



STUDENT RECORD MANAGEMENT SYSTEM

Submitted to the Graduate School of Computer Science
in partial fulfillment of the requirements for the
degree of:

MASTER OF COMPUTER SCIENCE

BY

GOVIND KRIPLANI
GRADUATE SCHOOL OF COMPUTER SC.
McGILL UNIVERSITY
MARCH 1975

ADVISER: PROF. T. H. MERRETT

ABSTRACT

The Student Record Management System was designed to create and maintain up-to-date students records for the school year and to print current class lists, student schedules and grade reports.

The database designed for the system corresponds to the proposals of the Data Base Task Group of CODASYL. This has been shown by describing the database of the system in terms of Schema and Sub-Schema using Data Description Language and Data Manipulation Language.

The various functions of the system has been cost analysed. For comparing the cost of this system, another system using different approach has been designed, costed and compared. The result of the cost analysis of the two systems indicate that the maintenance of the multi-linked list structured database, such as this system, is much more expensive.

The system has been tested, but it has not been implemented in any school for thorough testing with real data.

A C K N O W L E D G E M E N T

I would like to express my indebtedness to Professor T. H. Merrett, under whose supervision this system was designed, for his guidance and suggestions during all phases of the system.

For the purpose of costing the system, I have used Professor Merrett's unpublished paper on "Cost Analysis Techniques".

Govind Kriplani

TABLE OF CONTENTS

	PAGE
1. INTRODUCTION	3
IMPLEMENTATION OF SRMS - P A R T - I	
1.1 USING THE SYSTEM	7
1.1.1 General	7
1.1.2 Usage Details	15
1.1.2.1 Create Course-file	16
1.1.2.2 Update Course-file	17
1.1.2.3 Create Performance-file	18
1.1.2.4 Create Student-file, store info in Performance-file and link with the Course-file.	20
1.1.2.5 Update Student courses	22
1.1.2.6 Insert Marks or Marks & Grades	23
1.1.2.7 Print Grade Reports	25
1.1.2.8 Print Class Lists and/or Deck for Grades.	26
1.1.2.9 Update address changes	27
1.1.3 System Messages to User	28
1.2 RECORD LAYOUTS AND DESCRIPTIONS	
1.2.1 Introduction	29
1.2.2 Systems flowcharts	31
1.2.3 Considerations for file organization on DASD	33
1.2.4 Record layouts	35
1.3 PROGRAMS AND DESCRIPTIONS	
1.3.1 Introduction	40
1.3.2 Program Logic and flowcharts	42
1.3.3 J.C.L. Procedures	65

TABLE OF CONTENTS

	PAGE
ANALYSIS AND COMPARISON WITH ALTERNATIVE SYSTEMS - P A R T - II	
2.1 DATA BASE TASK GROUP	75
2.1.1 Introduction	75
2.1.2 Principles of DBTG	76
2.1.3 Data Structures	79
2.2 IMPLEMENTATION OF SRMS IN DBTG	83
2.2.1 SRMS's Database	87
2.2.1.1 Schema	87
2.2.1.2 Sub-Schema	91
2.2.1.3 Programs and flowcharts	94
2.2.2 Conclusion	103
2.3 COST ANALYSIS TECHNIQUES	105
2.3.1 Introduction	105
2.3.2 Definitions	106
2.3.3 Access Cost	108
2.3.4 Storage Cost	109
2.3.5 Analysis of sorting	109
2.3.6 SRMS Databank Description	112
2.3.7 I/O Operations	117
2.3.8 Cost of operations using different access methods	123
2.3.8.1 Cost of operations using DAM	124
2.3.8.2 Cost of operations using SAM	130
2.3.9 Conclusion	139
2.4 COMPARISON WITH ALTERNATIVE SYSTEMS	141
2.4.1 McGill University System	141
2.4.2 CASSARMS	143
2.4.3 Conclusion	146
APPENDIX - A SORTING & SEARCHING TECHNIQUES	149
APPENDIX - B PROGRAM LISTINGS	162
.Create Course-file	162
.Update Course-file	167
.Create Performance-file	171
.Create Student-file and link with Course-file and Performance-file	174
.Update Student-file, Course-file and Performance-file	182
.Insert Grades	193
.Grade Reports	199
.Class Lists	204
.Address Changes	210
APPENDIX - C PROGRAM FUNCTIONS	213

1. INTRODUCTION

In any educational environment the number of students keeps on changing. The students normally register for a year, but some leave before completing a semester or completing the full year. The number of courses taken by students also varies. The Professors report to the administration the performance of students in different courses.

In this complex environment the administration has the task of maintaining and putting together all the information required to inform the students of their performance during the semester or the period of their stay in the school. This task has been very time consuming for Professors and school administration. Moreover, it has been found that due to using manual system throughout there has been many errors in all areas. It has been realized that the computers must be used to minimize the errors and reduce the burden on the professors and administration.

The Student Record Management System has been designed to aid educational institutions in maintaining up-to-date student records for the current school year. The system will be able to add or delete courses due to course changes made by the students after the initial registration. It will also produce required reports such

as class lists, transcripts, and current schedules of the students. The system will not schedule the students in classes, but will print out their schedules.

The system uses nine computer programs to maintain the records and print out the reports. The information such as student number, name, address, telephone number, sex, degree, courses taken, performance in each course, is maintained on-line for easy access.

The section "Using the System" is written from the user's point of view and describes when to use the different programs. The section also contains the function of each program and the necessary input required.

The record layouts and description of the files are detailed in section "Record Layouts and Descriptions". In case more reports are required special programs can be written. This section may also be used to find out where the information required is located. The system flowchart shows the different programs and how the databank is maintained in the system.

With the intention of making the maintenance of the programs easy the general flowcharts and the logic of each program is also included.

The proposals of the Data Base Task Group is discussed in depth. The DBTG introduced several concepts, such as, Schema and Sub-Schema. A Schema consists of Data Description Language entries and is a complete description of a database. A Sub-Schema also consists

of data description language entries. It, however, need not describe the entire database but only those areas, sets, records and data items, which are known to one or more specific programs.

The database of the Student Record Management System has been compared with the proposals of the DBTG. It also describes the database of the SRMS using Data Description Language and the Data Manipulation Language. However, the programs have not been run on any DBTG Compiler.

Section "Cost Analysis Techniques" is devoted to analysing the costs of computer operations using SRMS. The cost associated with SRMS is primarily the cost of peripheral device access and peripheral device storage. However, before analyzing the cost of any particular system, such as this, techniques of analyzing the cost of any system have to be developed. Since the databank of this system is organized on DASD using Relative Access Method, another system was also designed to use Sequential Access Method, cost analyzed and compared. The cost of the above two systems is also compared with the cost of a system using tapes instead of DASD.

The last part of the manual discusses a couple of existing Student Record Management Systems. The result shows that the goal of all the SRMSs is the same, maintaining up-to-date student records, even though methods, techniques and data structures may vary.

P A R T - I

IMPLEMENTATION OF SRMS

1.1 USING THE SYSTEM

1.1.1 GENERAL

The Student Record Management System is designed to maintain up-to-date records of the students by adding and/or deleting the courses and printing out certain reports.

The system has nine programs to perform various functions, such as creating Course-file, updating Course-file, creating files with students' information and courses taken, printing class lists, updating the files to effect the course changes made by the students, address changes, inserting grades in the files, and finally printing grade reports.

The programs are written in ANS Cobol and can be used on any computer having ANS Cobol Compiler simply by writing Job Control Statements of that particular computer.

The Section "Schedule of Events" describes the order in which the programs are normally used. However, the order may be changed to meet the requirements of a particular institution.

SCHEDULE OF EVENTS

Every year new files are created for all the students and maintained on a Direct Access Device. At the end of the school year the files are transferred to a tape.

BEFORE REGISTRATION

At this time the Department decides the courses they are going to offer during the school year. These courses are punched according to the layout of card type 01. By using the program "CR.CRSE.FILE" with the data cards the system will set up a file on Disk containing the courses offered by the Department.

However, if the Department decides to add some more courses to the above file later during the school year, they could do so by using the program "UP.CRSE.FILE" with the data punched as described above.

These cards do not have to be in any particular order for creating the initial course file or adding the courses. The programs sort the cards in the required order.

Also at this time, use the program "CR.PERF.FILE" which creates the Performance-file with dummy records. This file is used later after the registration. The program expects a function card which tells the computer the number of dummy records needed in the Performance-file. The function card required is card type F1.

DURING REGISTRATION

At this point the system needs information about the students for two files. One file needs their student-number, student-name, address, sex, degree, level, telephone number, etc. according to the layout of card type 02, and the second file needs information indicating the courses taken by the students. The information is punched according to the card type 03.

ON COMPLETION OF REGISTRATION

When the registration is completed the system expects card types 02 and 03 for the program "CR.STUD.LINK". This information is linked up by the computer with the third file containing courses created before registration. Card types 02 and 03 do not have to be in any particular order. The system sorts the cards and link the three files together.

DURING COURSE CHANGES

Normally, after attending some classes students like to change some courses. To make course changes the system expects one card for each course added or dropped. For the "drop course" the information should be punched according to the card type 04, and the "add course" according to the card type 05.

ON COMPLETION OF COURSE CHANGES

The schools normally have a fixed date after which no course changes are allowed. Till this date the above two card types, 04 and 05 are collected. After the last date for making changes has expired these cards are used with the program "UP.THRE.FILE" to make the desired additions and deletions in their schedules.

BEFORE GRADING

The system has a program to punch out cards for the courses the students are taking, i.e. one card per course. The cards will be punched out sorted in the order of course code, so that the cards can be handed out to the professors for grading without rearranging. Each card contains student number, student name, course code, semester and a character "M" to indicate the marking card. These cards are type 06 and used by the professors for placing the marks earned by the students on their cards. The marks are then punched on the same pre-punched cards.

DURING GRADING

The above cards type 06 are marked by the professors for entering in the system. If a professor wants to enter test marks only, which weighs 25% of the course and has been marked out of 100, he needs to write the weight of the test, i.e. 25 and the marks obtained by the student.

The system can also grade students according to the marks earned by them till then. The program expects a function card - card type F2 which tells the system to add the marks only or add and grade them according to the grading scale provided on the function card.

END OF THE SEMESTER

The files are now up-to-date and contain all the information about the students for the whole semester. They are ready to be used for different kind of reports. The system can print out the following reports.

1. The grade reports for the semester or for the whole year at the end of the second semester. The program expects a Function Card - card type F3 which tells the system the option chosen for printing out grade reports.

2. The class lists can be printed out. In this case also the system provides options to print out class lists for the first semester students or all the students who took this course during the first and the second semester. The option must be specified on the function card - card type F4.

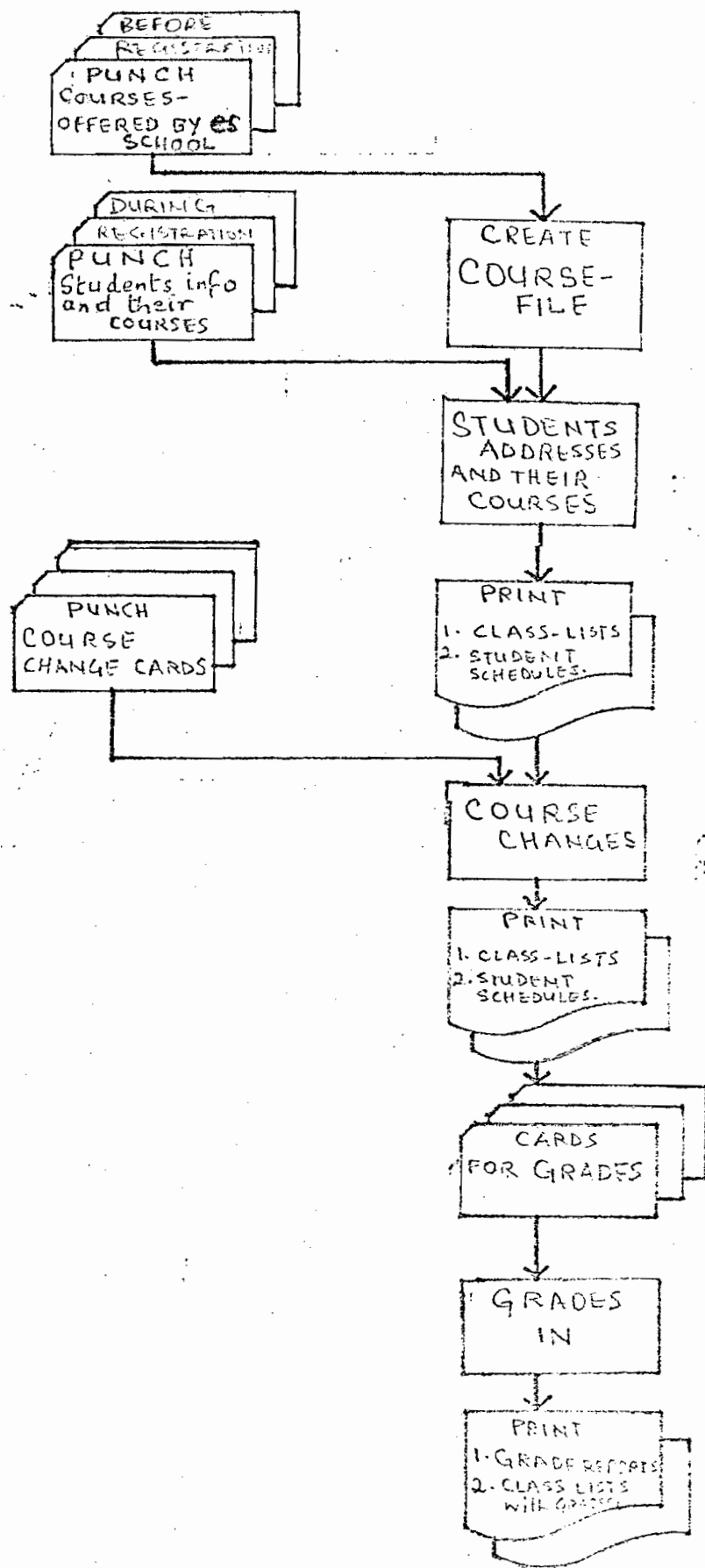
SUMMARY

The system is quite simple and straight forward. It sets up files for the students' information to be used during and at the end of the semester. Normally, all the schools have the requirement of maintaining the students' up-to-date records and printing out certain

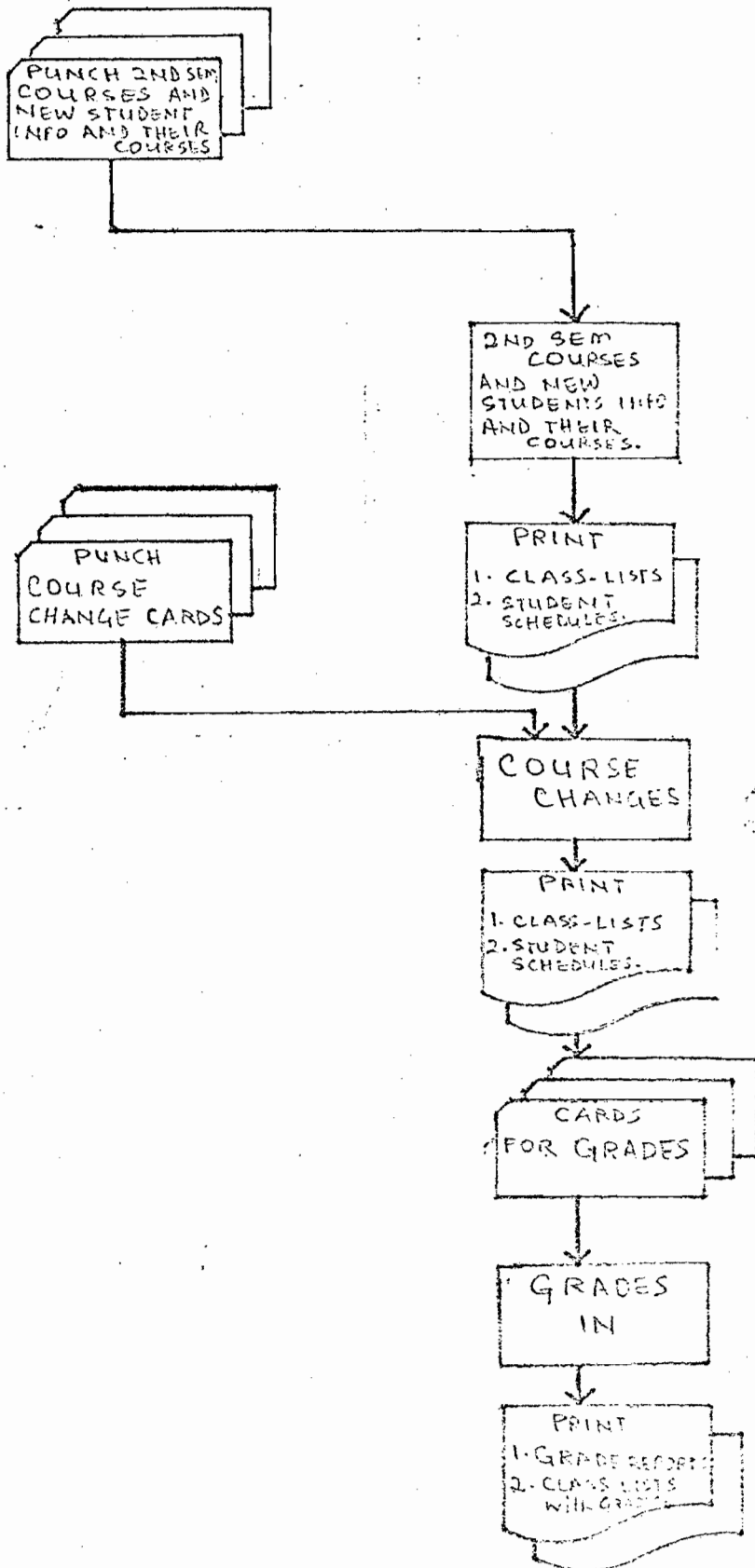
reports, such as grade reports and class lists which is fulfilled by this system. The grade report program can also be used to get the current schedules of the students during any time of the school year. It will look like grade reports but without grades. It indicates the courses a student is taking.

Similarly class lists can be used for several purposes. It could be printed out after registration for the professors to see who and how many students are in their classes. It can also be printed out at the end of the semester for the professors to verify the grades.

The section "Usage Details" contains the layout of all the cards required for the system. The programs are also described in this section.



SECOND SEMESTER



1.1.2 USAGE DETAILS

The table below shows the programs to be used to perform the different functions. The programs have been tested on McGill's IBM 360/75 OS Computer. The JCLs for the programs are written as separate procedures. To execute any program on the above computer only few cards are required which are described under each program separately.

SCHEDULE OF EVENTS	PROGRAM	PAGE No
BEFORE REGISTRATION	CR.CRSE.FILE UP.CRSE.FILE CR.PERF.FILE	
ON COMPLETION OF REGISTRATION	CR.STUD.LINK	
ON COMPLETION OF COURSE CHANGES	UP.THRE.FILE	
BEFORE GRADING	PR.CL.CD	
DURING GRADING	INS.MARK.GRAD	
END OF THE SEMESTER	PR.GR.REP PR.CL.CD	

1.1.2.1 CREATE COURSE-FILE (CR.CRSE.FILE)

This is the program used by the administration to create the Course-file.

Once the department has decided to teach certain courses during the current year they can be punched according to the following format.

CARD TYPE 01

<u>Cols.</u>	<u>Contents</u>	<u>Bytes</u>
4-9	Course-Code	6
10	Semester	1
11-35	Course Description	25
36	Credits	1
80	'S'	1

These are the courses offered by the department from which the students make selection for their schedule. To create the Course-file the following cards are required.

```
//JOB
//          EXEC      CRCSRSEFL
//CCF.CARDS DD *,DCB=BLKSIZE=80
```

CARD TYPE 01

```
/*
```

1.1.2.2 UPDATE COURSE-FILE (UP.CRSE.FILE)

This program is used to add more courses to the Course-file.

In case, after the initial set-up of the Course-file, the department decides to offer some more courses this program can be used with the cards punched according to the following format.

CARD TYPE 01

<u>Cols.</u>	<u>Contents</u>	<u>Bytes</u>
4-9	Course-Code	6
10	Semester	1
11-35	Course Description	25
36	Credits	1
80	'S'	1

The program "UP.CRSE.FILE" is used for adding the courses to the original Course-file. To execute the program the following cards are required.

```
//JOB
//      EXEC      UPCRSEFL
//UCF.CARDS DD *,DCB=BLKSIZE=80
```

CARD TYPE 01

```
/*
```

1.1.2.3 CREATE PERFORMANCE-FILE (CR.PERF.FILE)

This program creates the Performance-file with the dummy records.

The Performance-file is used by the program "CR.STUD.LINK" to store a part of the student information. The program expects the function card - card type F1 which tells the system the number of dummy records required in the Performance-file.

CARD TYPE F1

<u>Cols.</u>	<u>Contents</u>	<u>Bytes</u>
1-6	No. of dummy records	6
80	'A'	1

The number of dummy records for the first semester is equal to the number of students times courses allowed to take per student. For the second semester the number of dummy records is equal to the number of students times the courses allowed to take per student plus the number of courses a student was allowed to take during the first semester.

The Performance-file with the dummy records is created every year before the first semester and before the second semester starts the first semester's Performance-file is transferred to the new Performance-file. The idea behind this is not to block extra space on the Disk. For example, during the first semester a student will have

only five courses in the file and second semester ten, so why create a Performance-file with ten dummy records when only five will be used.

IBM Utility Program "IEHCOPY" can be used to transfer the first semester's Performance-file to the new Performance-file.

To create the Performance-file for the first semester the following cards are required. .

```
//JOB  
//          EXEC  CRPERFF  
//CRP.CARD  DD  *,DCB=BLKSIZE=80
```

CARD TYPE F1

```
/*
```

1.1.2.4 CREATE STUDENT-FILE, STORE INFO IN PERFORMANCE-
FILE AND LINK THEM WITH THE COURSE-FILE
(CR.STUD.LINK)

After the completion of the registration the card types 02 and 03 are used with this program to store students' information in the Student-file and Performance-file, and link the two files with the Course-file.

CARD TYPE 02

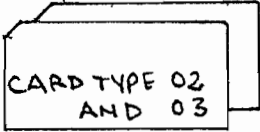
<u>Cols.</u>	<u>Contents</u>	<u>Bytes</u>
1-3	Department	3
4-9	Student Number	6
10-29	Student Name	20
30-33	Degree	4
34	Level	1
35	Sex	1
36-52	Number and Street	17
53-56	Apartment Number	4
57-72	Town	16
73-79	Telephone Number	7
80	'F'	1

CARD TYPE 03

4-9	Student Number	6
10-11	Current Year	2
12	Semester	1
16-21	Course Code	6
80	'G'	1

The data cards do not have to be in any sorted order. To execute the program the following cards are required.

```
//JOB  
//  
//          EXEC      CRSTLINK  
//CSL.CARDS DD *,DCB=BLKSIZE=80
```



CARD TYPE 02
AND 03

/*

1.1.2.5 UPDATE STUDENT COURSES (UP.THRE.FILE)

The program is used to update the files after the students have made changes in their courses.

The program expects only two type of cards, card type 04 and 05 one for each deletion or addition of courses.

CARD TYPE 04

<u>Cols.</u>	<u>Contents</u>	<u>Bytes</u>
4-9	Student Number	6
10-11	Current Year	2
12	Semester	1
16-21	Course Code	6
80	'A'	1

CARD TYPE 05

4-9	Student Number	6
10-11	Current Year	2
12	Semester	1
16-21	Course Code	6
80	'D'	1

The cards do not have to be in any sorted order.
To execute the program the following cards are required.

```
//JOB
//      EXEC      UPTHREFL
//UTP CARDS DD *,DCB=BLKSIZE=80
```

CARD TYPE 04
AND 05

```
/*
```

1.1.2.6 INSERT MARKS OR MARKS AND GRADES (INS.MARK.GRAD)

The program will add marks in the student records. The marks and the percentage of the marks is punched on the card type P6. The cards are pre-punched and needs only two numbers to be punched.

It can also add marks and grade the students according to the Grading-Scale specified on the Function Card - card type F2. The Function Card also contains the length of the Grading-Scale.

CARD TYPE P6

<u>Cols.</u>	<u>Contents</u>	<u>Bytes</u>
1-3	Percentage	3
44-46	Marks	3

If the marks for an assignment are being entered and it weighs 33% of the course marks, 33 is punched in percentage field and the marks obtained in marks field. It is assumed that marking is done on a scale of 100.

CARD TYPE F2

1-3	'YES' if grades to be given or ' NO' if grades not to be given.	3
4-5	Marks obtained - Grading Scale.	2
78-79	Length of the above Grading Scale.	2
80	'A'	1

The following is an example of the Function Card with the Grading Scale option.

YES80A65B50C

3A

This means if a student earned 80 or over gets A, 65 or over gets B and so on. The number of the grades in the above example is three and therefore 3 punched in columns 78-79, right adjusted. Column 80 contains "A" to identify the Function Card. In the absence of the Function Card a message will be printed out and the program will stop execution.

To execute the program the following cards are required.

```
//JOB
//          EXEC      INSMRKGR
//IMG.CARDS DD  *,DCB=BLKSIZE=80
```

CARD TYPE F2

/*

1.1.2.7 PRINT GRADE REPORTS (PR.GR.REP)

The program prints the Grade Reports of the students for the first semester and/or the second semester.

It expects a Function Card - card type F3 which indicates the option chosen for printing the Grade Reports. In the absence of the Function Card it will print out a message and stop the execution.

CARD TYPE F3

<u>Cols.</u>	<u>Contents</u>	<u>Bytes</u>
1-2	Year - for example "72"	2
3	Semester	1
80	"A"	1

The Function Card is the only data card with the program which will produce the required Grade Reports. To execute the program the following cards are required.

```
//JOB
//          EXEC    PRGRREC
//PGR.CARDS DD  *,DCB=BLKSIZE=80
```

CARD TYPE F3

```
/*
```

1.1.2.8 PRINT CLASS LISTS AND/OR DECK FOR GRADES (PR.CL.CD)

The program will print out class lists and/or punch cards - type P6. These cards are meant for the professors to place marks on them. It expects the Function Card - card type F4 which indicates the option selected.

CARD TYPE F4

<u>Cols.</u>	<u>Contents</u>	<u>Bytes</u>
1-2	'AL' - class lists with all the students needed.	2
or	'74' - class lists of the students of class 74 needed.	
or	'NO' - class lists not needed.	
3	Semester	1
4-5	'OO' - Card type P6 not needed.	2
or	'74' - Cards for the students of class 74 needed	
6	Semester - for cards.	
80	'A'	1

The Function Card is the only data card for the program which indicates the option for the output. To execute the program the following cards are required.

```
//JOB
//      EXEC      PRCLCD
//PCD.CARDS      DD      *,DCB=BLKSIZE=80
```

CARDTYPE F4

```
/*
```

1.1.2.9 UPDATE ADDRESS CHANGES (UP.STUD.FILE)

The program updates the students' addresses. It expects one address change card - card type 06 for each student whose address is to be changed. The cards must be punched according to the format given below.

CARD TYPE 06

<u>Cols.</u>	<u>Contents</u>	<u>Bytes</u>
1-6	Student Number	6
7-26	Student Name	20
27-30	Degree	4
31-32	Level	2
33	Sex	1
34-50	Number & Street	17
51-54	Apartment Number	4
55-70	Town	16
71-77	Telephone Number	7
80	'C'	1

To execute the program the following cards are required.

```
//JOB
//      EXEC      UPSTUDFL
//USF.CARDS DD  *,DCB=BLKSIZE=80
```

CARDTYPE F6

```
/*
```

1.1.3 SYSTEM MESSAGES TO USER

ASKED TO ADD - ALREADY TAKING

CHECK COL - 80

COURSE NOT FOUND

CHECK FUNCTION CARD

FUNCTION CARD MISSING

NO MORE DUMMY RECORDS

NO. OF DUMMY RECORDS CREATED XXXXXX

RECORD NOT WRITTEN

SEARCH UNSUCCESSFUL

SORT UNSUCCESSFUL

STUDENT NOT FOUND

STUDENT NOT TAKING THIS COURSE- NOT DELETED

STUDENT NOT IN THIS CLASS

1.2 RECORD LAYOUTS AND DESCRIPTIONS

1.2.1 INTRODUCTION

The system is designed to maintain up-to-date records of the students by adding, deleting and updating the files. It can also print class lists, current schedules and grade reports.

The system has the following three files and two directories:

The Student-file contains the students' information such as name, address, sex, telephone number, degree, level and student number. There is a Student-file-directory used by the system to access the students' records directly.

The Course-file contains the courses offered by the school. The Course-file has a Course-file-directory which enables the system to access any course record directly.

The Performance-file contains information such as semester, year, marks and grade earned by students. The number of performance records for a student is equal to the number of the courses he is enrolled in.

All the three files of the system, Student-file, Course-file and Performance-file are organized on a Disk. The files are linked with each other by pointers which are addresses of the connecting records in the other files. This is shown in the Figure "SRMSS Data Structure".

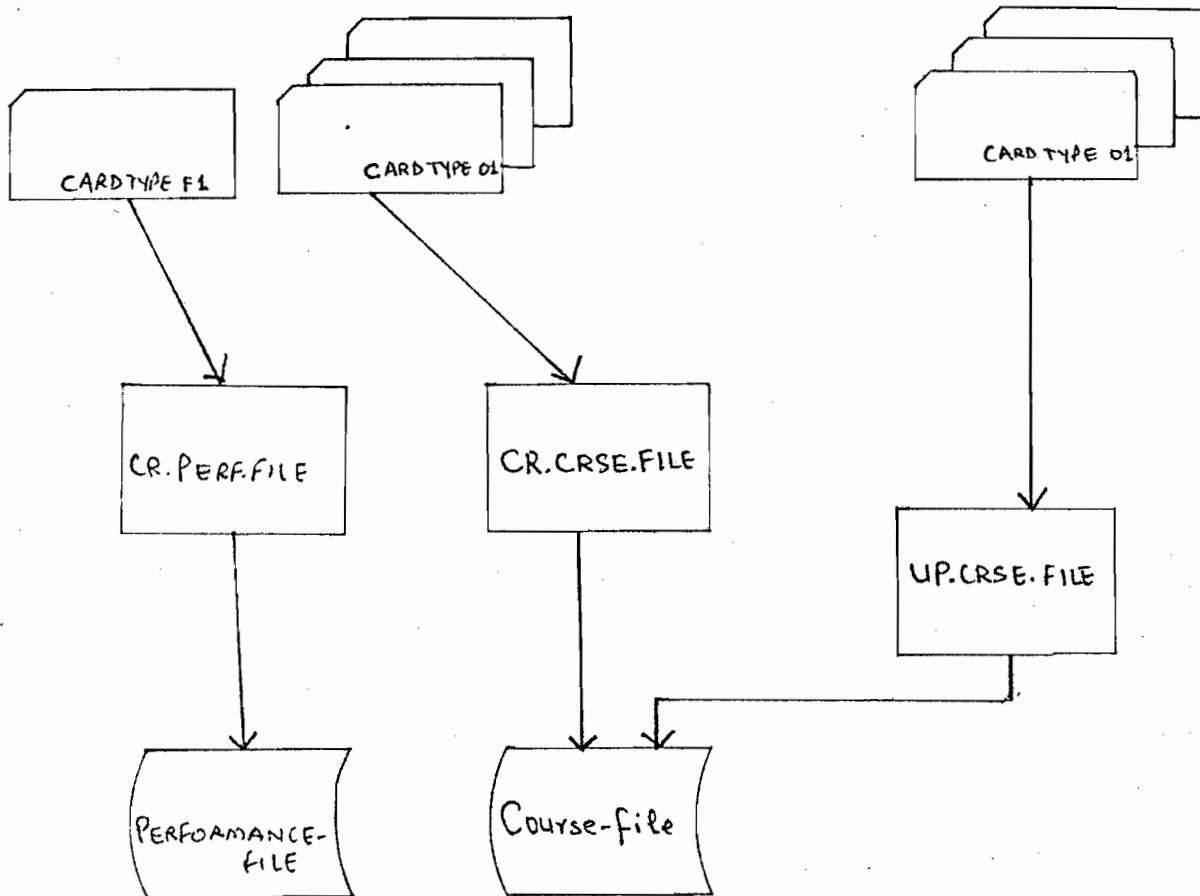
The records in the files are accessed either by Student number or Course code which are in Student-file and Course-file.

No searching is required in the Performance-file, because the pointers of these records are stored in records of the Student-file, Course-file or even Performance-file. To reduce the searching time to minimum the above two directories for the Student-file and the Course-file are maintained on a DASD. The layouts of the three files and the two directories are given in this section.

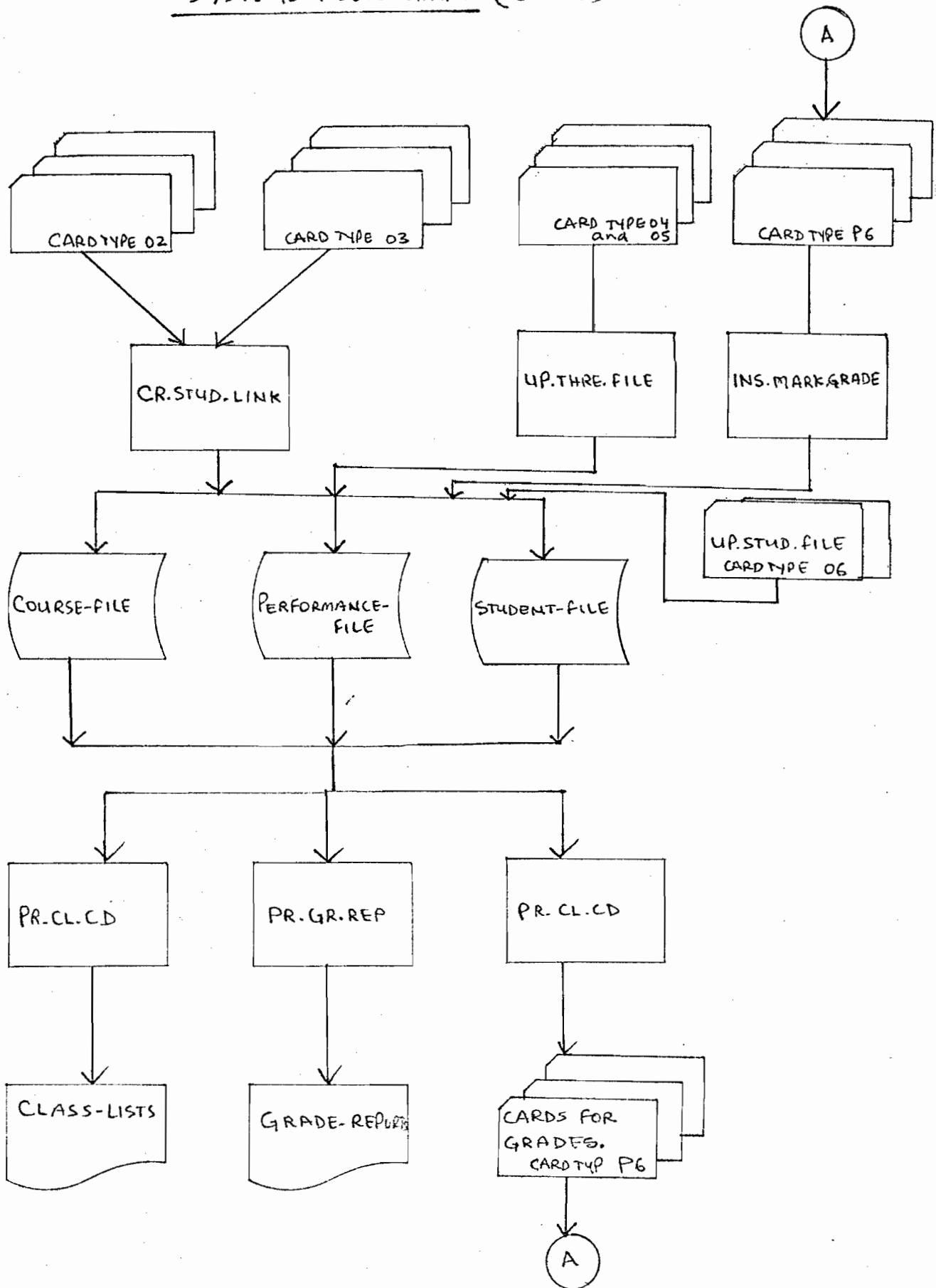
The system consists of nine programs to perform the various functions. All the programs are listed in the section "Usage Details". The system flowchart shows the functions of the different programs. For the purpose of sorting records Sort verb of ANS Cobol is coded in the programs.

The Job Control Language for each program is written as a separate procedure. To execute any program of the system on IBM 360 OS Computers only couple of statements are required which are described under each program.

1.2.2

SYSTEMS FLOW CHART

SYSTEMS FLOW CHART (CONT)



1.2.3 CONSIDERATIONS FOR FILE ORGANIZATION ON DIRECT ACCESS DEVICE

The system needs to have all the three files on a direct access device so that they are easily accessible for using or updating. There are many considerations before it could be decided whether the files should be organized sequential, relative, index sequential or direct access, which depends on the use of that particular file.

COURSE-FILE

The Course-file contains the courses offered by the school. The usage of the file is as follows:

1. We may want to add new courses to the file. However, this will be done only once a semester. For this purpose the courses can be sorted added even in a file organized sequentially.

2. We will use this file to link the Student-file and Performance-file with it which will be done two times every semester, first time when the students register for the courses and the second time when they make changes in their courses. For this purpose we need to access records from the Course-file, store the address of performance record in it and store the record back in its original position. This is only possible if the Course-file is organized index sequential, relative or direct access so that any record can be accessed and stored back after updating the pointers.

3. The system will use this file to insert the grades and marks earned by the students. The grades will be entered about three times a semester, i.e. six times a year. For this purpose also the Course-file should be organized by index sequential, relative or direct access method.

4. When the grade reports or class lists are printed the system accesses the Course-file randomly, which also requires the Course-file to be organized by index sequential, relative or direct access method.

After considering the above factors it is desirable to have the Course-file organized by relative method on a Direct Access Storage Device.

Initially the records in the Course-file will be sorted but when the additions are made the new records will be stored at the end in the Course-file.

STUDENT-FILE AND PERFORMANCE-FILE

The above two files should also be organized by relative access method on a DASD since they will also be accessed in the same way as the Course-file.

The two directories for the Student-file and the Course-file are accessed sequentially and therefore, stored by sequential access method.

