A COMPUTER-AIDED PERSONALIZED SYSTEM OF INSTRUCTION

by
Yiu-Man Leung

A thesis
presented to the University of Manitoba
in partial fulfillment of the
requirements for the degree of
Master of Science
in
Electrical Engineering

Winnipeg, Manitoba, Canada
August 1988

ISBN 0-315-48057-2

A COMPUTER-AIDED PERSONALIZED SYSTEM OF INSTRUCTION

by

YIU-MAN LEUNG

A thesis submitted to the Faculty of Graduate Studies of
the University of Manitoba in partial fulfillment of the requirements
of the degree of

MASTER OF SCIENCE

ABSTRACT

Computer–Aided Personalized System of Instruction (CAPSI) is a new automated teaching and learning method rooted in the Personalized System of Instruction (PSI) and the learning–reinforcement theory, involving highly systematic, interactive, student–participated teaching and evaluation through (1) structured course material with clear specification of objectives, (2) frequent and immediate reinforcement through instructor–student and student–student markings, (3) minimization of punishment, and (4) self–pacing by the student. CAPSI is a technological educational innovation with the combination of physical and behavioral technologies to produce an engineering approach to education. Unlike PSI, CAPSI adopts computer technology with a finite–state modelling implementation to free the instructor from administration, promote individualized learning, and provide tools for the analysis of the dynamical educational process and its optimization. Developed and extensively tested over the past five years at the University of Manitoba, CAPSI has gone through a natural evolution from a single terminal to multiple terminals, with direct and remote links, and electronic mailing and messaging. CAPSI has been used successfully, effectively, and economically in short–distance (on campus) and long–distance (off campus) teaching, thus extending the fundamental concepts of education from a physical classroom to a virtual classroom and even to a virtual campus which eliminates the need for the instructor and the students to be at the same place and at the same time.

ACKNOWLEDGEMENTS

I would like to extend my sincere thanks and appreciation to Dr. Witold Kinsner, my advisor, for his supervision, teaching, assistance, perseverance, and patience during the course of my research and also for his suggestions of more than one research topics.

Special thanks to Dr. Joseph Pear and Dr. Kinsner who have contributed the main research ideas to the development of the CAPSI system. Also thanks go to Frank Herzog for the initial coding of the program. In addition, thanks to Dr. J. Poltz for his comment on this thesis.

I would like to thank the financial support from the Industrial Applications of Microelectronics Centre Inc., the University of Manitoba Academic Development Fund, the Natural Sciences and Engineering Research Council of Canada, and private funds.

The last but not least, I thank my parents and family for their financial and emotional support of my stay in Canada, and I finally acknowledge the Christian faith that carries me through difficulties and hardship.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Human beings have high capacity to both teach and learn knowledge as well as wisdom in a concatenated process of instruction (i.e., the first person instructs the second, the second then instructs the third, and so on). The formal educational structure preserves the concatenated process of instruction in such a way that a qualified scholar with more knowledge and expertise but without any guarantee of effectiveness teaches others. The process of education by far is only localized both spatially in classroom and temporally of class period.

The concatenated process of instruction within contemporary educational institutions has four principles:

1. Knowledge is organized hierarchically from trivial level to advanced level.
2. Knowledge has no boundary.
3. Learning is not only unidirectional from the scholar to the learner, but also bidirectional from the learner to the scholar and from the process of teaching.
4. Peer learning can occur at the same level of knowledge.

Since peer teaching and learning are not commonly recognized and utilized within classes of conventional education, Fred S. Keller [Kell68] introduced the Personalized System of Instruction (PSI) as a systematic educational approach to enhance the utilization of peer teaching and learning among

students. Also, PSI is firmly based on the reinforcement theory arose from the applied and behavioral psychology advocated by B. F. Skinner ([Skin53], [Skin54], [Skin61]).

1.1 Keller's Personalized System of Instruction

First formally described by Keller in his classic paper "Goodbye, Teacher ..." [Kell68], this teaching method has come to be recognized as an important applied contribution of psychology. Keller himself received the Distinguished Contributions to Education in Psychology Award from the American Psychological Association in 1970 and received the Association's Distinguished Contribution for Applications in Psychology Award in 1976. Since then, thousands of instructors in most parts of the world have given PSI courses of almost every discipline; hundreds of thousands of students have taken them; and hundreds of researchers have published papers on this teaching method. PSI has been one of the most widely used and thorough researched forms of instruction in use since 1968.

PSI has usually been associated with five defining characteristics [Kell68]:

1. The unit perfection requirement (mastery learning) for advance, which lets the student go ahead to new material only after demonstrating mastery of that which preceded;
2. The go-at-your-own-pace (self-pacing) feature, which permits a student to move through the course at a speed commensurate with his/her ability and other demands upon his/her time;
3. The related stress upon the written word in instructor-student

communication;

4. The use of proctors, which permits repeated testing, immediate scoring, almost unavoidable tutoring, and a marked enhancement of the personal-social aspect of the educational process; and, finally,
5. The use of lectures and demonstrations as vehicles of motivation, rather than sources of critical information.

Each feature has some natural corollaries.

A mastery requirement means that a student can answer a large percentage of the questions about the course material. The course material is broken down into small sequential units with clear specification of objectives. The unit tests are designed to increase in difficulty gradually so as to minimize the probability of failing a test. The immediate feedback and frequent reinforcement, the rough equivalent of *reward*, on the tests are provided. In case of not passing a test, the student has the only penalty to restudy and rewrite another test on that unit. Thus, the minimization of punishment is achieved by the well designed unit tests and by the replacement of the word *fail* from *restudy*. The student is informed exactly how mastery of the course material will be assessed, and he/she must demonstrate mastery of each unit before proceeding to the next one.

Given with the first criterion of mastery requirement, the second characteristic of go-at-your-own-pace feature is mandatory. Individuals have different schedules and time requirements to excellently master a given subject matter. In the traditional education, the time required for all students to achieve their excellence is constant, and this approach produces the grade distribution normally found. On the contrary, PSI provides an

alternative to permit individual students to move through the course at their own rates within the last day of the course set by the academic institution.

The last three features follow indirectly from the first two. Once self-pacing is allowed, it will occur and be incompatible with a lockstep lecture approach. Written material becomes the major informational source for the instructor-student communications. The written word allows the objectives of all unit tests clearly described by the instructor, and allows the test clearly presented by the students.

Given a number of students who are engaged in working at different speeds, testing in repetitive fashion, and dealing with a wide range of material at any point in time, an instructor must find proctors or tutors to help out during the course. The proctors, who provide nearly limitless individual attention to the students, also benefit through learning by teaching, and allow the instructor to devote more time to those individuals and problems that require the instructor's level of expertise. The proctor is not only an essential feature but perhaps the most valuable contribution of PSI.

Finally, the function of lectures is substantially different. As lecturing is no longer the major commitment, the instructor can meet individual students when they need help, and becomes the roles of a creator of course material and a manager of a learning system. Lectures are used to motivate rather than to supply essential information to students.

In conclusion, PSI has proven superior to the conventional teaching methods [ShRS82]. Its superiority over conventional methods is accounted for the three of its five features: the go-at-your-own-pace feature, the unit-perfection requirement, and the use-of-proctors feature. As well, PSI is based on well-known principles of positive reinforcement in learning theory.

1.2 PSI and Reinforcement Theory

Unlike some educational innovations, PSI is rooted in a system: learning–reinforcement theory. In the language of the psychologist, the student is reinforced by his/her successes and will tend to continue the behavior that leads to more successes. More formally, reinforcement is a process whereby a particular behavior is strengthened, making it likely that the behavior will occur more frequently. The stimulus that strengthens the behavior is called a reinforcer, either positive or negative. A positive reinforcer is something pleasant -- a piece of candy, perhaps, or a word of praise. A negative reinforcer is an unpleasant stimulus that is removed when a particular behavior is performed, making it likely that the behavior will be repeated. PSI fully applies the positive reinforcers and possibly eliminates the negative reinforcers for students to achieve their successes of learning.

The reinforcement theory applied in PSI holds that a learner's progress depends on three things:

1. the information, materials, and situation presented;
2. the performance required; and

3. the feedback provided.

All three components are critical and any one of them neglected is inadequate.

Firstly, the PSI materials are presented accordingly to the reinforcement theory. Small work units lead to greater density of reinforcement than do large ones in the lecture system. Since textbooks usually lack the built–in reinforcements, the PSI study questions and objectives are arranged in schedule of reinforcements that will encourage the students to learn the course materials in the allotted time. The reinforcement theory suggests that the students will lose interest for infrequent reinforcements and will satiate for too frequent reinforcements. Basically the function of the unit in PSI is to allow students to see their progresses through the course and hence be rewarded for learning the manageable, well–defined materials.

Secondly, the students are rewarded and motivated as their performance reaching or moving towards the final A. In PSI, each unit completed produces a fraction of their final A. This token of reward will directly strengthen their engagement in the testing–grading process, and indirectly help them with their study. The students are positively reinforced by advancing the course in unit by unit to achieve the final A, and negatively reinforced to restudy rather than to fail an unmastered unit.

Thirdly, immediate grading and feedback of unit tests encourage the reinstatement of responses made to questions and to strengthen further those that were correct. As well, students immediately receive close

attention and approval regarding their results and performance. The students may need to further clarify or defend their answer which may sharpen their discriminations and lead to a refinement of their concepts. The immediate response of grading and feedback is critical and made possible by the help of student proctors.

The proctors are also reinforced throughout the process of marking. The proctors are treated with respect regarding to their approval, judgement, and advice. Proctors not only receive credit points, but also benefit from increasing their knowledge through marking and teaching. Also, proctors are reinforced by the signs of individual student progress, which can be a measure of the proctors' own success.

At last, the instructor is also reinforced by the successful operation of the PSI system as a whole and by the satisfaction in the academic progress of each student.

To conclude, the existence of and reliance upon the reinforcement theory is the most distinguishing feature of PSI.

1.3 History of PSI

With all the distinguishing features and theoretical foundation, PSI is a powerful and innovative teaching method, but why it is currently one of the most outstanding educational innovations in higher education. Hence, a review of historical background is provided to respond the inquiry.

PSI was developed and implemented by F. Keller for the first time in 1962, together with J. G. Sherman, R. Azzi, and C. Martuscelli, in a course given at Columbia University, and the following year in Brasilia, within the context of an introductory course in psychology.

PSI first began to excite interest during 1965–67 when it was tried at Arizona State University and discussed at conferences such as those of the American Psychological Association and American Educational Research Association. Perhaps the most influential publication was Keller's "Goodbye, Teacher ..." which was read by many people, yet not all of them psychologists. B. A. Green, Jr., a physicist at Massachusetts Institute of Technology, and B. V. Koen, a nuclear engineer at the University of Texas at Austin, started PSI courses and published their experience accounts.

The widespread of PSI usage in physics and engineering courses might be related to the first large–scale use of PSI in physics took place at M.I.T. — one of the leading engineering schools in the United States. The M.I.T. experience was well publicized in physics journals ([Gree71], [FHBT76]) and at professional conferences, which would have attracted the attention of physicists and engineering instructors. It was possible that M.I.T.'s pioneering effects in PSI might have served as a stimulus to the diffusion of this innovation.

But PSI attracted people from other disciplines as well. Workshops and conferences attracted instructors from the diversity of disciplines as well as the heterogeneous character of the institutions. The wide range of institutions were from junior and community colleges, vocational schools,

secondary schools, and educational institutions other than four-year colleges and universities.

In June 1971 the *PSI Newsletter* began publication and chronicled the development of PSI in 31 issues through 1979. It was designed to facilitate communication between instructors implementing PSI courses. Furthermore, it led to the establishment of the Center for Personalized Instruction in September 1973.

The Center served an information clearing-house function, surveying PSI courses in different disciplines and at all educational levels throughout the country. Also, the Center organized workshops and national conferences, published the *PSI Newsletter* and other publications, probably the most important publication was the *Journal of Personalized Instruction*.

Grants from federal agencies, private foundations, and local institutions supported a wide range of implementation and research projects. Literally hundreds of supported projects were reported in the *PSI Newsletter*. Simultaneously, national and international agencies sponsored workshops, conferences, and course development efforts in Brazil, Venezuela, Chile, Argentina, Mexico, Germany, the Netherlands, English, Samoa, and India.

The activity level was high and the documentation of events was exhaustive. A few numbers should be able to characterize what happened. In 1972 the *PSI Newsletter* reported that 190 PSI courses existed; by 1979

over 5,000 courses existed. The mailing list of those involved with PSI exceeded 10,000. In 1973, an extensive literature search resulted in just over 300 articles reporting PSI research and/or implementation; by 1979 approximately 3,000 articles of this nature were reported.

Statistics could not convey the diversity of applications that developed during the '70s. In 1972, 173 of 190 courses reported were in psychology, physics, engineering, mathematics, chemistry, and biology. The topics reported at the fifth national conference in 1979 included community college education; elementary, intermediate, and secondary education; adult, continuing, and vocational education; health, nursing, and medical education; home–based instruction and special education; and teaching in business, prisons, industry, banking, and military. More, papers reported in the level of four–year college were sciences, social sciences, languages, humanities, and engineering.

