVGCOL/0/
8,  WTOTAL/0/
9,  TSEC/0/
*,  CSEC/0/
1,  GRADED/11*0/
2,  PSFAIL/11*0/
3,  LISTEN/11*0/
4,  DROP/11*0/
5,  GSECOL/140*0/
6,  NSECOL/140*0/
7,  MASAVG/154*0/
8,  MASPCT/154*0/
 DATA
1 SEC/800*0/
 END
```

COMMOM /COM1/

```
 COMMOM /COM1/
1   INFILE(3)        /* INPUT FILENAME */
2,  SBHEAD(40)       /* SUBJECT HEADING */
3,  SCHEAD(40,10)    /* SECTION HEADINGS */
4,  GRHEAD(2,14)     /* GRADE HEADINGS */
5,  STUDC(20,300)    /* STUDENT CHARACTER INFO */
6,  NUMBER(10)       /* ASCII 1 TO 10 */
7,  PRFILE(3)        /* PRINT (OUTPUT) FILENAME */
8,  CGHEAD(20)       /* COMMON GRADE ROW HEADING */
9,  CRHEAD(10)       /* COMMON REPORT HEADING */
*,  RPHEAD(5,8)       /* REPORT HEADINGS */
```

COMMON /COM2/

```
 COMMON /COM2/
1 ISEC,              /* NUMBER OF SECTIONS */
2 IGRADE,            /* NUMBER OF GRADES PRESENT */
3 STUDN(14,300),     /* STUDENT GRADE INFO */
4 WEIGHT(13),        /* WEIGHT OF GRADE AS A PERCENTAGE */
5 NCLASS,            /* TOTAL STUDENTS IN CLASS */
6 NSEC(10),          /* STUDENTS PER SECTION */
7 AVGCOL           /* IDENTIFIES COLUMN FOR AVERAGE, IGRADE+1 */
8, NTOTAL           /* SUM OF WEIGHTS */
9, TSEC           /* ISEC+1, THE UNASSIGNED SECTION */
*, CSEC           /* ISEC+2, REFERS TO CLASS AVG AND PCT */
1, GRADED(11)     /* # OF GRADED STUDENTS PER SECTION */
2, PSFAIL(11)     /* # OF PASS-FAIL STUDENTS PER SECTION */
3, LISTEN(11)     /* # OF LISTENERS PER SECTION */
4, DROP(11)       /* # OF DROPPED STUDENTS PER SECTION */
5, GSECOL(10,14)  /* TOTAL GRADES BY SEC & COL OF CR STUDENTS */
6, NSECOL(10,14)  /* # OF STUDENTS IN ABOVE COUNT */
7, MASAVG(11,14)  /* MASTER AVERAGES BY SECTION AND COLUMN */
8, MASPCT(11,14)  /* MASTER PERCENTAGES OF CREDIT STUDS WHO DID */
```

COMMON /COM3/

```
 COMMON /COM3/
1 RTEMP             /* REAL TEMPORARY */
 REAL RTEMP
```

COMMON /COM4/

```
 COMMON /COM4/
1 OLIST(8),         /* OVERALL LISTING LOGICALS */
2 SLIST(3)          /* SECTIONAL LISTING LOGICALS */
 LOGICAL OLIST,SLIST
```

COMMON /COM5/

```
 COMMON /COM5/
1 CLASS(300),       /* CLASS PTR */
2 SEC(80,10)        /* SECTION PTR */
```

APPENDIX II: SAMPLE OF AN INPUT FILE

Assume a subject is divided in 3 sections. There are 4 grades available at the time of the run. They are problem sets 1 and 2, machine problem 1 and quiz 1. The weights for these grades are 5%, 5%, 10%, and 15% respectively. The user wants class reports ordered by the student names and their average. He also wants section reports ordered by the latest grade, i.e. quiz 1.

The input file should contain the following:

<1 line to describe the subject>

3

<1 line to describe section 1>

<1 line to describe section 2>

<1 line to describe section 3>

4

PS1 5

PS2 5

MP1 10

QZ1 15

FFFTFFFT

FTF

<data cards for students>

Control Card Formats

Subject Statement	1 line of up to 80 characters
section count	col. 1 & 2
Section Statements (one per section)	1 line of up to 80 characters
grade count	col. 1 & 2
Grade Statements (one per grade)	col. 1 thru 3: a 3 character description, e.g. PS1, QZ1, etc. col. 5 thru 7: the grade weighing factor
Listing Statements (two required)	enter a T in the column to generate report, an F to suppress it. The first card controls class sorts. col. 1: by sequence number col. 2: by year col. 3: by course col. 4: by student name col. 5: by status col. 6: by section col. 7: by latest grade col. 8: by overall average The second card controls section sorts. col. 1: by student name col. 2: by latest grade col. 3: by overall average

Data Card Format

col. 1 to 5	Student sequence number
col. 7 to 8	Year or class of student
col. 10 to 12	Course of department
col. 14 to 36	Student name
col. 37	Status: N,L,C, or blank for graded students
col. 40	Section
col. 41 to 43	First grade
. . .	(the grades should be in the same order as the grade statements)
col. 77 to 79	the 13th grade

APPENDIX III: LOADING INSTRUCTIONS

The command file C←LOAD contains the following loading commands. The user invokes it by typing COMINPUT C←LOAD.

```
FILMEM                zeroes out memory
LOAD                  invokes the linking loader
COMMON 47777          sets COMMON loading address
HARDWARE 17           informs loader of CPU hardware configuration
LOAD B←MAIN           loads binary modules
LOAD B←PROCES
LOAD B←OUTPUT
LOAD B←CSORT
LOAD B←SSORT
LOAD B←CPRINT
LOAD B←SPRINT
LOAD B←BLOCKD
SAVE *GRG             saves load module
QUIT                  terminates loader
RESTORE *GRG          restores it in memory
SAVE *GRG 74 47777    now save memory up to COMMON area
COMINPUT TTY          terminates command file
```

The first SAVE is a loader command which sets up the keys, starting address, the locations occupied by the program etc. However, the COMMON area which contains initial values is not saved by this loader SAVE command. Therefore, the load module *GRG is restored to memory,

and a second SAVE with the same name is performed, with the beginning and ending memory locations specified explicitly.