1.2.4 RECORD LAYOUTS

STUDENT-FILE

<u>Cols</u>	<u>Contents</u>	<u>Bytes</u>
1 - 3	Department	3
4 - 9	Student Number	6
10 - 29	Student Name	20
30 - 33	Degree	4
34 - 35	Level	2
36	Sex	1
37 - 53	Number & Street	17
54 - 57	Apartment Number	4
58 - 73	Town	16
74 - 80	Telephone Number	7
81 - 84	Address of a record in the Performance- file.	4

84

STUDENT-FILE-DIRECTORY

<u>Cols.</u>	<u>Contents</u>	<u>Bytes</u>
1 - 6	Student Number	6
7 - 12	Address of a record in the student-file.	6
.		.
.		.
.		.
.		.
989 - 994	Student Number	6
995 - 1000	Address of a record in the Student-file.	6

 12,000

PERFORMANCE-FILE

<u>Cols.</u>	<u>Contents</u>	<u>Bytes</u>
1 - 4	Address of a record in the Student-file.	4
5 - 8	Address of a record in the Performance- file for the next student in the same class.	4
9 - 10	Year.	2
11	Semester.	1
12	Grade obtained.	1
13 - 14	Percentage.	2
15 - 18	Address of a record in the Performance- file for the next course of the same student.	4
19 - 22	Address of a record in the Course-file.	4

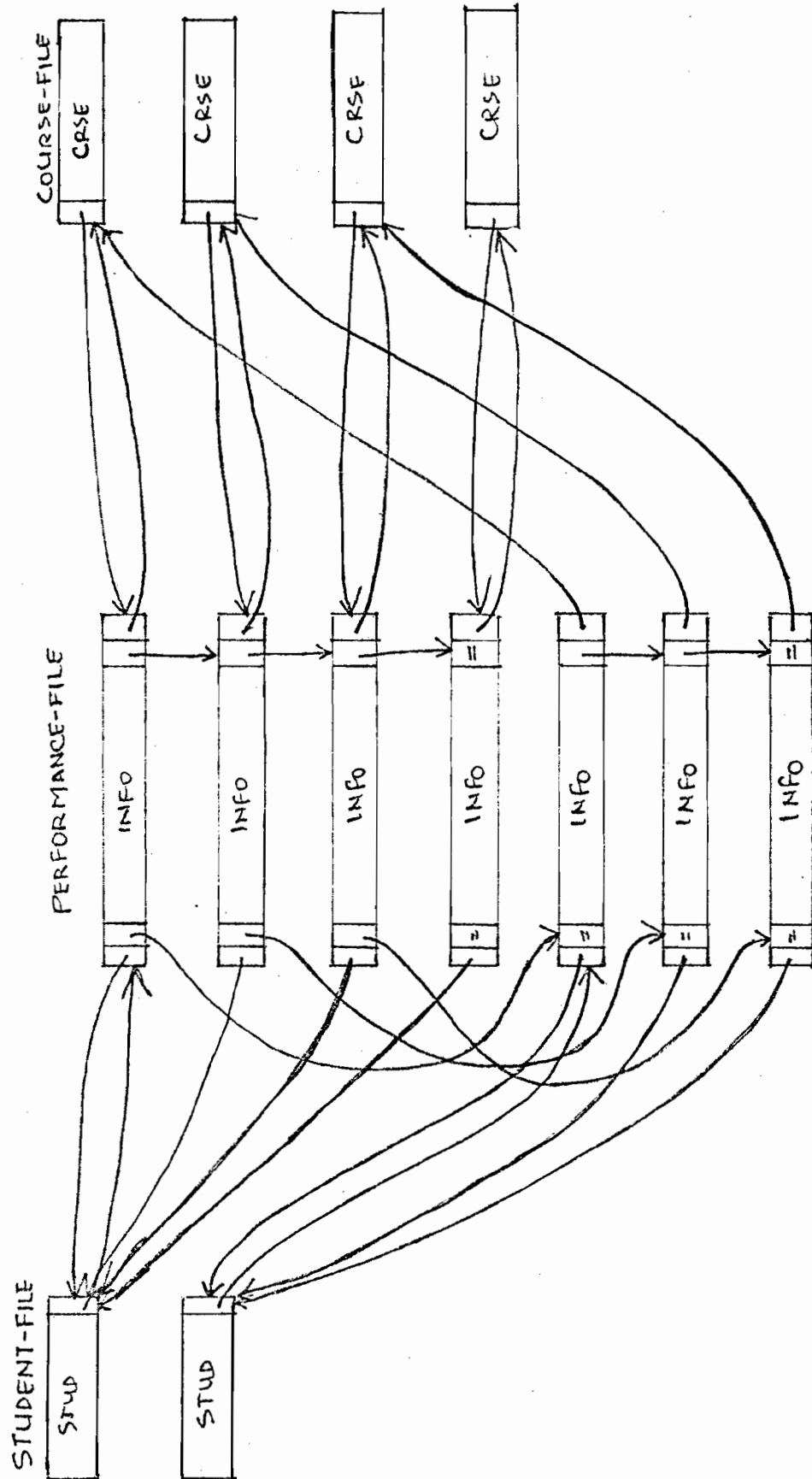
 22

COURSE-FILE

<u>Cols.</u>	<u>Contents</u>	<u>Bytes</u>
1 - 4	Address of a record in the Performance-file for the first student in this class.	4
5 - 10	Course Code	6
11	Semester Code	1
12 -36	Course Description	25
37	Number of Credits	1
		<hr/> 37 <hr/>

COURSE-FILE-DIRECTORY

1 - 6	Course Code	6
7 - 12	Address of a record in the Course-file.	6
.		.
.		.
.		.
.		.
189 - 194	Course Code	6
195 - 200	Address of a record in the Course-file.	6
		<hr/> 2,400 <hr/>



THREE FILES LINKED TOGETHER

1.3 PROGRAMS & DESCRIPTIONS

1.3.1. INTRODUCTION

There are nine programs in the system to create the files, update them and print out reports. The function of each program is described separately with the necessary input data and output.

This section also contains the documentation and a general flowchart for each program. The programs are written in ANS Cobol using list processing techniques. For the purpose of sorting and searching the verbs Sort and Search of ANS Cobol are used.

The programs have been tested on IBM 360/75 OS Computer. The procedures for JCL are written which can be cataloged and jobs executed by calling the procedures.

The database consists of three major files, Student-file, Performance-file and Course-file, and two directories. The two directories are for the Course-file and the Student-file sorted on Course code and Student number respectively. The directories are stored as two separate files.

The Student-file contains information, such as student number, name, degree, level, sex, street, apartment number, town and telephone number.

The Course-file contains information, such as course code, semester, course description and credits. This is the file of the courses offered by the school.

New courses can be added to this file.

The Performance-file contains information, such as the year, semester, grade earned, and the marks obtained. If a student takes five courses he will have five records of this type but only one record in the Student-file.

At the beginning of the school year the files are set-up on a DASD. The files are kept up-to-date, by adding/deleting the courses, entering marks for tests/assignments, grading the students, and finally printing out the grade reports.

1.3.3 PROGRAM DESCRIPTIONS AND FLOWCHARTS

CREATE COURSE-FILE & DIRECTORY (CR.CRSE.FILE)

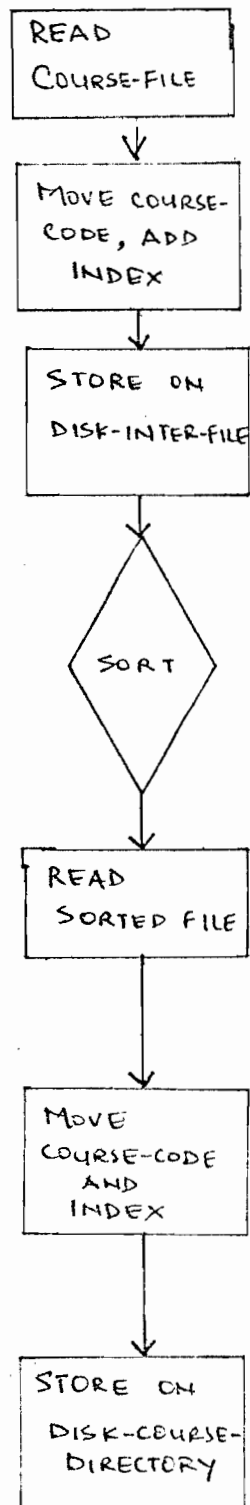
This program creates the Course-file and its directory. The initial Course-file is in sorted order.

Normally course-code-field has the course-code, but if the course-code-field has zeros in it, indicates the first dummy record and new courses can be added from hereon. These dummy records are at the end of the file.

In the directory there are two fields for each course, i.e. course-code in sorted order and a pointer for the course-record in the Course-file. However, after all the pointers in the directory the next address field contains zeros to indicate the end of the pointers in the directory.

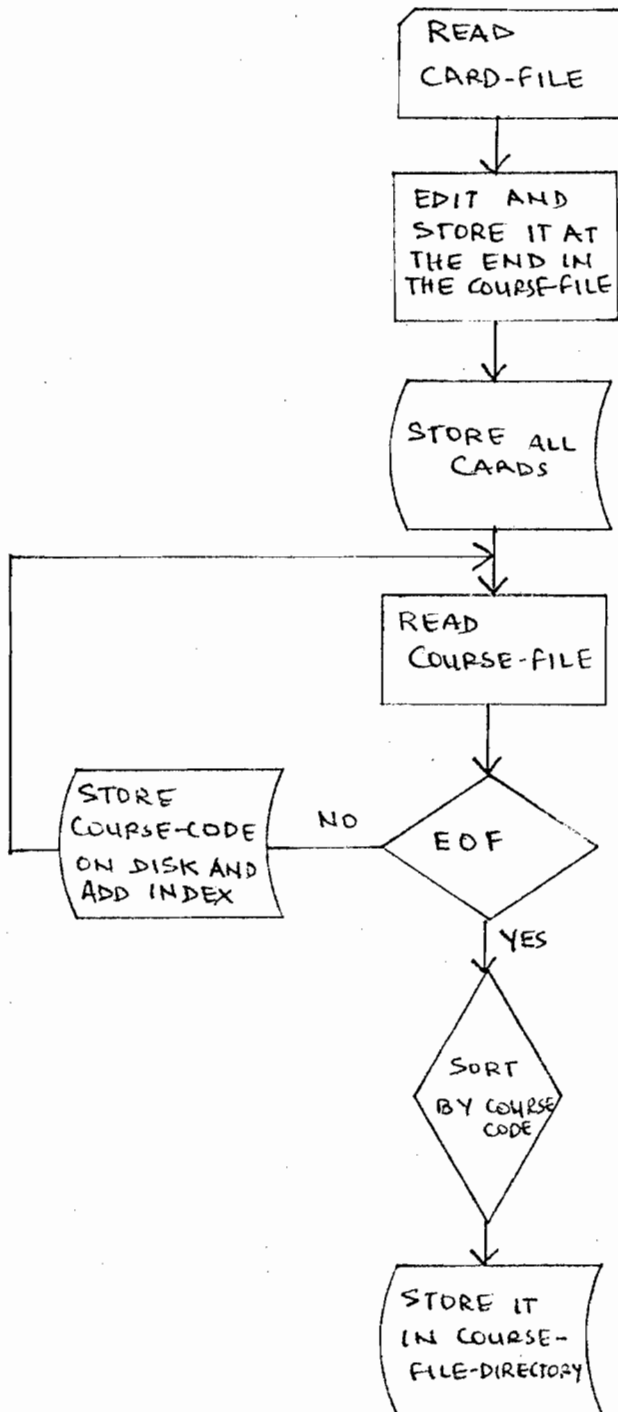
To create the Course-file cards are read and stored in a temporary Disk-file. These cards should contain 'S' in Col-80 otherwise they will be rejected and not stored in the Disk-file. A message saying 'wrong card' will be printed out. The Disk-file is then sorted and stored in a relative file called Course-file. The records in the Course-file are stored relative to 1. At the same time it also stores course-code with a pointer starting from 1 in the memory and the address field is incremented

by 1 every time a new course-code is stored. This points to the record in the Course-file. After all the courses are stored in the Course-file their course-codes and addresses are also stored in a sequential file as a single record. This is the Course-file-directory.

COURSE-FILE-DIRECTORY

UPDATE COURSE-FILE (UP.CRSE.FILE)

This program adds new courses to the Course-file in the end of the file. It also stores the course-codes and their keys, i.e. pointers where they are stored in the Course-file, in the Course-file-directory. The Course-file-directory is moved in memory and while the courses are being added to the Course-file, the course-code and its address is being printed out and also stored in the Course-file-directory in the memory. When all the courses are added the Course-file-directory is sorted on ascending key using course-code and stored back on a Disk as a single record. The first available slot in the Course-file-directory is indicated by zeros in the address field. After updating the Course-file-directory again zeros are left in the address field for future updates. Similarly the first available dummy record in the Course-file contains zeros in the course-code field.

UPDATE COURSE-FILE

CREATE PERFORMANCE-FILE WITH DUMMY RECORDS
(CR.PERF.FILE)

The program creates the Performance-file with dummy records. The file is used by the system for storing students' course performance.

The program reads the card-file and the only data card is the function card which indicates the number of dummy records required in the file. The number is moved to the nominal-key of the Performance-file. Now just by saying once 'write Stud-work-rec from new-rec, it creates the required number of dummy records. The new-rec area contains zeros and therefore all the dummy records contain zeros.

CREATE STUDENT-FILE, ITS DIRECTORY, SORE INFO IN
PERFORMANCE-FILE AND LINK THEM WITH THE COURSE-FILE
(CR.STUD.LINK)

This program creates the Student-file with information like Student number, Name, Address, Telephone number, etc., stores information such as Grade, Marks, year, in the Performance-file and links them to the Course-file.

The program reads the Card-file and stores all the cards containing 'F' and 'G' in Col-80 in a temporary Disk-file. The letter 'F' indicates card meant for the Student-file and 'G' for the Performance-file. If col-80 contains something else the card is printed out with a message to check the card. It is probably the wrong card. The Disk-file is sorted before processing. If the record read is F-type the student information is moved in memory and next record is read. If the record belongs to the same student, which is determined from the Student number, it must be G-type record because the Disk-file is sorted so that any student's 'F' record will be the first and 'G' records follow it. Since it is 'G' type record the information is moved in memory for the Performance-file. Thus all the records read after 'F' type card having the same student number must be G-type records. When all the cards of a student have been moved in memory, i.e. when student number changes, F-type record is stored in the Student-file and G-type in the Performance-file.

The address of the first Performance-file-record is stored in the Student-file-record. There is a variable in

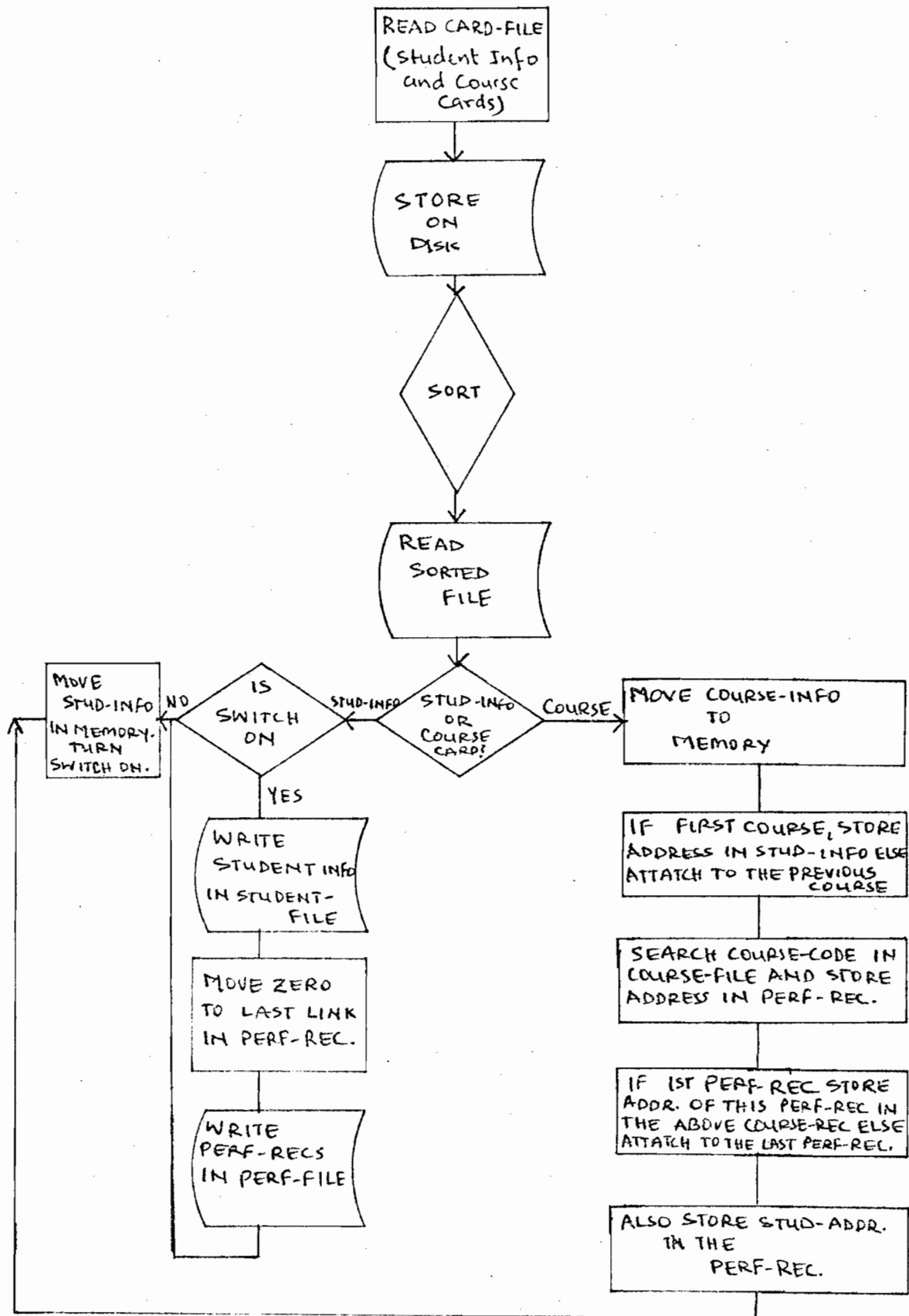
memory which contains zeros to start with and also zeros moved in it when the student number changes. Hence when the first G-type record is read this variable contains zeros and the address of the Performance-file-record is stored in the Student-file-record and in this variable, which is a pointer to the next Performance-file-record and this record should contain pointer to the next Performance-file-record.

Similarly, there is an array corresponding to the Course-file-directory, which contains zeros initially to indicate that the courses in the Course-file are not pointing to any Performance-file-record. The first Performance-file-records address is stored in the Course-file-record and when it is done so, the corresponding element of the above array also contains this address. Next time when a Performance-file-record is added the corresponding element of the array is checked. This time it contains an address of a Performance-file-record. Hence this address is stored in the Performance-file-record.

The first four bytes of the record number 0 in the Performance-file contain an address of the first available record in the Performance-file. Also the first four bytes of the first available record contains zeros.

The Student number field of the first available record in the Student-file contains zeros. Similarly, the course code field of the first available record in the Course-file contains zeros.

CREATE STUDENT-FILE, AND LINK WITH PERFORMANCE-FILE AND COURSE-FILE



UPDATE STUDENT-FILE, COURSE-FILE AND PERFORMANCE-FILE -
ADDITIONS & DELETION OF COURSES (UP.THRE.FILE)

The program updates the three files Course-file, Student-file and the Performance-file. Updating means rearranging the links so as to have the pointers to the student's current courses. The program is used when the students change their courses which is normally done once a semester. The program disconnects the links from the courses the students do not want to take and connects with the one they want to add. The deletion and addition of courses are punched on card type 04 and 05. The addition is made in the sequential order, in two directions i.e. student's courses and class list.

The cards meant for deleting courses contain 'D' in col-80 and the one meant for addition contains 'A' in col-80. The Card-file is read and the one containing 'A' or 'D' are stored in a temporary Disk-file and others are printed out with a message to check the cards. The Disk-file is sorted major field being student-number in ascending order and minor field col-80 in decending order. Thus a student's deletion card will be first and then addition cards, if any.

The records are added or deleted from the Performance-file but in the Course-file and Student-file only the links are changed. When a student drops a course a record is released in the Performance-file and is now available for use. Ther is an array in momory where the address of

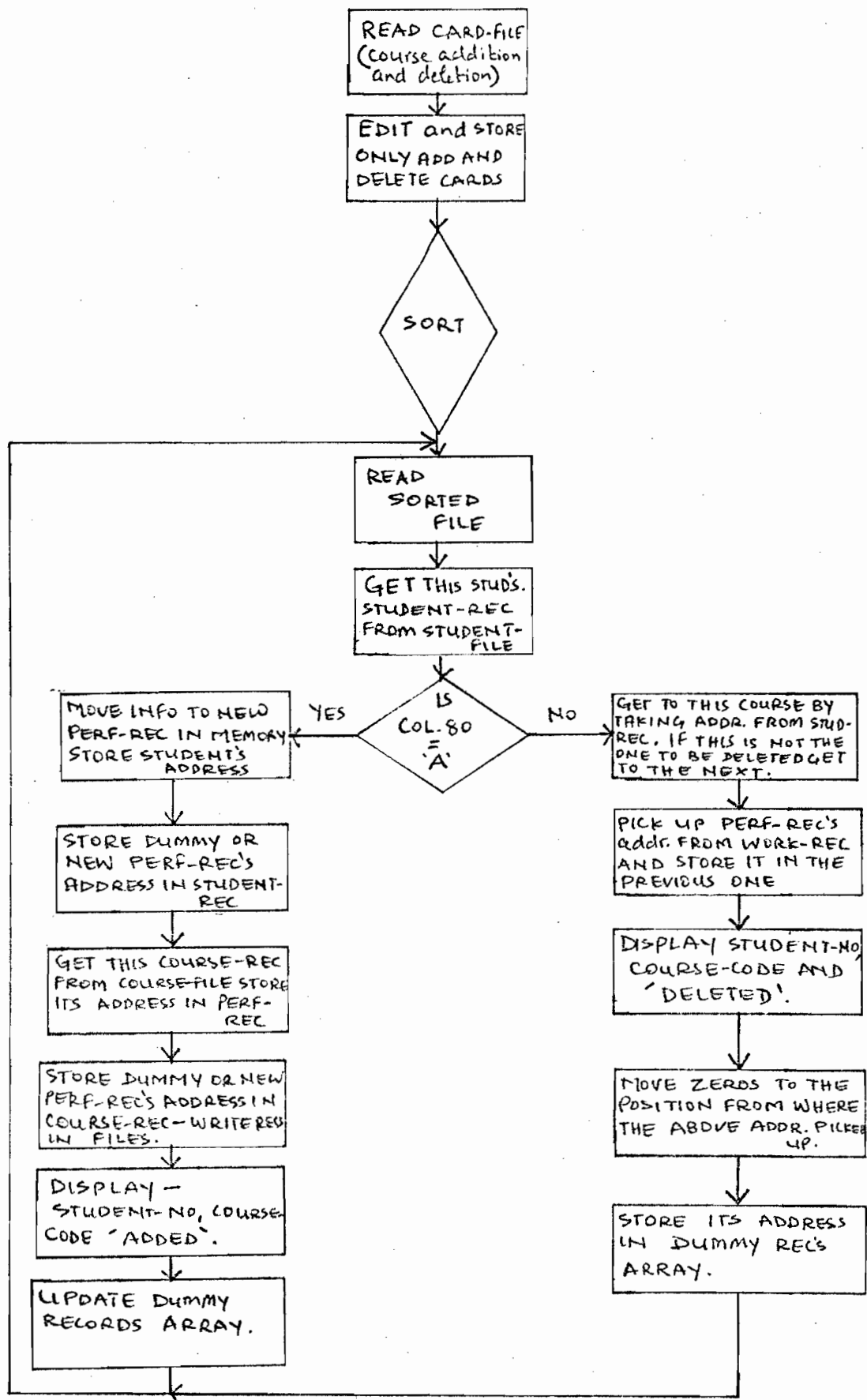
the records, which are available due to deletions, are stored. Now when an addition record arrives the program checks this array to see if there is a pointer to any available record for use. Thus this array never fills up. By deletion addresses are stored in it and additions take them away. However, when there is no address in it the program picks up the address of the next available record, which is in the end of the Performance-file. This address is stored in the record number 0 of the Performance-file by the previous program. At the end of this program the updated address of the next available record in the Performance-file is stored back in the record number 0 for updating files in future.

For deletion, the particular student's records from the Performance-file are read in memory one by one, checking for the one to be deleted. When the record is found the links are changed in the preceding and the following records and written back in the Performance-file. This is done in two directions, i.e. student's courses and class list. Also the address of the free record is stored in the above array.

Similarly, for addition records from the Performance-file are read in memory checking for where the new record fits in sequentially. When found links are changed in the preceding and the following records and written back in the Performance-file. The address of the new record is picked up from the array, if there is any, otherwise address of the next available record is picked from the

record number 0 in the Performance-file. This linking is also done in both the directions, student's courses and class list.

UPDATE FILES (ADD AND DELETE COURSES)



INSERT MARKS OR MARKS AND GRADES (INS.MARK.GRADE)

The program adds the marks given by the Professors in the students' records or adds and grade them. The Function Card indicates the option selected. The Function Card is suppose to be the first data card.

The program reads the Card-file and encounters the Function Card which it holds in memory till the end of the execution. In case, it is not the first card it will print out a message and stop further execution. Other data cards which are only M-type cards, are stored in a temporary Disk-file and at the end sorted on ascending keys, student-number being major field and course-code minor.

The Student-file-directory is read in memory to access the students' records directly.

The sorted Disk-file is now read, the student's fetched from the Student-file-directory and the record is accessed from the Student-file. This record contains the address of his first performance-record which is read. The performance-record contains the address of the course-record which is also read.

Now the course-code is compared with the course-code in M-type card and if it agrees the marks given by the Professor are added to the performance-record. If marking and grading option is selected, the program will check the Grading-Scale given through the function card and grade the student. If the course-code does not agree there could either of the two reasons given below:

1. If the course-code of M-type card is smaller than the course-code of the course-record means the student is not taking this course - the card is erroneous. A message to this effect is printed out.

2. If the course-code of M-type card is bigger than the course-code of the course-record next performance-record is read, whose address is also in the performance-record just read. From this second Performance-record the address of the course-record is picked up and read. The course-code of the course-record and M-type card are compared and the above procedure is repeated.

The procedure is repeated to handle all the M-type cards.

PRINT GRADE REPORTS (PR.GR.REP)

The program prints out students' Grade Reports. It can print out Grade Reports for a semester or for the whole year. The option is indicated by the Function Card.

The program reads the Card-file and expects the Function Card to be the first and the only card. It holds the function card in memory till the end of the execution. If zeros are punched instead of the year, it means Grade Reports for the whole year are required. In case, the first card is not the Function Card it prints out a message to this effect and stops further execution.

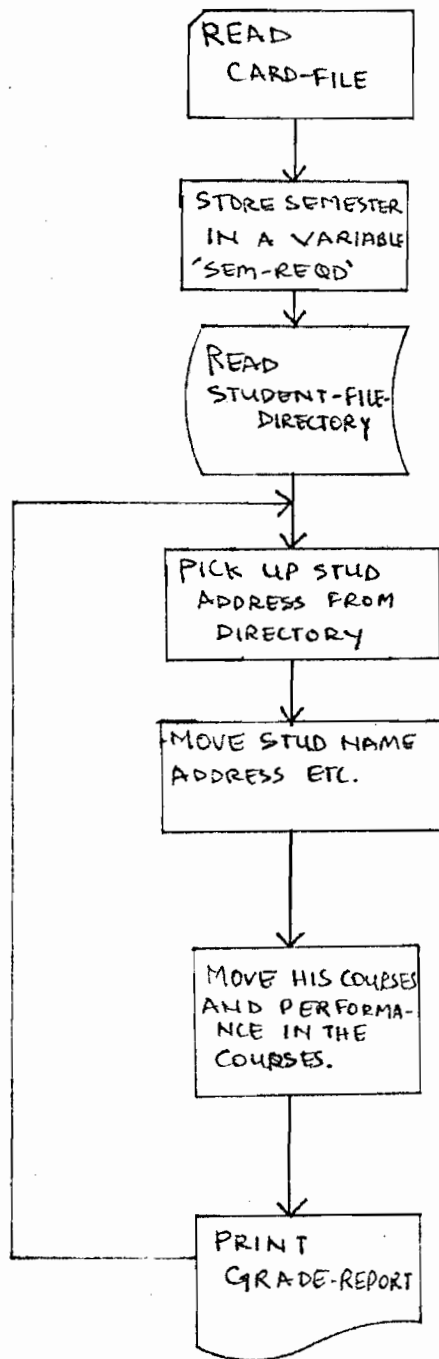
It reads the Student-file-directory in memory for direct access to the students' records.

The first address of the Student-file-directory is moved to the nominal-key of the Student-file and read. In this record the address of his first performance-record is stored, which is moved to the nominal-key of the Performance-file and read. The performance-record contains the address of his next performance-record and also the address of the course-record in which he obtained the grade and marks shown in the performance-record. The record is compared against the requirement given in the Function Card and the following action is taken.

1. If the semester-required in the Function Card is equal to zeros means the Grade Reports for the whole year are required, in which case no further checking is necessary.

2. If they are equal the address of the course-record is moved to the nominal key of the Course-file and read. Now the course-code, course description, grade, etc. are printed out. Then the address of the next performance-record from this performance-record is moved to the nominal-key of the Performance-file and read, and the above procedure is repeated. When this student's performance-records are finished, which is indicated by zeros in place of the next performance-record's address the next student is taken up and the whole procedure is repeated.

3. If they are not equal the address of the next performance-record is moved to the nominal-key of the Performance-file and read, and the procedure repeated.

PRINT GRADE-REPORTS

PRINT CLASS LISTS AND/OR PUNCH CARD DECK FOR GRADES
(PR.CL.CD)

The program prints out class lists and/or punches M-type cards for the professors to use for marking the students. The Function Card indicates the option selected.

The program reads the Function Card and holds it in memory till the end of the execution. If, however, the first card is not the Function Card it prints out a message and stops execution. The Function Card is the only data card but if there are more data cards behind it they are ignored.

It reads the Course-file-directory in memory for direct access to the course-records and holds it till the end of the execution.

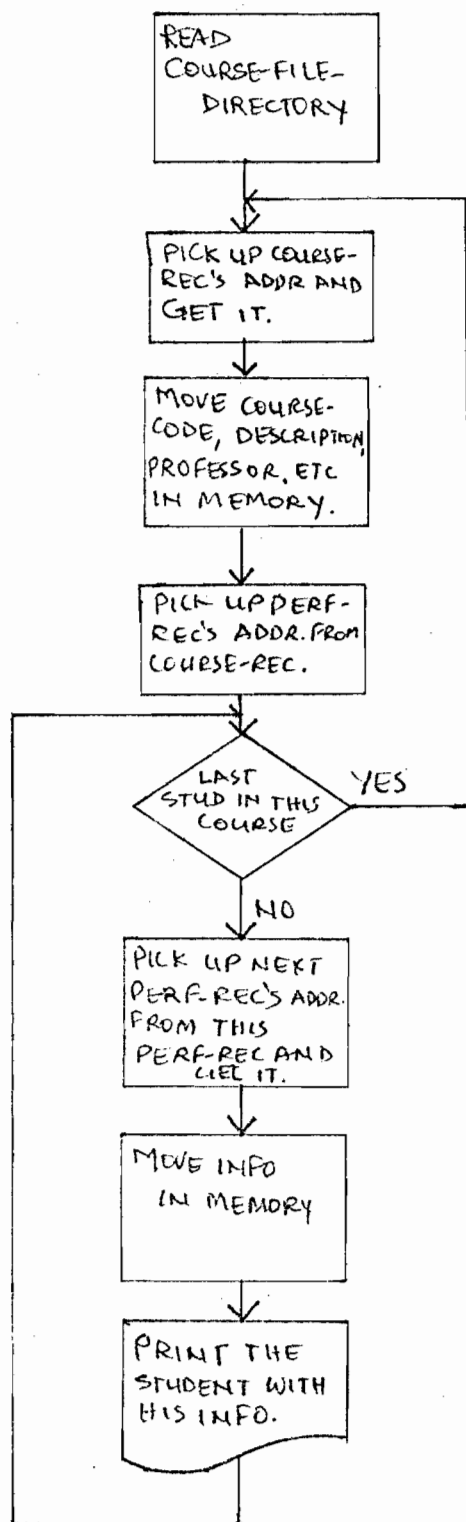
The address of the first course-record from the Course-file-direct is moved to the nominal-key of the Course-file and read. The course-record contains the address of the first performance-record of the student in this course which is moved to the nominal-key of the Performance-file and read. The performance-record contains the address of the next student's performance-record address in this class and also the address of the student whose work is shown in this performance-record. At this point the semester-required in the Function Card is compared with the semester of the performance-record.

If they are equal the address of the student-record

is moved to the nominal-key of the Student-file and read. Now the student's information from the student-record and his work from the performance-record is printed out. Next, it checks if the M-type card is required. If the M-type card is required punches a card, and if not, it moves the address of the next performance-record to the nominal-key of the Performance-file and repeats the whole procedure.

However, if the semester-required in the Function Card is 'AL' no checking is done and all the records are printed out. If the semester-required is 'NO', no checking is done and no records printed out.

The last performance-record is indicated by zeros in place of the address of the next performance-record. When it is encountered the program moves the address of the next course from the Course-file-directory to the nominal-key of the Course-file and the whole procedure is repeated from the beginning till it reaches the end of the Course-file-directory.

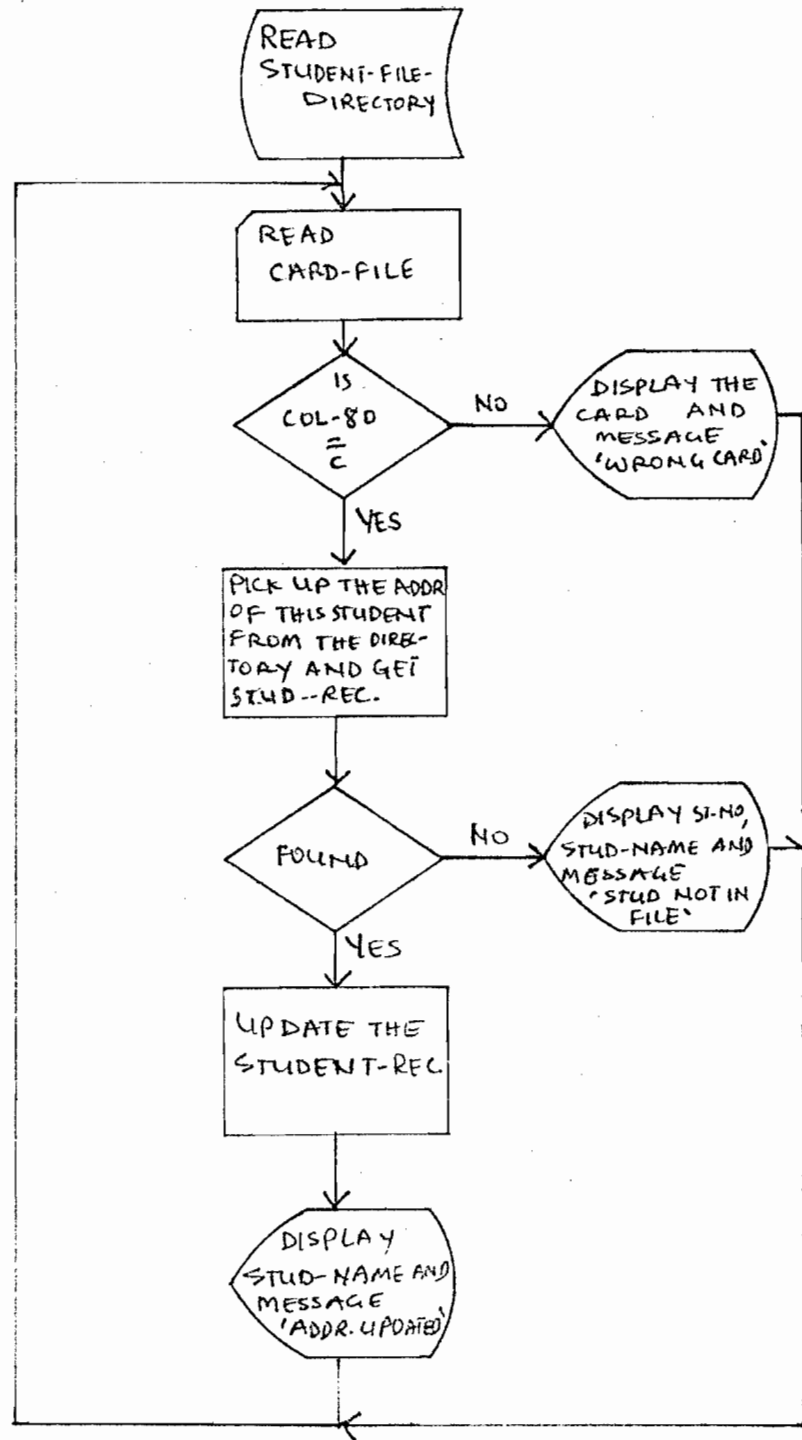
PRINT CLASS LISTS

UPDATE STUDENT-FILE - ADDRESS CHANGES (UP.STUD.FILE)

The program is meant for changing the students' address in the Student-file. New addresses of students are punched on cards according to the card type 02.

The Student-file-directory is read in memory and held till the end of the execution, which enables direct access of the desired records.

The Card-file is read and if the card col-80 does not contain 'C' it is printed out with a message 'wrong card' and the next card is read. If the Col-80 does contain 'C' the program gets the address of this student from the Student-file-directory, reads the record from the Student-file and replaces the address with the new one. It also prints out the old and the new address. Thus the whole procedure is repeated till all the cards are dealt with.