In summary, three major factors have brought about this extraordinary propagation of PSI activities. Firstly, the Center for Personalized Instruction, with headquarters in Washington, DC. from 1976 to 1979, publicized the periodicals of the *PSI Newsletter* and *Journal of Personalized Instruction*. Secondly, a series of yearly conferences was held on the subject of personalized instruction with the aim of encouraging professionals to share their experience with the method. Finally, certain authors attempted to integrate and systematize the results of hundreds of case studies reported in the bibliographies. As described from above, PSI has widely grown in its popularity and acceptance in higher education because of its distinguishing features and theoretical foundation as well as its

supportive research evidences.

1.4 Research on PSI and its Superiority

The original PSI method has a considerable body of research data supporting its effectiveness. The most noteworthy research efforts were from the following: Ryan [Ryan74]; Taveggia [Tave76]; Robin [Robi76]; Block and Burns [BlBu76]; Hursh [Hurs76]; Kulik, Kulik, and Smith [KuKS76]; Johnson and Ruskin [JoRu77]; Kulik, Kulik, and Cohen [KuKC79]; Sherman [ShRS82a]; and Benaim ([Bena84a], [Bena84b]). These authors did comparison in the majority of parameters and proved that PSI is superior to the traditional method.

Ryan [Ryan74] used the grade distribution, dropouts, experimental comparisons to other methods, and the students' opinions as the determining criteria of effectiveness.

Taveggia [Tave76] reviewed 14 studies carried out from 1967 to 1974 and compared the effectiveness of PSI in terms of the students' performance in examinations. The major conclusion suggested that, when evaluated by average student performance on course content examinations, PSI has proven superior to the conventional teaching methods with which it has been compared. Taveggia examined and suggested that three of five features probably account for the superiority of PSI over the conventional methods: the go–at–your–own–pace feature, the unit–perfection requirement, and the use–of–proctors feature.

Johnson and Ruskin [JoRu77] referred to the study results of 39 articles reported by Kulik *et al.* [KuKS76] and adopted the following criteria for the determination of effectiveness: final grades in the course, retention and transfer of learning, and an overall evaluation of the course by the students.

Kulik *et al.* [KuKC79] based on a systematic meta–analysis, the application of statistical methods to results from a large collection of individual studies, of 75 comparative studies relating to five major types: final examination scores, instructor–assigned course grades, course ratings, course completions, and student study time. The analysis established that PSI generally produces superior student achievement, less variation in achievement, and higher student ratings in college courses, but does not affect course withdrawal or student study time in these courses. The analysis also showed that PSI's superiority can be demonstrated in a variety of course settings with a number of different research designs. Certain settings and research designs, however, produce especially sharp differences between PSI and conventional courses.

Furthermore, a collection of 40 comprehensive studies chosen from more than 3,000 PSI literature was compiled by Sherman *et al.* [ShRS82a]. The full ranges of research strategies and results for the comparison of PSI with traditional and other techniques of teaching and learning were critically discussed in this volume and literature by Benaim ([Bena84a], [Bena84b]). In summary, the comparisons were mainly done on the basis of the following criteria:

1. final grades,

2. grade distribution,
3. dropouts,
4. cost,
5. retention of learning,
6. transfer of learning,
7. student attitudes or opinions,
8. instructor attitudes or opinions,
9. tutor attitudes or opinions, and
10. cost–benefit.

In general, PSI compares favorably on all of them.

Still, a lot of other PSI studies can be found on the reference lists of the above mentioned authors and by the search of library databases (e.g., *Psychological Abstracts, Comprehensive Dissertation Index,* and *Research in Education*), and recent issues of major disciplinary and interdisciplinary journals and magazines on teaching (e.g., *Journal of Chemical Education, Journal of College Science Teaching, Journal of Personalized Instruction, Teaching of Psychology, Proceedings of the 1st, 2nd, 3rd, and 4th National Conferences, Machine–Mediated Learning, Canadian Journal of Educational Communication, IEEE Transaction on Education, PSI:41 Germinal Papers* and *PSI:48 Seminal Papers*). Above all, PSI has an existing body of research indicating that PSI is at least as effective as other teaching methods in higher education, but PSI still needs to evolve with the advanced communication and computer technologies and to integrate new ideas and philosophy to enhance its effectiveness and capabilities.

1.5 PSI to CAPSI

PSI lends itself well to computerization because it is a highly systematic procedure. Computer–Aided PSI, CAPSI, is an application of computer as management tools to PSI with highly feasible abilities as follows:

1. to provide instructional service on a mass scale;
2. to generate test and learning materials;
3. to select proctors, teaching assistant, or instructor for marking;
4. to keep track of each student's scholar or learner state relative to every other student on a moment–to–moment basis; and
5. to provide structure for the proctor–learner interactions with records, thus providing a well–designed research database for the analysis of dynamical education process and its optimization, and for the answer of some research questions.

Moreover, CAPSI, like its parent PSI, is inherent with all the outstanding principles of PSI as described before. Above all, CAPSI is fundamentally different from PSI with regard to the novelty and philosophy of CAPSI.

1.6 The Novelty of CAPSI

CAPSI is a generalized computer-aided teaching and learning method in which the computer is involved in the technological innovation process of education. This method is a natural evolution of PSI, its variations, and complementary teaching methods in different educational environments, with emphasis on engineering concepts embedded in CAPSI. Therefore,

CAPSI is fully inherited from PSI of the reinforcement theory, in which the student is not a spectator or sponge but an active and systematic participant, founded on the concept of behavioral and applied psychology. The success of CAPSI is due to the explicit incorporation of engineering concepts, behavioral technology, and physical technology.

CAPSI is a highly–structured form of communications to fully incorporate and utilize the technology of computer communications. With the usage of electronic mailing and messaging system of the mainframe computer that CAPSI is run on, the method shows considerable promise for effectively and economically delivering on–campus and long–distance education. CAPSI can be used not only in a physical classroom, but also in a virtual classroom ([Hilt86], [KiPe88a]), with no scheduled classes and specific location, and even on a virtual campus. Local area networks can also be used to further extend CAPSI to allow stand–alone implementations [KiPe88a]. In a broad sense, CAPSI can also further exploit all aspects of computer communications technology such as on-line databases, bulletin board systems, and computer conferencing systems.

Data obtained by CAPSI is important for answering some research questions. Providing with a complete and readily accessible record of all testing and marking interactions, CAPSI makes possible to thoroughly monitor, analyze, and evaluate a significant portion of the behavior and learning in the course. Also, CAPSI provides a basis for a more further formal study [KiPe88a] of the modelling, parameter estimation, and optimization of the dynamical educational process. This should be useful in learning how to improve the educational process, including the

instructional presentation, upgrading of the objective, enhancement of the students' learning and long-term retention of the material in a given course.

Based on the engineering approach, CAPSI adopts the modular approach which does not rule out any device or procedure but ideally incorporates it in a manner that maximizes its effectiveness. For example, video tapes, lectures, discussions, laboratories, and term projects are incorporated and used whenever desirable and feasible.

Finally, CAPSI opens the future of computer-aided instruction into a new stage, in which the computer will become more intimately involved in the educational process:

1. by developing of an authorizing system [NATA81] for generating course material, study objectives, and test questions; and
2. by creating of a knowledge base using the recent approach of knowledge representation and knowledge engineering for intelligent tutors [Wool87] to assist in marking tests.

This research in artificial intelligence will extend CAPSI to a new tool for the teaching and learning of design, as defined by Pear and Kinsner [PeKi87].

In conclusion, CAPSI has demonstrated its novelties and advantages over its ancestor PSI. Indeed, CAPSI has shown evidences and proven itself as a powerful, innovative, technological educational method in both teaching and learning. Realizing its full potential, CAPSI is advancing the technological innovation process of education.

1.7 Motivation

The original motivation of implementing the CAPSI program is to computer manage the PSI course of Behavior Modification Principles at the University of Manitoba. Since it is inefficient and costly to provide PSI instructional service on a mass scale, computer is used to solve the problems and to assist the functions of communications, management, measurement, quality control, and research. With the availability of inexpensive computer system, CAPSI is intended to provide high quality education at a low cost. It was later discovered that CAPSI could be very effective in providing long–distance education. Thus, students are provided an easy accessible educational environment in both substantial learning and teaching at any time and from any place. The implementation goal of CAPSI is to provide the students with the simplicity and reliability of using the program and to provide the instructor or programmers with the flexibility and expansibility of developing the program. By providing new significantly enhanced features and new research and development tools and ideas, CAPSI is an attempt to apply the engineering concepts, behavioral technology, communication technology, and computer technology to advancing the technological innovation process of education.

1.8 Thesis Objective

The objectives of this thesis are:

1. To identify the limitations of the existing PSI approach.
2. To computerize the PSI into CAPSI.

3. To develop CAPSI rather than to conduct systematic research on its effectiveness.

4. To exploit the computer communications technology for structuring man–man interactions as the following sub–objectives:
   a) To facilitate long–distance education.
   b) To eliminate the spatial and temporal restrictions of both on–campus and off–campus education.
   c) To provide ability with the formation of virtual classroom and even virtual campus.
   d) To promote individualized learning in time and space.

5. To utilize the computer technology as a research tool as the following sub–objectives:
   a) To monitor, record, and analyze research data in courses.
   b) To provide a basis for modelling, parameter estimation, and optimization of the dynamical educational process.

6. To incorporate the engineering concepts, behavioral and physical technologies into CAPSI.

7. To conceptualize CAPSI as a modular approach to ideally incorporate other effective tools and procedures that complement to CAPSI.

8. To present the design and implementation of CAPSI.

9. To demonstrate CAPSI at least as effective as other techniques and traditional of teaching and learning.

10. To attempt to enhance the educational process through the technological innovation.

1.9 Thesis Organization

This thesis addresses a generalized computer–aided teaching and learning method called CAPSI. It is introduced in Chapter I with the background, philosophy, and superiority of its ancestor PSI. This chapter also describes the novelty of CAPSI and presents what are the motivation and objectives behind CAPSI. The next chapter reviews the learning environments and other teaching methods. It is then followed by Chapter III on how CAPSI is used in courses. In Chapter IV, the design of CAPSI is described in terms of its engineering concepts, problem specification, database, and finite–state machine modelling. Furthermore, a description of its implementation in PL/I language running on the Amdahl mainframe computer is presented in Chapter V. In the second last chapter, the experimental results of CAPSI are described along with their discussions. Finally, conclusions for the CAPSI method are drawn and recommendations for further research and development are suggested in the last chapter of the thesis.

CHAPTER II

LEARNING ENVIRONMENTS AND TEACHING METHODS

With the introduction in the previous chapter, CAPSI is further described in Chapter II as an innovative teaching method that can be used in various learning environments. As well, CAPSI is compared with other educational procedures and devices indicating how they are compatible with CAPSI. Since CAPSI adopts a modular approach, other procedures and devices should been seen as potential components of and complementary to CAPSI rather than as being in opposition to it.

2.1 Educational Learning Environments

PSI and CAPSI can be applied in different educational environments classified in terms of Jung's psychological typology [SiHa80], such as:

1. Sensing-Thinking type – students are occupied by organized work in a instructor–mediated and activity–oriented environment.
2. Sensing–Feeling type – students share and interact with others in a friendly and supportive environment.
3. Intuitive–Thinking type – students develop their critical thinking in a stimulating and challenging environment.
4. Intuitive–Feeling type – students create their own learning activities in a flexible and innovative environment.

2.2 Review of other teaching methods

CAPSI is a modular approach which can incorporate rather than oppose any other approach that can advance the educational goal of CAPSI.

2.2.1 Programmed Instruction (PI)

Programmed instruction is closely related to CAPSI. The material is presented sequentially in small units, assessment of student program is frequent, and feedback to the student is immediate. Active responding and student self–pacing are emphasized. However, no instructor or proctors are employed.

2.2.2 Self–Paced Instruction (SPI)

Self–paced instruction is another approach closely related to CAPSI. The structured material is presented in the form of a study guide. The speeds of students to learn and advance in the course are varied. Proctors are used to evaluate the tests.

2.2.3 Computer–Assisted Instruction (CAI)

No instructor is required in computer–assisted instruction. A student interacts with a computer which presents drills, practice exercises, and tutorial sequences to the student, and perhaps to engage the student in a dialog about the substance of the instruction. The computer also keeps track

of the student's progress.

2.2.4 Computer–Managed Instruction (CMI)

The computer–managed instruction is similar to CAI. The roles of the computer is to assist the instructor in managing instruction and to direct the entire instructional process.

2.2.5 Self and Peer Marking

The course is prepared by the instructor, but the markings of assignments, laboratory reports, and the mid–term and final examinations are done by the students themselves and their peers.

2.2.6 Guided Design (GD)

This is a teaching–learning system approach based on a set of open–ended problems for students to study at home and for small groups of students to make decision in class with the guidance of instructor. The attributes of Guided Design includes: creating the desire to know, development of team–work skills, and integration of old and new learning.

2.3 Other Technology–Oriented Techniques

Although the computer is the primarily technological tool of CAPSI, other tools can be conceptualized as modules to be incorporated into CAPSI if they advance the educational goal of CAPSI.

2.3.1 Tutorials on Audio–Cassettes

The audio–cassette recorder is so popular that educational material can be delivered to and heard by any student. The audio tape can be heard and replayed at any time and place. Printed and other material may also be accompanied with the audio tape.

2.3.2 Tutorials with Discrete Visuals and Synchronized Audio

This technique was used commonly for industrial training prior to the invention of the video tape. While the discrete visuals are being viewed, a student can hear the sound–track synchronized with the scenes.

2.3.3 Tutorials on Film or Video Tape or Compact Disc

The 8 and 16–mm films have now been replaced by video tape. The video tape incorporates the advantages of audio–taping and video–based media. The reasons for using video tape are numerous such as the popularity of the equipment, the ease of equipment operation, the lack of processing before viewing, and the reusable nature of the tapes. A compact disc resembles a phonograph record and is used as an information storage medium. The information that is stored digitally on a disc can be taken from a slide, a video tape, a film, an audio tape or a printed text and be displayed on a television with the using of a compact disc player.

2.3.4 Audio and Video via Communication Satellite

Recent developments in satellite communication technology have provided educators with new opportunities for long–distance education. Communication satellites are most widely used in long–distance education because satellites have been in existence for over 25 years. Also, the recent advances in microelectronics and in launch vehicles have significantly reduced the costs of communication satellites.

2.4 Summary

CAPSI is a generalized teaching and learning method that can be applied in various educational environments with any additional technology–oriented techniques. CAPSI adopts a modular approach and does not rule out any device or procedure that is useful in the learning process. For example, audio tapes, video tapes, compact disc, lectures, discussions, laboratories, and term projects are used whenever desirable. In fact, the technological voice teleconferencing has been used with CAPSI to successfully deliver courses to distant communities. As a result, the effective usage of voice teleconferencing can demonstrate that the modular approach of CAPSI is practical and advisable. In the following chapter, a presentation of CAPSI course is described with some examples.

CHAPTER III

CAPSI: A GENERALIZED COMPUTER-AIDED TEACHING AND LEARNING METHOD

3.1 General Course Structure

Like PSI course, the instructor using CAPSI prepares the course materials in advance before the course starts. The text book and other additional materials are selected and prepared, and then divided into about as many units as there are weeks in the course. Next, the instructor writes study objectives for each unit which are typically in question form requiring written answers rather than multiple choice or true-false responses. Typically, each unit consists of 20 to 30 questions in short-essay or short-answer form. The instructor also writes a description to inform the students about the course procedures, the use of the computer, the grade point system, and other requirements in the course, such as a term project, a mid-term examination, and a final examination. Finally, the instructor assigns points for passing a unit test, for proctoring a test, for the other course requirements, and for each final letter grade.

During the course, the instructor has an impossible burden to mark all the written tests from the students and respond the immediate feedback to them. Therefore, proctors are used. Since it is usually costly to hire a number of proctors and often unavailable to request students from higher level courses to proctor, CAPSI adopts a new method and utilizes students in the course who have passed and mastered a given unit to proctor others

with units up to that unit. For the purpose of quality control and to increase the times of proctoring, two proctors are required to mark independently of a given test. Students within the course benefit from proctoring such as enhancement of knowledge, retention of the material, reinforcement of learning, and increment of credit point.

The course may be offered in regular class schedules running the CAPSI computer program in one or more classrooms. However, it is more efficient for students to use the computer program outside the regular class periods. When this is the case, the class periods may be removed and changed to other activities such as lectures, discussions, or team–work. Therefore, the use of CAPSI is suitable for correspondence courses, no scheduled class is needed, provided that the students can frequently access to the computer. The computer can be accessed by a terminal or microcomputer connected to the mainframe computer directly by telephone or through one of the Canadian data networks such as Datapac. Since Datapac can be accessed through data networks in many other other countries, the locations from which a course using CAPSI may be taken extend well beyond the boundaries of Canada.

3.2 The Computer Program

The CAPSI computer program was implemented in PL/I on the Amdahl mainframe computer. Each student is given a personal computer account on the mainframe computer which provides electronic mailing and messaging facilities. Each account contains a mailbox for sending and receiving electronic mail to and from other accounts. In addition, the

CAPSI program can be invoked from the accounts of the students enrolled in the course.

3.2.1 Student–Computer Interaction

Figure 3.1 shows how a student interacts with CAPSI [HeKP84a]. An identification of a registered student and an associated password are required for the student to interact with CAPSI. Then the student has the opportunity to view his/her course standing points earned through by passing and proctoring tests. If the student has not selected to proctor a test, he/she has opportunity to volunteer as a proctor. The volunteered student has the responsibility to check his/her account within 24 hours to see whether he/she has been selected to proctor a test.

If the student is not writing a test, the student can require the computer to generate and mail a test by randomly selecting three questions from the study questions on the student's current unit. After receiving a test at the mailbox file by the use of an ACCEPT command, the student types the answers directly into the file on the mainframe computer, or downloads the file to his/her microcomputer, logs off the mainframe computer, types the answers into the downloaded file on his/her microcomputer, again logs onto the mainframe computer, and uploads the answered file to the mainframe computer.

After the student completes the test, he/she starts the CAPSI and decides whether he/she wants the test cancelled or proctored. If the student feels the material is not well mastered, CAPSI allows him/her to cancel the

test and will not issue him/her another test on the same unit for at least one hour following the test cancellation. If the student wants to have his/her test proctored, either the instructor or two student proctors will be selected by CAPSI for marking according to the proctor selection algorithm, which will be discussed in the later chapter. The program then electronically mails the answered test to the two designated proctors or to the instructor if two eligible proctors are not available.

Potential proctors check their computer mailbox files for tests as soon as they log on to their computer accounts indicated by messages that new mails arrived from other student accounts. A proctor first marks the arrived answered test immediately with feedback, then starts the CAPSI program which mails the marked test with the result mark to the student who wrote the test and to the instructor for marking verification. The proctor is verified to mark a given test by entering the unit number and question numbers of the test. Then, the proctor enters one of the pass, conditional pass, or restudy result. A pass is entered if the student's test performance demonstrated clear mastery of the unit; a conditional pass is entered if the student made a minor error and corrected it after it was pointed out by the proctor; a restudy is entered if the student made minor errors on more than one question or any major errors. In order for the student to pass the unit, both proctors or the instructor must enter a pass or conditional pass. Otherwise, the student with restudy result must wait at least an hour before he/she can generate another test on the same unit. The instructor receives and maintains a copy of the answered test from the student who wrote the test, and copies of the marked tests from both the two proctors, teaching assistant, or the instructor.

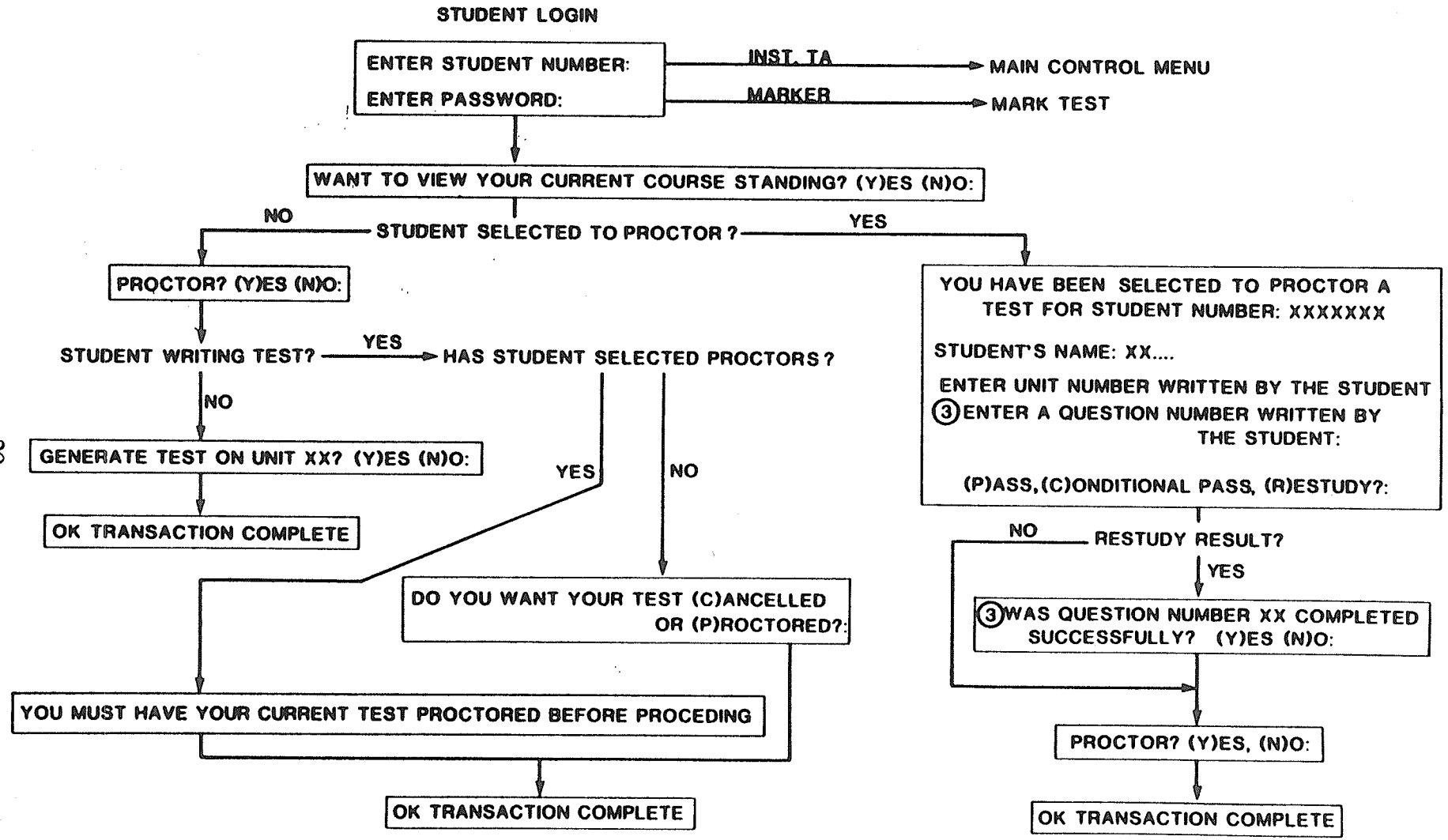

Fig 3.1. Student terminal prompt summary [From HeKP84a].

3.2.2 Main Control Menu

Figure 3.2 shows the main control menu [After HeKP84b] which is available only to the instructor for database administration, monitoring transactions, and statistical analysis. The options of the choices are as follows:

1. It enables the instructor to start a session of the regularly scheduled class, set the cutoff time for it, and decide whether it is a new or continued session.
2. It permits the instructor to terminate a session with the unmarked students listed.
3. It allows the instructor to create and delete student records in the class, and to modify the personal data and course data of a student.
4. It permits the instructor to modify the system parameters of the course such as the grade point system.
5. It enables the instructor to mark test and enter the result.
6. It provides the instructor to send messages to and communicate with the students.
7. It provides the instructor to centrally monitor all the student activities.
8. It disables the computer program.