ADDRESS CHANGES

1.3.4 JCL PROCEDURES

```

//JOB
//CRCRSEFL PROC          "CR.CRSE.FILE"
//CCF      EXEC          PGM=CRSEFILE
//INDISK    DD           DSN=INDISK,
//                               UNIT=ONLY
//                               DISP=(NEW,DELETE),
//                               SPACE=(TRK,(10,5)
//                               DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//CLASSDIR   DD          DSN=CLASSDIR,
//                               UNIT=ONLN,
//                               SPACE=(TRK,(2,2),RLSE)
//                               DCB=(RECFM=FB,LRECL= 80,BLKSIZE=800)
//                               DISP=(NEW,DELETE)
//SORTWK01   DD          UNIT=ONLN,SPACE=(CYL,(3),,CONTIG)
//SORTWK02   DD          UNIT=ONLN,SPACE=(CYL,(3),,CONTIG)
//SORTWK03   DD          UNIT=ONLN,SPACE=(CYL,(3),,CONTIG)
//SORTWK04   DD          UNIT=ONLN,SPACE=(CYL,(3),,CONTIG)
//CFD        DD          DSN=CS13019.CFD,
//                               UNIT=ONLN,
//                               DISP=(NEW,CATLG,DELETE),
//                               SPACE=(TRK,(2,2),RLSE),
//                               DCB=(RECFM=FB,LRECL=2400,BLKSIZE=2400)
//CF          DD          DSN=CS13019.CF,
//                               UNIT=ONLN
//                               DISP=(NEW,CATLG,DELETE),
//                               SPACE=(TRK,(5,5),RLSE),
//                               DCB=(RECFM=FB,LRECL=37,BLKSIZE=37)
//SORTLIB     DD          DSN=SYS1.SORTLIB,DISP=SHR
//PRINTS      DD          SYSOUT=A
//CARDS        DD          DUMMY,
//                               DCB=BLKSIZE=80
//
//                               PEND

```

```

//JOB
//UPCRSEFL      PROC.      "UP.CRSE.FILE"
//UCF           EXEC      PGM=UCRSEFIL
//CFD           DD        DSN=CS13011.CFD,
//              UNIT=ONLN
//              DISP=OLD
//INDISK        DD        DSN=INDISK
//              UNIT=ONLN
//              DISP=(NEW, PASS,DELETE)
//              SPACE=(TRK,(10,2)),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//CLASSDIR      DD        DSN=CLASSDIR
//              UNIT=ONLN
//              DISP=(NEW,PASS,DELETE)
//              SPACE=(TRK(10,2)),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SRF           DD        DSN=CS13011.STUDRF,
//              UNIT=ONLN
//              DISP=OLD
//SORTLIB       DD        DSN=SYS1.SORTLIB,DISP=SHR
//SORTWK01      DD        UNIT=2314,SPACE=(CYL,(3),,CONTIG)
//SORTWK02      DD        UNIT=2314,SPACE=(CYL,(3),,CONTIG)
//SORTWK03      DD        UNIT=2314,SPACE=(CYL,(3),,CONTIG)
//SORTWK04      DD        UNIT=2314,SPACE=(CYL,(3),,CONTIG)
//CARDS         DD        DUMMY
//              DCB=BLKSIZE=80
//              PEND

```

```

//JOB
//CRSTLINK      PROC          "CR.STUD.LINK"
//CSL           EXEC      PGM=CRSTUDLK
//CFD           DD        DSN=CS13011,CFD,
//              UNIT=ONLN
//              DISP=OLD
//SFD           DD        DSN=CS13011.SFD
//              UNIT=ONLN
//              DISP=(NEW,CATLG,DELETE)
//              DCB=(RECFM=FB,LRECL=2400,BLKSIZE=2400)
//              SPACE=(TRK(2,2),RLSE)
//INDISK        DD        DSN=INDISK
//              UNIT=ONLN
//              DISP=(NEW,PASS,DELETE)
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//              SPACE=(TRK,(5,5))
//OUTDISK       DD        DSN=OUTDISK
//              UNIT=ONLN
//              DISP=(NEW,PASS,DELETE)
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//              SPACE=(TRK(5,5))
//SF            DD        DSN=CS13011.SF
//              UNIT=ONLN
//              DISP=(NEW,CATLG,DELETE)
//              DCB=(RECFM=FB,LRECL=84,BLKSIZE=84)
//              SPACE=(TRK95,5))
//SWF           DD        DSN=CS13011.STUDWF
//              UNIT=ONLN
//              DISP=OLD
//SRF           DD        DSN=CS13011.STUDRF
//              UNIT=ONLN
//              DISP=OLD
//SORTLIB       DD        DSN=SYS1,SORTLIB,DISP=SHR
//SORTWK01      DD        UNIT=2314,SPACE=(CYL,(3),,CONTIG)
//SORTWK02      DD        UNIT=2314,SPACE=(CYL,(3),,CONTIG)
//SORTWK03      DD        UNIT=2314,SPACE=(CYL,(3),,CONTIG)
//SORTWK04      DD        UNIT=2314,SPACE=(CYL,(3),,CONTIG)
//PRINTS        DD        SYSOUT=A
//CARDS         DD        DUMMY

```

DCB=BLKSIZE=80

PEND

//

//


```

//JOB
//UPTHREFL      PROC          "UP.THRE.FILE"
//UTF           EXEC          PGM=UPDATE
//CFD           DD            DSN=CS13011.CFD
//              UNIT=ONLN
//              DISP=OLD
//SFD           DD            DSN=CS13011.SFD
//              UNIT=ONLN
//              DISP=OLD
//INDISK        DD            DSN=INDISK
//              UNIT=ONLN
//              DISP=(NEW,PASS,DELETE)
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//              SPACE=(TRK(5,5),)
//OUTDISK       DD            DSN=OUTDISK
//              UNIT=ONLN
//              DISP=(NEW,PASS,DELETE)
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//              SPACE=(TRK(5,5))
//SF            DD            DSN=CS13011.SF
//              UNIT=ONLN
//              DISP=OLD
//SWF           DD            DSN=CS13011.STUDWF
//              UNIT=ONLN
//              DISP=OLD
//SRF           DD            DSN=CS13011.STUDRF
//              UNIT=ONLN
//              Disp=OLD
//SORTLIB       DD            DSN=SYS1.SORTLIB,DISP=SHR
//SORTWK01      DD            UNIT=2314,SPACE=(CYL,(3),,CONTIG)
//SORTWK02      DD            UNIT=2314.SPACE=(CYL,(3),,CONTIG)
//SORTWK03      DD            UNIT=2314.SPACE=(CYL,(3),,CONTIG)
//SORTWK04      DD            UNIT=3214.SPACE=(CYL,(3),,CONTIG)
//PRINTS       DD            SYSOUT=A
//CARDS         DD            DUMMY
//              DCB=BLKSIZE=80
//              PEND

```

```

//JOB
//INSMRKGR      PROC          "INSMARKGRADE"
//IMG           EXEC          PGM=GRADING
//SFD           DD           DSN=CS13011.SFD
//              UNIT=ONLN
//              DISP=OLD
//INDISK        DD           DSN=INDISK
//              UNIT=ONLN
//              DISP=(NEW,PASS,DELETE)
//              SPACE=(TRK(5,5))
//              DCB=(RECFM=FB,LRECL=2400,BLKSIZE=2400)
//OUTDISK       DD           DSN=OUTDISK
//              UNIT=ONLN
//              DISP=(NEW,PASS,DELETE)
//              SPACE=(TRK,(5,5))
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SF            DD           DSN=CS13011.SF
//              UNIT=ONLN
//              DISP=OLD
//SWF           DD           DSN=CS13011.STUDWF,
//              UNIT=ONLN
//              DISP=OLD
//SRF           DD           DSN=CS13011.STUDWF
//              UNIT=ONLN
//              DISP=OLD
//PRINTS        DD           SYSOUT=A
//CARDS         DD           DUMMY
//              DCB=BLKSIZE=80
//SORTLIB       DD           DSN=SYS1.SORTLIB,DISP=SHR
//SORTWK01      DD           UNIT=DISK,SPACE=(CYL,(3),,CONTIG)
//SORTWK02      DD           UNIT=DISK,SPACE=(CYL,(3),,CONTIG)
//SORTWK03      DD           UNIT=DISK,SPACE=(CYL,(3),,CONTIG)
//SORTWK04      DD           UNIT=DISK,SPACE=(CYL,(3),,CONTIG)
//              PEND

```

```
//JOB
//PRGRREC      PROC          "PR.GR.REP."
//PRG          EXEC      PGM=PRINTGR
//SFD          DD        DSN=CS13011.SFD
//              UNIT=ONLN
//              DISP=OLD
//SF           DD        DSN=CS13011.SF
//              UNIT=ONLN
//              DISP=OLD
//SWF          DD        DSN=CS13011.STUDWF
//              UNIT=ONLN
//              DISP=OLD
//SRF          DD        DSN=CS13011.STUDRF,
//              UNIT=ONLN
//              DISP=OLD
//PRINTS       DD        SYSOUT=A
//CARDS        DD        DUMMY
//              DCB=BLKSIZE=80
//              PEND
```

```
//JOB
//PRCLCD      PROC          "PR.CL.CD."
//PCD         EXEC        PGM=CLASSLST.
//CFD         DD          DSN=CS13011.CFD
//            UNIT=ONLN
//            DISP=OLD
//SF          DD          DSN=CS13011.SF
//            UNIT=ONLN
//            DISP=OLD
//SWF         DD          DSN=CS13011.STUDWF
//            UNIT=ONLN
//            DISP=OLD
//SRF         DD          DSN=CS13011.STUDRF
//            UNIT=ONLN
//            DISP=OLD
//PRINTS      DD          SYSOUT=A
//CARDS       DD          DUMMY
//            DCB=BLKSIZE=80
//            PEND
```

```
//JOB
//UPSTUDFL      PROC          "UP.STUD.FILE."
//USF           EXEC          PGM=ADDRIHNG
//SFD           DD            DSN=CS13011.SFD
//              UNIT=ONLN
//              DISP=OLD
//SF            DD            DSN=CS13011.SF
//              UNIT=ONLN
//              DISP=OLD
//CARDS         DD            DUMMY
//              DCB=BLKSIZE=80
//              PEND
```

P A R T - II

A N A L Y S I S
AND
COMPARISON WITH ALTERNATIVE SYSTEMS

2.1 DATA BASE TASK GROUP

2.1.1 INTRODUCTION

Normally, files are designed to optimize the processing of a run-unit; for other processing to be performed on the same data, the files are resorted or new files are created which redundantly include the same data. The traditional approach to data was thus to create process-oriented files. This is adequate in some circumstances, but is too costly or impractical in others. Therefore, to design systems capable of handling our current demands, it is essential to develop databases that are available to and suitable for processing by multiple applications.

The Data Base Task Group has proposed a Data Description Language for describing a database, a Data Description Language for describing that part of the database known to a program, and a Data Manipulation Language. The Data Manipulation Language is the language which the programmer uses to cause data to be transferred between his program and the database.

The principles of the Data Base Task Group and the concepts are discussed in this section.

2.1.2 PRINCIPLES OF DBTG

The principles of the Data Base Task Group are given below:

1. Allow data to be structured in the manner most suitable to each application, regardless of the fact that some or all of data may be used by other applications - such flexibility to be achieved without requiring data redundancy.
2. Allow more than one run-unit to concurrently retrieve or update the data in the database.
3. Provide protection of the database against unauthorized access of data.
4. Provide for centralized capability to control the physical placement of data.
5. Allow the declaration of a variety of data structure ranging from those in which no connection exists between data-items to network structures.
6. Allow the users to interact with the data while being relieved of the mechanics of maintaining the structural associations which have been declared.
7. Provide for separate descriptions of the data in the database and of the data known to the program.

These features provide both generality and flexibility and allow the building and manipulation of data structures as complex as necessary.

DBTG CONCEPTS

The Data Base Task Group introduced several concepts for describing database. Some of the major concepts are:

SCHEMA

A Schema consists of Data Description Language entries and is a complete description of a database. It includes the names and descriptions of all the areas, set occurrences, record occurrences and associated data-items and data-aggregates as they exist in the database.

SUB-SCHEMA

A Sub-Schema also consists of Data Description Language entries. It, however, need not describe the entire database but only those areas, sets, records, data-items and data-aggregates which are known to one or more specific programs. Further, it describes them in the form in which they are known to those specific programs and it may also rename them.

AREA

An Area is a named sub-division of the addressable storage space in the database and may contain occurrences of records and sets or parts of sets of various types.

SET

A set is a named collection of record types with an owner and may have one or an arbitrary number of member

records declared for it in the Schema. The tree and network structure figures show the sets.

2.1.3 DATA STRUCTURES

SEQUENTIAL STRUCTURE

A sequential data structure is one in which each element in the structure, except the first and last, is related to the element preceding it and the element following it. A list is an example of a sequential data structure. A list may be a one-way list, where each element points only to the next, a two-way or a circular list. There is "one-to-n" relationship between the owner records and the member records. Thus, for each occurrence of the owner record, there may be "n" occurrence of the member records. Diagram 1 is a set representation of a one-way, two-way and a circular sequential data structure. Tree structure has several sets of one or more members, but it is not a sequential structure.

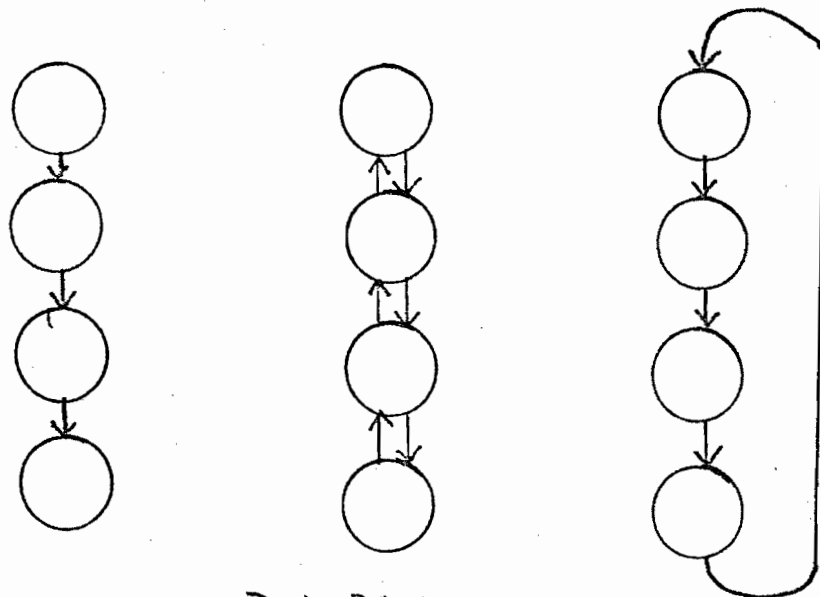
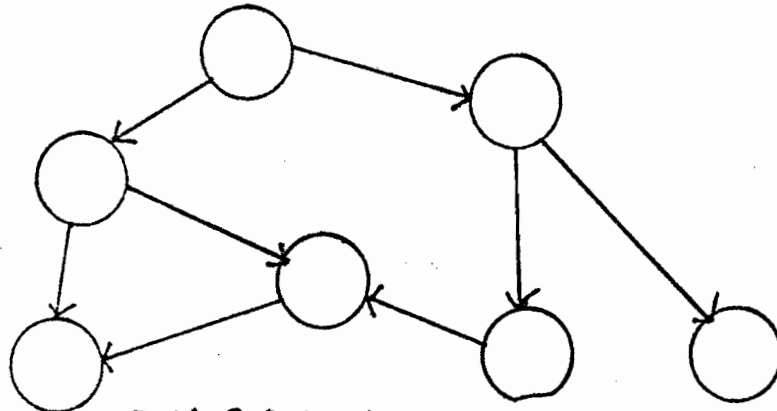
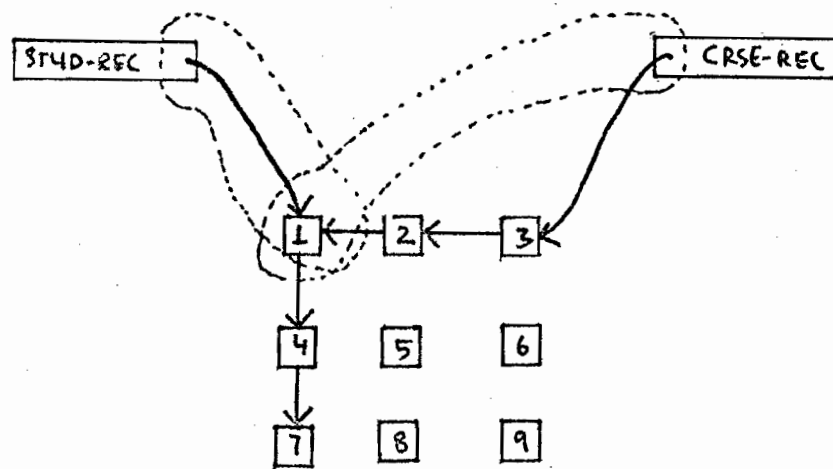


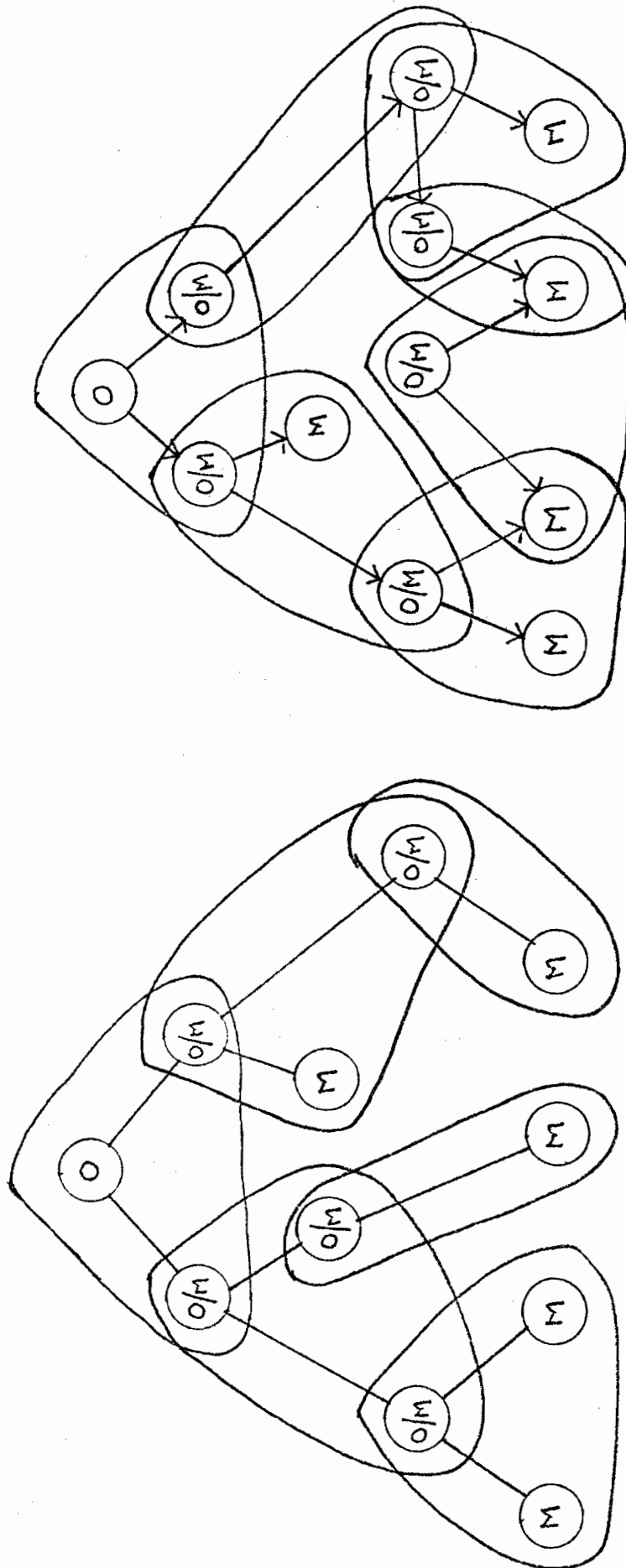
DIAGRAM - 1

NETWORK STRUCTURE

A network is the most general form of data structures. In a network any given element may be related to any other.

Diagram 4 shows a network structure in which a node has more than one branches into it. In network structure a record may participate as a member in more than one set, and therefore may have more than one owner, which allows networks to be built. In using sets to represent networks each set represents a collection of one-to-many relationships. Where there is many-to-many relationship, for example that between students and classes in a school, it can be represented by two sets, the owners of which are the records involved in the relationship. The members of both sets are a third record type, each occurrence of which represents one association between two occurrences of the owner records, for example, the performance of a given student in a given class. Since it is often desirable to store information about the associations themselves, for example, the grade for a student in a class, these records are useful apart from their role in the representation of many-to-many relationships. This corresponds to the performance record which participates in two sets, whose owners are "student" and "class". In diagram 5, record no. 1 represents one association between two occurrences of the owner records, i.e. a given student in a given class.

NETWORK-STRUCTUREDIAGRAM - 4DIAGRAM - 5



NETWORK STRUCTURE

TREE STRUCTURE

2.2 IMPLEMENTATION OF SRMS IN DBTG

The database of the Student Record Management System has been structured, which provides generality and flexibility and is suitable for each application without requiring redundancy. It has one-to-one correspondence with the proposals of the Data Base Task Group.

The database is structured in a manner which permits access in different sorted ways without resorting. The directories have pointers to the students and classes, which in turn have pointers to the connecting records in sequential order. The students with their courses and courses with their students can be accessed without any sorting.

The database will be accessed only by one run-unit at a time, and therefore, it is not necessary for the protection against concurrent retrieval and updating data in the database.

The programs of the system provide for control of relative placement of records in the database to increase efficiency.

The database is structured which corresponds to the proposals of the Data Base Task Group shown in diagrams 6, 7 and 8.

It permits the users to retrieve and update data without being concerned of the mechanics of maintaining the structure.

Each program is only concerned with the data known to it and not the entire database.

DATA STRUCTURE

Since there is one-to-one correspondence of database of the Student Record Management System with the proposals of the Data Base Task Group, it can be described in terms of Schema and Sub-Schema using Set concept. The Schema Data Description Language provides for an arbitrary number of Set modes and representation of Data Structure. The data structure of the SRMS's database corresponds to the DBTG's structure.

The one-way list data structure corresponds to a set composed of student-information-record as owner and performance-records as its members, shown in diagram 2. It also corresponds to a set of class-record as owner and performance-records as its members, which is shown in diagram 3.

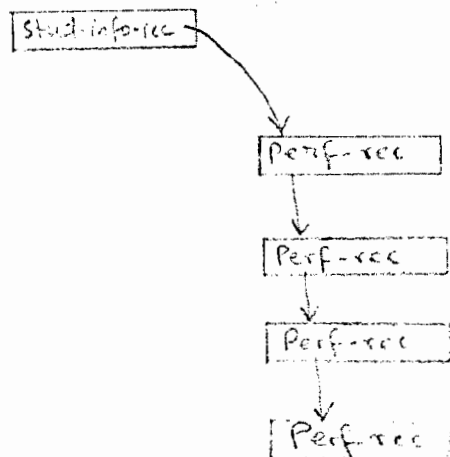


DIAGRAM 2

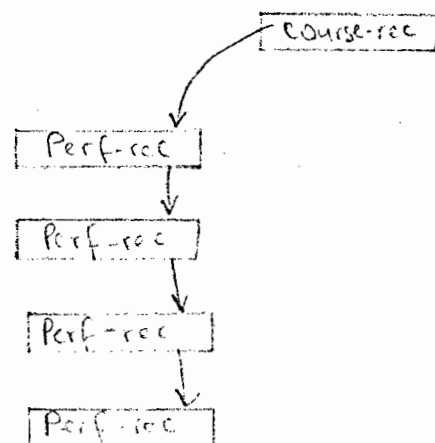
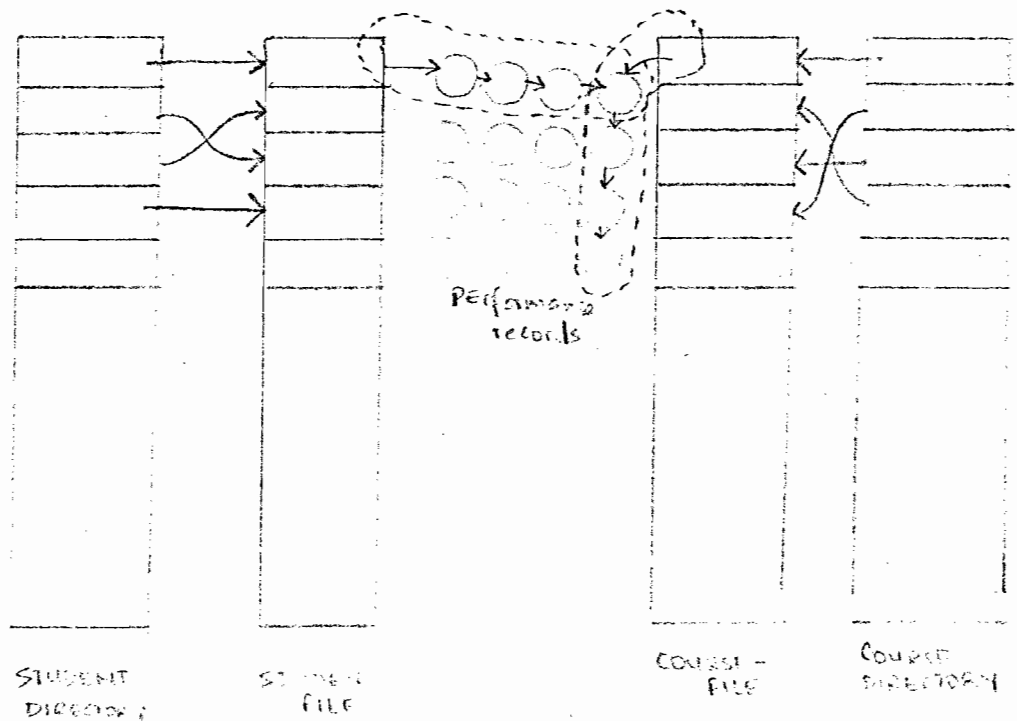


DIAGRAM 3

SRMS DATABASE IN SETS

The database of the Student Record Management System has been described in the Schema in six sets. The system has two directories, a student-file and a course-file, which are described in four separate sets with dummy-records as their owners. The fifth set consists of student-record as owner with performance-records as members, and the sixth one with course-record as owner and performance-records as its members. The systems database is illustrated by diagram 4.

DIAGRAM 4

The sets in the database are required to be sorted in some order. Several sorting and searching techniques have been discussed in Appendix A. The analysis of the different sorting techniques indicate that Radix sort will be quite suitable for SRMS's database.

For exercise few Sub-Schemas and programs have been written. The flowcharts for these programs have also been drawn to show how to invoke a Sub-Schema of the Schema in programs. There are some verbs, such as, Insert, Search, Modify, Sort, etc. in the Schema. Attempts have been made to expand the internal steps of these verbs by flow-charts. However, the programs were not run on DETG Compiler.

2.2.1 SRM's DATABASE2.2.1.1 SCHEMA NAME IS STUDENT-RECORD-SYSTEM;PRIVACY LOCK IS STUDSYS.AREA NAME IS STUD-INFO-AREA;PRIVACY LOCK IS ST-IN-KEY.RECORD NAME IS STUDENT-REC;LOCATION MODE IS VIA STUD-INFO-SET SET;WITHIN STUD-INFO-AREA;PRIVACY LOCK IS ST-REC-KEY.

02 DEPT PICTURE XXX.

02 STUD-NO PICTURE 9(06).

02 STUD-NAME PICTURE X(20)

02 DEGREE PICTURE XXXX.

02 LEVEL PICTURE XX.

02 SEX PICTURE X.

02 STREET PICTURE X(17).

02 APT-NO PICTURE XXXX.

02 TOWN PICTURE X(16).

02 TELEPHONE PICTURE X(07).

02 PERF-ADDR PICTURE 9999.

SET NAME IS STUD-INFO-SET;MODE IS POINTER-ARRAY;ORDER IS SORTED;OWNER IS STUD-DUMMY-REC.MEMBER IS STUDENT-REC OPTIONAL AUTOMATIC;ASCENDING KEY IS STUD-NO, DEPT;SEARCH KEY IS STUD-NO, DEPT;SET OCCURENCE SELECTION IS THRULOCATION MODE OF OWNERUSING STUD-NO.

AREA NAME IS COURSE-INFO-AREA;

PRIVACY LOCK IS CR-IN-KEY.

RECORD NAME IS COURSE-REC;

LOCATION MODE IS VIA COURSE-INFO-SET SET;

WITHIN COURSE-INFO-AREA;

PRIVACY LOCK IS CR-REC-KEY.

02 WORK-ADDR PICTURE 9(05).

02 CRSE-NO PICTURE 9(06).

02 SEMESTER PICTURE X.

02 COURSE-DESC PICTURE X(25).

02 COURSE-CREDIT PICTURE 9.

SET NAME IS COURSE-INFO-SET;

MODE IS POINTER-ARRAY;

ORDER IS SORTED;

OWNER IS COURSE-DUMMY-REC.

MEMBER IS COURSE-REC OPTIONAL AUTOMATIC;

ASCENDING KEY IS CRSE-NO;

SEARCH KEY IS CRSE-NO;

SET OCCURENCE SELECTION IS THRU

LOCATION MODE OF OWNER

USING CRSE-NO.

STUDENT WITH PERF-RECS-SET

AREA NAME IS ST-PERF-INFO-AREA;

PRIVACY LOCK IS ST-PR-IN-KEY.

RECORD NAME IS PERF-REC;

LOCATION MODE IS VIA ST-PERF-INFO-SET SET;

WITHIN ST-PERF-INFO-AREA;

PRIVACY KEY IS ST-PR-REC-KEY.

02 ADDR-1 PICTURE 9999.

02 ADDR-2 PICTURE 9999.

02 YEAR PICTURE 99.

02 TERM PICTURE X.

02 GRADE PICTURE X.

02 MARK PICTURE 99.

02 ADDR-3 PICTURE 9999.

02 ADDR-4 PICTURE 9999.

SET NAME IS ST-PERF-INFO-SET;

MODE IS ONE-WAY-LIST;

ORDER IS FIRST;

OWNER IS STUDENT-REC.

MEMBER IS PERF-REC OPTIONAL AUTOMATIC;

SET OCCURENCE SELECTION IS THRU

STUD-INFO-SET USING

CURRENT OF SET

STUDENT-REC USING STUD-NO.

COURSE-REC WITH PERF-RECS-SET

SET NAME IS CR-PERF-INFO-SET;

MODE IS ONE-WAY-LIST;

ORDER IS NEXT;

OWNER IS COURSE-REC.

MEMBER IS PERF-REC OPTIONAL AUTOMATIC;

SET OCCURENCE SELECTION IS THRU

COURSE-INFO-SET USING

CURRENT OF SET

COURSE-REC USING CRSE-NO.

2.2.2.2 SUB-SCHEMAS CREATE-COURSE-FILE

SUB-SCHEMA IDENTIFICATION DIVISION.

SUB-SCHEMA NAME IS CREATE-CRSE-FILE OF

SCHEMA NAME STUDENT-RECORD-SYSTEM

PRIVACY LOCK IS CCF-KEY

PRIVACY KEY FOR COPY IS STUDSYS.

AREA SECTION

COPY COURSE-INFO-AREA

PRIVACY LOCK IS CI-KEY.

RECORD SECTION

01 COURSE-REC;

WITHIN COURSE-INFO-AREA;

PRIVACY LOCK IS CR-KEY.

02 WORK-ADDR PICTURE 9(05).

02 CRSE-NO PICTURE 9(06)

02 SEMESTER PICTURE X.

02 COURSE-DESC PICTURE X(25).

02 COURSE-CREDIT PICTURE 9.

SET SECTION

COPY COURSE-INFO-SET.

CREATE STUDENT-FILE, PERFORMANCE-FILE, AND LINK THEM WITH
COURSE-FILE

SUB-SCHEMA IDENTIFICATION DIVISION.

SUB-SCHEMA NAME IS CR-STUD-WDRIC-ATTACH-CRSE.

OF SCHEMA STUDENT-RECORD-SYSTEM

PRIVACY LOCK IS SWC-KEY

PRIVACY KEY FOR COPY IS STUDSYS.

AREA SECTION

COPY ALL AREAS.

RECORD SECTION

01 STUDENT-REC;

WITHIN STUD-INFO-AREA;

PRIVACY LOCK IS SI-KEY.

02 DEBT PICTURE XXX.

02 STUD-NO PICTURE 9(06).

02 STUD-NAME PICTURE X(20).

02 DEGREE PICTURE XXXX.

02 LEVEL PICTURE XX.

02 SEX PICTURE X.

02 STREET PICTURE X(17).

02 APT-NO PICTURE XXXX.

02 TOWN PICTURE X(16).

02 TELEPHONE PICTURE X(07).

02 PERF-ADDR PICTURE 9999.

01 COURSE-REC;

WITHIN COURSE-INFO-AREA;

PRIVACY LOCK IS CI-KEY.

02 WORK-ADDR PICTURE 9(05).

02 CRSE-NO PICTURE 9(06).

02 SEMESTER PICTURE X.

02 COURSE-DESC PICTURE X(25).

02 COURSE-CREDIT PICTURE 9.

01 PERF-REC

WITHIN ST-PERF-INFO-AREA,

CR-PERF-INFO-AREA

PRIVACY LOCK IS ST-CR-KEY.

02 ADDR-1 PICTURE 9999.

02 ADDR-2 PICTURE 9999.

02 YEAR PICTURE 99.

02 TERM PICTURE X.

02 GRADE PICTURE X.

02 MARK PICTURE 99.

02 ADDR-3 PICTURE 9999.

02 ADDR-4 PICTURE 9999.

SET SECTION

COPY ALL SETS.

2.2.1.3 PROGRAMS AND FLOWCHARTS

CREATE COURSE-FILE

IDENTIFICATION DIVISION.

PROGRAM-ID.

PRIVACY KEY OF COURSE-INFO-AREA IS CI-KEY

PRIVACY KEY OF COURSE-REC IS CR-KEY.

DATA DIVISION.

SCHEMA SECTION

INVOKE SUB-SCHEMA CREATE-CRSE-FILE

OF SCHEMA STUDENT-RECORD-SYSTEM

PROCEDURE DIVISION

OPEN AREA COURSE-INFO-AREA

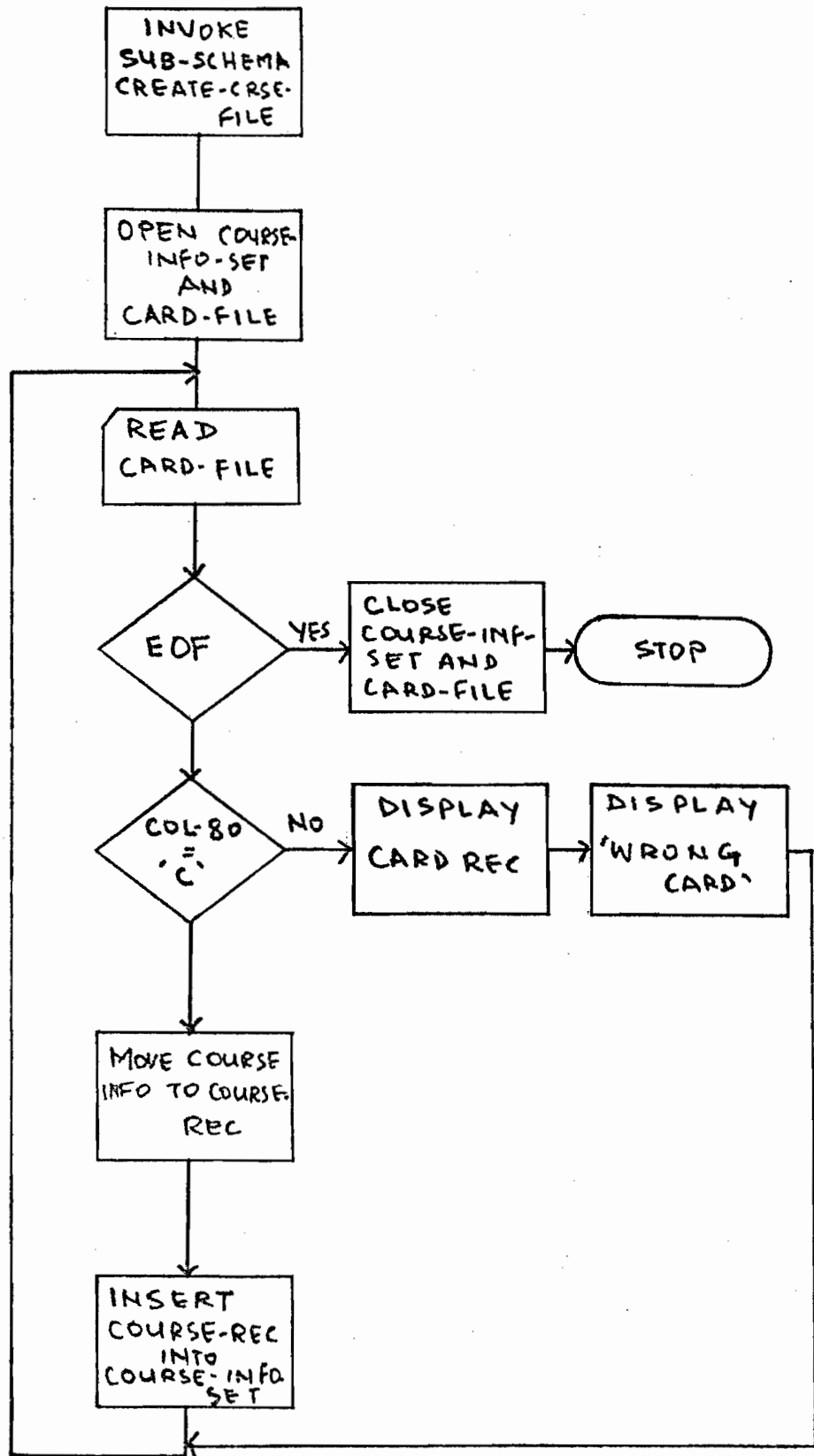
USAGE-MODE IS UPDATE

PROCESS IT.

INSERT COURSE-REC INTO COURSE-INFO-SET.

GO TO PROCESS IT.

CLOSE AREA COURSE-INFO-AREA.

CREATE COURSE-FILE

CREATE STUD-FILE, PERFORMANCE FILE AND LINK THEM
WITH COURSE-FILE.

IDENTIFICATION DIVISION.

PROGRAM-ID.

PRIVACY KEY OF ALL AREAS IS ST-IN-KEY.

CR-IN-KEY, ST-PR-IN-KEY, CR-PR-IN-KEY

PRIVACY KEY OF ALL RECORDS IS SI-KEY,

CI-KEY, ST-CR-KEY.

DATA DIVISION.

SCHEMA SECTION

INVOKE SUB-SCHEMA CR-STUD-WORK-ATTACH-CRSE.

OF SCHEMA STUDENT-RECORD-SYSTEM

PROCEDURE DIVISION.

OPEN ALL

PROCESS IT.

IF COL-80 IS EQUAL TO 'F' MOVE DEPARTMENT TO DEPT
MOVE STUDENT-NO TO STUD-NO...

ELSE GO TO MOVE-PERF.

INSERT STUDENT-REC INTO STUD-INFO-SET.

GO TO PROCESS IT.

MOVE-PERF.