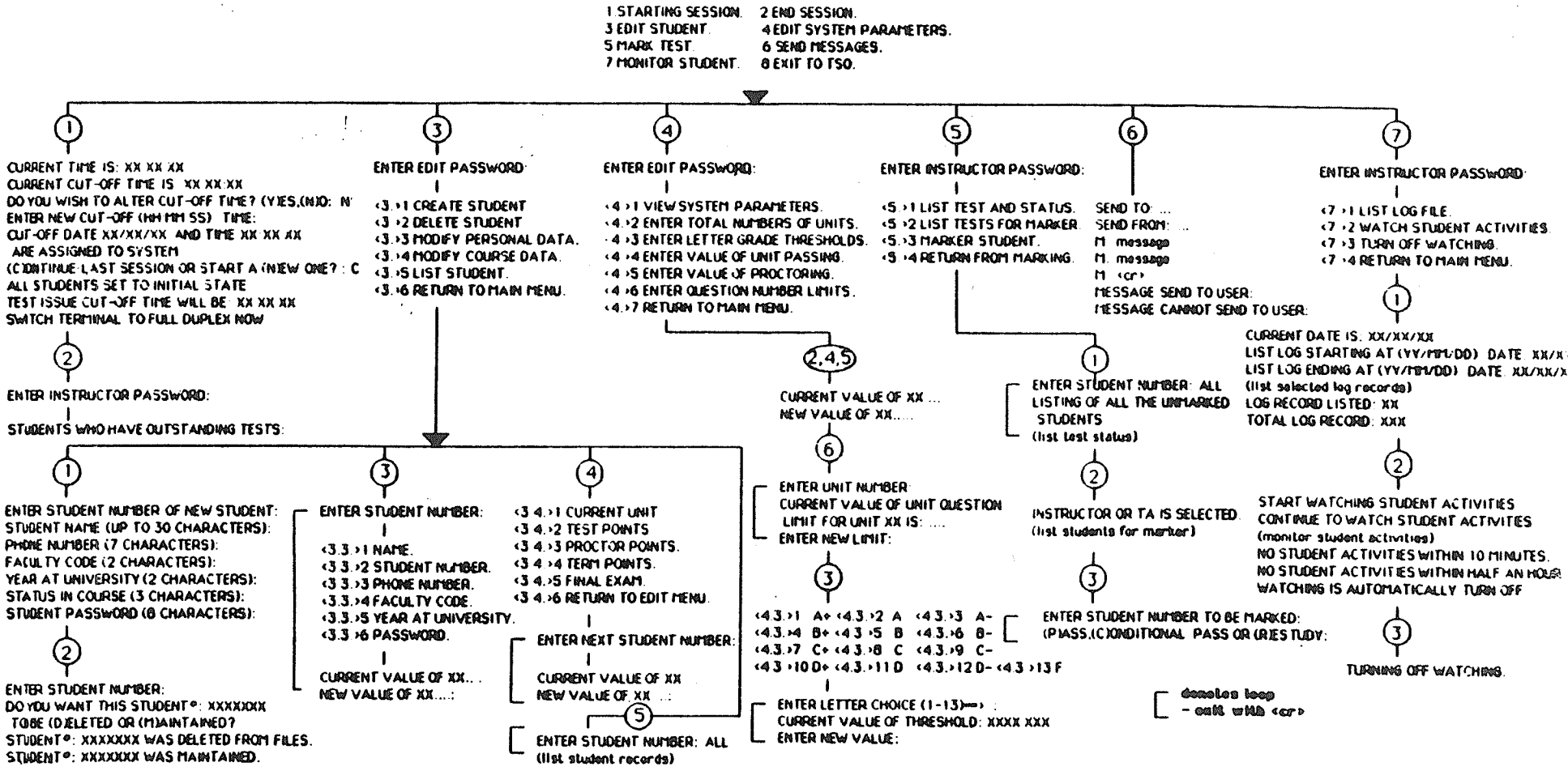

Fig 3.2. Main program menu summary [After HeKP84b].

3.3 Examples in a Course

3.3.1 Instructional Objectives, Frames, and Student Responses

The instructional objectives emphasize the understanding of the subject matter by definition and through its use. The following example shows the instructional objectives in a behavior modification course:

1. Student should be able to state an acceptable definition of positive reinforcer, and identify examples in everyday life.
2. Student should be able to identify the basic components of a behavior modification project, and describe the steps by which such a project would be carried out.

The instructional frames highlight the personal, aspect in the communications between the student and the instruction. The following example shows instructional frames, student responses, and instructor's comments:

1. Date: Sun, 20 Sep 87 20:35 CDT
2. To: PEAR
3. From: [Student.ID#GB]
4. Subject: [ID] 09/20 11:46 Unit 2 : 2 6 10 ANSWERED PASS
5. Result: PASS
6.
7. 17.244 BEHAVIOR MODIFICATION PRINCIPLES
8. Question 2:
9. Why would a behavior modifier say that we should
10. talk about retarded behavior, not retarded people;
11. or autistic behavior, not autistic children.
12. Answer 2:
13. The behavior modifier talks about retarded or autistic

behavior because he/she makes a judgment based on observable behavior and not on invisible mental abnormalities. They arrive at a decision that behavioral problems are exhibited when an individual is compared with others of approximately the same age, training, or background and behavioral deficits, excesses or inappropriatenesses are observable in the particular individual. In such case the behavior modifier attempts to change the particular behavior by using specific procedures which have been shown to be effective in establishing more desirable behaviors.

Question 6:
What is a behavioral excess. Give two examples.
Answer 6:
A behavioral excess occurs when an individual displays too much behavior of a certain type. For example, a behavioral excess occurs when a teenager spends hours talking on the telephone or a person watches television all evening.

Question 10:
What are some of the objectives of the textbook ?
Answer 10:
By using actual case studies to illustrate behavior modification principles, the authors attempt to provide satisfactory answers to concerned individuals (parents, teachers, etc.) about what can be done to change the behavior of individuals who are given various labels (retarded, etc.) due to their undesirable behaviors be it deficits, excesses, or inappropriatenesses. The authors also try to provide students with an understanding of why certain behavior modification procedures are effective and discuss ethical issues in their use.

comments: [INSTRUCTOR'S COMMENTS]
Your answers to questions 6 and 2 are good; the answer to question 10 could have been a bit better, but does cover what was said about the objectives of the book in Chapter 1. Therefore, this test is a pass.
However, you might note that a clearer statement of the objectives occurs on p.xviii of the Preface.

3.3.2 Mastery Test

An extensive example of mastery test which was evaluated by the instructor is as following:

1. Date: Fri, 25 Sep 87 14:25 CDT
2. To: PEAR
3. From: [Student.Name#JS]
4. Subject: [Name] 09/25 9:20 Unit 4 : 3 6 17 ANSWERED PASS
5. Result: PASS
6.
7. 17.243 HUMANISTIC AND TRANSPERSONAL PSYCHOLOGY
8. Question 3:
9. According to Carl Rogers, what is a "field of experience" ?
10. Briefly describe the two limitations on awareness
11. postulated by Rogers.
12. Answer 3:
13. A "field of experience" is the envelope surrounding
14. an organism in which the organism has the potential
15. to be aware of external stimuli. This field of
16. experience includes all stimuli that the organism
17. can potentially conscious of whether it is or not.
18. Limitations on awareness include psychological
19. limitations (or the willingness of the organism
20. to be aware), and biological limitations (or
21. the capability of the organism to be aware).
22.
23. Question 6:
24. According to Maslow, what is "plateau experience" ?
25. How is it similar to and how does it differ
26. from a peak experience ?
27. Answer 6:
28. A plateau experience is the simultaneous perception of
29. the sacred and the ordinary. Maslow
30. referred to "plateau experience" as a state of unified
31. consciousness. It is similar to peak experience in that the
32. miraculous feelings or states are experienced by the individual

33. but unlike the peak experience the individual
34. does not experience a feeling like an atomic blast from
35. the plateau experience. That is, a peak experience
36. is a much more intense experience but does not usually
37. last long (a few minutes to a few hours usually).
38.
39. Question 17:
40. Who developed the theory of operationism ?
41. On whose theory was the doctrine based?
42. What was the name of this theory ?
43. Answer 17:
44. The doctrine of operationism was developed by Bridgman.
45. His theory was based on Einstein's theory of relativity.
46.
47. comments: [INSTRUCTOR'S COMMENTS]
48. Your answers are very good. One minor point – on line
49. 24 you describe an atomic blast. I hope you meant an autonomic
50. burst. PASS ENTERED.

3.3.3 The Use of Electronic Mail

The following are the two examples of the use of electronic mail. The first example shows how a student requests help from the instructor about the current test as following:

1. Date: Wed, 04 Feb 87 20:58 CST
2. To: JPEAR
3. From: [Student.ID#NM]
4.
5. In chapter 18, I was unable to think of examples for 3
6 categories of question 15. The categories were:
7. 1. inappropriate environmental stimulus control.
8. 2. inappropriate self–generated stimulus control.
9. 3. problematic reenforcement contingencies.
10. Could you start me in the "right direction" please ?
11. Thanx, [Student.Name#NM].

1. To: [Student.ID#NM]
2. Subject: REPLY TO YOUR QUESTION
3.
4. [Student.Name#NM],
5. 1. An example of inappropriate environmental stimulus
6. control might be friends who exert a "bad" influence
7. on a child by, for example, encouraging him to
8. engage in trouble–making behavior.
9. 2. An example of inappropriate self–generated stimulus
10. control might be statements the individual makes to
11. him/herself which evoke undesirable behavior;
12. for example, saying to oneself "I have to smoke, drink,
13. and be rowdy or my friends won't like me."
14. 3. An example of problematic reinforcement
15. contingencies might be the contingencies provided
16. by the above individual's friends when they
17. reinforce him/her with their approval for smoking,
18. drinking, and being rowdy. I hope you find these
19. examples helpful in answering the question.

The second example illustrates how a proctor improves the clarity of a test question as following:

1. Date: Sun, 27 Sep 87 20:30 CDT
2. To: PEAR
3. From: [Student.ID#PA]
4.
5. Hi... I was proctoring a Unit 3 test and I found an
6. apparent ambiguity in question #13. It asks
7. "What two kinds of behavior does thinking
8. consist of from the perspective of radical behaviorism ?".
9. It seems that one answer could be "covert speaking and
10. covert listening" and another answer could be
11. "covert speaking and conditioned seeing";
12. it depends where in the text you look.
13. What should the answer be ?
14. [Student.Name#PA]
15. **********

16. Reply from PEAR:
17. [Student.Name#PA]
18. What I had in mind when I wrote the notes was
19. the latter (except that I would include both covert and
21. overt verbal behavior under the rubric of "thinking").
22. You have to recognize, however, that the concept of
23. conditioned seeing can be broadened to include
24. conditioned sensing; e.g., conditioned hearing,
25. conditioned feeling, conditioned smelling.
26. I'm not sure exactly what you mean by covert listening;
27. if you mean it in the sense of conditioned hearing,
28. then it could be considered a type of thinking.
29. Or you might mean it in the sense of "hearing"
30. what one is "saying covertly" to oneself. I would
31. accept any of these as part of the answer
32. (with covert or overt verbal behavior being the other part)
33. if the student makes a good case for it.

3.3.4 Marking Verification

The following example shows an illustration of such a marking verification requested by an instructor. Moreover, proctors may themselves request verification of their marking.

1. Date: Wed, 23 Sep 87 14:32 CDT
2. To: [Proctor.ID#BE, Proctor.ID#SI]
3. From: [Student.id#RN]
4. {Answers to questions of Test 1}
5. {Marked and passed by [ID#SI] on Wed, 23 Sep 87 18:33 CDT}

1. Date: Thu, 24 Sep 87 16:36 CDT
2. To: [Proctor.ID#SI]
3. From: PEAR
4.
5. How do you justify giving this test a pass ?
6. It's not even clear what the question numbers are,
7. let alone the answers.

1. Date: Thu, 24 Sep 87 21:15 CDT
2. To: PEAR
3. From: [Proctor.ID#SI]
4.
5. The question numbers are 2, 10, 7 and the answers
6. appear in that order. Although [Student.ID#RN] has
7. chosen to answer these questions in that order,
8. it is not a literate way to answer them. The test should be
9. revised into long answer form with question # headers.
10. **********
11. Reply from PEAR:
12. OK. Please inform the student of this.
13. Please try to give better feedback in the future; otherwise,
14. the student may do poorly on the midterm and final.
15. Thank you.

3.4 Summary

This chapter firstly describes how a CAPSI course is structured. The students are informed in advance exactly how mastery and requirements of the course will be assessed. Secondly, the general operation of the CAPSI program which is designed to facilitate any course is described. The way students interact with the program and the major controls of the program are briefly described. Finally, there are various examples of the electronic mails used in courses for the communications among students, proctors, teaching assistants, and instructor. As will be discussed in the next chapter on the design of CAPSI, the essential concepts, specification, file systems, and finite–state modelling of the CAPSI program are addressed.

CHAPTER IV

DESIGN OF CAPSI

4.1 Engineering Concepts in CAPSI

CAPSI employs the engineering concepts [KiPe88b] of product specification, design, utilization of technology, quality control, cost effectiveness, and systems analysis.

4.1.1 Product Specification

In order to apply engineering concepts to education, learning is considered as the intended product of education and is specified as clearly as any other product. CAPSI, like PSI, has a clear specification of learning objectives generally in terms of answering questions about or solving problems relating to the course material. Hence, CAPSI can ultimately achieve the specification of educational objectives in terms of student performance in acquiring the substantive knowledge, conceptual skill, or attitudes from the intended course.

4.1.2 Design

Firstly, CAPSI involves the design of the course objectives, such as choosing the course material and setting up the student questions. Secondly, CAPSI involves the design of ensuring the maximum number of students will come close to meet the course objectives, such as designing a

self–pacing system for students to have a greater efficiency to master the course material.

4.1.3 Utilization of Technology

Engineering is most characterized by the use of technology. The main difference between PSI and CAPSI is the utilization of technology in computer and its telecommunication capabilities. Essentially, CAPSI adopts a modular approach to facilitate the integration of various technological devices into CAPSI to maximize the learning of students.

4.1.4 Quality Control

Similar to any high–quality product, procedures are taken to insure the highest quality of the student's learning. Firstly, built–in redundancy of using two proctors is used to evaluate the mastery requirement of each student's test. Secondly, passed tests evaluated by proctors are sampled by instructor to ensure maintenance of the mastery criteria and feedback is provided by instructor to the proctors. Thirdly, supervision is ensured that students writing tests on terminals do not make unauthorized use of materials or receive unauthorized help. When unit tests are not supervised, students are trusted not to cheat and required to write one or more supervised examinations to insure quality of learning.

4.1.5 Cost Effectiveness

Cost effectiveness implies using the cheapest rate to gain the highest

quality or using the available resources to the fullest. Firstly, CAPSI fully utilizes the available students in the course to proctor other students. Thus, proctor benefits from proctoring in enhancing his/her mastery of the material and development his/her evaluative skills. Secondly, CAPSI fully utilizes the computer to facilitate the course efficiently at a low cost. In addition, further discussion about the cost of using CAPSI is in section 6.12.

4.1.6 Systems Analysis

With the ingredients of CAPSI precisely specified, data collected from CAPSI can be used for later analysis, the effects of various components of CAPSI on the student and system performances.

4.2 Problem Specification of CAPSI

CAPSI is programmed to be a computerized PSI with the following programming requirements:

1. CAPSI must keep a record on each student. Each student's record will contain personal data as follows:
   a) name,
   b) student number or student's computer account identification,
   c) phone number,
   d) faculty code,
   e) year at university, and
   f) status in course;

   and course data as follows:
   a) highest test unit the student has passed,

b) number of test points earned (i.e., sum of the value of all tests passed).

c) number of proctor points earned (i.e., the product of the number of times the student has proctored and the value of proctoring),

d) points earned on term project,

e) points earned on final examination,

f) total points earned, and

g) letter grade (based on transformation from total points to letter grades as defined by instructor).

2. CAPSI must keep a system parameter file which contains:

a) the number of units in a course,

b) the letter grade thresholds,

c) the point value of passing a unit test,

d) the point value of proctoring a test, and

e) a question number limit in every unit test.

3. CAPSI must permit the instructor to edit information in the student and system parameter files at any time. Editing may be done on a single record or sets of records on the specific field in the records.

4. CAPSI must regulate all test–assignment, proctor–assignment, and test–outcome interactions during each class session, or during no scheduled class session, virtual classroom. The programming requirements and functions of CAPSI with or without the option of class session are different and will be discussed in turn. Furthermore, CAPSI continuously updates the information in the student file on the basis of these interactions.

a) Test–assignment interaction.

i) The student may request a test from CAPSI in a class session

at any time. Provided that there are at least 20 minutes remaining in the class period and that the student has not received a restudy request within the previous 20 minutes, CAPSI will comply by printing the numbers of three study questions randomly chosen from the instructor–designated categories of next unit test in which the student is eligible to write on (i.e., the unit immediately after the highest unit passed). The length of the session can be adjusted by instructor.

ii) For virtual classroom, the student may request a test from CAPSI at any time of any day. If the student has not received a restudy within the previous one hour, the computer will compile and mail the text of the three study questions randomly chosen from next unit to the student by electronic mail. Since there is no supervision on the student, he/she must complete and return the test within one hour. Otherwise, the test will be automatically cancelled and the restudy result will be issued by CAPSI.

b) Proctor–assignment interactions.

When the student finishes writing the test, he/she requests proctors. A eligible proctor is a student who has indicated to CAPSI that he/she is willing to serve as proctor, who is not currently proctoring or writing a test, who is still in his/her restudy period, and who is qualified to proctor that unit by virtue of having passed a test on the unit. From the list of eligible proctors, CAPSI then select the two proctors with the lowest number of proctor points. If more than two qualified and

available students have the lowest number of proctor points, the proctors will be selected from them randomly.

i) If two qualified proctors are not available to proctor a given test, a teaching assistant or instructor will be selected to mark it, unless more than N tests are currently assigned to them within a class session. Therefore, the student required proctoring needs to wait and try again. For virtual classroom, there is no limitation of N tests for teaching assistant or instructor to mark tests.

ii) A student who is willing to serve as a proctor in class is expected to present in class. All proctors are automatically signed off at the end of the class period, and any test that has not been proctored by the end of the period is automatically assigned to the instructor to be marked outside of class. For virtual classroom, all proctors are expected to mark the tests within 24 hours; otherwise, the instructor is noticed to mark the outstanding test.

iii) If a student decides not to write a test after receiving it, he/she may cancel the test and it will not be marked. In this case the outcome is automatically treated as a restudy result.

c) Test–outcome interactions.

After the proctors marked a test, the two proctors (or teaching assistant or instructor) enter the unit number and question numbers, to verify that the test written was the one that was assigned, and the result of the test. It will be one of pass, conditional pass or restudy.

i) If the result is a pass, this is immediately entered in the

student's record. The student is then immediately qualified to proctor for that unit or any below it, and to write a test on the next unit.

ii) For classroom setting, the pass and conditional pass results are only functionally different for statistical purposes when recorded in the log file otherwise they are treated the same. For virtual classroom, a student has a conditional pass if he/she make a minor error and corrected it after it was pointed out by the proctor. The proctor waits for the student's correction and marks again for the result.

iii) If the result is a restudy, the student must wait for at least 20 minutes of restudying time with class session or one hour of restudying time without class session before attempting another test on that unit. No proctoring will be assigned to students within restudying period.

iv) If the student did not write the appropriate test, the result is treated as a restudy.

v) If the proctor cannot verify the unit number and question numbers of the test that was assigned for him/her to mark, he/she can either reenter the result later or issue a restudy result.

vi) An error correction feature should be available for test–outcome interactions in case the wrong information is entered accidentally.

5. During a given class period, a student may leave class before the end of the period provided that he/she turns off the option of wanting to be a proctor. A student of virtual classroom can also turn off the

proctoring option; otherwise, he/she has to check whether he/she is assigned to be a proctor within 24 hours. Students violating these requirements may be penalized by the loss of proctor points and/or not being permitted to proctor for a certain period.

6. Students may have their course data printed out at any time for their own information.

7. The following security precautions are necessary:

   a) The instructor has a password to edit files, to start and terminate the program during a class session, and to enter the results of tests.

   b) Teaching assistant has a password to enter the results of tests.

   c) Each student has a password to interact with CAPSI.

   d) The ability for the instructor, the teaching assistant, and the students to change their passwords at any time.

8. Communication can be established by sending messages between students, teaching assistant, and instructor. A message can be sent to multiple destinations at the same time.

9. For research purposes, CAPSI must permanently store the date, time, and specific details of every interaction with it. Easy access to these data must be available for examination and analysis in any desired manner.

10. CAPSI can be used by multiple users at the same time to access on a shared database concurrently. In other words, mutual exclusion is needed to be enforced when two students or more use CAPSI to access the shared files. Therefore, concurrent programming techniques and file system with exclusive capabilities are used to develop CAPSI.

11. For virtual classroom, students, teaching assistants, and instructor can access the CAPSI program from their computer accounts. CAPSI can handle all the interactions between students and students, students and teaching assistants, and students and instructor by delivering and receiving electronic mails of the students' tests. Students, if request, will receive fully described tests through electronic mailbox from CAPSI.

12. CAPSI has the capability of monitoring the student activities continuously. The instructor is noticed by descriptive messages from CAPSI whenever student transactions occur. These descriptive messages of transactions can accessed by the instructor on–line.

13. A number of utility programs are used to create, maintain, and analyze the files. A format utility is used to allocate and initialize the database for a course. Some maintenance utilities are used to list the student records and security passwords, to enlarge the database when it is full, and to archive the database on tape for backup purposes. Some analytical utilities are used to analyze the research data recorded by CAPSI.

CAPSI was originally implemented for Professor Joseph Pear of the Psychology Department, the University of Manitoba to be used in his course: Behavior Modification Principles. The development of the CAPSI system required considerable thought and effort since 1983.

CAPSI was designed to be simple to learn and easy to use especially for those students who are the first time to use computer system or computer–aided instruction. The instructions and messages from CAPSI to

students are descriptive and self–contained. CAPSI has to be efficiency and low cost to run because high frequent calls are invoked by students. Finally, high structured and well maintainable coding are required by CAPSI.

4.3 Database Design

A database of a course is divided into four separated files which are student record file, system parameter file, question bank file, and transaction log file. A file is a collection of individual records which contain interrelated fields of information.

4.3.1 The Student Record File Design

The student record file contains all records of students' personal and course information. The key of the file is indexed by the student numbers or student computer account identifications. A particular record can be accessed immediately by providing the key of the student identification using the method of Indexed Direct Access Method; or all the records can be accessed by the ascending order of the student identifications using the method of Indexed Sequential Access Method. The methods of direct access and sequential access through the indexed keys are necessary for CAPSI to have the following operations on a particular record or the whole records of the students in a course:

1. Read operation allows CAPSI to retrieve data in a record,
2. Write operation allows CAPSI to create a new record,
3. Update operation allows CAPSI to modify data in a record, and
4. Delete operation allows CAPSI to eliminate a record from the file.

Since CAPSI provides multiple users to access the file concurrently, the above operations must have the exclusive capability to lock the accessing record by preventing interference from another operation. Any operation refers to a locked record will wait at that point until the record is unlocked and then proceed. Multiple read operations on a record are allowed provided that the operations read, without alter, the data.

There are three additional records stored in the student file with the special student identifications of INST, EDIT, and TA. These records stored the passwords for the instructor, editing privilege, and teaching assistant respectively. All the student records, except the three reserved records, are not only contain the personal and course data, but also contain the passwords to log onto CAPSI. In addition, the student records contain their current test and proctor states for the use of the finite–state machine which is the heart of CAPSI.

In order to keep track of a student's data, the conceptual structure of the data dictionary looks like these:

1. Identification of the student by his/her student number or student computer account identification: seven characters long.
2. Name of the student: thirty characters long.
3. Phone number of the student: seven characters long.
4. Faculty code of the student: two characters long.
5. Year of the student at the university: two characters long.
6. Status of the student in the course: three characters long.
7. Unit test that the student is currently reached: a numerical integer.
8. Question number one of the unit test: a numerical integer.

9. Question number two of the unit test: a numerical integer.
10. Question number three of the unit test: a numerical integer.
11. Test points of the student accumulated: a numerical decimal number.
12. Proctor points of the student accumulated: a numerical decimal number.
13. Term project points of the student achieved: a numerical decimal number.
14. Examination points of the student achieved: a numerical decimal number.
15. Total points of the student accumulated: a numerical decimal number.
16. Letter grade assigned according to the total points: two characters long.
17. Password associated with the student: eight characters long.
18. Proctor state of the student: a numerical integer.
19. Test state of the student: a numerical integer.
20. Another student identification whom is being proctored by the student of this record data: seven characters long.
21. Restudy time stamp of the student needed to fulfill when he/she is at the restudy test state, or the test time stamp when he/she generates a test: a time stamp of six characters long.

4.3.2 The System Parameter File Design

The system parameter contains all the information for a course. It provides all students of the grading system and the test schedule of the course. Moreover, CAPSI uses this file as internal storage which is

transparent to the students, teaching assistants, and instructor. The course information can be concurrently retrieved by multiple students, but only be modified by the instructor. There is only one record in this file; and the record can be accessed by Direct Access Method.

The conceptual structure of the system parameter record for the course is as follows:

1. Number of units in a course: a numerical integer.
2. An array of question number limits of all units: an array of numerical integers.
3. Points credited for passing a unit: a numerical decimal number.
4. Points credited for proctoring a test: a numerical decimal number.
5. An array of letter grade thresholds: an array of numerical decimal numbers.

Other data which are defined to be used internally by CAPSI are as follows:

1. Date stamp of the last session: a date stamp of six characters long.
2. Time stamp of the last session: a time stamp of six characters long.
3. Cutoff time stamp of the last session: a time stamp of six characters long.
4. Current number of tests assigned to the teaching assistants and instructor: a numerical integer.
5. Pointer to the teaching assistant who did the last marking: a numerical integer.
6. Number of log records in the log file: a numerical integer

4.3.3 The Question Bank File Design

The question bank file contains all the unit tests with typed descriptive questions. CAPSI randomly retrieves three questions of a unit from this file when a student is allowed to generate a test. The student will receive the test, with three questions copied from the question bank, through electronic mail. When the student completed and hands in the test through CAPSI, CAPSI will verify the same wording of the questions from the student with the same questions from the question bank.

The question bank file is organized in a hierarchical tree structure with all questions of a unit as leaf nodes, the unit as the parent of nodes, and the course of all units as the root of the tree. This organization provides quick direct access of questions by students and easy operation and maintenance by the instructor. Students have no right to modify the questions of any unit but to read the questions only. Figure 4.1 shows the hierarchical tree structure of the question bank file.