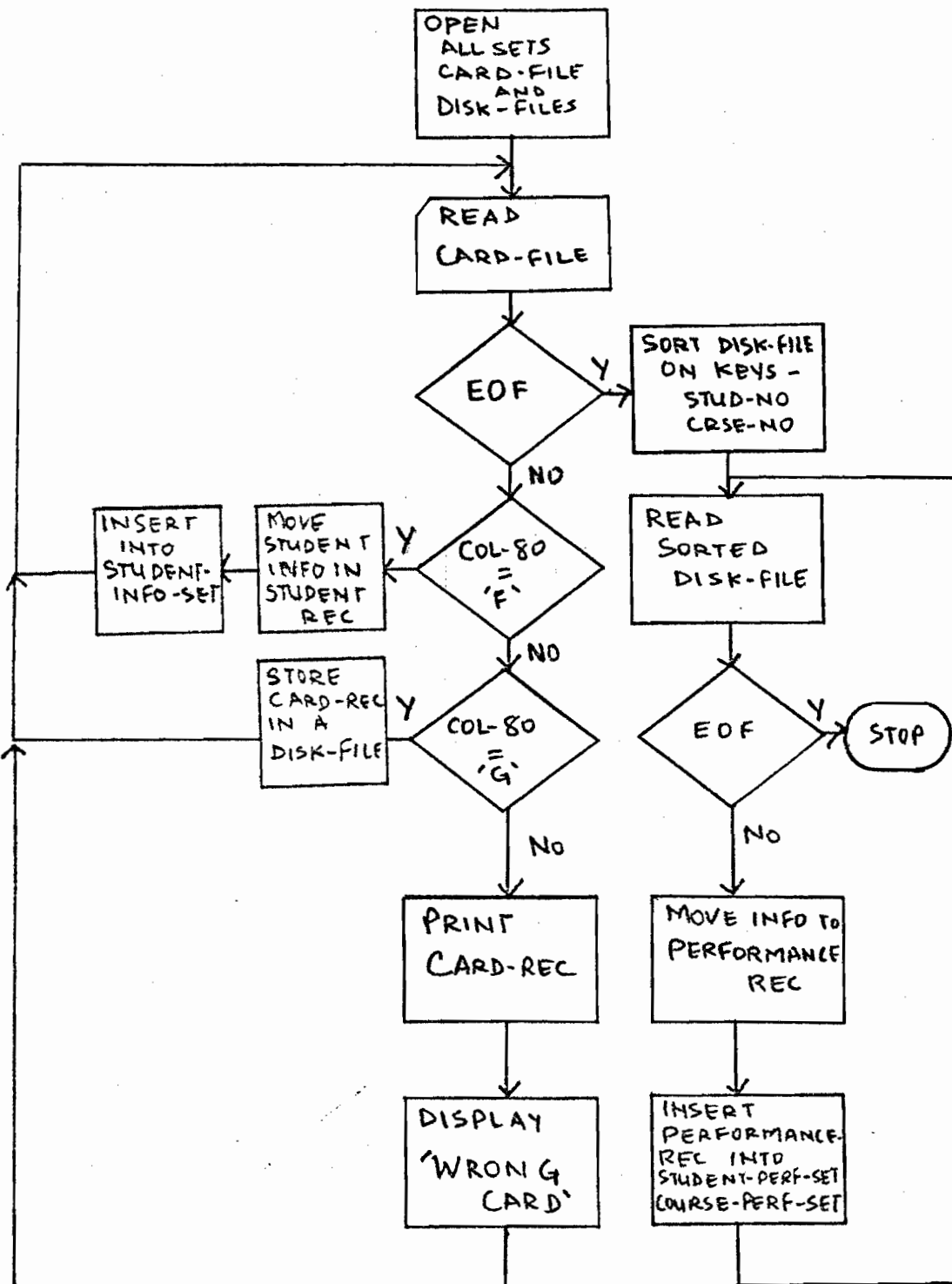
MOVE YEAR-IN TO YEAR TERM-IN TO TERM...

INSERT PERF-REC INTO CR-PERF-INFO-SET,

ST-PERF-INFO-SET. GO TO PROCESS IT.

CLOSE ALL

CREATE STUDENT-FILE, PERFORMANCE-FILE
AND LINK THEM WITH COURSE-FILE



INSERT MARKS OR MARKS AND GRADES

IDENTIFICATION DIVISION.

PROGRAM-ID.

PRIVACY KEY OF ALL AREAS IS ST-IN-KEY,

CR-IN-KEY, ST-PR-IN-KEY, CR-PR-IN-KEY

PRIVACY KEY OF ALL RECORDS IS SI-KEY, CI-KEY, ST-CR-KEY.

DATA DIVISION

SCHEMA SECTION

INVOKE SUB-SCHEMA CR-STUD-WORK-ATTACH-CRSE

OF SCHEMA STUDENT-RECORD-SYSTEM.

PROCEDURE DIVISION

OPEN ALL

PROCESS IT

FIND COURSE-REC USING CRSE-NO.

GET COURSE-REC.

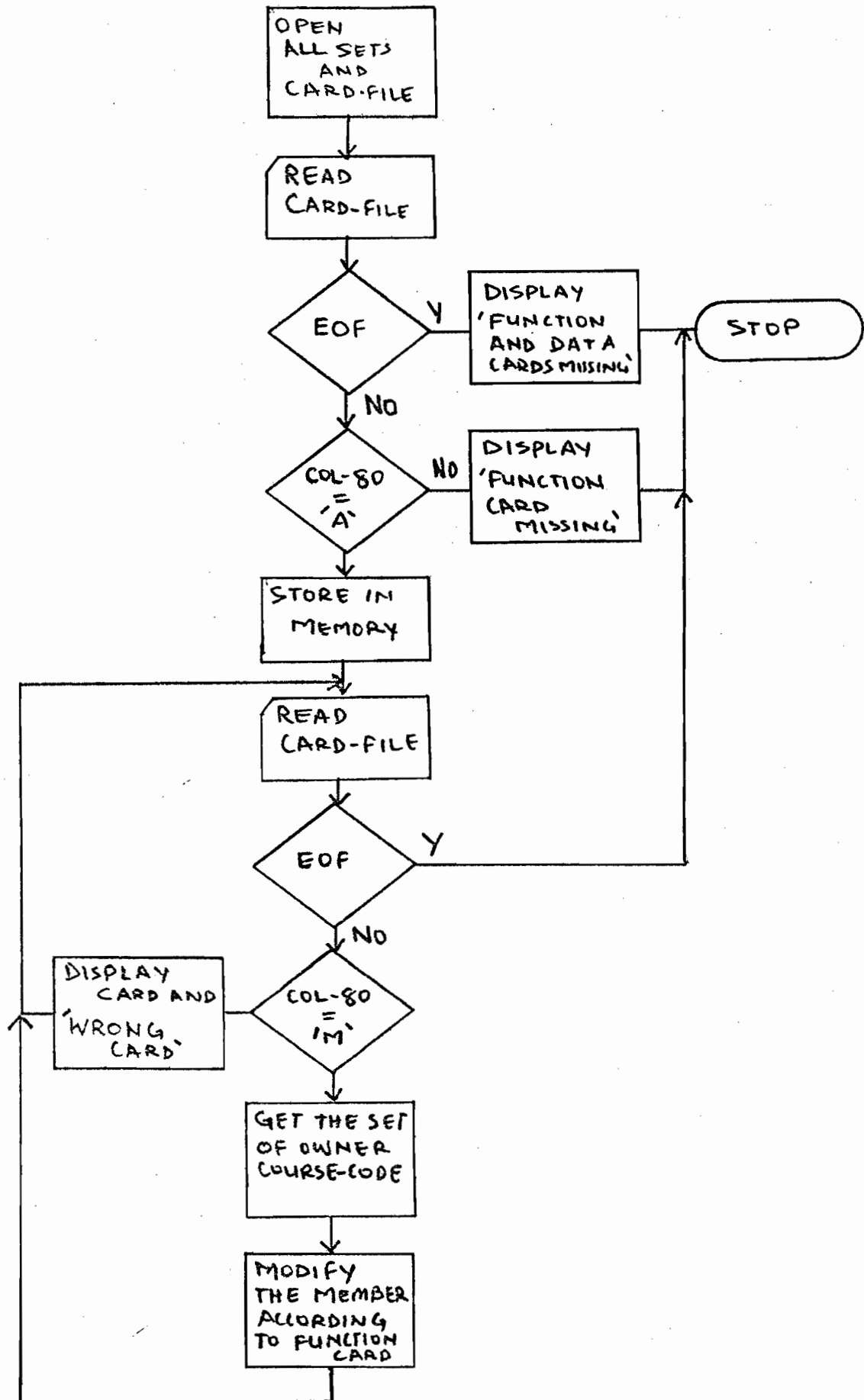
PROCESS AGAIN.

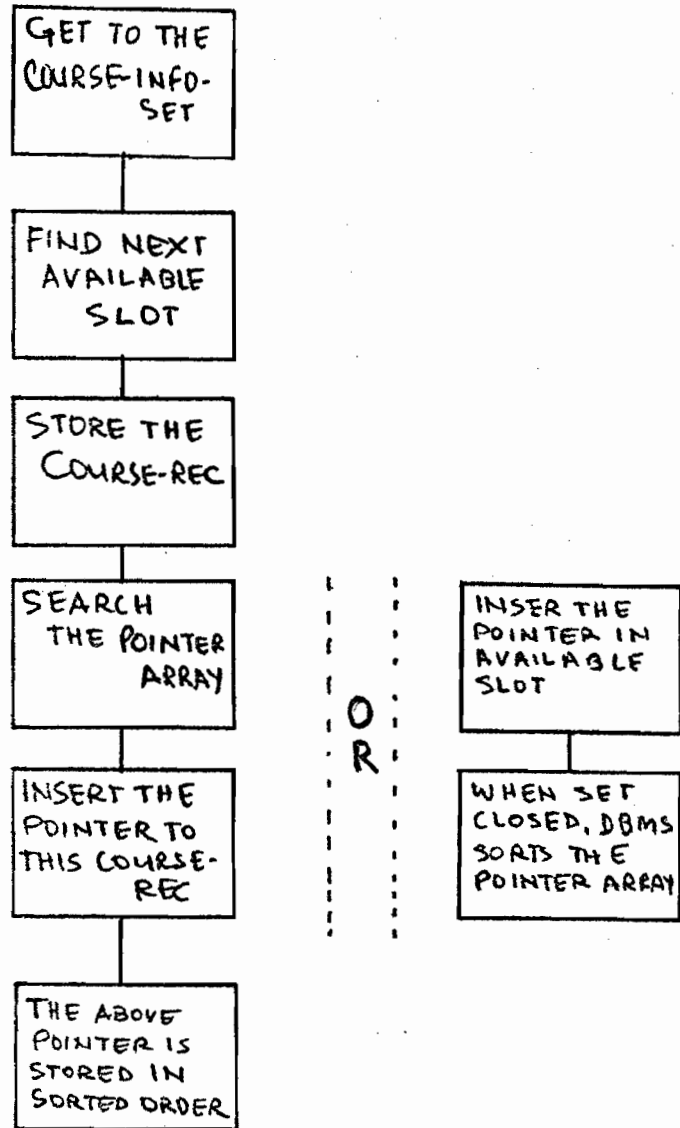
FIND NEXT PERF-REC RECORD OF CR-PERF-INFO-SET SET

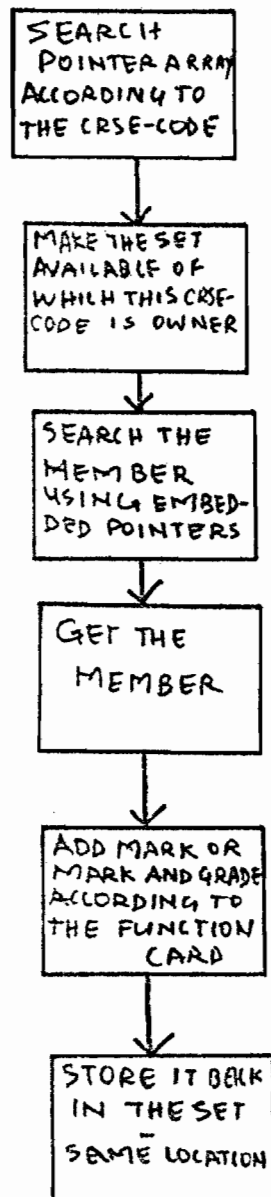
MODIFY PERF-REC USING GRADE MARK.

GO TO PROC-AGAIN.

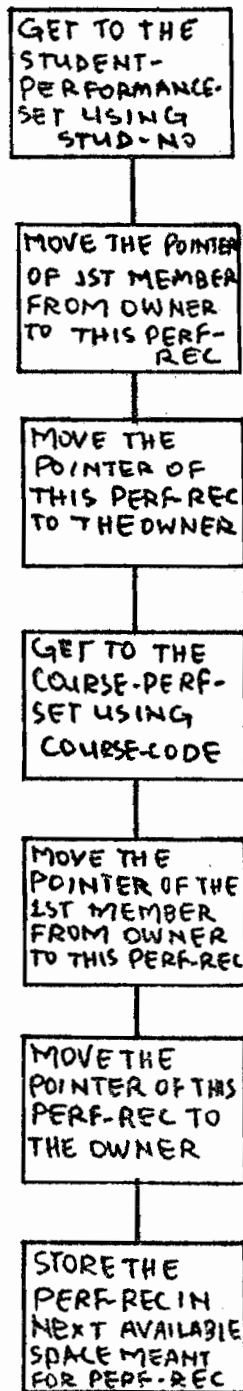
55
INSERT MARKS OR/AND GRADES



INSERT

MODIFY

INSERT PERFORMANCE RECORDS
INTO COURSE AND STUDENT SETS



2.2.2 CONCLUSION

The data base is the foundation of the information system. In reality the data base is a collection of data files. A data file in turn is a collection of data records that are composed of data fields. There are many advantages and some disadvantages of the generalized data base. Some of them are given below:

1. The ability to organize data in a manner which is suitable and appropriate to the interrelated functions of the organization.

2. Data description is contained in the data base independent from programming functions, thus relieving programmers of data management.

3. Ability to provide users with a direct interface with the data base.

4. Allows more integration of data elements to minimize redundancy.

5. Ability to grow without a major overhaul of the system.

6. Gives faster response to user needs.

7. Allows users to interrogate the data base and make inquiries.

8. Ability to meet changing needs of users over time.

9. Data errors and inconsistencies are reduced because duplication is reduced.

10. Cost savings are effected.

The disadvantages of the generalized data base management system are:

1. The design and implementation of the data base approach requires highly skilled professionals.
2. The initial investment is extremely high.
3. A sophisticated level of hardware and software is necessary.
4. High level of security safeguards and backup is required.
5. Errors might develop throughout the data base because of a single error emanating from a source document.

2.3 COST ANALYSIS TECHNIQUES

2.3.1 INTRODUCTION

The objective of this part of the Student Record Management System is to develop techniques to analyse various costs.

We shall restrict ourselves to computer operations involving peripheral devices, which is basically the cost of peripheral device access and peripheral device storage. However, there are other factors to be taken into consideration such as personnel, CPU time, maintenance of computers, etc., but for our purpose we shall not consider them.

The cost of any I/O operation can be analysed by the number of accesses and auxiliary storage. For instance, the cost of retrieving, deleting, adding or storing information can be computed by the number of accesses and/or storage required.

The number of peripheral device access depends on the organization of the databank. For instance, a request to retrieve an item from the databank may require only one peripheral device access or it may require many, depending on the organization.

The organization of the databank and all I/O operations are discussed in detail in this part. The total cost is the summation of the cost of I/O operations and storage.

2.3.2 DEFINITIONS

Before we start the SRMS databank, it is necessary to start with its description and some definitions used very often:

A databank is a collection of files, related to each other. The databank of SRMS has five files.

A file is a collection of identical records on a secondary storage device. In case of SRMS the files are on IBM 3330 DASD.

A record is a named collection of one or more data-fields or data-aggregates. There may be an arbitrary number of occurrences of a record in a file of each record type. For example, there would be one occurrence of the record type STUDENT-INFC-RECORD for each student. This distinction between the actual occurrences of a record and a type of the record is an important one.

A data-field is the smallest unit named data. The amount of storage defined for a field depends on the type and range of data to be stored. For instance, Student-number is a data-field and is 6 bytes long.

Files in a databank may be organized in different ways determined by the access method. Three of the five files of the SRMS are organized using relative access and two using sequential access method.

Relative access: Each record is stored at a unique position and accessed by the address of this position. The address is measured relative to the beginning of the file.

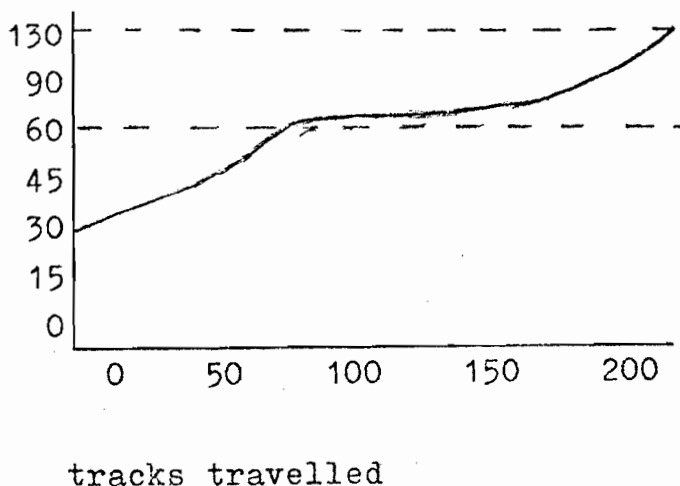
In this case any record of the file can be accessed from secondary to primary storage without accessing the preceding records. Three files of the SRMS organized by relative access are cross-referenced, which makes it faster and cheaper to access the connecting records directly.

Sequential access: The records of a file are organized in sequence and a record can be accessed only after all the preceding records have been accessed. Two files of the SRMS organized by sequential access method have only a few records in each files for directories.

A block is the unit of peripheral storage which is read or written in a single access.

An access is the operation of copying a block into a buffer area in core, or conversely copying a block from core into secondary storage.

Access time differs from one device to another. However we will restrict ourselves to IBM 3330, which is used for the system. Table 1 shows the average access for 2314 and 3330.



Device	Track Capacity (Bytes)	Number of tracks cyl.	Number of cyl.	Seek time		
				Min (MS)	Max (MS)	Ave (MS)
2314	7294	20	200	25	130	60
3330	13030	40	400	10	30	55

Table 1

2.3.3 ACCESS COST

The access cost, C_A is the cost per access to data on peripheral devices. The access cost can be expressed as:

$$C_A = u C_{IO}$$

where C_{IO} is the actual charge for access or I/O request and

$$u = \begin{cases} 1 & \text{if the peripheral unit is on line} \\ & \text{and} \\ 1 + \frac{C_m}{v C_{IO}} & \text{if the peripheral unit must be mounted} \end{cases}$$

where, C_m is the mounting charge and v blocks per volume. A volume must be mounted after every v I/O requests.

The charge C_{IO} will usually vary in an installation depending on time of day, priority, core, etc. At McGill an access $C_{IO} = 0.133\phi$, if programs run in 100k at priority 2 and $C_{IO} = 0.167\phi$, if a program runs in 200k at priority 2. In our case we will presume that files are stored on-line so that there is no mounting charge and

$$C_A = C_{IO} = 0.133\phi$$

2.3.4 STORAGE COST

To find the storage cost, C_s , we must relate the block size, B , to the unit of storage, Q . Let the number of blocks per unit of storage, b , be so that

$$bB=pQ$$

where p is the proportion of the storage unit actually occupied by data. The storage cost is

$$\begin{aligned} C_s &= \frac{C_o}{b} \Delta t (n_0 T + n_1 \sum_{i=1}^{T-1} i) \\ &= \frac{C_o}{b} t (n_0 + n_1 (T-1)) \end{aligned}$$

where C_o is the cost per unit time of a unit storage space. The storage cost will vary, depending on whether the file is stored on tape, on-line or off-line. At McGill storage charge for on-line is $C=20\text{¢}$ per track/week. This can be expressed in terms of $n(t)$:

$$C_s = \frac{C_o}{b} \int_0^t n(t') dt'$$

2.3.5 ANALYSIS OF SORTING

External sorting can be made fairly straightforward with a few simplifying assumptions. We follow Knuth's "The Art of Computer Programming", sections 5.4.1, 5.4.6., and 5.4.9. The time required to do a sort-merge on a disk can be expressed

$$NCwt(1 + \lceil \log p S \rceil)$$

where N = number of records in a file

C = number of characters per record

w = "overhead ratio"-the ratio of t to the effective time to read or write a character. On IBM 2314 will full cylinders and tracks,

$$w = \begin{cases} 1 & \text{if the file is on single cylinders} \\ 1.05 & \text{if the file is on contiguous cylinders} \\ 1.12 & \text{if the file is on non-contiguous cylinders} \\ & \text{or if the multi-tasking causes arm contention} \end{cases}$$

t = the time required to read or write a single character

On IBM 2314, $t = \frac{25 \text{ ms/tracks}}{7294 \text{ chars/track}} = 3.44 \text{ usec}$

S = the number of initial runs, i.e. the number of subfiles that are sorted with an internal memory independently and placed on external files before merging begins

P = the number of simultaneous merges used to merge the resulting subfiles together

The expression of $NCwt$ is the time required to read a single pass of the file and $1 + \lceil \log_p S \rceil$ is the number of passes, i.e. a pass to distribute the S initial runs and $\log_p S$ passes to do the P -ways merge. S is determined by the number of records in the placement section, (see Knuth 5.4.1), P that can fit only into core memory (M characters) less three buffers (B characters each);

$$P^1 = (M - 3B) / C$$

According to Knuth for random data the number of initial runs S can be estimated as

$$S \simeq \left\lceil \frac{N}{2P} + \frac{7}{6} \right\rceil.$$

Once S is determined the order of the merge P , can be found which gives the smallest number of passes, $\lceil \log_p S \rceil$, subject to P being small enough that P buffers will fit

into core memory. The appropriate relationship among P, S and $m = \lceil \log_p S \rceil$ is:

$$P = \left\lceil S^{\frac{1}{m}} \right\rceil$$

since $m = \lceil \log_p S \rceil = \lceil \ln S / \ln P \rceil$

implies $(m-1) \ln P < \ln S \leq m \ln P$

i.e. $P^{m-1} < S \leq P^m$

This relation can be used as follows: given S find the smallest m for which $P = \left\lceil S^{\frac{1}{m}} \right\rceil$ is not greater than the number of buffers that can fit into core. Thus in Knuth's example (P3.6.4) $B=5000$ and $S=60$ indicates that for an 8 way merge two passes are required.

An expression for the number of access required in a sort can be obtained from (1) by replacing the time wt by $2/B$. Since $NC/B=n$, the number of blocks of data in the file, we have the cost for sorting:

$$C_s = 2n (1 + \lceil \log_p S \rceil) C_A \text{ or}$$

$$2n (1+m) C_A$$

The factor of the two enters because each pass involves simultaneous reading and writing on different disk files. In analysis of real life sorting the parameters P and S must be chosen to correspond the sort parameters actually used.

Analysis indicate that the sort could be done either with $m=3$ in 100k or with $m=2$ in 200k.

In our case we have: $M=200k$, $B \leq 13030$ bytes.

2.3.6 SRMS DATABANK DESCRIPTION

The SRMS has five files with a fixed number of blocks, which makes it easier to analyze the cost. Two files are organized by SAM (Sequential Access Method) with one record in one file and two records in the other file. The rest of the three files known as Student-file, Course-file and Performance-file are organized by RAM (Relative Access Method) which permits the access of any record directly if the address is known. The two sequential files are in fact the two directories for the student-file and course-file. The three files, Student-file, Course-file and Performance-file are linked up with each other by embedded pointers.

The SRMS is designed to maintain records of upto 1000 students. The length of files in bytes at any given time t is:

$$\begin{aligned} \text{CFD} &= N_c(t) * (K_l + A_l) \\ \text{CF} &= N_c(t) * R_l \\ \text{SFD} &= N_s(t) * (K_l + A_l) \\ \text{SF} &= N_s(t) * R_l \\ \text{PF} &= N_s(t) * (N_p(t) * R_l) \end{aligned}$$

where, N_c is the number of courses offered, N_s is the number of students, N_p the number of Performance-records, R_l , record length in bytes, K_l , key length in bytes and A_l , address length in bytes.

However, to analyze the cost of various operations and compare it with another system, we assume the number of students is 1000, the number of courses offered by the department is 200 and finally the number of Performance-

records is 10,000, @ 10/students. With this assumption we can see the strength of the files with respect to the number of records/track and the number of tracks occupied by each file which is shown in the table below:

FILES	No. of Stds	No. of CRSES	KEY LENGTH	ADDR. LENGTH	RECORD LENGTH	No. of RECORDS	No. of RECS/ TRKS (Nr/T)	No. of TRACKS 1ST SEM	TRACKS 2ND SEM
	(Ns)	(Nc)	(Kl)	(Al)	(Rl)		(Nr/T)	(Nt)	(Nt)
CFD		200	6	3	1,800	1	1	1	1
SFD	1000		6	4	12,000	2	1	2	2
CF		200			36	200	76	3	3
SF	1000				84	1,000	60	18	18
PF	1000				26		81	62	124
TOTAL NO. OF TRACKS REQD.								96	148

Table 2

The number of tracks required for each file is obtained by using the formula given in IBM DASD book. The DASD used 3330.

The number of records of 80 bytes/record, if blocked is 162/track.

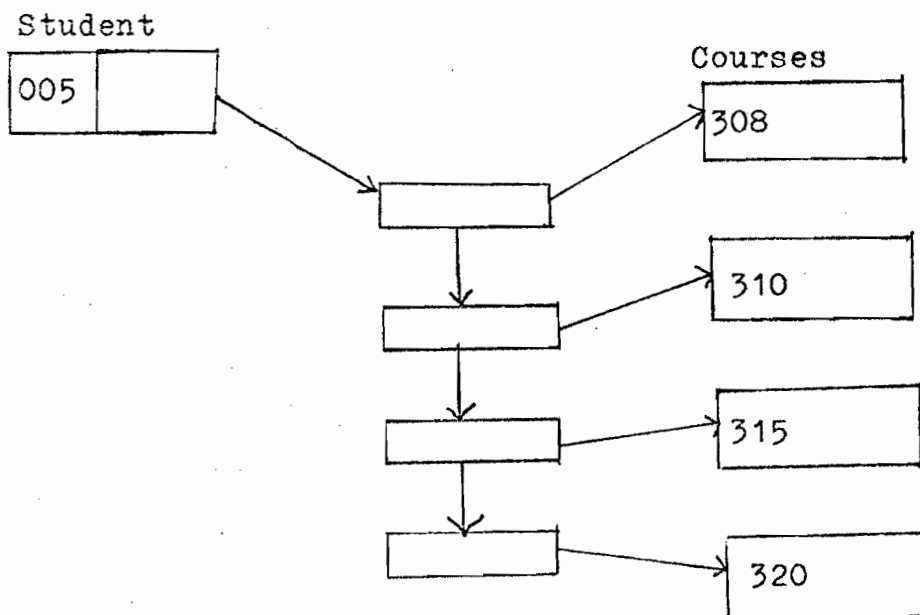
We could use different distributions to find out the average number of courses per student, the average number of students in a class and also the average number of courses added or deleted every semester. However, it may not be appropriate to use various distributions to find out this information.

It is up to the educational institution to decide the minimum and maximum number of students in a class and hence the average will differ in each school. Therefore, the average could be found by using the previous or past couple of years real life data.

As far as the average number of courses per student/semester is concerned the schools require a certain number of credits to be earned for a particular degree. For example, a student is required to earn 120 credits for a Bachelor's degree. Full time students are expected to earn 120 credits over a four year period. This tells us the average number of courses a student should take is 5 per semester to earn 120 credits in four years.

If we decide to use various distributions to find out the average every distribution will give us different figures, anyway. Therefore, for the sake of the cost analysis we have decided to take an arbitrary number of 100 as, average number of students in a class and 5, as average number of courses per student which seems to be quite generally applicable.

The files of the SRMS are set-up on DASD by relative access method, which means any student or course can be accessed directly. However the courses of a student can be accessed only sequentially. Similarly, students in a course also can be accessed only sequentially. We can see from the example below that if we wanted to access a students record 320 we would have to read all his preceding courses till we reached to course 320. We also notice that to reach to a course we have to access two records, ie. performance-record and course-record.



This indicates that if in a course there are 100 students we have to access 200 records to go through the 100 students. Similarly, if a student is taking 5 courses, we have to access 10 records. Thus we can imagine that the data bank consists of 200 course-files of 200 records, (100 students) in each course-file and 1000 student-file

of 5/10 records depending on 1st or 2nd semester, in each student-file.

Now that we have the technique of analyzing the cost of various operations we will carry on with our assumption of 1000 students in a department, 200 courses offered by it, average no. of students in a class is 100 and a student normally takes 5 courses per semester. With this assumption the cost of storage and various other I/O operations will be as follows:

2.3.7 I/O OPERATIONS

The cost of the operations on the organization and maintenance of the databank can be analyzed as updates, deletions, additions, printing grade reports, etc. We define these operations for the basic access unit, the block. The number of accesses in each case is analysed separately. The access is the elementary costing unit for computer operations.

An update is the operations of adding or deleting information in an existing record. For instance, entering students grade in record.

Printing grade reports involve obtaining data from the existing blocks and printing out.

A deletion is the operation of excluding a block of data. For instance, a course dropped.

An addition is the operation of including a new block of data. For instance a new course taken by student.

All the above operations involve retrieving (accessing) blocks from databank.

ANALYSIS OF ACCESSES

The average number of accesses required in each operation can be analyzed separately. The cost for retrieving (accessing) blocks is discussed in the following pages.

INITIAL COST

The initial cost is to set-up the databank on a DASD (Direct Access Storage Device) or add courses during the following semesters to the existing databank. Setting-up databank occurs only once a year, i.e. in the beginning of the year. During the following semester the initial cost is due to the addition of courses.

DATABANK SET-UP

To set-up the databank on the DASD there are several operations involved, such as creating Student-file and its directory, Course-file and its directory, Performance-file, and linking the three files together. The operations are performed by the following computer programs.

The program CR.CRSE.FILE stores the course-records in a relative access file and the directory as a single and the only record in a sequential access file. The number of accesses is the number of courses, plus 1 for storing the directory. The program also uses an external sort before transferring the records to the course-file. For sorting, physical block size is 61 card image records. Thus, the cost for creating the course-file is

$$=((N_c(t) + 1) * C_\alpha) + C_s',$$

where $N_c(t)$ is the number of courses at any time, C_α is the cost/access and C_s' is the cost for sorting.

The program CR.PERF.FILE creates a file of 10,000 dummy records in a relative access file, known as Performance-file. The number of accesses is the number of records created and therefore the cost is

$$=N_p * C_{\alpha} ,$$

where N_p is the number of performance-records and C_{α} is the cost/access.

The program CR.STUD.LINK creates student-file and its directory, stores the student-file in a relative access file, the directory as the first two records in a sequential access file and links the student-records with the performance and course records. The number of accesses involved, is 2 for storing the directory, plus N_s , the number of students times N_c , the number of courses a student can take, plus $2N_c$ for storing the addresses of the first student in each course in Course-records. The factor of two enters because to store the addresses the course-records have to be accessed and then stored back. The program also sorts a file of students records before linking them with Performance and course records. Thus the cost is

$$= ((2+N_s(1+N_c) + 2N_c)C_{\alpha} + C_s)$$

NEW SEMESTER

After the initial set-up of the files, the courses are added to the databank in the following semester. The program sorts the course-records before adding to the data bank. The number of records sorted is $N_s(t)$, the number of

students, times, N_c' the number of courses a student can take. Thus the cost is

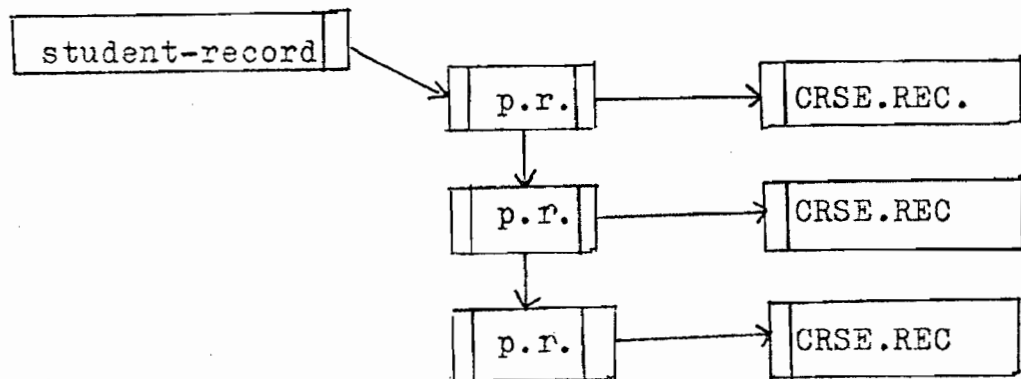
$$= (N_s(t) * Cr) + C_s'$$

where $N_s(t)$ is the number of students, Cr ($Cr = \frac{nr}{r+1}$)

is the retrieval cost and C_s' is the sorting cost. In this case n , in $Cr = \frac{nr}{r+1}$, is

$$= n = 2(N_{cs} + N_{sc}),$$

where, N_{cs} is the number of courses a student takes, N_{sc} is the number of students in a class and the factor of 2 enters because to access a course-record, a performance-record is accessed and then the course-record is accessed. Similarly, to access a student-record, a performance-record is accessed and then the Student-record is accessed.



ADDITIONS

It is quite difficult to estimate the number of course additions every semester. However, we can assume a student will add one course. In this way, some students may not add any and some may add a couple, which will balance it all out. Thus, the cost for course addition is

$$C_A = (N_s(t) * Cr) + C_s'$$

where $N_s(t)$ is the number of students in the department at any time t ; $Cr(Cr = \frac{nr}{r+1})$, the average cost for a batch of requests (accesses) and C_s' , the cost for sorting. In this case also n in $Cr = \frac{nr}{r+1}$ is $n = 2(N_{cs} + N_{sc})$, which is described under new semester.

DELETIONS

In the case of deletions too, we can assume a course deletion for each student each semester, which will permit some students to drop more than one course and some may not drop any. Thus the cost for deletion is

$$C_D = (N_s(t) * Cr) + C_s'$$

where $N_s(t)$ is the number of students in the department at any time t ; $Cr(Cr = \frac{nr}{r+1})$, the average cost for accesses and C_s' is the cost for sorting. In this case also, n in $Cr = \frac{nr}{r+1}$ is $n = 2(N_{cs} + N_{sc})$, as above.

UPDATES

This is normally the cost of entering the grades in the performance-records. The grade-cards are sorted in class order, before they are entered in their respective records. Thus, the cost for entering grades is

$$U = T (N_c(t) * Cr) + C_s',$$

where $N_c(t)$ is the number of courses offered by the department at any time t , $Cr(Cr = \frac{nr}{r+1})$ is the cost for accessing the records, C_s' is the cost for sorting and finally T , is the number of times the grades are entered. In this case the

length of the file, n in $Cr = \frac{nr}{r+1}$ is

$$n = 2 Nsc,$$

where Nsc is the number of students in a course and 2 enters because to access each student-record, a performance-record and then the student-record is accessed. Even though the grades are entered in the performance-records, student-records are read to check if the grades are being entered in the right performance-records.

GRADE REPORTS

The grade-reports are printed out at the end of each semester. In this case no sorting is required. The files are simply read and printed out. The cost for printing out the grade report is

$$GR = Ns(t) * Cr,$$

where $Ns(t)$ is the number of students in the department at any time t , and $Cr(Cr = \frac{nr}{r+1})$ is the average cost for accessing records. In this case the length of the file n is $Cr = \frac{nr}{r+1}$ is

$$n = 2Ncs,$$

where Ncs is the number of courses a student takes and 2 enters because to obtain each course-information two records are read.

2.3.8 COST OF OPERATIONS USING DIFFERENT ACCESS METHODS

After having developed the techniques of cost analysis in the previous sections, we can now cost the various functions of the Student Record Management System using different access methods and storage devices.

The data bank of this system is organized on a Direct Storage Device using Relative Access Method. However, for comparing the cost of this system, another system using Sequential Access Method was also designed, costed and compared.

The section 'Cost of Operations using DAM' computes the cost of storage and the various function, such as create and update files, performed by this system throughout the school year. It also computes the cost of printing out the reports.

The section 'Cost of Operations using SAM' also computes the cost of all the operations performed throughout the school year. However, in this case all the records are organized on a Disk using Sequential Access Method.

Both the systems perform exactly the same functions. The only difference is the organization of the files, and therefore the cost.

2.3.8.1 COST OF OPERATIONS USING DAM

We maintain all the students' current years records on a DASD and the previous years records on tapes, which reduces the storage cost considerably. The cost per track at McGill is .20¢ per week. Whereas cost of storage on tapes is negligible. Therefore, the cost for storage on DASD for the current year is as follows:

1st Sem

The average number of courses per student is 5. Therefore the total number of courses for the first semester is

$$= N_s * N_c'$$

$$= 1000 * 5 = 5000$$

We can see from table 2, the number of tracks required for the first semester is 96. Thus the cost is

$$= N_T(t) * C_d$$

$$= 96 * .20¢ \text{ per week}$$

$$= \$19.20 \text{ per week}$$

$$\therefore \text{1st sem} = \$19.20 * 26 \text{ weeks} \quad \$499.2000$$

2nd Sem

The number of tracks required for the second semester is 148. (see table 2)
Thus the cost is

$$= N_T(t) * C_d$$

$$= 148 * .20¢ \text{ per week}$$

$$= \$29.60 \text{ per week}$$

$$\therefore \text{2nd sem} = \$29.60 * 26 \text{ weeks} \quad \$769.6000$$

2) DATABANK SET-UP

A. CREATE COURSE-FILE

$$\begin{aligned}
 &= (N_c(t) + 1 * C_{\alpha}) + C_s' \\
 &= (201 * .0013) + C_s' \\
 &= .2673 + (2n(1+2) * .00133) \\
 &= .2673 + (2 * 2(3) * .00133) \\
 &= .2673 + .0106 = .2779
 \end{aligned}$$

B. CREATE PERFORMANCE-FILE

$$\begin{aligned}
 &= N_p(t) * C \\
 &= 10,000 * .00133 = 13.3000
 \end{aligned}$$

C. CREATE STUDENT-FILE AND LINK WITH
COURSE-FILE AND PERFORMANCE-FILE

$$\begin{aligned}
 &= (2+N_s(1+N_c') + 2N_c) C_{\alpha} + C_s' \\
 &= ((6002 + 400) * .00133) + C_s' \\
 &= 7.1833 + 2n(1+2)C_{\alpha} \\
 &= 7.1833 + (2*99(3) * .00133) \\
 &= 7.1833 + .7900 = 7.9733
 \end{aligned}$$

3) NEW SEMESTER

$$\begin{aligned}
 &(N_s(t) * C_r) + C_s' \\
 &= 1000 * \left(\frac{nr}{r+1} C_{\alpha} \right) + C_s' \\
 &= 1000 * \left(\frac{210 * 5}{6} * .00133 \right) + C_s' \\
 &= (1000 * .2328) + C_s' \\
 &= 232.80 + (2n(1+2)C_{\alpha}) \\
 &= 232.80 + (2*31(3)*.00133) \\
 &= 232.80 + .1649 = 232.9649
 \end{aligned}$$

The length of the files are 210, because
the five courses are to be added in 2
directions, student-file (5 records)

and course-file (100 records). Two accesses are required to access each record.

4) COURSE DELETIONS -1st SEM

A. For course deletions we have assumed 1000 course deletions every semester.

$$\begin{aligned}
 &= (Ns(t) * Cr) + Cs' \\
 &= (1000 * (\frac{210 * 1}{2} * .00133) + Cs' \\
 &= (1000 * .1397) + (2n(1+2)*.00133) \\
 &= 139.7000 + (2*7(3) *.00133) \\
 &= 139.7000 + .0559 \qquad \qquad \qquad 139.7559
 \end{aligned}$$

B. 2nd SEM

$$\begin{aligned}
 &= (Ns(t) * Cr) + Cs' \\
 &= (1000 * (\frac{220 * 1}{2} * .00133) + Cs' \\
 &= (1000 * .1463) + (2n(1+2) *.00133) \\
 &= 146.3000 + (2*7(3) *.00133) \\
 &= 146.3000 + .0559 = \qquad \qquad \qquad 146.3559
 \end{aligned}$$

During the second semester the length of the file in $Cr = \frac{nr}{r+1}$, is 220 because the average no. of students in a class is 100 and 10 courses per student makes it 110. To access each record, 2 accesses are required, which makes it 220.

5) COURSE ADDITIONS -1st SEM

For course additions too, we have assumed 1000 course additions every semester. During the 1st semester the average number of courses per student is 5. If the course deletions are made first the average number of courses per student left in the file is 4. This means the length of file in $Cr = \frac{nr}{r+1}$ is

$$\begin{aligned}
 n &= 2(N_{cs} + N_{sc}) \\
 &= 2 * (4 + 100) \\
 &= 208
 \end{aligned}$$

Therefore, the average cost of the additions during the first semester is

$$\begin{aligned}
 &(N_s(t) * Cr) + C_s' \\
 &= (1000 * (\frac{208 * 1}{2} * .00133)) + C_s' \\
 &= (1000 * .1383) + (2n(1+2) * .00133) \\
 &= 138.3200 + (2*7(3) * .00133) \\
 &= 138.3200 + .0559 = 138.3759
 \end{aligned}$$

B. 2nd SEM

Similarly, the length of file in $Cr = \frac{nr}{r+1}$, is 218. The average cost is

$$\begin{aligned}
 &(N_s(t) * Cr) + C_s' \\
 &= (1000 * (\frac{218 * 1}{2} * .00133)) + C_s' \\
 &= (1000 * .14297) + (2n(1+3) * .00133) \\
 &= 142.9700 + .0559 = 143.0259
 \end{aligned}$$

6. A) UPDATES -1st SEM

$$\begin{aligned}
 &T(N_c(t) * Cr) + C_s' \\
 &= 3((200 * (\frac{200*100}{101}) * .00133) + C_s') \\
 &= 3((200 * .26334) + C_s') \\
 &= 3((52.6680 + (2n(1+2)C_{\alpha})) \\
 &= 3(52.6680 + (2*31(3) * .00133) \\
 &= 3(52.6680 + .1649) \\
 &= 3 * 52.8329 = 158.4987
 \end{aligned}$$

B) UPDATES -2nd SEM

$$\begin{aligned}
& T(Nc(t) * Cr) + Cs' \\
& = 3((200 * (\frac{nr}{r+1} C_{\alpha})) + Cs') \\
& = 3((200 * (\frac{400*100}{101} * .00133)) + Cs') \\
& = 3((200 * .5267) + (2n(1+2)C_{\alpha})) \\
& = 3(105.3400 + (2*31(3) * .00133)) \\
& = 3(105.3400 + .1649) \\
& = 3 * 105.5049 = 316.5147
\end{aligned}$$

During the first semester the length of the file, i.e. $n=200$, in $Cr = \frac{nr}{r+1}$. This is because the average number of students in a class is 100 but to access each record two accesses are involved.

However, during the second semester $n=400$, because the file of the courses is now doubled, even though the average no. of students in the class are still 100. Therefore, the request is still 100.

7. A) GRADE REPORTS -1st SEM

$$\begin{aligned}
& Ns(t) * Cr \\
& = 1000 * (\frac{nr}{r+1}) C_{\alpha} \\
& = 1000 * ((\frac{10*5}{6}) * .00133) \\
& = 1000 * .01064 \quad 10.6400
\end{aligned}$$

A student takes on the average, 5 courses and therefore, there are five requests. The length of file, $n=10$, because to access a record two records are read.

B) GRADE REPORTS-2nd SEM

$$\begin{aligned}
& Ns(t) * Cr \\
& = 1000 * (\frac{nr}{r+1}) C_{\alpha}
\end{aligned}$$

$$=1000 * ((\frac{20*10}{11}) * .00133)$$

$$=1000 * .02394$$

23.9400

During the second semester a students' file has doubled because of 1st and 2nd semester courses. Therefore, the file is equal to 20, and request 10.

COST FOR ONE YEAR = \$2590.42

COST PER STUDENT = 2.59 per year

2.3.8.2 COST OF OPERATIONS USING SAM

INTRODUCTION

The purpose of the Student Record Management System is to maintain up-to-date students records and print out transcripts at the end of each semester.

The Databank of the SRMS has only one file called Student-file, set-up on a DASD by Sequential Access Method. A student has one record per course, i.e. if he is taking five courses, he has five records in the student-file. Each record contains Student-code, student-name, sex, degree, level, address, telephone number, course-code, course-description, class performance etc.

The main purpose of the SRMS is to maintain students up-to-date records by deletion or addition of courses, entering grades and marks obtained in different classes and printing out the grade reports at the end of each semester. However, the data bank can be used for any other purpose by writing specific programs. These requirements will differ from school to school.

STUDENT-FILE

Each record of the student-file has the following information:

INFORMATION	COLS	BYTES
Dept.	1-3	3
Student code	4-9	6
Student name	10-29	20
Degree	30-33	4
Level	34-35	2
Sex	36	1
Street	37-53	17
Apt.No.	54-57	4
Town	58-73	16
Telephone No.	74-80	7
Course-code	81-86	6
Semester	87	1
Year	88-89	2
Course-description	90-114	25
No. of credits	115	1
Grades Obtained	116	1
Marks Obtained	117-118	2
Record Mark	119	<u>1</u>
	TOTAL	119

At the time of registration two cards per course for each student is punched and stored as one record per course, as described above. Thus each student will have five or six records, depending on the no. of courses he takes. However, the normal load is five courses per semester.

The cost for setting up the databank on a DASD is

$$\left(\frac{Ns(t) * 5}{u} C_{\alpha} \right) + Cs',$$

where, $Ns(t)$ is the number of students at any time, 5 courses is the normal load, u is the number of records per track, C_{α} is the cost per access and Cs' the cost of sorting.

NEW SEMESTER

When the second semester starts each student takes 5 new courses, which should be added to the databank. The databank is organized sequentially and therefore, new records are stored on a Disk-file, the old records added to it, after sorting the new updated file is stored back to its original position. The cost is

$$=(2n_0 + 2n_1(t)) C_{\alpha} + Cs'$$

where, n is the number of physical blocks in the original file, $n_1(t)$ is the addition of blocks, C_{α} , the cost per access and Cs' is the cost for sorting the file.

ADDITIONS

After the school has started the students are allowed to change any course they wish within a certain period. These changes cause additions and deletions of the course records in the databank. However, it is done only once a semester. We will assume one addition course per student, which will account for some students adding a couple and some none. The addition of courses is also handled similar to the new semester course additions. Thus the cost is

$$=(2n_0 + 2n_1(t))C_{\alpha} + Cs'$$

The variables are the same as explained under new semester.

DELETIONS

Changing courses cause additions and deletions of course-records in the databank. We have assumed a student will add a new course, which means he is most likely deleting a course. The number of course deletions, therefore will be approximately equal to the number of students. In the case of course deletions, the cards are sorted on a Disk-file in Student-code order. The Student-file is also sorted in Student-code order. Now both files are read and the Student-file is stored, deleting the records, on its original Position. Thus the cost is

$$=(2n_0 + 2n_1 (t)) C_{\alpha} + C_s'$$

UPDATES

The updates are normally entering grades and marks in the student-file records. The grades are entering^{ed} through data cards. The cards are stored on a disk-file in Course-code order and then the grades are entered by reading the sequential Student-file. However, the student-file is also sorted in Course-code order before reading it. The cost of entering the grades in the Student-file is

$$=T(C_s' + C_r)$$

where T is the number of times the grades are entered, C_s' is the cost of sorting the grades card-file on Disk, and

Cr, is the cost of retrieval of records for entering the grades.

GRADE REPORTS

The grade reports are printed out every semester. The file is sorted in Student-code and Course-code order and then simply reading and printing is performed. The cost for printing grade reports is

$$=T(Cs' + Cr).$$

The variables are the same as described under Updates.

STORAGE

To access the databank when needed, the student-file is stored on direct access storage device. The cost of the file is determined from the number of tracks occupied by the databank which depends on the number of students. Thus the cost is

$$=N_T(t) * C_p,$$

where, $N_T(t)$ is the number of tracks required at any time, and C_p is the cost per track. The present cost per track on IBM 3330 at McGill is .20¢ per week.

Records are blocked 109 per track, and therefore for 5000 records 46 tracks are required.

COST ANALYSIS OF SRMS USING SAM

To analyze the cost of storage and various I/O operations of the SRMS, we must know the number of students in the department using the system. For the purpose of analysis we assume the number of students is 1000 in the department.

With this assumption the cost of the storage and I/O operations will be as follows.

STORAGE -1st SEM

Each student takes five courses per semester.

Therefore, the total number of courses for the 1st semester is

$$=N_s * N_c'$$

$$=1000 * 5 = 5000.$$

The course records are blocked 109/track, and therefore the number of tracks required is

$$=5000/109 = 46.$$

The cost for storage is

$$=N_T(t) * C_g$$

$$= 46 * .20¢ \text{ per week}$$

$$= \$9.20 \text{ per week}$$

$$\therefore \text{1st sem} = \$9.20 * 26$$

$$\$239.20$$

STORAGE-2nd SEM

The cost for 2nd sem is the number of tracks from the 1st sem plus the number of tracks required for the 2nd semester. During the 2nd semester also, each student takes 5 courses, which means the number of tracks required is

$$=46 + 46 =92.$$

The cost for storage is

$$=N_T(t) * C_g$$

$$=92 * .20¢ \text{ per week}$$

$$= \$18.40 \text{ per week}$$

$$\therefore \text{2nd sem} = \$18.40 * 26 =$$

$$\$478.40$$

NEW SEMESTER

The cost for adding 5 courses for each student in databank is

$$\begin{aligned}
 &= (2n_0 + 2n_1(t))C_{\alpha} + C's \\
 &= (92 + 92 * .00133) + C's \\
 &= (.2447 + C's) \\
 &= (.2447 + (2n(1+2)C_{\alpha})) \\
 &= (.2447 + (2*92(3)*.00133)) \\
 &= (.2447 + .7342) = 0.98
 \end{aligned}$$

DELETION-1st SEM

In this case $n_1(t)$ is 7, whereas n_0 , is 46. The cost is

$$\begin{aligned}
 &= (2n_0 + 2n_1(t))C_{\alpha} + C's \\
 &= (92 + 14) * .00133 + C's \\
 &= (.1410 + (2n(1+2)C_{\alpha})) \\
 &= (.1410 + (2*53(3)*.00133)) \\
 &= (.1410 + .4229) = .4229
 \end{aligned}$$

DELETION-2nd SEM

In this case $n_1(t)$ is 7, whereas n_0 , is 92. The cost is

$$\begin{aligned}
 &= (2n_0 + 2n_1(t))C_{\alpha} + C's \\
 &= (184 + 14)C_{\alpha} + C's \\
 &= (.2633 + (2n(1+2)C_{\alpha})) \\
 &= (.2633 + (2*99(3)*.00133)) \\
 &= (.2633 + 0.7900) + \$1.0533
 \end{aligned}$$

ADDITION-1st SEM

Initially n_0 was 46, but after deleting 1000 courses (7 tracks), n_0 is now only 39, and $n_1(t)$ is 7. The cost for addition is

$$\begin{aligned}
& 137 \\
& = (2n_0 + 2n_1(t)C_{\alpha} + C'_s) \\
& = (78 * .00133) + C'_s \\
& = (.1037 + (2n(1+2)C_{\alpha})) \\
& = (.1037 + (2*46(3) * .00133)) \\
& = .1037 + .3671 \qquad .4708
\end{aligned}$$

ADDITIONS -2nd SEM

After deleting 1000 courses (7 tracks), n_0 is only 85 tracks and $n_1(t)$ is 7 tracks. The cost for adding 1000 courses during the 2nd semester is

$$\begin{aligned}
& = (2n_0 + 2n_1(t)C_{\alpha} + C'_s) \\
& = (170 + 14 * .00133) + C'_s \\
& = (.2447 + (2n(1+2)C_{\alpha})) \\
& = (.2447 + (2*92(3) * .00133)) \\
& = (.2447 + .7342) = .9789
\end{aligned}$$

UPDATES-1st SEM

Assuming the grades are entered 3 times a semester, the cost for entering 5000 grades is computed from the cost of sorting grade cards and student-file records, and the retrieval cost for entering grades. The number of tracks required for 5000 grade cards is

$$5000/80 \approx 32,$$

which is used to compute the cost of sorting. Thus the cost for entering grades is

$$\begin{aligned}
& T(C'_s + Cr) \\
& = 3((2n(1+2)C_{\alpha}) + Cr) \\
& = 3((2*32(3) * .00133) + Cr) \\
& = 3(.2554 + (\frac{nr}{r+1} C_{\alpha})) \\
& = 3(.2554 + (\frac{46*5000}{5001} * .00133)) \\
& = 3(.2554 + .0612) = 3 * .3166 = .9498
\end{aligned}$$

UPDATES-2nd SEM

During the 2nd semester the student-file will be twice as big as the first semester, and therefore, it will occupy 200 tracks. However, the tracks required for sorting the grade cards will be the same i.e.

84. Assuming the grades will be entered 3 times, the cost is

$$\begin{aligned}
 &T(Cs' + Cr) \\
 &= 3((2n(1+2)C_{\alpha}) + Cr) \\
 &= 3((2*32(3) * .00133) + Cr) \\
 &= 3(.2554 + (\frac{nr}{r+1} C_{\alpha})) \\
 &= 3(.2554 + (\frac{92 * 5000}{5001} * .00133)) \\
 &= 3(.2554 + .1224) \\
 &= 3 * .3778 = \qquad \qquad \qquad \$1.1334
 \end{aligned}$$

GRADE REPORTS-1st SEM

The cost for printing grade reports during the 1st semester is

$$\begin{aligned}
 &T(Cs' + Cr) \\
 &= 1((2n(1+2)C_{\alpha}) + Cr) \\
 &= (2*46(3) * .00133) + Cr \\
 &= .3671 + (\frac{nr}{r+1} * C_{\alpha}) \\
 &= .3671 + (\frac{46 * 5000}{5001} * .00133) \\
 &= .3671 + .0612 = \qquad \qquad \qquad .4283
 \end{aligned}$$

GRADE REPORTS-2nd SEM

Assuming the grade reports are printed out only once during the second semester, the cost is

$$\begin{aligned}
 &T(Cs' + Cr) \\
 &= 1((2n(1+2)C_{\alpha}) + Cr) \\
 &= (2*92(3) * .00133) + Cr
 \end{aligned}$$

$$\begin{aligned}
 &= .7342 + \left(\frac{nr}{r+1} C_q \right) \\
 &= .7342 + \left(\frac{92 * 5000}{5001} * .00133 \right) \\
 &= .7342 + .1224 = .8566 \\
 \text{TOTAL} & \qquad \qquad \qquad \underline{\$724.8740}
 \end{aligned}$$

The cost for maintaining a students up-to-date records per year is $724.87/1000 \approx .72$ per student/year.

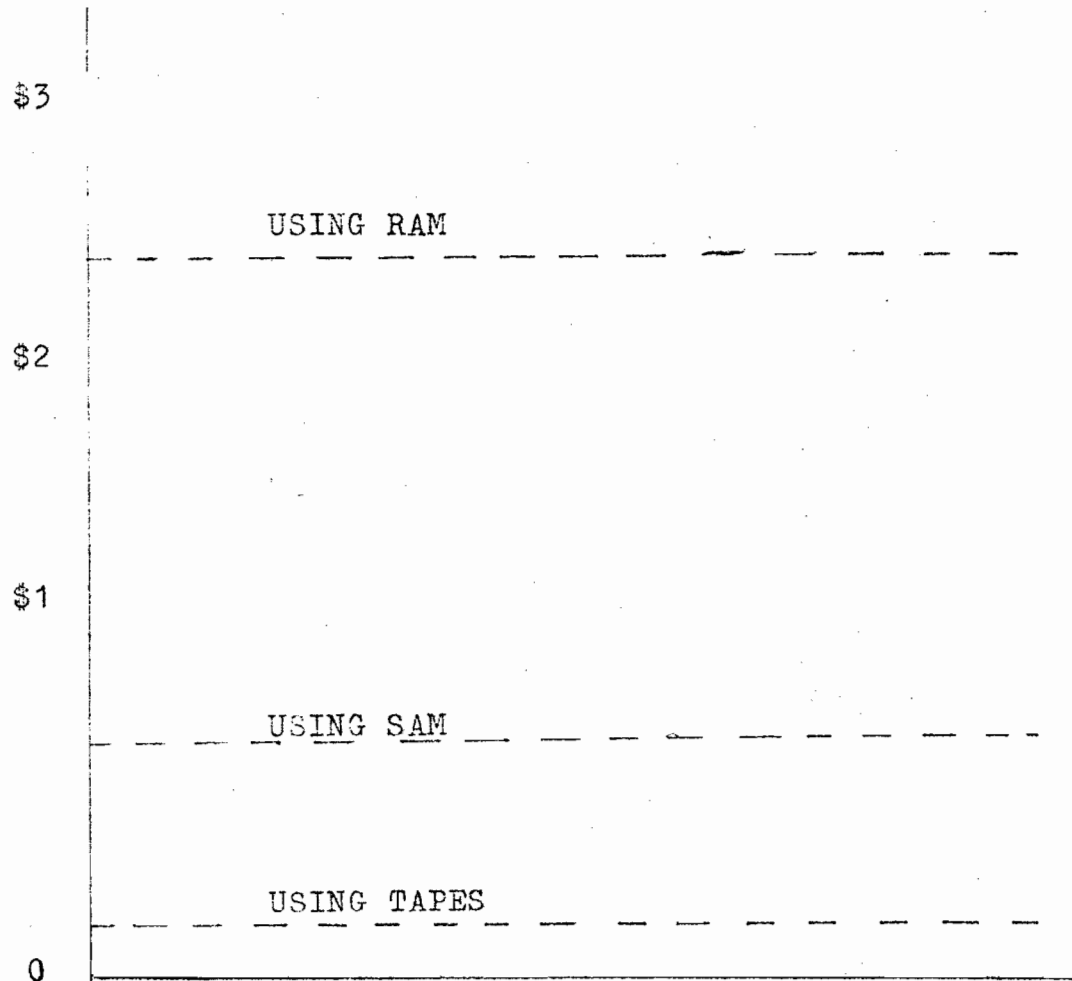
However, if the data is stored on tapes instead of on-line device, the cost is negligible which is:

$$\begin{aligned}
 &\$7.27/1000 \qquad \qquad \qquad .01 \\
 &\text{per student/year.}
 \end{aligned}$$

2.3.9 CONCLUSION

We can see from the cost analysis of the two SRMS's, the main cost^{is} for on-line storage. RAM is more expensive than SAM on DASD. If we use tapes instead of on-line storage, the cost of SRMS is negligible. However, with the tapes, only SAM can be used. We should not forget that we have not cost analyzed many items, like part of the CPU cost, personnel key punching, unit record, papers or forms used for output, etc. Since these items are common for both the systems, it will just increase the cost of both the systems by the same amount.

In short, comparisons of the SRMS's using RAM and SAM shows SAM is quite cheaper. Moreover, if tapes are used the cost goes down to the bottom, which is shown below.



2.4 COMPARISON WITH ALTERNATIVE SYSTEMS

2.4.1 MCGILL UNIVERSITY SYSTEM

The Student Record Management System could be very simple and yet meet all the requirements of the institution. We can look at the McGill Student Record Management System which is quite simple and meets all their requirements.

McGill University maintains the students records on tapes. Each student has two records for the current year, of 750 bytes each, one containing his academic information, such as courses, grades, marks, semester, etc. and the other record containing his personal information such as name, local address, permanent address, telephone number, sex, etc. The reason for having so big records is that they allow enough space for storing up to 20 course information.

The previous year's academic and personal information is stored on a different tape, which has variable length records. The length of the records depends on the amount of the information of the students. Normally, the information is added or changed only on the current tape, and therefore, the length of the records have to be big enough to be able to store information in them.

When students register, the forms are sent from each department to the main office, where academic and personnel information is punched on cards and stored on the tape. The two records per students are fixed length of 750 bytes each and therefore, there is still a lot of space reserved for each student on the tape.

Similarly, the course changes are also sent by all the departments to the main office, where the cards are punched and the tape updated. The grades are also handled the same way.

To meet the requirements of each department several different reports are being produced from these tapes. The grade reports for the students are also being printed out from this tape.

We can see the data structure of the McGill system is simply sequential and on tape. The length of the records is 750 bytes, whether it is used or not and ⁱⁿ most cases we will see, it is not used, since there are not many students who will take 20 courses. SRMS is organized on DASD with links to the connecting records. This means, if a student is taking only 3 courses he has only 3 records of 22 bytes each. When he takes the 4th course one more record of 22 bytes gets attached to his other records. Thus, the students have the exact amount of space they need.

In McGill System, all the records are sequential which means every time different reports are required, the tapes are to be sorted before each report is printed out, whereas in SRMS the databank is linked in two orders, i.e. courses within students and students within courses. Most of the reports can be printed out without any sorting.

However, we may say that McGill System has information on tapes, which is much cheaper than DASD and the cost of sorting is also very cheap. This is true, but then SRMS's

files can also be transferred from DASD to tape after use. The SRMS has student-file, course-file, performance-file, student-file-directory and course-file-directory, which could be stored as five separate sequential files on a tape and every time used can be stored on DASD and then transferred back on the tape.

However, if the student information is on DASD the system keeps track of the different files. No tape handling is required, which eliminates human errors such as submitting wrong tape with program. If a couple of wrong reports are produced in a year this will justify the extra cost for using DASD. If the institution is not using their own computer, they have to pay a certain amount, like \$2.00 every time the tape is required to be mounted. It costs to transfer the data from tape to disks to produce a report and then back to tape.

2.4.2 CASSARMS

The System Cassarms, the Computer Assisted Student Scheduling and Record Management System, is being used by USDESEA, United States Dependent Schools in European Area. The system is based on a number of computer programs developed by USDESEA personnel, supplemented by programs supplied by the IBM Corporation. The Scheduling phase of the system utilizes the IBM programs, while the student record management phase applies the USDESEA programs in a logical sequence. We shall discuss only the student record management phase of CASSARM.

The scheduling phase of CASSARMS produces a student schedule tape from which a new card for each student, listing the course-code to which he is assigned, is produced. These cards are called L-deck, which lists the students actual course assignments. If a student has six or fewer course assignments, he has one L-card; if he has more than six he has two L-cards. The system will accommodate a maximum of twelve course assignments.

These cards updated manually as course changes are made form the bases for different reports. The department has the listing of the course corresponding to the course codes, from which the course change forms are filled out and sent to the Key punch Dept. The Key punch Dept will update the computer cards.

The Master deck containing course-code, course-description, section, period, instructor, class rooms/etc, prepared by the School Administration for the scheduling phase is used through out the student record management phase. This Master deck is known as the G-deck.

Just prior to a grading period this G-deck and L-deck are processed by the computer and one card for each course a student takes, is produced in the course-code order. This new deck is known as E-deck. An "E" card depicts a given course in which a student is enrolled. He has one "E" card for each course, containing all the information necessary to define the particular class and section. The "E" cards are given to the teachers for marking grades. The teachers mark on each card the grade received for that marking period, sorts the cards grade-wise and returns to the keypunch department. The keypunch department has card

boxes marked with A,B,C,D,E,F,etc. where cards are placed in the appropriate boxes. After all the cards are received the grades are punched on the same cards.

The deck of cards is now ready for various reports, such as class-lists with grades on them, grade reports for mail-ing to the students, grade reports on gum labels for their permanent record card,etc. The students are graded twice a semester and the same computer cards are used for the second grading period too. For the second semester a new set of cards is produced using the L-deck and the G-deck of the second semester.

2.4.3 CONCLUSION

The Student Record Management System will maintain up-to-date records of the students. It updates the various files to accommodate the changes made by the students in their programs.

The system has nine computer programs to maintain the files and print out reports. The section "Using the System" describes how to use the programs to perform the various functions. The system is quite simple and to perform any function a cataloged procedure can be invoked. Only a few cards are required to call any cataloged procedure which is discussed in the above section.

In the databank the connecting records are linked by embedded pointers. This provides faster access to the logically connected records.

The Data Base Task Group has proposed a Data Description Language and a Data Manipulation Language to manage the database efficiently. These languages have been applied in a few programs of the Student Record Management System to determine if it falls on the lines of the proposals. It can be seen in the section "Implementation of SRMS in DBTG" that the database of the SRMS does fall on the lines of the DBTG's proposals. It has pointers in different directions to all the connecting records. There is no redundancy of data. It has a

centralized database, parts of which are used by different programs for performing the various functions. It is organized on a Direct Access Storage Device. However, to maximize the advantages of this database structure, it should be interrogative, using TSO or some other method. This will permit retrieving and entering specific information via a terminal device when it is required.

The section "Cost Analysis Techniques" indicates that the maintenance of the databank, such as above, is quite expensive. This is mainly because of the expensive storage media and the high cost of updating the integrated database. However, the advantages of the integrated database justifies the extra cost.

References:

1. Compiler Construction for Digital Computers
by David Gries.
2. The Art of Computer Programming Vol.III
by Donald E Knuth.
3. Unpublished paper on "Cost Analysis Techniques"
by Professor Tim H. Merrett.
4. Data Structures and Programming
by Malcolm C. Harrison.
5. Data Base Task Group (Codasyl)
Report of April 1971.

SORTING AND SEARCHING TECHNIQUES

The natural physical characteristics of conventional memory make it more suitable for representing ordered sets than unordered sets. The elements of a set must be ordered according to some property of the elements, which permits us to locate a particular item with greater ease than if the elements are not ordered. There are several sorting techniques and any of them can be used to order a set. Unfortunately, there is no known best way to sort: there are many best methods, depending on what is to be sorted on what machine, for what purpose. In words of Rudyard Kipling, "There are nine and sixty ways of constructing tribal laws, and every single one is right." Some of the common sorting techniques are discussed below:

SEQUENTIAL SORT

We will suppose that an item contains a key which is to be used to order the items. The items are to be sorted so that their keys are in increasing order. The simplest method of doing this is to search the whole sequence for the smallest item and then to exchange it with the first position in the list. Then the list except for this smallest item is searched for the next smallest and this is exchanged with the second item on the list, etc..

This is simple to program, but it has the disadvantage that, for a sequence of n items, it requires $n(n-1)/2$ tests.

It also has the disadvantage that it will take just as long, even if the list is in the correct order initially.

BUBBLE SORT

This sort starts by comparing the first two items in the sequence. If these are in incorrect order, they are interchanged. The procedure then goes on to examine the second and third items in the sequence. In general, if the i -th and $(i+1)$ -st elements are in incorrect order, the lower one is moved upwards until it is in the correct position. Otherwise the comparison continues with the $(i+1)$ -st and $(i+2)$ -nd elements until the end of the list is reached. The whole procedure is repeated until a pass is found without a pair out of order. Bubble sort, in the worst case will require $n(n-1)/2$ tests and exchanges, while in the best case, it will require $(n-1)$ tests and zero exchanges.

MERGE SORT

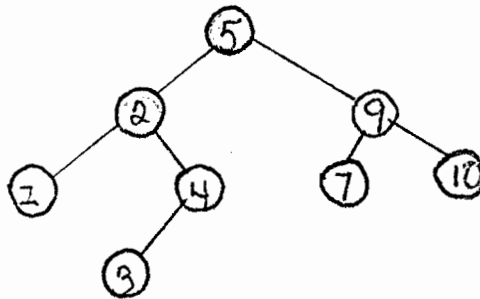
The essential idea of the merge sort is that two ordered sequences of lengths m_1 and m_2 can be merged to provide a completely ordered sequence length $m_1 + m_2$, using approximately $m_1 + m_2$ moves and less than $m_1 + m_2$ tests. This can be accomplished simply by removing at each step the smallest element of either of the two lists onto a merged list. The complete merge sort can then be accomplished by first converting the sequence of length n into approximately $n/2$ sequences, each of which is of length 2 and in which

the elements are ordered. The sequences are then merged in pairs to form approximately $n/4$ sequences of length 4, etc. Thus the total number of comparisons required to sort a sequence of length n is approximately $n \log_2 n$.

The merge sort is very efficient when it is possible to collect all the items to be sorted before the sorting operation begins. In some situations, it is necessary to sort a sequence of items and then to add a number of further items to the sequence in their appropriate position. If a linked representation is satisfactory, then the new items can be inserted without moving the remaining items. However, finding the position for the insertion will on average require $n/2$ tests, when n is the number of items in the sequence. This can be improved considerably if the sorted sequence is stored as a tree structure.

BINARY TREE SORT

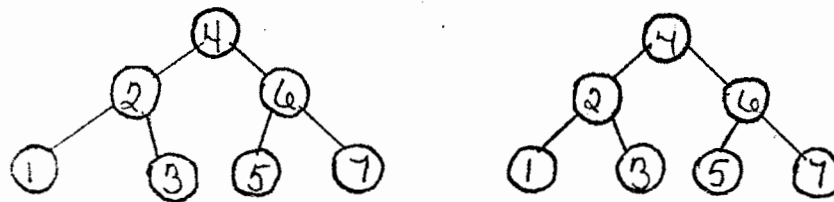
The binary tree of l levels can have $2^l - 1$ nodes. Each node can be used to store an element, so $2^l - 1$ elements can be located by tracing down l links. A sorted binary tree is built in such a way that both its left and right subtrees are sorted, and all those items in the left subtree occur before the items in the node, and all the items in the right subtree occur later. For example the following tree is sorted.



In this example, some of the subtrees contain one or zero branches. This tree has four levels and could contain $2^4 - 1$ items, which is equal to 15 items. Inserting a new item in a sorted binary tree is simple. It is added as a new terminal node, whose position in the tree is obtained by tracing down the tree, taking each node if the item is before the node item in the required order, and the right branch if it is after the node item. For example, the number 6 would be inserted to the right of 5 and the left of 7.

A new item can be inserted using l tests. However, if the tree is completely unbalanced, that is, if the entries are made in it in such an order that each node specifies at most one subtree, then we have effectively generated a table

which is searched linearly, and will thus require an average of $n/2$ tests per insertion. For maximum efficiency it is necessary to keep the tree as balanced as possible. In some cases this will require that the tree be reorganized, putting different nodes at the head of subtrees, and re-organizing the subtrees accordingly. Thus for 7 items the first item should be 4th, and the second item should be either second or sixth. Both the order 4,2,6,1,3,5,7 and the order 4,2,1,3,6,5,7 can be used.



However, to rebuild the tree from top down is not trivial. It is much easier to build it from the bottom up, constructing a subtree with the first 3 items on it, and then a subtree with the 5th-7th items and joining these using the 4th.

RADIX SORT

The radix sort is a modification of the procedure used by physical card sorters. It is convenient when the key on which sorting is to be done^{is} relatively short. The procedure takes the form of sorting the items first on the least significant symbol. The result is then sorted on the second least significant symbol, then on the third, and so on.

For example, if we were to sort 51,40,60,80,20, after the first pass we would have 40,60,80,20,51 and after the second pass 20,40,51,60,80. If the numbers 1,2,3,...4096 are to be sorted it will take 4 passes.

This is much better then the merge sort which takes of the order $4096 \log_2 4096 = 12 \times 512$ comparisons.

SEARCHING

For locating a particular item of a set we will assume that an item has a key which will identify it. There are several searching methods available, but some of the common methods are discussed below:

SEQUENTIAL SEARCH

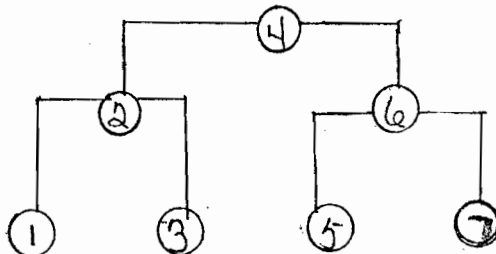
The simplest search procedure is the linear search, in which all the items are examined in turn until the correct one is found. This can be used when the items are stored sequentially in memory, or in a chain, but it has a disadvantage that locating a random item will require an average $n/2$ attempts where n is the number of items.

However, there are certain situations where this process can be effective. If all items are not searched for equally often, the more frequent ones can be placed at the beginning. Also this process is so simple that it can easily be implemented in the hardware, thus speeding up the search. An example of this is found in the address search in the CDC STAR

machine, which looks up all addresses in a table to find out the physical memory location assigned to the block of memory in which the address lies. The top 16 entries in the table, are searched in parallel, but if the block address is not found the table is searched linearly by hardware at a great speed. When an address is found, it is moved to the top of the table, and the entries above it are moved down one place. Since the address generated by a program tend to be clustered, most addresses will be found in the top 16 entries, and little time will be spent in the linear search.

BINARY SEARCH

If the items are sorted, for instance, retrieval can be done from n items by using $\log_2 n$ tests, which is known as a binary search. In binary search, the first item to be examined in the table is the item in the middle. By comparing this with the object found, it can be determined if the object is in the first half of the table or the second. Subsequent tests on items at the centre of the appropriate half of the table will determine which quarter of the table the particular object is in, etc. A binary search of 7 items is illustrated below. The 4 is compared with the search argument. Depending on the result, the comparison is performed with 2 or 6 again, and so on.



This procedure does have the disadvantage that all elements in the table must be in sorted order, and any insertions into the table require resorting.

HASH ADDRESSING

Hash coding has applications in a number of areas including inserting and searching items. This is a method for converting items or its key to indexes of entries in the table (the entries are numbered $0, \dots, N-1$ where the table has N entries). The index is obtained by "Hashing" the symbol by performing some simple arithmetic or logical operation on the symbol and possibly its length. As long as two symbols do not hash to the same index the cost of a search is just the cost of doing the hashing. Trouble occurs however, if two symbols hash to the same index, which is called collision. Only one symbol can be placed at that entry, so we must find another spot for the second. We can use chaining or rehashing to solve the collision problem.

The chained hash addressing technique uses a hash table whose elements, called buckets, are initially zero. The symbol table itself which is initially empty and a pointer which points to the current last entry in the symbol table initially points to the location before the beginning of the table. The symbol table has another chain field, which may contain zero or an address of another entry in the symbol table. The initial tables look like this:

HASH TABLE
Buckets

1	0
2	0
3	0
4	0
5	0
6	0

SYMBOL TABLE

ARG	CHAIN

POINTER

Each bucket is zero or points to the symbol table entry for the symbol which hashed to it. The chain field of each entry is used to chain entries whose symbols hash to the same bucket. Suppose that a symbol, S1 is to be entered into the symbol table. The hash function produces the address of a hash table entry, for example bucket 4, which at this point is zero. We do the following.

1. Add 1 to pointer.
2. Insert the value S1 and zero in chain column into the symbol table entry pointed at by pointer.
3. Put pointer in bucket 4.

Now if we want to enter symbols S2, S3 and S4 which hash to buckets 1,3 and 6 respectively, the tables would look like this:

HASH TABLE

Buckets

1	
2	0
3	
4	
5	0
6	
7	0

SYMBOL TABLE

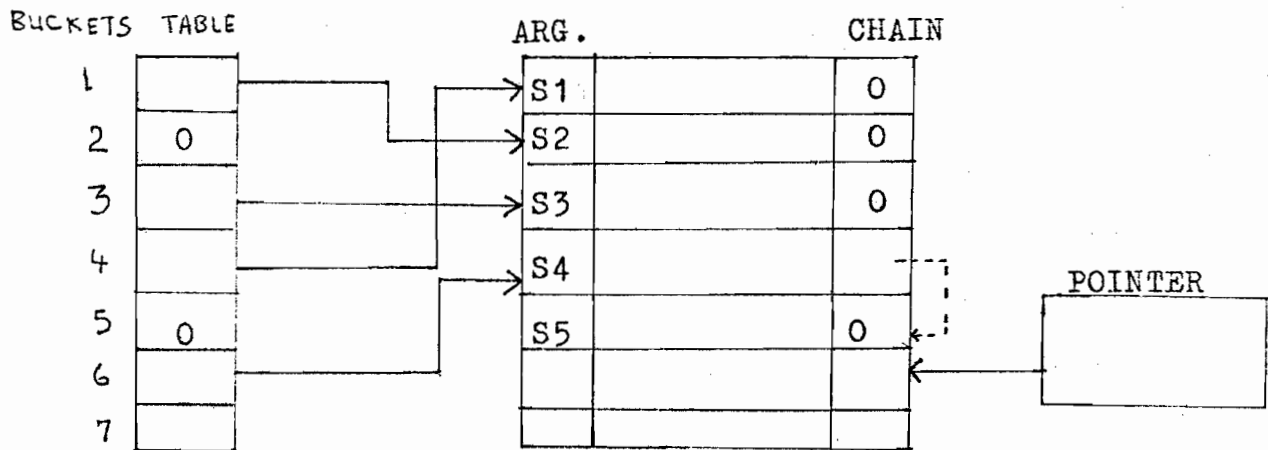
ARG	CHAIN
S1	0
S2	0
S3	0
S4	0

POINTER

However, when a symbol S5 will be entered which hashes to a bucket which has been used before, the chain field comes into play. S5 will be entered into the symbol table and added to the end of the chain for that bucket. Thus if S5 hashes to bucket 6, we have the following structure.

HASH TABLE

SYMBOL TABLE



In this procedure only the buckets need to be initialized not the entries themselves. The actual symbol table entries should be more than the buckets. Once all entries have been entered the hash tables can be thrown away and its space released for other purposes. For retrieval of an item also, hashing is used to generate the index in the table, which is then searched.

HASHING FUNCTION

There are a number of ways of obtaining such indexes (Hash Code).

1. Multiply the key by itself and use the middle n bits as the hash(if the table has 2^n entries). The middle n bits depend upon every bit of key.
2. Use some logical operation, such as EXCLUSIVE OR, on certain parts of the key.
3. If there are 2^n entries in the table, split the key up into n bit sections and add them together. Use the rightmost n bits of the result.
4. Divide the key by the size of the table and use the remainder as the hash index.

There could be other methods devised besides the above. However, these methods give satisfactory results.