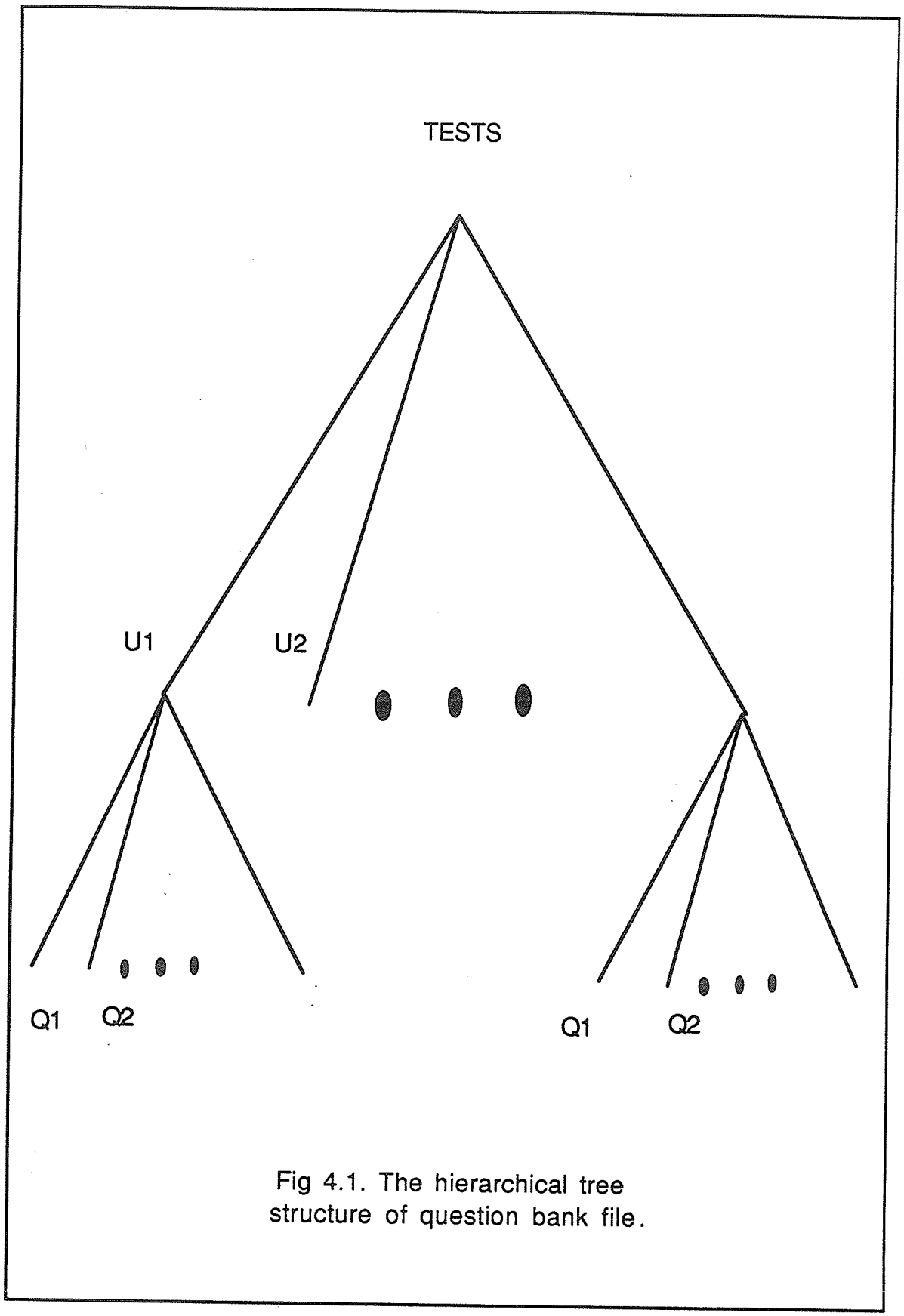

Fig 4.1. The hierarchical tree structure of question bank file.

4.3.4 The Transaction Log File Design

The log file is the most important product of CAPSI to do research work in the behavioral engineering. CAPSI chronologically records all student transactions by appending the transactions at the end of the log file. Furthermore, the file can be concurrently acquiring new research information by as many students using CAPSI at the same time.

The conceptual structure of the log record is as follows:

1. Type of the transaction record: one character long.
2. Identification of student being logged: seven characters long.
3. Test state at the time of transaction: a numerical integer.
4. Proctor state for the transaction: a numerical integer.
5. Highest unit test reached for the student at the time of the transaction: a numerical integer.
6. Question number one of the current test; if this number is negative, it indicates the proctor thought the question was not answered correctly: a numerical integer.
7. Question number two of the current test: a numerical integer.
8. Question number three of the current test: a numerical integer.
9. Test points of the student at the time of the transaction: a numerical decimal number.
10. Proctor points of the student at the time of the transaction: a numerical decimal number.
11. Time stamp of the transaction logged: time stamp of six characters long.

12. Another student identification, whom is being proctored by or who is proctoring the student of this transaction; or date stamp of the transaction only for some special types of transactions: seven characters long.

4.3.5 Type of Transaction Design

The type field in the log record determines the nature of the transaction being logged. There are two basic types of transactions. Firstly, transactions are caused by CAPSI interacting with students, teaching assistants, and instructor. Secondly, transactions are written by CAPSI itself when the log file is created, CAPSI is called and terminated, and a session is started and terminated.

The types of transactions are designed as follows:

1. Transaction log file is created.
2. CAPSI is called.
3. CAPSI is terminated.
4. New session is started.
5. A session is terminated.
6. Test is generated.
7. Test is cancelled.
8. Test is wanted to be marked.
9. Proctor is selected to mark a test.
10. Pass result is entered.
11. Conditional pass result is entered.
12. Restudy result is entered.

4.4 Finite–State Machine Design

The success of CAPSI can be attributed to the finite–state modelling of all the transactions that take place during the course offered. The process of course transactions can be described by two state transition diagrams, as shown in Figures 4.2 and 4.3, representing test state and proctor state of a student. The test states along with the proctor states determines the CAPSI behavior once a student has logged on.

4.4.1 Test–State Transition Design

In the test state diagram, there are six distinguishable internal configurations of CAPSI to identify the student's test statuses represented by the circles. The directed arcs representing the transitions between the states are triggered by inputs such as the action keywords in CAPSI.

The test state transition diagram describes and summarizes the process of a student to have his/her test from being generated to being proctored. When a session started, a student is at an initial test state of not writing any test. Since the student is not writing a test, he/she has an option to proctor other students. State transition to the state of writing a test if the student generates a test from CAPSI. The student will be at the restudy state, of not being allowed to issue a retest until the restudy period elapsed, if he/she voluntarily cancels the test or is automatically cancelled the test by CAPSI for not returning the answered test within an hour. The state of no mark yet is transited to when the student wants the test to be

marked and has proctors assigned but has no result replied. While the student waits for the result, he/she has the option to proctor other students. If only one pass result is given by a teaching assistant or instructor, the student passes the test and will be at the final state, same as the initial state, of not writing any test. On the other hand, the student does not pass the test marked by a teaching assistant or instructor and needs to restudy and rewrite the test again. If the test is marked by the first proctor, the student will enter the state of having either one pass or one fail. While the student is waiting for the other result from the second proctor, he/she has the option to do proctoring for others. In case of the second proctor does not mark the student in some reasons, the instructor can issue a pass or a fail to the student for his/her final test result. Otherwise, the student needs both passes from the first and the second proctors in order to finally pass the test. The student will be at the restudy state if only one of the proctors fails the test of the student. In this case, the student is required to restudy his/her material within the restudy period and then allowed to write the same unit test. No proctoring is allowed by the student when he/she is restudying the material or writing a test.

4.4.2 Proctor–State Transition Design

In the proctor state diagram, there are three states with the initial state same as the final state of not proctoring and not available for proctoring. A student has an option to choose whether he/she is willing or not willing for proctoring. The student may be selected to proctor if he/she is willing and will be at the state of proctoring. Once the student finishes proctoring, he/she will choose his/her availability for further proctoring.

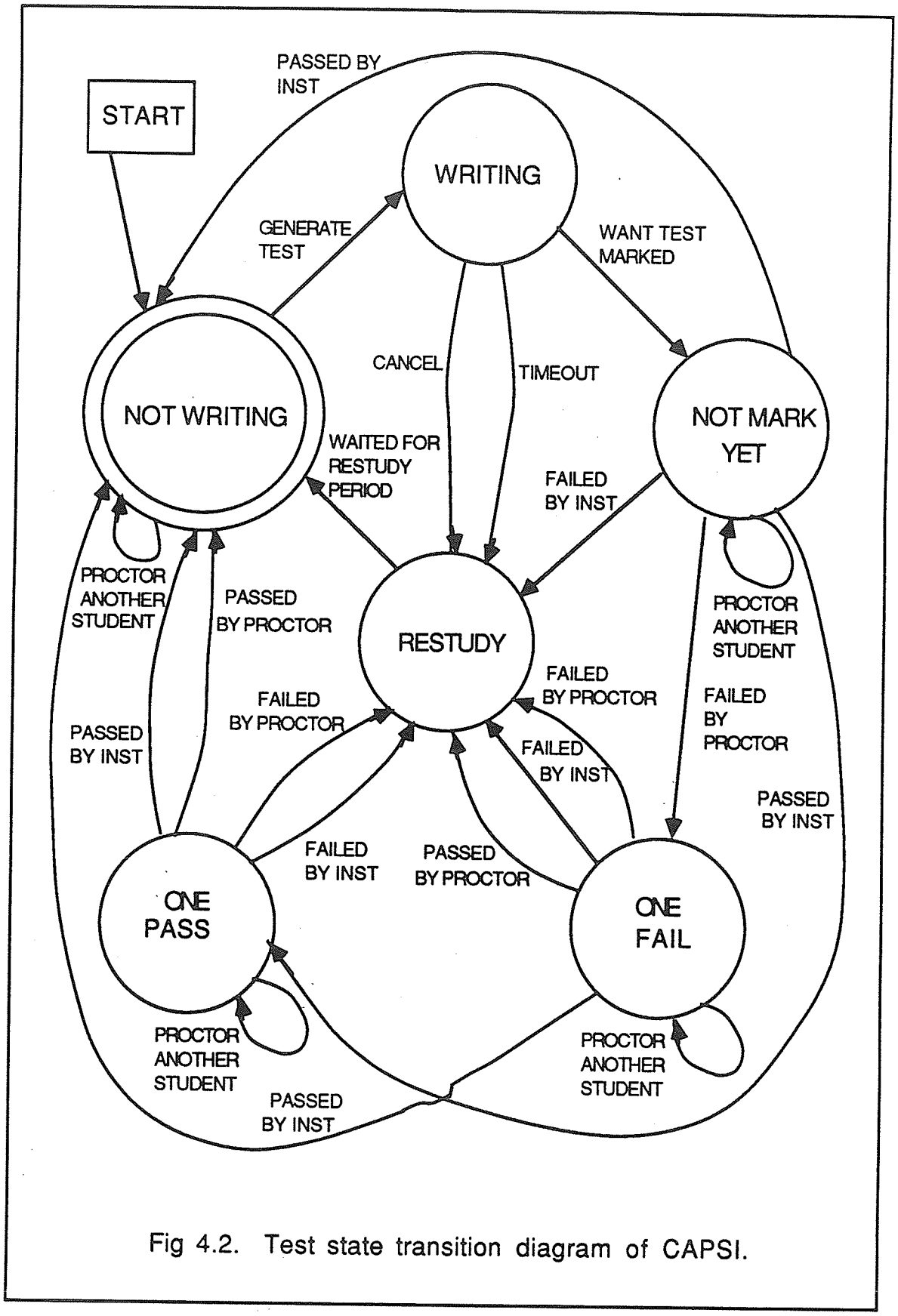

Fig 4.2. Test state transition diagram of CAPSI.