LINEAR REHASH

In this method if the collision occurs the item will be stored in the next sequential entry. In the example below symbols S1 and S2 were hashed and entered at entries 2 and 4 respectively (fig. A). Suppose now that symbol S3 also hashes to entry 2. Because of the collision it will be stored in entry 3(fig.B). Finally, suppose the next symbol S4 also hashes to entry 2. There will be 3 collisions with S1, S3 and S2 in that order- before S4 is finally stored at the 5th entry (fig.C)

A		B		C	
1		1		1	
2	S1	2	S1	2	S1
3		3	S3	3	S3
4	S2	4	S2	4	S2
5		5		5	S4

An approximation to the average number E of comparisons necessary to search for an item is

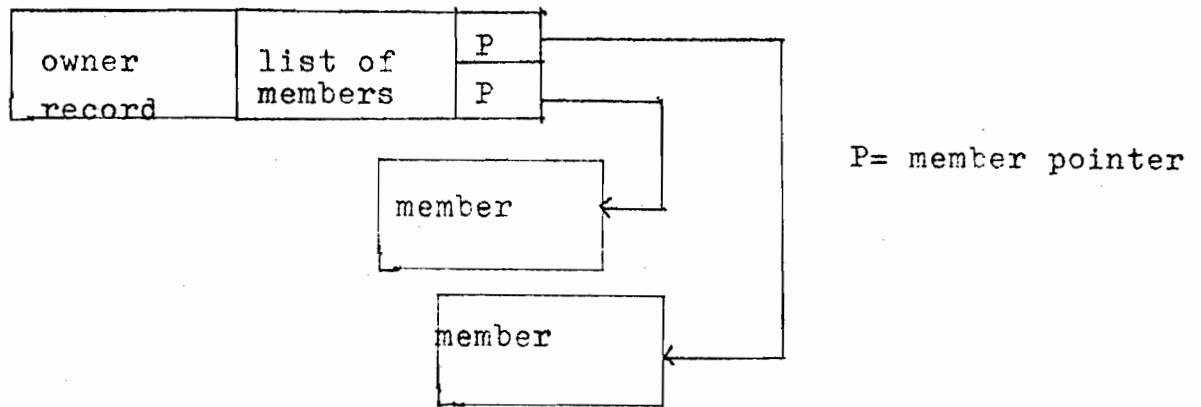
$$E = (1 - lf/2)(1 - lf)$$

where, lf is the load factor, i.e. current number of entries n divided by the maximum number of N entries possible ($lf = n/N$). This method is not very efficient but still much faster than the binary tree search. Suppose a table of 1024 entries is half-filled. Thus 512 entries are filled. In binary search we expect 9 to 10 comparisons while here we expect only 1.5. The search time for sorted or unsorted tables depends not on the maximum size of the table but on the current number of entries. Thus if the table is 10% full, we would expect 1.06 comparisons; if half full 1.5 comparisons; and if 90% full 5.5, comparisons

CONCLUSION

The straightforward implementation of a set is very inefficient. The elements must be in some order which will make the searching of the elements easier and efficient. However, implementing sort and search routines differ from set to set. Let us consider the database of Student Record Management System using Set concept. In our case, we have four sets.

The Student-Info-Set and the Course-Info-Set are desired to have the order in which the members can be accessed sequentially or randomly. Therefore, these sets are defined as ^{of} mode Pointer-Array. It looks like the structure on the next page:



It does not matter where the members are physically located in the set. However, in the array the pointers must be sorted to be able to access the members sequentially. From our discussion about sorts and searches, we find that the Radix sort will be quite suitable for sorting the array. Since the key, in our case, is 6 digits long, it will require only 6 passes and no comparisons to sort the array. For accessing members, binary sort can be applied, which requires $\log_2 n$ tests. However, if the members were not desired sequentially, Hash Addressing could have been used for entering and searching members in the set, which does not require sorting, and search too is faster than binary search.

The Student-Perf-Set and Course-Perf-Set are not sorted. They have embedded pointers, the members of which point to the next member in the set. The search is therefore, sequential. The physical location of the members could be anywhere in the set.

CREATE COURSE-FILE

IDENTIFICATION DIVISION.

PROGRAM-ID. CREATES. .

AUTHOR. GOVIND K R I P L A N I.

REMARKS. THIS PROGRAM CREATES COURSE-FILE ON DISK.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-360-F75.

OBJECT-COMPUTER. IBM-360-F75.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT CARD-FILE ASSIGN TO UR-2540R-S-CARDS.

SELECT PRINT-FILE ASSIGN TO UR-1402-S-PRINTS.

SELECT DISK-IN ASSIGN TO UT-2314-S-INDISK.

SELECT COURSE-FILE-DIRECTORY ASSIGN TO UT-2314-S-CFD.

SELECT WORK-FILE ASSIGN TO 4 DA-2314-D-SORTWK01.

SELECT INTER-FILE ASSIGN TO UT-2314-S-CLASSDIR.

SELECT COURSE-FILE ASSIGN TO DA-2314-R-SRF

ACCESS IS SEQUENTIAL

NOMINAL KEY IS NOM-KEY.

DATA DIVISION.

FILE SECTION.

SD WORK-FILE

RECORDING MODE IS F

RECORD CONTAINS 80 CHARACTERS

LABEL RECORD IS STANDARD

DATA RECORD IS WORKS.

01 WORKS.

02 DEPT-WORK PICTURE XXX.

02 STUD-NO-WORK PICTURE 9(06).

02 SEM-WORK PICTURE X.

02 GRADE-WORK PICTURE X.

02 MARK-WORK PICTURE XX.

02 YEAR-WORK PICTURE XX.

02 COURSE-CODE-WORK PICTURE X(06).

02 FILLER PICTURE X(58).

02 COL-80-WORK PICTURE X.

FD CARD-FILE

RECORDING MODE IS F

RECORD CONTAINS 80 CHARACTERS

LABEL RECORD IS OMITTED

DATA RECORD IS CARD-REC.

01 CARD-REC.

02 FILLER PICTURE X(79).

02 COL-80 PICTURE X.

FD PRINT-FILE

RECORDING MODE IS F

LABEL RECORD IS OMITTED

DATA RECORD IS LINE-FORMAT.

01 LINE-FORMAT.

02 FILLER PICTURE X(133).

```

FD  DISK-IN
    RECORDING MODE IS F
    RECORD CONTAINS 90 CHARACTERS
    BLOCK CONTAINS 10 RECORDS
    LABEL RECORD IS STANDARD
    DATA RECORD IS DISK-IN-REC.
01  DISK-IN-REC.
    02  FILLER PICTURE XXX.
    02  STUD-NO-IN PICTURE 9(06).
    02  YEAR-IN PICTURE XX.
    02  SEM-IN PICTURE X.
    02  FILLER PICTURE X(67)..
    02  CDL-30-IN PICTURE X.
FD  INTER-FILE
    RECORDING MODE IS F
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORD IS STANDARD
    DATA RECORD IS INTER-REC.
01  INTER-REC.
    02  FILLER PICTURE XXX.
    02  COURSE-I PICTURE 9(06).
    02  SEM-I PICTURE X.
    02  COURSE-DES-I PICTURE X(25).
    02  CR-I PICTURE X.
    02  FILLER PICTURE X(44).
FD  COURSE-FILE
    RECORDING MODE IS F
    RECORD CONTAINS 37 CHARACTERS
    LABEL RECORD IS STANDARD
    DATA RECORD IS COURSE-REC.
01  COURSE-REC.
    02  FILLER PICTURE 9(04).
    02  CRSE-KEY PICTURE 9(06).
    02  FILLER PICTURE X(27).
FD  COURSE-FILE-DIRECTORY
    RECORDING MODE IS F
    RECORD CONTAINS 2400 CHARACTERS
    LABEL RECORD IS STANDARD
    DATA RECORD IS COURSE-FILE-DIRECT.
01  COURSE-FILE-DIRECT.
    02  FILLER PICTURE X(2400).
WORKING-STORAGE SECTION.
77  NON-KEY PICTURE S9(8) COMP SYNC.
77  CN PICTURE 9(06) VALUE 0.
77  IND PICTURE 9(06) VALUE ZEROS.
01  COURSE-CC.
    02  ADDR-C PICTURE 9999 VALUE ZEROS.
    02  COURSE-C PICTURE 9(06).
    02  SEM-C PICTURE X.
    02  COURSE-DES-C PICTURE X(25).
    02  CR-C PICTURE X.
01  CFD-WRK.
    02  CRSE-DIRECT OCCURS 200 TIMES.

```



```

03 CRSE-D PICTURE 9(06).
05 INDX-D PICTURE 9(06).
PROCEDURE DIVISION.
STARTS.
    OPEN INPUT CARD-FILE OUTPUT DISK-IN.
READ-CARDS.
    READ CARD-FILE AT END GO TO CLOSE-CODIN.
    IF COL-80 IS EQUAL TO 'S' WRITE DISK-IN-REC FROM CARD-REC
    GO TO READ-CARDS. DISPLAY CARD-REC 'WRONG CARD' GO TO
    READ-CARDS.
CLOSE-CODIN.
    CLOSE CARD-FILE DISK-IN.
SORT-1.
    SORT WORK-FILE
        ON ASCENDING KEY STUD-NO-WORK
        USING DISK-IN
        GIVING INTER-FILE.
CHECK-SORT-1.
    IF SORT-RETURN IS NOT EQUAL TO 0 DISPLAY
    'SORT-1 UNSUCCESSFUL' GO TO WRAP-UP.
    DISPLAY 'SORT - 1 OK'.
OPEN-INTER-1.
    OPEN INPUT INTER-FILE OUTPUT COURSE-FILE
    COURSE-FILE-DIRECTORY.
TRANSFER.
    ADD 1 TO CN MOVE CN TO NOM-KEY.
    READ INTER-FILE AT END GO TO WRAP-UP-1.
    MOVE COURSE-I TO COURSE-C MOVE SEM-I TO SEM-C MOVE
    COURSE-DES-I TO COURSE-DES-C MOVE CR-I TO CR-C
    DISPLAY COURSE-CC. WRITE COURSE-REC FROM COURSE-CC
    ADD 1 TO IND MOVE COURSE-I TO CRSE-D (IND) MOVE IND TO
    INDX-D (IND) GO TO TRANSFER.
ERRORS.
    DISPLAY COURSE-CC 'RECORD NOT WRITTEN' GO TO TRANSFER.
WRAP-UP-1.
    ADD 1 TO IND MOVE ZEROS TO INDX-D (IND) MOVE 0 TO IND WRITE
    COURSE-FILE-DIRECT FROM CFD-WORK.
    ADD 10 TO CN MOVE CN TO NOM-KEY MOVE ZEROS TO COURSE-C
    WRITE COURSE-REC FROM COURSE-CC.
    CLOSE INTER-FILE COURSE-FILE COURSE-FILE-DIRECTORY.
    DISPLAY ' '.
    DISPLAY ' COURSE-FILE-DIRECTORY'.
    DISPLAY ' -----'.
PRINT-DIR.
    ADD 1 TO IND.
    IF INDX-D (IND) IS NOT EQUAL TO ZEROS DISPLAY
    CRSE-DIRECT (IND) GO TO PRINT-DIR.
    MOVE 1 TO CN. OPEN INPUT COURSE-FILE-DIRECTORY. READ
    COURSE-FILE-DIRECTORY INTO CFD-WORK AT END GO TO DISPLAY-IT.
    DISPLAY 'CFD FROM DISK'.
    DISPLAY ' -----'.
DISPLAY-IT.
    DISPLAY CRSE-DIRECT (CN). IF CN IS NOT EQUAL TO IND ADD 1 TO

```

ON GO TO DISPLAY-IT. ADD 1 TO ON DISPLAY CRSE-DIRECT (CN)
' LAST ONE'. CLOSE COURSE-FILE-DIRECTORY.
WRAP-UP.
STOP RUN.

UPDATE COURSE-FILE

IDENTIFICATION DIVISION.

PROGRAM-ID. UPDTCRSE.

AUTHOR. GOVIND K R I P L A N I.

REMARKS. THIS PROGRAM UPDATES COURSE-FILE.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-360-F75.

OBJECT-COMPUTER. IBM-360-F75.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

```

SELECT CARD-FILE ASSIGN TO          UR-2540R-S-CARDS
    RESERVE NO ALTERNATE AREA.
    SELECT DISK-IN ASSIGN TO UT-2314-S-INDISK.
    SELECT COURSE-FILE-DIRECTORY ASSIGN TO UT-2314-S-CFD.
    SELECT WORK-FILE ASSIGN TO 4 DA-2314-D-SORTWK01.
    SELECT INTER-FILE              ASSIGN TO UT-2314-S-CLASSDIR.
    SELECT COURSE-FILE              ASSIGN TO DA-2314-R-SRF
        ACCESS IS RANDOM
        NOMINAL KEY IS NOM-KEY.

```

DATA DIVISION.

FILE SECTION.

SD WORK-FILE

```

    RECORDING MODE IS F
    RECORD CONTAINS 20 CHARACTERS
    LABEL RECORD IS STANDARD
    DATA RECORD IS WORKS.

```

01 WORKS.

```

    02 STUD-NO-WORK PICTURE 9(06).
    02 FILLER PICTURE X(14).

```

FD CARD-FILE

```

    RECORDING MODE IS F
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORD IS OMITTED
    DATA RECORD IS CARD-REC.

```

01 CARD-REC.

```

    02 FILLER PICTURE XXX.
    02 COURSE-CD PICTURE 9(06).
    02 SEM-CD PICTURE X.
    02 COURSE-DES-CD PICTURE X(25).
    02 CR-CD PICTURE X.
    02 FILLER PICTURE X(43).
    02 COL-80 PICTURE X.

```

FD DISK-IN

```

    RECORDING MODE IS F
    RECORD CONTAINS 20 CHARACTERS
    BLOCK CONTAINS 10 RECORDS
    LABEL RECORD IS STANDARD
    DATA RECORD IS DISK-IN-REC.

```

01 DISK-IN-REC.

```

    02 FILLER PICTURE X(20).

```

```

FD  INTER-FILE
    RECORDING MODE IS F
    RECORD CONTAINS 20 CHARACTERS
    LABEL RECORD IS STANDARD
    DATA RECORD IS INTER-REC.
01  INTER-REC.
    02  COURSE-I PICTURE 9(06).
    02  INDX-I PICTURE 9(06).
    02  FILLER PICTURE X(08).
FD  COURSE-FILE
    RECORDING MODE IS F
    RECORD CONTAINS 37 CHARACTERS
    LABEL RECORD IS STANDARD
    DATA RECORD IS COURSE-REC.
01  COURSE-REC.
    02  FILLER PICTURE 9(04).
    02  CRSE-KEY PICTURE 9(06).
    02  FILLER PICTURE X(27).
FD  COURSE-FILE-DIRECTORY
    RECORDING MODE IS F
    RECORD CONTAINS 2400 CHARACTERS
    LABEL RECORD IS STANDARD
    DATA RECORD IS COURSE-FILE-DIRECT.
01  COURSE-FILE-DIRECT.
    02  FILLER PICTURE X(2400).
WORKING-STORAGE SECTION.
77  CN PICTURE 9999 VALUE 1.
77  IND PICTURE 9(06) VALUE ZEROS.
77  NOM-KEY PICTURE S9(8) COMP SYNC.
01  COURSE-CC.
    02  ADDR-C PICTURE 9999 VALUE ZEROS.
    02  COURSE-C PICTURE 9(06).
    02  SEM-C PICTURE X.
    02  COURSE-DES-C PICTURE X(25).
    02  CR-C PICTURE X.
01  CDF-WORK-A.
    02  CRSE-DIRECTA OCCURS 200 TIMES.
        03  CRSE-A PICTURE 9(06).
        03  INDX-A PICTURE 9(06).
01  DISK-IN-WORK.
    02  COURSE-DIN PICTURE 9(06).
    02  INDX-DIN PICTURE 9(06).
    02  FILLER PICTURE X(08).
PROCEDURE DIVISION.
STARTS.
    OPEN INPUT CARD-FILE OUTPUT DISK-IN COURSE-FILE-DIRECTORY
    I-D COURSE-FILE.
READING.
    ADD 1 TO IND MOVE IND TO NOM-KEY. READ COURSE-FILE INTO
    COURSE-CC INVALID KEY DISPLAY 'END OF COURSE-FILE' GO TO
    WRAP-UP.
    IF COURSE-C IS EQUAL TO ZEROS GO TO RD-CARD.
    MOVE COURSE-C TO

```

COURSE-DIN MOVE IND TO INDX-DIN WRITE DISK-IN-REC FROM
 DISK-IN-WORK DISPLAY COURSE-CC GO TO READING.
 RD-CRSE-FILE.
 ADD 1 TO IND MOVE IND TO NOM-KEY.
 READ COURSE-FILE INTO COURSE-CC INVALID KEY DISPLAY
 ' NO MORE DUMMY RECORDS' GO TO WRAP-UP.
 RD-CARD.
 READ CARD-FILE AT END GO TO SORT-DIR.
 MOVE-IT.
 IF COL-80 IS NOT EQUAL TO 'S' DISPLAY CARD-REC ' WRONG CARD'
 GO TO RD-CARD. MOVE
 COURSE-CD TO COURSE-C COURSE-DIN MOVE SEM-CD TO SEM-C MOVE
 COURSE-DES-CD TO COURSE-DES-C MOVE CR-CD TO CR-C MOVE IND
 TO INDX-DIN.
 UPDATE-COURSE-FILE.
 REWRITE COURSE-REC FROM COURSE-CC INVALID KEY GO TO ERRORS.
 DISPLAY COURSE-CC.
 UPDATE-DISK-IN.
 WRITE DISK-IN-REC FROM DISK-IN-WORK GO TO RD-CRSE-FILE.
 ERRORS.
 DISPLAY IND COURSE-CC ' REC NOT WRITTEN' GO TO WRAP-UP.
 SORT-DIR.
 DISPLAY ' '.
 CLOSE DISK-IN.
 SORT WORK-FILE
 ON ASCENDING KEY STUD-NO-WORK
 USING DISK-IN
 GIVING INTER-FILE.
 CHECK-RESULT.
 IF SORT-RETURN IS NOT EQUAL TO 0 DISPLAY 'SORT UNSUCCESSFUL'
 GO TO WRAP-UP. DISPLAY 'SORT OK'.
 MOVE ZEROS TO IND.
 OPEN-INTER.
 OPEN INPUT INTER-FILE.
 MOVE ZEROS TO IND. DISPLAY ' '.
 DISPLAY 'COURSE-FILE-DIRECTORY'.
 DISPLAY '-----'.
 READ-INTER.
 READ INTER-FILE AT END GO TO WRAP-UP.
 ADD 1 TO IND MOVE COURSE-I TO CRSE-A (IND) MOVE INDX-I TO
 INDX-A (IND)
 DISPLAY CRSE-DIRECTA (IND) GO TO READ-INTER.
 WRAP-UP.
 ADD 1 TO IND MOVE ZEROS TO INDX-A (IND).
 WRITE COURSE-FILE-DIRECT FROM CDF-WORK-A.
 CLOSE COURSE-FILE-DIRECTORY INTER-FILE CARD-FILE
 COURSE-FILE.
 OPEN INPUT COURSE-FILE-DIRECTORY. READ COURSE-FILE-DIRECTORY
 INTO CDF-WORK-A AT END GO TO NEXT-SEN.
 NEXT-SEN.
 DISPLAY CRSE-DIRECTA (CN). IF CN IS NOT EQUAL TO IND ADD 1
 TO CN GO TO NEXT-SEN. CLOSE COURSE-FILE-DIRECTORY.
 STOP RUN.

CREATE PERFORMANCE-FILE

IDENTIFICATION DIVISION.

PROGRAM-ID. CRWORKF.

AUTHOR. GOVIND K R I P L A N I.

REMARKS. THIS PROGRAM CREATES WORK-FILE WITH DUMMY RECCRDS.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-360-F75.

OBJECT-COMPUTER. IBM-360-F75.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT SW-FILE ASSIGN TO DA-2314-R-SSS

ACCESS IS SEQUENTIAL

NOMINAL KEY IS WK-KEY.

SELECT CARD-FILE ASSIGN TO UR-2540R-S-CARDS.

DATA DIVISION.

FILE SECTION.

FD CARD-FILE

RECORDING MODE IS F

RECORD CONTAINS 80 CHARACTERS

LABEL RECORD IS OMITTED

DATA RECORD IS CARD-REC.

01 CARD-REC.

02 DUMMY-RECS PIC 9(06).

02 FILLER PIC X(73).

02 LAST-COL PIC X.

FD SW-FILE

RECORDING MODE IS F

RECORD CONTAINS 22 CHARACTERS

LABEL RECORD IS STANDARD

DATA RECORD IS SW-REC.

01 SW-REC.

02 FILLER PICTURE X(22).

WORKING-STORAGE SECTION.

77 W-KEY PIC 9(06).

77 W-COUNT PIC 9(06) VALUE ZEROS.

77 WK-KEY PICTURE S9(8) COMP SYNC.

01 NEW-AREA.

02 FILLER PICTURE X(22) VALUE ZEROS.

PROCEDURE DIVISION.

CREATE-IT.

OPEN INPUT CARD-FILE OUTPUT SW-FILE.

READ-CARDS.

READ CARD-FILE AT END GO TO WRAP-UP.

IF LAST-COL IS NOT EQUAL TO 'A'

DISPLAY 'FUNCTION CARD MISSING', GO TO WRAP-UP.

MOVE DUMMY-RECS TO W-KEY.

LOOP-BACK.

MOVE W-COUNT TO WK-KEY. WRITE SW-REC FROM NEW-AREA

INVALID KEY DISPLAY 'NO DUMMY-RECS CREATED'

GO TO WRAP-UP. IF W-COUNT IS NOT EQUAL TO W-KEY ADD 1 TO

W-COUNT GC TO LOOP-BACK.
DISPLAY 'NO OF DUMMY-RECS CREATED = ' DUMMY-RECS.
WRAP-UP.
CLOSE CARD-FILE SW-FILE.
STOP RUN.

CREATE STUDENT-FILE
ITS DIRECTORY
AND LINK WITH
COURSE-FILE AND PERFORMANCE-FILE

IDENTIFICATION DIVISION.

PROGRAM-ID. CREATEF.

AUTHOR. GOVIND K R I P L A N I.

REMARKS. THIS PROGRAM CREATES THE INITIAL FILE ON DISK.
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-360-F75.

OBJECT-COMPUTER. IBM-360-F75.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT COURSE-FILE-DIRECTORY ASSIGN TO UT-2314-S-CFD.

SELECT STUD-FILE-DIRECTORY ASSIGN TO UT-2314-S-SFD.

SELECT CARD-FILE ASSIGN TO UR-2540R-S-CARDS.

SELECT PRINT-FILE ASSIGN TO UR-1403-S-PRINTS.

SELECT DISK-IN ASSIGN TO UT-2314-S-INDISK.

SELECT DISK-OUT ASSIGN TO UT-2314-S-OUTDISK.

SELECT WORK-FILE ASSIGN TO 4 DA-2314-D-SORTWK01.

SELECT STUDENT-FILE ASSIGN TO DA-2314-R-SF

ACCESS IS SEQUENTIAL

NOMINAL KEY IS STUD-KEY.

SELECT STUD-WORK-FILE ASSIGN TO DA-2314-R-SWF

ACCESS IS RANDOM

NOMINAL KEY IS WORK-KEY.

SELECT COURSE-FILE ASSIGN TO DA-2314-R-SRF

ACCESS IS RANDOM

NOMINAL KEY IS COURSE-KEY.

DATA DIVISION.

FILE SECTION.

SD WORK-FILE

RECORDING MODE IS F

RECORD CONTAINS 80 CHARACTERS

LABEL RECORD IS STANDARD

DATA RECORD IS WORKS.

01 WORKS.

02 DEPT-WORK PICTURE XXX.

02 STUD-NO-WORK PICTURE 9(06).

02 YEAR-WORK PICTURE XX.

02 SEM-WORK PICTURE X.

02 GRADE-WORK PICTURE X.

02 MARK-WORK PICTURE XX.

02 COURSE-CODE-WORK PICTURE 9(06).

02 FILLER PICTURE X(58).

02 COL-80-WORK PICTURE X.

FD CARD-FILE

RECORDING MODE IS F

RECORD CONTAINS 80 CHARACTERS

LABEL RECORD IS OMITTED

DATA RECORD IS CARD-REC.

01 CARD-REC.

02 FILLER PICTURE X(79).

```

02 COL-80 PICTURE X.
FD PRINT-FILE
  RECORDING MODE IS F
  LABEL RECORD IS OMITTED
  RECORD CONTAINS 133 CHARACTERS
  DATA RECORD IS LINE-FORMAT.
01 LINE-FORMAT.
  02 FILLER PICTURE X(133).
FD DISK-IN
  RECORDING MODE IS F
  RECORD CONTAINS 80 CHARACTERS
  BLOCK CONTAINS 10 RECORDS
  LABEL RECORD IS STANDARD
  DATA RECORD IS DISK-IN-REC.
01 DISK-IN-REC.
  02 FILLER PICTURE XXX.
  02 STUD-NO-IN PICTURE 9(06).
  02 YEAR-IN PICTURE XX.
  02 SEM-IN PICTURE X.
  02 FILLER PICTURE X(67).
  02 COL-80-IN PICTURE X.
FD DISK-OUT
  RECORDING MODE IS F
  RECORD CONTAINS 80 CHARACTERS
  LABEL RECORD IS STANDARD
  DATA RECORD IS DIS-OUT-REC.
01 DISK-OUT-REC.
  02 FILLER PICTURE X(80).
FD STUDENT-FILE
  RECORDING MODE IS F
  RECORD CONTAINS 84 CHARACTERS
  LABEL RECORD IS STANDARD
  DATA RECORD IS STUDENT-REC.
01 STUDENT-REC.
  02 FILLER PICTURE XXX.
  02 STUDENT-NO PICTURE 9(06).
  02 FILLER PICTURE X(75).
FD STUD-WORK-FILE
  RECORDING MODE IS F
  RECORD CONTAINS 22 CHARACTERS
  LABEL RECORD IS STANDARD
  DATA RECORD IS ST-WORK-REC.
01 ST-WORK-REC.
  02 KEY-ST-WK PICTURE 9999.
  02 FILLER PICTURE X(18).
FD COURSE-FILE
  RECORDING MODE IS F
  RECORD CONTAINS 37 CHARACTERS
  LABEL RECORD IS STANDARD
  DATA RECORD IS COURSE-REC.
01 COURSE-REC.
  02 FILLER PICTURE X(04).
  02 CRSE-KEY PICTURE 9(06).

```

```

      02 FILLER PICTURE X(27).
FD  COURSE-FILE-DIRECTORY
    RECORDING MODE IS F
    RECORD CONTAINS 2400 CHARACTERS
    LABEL RECORD IS STANDARD
    DATA RECORD IS COURSE-FILE-DIRECT.
01  COURSE-FILE-DIRECT.
      02 FILLER PICTURE X(2400).
FD  STUD-FILE-DIRECTORY
    RECORDING MODE IS F
    RECORD CONTAINS 2400 CHARACTERS
    LABEL RECORD IS STANDARD
    DATA RECORD IS STUD-FILE-DIRECT.
01  STUD-FILE-DIRECT.
      02 FILLER PICTURE X(2400).
WORKING-STORAGE SECTION.
77  WORK-KEY PICTURE S9(8) COMP SYNC.
77  WRK PICTURE 9999 VALUE 0.
77  SK PICTURE 9999 VALUE 0.
77  STUD-KEY PICTURE S9(8) COMP SYNC.
77  COURSE-KEY PICTURE S9(8) COMP SYNC.
77  IND PICTURE 9999 VALUE 0.
77  EXT PICTURE 9999 VALUE 1.
77  GG PICTURE 9999 VALUE 1.
77  WK PICTURE 9999 VALUE 0.
77  SI PICTURE 9999 VALUE 0.
77  CT PICTURE 9999 VALUE 0.
77  CN PICTURE 9999 VALUE 1.
01  CFD-FILL.
      02 FILLER PICTURE X(2400).
01  CFD-WRK REDEFINES CFD-FILL.
      02 CRSE-DIRECT OCCURS 200 TIMES INDEXED BY IN2.
          03 CRSE-D PICTURE 9(06).
          03 INDX-D PICTURE 9(06).
01  COURSE-REL.
      02 ADDR-REL PICTURE 9999.
      02 COURSE-CD-REL PICTURE 9(06).
      02 FILLER PICTURE X(27).
01  COURSE-CHECK.
      02 FRST-OR-NO PICTURE 9999 OCCURS 200 TIMES.
01  WORK-REC-OLD.
      02 LINK-1-OLD PICTURE 9(04).
      02 LINK-2-OLD PICTURE 9999.
      02 YEAR-OLD PICTURE XX.
      02 TERM-OLD PICTURE X.
      02 GRADE-OLD PICTURE X.
      02 MARK-OLD PICTURE XX.
      02 LINK-3-OLD PICTURE 9999.
      02 LINK-4-OLD PICTURE 9999.
01  COURSE-RE.
      02 ADR-C PICTURE 9999.
      02 CRSE-KAY PICTURE 9(06).
      02 FILLER PICTURE X(27).

```

```

01  DISK-OUT-RE.
    02  FILLER PICTURE XXX.
    02  STUD-NO-OUT PICTURE 9(06).
    02  YEAR-OUT PICTURE XX.
    02  SEM-IN PICTURE X.
    02  GR-RE PICTURE X.
    02  MARK-RE PICTURE 99.
    02  COURSE-CD-RE PICTURE 9(06).
    02  FILLER PICTURE X(58).
    02  COL-80-OUT PICTURE X.
01  FORM-F REDEFINES DISK-OUT-RE.
    02  DEPT-F PICTURE XXX.
    02  ST-NO-F PICTURE 9(06).
    02  ST-NAME-F PICTURE X(20).
    02  DEGREE-F PICTURE XXXX.
    02  LEVEL-F PICTURE X.
    02  SEX-F PICTURE X.
    02  STREET-F PICTURE X(17).
    02  APT-F PICTURE XXXX.
    02  TOWN-F PICTURE X(16).
    02  TEL-F PICTURE X(07).
    02  COL-80-F PICTURE X.
01  FORM-G REDEFINES DISK-OUT-RE.
    02  DEPT-G PICTURE XXX.
    02  ST-NO-G PICTURE 9(06).
    02  YEAR-G PICTURE XX.
    02  TERM-G PICTURE X.
    02  GRADE-G PICTURE X.
    02  MARK-G PICTURE XX.
    02  COURSE-CODE-G PICTURE 9(06).
    02  FILLER PICTURE X(58).
    02  COL-80-G PICTURE X.
01  FORM-F-IN.
    02  DEPT-F-IN PICTURE XXX.
    02  ST-NO-F-IN PICTURE 9(06).
    02  ST-NAME-F-IN PICTURE X(20).
    02  DEGREE-F-IN PICTURE XXXX.
    02  LEVEL-F-IN PICTURE XX.
    02  SEX-F-IN PICTURE X.
    02  STREET-F-IN PICTURE X(17).
    02  APT-F-IN PICTURE XXXX.
    02  TOWN-F-IN PICTURE X(16).
    02  TEL-F-IN PICTURE X(07).
    02  KEY-F PICTURE 9999.
    02  LL-S PICTURE 9999 VALUE 0.
01  FORM-G-IN.
    02  ST-WORK-CODE OCCURS 15 TIMES.
        03  LINK-1 PICTURE 9999.
        03  LINK-2 PICTURE 9999.
        03  YEAR-G-IN PICTURE XX.
        03  TERM-G-IN PICTURE X.
        03  GRADE-G-IN PICTURE X.
        03  MARK-G-IN PICTURE XX.

```

```

03 LINK-3 PICTURE 9999.
03 LINK-4 PICTURE 9999.
01 FFI-WORK,
02 STUD-DIRECT OCCURS 200 TIMES.
03 STUD-NO-D PICTURE 9(06).
03 STUD-INDX-D PICTURE 9(06).
01 NEW-AREA.
02 FIL-ZEROS PICTURE 9999.
02 FILLER PICTURE X(18).
PROCEDURE DIVISION.
ZERO-IT.
    MOVE ZEROS TO FRST-OR-NO (EXT). IF EXT IS NOT EQUAL TO
    200 ADD 1 TO EXT GO TO ZERO-IT. MOVE 1 TO EXT.
STARTS.
    OPEN INPUT CARD-FILE OUTPUT          DISK-OUT PRINT-FILE.
READ-CARDS.
    READ CARD-FILE AT END GO TO CLOSE-1.
    IF COL-80 IS EQUAL TO 'C' MOVE CARD-REC TO DISK-OUT-RE
    MOVE ST-NO-3 TO CRSE-D (EXT) MOVE COURSE-CODE-G
    TO INDX-D (EXT)
    ADD 1 TO EXT GO TO READ-CARDS.
    IF COL-80 IS EQUAL TO 'F' GO TO WRITE-IT.
    IF COL-80 IS EQUAL TO 'G' GO TO WRITE-IT.
    DISPLAY CARD-REC ' CHECK COL-80' GO TO READ-CARDS.
WRITE-IT.
    WRITE DISK-OUT-REC FROM CARD-REC GO TO READ-CARDS.
CLOSE-1.
    CLOSE CARD-FILE DISK-OUT.
SORT-2.
    DISPLAY '          '.
    SORT WORK-FILE
        ON ASCENDING KEY STUD-NO-WORK, YEAR-WORK,
        COURSE-CODE-WORK, COL-80-WORK
    USING DISK-OUT
    GIVING DISK-IN.
CHECK-SORT-2.
    IF SORT-RETURN IS NOT EQUAL TO 0 DISPLAY
    'SORT-2 UNSUCCESSFUL' GO TO WRAP-UP.
    DISPLAY 'SORT - 2 OK'.
OPEN-DISK-IN.
    OPEN INPUT COURSE-FILE-DIRECTORY.
    READ COURSE-FILE-DIRECTORY INTO CFD-FILL AT END GO TO
    CLOSE-CFD.
CLOSE-CFD.
    CLOSE COURSE-FILE-DIRECTORY.
    OPEN INPUT DISK-IN OUTPUT STUDENT-FILE
    STUD-FILE-DIRECTORY I-O STUD-WORK-FILE COURSE-FILE.
    MOVE 1 TO EXT.
DISP-DISK.
    MOVE EXT TO COURSE-KEY. READ COURSE-FILE INTO COURSE-REL
    INVALID KEY DISPLAY 'CHECK NO 1' GO TO ERR. IF COURSE-CD-REL
    IS EQUAL TO ZEROS MOVE 1 TO EXT GO TO READ-DISK-IN. MOVE
    ZEROS TO ADDR-REL REWRITE COURSE-REC FROM COURSE-REL

```

INVALID KEY DISPLAY 'CHECK NO 2' GO TO EPR. ADD 1 TO EXT GO
 TO DISP-DISK.
 READ-DISK-IN.
 READ DISK-IN INTO DISK-OUT-RE AT END GO TO WRAP-UP.
 PROC-DISK-IN.
 IF COL-80-F IS EQUAL TO 'F' GO TO MOVE-STUD. ADD 1 TO WK CT
 MOVE YEAR-G TO YEAR-G-IN (CT) MOVE TERM-G TO TERM-G-IN (CT)
 MOVE GRADE-G TO GRADE-G-IN (CT) MOVE MARK-G TO
 MARK-G-IN (CT). IF LL-S IS EQUAL TO ZEROS GO TO MOVE-WK.
 MOVE LL-S TO EXT MOVE WK TO LINK-3 (EXT) MOVE CT
 TO LL-S GO TO READ-IS.
 MOVE-WK.
 MOVE WK TO KEY-F MOVE CT TO LL-S.
 READ-IS.
 SET IN2 TO 1.
 SEARCH CRSE-DIRECT AT END GO TO ERR WHEN
 COURSE-CD-RE = CRSE-D (IN2) NEXT SENTENCE.
 NOW-MOVE. MOVE INDX-D (IN2) TO COURSE-KEY.
 MOVE INDX-D (IN2) TO LINK-4 (CT).
 CHECK-IT.
 IF FRST-OR-NO (IN2) IS NOT EQUAL TO 0 GO TO MOVE-FRST-OR-NO.
 READ COURSE-FILE INTO COURSE-REL MOVE WK TO ADDR-REL
 FRST-OR-NO (IN2) MOVE SI TO LINK-1 (CT) REWRITE COURSE-REC
 FROM COURSE-REL GO TO READ-DISK-IN.
 MOVE-FRST-OR-NO.
 MOVE FRST-OR-NO (IN2) TO WORK-KEY READ STUD-WORK-FILE INTO
 WORK-REC-OLD. MOVE WK TO LINK-2-OLD FRST-OR-NO (IN2)
 MOVE SI TO LINK-1 (CT).
 REWRITE ST-WORK-REC FROM WORK-REC-OLD GO TO READ-DISK-IN.
 MOVE-STUD.
 IF SI IS NOT EQUAL TO 0 GO TO WRITE-SI.
 ADD-SI.
 ADD 1 TO SI MOVE DEPT-F TO DEPT-F-IN MOVE ST-NO-F TO
 ST-NO-F-IN MOVE ST-NAME-F TO ST-NAME-F-IN MOVE DEGREE-F
 TO DEGREE-F-IN MOVE LEVEL-F TO LEVEL-F-IN MOVE SEX-F TO
 SEX-F-IN MOVE STREET-F TO STREET-F-IN MOVE APT-F TO APT-F-IN
 MOVE TOWN-F TO TOWN-F-IN MOVE TEL-F TO TEL-F-IN GO TO
 READ-DISK-IN.
 WRITE-SI.
 ADD 1 TO SK.
 MOVE SK TO STUD-KEY.
 WRITE STUDENT-REC FROM FORM-F-IN INVALID KEY DISPLAY
 'FORM-F-IN ERROR' GO TO ERR.
 MOVE ZEROS TO LINK-3 (CT). MOVE ST-NO-F-IN TO
 STUD-NO-D (SK) MOVE SK TO STUD-INDX-D (SK). ADD 1 TO GG.
 WRITE-SW.
 ADD 1 TO WRK MOVE WRK TO WORK-KEY MOVE 0 TO LINK-2 (WRK).
 MOVE ST-WORK-CODE (CN) TO NEW-AREA. REWRITE ST-WORK-REC
 FROM NEW-AREA INVALID KEY
 DISPLAY 'STUD-WORK-CODE ERROR' GO TO ERR. IF CN IS NOT
 EQUAL TO CT ADD 1 TO CN GO TO WRITE-SW. MOVE 1 TO CN
 MOVE ZEROS TO CT LL-S GO TO ADD-SI.
 ERR.

DISPLAY 'SEARCH UNSUCCESSFUL' GO TO WRAP-UP.
 WRAP-UP.
 MOVE ZEROS TO ST-NO-F-IN ADD 1 TO SK MOVE SK TO STUD-KEY.
 WRITE STUDENT-REC FROM FORM-F-IN INVALID KEY DISPLAY
 'LAST NOT WRITTEN' GO TO WRAP-UP-1.
 MOVE ZEROS TO FIL-ZEROS ADD 1 TO WRK MOVE WRK TO WORK-KEY.
 REWRITE ST-WORK-REC FROM NEW-AREA INVALID KEY DISPLAY
 'LAST WORK NOT WRITTEN' GO TO WRAP-UP-1. MOVE WRK TO
 FIL-ZEROS MOVE 0 TO WORK-KEY. REWRITE ST-WORK-REC FROM
 NEW-AREA INVALID KEY DISPLAY 'CHECK ZERO STUD-WORK-REC' GO
 TO WRAP-UP. MOVE 1 TO WRK.
 CLOSE STUDENT-FILE. OPEN INPUT STUDENT-FILE.
 PRINT-WF.
 MOVE WRK TO WORK-KEY. READ STUD-WORK-FILE INTO NEW-AREA
 INVALID KEY DISPLAY 'NOT OK-1' GO TO WRAP-UP-1. IF FIL-ZEROS
 IS NOT EQUAL TO ZEROS WRITE LINE-FORMAT FROM NEW-AREA AFTER
 POSITIONING 3 ADD 1 TO WRK GO TO PRINT-WF. MOVE 1 TO SK.
 PRINT-SF.
 MOVE SK TO STUD-KEY. READ STUDENT-FILE
 AT END GO TO WRAP-UP-1. IF STUDENT-NO IS NOT EQUAL TO
 ZEROS WRITE LINE-FORMAT FROM STUDENT-REC AFTER POSITIONING
 3 ADD 1 TO SK GO TO PRINT-SF. MOVE 1 TO CT.
 PRINT-CF.
 MOVE CT TO COURSE-KEY. READ COURSE-FILE INTO COURSE-REL
 INVALID KEY DISPLAY 'NOT OK-3' GO TO WRAP-UP-1. IF CT IS
 NOT EQUAL TO 21 WRITE LINE-FORMAT FROM COURSE-REL AFTER
 POSITIONING 3 ADD 1 TO CT GO TO PRINT-CF.
 MOVE 1 TO CT. DISPLAY 'STUDENT FILE DIRECTORY'.
 DISPLAY '-----'.
 PRINT-SFDIR.
 DISPLAY STUD-DIRECT (CT). IF CT IS NOT EQUAL TO GG ADD 1 TO
 CT GO TO PRINT-SFDIR. MOVE 1 TO CT. DISPLAY ' '.
 DISPLAY 'COURSE FILE DIRECTORY'. DISPLAY '-----'.
 PRINT-CFDIR.
 DISPLAY CRSE-DIRECT (CT). IF CT IS NOT EQUAL TO 21 ADD 1 TO
 CT GO TO PRINT-CFDIR.
 WRAP-UP-1.
 MOVE ZEROS TO STUD-NO-D (GG).
 WRITE STUD-FILE-DIRECT FROM FFI-WORK.
 CLOSE PRINT-FILE STUD-WORK-FILE STUDENT-FILE
 DISK-IN COURSE-FILE STUD-FILE-DIRECTORY.
 STOP RUN.

UPDATE STUDENT-FILE,
COURSE-FILE AND PERFORMANCE-FILE

IDENTIFICATION DIVISION.
 PROGRAM-ID. UPDTSWC.
 AUTHOR. GOVIND K R I P L A N I.
 REMARKS.

```

*****
* THIS PROGRAM UPDATES STUDENT-FILE, STUD-WORK-FILE AND *
* COURSE-FILE. IT IS REQUIRED TO UPDATE THESE FILES WHEN *
* THE STUDENTS MAKE CHANGES IN THEIR COURSES. WHICH THEY *
* NORMALLY DO ONCE A SEMESTER. IT WILL DELETE THE COURSES *
* THEY DONT WANT TO TAKE AND ADD THE COURSES AND PLACE *
* THEM IN THE SEQUENTIAL ORDER. *
* *
* INPUT DATA FOR DELETION. *
* ----- *
* COL 4 - 9          STUDENT NO. *
* COL 16 - 21        COURSE CODE *
* COL 80             ' D ' *
* *
* INPUT DATA FOR ADDITION *
* ----- *
* COL 4 - 9          STUDENT NO *
* COL 10 - 11        YEAR *
* COL 12             SEMESTER *
* COL 16 - 21        COURSE CODE *
* COL 80             ' A ' *
*****

```

ENVIRONMENT DIVISION.
 CONFIGURATION SECTION:
 SOURCE-COMPUTER. IBM-360-F75.
 OBJECT-COMPUTER. IBM-360-F75.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.

```

SELECT COURSE-FILE-DIRECTORY ASSIGN TO UT-2314-S-CFD.
SELECT  STUD-FILE-DIRECTORY ASSIGN TO UT-2314-S-SFD.
SELECT DISK-IN ASSIGN TO UT-2314-S-INDISK.
SELECT CARD-FILE ASSIGN TO      UR-2540R-S-CARDS.
SELECT PRINT-FILE ASSIGN TO      UR-1403-S-PRINTS.
SELECT DISK-OUT ASSIGN TO UT-2314-S-OUTDISK.
SELECT WORK-FILE ASSIGN TO 4 DA-2314-D-SORTWK01.
SELECT STUDENT-FILE ASSIGN TO DA-2314-R-SF
ACCESS IS RANDOM
NOMINAL KEY IS STUD-KEY.
SELECT STUD-WORK-FILE ASSIGN TO DA-2314-R-SWF
ACCESS IS RANDOM
NOMINAL KEY IS WORK-KEY.
SELECT COURSE-FILE          ASSIGN TO DA-2314-R-SRF
ACCESS IS RANDOM
NOMINAL KEY IS COURSE-KEY.

```

DATA DIVISION.
 FILE SECTION.

SD WORK-FILE
RECORDING MODE IS F
RECORD CONTAINS 80 CHARACTERS
LABEL RECORD IS STANDARD
DATA RECORD IS WORKS,
01 WORKS,
02 DEPT-WORK PICTURE XXX,
02 STUD-NO-WORK PICTURE 9(06),
02 YEAR-WORK PICTURE 99,
02 SEM-WORK PICTURE X,
02 GRADE-WORK PICTURE X,
02 MARK-WORK PICTURE XX,
02 COURSE-CODE-WORK PICTURE 9(06),
02 FILLER PICTURE X(58),
02 COL-80-WORK PICTURE X,
FD CARD-FILE
RECORDING MODE IS F
RECORD CONTAINS 80 CHARACTERS
LABEL RECORD IS OMITTED
DATA RECORD IS CARD-REC,
01 CARD-REC,
02 FILLER PICTURE X(79),
02 COL-80 PICTURE X,
FD PRINT-FILE
RECORDING MODE IS F
LABEL RECORD IS OMITTED
RECORD CONTAINS 133 CHARACTERS
DATA RECORD IS LINE-FORMAT,
01 LINE-FORMAT,
02 FILLER PICTURE X(133),
FD DISK-IN
RECORDING MODE IS F
RECORD CONTAINS 80 CHARACTERS
BLOCK CONTAINS 10 RECORDS
LABEL RECORD IS STANDARD
DATA RECORD IS DISK-IN-REC,
01 DISK-IN-REC,
02 FILLER PICTURE XXX,
02 STUD-NO-IN PICTURE 9(06),
02 YEAR-IN PICTURE 99,
02 SEM-IN PICTURE X,
02 GRADE-IN PICTURE X,
02 MARK-IN PICTURE 99,
02 COURSE-IN PICTURE 9(06),
02 FILLER PICTURE X(58),
02 COL-80-IN PICTURE X,
FD DISK-OUT
RECORDING MODE IS F
RECORD CONTAINS 80 CHARACTERS
LABEL RECORD IS STANDARD
DATA RECORD IS DIS-OUT-REC,
01 DISK-OUT-REC,
02 FILLER PICTURE X(80).

```

FD  STUDENT-FILE
    RECORDING MODE IS F
    RECORD CONTAINS 84 CHARACTERS
    LABEL RECORD IS STANDARD
    DATA RECORD IS STUDENT-REC.
01  STUDENT-REC.
    02  FILLER PICTURE XXX.
    02  STUDENT-NO PICTURE 9(06).
    02  FILLER PICTURE X(75).
FD  STUD-WORK-FILE
    RECORDING MODE IS F
    RECORD CONTAINS 22 CHARACTERS
    LABEL RECORD IS STANDARD
    DATA RECORD IS ST-WORK-REC.
01  ST-WORK-REC.
    02  KEY-ST-WK PICTURE 9999.
    02  FILLER PICTURE X(18).
FD  COURSE-FILE
    RECORDING MODE IS F
    RECORD CONTAINS 37 CHARACTERS
    LABEL RECORD IS STANDARD
    DATA RECORD IS COURSE-REC.
01  COURSE-REC.
    02  FILLER PICTURE X(04).
    02  CRSE-KEY PICTURE 9(06).
    02  FILLER PICTURE X(27).
FD  COURSE-FILE-DIRECTORY
    RECORDING MODE IS F
    RECORD CONTAINS 2400 CHARACTERS
    LABEL RECORD IS STANDARD
    DATA RECORD IS COURSE-FILE-DIRECT.
01  COURSE-FILE-DIRECT.
    02  FILLER PICTURE X(2400).
FD  STUD-FILE-DIRECTORY
    RECORDING MODE IS F
    RECORD CONTAINS 2400 CHARACTERS
    LABEL RECORD IS STANDARD
    DATA RECORD IS STUD-FILE-DIRECT.
01  STUD-FILE-DIRECT.
    02  FILLER PICTURE X(2400).
WORKING-STORAGE SECTION.
77  WRK PICTURE 9999.
77  WORK-KEY PICTURE S9(8) COMP SYNC.
77  STUD-KEY PICTURE S9(8) COMP SYNC.
77  COURSE-KEY PICTURE S9(8) COMP SYNC.
77  EXT PICTURE 9999.
77  RR PICTURE 9999 VALUE 1.
77  CD PICTURE 9999 VALUE 1.
77  SR PICTURE 9999 VALUE 1.
77  WR PICTURE 9999 VALUE 1.
77  CT PICTURE 9999 VALUE 1.
77  CXT PICTURE 99 VALUE 2.
77  FR PICTURE 9999 VALUE 1.

```

```

77 AXT PICTURE 9999.
01 HEAD-1.
   02 FILER PICTURE X(33) VALUE
      'FILES AFTER DELETION AND ADDITION'.
01 FILL-KEYS.
   02 KEYS-IN PICTURE 9999 OCCURS 15 TIMES.
01 FFI-WORK.
   02 STUD-DIRECT OCCURS 200 TIMES INDEXED BY IN1.
      03 STUD-NO-D PICTURE 9(06).
      03 STUD-INDX-D PICTURE 9(06).
01 CNT-IN.
   02 CNT PICTURE 9999 OCCURS 20 TIMES.
01 COURSE-REL.
   02 ADDR-REL PICTURE 9999.
   02 COURSE-CD-REL PICTURE 9(06).
   02 FILLER PICTURE X(27).
01 DUMMY-REC-TRACK.
   02 DUMMY-ADDR PICTURE 9999 OCCURS 10 TIMES.
01 FORM-G-IN.
   02 ST-WORK-CODE OCCURS 20 TIMES.
      03 LINK-1 PICTURE 9999.
      03 LINK-2 PICTURE 9999.
      03 YEAR-G-IN PICTURE 99.
      03 TERM-G-IN PICTURE X.
      03 GRADE-G-IN PICTURE X.
      03 MARK-G-IN PICTURE 99.
      03 LINK-3 PICTURE 9999.
      03 LINK-4 PICTURE 9999.
01 CFD-FILL.
   02 CRSE-DIRECT OCCURS 200 TIMES INDEXED BY IN2.
      03 CRSE-D PICTURE 9(06).
      03 INDX-D PICTURE 9(06).
01 CARD-RECS.
   02 SW-FORM PICTURE X(22).
   02 FILLER PICTURE X(57).
   02 KOL-80 PICTURE X.
01 CF-REC REDEFINES CARD-RECS.
   02 CF-FORM PICTURE X(37).
   02 FILLER PICTURE X(43).
01 DIR-REC REDEFINES CARD-RECS.
   02 CRSE-WRK-DIRECT PICTURE 9(12).
   02 FILLER PICTURE X(68).
01 AD-REC REDEFINES CARD-RECS.
   02 FILLER PICTURE XXX.
   02 ST-NO-AD PICTURE 9(06).
   02 YEAR-AD PICTURE 99.
   02 TERM-AD PICTURE X.
   02 GRADE-AD PICTURE X.
   02 MARK-AD PICTURE 99.
   02 CRSE-CD-AD PICTURE 9(06).
   02 FILLER PICTURE X(59).
01 F-REC.
   02 SF-FORM.

```

```

03 SF-STUD PICTURE 9(06),
03 FILLER PICTURE X(24),
02 SF-NO PICTURE 9(04) VALUE 1,
01 SWF-AREA,
02 SWF-IN,
03 SWF-LINK-1 PICTURE 9999,
03 SWF-LINK-2 PICTURE 9999,
03 SWF-YEAR PICTURE 99,
03 SWF-SEM PICTURE X,
03 SWF-GRADE PICTURE X,
03 SWF-MARK PICTURE 99,
03 SWF-LINK-3 PICTURE 9999,
03 SWF-LINK-4 PICTURE 9999.

```

PROCEDURE DIVISION.

STARTS,

OPEN INPUT CARD-FILE OUTPUT DISK-OUT PRINT-FILE

I-O COURSE-FILE STUD-WORK-FILE STUDENT-FILE.

MOVE 0 TO WORK-KEY. READ STUD-WORK-FILE INTO SWF-AREA INVALID
KEY DISPLAY 'CHECK STARTS' GO TO WRAP-UP. MOVE SWF-LINK-1
TO WRK.

READ-CARDS.

READ CARD-FILE INTO CARD-RECS AT END GO TO DISPLAY-FILE1.

IF KOL-80 IS EQUAL TO 'A' GO TO MOVE-TO-DISK.

IF KOL-80 IS EQUAL TO 'D' GO TO MOVE-TO-DISK.

DISPLAY CARD-RECS 'CHECK COL-80' GO TO READ-CARDS.

MOVE-TO-DISK.

WRITE DISK-OUT-REC FROM CARD-RECS GO TO READ-CARDS.

DISPLAY-FILE1.

MOVE CT TO WORK-KEY. READ STUD-WORK-FILE INVALID KEY DISPLAY

'CHECK 1' GO TO WRAP-UP. IF KEY-ST-WK IS NOT EQUAL TO

ZEROS WRITE LINE-FORMAT FROM ST-WORK-REC AFTER POSITIONING

3 ADD 1 TO CT GO TO DISPLAY-FILE1. MOVE 1 TO FR CT.

DISPLAY-STU1.

MOVE FR TO STUD-KEY. READ STUDENT-FILE INVALID KEY DISPLAY

'CHECK 2' GO TO WRAP-UP. IF STUDENT-NO IS NOT EQUAL TO

ZEROS WRITE LINE-FORMAT FROM STUDENT-REC AFTER POSITIONING

3 ADD 1 TO FR GO TO DISPLAY-STU1. MOVE 1 TO FR.

DISPLAY-COURS1.

MOVE FR TO COURSE-KEY. READ COURSE-FILE INVALID KEY DISPLAY

'CHECK 3' GO TO WRAP-UP. IF CRSE-KEY IS NOT EQUAL TO ZEROS

WRITE LINE-FORMAT FROM COURSE-REC AFTER POSITIONING 3 ADD 1

TO FR GO TO DISPLAY-COURS1. MOVE 1 TO FR.

CLOSE-DISK-OUT.

CLOSE DISK-OUT.

DISPLAY ' '.

SORT-IT.

SORT WORK-FILE ON

ASCENDING KEY STUD-NO-WORK

DESCENDING KEY COL-80-WORK

ASCENDING KEY COURSE-CODE-WORK

USING DISK-OUT

GIVING DISK-IN.

CHECK-RESULT.

```

IF SORT-RETURN IS NOT EQUAL TO 0 DISPLAY 'SORT UNSUCCESSFUL'
GO TO WRAP-UP, DISPLAY 'SORT OK'.
OPEN-DISK-IN.
OPEN-INPUT DISK-IN COURSE-FILE-DIRECTORY STUD-FILE-DIRECTORY,
READ COURSE-FILE-DIRECTORY INTO CFD-FILL AT END GO TO
READ-SFD.
READ-SFD.
READ STUD-FILE-DIRECTORY INTO FFI-WORK AT END GO TO
CLOSE-DIRECTS.
CLOSE-DIRECTS.
CLOSE COURSE-FILE-DIRECTORY STUD-FILE-DIRECTORY.
MOVE 1 TO CD RR SR WR FR. MOVE 2 TO DUMMY-ADDR (1).
READ-DISK-IN.
READ DISK-IN AT END GO TO SET-WORK-FILE.
IF COL-80-IN IS EQUAL TO 'A' GO TO SEARCH-STUD-A.
SEARCH-STUD.
SET IN1 TO 1.
SEARCH STUD-DIRECT AT END DISPLAY 'STUDENT NOT FOUND' GO TO
WRAP-UP WHEN STUD-NO-IN = STUD-NO-D (IN1) NEXT SENTENCE.
MOVE-STUD-ADDR.
MOVE STUD-INDX-D (IN1) TO STUD-KEY. READ STUDENT-FILE INTO
F-REC INVALID KEY DISPLAY 'CHECK STUD-KEY' GO TO WRAP-UP.
MOVE SF-NO TO WORK-KEY CNT (CT). READ STUD-WORK-FILE INTO
SWF-AREA INVALID KEY DISPLAY 'WORK-FILE-1' GO TO
WRAP-UP. MOVE SWF-IN TO ST-WORK-CODE (CD) DISPLAY SWF-AREA.
MOVE LINK-4 (CD) TO COURSE-KEY. READ COURSE-FILE
INTO COURSE-REL INVALID KEY DISPLAY 'CHECK MOVE STUD-ADDR'
GO TO WRAP-UP.
IF COURSE-CD-REL IS EQUAL TO COURSE-IN GO TO ADJUST-1.
NEXT-WORK.
IF LINK-3 (CD) IS EQUAL TO ZEROS DISPLAY DISK-IN-REC
* STUDENT NOT TAKING THIS COURSE - NOT DELETED' GO TO
READ-DISK-IN. ADD 1 TO CT MOVE LINK-3 (CD) TO WORK-KEY
CNT (CT) ADD 1 TO CD READ STUD-WORK-FILE INTO SWF-AREA
INVALID KEY DISPLAY 'WORK-FILE-2' GO TO
WRAP-UP. MOVE SWF-IN TO ST-WORK-CODE (CD)
MOVE LINK-4 (CD) TO COURSE-KEY READ COURSE-FILE
INTO COURSE-REL INVALID KEY DISPLAY 'CHECK COURSE-FILE 2' GO
TO WRAP-UP. IF COURSE-CD-REL IS NOT EQUAL TO COURSE-IN GO TO
NEXT-WORK. MOVE CD TO SR SUBTRACT 1 FROM SR MOVE LINK-3 (CD)
TO LINK-3 (SR) MOVE ZEROS TO LINK-3 (CD) MOVE CNT (CD) TO
WORK-KEY MOVE ST-WORK-CODE (CD) TO SWF-IN
REWRITE ST-WORK-REC FROM SWF-AREA INVALID
KEY DISPLAY 'CHECK 1' GO TO WRAP-UP. MOVE CNT (SR) TO
WORK-KEY MOVE ST-WORK-CODE (SR) TO SWF-IN
REWRITE ST-WORK-REC FROM SWF-AREA
INVALID KEY DISPLAY 'CHECK NEXT-WORK' GO TO WRAP-UP.
MOVE 1 TO CD CT GO TO SEARCH-COURSE.
ADJUST-1.
MOVE LINK-3 (CD) TO SF-NO MOVE ZEROS TO LINK-3 (CD)
MOVE ST-WORK-CODE (CD) TO SWF-IN
REWRITE ST-WORK-REC FROM SWF-AREA INVALID KEY
DISPLAY 'CHECK ADJUST-1' GO TO WRAP-UP.

```



```

REWRITE STUDENT-REC FROM F-REC INVALID KEY DISPLAY
'CHECK ADJUST-1 - 2' GO TO WRAP-UP.
SEARCH-COURSE.
  SET IN2 TO 1.
  SEARCH CRSE-DIRECT AT END DISPLAY 'COURSE NOT FOUND' GO TO
  WRAP-UP WHEN COURSE-IN = CRSE-D (IN2) NEXT SENTENCE.
MOVE-COURSE-ADDR.
  MOVE INDX-D (IN2) TO COURSE-KEY. READ COURSE-FILE INTO
  COURSE-REL INVALID KEY DISPLAY 'CHECK MOVE-COURSE-ADDR' GO TO
  WRAP-UP. MOVE ADDR-REL TO WORK-KEY CNT (CT). READ
  STUD-WORK-FILE INTO SWF-AREA INVALID KEY DISPLAY
  'CHECK MOVE-COURSE-ADDR - 2' GO TO WRAP-UP.
  MOVE SWF-IN TO ST-WORK-CODE (CD) MOVE LINK-1 (CD)
  TO STUD-KEY READ STUDENT-FILE INVALID KEY DISPLAY
  'CHECK MOVE COURSE-ADDR - 3' GO TO WRAP-UP. IF
  STUD-NO-IN = STUDENT-NO GO TO ADJUST-2.
NEXT-STUD.
  IF LINK-2 (CD) = ZEROS DISPLAY 'STUDENT NOT IN THIS CLASS'
  GO TO READ-DISK-IN. ADD 1 TO CT MOVE LINK-2 (CD) TO WORK-KEY
  CNT (CT) ADD 1 TO CD. READ STUD-WORK-FILE INTO
  SWF-AREA INVALID KEY DISPLAY 'CHECK NEXT-STUD' GO
  TO WRAP-UP. MOVE SWF-IN TO ST-WORK-CODE (CD)
  MOVE LINK-1 (CD) TO STUD-KEY READ STUDENT-FILE
  INVALID KEY DISPLAY 'CHECK NEXT-STUD 1' GO TO WRAP-UP. IF
  STUD-NO-IN IS NOT EQUAL TO STUDENT-NO GO TO NEXT-STUD.
  MOVE CD TO SR SUBTRACT 1 FROM SR MOVE LINK-2 (CD) TO
  LINK-2 (SR) MOVE ZEROS TO LINK-2 (CD) MOVE CNT (CD) TO
  WORK-KEY. MOVE ST-WORK-CODE (CD) TO SWF-IN. REWRITE
  ST-WORK-REC FROM SWF-AREA INVALID
  KEY DISPLAY 'CHECK NEXT-STUD-2' GO TO WRAP-UP. MOVE CNT (SR)
  TO WORK-KEY. MOVE ST-WORK-CODE (SR) TO SWF-IN REWRITE
  ST-WORK-REC FROM SWF-AREA
  INVALID KEY DISPLAY 'CHECK NEXT-STUD 3' GO TO WRAP-UP.
  DISPLAY DISK-IN-REC 'DELETED' GO TO DUMMY-TRACK.
ADJUST-2.
  MOVE LINK-2 (CD) TO ADDR-REL MOVE ZEROS TO LINK-2 (CD)
  REWRITE COURSE-REC FROM COURSE-REL INVALID KEY DISPLAY
  'CHECK ADJUST 2' GO TO WRAP-UP.
  DISPLAY DISK-IN-REC 'DELETED'.
DUMMY-TRACK.
  MOVE DUMMY-ADDR (1) TO EXT MOVE CNT (CT) TO DUMMY-ADDR (EXT)
  ADD 1 TO EXT MOVE EXT TO DUMMY-ADDR (1) MOVE 1 TO CD CT GO
  TO READ-DISK-IN.
SEARCH-STUD-A.
  MOVE 1 TO CD.
  SET IN1 TO 1.
  SEARCH STUD-DIRECT AT END DISPLAY 'STUDENT NOT FOUND - A'
  GO TO WRAP-UP WHEN STUD-NO-IN = STUD-NO-D (IN1) NEXT
  SENTENCE.
MOVE-STUD-ADDR-A.
  MOVE STUD-INDX-D (IN1) TO STUD-KEY SWF-LINK-1. READ
  STUDENT-FILE INTO F-REC INVALID KEY DISPLAY 'CHECK S-KEY A'
  GO TO WRAP-UP.

```

MOVE SF-ND TO WORK-KEY KEYS-IN (CD), READ STUD-WORK-FILE
 INTO ST-WORK-CODE (CD) INVALID KEY DISPLAY 'CHECK STUD-A-1'
 GO TO WRAP-UP. MOVE LINK-4 (CD) TO COURSE-KEY, READ
 COURSE-FILE INTO COURSE-REL INVALID KEY DISPLAY 'CHECK ST2'
 GO TO WRAP-UP. IF COURSE-IN IS NOT GREATER THAN
 COURSE-CD-REL GO TO ADJUST-1-A.

NEXT-WORK-A.
 IF LINK-3 (CD) IS EQUAL TO ZEROS GO TO ADJUST-2-A. MOVE
 LINK-3 (CD) TO WORK-KEY. MOVE CD TO AXT ADD 1 TO CD.
 MOVE LINK-3 (AXT)
 TO KEYS-IN (CD). READ STUD-WORK-FILE INTO ST-WORK-CODE (CD)
 INVALID KEY DISPLAY 'CHECK NEXT-WORK ' GO TO WRAP-UP. IF
 YEAR-G-IN (CD) IS NOT EQUAL TO YEAR-G-IN (AXT) GO TO
 ADJUST-3-A. MOVE LINK-4 (CD) TO COURSE-KEY. READ COURSE-FILE
 INTO COURSE-REL INVALID KEY DISPLAY 'CHECK NEXT-WORK 2' GO
 TO WRAP-UP. IF COURSE-IN IS GREATER THAN COURSE-CD-REL GO TO
 NEXT-WORK-A. IF COURSE-IN IS EQUAL TO COURSE-CD-REL GO TO
 ADJUST-1-A.

ADJUST-LESS.
 MOVE LINK-3 (AXT) TO SWF-LINK-3 PERFORM MOVE-SWF.
 SET IN2 TO 1.
 SEARCH CRSE-DIRECT AT END DISPLAY 'COURSE NOT FOUND - B' GO
 TO WRAP-UP WHEN COURSE-IN = CRSE-D (IN2) NEXT SENTENCE.
 MOVE CD TO AXT SUBTRACT 1 FROM AXT. MOVE INDX-D (IN2) TO
 SWF-LINK-4. IF CXT IS EQUAL TO DUMMY-ADDR (1) MOVE WRK TO
 LINK-3 (AXT) WR ADD 1 TO WRK GO TO WRITE-SWF-B. MOVE
 DUMMY-ADDR (CXT) TO LINK-3 (AXT) WR ADD 1 TO CXT.

WRITE-SWF-B.
 MOVE KEYS-IN (AXT) TO WORK-KEY REWRITE ST-WORK-REC FROM
 ST-WORK-CODE (AXT) INVALID KEY DISPLAY 'CHECK WRITE-SWF-B'
 GO TO WRAP-UP. GO TO ADJUST-CLASS-A.

ADJUST-3-A.
 MOVE LINK-3 (CD) TO SWF-LINK-3 GO TO MOVE-SWF.

ADJUST-2-A.
 MOVE ZEROS TO SWF-LINK-3.

MOVE-SWF.
 MOVE YEAR-IN TO SWF-YEAR MOVE SEM-IN TO SWF-SEM MOVE
 GRADE-IN TO SWF-GRADE MOVE MARK-IN TO SWF-MARK.

SEARCH-COURSE-A.
 SET IN2 TO 1.
 SEARCH CRSE-DIRECT AT END DISPLAY 'COURSE NOT FOUND - A'
 GO TO WRAP-UP WHEN COURSE-IN = CRSE-D (IN2) NEXT SENTENCE.

MOVE-COURSE-ADDR-A.
 MOVE CD TO AXT SUBTRACT 1 FROM AXT.
 MOVE INDX-D (IN2) TO SWF-LINK-4.
 IF CXT IS EQUAL TO DUMMY-ADDR (1) GO TO MOVE-WRK. MOVE
 DUMMY-ADDR (CXT) TO LINK-3 (CD) WR ADD 1 TO CXT.

WRITE-SWF-A.
 MOVE KEYS-IN (CD) TO WORK-KEY REWRITE ST-WORK-REC FROM
 ST-WORK-CODE (CD) INVALID KEY DISPLAY 'CHECK WRITE-SWF-A'
 GO TO WRAP-UP. GO TO ADJUST-CLASS-A.

MOVE-WRK.
 MOVE WRK TO LINK-3 (CD) WR ADD 1 TO WRK GO TO WRITE-SWF-A.

ADJUST-1-A.

IF COURSE-IN IS EQUAL TO COURSE-CD-REL MOVE 1 TO CD DISPLAY
 DISK-IN-REC 'ASKED TO ADD - ALREADY TAKING'
 GO TO READ-DISK-IN. MOVE YEAR-IN TO SWF-YEAR MOVE SEM-IN
 TO SWF-SEM MOVE GRADE-IN TO SWF-GRADE MOVE MARK-IN TO
 SWF-MARK MOVE SF-NO TO SWF-LINK-3.

SEARCH-COURSE-A-2.

SET IN2 TO 1.

SEARCH CRSE-DIRECT AT END DISPLAY 'CHECK A-2' GO TO WRAP-UP
 WHEN COURSE-IN = CRSE-D (IN2) NEXT SENTENCE.

MOVE-A-2.

MOVE INDX-D (IN2) TO SWF-LINK-4.

IF DUMMY-ADDR (1) IS EQUAL TO 2 GO TO MOVE-WRK-1. SUBTRACT
 1 FROM DUMMY-ADDR (1) MOVE DUMMY-ADDR (1) TO CXT. MOVE
 DUMMY-ADDR (CXT) TO SF-NO WR GO TO WRITE-A-2.

MOVE-WRK-1.

MOVE WRK TO SF-NO WR ADD 1 TO WRK.

WRITE-A-2.

REWRITE STUDENT-REC FROM F-REC INVALID KEY DISPLAY
 'CHECK WRITE-A-2' GO TO WRAP-UP.

ADJUST-CLASS-A.

MOVE 1 TO CD MOVE INDX-D (IN2) TO COURSE-KEY READ COURSE-FILE
 INTO COURSE-REL INVALID KEY DISPLAY 'CHECK ADJUST-CLASS A'
 GO TO WRAP-UP. IF ADDR-REL IS EQUAL TO ZEROS GO TO
 ZERO-ADDR-REL. MOVE ADDR-REL TO WORK-KEY READ STUD-WORK-FILE
 INTO ST-WORK-CODE (CD) INVALID KEY DISPLAY 'CHECK-ACA-1' GO
 TO WRAP-UP. MOVE LINK-1 (CD) TO STUD-KEY READ STUDENT-FILE
 INTO F-REC INVALID KEY DISPLAY 'CHECK ACA - 2' GO TO
 WRAP-UP. IF STUD-NO-IN IS NOT GREATER THAN SF-STUD GO TO
 ZERO-ADDR-REL.

CHECK-LINK-2.

IF LINK-2 (CD) IS EQUAL TO ZEROS GO TO MOVE-ZEROS. MOVE
 LINK-2 (CD) TO WORK-KEY EXT ADD 1 TO CD READ STUD-WORK-FILE
 INTO ST-WORK-CODE (CD) INVALID KEY DISPLAY 'CHECK CHECK-L2'
 GO TO WRAP-UP. MOVE EXT TO KEYS-IN (CD) MOVE LINK-1 (CD) TO
 STUD-KEY READ STUDENT-FILE INTO F-REC INVALID KEY DISPLAY
 'CHECK CHK-LINK-2' GO TO WRAP-UP. IF STUD-NO-IN
 IS GREATER THAN SF-STUD

GO TO CHECK-LINK-2. MOVE CD TO AXT SUBTRACT 1 FROM AXT.

MOVE LINK-2 (AXT) TO SWF-LINK-2 MOVE
 WR TO LINK-2 (AXT) WORK-KEY REWRITE ST-WORK-REC FROM
 SWF-AREA INVALID KEY DISPLAY 'CHECK CL-2 - 1' GO TO WRAP-UP.
 MOVE KEYS-IN (AXT) TO WORK-KEY REWRITE ST-WORK-REC
 FROM ST-WORK-CODE (AXT) GO TO DISPLAY-DISK-IN.

MOVE-ZEROS.

MOVE ZEROS TO SWF-LINK-2 MOVE WR TO LINK-2 (CD) WORK-KEY
 REWRITE ST-WORK-REC FROM SWF-AREA INVALID KEY DISPLAY
 'CHECK MOVE-ZEROS 1' GO TO WRAP-UP. MOVE KEYS-IN (CD) TO
 WORK-KEY REWRITE ST-WORK-REC FROM ST-WORK-CODE (CD) INVALID
 KEY DISPLAY 'CHECK MOVE-ZEROS 2' GO TO WRAP-UP. GO TO
 DISPLAY-DISK-IN.

ZERO-ADDR-REL.

MOVE ADDR-REL TO SWF-LINK-2 MOVE WR TO ADDR-REL REWRITE

COURSE-REC FROM COURSE-REL INVALID KEY DISPLAY 'CHECK ZAR 1'
 GO TO WRAP-UP. MOVE WR TO WORK-KEY REWRITE ST-WORK-REC
 FROM SWF-AREA INVALID KEY DISPLAY 'CHECK ZAR 2' GO TO
 WRAP-UP.
 DISPLAY-DISK-IN.
 DISPLAY DISK-IN-REC ' COURSE ADDED'.
 MOVE 1 TO CD GO TO READ-DISK-IN.
 SET-WORK-FILE.
 WRITE LINE-FORMAT FROM HEAD-1 AFTER POSITIONING 0. MOVE ALL
 '-' TO FILER. WRITE LINE-FORMAT FROM HEAD-1 AFTER
 POSITIONING 1.
 MOVE ZEROS TO SWF-LINK-1 MOVE WRK TO WORK-KEY. REWRITE
 ST-WORK-REC FROM SWF-AREA INVALID KEY DISPLAY
 'CHECK SET-WORK-FILE' GO TO WRAP-UP.
 DISPLAY-FILES.
 MOVE FR TO WORK-KEY. READ STUD-WORK-FILE INVALID KEY DISPLAY
 'CHECK 1' GO TO WRAP-UP. IF KEY-ST-WK IS NOT EQUAL TO
 ZEROS WRITE LINE-FORMAT FROM ST-WORK-REC AFTER POSITIONING
 3 ADD 1 TO FR GO TO DISPLAY-FILES. MOVE 1 TO FR.
 DISPLAY-STUD.
 MOVE FR TO STUD-KEY. READ STUDENT-FILE INVALID KEY DISPLAY
 'CHECK 2' GO TO WRAP-UP. IF STUDENT-NO IS NOT EQUAL TO
 ZEROS WRITE LINE-FORMAT FROM STUDENT-REC AFTER POSITIONING
 3 ADD 1 TO FR GO TO DISPLAY-STUD. MOVE 1 TO FR.
 DISPLAY-COURSE.
 MOVE FR TO COURSE-KEY. READ COURSE-FILE INVALID KEY DISPLAY
 'CHECK 3' GO TO WRAP-UP. IF CRSE-KEY IS NOT EQUAL TO ZEROS
 WRITE LINE-FORMAT FROM COURSE-REC AFTER POSITIONING 3 ADD 1
 TO FR GO TO DISPLAY-COURSE.
 WRAP-UP.
 CLOSE CARD-FILE PRINT-FILE COURSE-FILE STUD-WORK-FILE
 STUDENT-FILE DISK-IN.
 STOP RUN.

INSERT GRADES

IDENTIFICATION DIVISION.
 PROGRAM-ID. INSGRADE.
 AUTHOR. GOVIND K P I P L A N I.
 REMARKS.

```

*****
* THIS PRGMA EXPECTS A FUNCTION CARD, WHICH CONTAINS *
* GRADE-SCALE AND ALSO TELLS WHETHER TO GIVE GRADE OR NOT *
* THE FORMAT FOLLOWS. *
* COLS 1 - 3 'YES' IF GRADE TO BE GIVEN *
* COLS 4 - 5 MARKS. EXAMPLE 79, MEANS ANY STUDENT *
* HAVING MORE THEN THIS GIVE HIM THE GRADE *
* THAT FOLLOWS ON THE NEXT COL. *
* COLS 6 GRADE TO BE GIVEN IF THE PREVICUS *
* CONDITION IS MET. *
* COLS 7 - 33 ACCORDING TO THE FORM ABOVE, EVERY THREE *
* COLS. CONTAIN MAPKS AND THE GRADE TO BE *
* GIVEN IF THE STUDENT HAS ABOVE THIS MARK. *
* COLS 78 - 79 THE LENGTH OF GRADING SCALE. *
* COLS 80 'A' TO INDICATE THIS IS THE FUNCTION CARD *
* ..... *
* COLS 1 - 2 'NO' IF GRADES NOT TO BE GIVEN. *
* COLS 80 'A' TO INDICATE THIS IS THE FUNCTION CARD *
*****

```

ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. IBM-360-F75.
 OBJECT-COMPUTER. IBM-360-F75.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.

```

SELECT STUD-FILE-DIRECTORY ASSIGN TO UT-2314-S-SFD.
SELECT CARD-FILE ASSIGN TO UR-2540R-S-CARDS.
SELECT PRINT-FILE ASSIGN TO UR-1403-S-PRINTS.
SELECT DISK-IN ASSIGN TO UT-2314-S-INDISK.
SELECT DISK-OUT ASSIGN TO UT-2314-S-OUTDISK.
SELECT WORK-FILE ASSIGN TO 4 DA-2314-D-SORTWK01.
SELECT STUDENT-FILE ASSIGN TO DA-2314-R-SF
ACCESS IS RANDOM
NOMINAL KEY IS STUD-KEY.
SELECT STUD-WORK-FILE ASSIGN TO DA-2314-R-SWF
ACCESS IS RANDOM
NOMINAL KEY IS WORK-KEY.
SELECT COURSE-FILE ASSIGN TO DA-2314-R-SRF
ACCESS IS RANDOM
NOMINAL KEY IS COURSE-KEY.