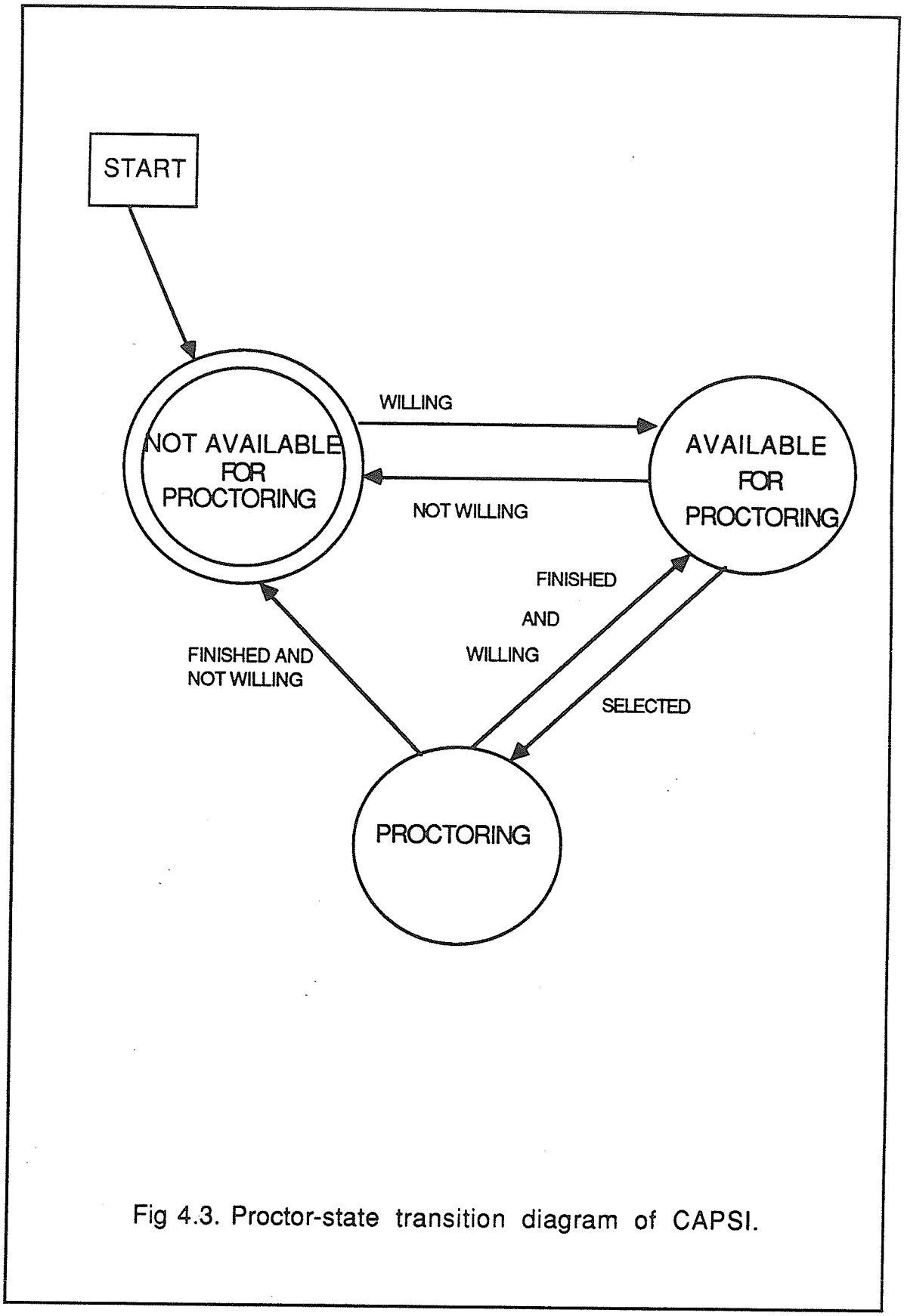

Fig 4.3. Proctor-state transition diagram of CAPSI.

4.4.3 Formal Definition for the Test–State Transition

The above two transition diagrams can be mathematically described by the definitions of finite–state machine. A finite–state machine is formally defined by a 5–tuple [HoUl79]

$$M = (Q, \Sigma, \delta, q_0, F),$$

where Q is a finite set of states,

Σ is a finite input alphabet,

q_0 in Q is the initial state,

F is a subset of Q and is the set of final states, and

δ is the transition function mapping $Q \times \Sigma$ to Q (i.e., $\delta (q, a)$ is the next state for each state q and input symbol a.)

Let M_T be the formal definition of the test state finite–state machine. First of all, different test states are assigned with state symbol q as follows:

q_0 is the initial state of not writing a test,

q_1 is the state of writing a test,

q_2 is the state of getting no result from any proctor,

q_3 is the state of getting one pass result from a proctor,

q_4 is the state of getting one fail result from a proctor,

q_5 is the state of getting the restudy result.

Secondly, different causes of each state transition are assigned with a finite input letter alphabets as follows:

a is the input symbol of generating a test,

b is the input symbol of wanting a test to be marked,

c is the input symbol of cancelling a test,

d is the input symbol of test being passed by a proctor,

e is the input symbol of test being failed by a proctor,

f is the input symbol of the restudy period waited,

g is the input symbol of proctoring another student,

h is the input symbol of test being passed by instructor,

i is the input symbol of test being passed by instructor.

From above, Q, Σ, and F are known as follows:

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$, the set of states,

$\Sigma = \{a, b, c, d, e, f, g, h, i\}$, the set of input; and

$F = \{q_0\}$, the set of final states.

Therefore, M_T is formally defined as

$$M_T = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{a, b, c, d, e, f, g, h, i\}, \delta, q_0, \{q_0\}),$$

where

$\delta(q_0, a) = q_1$, $\delta(q_0, g) = q_0$,

$\delta(q_1, b) = q_2$, $\delta(q_1, c) = q_5$, $\delta(q_1, e) = q_5$,

$\delta(q_2, d) = q_3$, $\delta(q_2, e) = q_4$, $\delta(q_2, h) = q_0$,

$\delta(q_2, i) = q_5$, $\delta(q_2, g) = q_2$,

$\delta(q_3, d) = q_0$, $\delta(q_3, e) = q_5$, $\delta(q_3, h) = q_0$,

$\delta(q_3, i) = q_5$, $\delta(q_3, g) = q_3$,

$\delta(q_4, d) = q_5$, $\delta(q_4, e) = q_5$, $\delta(q_4, h) = q_0$,

$\delta(q_4, i) = q_5$, $\delta(q_4, g) = q_4$,

$\delta(q_5, f) = q_0$.

The transition diagram of this finite–state machine is shown in Figure 4.4.

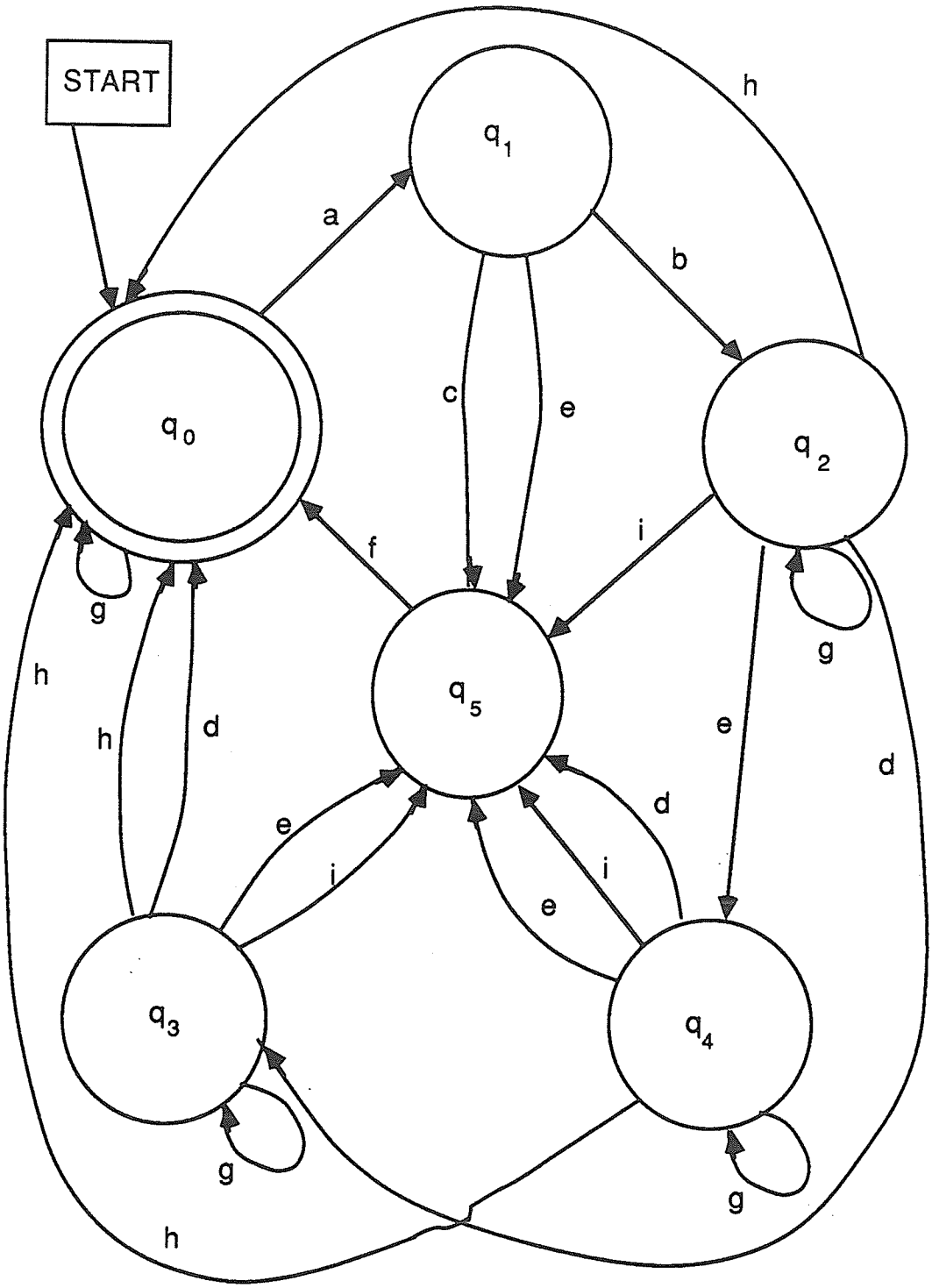

Fig 4.4. Finite–state machine of test–state transition diagram.

4.4.4 Formal Definition for the Proctor–State Transition

Let M_P be the formal definition of the proctor state finite–state machine. The proctor states are assigned as follows:

q_0 is the state of proctoring not available,

q_1 is the state of proctoring available,

q_2 is the state of proctoring another student's test.

The input alphabets are assigned as follows:

a is willing to proctor,

b is not willing to proctor,

c is selected to proctor,

d is finished proctoring and willing to proctor,

e is finished proctoring and not willing to proctor.

The set of final states is $F = \{ q_0 \}$.

Therefore, M_P is formally defined as

$$M_P = (\{ q_0, q_1, q_2, q_3 \}, \{ a, b, c, d, e \}, \delta, q_0, \{ q_0 \}),$$

where

$$\delta (q_0, a) = q_1 ,$$

$$\delta (q_1, b) = q_0 , \quad \delta (q_1, c) = q_2 ,$$

$$\delta (q_2, d) = q_1 , \quad \delta (q_2, e) = q_0 .$$

The transition diagram of this finite–state machine is shown in Figure 4.5.

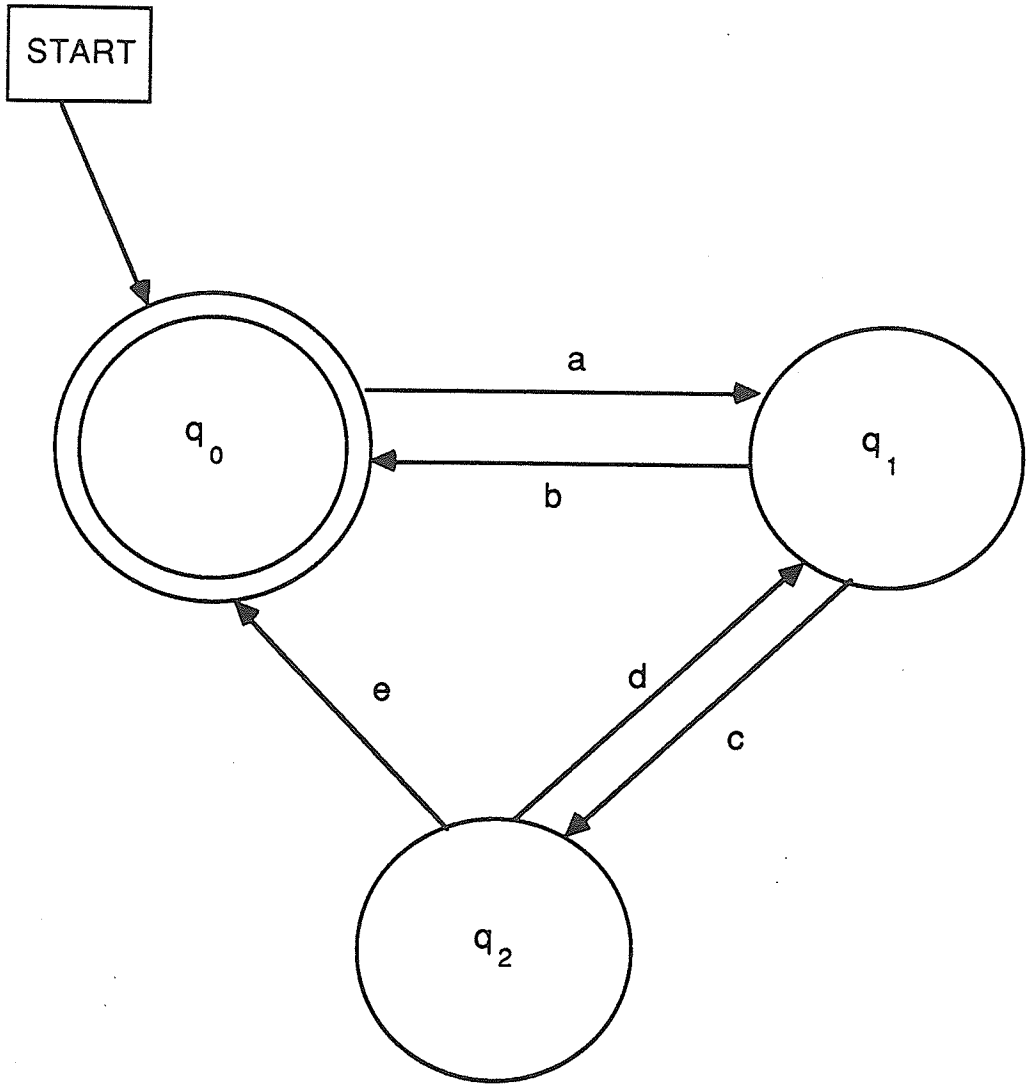

Fig 4.5. Finite-state machine
of proctor-state transition diagram.