```

DATA DIVISION.
 FILE SECTION.
 SD WORK-FILE

```

RECORDING MODE IS F
RECORD CONTAINS 80 CHARACTERS
LABEL RECORD IS STANDARD

```

```

DATA RECORD IS WORKS.
01 WORKS.
   02 DEPT-WORK PICTURE XXX.
   02 STUD-NO-WORK PICTURE 9(06).
   02 FILLER PICTURE X(24).
   02 COURSE-CODE-WORK PICTURE 9(06).
   02 FILLER PICTURE X(41).
FD CARD-FILE
   RECORDING MODE IS F
   RECORD CONTAINS 80 CHARACTERS
   LABEL RECORD IS OMITTED
   DATA RECORD IS CARD-REC.
01 CARD-REC.
   02 FILLER PICTURE X(79).
   02 COL-80 PICTURE X.
FD PRINT-FILE
   RECORDING MODE IS F
   LABEL RECORD IS OMITTED
   RECORD CONTAINS 133 CHARACTERS
   DATA RECORD IS LINE-FORMAT.
01 LINE-FORMAT.
   02 FILLER PICTURE X(133).
FD DISK-IN
   RECORDING MODE IS F
   RECORD CONTAINS 80 CHARACTERS
   LABEL RECORD IS STANDARD
   DATA RECORD IS DISK-IN-REC.
01 DISK-IN-REC.
   02 FILLER PICTURE X(80).
FD DISK-OUT
   RECORDING MODE IS F
   RECORD CONTAINS 80 CHARACTERS
   BLOCK CONTAINS 10 RECORDS
   LABEL RECORD IS STANDARD
   DATA RECORD IS DIS-CUT-REC.
01 DISK-OUT-REC.
   02 PERCENT PICTURE 999.
   02 ST-NO-I PICTURE 9(06).
   02 FILLER PICTURE X.
   02 ST-NAME-I PICTURE X(20).
   02 FILLER PICTURE XXX.
   02 COURSE-CD-I PICTURE 9(06).
   02 SEM-I PICTURE X.
   02 FILLER PICTURE XXX.
   02 GGT PICTURE 999.
   02 FILLER PICTURE X(33).
   02 COL-80-I PICTURE X.
FD STUDENT-FILE
   RECORDING MODE IS F
   RECORD CONTAINS 84 CHARACTERS
   LABEL RECORD IS STANDARD
   DATA RECORD IS STUDENT-REC.
01 STUDENT-REC.

```

```

02 FILLER PICTURE XXX.
02 STUDENT-NO PICTURE 9(06).
02 FILLER PICTURE X(71).
02 STUD-LINK PICTURE 9999.
FD STUD-WORK-FILE
RECORDING MODE IS F
RECORD CONTAINS 22 CHARACTERS
LABEL RECORD IS STANDARD
DATA RECORD IS ST-WORK-REC.
01 ST-WORK-REC.
02 FILLER PICTURE X(22).
FD COURSE-FILE
RECORDING MODE IS F
RECORD CONTAINS 37 CHARACTERS
LABEL RECORD IS STANDARD
DATA RECORD IS COURSE-REC.
01 COURSE-REC.
02 FILLER PICTURE X(04).
02 CPSE-KEY PICTURE 9(06).
02 FILLER PICTURE X(27).
FD STUD-FILE-DIRECTORY
RECORDING MODE IS F
RECORD CONTAINS 2400 CHARACTERS
LABEL RECORD IS STANDARD
DATA RECORD IS STUD-FILE-DIRECT.
01 STUD-FILE-DIRECT.
02 FILLER PICTURE X(2400).
WORKING-STORAGE SECTION.
77 STUD-KEY PICTURE S9(8) COMP SYNC.
77 COURSE-KEY PICTURE S9(8) COMP SYNC.
77 WORK-KEY PICTURE S9(8) COMP SYNC.
77 SW PICTURE 9 VALUE 0.
77 STUD-CODE PICTURE 9(06).
77 ST PICTURE 99 VALUE 1.
77 CALC PICTURE 99.
01 SFD-FILL.
02 STUD-DIRECT OCCURS 200 TIMES INDEXED BY IN2.
03 STUD-NO-D PICTURE 9(06).
03 STUD-INDX-D PICTURE 9(06).
01 FUNCTION-REC.
02 YES-OR-NO PICTURE XXX.
02 SCALING OCCURS 10 TIMES.
03 MARKING PICTURE 99.
03 GRADING PICTURE X.
02 FILLER PICTURE X(44).
02 MAX-SCALE PICTURE 99.
02 FILLER PICTURE X.
01 WORKING-REC.
02 LINK-1 PICTURE 9999.
02 LINK-2 PICTURE 9999.
02 YEAR-W PICTURE 99.
02 TERM-W PICTURE 9.
02 GRADE-W PICTURE X.

```



```

02 MARKS-W PICTURE 99.
02 LINK-3 PICTURE 9999.
02 LINK-4 PICTURE 9999.
PROCEDURE DIVISION.
STARTS.
    OPEN INPUT CARD-FILE OUTPUT PRINT-FILE DISK-IN.
    READ CARD-FILE AT END GO TO DISPLAY-MISSING. IF COL-80
    IS NOT EQUAL TO 'A' DISPLAY 'FUNCTION CARD MISSING' GO TO
    WRAP-UP. MOVE CARD-REC TO FUNCTION-REC.
    DISPLAY FUNCTION-REC.
READ-CARDS.
    READ CARD-FILE AT END GO TO CLOSE-CARD-FILE. IF COL-80 IS
    NOT EQUAL TO 'M' DISPLAY CARD-REC 'WRONG CARD' GO TO
    READ-CARDS. WRITE DISK-IN-REC FROM CARD-REC GO TO READ-CARDS.
CLOSE-CARD-FILE.
    CLOSE CARD-FILE DISK-IN. OPEN INPUT STUD-FILE-DIRECTORY.
    READ STUD-FILE-DIRECTORY INTO SFD-FILL AT END GO TO
    CLOSE-SFD.
    DISPLAY 'OK-1'.
CLOSE-SFD.
    DISPLAY 'OK-2'.
    CLOSE STUD-FILE-DIRECTORY.
SORTING.
    DISPLAY 'OK-3'.
    SORT WORK-FILE ON ASCENDING KEY STUD-NO-WORK COURSE-CCDE-WORK
    USING DISK-IN
    GIVING DISK-OUT.
CHECK-RESULT.
    DISPLAY 'OK-4'.
    IF SORT-RETURN IS NOT EQUAL TO 0 DISPLAY 'SORT UNSUCCESSFUL'
    GO TO WRAP-UP. DISPLAY 'SORT OK'.
OPEN-IF.
    DISPLAY 'OK-5'.
    OPEN INPUT DISK-OUT I-O STUDENT-FILE STUD-WORK-FILE
    COURSE-FILE.
READ-IF.
    DISPLAY 'OK-6'.
    READ DISK-OUT AT END GO TO SET-ST. IF SW IS EQUAL TO 0
    GO TO MOVE-ST-NO. IF ST-NO-I IS NOT EQUAL TO STUD-CCDE GO TO
    MOVE-ST-NO.
MOVE-LINK-3.
    DISPLAY 'OK-7'.
    MOVE LINK-3 TO WORK-KEY GO TO READ-SWF.
MOVE-ST-NO.
    MOVE ST-NO-I TO STUD-CODE.
SEARCH-STUD.
    DISPLAY 'OK-8'.
    SET IN2 TO 1.
    SEARCH STUD-DIRECT AT END DISPLAY 'STUDENT NOT FOUND' GO TO
    WRAP-UP WHEN ST-NO-I = STUD-NO-D (IN2) MOVE STUD-INDEX-D (IN2)
    TO STUD-KEY. READ STUDENT-FILE INVALID KEY DISPLAY
    'CHECK SEARCH-STUD' GO TO WRAP-UP. MOVE STUD-LINK TO
    WORK-KEY.

```

READ-SWF.

DISPLAY 'OK-9'.

READ STUD-WORK-FILE INTO WORKING-REC INVALID KEY DISPLAY
'CHECK READ-SWF' GO TO WRAP-UP. MOVE LINK-4 TO COURSE-KEY.
READ COURSE-FILE INVALID KEY DISPLAY 'CHECK READ-SWF 1' GO
TO WRAP-UP.

CHECKING.

DISPLAY 'OK - 10'.

IF COURSE-CD-1 IS GREATER THAN CRSE-KEY GO TO
CHECK-LINK-3. IF COURSE-CD-1 IS LESS THAN CRSE-KEY
DISPLAY DISK-OUT-REC 'STUDENT NOT TAKING THIS COURSE'
PERFORM READ-IF GO TO CHECKING. COMPUTE
 $CALC = (PERCEN * GOT) / 100$ ADD CALC TO MARKS-W. IF
YES-OR-NO IS EQUAL TO 'YES' GO TO MOVE-GRADE.

REWRITE-SWR.

DISPLAY 'OK - 11'.

REWRITE ST-WORK-REC FROM WORKING-REC INVALID KEY DISPLAY
'CHECK CHECK-LINK-3' GO TO WRAP-UP.
GO TO READ-IF.

CHECK-LINK-3.

DISPLAY 'OK - 12'.

IF LINK-3 IS EQUAL TO ZEROS MOVE 0 TO SW GO TO READ-IF.
MOVE LINK-3 TO WORK-KEY GO TO READ-SWF.

MOVE-GRADE.

DISPLAY 'OK - 13'.

IF MARKS-W IS EQUAL TO MARKING (ST) OR MARKS-W IS GREATER
THAN MARKING (ST) MOVE GRADING (ST) TO
GRADE-W MOVE 1 TO ST GO TO REWRITE-SWR. IF ST IS NOT
EQUAL TO MAX-SCALE ADD 1 TO ST GO TO MOVE-GRADE. DISPLAY
'CHECK FUNCTION CARD - SOMETHING WRONG' GO TO WRAP-UP.

DISPLAY-MISSING.

DISPLAY 'FUNCTION CARD AND DATA MISSING' GO TO WRAP-UP.

SET-ST.

DISPLAY 'OK - 14'.

MOVE 0 TO ST.

DISPLAY-SWF.

DISPLAY 'OK - 15'.

MOVE ST TO WORK-KEY. READ STUD-WORK-FILE INTO WORKING-REC
INVALID KEY DISPLAY 'CHECK DISPLAY-SWF' GO TO WRAP-UP. IF
LINK-1 IS NOT EQUAL TO ZEROS WRITE LINE-FORMAT FROM
WORKING-REC AFTER POSITIONING 3 ADD 1 TO ST GO TO
DISPLAY-SWF.

WRAP-UP.

CLOSE DISK-OUT STUDENT-FILE STUD-WORK-FILE COURSE-FILE
PRINT-FILE.

STOP RUN.

GRADE REPORTS

IDENTIFICATION DIVISION.

PROGRAM-ID. GRADEREP.

AUTHOR. GUVIND K P I P L A N I.

REMARKS.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-360-F75.

OBJECT-COMPUTER. IBM-360-F75.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT STUD-FILE-DIRECTORY ASSIGN TO UT-2314-S-SFD.

SELECT CARD-FILE ASSIGN TO UR-2540R-S-CARDS.

SELECT PRINT-FILE ASSIGN TO UR-1403-S-PRINTS.

SELECT STUDENT-FILE ASSIGN TO DA-2314-R-SF

ACCESS IS RANDOM

NOMINAL KEY IS STUD-KEY.

SELECT STUD-WORK-FILE ASSIGN TO DA-2314-R-SWF

ACCESS IS RANDOM.

NOMINAL KEY IS WORK-KEY.

SELECT COURSE-FILE ASSIGN TO DA-2314-R-SRF

ACCESS IS RANDOM

NOMINAL KEY IS COURSE-KEY.

DATA DIVISION.

FILE SECTION.

FD CARD-FILE

RECORDING MODE IS F

RECORD CONTAINS 80 CHARACTERS

LABEL RECORD IS OMITTED

DATA RECORD IS CARD-REC.

01 CARD-REC.

02 YEAR-SEQ PICTURE 99.

02 FILLER PICTURE X(77).

02 CUL-80 PICTURE X.

FD PRINT-FILE

RECORDING MODE IS F

LABEL RECORD IS OMITTED

RECORD CONTAINS 133 CHARACTERS

DATA RECORD IS LINE-FORMAT.

01 LINE-FORMAT.

02 FILLER PICTURE X(133).

FD STUDENT-FILE

RECORDING MODE IS F

RECORD CONTAINS 84 CHARACTERS

LABEL RECORD IS STANDARD

DATA RECORD IS STUDENT-REC.

01 STUDENT-REC.

02 STUDENT-DEPT PICTURE XXX.

02 STUDENT-NO PICTURE 9(06).

02 STUDENT-NAME PICTURE X(20).

02 STUDENT-DEGREE PICTURE X(04).

```

02 STUDENT-LEVEL PICTURE XX,
02 STUDENT-SEX PICTURE X,
02 STUDENT-STREET PICTURE X(17),
02 STUDENT-APT PICTURE 9(14),
02 STUDENT-TOWN PICTURE X(16),
02 STUDENT-TELEPHONE PICTURE X(07),
02 STUDENT-LINK PICTURE 9999,
FD STUD-WORK-FILE
RECORDING MODE IS F
RECORD CONTAINS 22 CHARACTERS
LABEL RECORD IS STANDARD
DATA RECORD IS ST-WORK-REC,
01 ST-WORK-REC,
02 LINK-1 PICTURE 9999,
02 LINK-2 PICTURE 9999,
02 YEAR-G-IN PICTURE 99,
02 TERM-G-IN PICTURE X,
02 GRADE-G-IN PICTURE X,
02 MARK-G-IN PICTURE 99,
02 LINK-3 PICTURE 9999,
02 LINK-4 PICTURE 9999,
FD STUD-FILE-DIRECTORY
RECORDING MODE IS F
RECORD CONTAINS 2400 CHARACTERS
LABEL RECORD IS STANDARD
DATA RECORD IS STUD-FILE-DIRECT,
01 STUD-FILE-DIRECT,
02 FILLER PICTURE X(2400),
FD COURSE-FILE
RECORDING MODE IS F
RECORD CONTAINS 37 CHARACTERS
LABEL RECORD IS STANDARD
DATA RECORD IS COURSE-REC,
01 COURSE-REC,
02 COURSE-ADDR PICTURE 9999,
02 COURSE-CODE PICTURE 999,
02 COURSE-CODE-SEM PICTURE 999,
02 FILLER PICTURE X,
02 COURSE-DES PICTURE X(25),
02 COURSE-CREDITS PICTURE 9,
WORKING-STORAGE SECTION,
77 YEAR-REQD PICTURE 99,
77 ST PICTURE 9999 VALUE 1,
77 CT PICTURE 9 VALUE 0,
77 WORK-KEY PICTURE S9(8) COMP SYNC,
77 STUD-KEY PICTURE S9(8) COMP SYNC,
77 COURSE-KEY PICTURE S9(8) COMP SYNC,
01 CFD-FILL,
02 CPSE-DIRECT OCCURS 200 TIMES,
03 CPSE-D PICTURE 9(06),
03 INDX-D PICTURE 9(06),
01 ADDR-FORM,
02 FILLER PICTURE X(10) VALUE SPACES,

```

```

02 NO-IN PICTURE Z(06),
02 FILLER PICTURE X(03) VALUE SPACES,
02 NAME-IN,
03 APT-IN PICTURE XXXX,
03 APT-NO-IN PICTURE 9999,
03 FILL-DASH PICTURE X,
03 FILLER PICTURE X(11),
02 FILLER PICTURE X(16) VALUE SPACES,
02 LEVEL-IN PICTURE X(04),
01 COURSES-FORM,
02 FILLER PICTURE X(08) VALUE SPACES,
02 DEPT-IN PICTURE XXX,
02 FILLER PICTURE XXX VALUE SPACES,
02 COURSE-CODE-IN PICTURE 999,
02 FILLER PICTURE X VALUE '-',
02 COURSE-CODE-SEM-IN PICTURE 999,
02 SEM-IN PICTURE X,
02 FILLER PICTURE XX VALUE SPACES,
02 COURSE-DES-IN PICTURE X(25),
02 FILLER PICTURE X VALUE SPACES,
02 YR-IN PICTURE 99,
02 FILLER PICTURE XX VALUE SPACES,
02 COURSE-CREDITS-IN PICTURE 9,
02 FILLER PICTURE X(05) VALUE SPACES,
02 GRADE-IN PICTURE X,

```

PROCEDURE DIVISION.

STARTS.

OPEN INPUT CARD-FILE.

READ-CARDS.

READ CARD-FILE AT END GO TO CLOSE-IT.

IF COL-80 IS NOT EQUAL TO 'A' DISPLAY

'FOR WHICH YEAR YOU NEED GRADE-REPORTS - CHECK COL-80' GO TO
WRAP-UP.

MOVE YEAR-REQ TO YEAR-REQD.

CLOSE-IT.

CLOSE CARD-FILE, OPEN I-O COURSE-FILE STUD-WORK-FILE

STUDENT-FILE INPUT STUD-FILE-DIRECTORY OUTPUT PRINT-FILE.

READ-SFD.

READ STUD-FILE-DIRECTORY INTO CFD-FILL AT END GO TO CLOSE-SF.

CLOSE-SF.

CLOSE STUD-FILE-DIRECTORY.

START-PROCESS.

IF CPSE-D (ST) IS EQUAL TO ZEROS GO TO WRAP-UP. MOVE

INX-D (ST) TO STUD-KEY. READ STUDENT-FILE INVALID KEY

DISPLAY 'CHECK START-PROCESS' GO TO WRAP-UP. MOVE

STUDENT-NO TO NO-IN MOVE STUDENT-NAME TO NAME-IN MOVE

STUDENT-DEGREE TO LEVEL-IN. WRITE LINE-FORMAT FROM ADDR-FORM
AFTER POSITIONING 0. MOVE ZEROS TO NO-IN MOVE SPACES TO

NAME-IN MOVE SPACES TO LEVEL-IN MOVE STUDENT-STREET TO

NAME-IN. WRITE LINE-FORMAT FROM ADDR-FORM AFTER POSITIONING

1. MOVE SPACES TO NAME-IN MOVE 'APT' TO APT-IN MOVE

STUDENT-APT TO APT-NO-IN. WRITE LINE-FORMAT FROM ADDR-FORM

AFTER POSITIONING 1. MOVE STUDENT-TOWN TO NAME-IN. WRITE

```

LINE-FORMAT FROM ADDR-FORM AFTER POSITIONING 1; MOVE
STUDENT-LINK TO WORK-KEY;
DISPLAY STUDENT-LINK ' STUDENT LINK';
READ-SWF;
  DISPLAY 'OK C';
  DISPLAY YEAR-REQ ' YEAR REQ';
  READ STUD-WORK-FILE INVALID KEY DISPLAY 'CHECK READ-SWF'
  GO TO WRAP-UP; IF YEAR-REQD IS NOT EQUAL TO ZEROS GO TO
  CHECK-WORK-YEAR;
  DISPLAY ST-WORK-REC ' ST-WORK-REC';
MOVE-IN;
  DISPLAY 'OK B';
  MOVE LINK-4 TO COURSE-KEY; READ COURSE-FILE INVALID KEY
  DISPLAY 'CHECK MOVE-IN' GO TO WRAP-UP; MOVE YEAR-G-IN TO
  YR-IN MOVE TERM-G-IN TO SEM-IN MOVE GRADE-G-IN TO GRADE-IN
  MOVE COURSE-CODE TO COURSE-CODE-IN MOVE COURSE-CODE-SEM
  TO COURSE-CODE-SEM-IN MOVE COURSE-DES TO COURSE-DES-IN MOVE
  COURSE-CREDITS TO COURSE-CREDITS-IN; IF CT IS EQUAL TO
  ZERO WRITE LINE-FORMAT FROM COURSES-FORM AFTER POSITIONING
  3 MOVE 1 TO CT ELSE WRITE LINE-FORMAT FROM COURSES-FORM
  AFTER POSITIONING 1; IF LINK-3 IS NOT EQUAL TO ZEROS MOVE
  LINK-3 TO WORK-KEY GO TO READ-SWF; ADD 1 TO ST MOVE 0 TO
  CT GO TO START-PROCESS;
CHECK-WORK-YEAR;
  DISPLAY 'OK I';
  IF YEAR-G-IN IS EQUAL TO YEAR-REQD GO TO MOVE-IN; MOVE
  0 TO CT ADD 1 TO ST GO TO START-PROCESS;
WRAP-UP;
  DISPLAY 'OK 2';
  CLOSE COURSE-FILE STUD-WORK-FILE STUDENT-FILE PRINT-FILE;
  STOP RUN;

```

CLASS-LISTS

IDENTIFICATION DIVISION.

PROGRAM-ID: CLASSLIS.

AUTHOR: GOVIND K R I P L A N I.

REMARKS:

```

*****
* THIS PROGRAM PRINTS OUT EITHER CLASS-LISTS OR PUNCHES *
* DECKS MEANT FOR PROFESSORS FOR WRITING GRADES ON THEM, *
* OR BOTH. IT EXPECTS A FUNCTION CARD WHICH TELLS THE *
* PROGRAM WHAT TO DO. *
* *
* FORMAT FOR PUNCHING THE FUNCTION CARD. *
* ----- *
* COL 1 - 2 AL CLASS-LISTS WITH ALL THE STUDENTS. *
* COL 1 - 2 72 CLASS-LISTS WITH ALL THE STUDENTS OF 1972 *
* COL 1 - 2 NO DONT NEED CLASS-LISTS. *
* COL 3 A FIRST SEMESTER ONLY. *
* OR *
* COL 3 3 IN THIS CASE SECOND SEMESTER ONLY *
* *
* COL 4 - 5 NO DONT NEED DECKS. *
* COL 4 - 5 72 NEED DECKS FOR THE STUDENTS TAKING COURSE *
* DURING 1972. *
* COL 6 A FIRST SEMESTER ONLY. *
* OR *
* COL 6 3 IN THIS CASE SECOND SEMESTER ONLY *
* COL 80 INDICATES THIS IS FUNCTION CARD. *
*****

```

ENVIRONMENT DIVISION.

CONFIGURATION SECTION:

SOURCE-COMPUTER: IBM-360-F75.

OBJECT-COMPUTER: IBM-360-F75.

INPUT-OUTPUT SECTION:

FILE-CONTROL:

```

SELECT CARD-FILE ASSIGN TO UR-2540R-S-CARDS.
SELECT PUNCH-FILE ASSIGN TO UR-2540P-S-SYSPUNCH.
SELECT PRINT-FILE ASSIGN TO UR-1403-S-PRINTS.
SELECT COURSE-FILE-DIRECTORY ASSIGN TO UT-2314-S-CFD.
SELECT STUDENT-FILE ASSIGN TO DA-2314-R-SF
ACCESS IS RANDOM
NOMINAL KEY IS STUD-KEY.
SELECT STUD-WORK-FILE ASSIGN TO DA-2314-R-SWF
ACCESS IS RANDOM
NOMINAL KEY IS WORK-KEY.
SELECT COURSE-FILE ASSIGN TO DA-2314-R-SRF
ACCESS IS RANDOM
NOMINAL KEY IS COURSE-KEY.

```

DATA DIVISION.

FILE SECTION.

FD CARD-FILE

RECORDING MODE IS F

```

RECORD CONTAINS 80 CHARACTERS
LABEL RECORD IS OMITTED
DATA RECORD IS CARD-REC.
01 CARD-REC.
02 FILLER PICTURE X(73).
02 COL-80 PICTURE X.
FD PRINT-FILE
RECORDING MODE IS F
LABEL RECORD IS OMITTED
RECORD CONTAINS 133 CHARACTERS
DATA RECORD IS LINE-FORMAT.
01 LINE-FORMAT.
02 FILLER PICTURE X(133).
FD PUNCH-FILE
RECORDING MODE IS F
RECORD CONTAINS 80 CHARACTERS
LABEL RECORD IS OMITTED
DATA RECORD IS PUNCH-FORMAT.
01 PUNCH-FORMAT.
02 FILLER PICTURE X(80).
FD STUDENT-FILE
RECORDING MODE IS F
RECORD CONTAINS 84 CHARACTERS
LABEL RECORD IS STANDARD
DATA RECORD IS STUDENT-REC.
01 STUDENT-REC.
02 STUDENT-DEPT PICTURE XXX.
02 STUDENT-NO PICTURE 9(06).
02 STUDENT-NAME PICTURE X(20).
02 STUDENT-DEGREE PICTURE X(04).
02 STUDENT-LEVEL PICTURE XX.
02 STUDENT-SFX PICTURE X.
02 STUDENT-STREET PICTURE X(17).
02 STUDENT-APT PICTURE XXXX.
02 STUDENT-TOWN PICTURE X(16).
02 STUDENT-TELEPHONE PICTURE X(07).
02 STUDENT-LINK PICTURE 9999.
FD STUD-WORK-FILE
RECORDING MODE IS F
RECORD CONTAINS 22 CHARACTERS
LABEL RECORD IS STANDARD
DATA RECORD IS ST-WORK-REC.
01 ST-WORK-REC.
02 LINK-1 PICTURE 9999.
02 LINK-2 PICTURE 9999.
02 YEAR-G-IN PICTURE 99.
02 TERM-G-IN PICTURE X.
02 GRADE-G-IN PICTURE X.
02 MARK-G-IN PICTURE 99.
02 LINK-3 PICTURE 9999.
02 LINK-4 PICTURE 9999.
FD COURSE-FILE-DIRECTORY
RECORDING MODE IS F

```

```

RECORD CONTAINS 2400 CHARACTERS
LABEL RECORD IS STANDARD
DATA RECORD IS COURSE-FILE-DIRECT.
01 COURSE-FILE-DIRECT.
02 FILLER PICTURE X(2400).
FD COURSE-FILE
RECORDING MODE IS F
RECORD CONTAINS 37 CHARACTERS
LABEL RECORD IS STANDARD
DATA RECORD IS COURSE-REC.
01 COURSE-REC.
02 COURSE-ADDR PICTURE 9999.
02 COURSE-CODE PICTURE XXX.
02 COURSE-CODE-SEM PICTURE XXXX.
02 COURSE-DES PICTURE X(25).
02 COURSE-CREDITS PICTURE 9.
WORKING-STORAGE SECTION.
77 ST PICTURE 9999 VALUE 1.
77 CT PICTURE 9 VALUE 0.
77 WORK-KEY PICTURE S9(8) COMP SYNC.
77 STUD-KEY PICTURE S9(8) COMP SYNC.
77 COURSE-KEY PICTURE S9(8) COMP SYNC.
01 CARD-RECS.
02 YEAR-REQD PICTURE 99.
02 SEM-L PICTURE X.
02 DECK PICTURE 99.
02 SEM-D PICTURE X.
02 FILLER PICTURE X(74).
01 ALPHA-REC REDEFINES CARD-RECS.
02 YR-REQ PICTURE XX.
02 FILLER PICTURE X(78).
01 DECK-FORM.
02 FILLER PICTURE XXX VALUE SPACES.
02 DECK-NO-IN PICTURE 9(06).
02 FILLER PICTURE X VALUE SPACES.
02 DECK-NAME-IN PICTURE X(20).
02 FILLER PICTURE XXX VALUE SPACES.
02 DECK-CODE-1 PICTURE XXX.
02 DECK-CODE-2 PICTURE XXX.
02 ST-SEM-IN PICTURE X.
02 FILLER PICTURE X(39) VALUE SPACES.
02 FILLER PICTURE X VALUE 6.
01 CFI-FILL.
02 CRSE-DIRECT OCCURS 200 TIMES.
03 CRSE-D PICTURE 9(06).
03 INDX-D PICTURE 9(06).
01 ADDR-FORM.
02 FILLER PICTURE X(10) VALUE SPACES.
02 COURSE-CODE-IN PICTURE XXX.
02 FILLER PICTURE X VALUE '-'.
02 COURSE-CODE-SEM-IN PICTURE XXXX.
02 FILLER PICTURE X(05) VALUE SPACES.
02 COURSE-IN PICTURE X(25).

```

```

02 FILLER PICTURE X(18) VALUE SPACES.
02 COURSE-CREDITS-IN PICTURE 9.
02 FILLER PICTURE XXXX VALUE SPACES.
02 YEAR-REQD-IN PICTURE 99.
01 STUD-FORM.
02 FILLER PICTURE X(17) VALUE SPACES.
02 DEPT-IN PICTURE XXX.
02 FILLER PICTURE X VALUE SPACES.
02 NO-IN PICTURE Z(05).
02 FILLER PICTURE X VALUE SPACES.
02 NAME-IN PICTURE X(20).
02 FILLER PICTURE X(05) VALUE SPACES.
02 GRADE-IN PICTURE X.
02 FILLER PICTURE X VALUE SPACES.
02 MARK-IN PICTURE 99.
02 FILLER PICTURE XXX VALUE SPACES.
02 SEM-IN PICTURE X.
PROCEDURE DIVISION.
STARTS.
    OPEN INPUT CARD-FILE OUTPUT PUNCH-FILE.
READ-CARDS.
    READ CARD-FILE INTO CARD-RECS AT END GO TO CLOSE-IT.
    IF COL-50 IS NOT EQUAL
        TO 'A' DISPLAY 'FUNCTION CARD MISSING' GO TO WRAP-UP.
CLOSE-IT.
    CLOSE CARD-FILE.
    OPEN I-O COURSE-FILE STUD-WORK-FILE STUDENT-FILE INPUT
        COURSE-FILE-DIRECTORY OUTPUT PRINT-FILE.
READ-SFD.
    READ COURSE-FILE-DIRECTORY INTO CFD-FILL AT END GO TO
        CLOSE-SF.
CLOSE-SF.
    CLOSE COURSE-FILE-DIRECTORY.
START-PROCESS.
    DISPLAY 'OK - 1'.
    IF INDX-D (ST) IS EQUAL TO ZEROS GO TO WRAP-UP.
    IF CRSE-D (ST) IS EQUAL TO ZEROS GO TO WRAP-UP. MOVE
        INDX-D (ST) TO COURSE-KEY. READ COURSE-FILE INVALID KEY
        DISPLAY 'CHECK START-PROCESS' GO TO WRAP-UP. IF COURSE-ADDR
        IS EQUAL TO ZEROS ADD 1 TO ST GO TO START-PROCESS. IF
        YR-REQD IS EQUAL TO 'NO' GO TO MOVE-CRSE-ADDR. MOVE
        COURSE-CODE TO COURSE-CODE-IN MOVE COURSE-CODE-SEM TO
        COURSE-CODE-SEM-IN MOVE COURSE-DES TO COURSE-IN MOVE
        COURSE-CREDITS TO COURSE-CREDITS-IN. WRITE LINE-FORMAT FROM
        ADDR-FORM AFTER POSITIONING C.
MOVE-CRSE-ADDR.
    MOVE COURSE-ADDR TO WORK-KEY.
    DISPLAY COURSE-REC ' COURSE REC'.
MOVE-IT.
    DISPLAY 'OK - 2'.
    READ STUD-WORK-FILE INVALID KEY DISPLAY 'CHECK SWF - 1' GO
        TO WRAP-UP. MOVE LINK-1 TO STUD-KEY. READ STUDENT-FILE
        INVALID KEY DISPLAY 'CHECK SF - 1' GO TO WRAP-UP.

```

```

DISPLAY ST-WORK-REC ' ST-WORK-REC',
DISPLAY YEAR-PEOD ' YEAR-PEOD',
DISPLAY STUDENT-REC ' STUDENT-REC',
DISPLAY 'OK - 3',
IF YR-PEG IS NOT EQUAL TO 'AL' GO TO CHECK-YEAR,
DISPLAY 'OK - 4',

```

MOVE-INFO,

```

DISPLAY 'OK - 5',
MOVE STUDENT-DEPT TO DEPT-IN MOVE STUDENT-NO TO NO-IN MOVE
STUDENT-NAME TO NAME-IN MOVE GRADE-G-IN TO GRADE-IN MOVE
MARK-G-IN TO MARK-IN MOVE TERM-G-IN TO SEM-IN,
IF CT IS EQUAL TO 0 WRITE LINE-FORMAT
FROM STUD-FORM AFTER POSITIONING 3 ADD 1 TO CT ELSE
WRITE LINE-FORMAT FROM STUD-FORM AFTER POSITIONING 1,
DISPLAY 'OK - 6',

```

CHECK-DECK,

```

DISPLAY 'OK - 7',
IF DECK IS EQUAL TO ZEROS GO TO CHECK-LINK-2, IF DECK IS NOT
EQUAL TO YEAR-G-IN GO TO CHECK-LINK-2, IF TERM-G-IN IS NOT
EQUAL TO SEM-D GO TO CHECK-LINK-2, MOVE TERM-G-IN TO
ST-SEM-IN MOVE STUDENT-NO TO DECK-NO-IN MOVE
STUDENT-NAME TO DECK-NAME-IN MOVE COURSE-CODE TO DECK-CODE-1
MOVE COURSE-CODE-SEM TO DECK-CODE-2 WRITE PUNCH-FORMAT
FROM DECK-FORM,

```

CHECK-LINK-2,

```

IF LINK-2 IS EQUAL TO ZEROS ADD 1 TO ST MOVE 0 TO CT GO TO
START-PROCESS, MOVE LINK-2 TO WORK-KEY GO TO MOVE-IT,

```

CHECK-YEAR,

```

IF YEAR-PEOD IS NOT EQUAL TO YEAR-G-IN GO TO CHECK-DECK,
IF TERM-G-IN IS EQUAL TO SEM-L GO TO MOVE-INFO,
GO TO CHECK-DECK,

```

WRAP-UP,

```

CLOSE PUNCH-FILE,
CLOSE COURSE-FILE STUD-WORK-FILE STUDENT-FILE PRINT-FILE,
STOP RUN,

```

ADDRESS CHANGES

IDENTIFICATION DIVISION.

PROGRAM-ID, ADDRCHNG.

AUTHOR, GOVIND K R I P L A N I.

REMARKS.

ENVIRONMENT DIVISION,

CONFIGURATION SECTION,

SOURCE-COMPUTER, IBM-360-F75.

OBJECT-COMPUTER, IBM-360-F75.

INPUT-OUTPUT SECTION,

FILE-CONTROL.

SELECT STUD-FILE-DIRECTORY ASSIGN TO UT-2314-S-SFD.

SELECT CARD-FILE ASSIGN TO UR-2540R-S-CARDS.

SELECT STUDENT-FILE ASSIGN TO DA-2314-R-SF

ACCESS IS RANDOM

NOMINAL KEY IS STUD-KEY.

DATA DIVISION.

FILE SECTION.

FD STUD-FILE-DIRECTORY

RECORDING MODE IS F

RECORD CONTAINS 2400 CHARACTERS

LABEL RECORD IS STANDARD

DATA RECORD IS STUD-FILE-DIRECT.

01 STUD-FILE-DIRECT.

02 FILLER PICTURE X(2400).

FD CARD-FILE

RECORDING MODE IS F

RECORD CONTAINS 80 CHARACTERS

LABEL RECORD IS OMITTED

DATA RECORD IS CARD-REC.

01 CARD-REC.

02 STUD-NO-C PICTURE 9(06).

02 STUD-INFO-C PICTURE X(71).

02 FILLER PICTURE XX.

02 COL-80 PICTURE X.

FD STUDENT-FILE

RECORDING MODE IS F

RECORD CONTAINS 84 CHARACTERS

LABEL RECORD IS STANDARD

DATA RECORD IS STUDENT-REC.

01 STUDENT-REC.

02 FILLER PICTURE X(84).

WORKING-STORAGE SECTION,

77 STUD-KEY PICTURE S9(8) COMP SYNC.

01 STUD-REC.

02 FILLER PICTURE XXX.

02 STUDENT-NO PICTURE 9(06).

02 STUDENT-INFO PICTURE X(71).

02 STUDENT-LINK PICTURE 9(04).

01 SFD-FILL.

02 STUD-DIRECT OCCURS 200 TIMES INDEXED BY IN2.

03 STUD-NO-D PICTURE 9(06).

03 STUD-INDX-D PICTURE 9(06).

PROCEDURE DIVISION.

OPEN-SFD.

OPEN INPUT STUD-FILE-DIRECTORY.

READ-SFD.

READ STUD-FILE-DIRECTORY INTO SFD-FILL AT END GO TO

CLOSE-SFD.

CLOSE-SFD.

CLOSE STUD-FILE-DIRECTORY. OPEN INPUT CARD-FILE 1-0
STUDENT-FILE.

READ-CF.

READ CARD-FILE AT END GO TO WRAP-UP. IF COL-80 IS NOT EQUAL
TO 'C' DISPLAY 'WRONG CARD' GO TO READ-CF.

SEARCH-STUD.

SET IN2 TO 1.

SEARCH STUD-DIRECT AT END DISPLAY 'STUDENT NOT FOUND' GO
TO WRAP-UP WHEN STUD-NO-C = STUD-NO-D (IN2) NEXT SENTENCE.

MOVE-INFO.

MOVE STUD-INDX-D (IN2) TO STUD-KEY. READ STUDENT-FILE INTO
STUD-REC INVALID KEY DISPLAY 'CHECK MOVE-INFO' GO TO WRAP-UP.
DISPLAY STUD-REC ' OLD RECORD'.

MOVE 000005 TO STUDENT-NO MOVE 0001 TO STUDENT-LINK.

MOVE STUD-INFO-C TO STUDENT-INFO. REWRITE STUDENT-REC FROM
STUD-REC INVALID KEY DISPLAY 'CHECK MOVE-INFO - 2' GO TO
WRAP-UP.

DISPLAY STUD-REC ' CHANGED RECORD'.

GO TO READ-CF.

WRAP-UP.

CLOSE CARD-FILE STUDENT-FILE. STOP RUN.

PROGRAM FUNCTIONS

CREATE COURSE-FILE AND ITS DIRECTORY
(CR.CRSE.FILE)

Function: The program creates the Course-file and its directory on a Disk. The file contains the courses offered by the department/school.

Input: Card type 01.

Output: 1. Course-file.
2. Course-file-directory.

UPDATE COURSE-FILE AND ITS DIRECTORY
(UP.CRSE.FILE)

Function: The program adds new courses to the Course-file.
The courses are added if the department decides
to offer more courses after the initial set-up
of the Course-file.

Input: Card type 01.

Output: 1. Updated Course-file.
2. Updated Course-file-directory.

CREATE PERFORMANCE-FILE (CR.PERF.FILE)

Function: The program creates the Performance-file with dummy records. The program is run after the Course-file is created. The Performance-file is used to store the students' performance in each class. One record per course is used.

Input: Card type F1.

Output: Performance-file with dummy records.

CREATE STUDENT-FILE, ITS DIRECTORY AND LINK
WITH PERFORMANCE-FILE AND COURSE-FILE
(CR.STUD.LINK)

Function: The program creates the Student-file with students' information and links the records of this file with the connecting records in the Performance-file and the Course-file, created by previous programs.

Input: Card type 02 and 03.

Output: 1. Student-file.
2. Student-file-directory.
3. Pointers stored in the three files to link the connecting records.

UPDATE STUDENT-FILE, PERFORMANCE-FILE AND
COURSE-FILE (UP.THRE.FILE)

Function: The program updates the three files by
adding and deleting the courses due to course
changes made by the students.

Input: Card type 03 and 04.

Output: Updated Student-file,
Performance-file and
Course-file.

INSERT MARKS AND/OR GRADES (INS.MARK.GRADE)

Function: The program updates the Performance-file by adding the marks or adding and grading the students according to the option selected. The grading-scale is provided by the function card.

Input: Card type P6 and F2.

Output: Updated Performance-file.

PRINT GRADE REPORTS (PR.GR.REP)

Function: The program uses the database to print the Grade Reports. The Grade Reports can be printed out for one or both the semesters. The Function card indicates the option selected.

Input: Card type F3.

Output: Grade Reports.

PRINT CLASS LISTS (PR.CL.CD)

Function: The program prints the class lists or punches cards for the students one per course. The option selected is indicated on the Function card. The cards are meant for marking grades by the Professors.

Input: Card type F4.

Output: Class lists or Card type P6.