4.5 Summary

Firstly, this chapter presents the engineering concepts embodied in CAPSI. Starting with learning as the product of education, CAPSI attempts to enhance the educational process through product specification, design, utilization of technology, quality control, cost effectiveness, and systems analysis. Secondly, the problem specification of CAPSI described in this chapter summarizes the major designs and programming requirements of the CAPSI program. The program is designed to be simple to learn and easy to use by students. Thirdly, the design of the major files in CAPSI database is characterized by its function and data dictionary. The types of transaction record determine the nature of the transaction being logged and indicate the nature of the student activities in course. Finally, the finite–state modelling of all the transactions taken place in course is described by the state diagram and by the mathematical definition. The finite–state modelling is the most essential part in the CAPSI program and contributes the success to the program. In the next chapter, the actual implementation of CAPSI in hardware and software is presented.

CHAPTER V

CAPSI IMPLEMENTATION

5.1 Hardware System

CAPSI is running on the mainframe computer operated by the University of Manitoba Computer Services Department. The mainframe is an Amdahl 580/5870 dual processor CPU with 48 megabytes of main memory and 26.6 gigabytes of disk storage. The 5870 partitions itself into two logical CPU domains providing multiple operating environments of MVS370 (Multiple Virtual Storage) running IBM's MVS SP 1.3.3 with JES SP 1.3.0 (Job Entry System) and of VM370 (Virtual Machine) running IBM's VM/SP 5.09 for UTS 580 1.2.1 (Amdahl's UNIX) and CMS (Conversational Monitor System). The MVS370 provides the environment for the TSO (Timesharing Option), IMS (Information Management System), Library on-line systems, batch jobs, all printing and almost all communications with remote computers.

CAPSI has been implemented in IBM's Optimizing PL/I (Programming Language I) Release 4.0 programming language on the TSO operating system which supports up to 450 concurrent users primarily on ASCII terminals. CAPSI has been developed on a local editing system called MANTES (MANitoba Text Editing System) which provides an efficient file structure, editor, query facility, mail facility and also interfaces to the batch system for job submission and retrieval. All the features of TSO are available either directly or indirectly through MANTES. TSO users can

interactively access other computer sites on the DATAPAC, TELENET and TYMNET networks and can also send and receive messages and mail to many other university and research institutions all over the world via the NETNORTH (BITNET) system.

CAPSI can be accessed through TSO terminals on campus or by the telephone dial–up modems operation at 300, 1200, or 2400 baud via the UMnet, the campus wide data network. On campus, students and staffs can access CAPSI through 900 ASCII terminals and 250 microcomputers supplied and maintained by the Computer Services.

5.2 Software Implementation

5.2.1 Overview of the CAPSI Software Evolution

CAPSI was originally coded in 1800 PL/I executable statements by Frank Herzog for the computer management of PSI course in 1983. At that time, the program could only be called from one terminal at a classroom to allow a number of students lining up to access their accounts. This program performed the basic functions such as managing the student records and the system parameter record, controlling the student transactions, and logging down the student activities. This program was tested and amended extensively for two years.

Since 1985, the author of this thesis has contributed many major improvements to the code as well as the research in CAPSI. Due to limited access to CAPSI, it was under major reconstruction of the logic of the

program and the structure of the database in order to allow multiple users accessing the shared database concurrently. The concurrent programming techniques were integrated into the logic of the program. The PL/I locking and unlocking features were enforced on the retrieval and modification of the shared records within the database.

Furthermore, the facility of monitoring student activities was added. The instructor had the capability of continuously monitoring the student activities from his terminal even though the instructor might be hundred of kilometers away from students in the case of distant education. The instructor could monitor the student activities and work on the CAPSI simultaneously such as doing some editing or marking. This facility was made possible by the multi–tasking feature of PL/I. Actually, a background task was invoked and was executing concurrently with the main CAPSI program to examine the log file periodically. The priority and synchronization of tasks were also integrated into the program logic.

Another facility of sending messages between CAPSI users was added. This facility allowed the students in distant education to communicate with the instructor by sending messages instead of by using the telephone. This facility was implemented by the interfacing of the high level language PL/I with the low level language OS/360 Assembler Level G.

The changes from single user to multiple users and the new facilities were made at the end of 1985. Hence, the coding of CAPSI was increased to 2800 PL/I and 100 assembler executable statements. The version of CAPSI was tested extensively and conclusively for two years by actually using it to

offer PSI courses.

With the evolution of electronic mail, CAPSI was further extended to be called from the students' TSO accounts at the middle of 1987. A smaller version of CAPSI was introduced for students to access only the control of student session. Since the students could access the CAPSI by any time and any place, the constraints of spatial and temporal limitations of traditional PSI course were released. At that time, the students typed their answers on the files and electronically mailed them to the designated proctors, teaching assistants, or instructor. The test questions of all units in written form were prescribed to all students at the beginning of the course.

At the end of 1987, the mailing facility was fully integrated into CAPSI so that the students were no longer needed to mail out the test paper by themselves. The transparency of the test delivery was handled by CAPSI through submitting a background batch job interfacing with MANTES for the electronic postal services. The batch job was made use of the system dynamic allocation routine, SVC 99, interfacing with the high level language PL/I and the system reserved file INTRDR for job submission.

Moreover, this version of CAPSI allowed the instructor to have a question bank to store all the test questions of all units of a course into a MANTES group file. The students could read the questions on–line. When a test was requested by and generated for a student, the typed questions were included on the electronic test paper. Whenever a test was generated, the selected questions were retrieved from the MANTES group using the NAM Access Method (NAMAM) supported by the Computer Services. This

version of CAPSI was tested and used heavily throughout a year of courses offered.

5.2.2 Program Structure of CAPSI with Classroom Setting

There are two main versions of CAPSI: with or without the classroom setting. The original CAPSI with the main control menu is designed to be used in a classroom. CAPSI is further extended to even no classroom setting. First of all, the version with the classroom setting is described through the structured charts.

The organization of the program using structured charts [Page80] are illustrated in Figures 5.1.a to 5.1.g. The charts of the program are divided into groups of activities corresponding to the choices of the main control menu. The functions of these activities are performed by the following modules.

5.2.2.1 Initialization and Terminal Control

As shown in Figure 5.1.a, the initialization module first initializes all the variables and opens the student file, system parameter file, and transaction log file, then logs down the user identification with date and time, prints out the welcome messages, and calls the terminal–control module. Following the initialization, the terminal–control module is responsible for displaying the main control menu and performing a subroutine call according to the choices entered by the teaching assistants or the instructor. The first two choices are for starting and terminating a

session.

A student session can be continued or started by the instructor. The module of the start session enquires the instructor for the choices of continuing the old session or starting a new session, and the cutoff time of the session. If a new session is required, all students will be initialized to their initial states of not writing or proctoring. Then, the student transaction processing is handled by the student session control module.

The session is ended by providing the security password protection from the instructor. The end–session module lists out the unmarked tests, logs off the proctors by initializing their proctor states, logs down the end–session date, and records the information that this session is ended. Finally, exiting to the operating system is provided by the last choice of the main control menu.

5.2.2.2 Student Session Control

The session–control module is shown in Fig. 5.1.b. This module loops and prompts for valid student identification and associated password. Once the valid student identification is typed in, the corresponding student record is retrieved in order to verify the password field in the record with the logging on password. The instructor or teaching assistants can sign on with the reserved identification INST or TA to terminate the control of session and return to the main control menu. If the reserved identification MAKER signs on instead, the control of session goes to the module of marking test by teaching assistants or instructor. It is a convenient and easy

way to access the marking–test module instead of getting out of the session and calling the marking–test module through the main control menu.

After a student signs on, the session–control module looks at the test and proctor states of the student to determine which prompts to be printed and what actions to be performed. The action table is used to determine which one of the three actions to be performed. Firstly, the student has no unmarked test and no restudy restriction; he/she should be prompted to generate a test. When the student requires a test, three question numbers from the unit that he/she is writing are randomly selected by method of pseudo–random technique which is a combination of linear congruential and shift register techniques. The question numbers are displayed on the terminal.

Secondly, the student has generated a test but has not selected proctors; he/she should be prompted to cancel the test or have the test proctored. If the student cancels the test, the proctor state of the student will be set to restudy state. When the student requires the test to be proctored, a teaching assistant or instructor or two student proctors are selected according to the proctor selection algorithm mentioned before. The names and the identifications of the proctors will be displayed on the terminal.

Thirdly, the student has been selected to proctor the test of another student; he/she is prompted to enter the unit number and the question numbers of the test for verification and then to enter the result of the marking. The marking student will be notified if the test he/she supposed to mark has already been marked by the instructor.

A transaction log record is built from the type of transaction and the information of the student record which is conditionally updated with a processing (transaction) state indicator. Based on the student record and the type of transaction, the transaction is executed according to a state machine. During the execution, the student record is updated and the transaction log record is retained in the log file for future research and analysis.

5.2.2.3 Student Edit Control

Figure 5.1.c shows the student edit–control module. Secure editing control of students' personal and course information is important in the academic environment especially for the students' course results. Editing password is required for the security protection. The module allows the instructor to create, delete, and list student records.

No duplication of student records with the same student identification is allowed in the process of creating new students. A new student record is indexed into the student file in the ascending order by the student identifications. All the fields of the new student record are entered by the instructor interactively.

A student record can be permanently deleted from the student file. Fortunately, a verification is provided for safety in case of deleting the record by mistake.

A student record or all the student records can be listed. If a valid

student identification is entered, the information of this student is listed. All the students' information are listed when the reserved identification ALL is entered. When a student hits the attention button on the terminal while the information of student records is listing, a choice to continue or quit the listing is provided.

Moreover, the student edit–control module performs the functions of editing the student personal information as well as the course information. Firstly, a student identification is entered for changing the personal information. After then, a specific field can be selected and altered. The changing of the personal passwords is much the same is the replacement of other fields. Except, if it is the password of teaching assistant or instructor or the password for editing, a validation of instructor password is performed before any of such passwords to be altered.

Secondly, the function of editing the student course information is provided since the adjustment of student marks is likely to occur during the course. After selecting a course field to be altered, all students of this course field can be altered and stepped through one by one according to the student identification provided. The total points will be automatically recalculated and the letter grade will be readjusted when the course information of proctor points, test points, term points, or examination points is altered. When the course field of the highest unit reached is altered, the internal test and proctor states of the student will be initialized.

5.2.2.4 System Parameter Edit Control

Figure 5.1.d shows the system parameter edit–control module. To preserve the system integrity, the system parameters are accessible either to the instructor or designated teaching assistants only. Edit password protection is enforced for editing of the system parameters. This module allows the instructor to change the thresholds of the letter grading system and to set up the points for passing a unit and proctoring a test. In case of the threshold of letter grade being altered, the letter grades of all the students are recalculated. Whenever a student's mark is increased, his/her letter grade is recalculated as well. The total points of the student is compared to the system list of threshold values starting at the letter F and working up the letter A+. The last threshold in the system list which is less than or equal to the total points determines the new letter grade of the student. If the points for passing a unit and proctoring a test are altered after the course has been started for some time, the previously accumulated unit and proctor points will not be recalculated. However all tests passed and proctored subsequent to the changes are worth the new values.

5.2.2.5 Mark Student Control

Figure 5.1.e shows the mark–student control module. Illegal marking is prevented by security password protection. This module can also be invoked from the student session–control module whenever the teaching assistants or instructor signs on using the reserved identification MAKER.

This module performs three functions. Firstly, it can list out the test status of any student, or list out all the students that are writing a test or waiting for their tests to be marked. Secondly, it lists out the students who have no student proctors assigned but have the instructor or the teaching assistants as their markers. The students who are still writing a test will also be listed. The first two functions of the module facilitate the instructor and teaching assistants to handle the whole situation of the session as well as to provide a list of their job duties.

Thirdly, the module allows the instructor or teaching assistants to have a full control of the marking of any students. The result of pass, conditional pass, or restudy can be graded by the instructor for the student who is waiting for the instructor or teaching assistant as the marker, or even writing a test, or waiting for the first or the second proctor to reply the result. The instructor or teaching assistants have the full privilege to override any result previously entered by student proctors, although the proctors still retain their proctor points. Once the result is entered, the internal states and course information of the student are updated and a permanent transaction record is logged.

5.2.2.6 Send Messages Control

Figure 5.1.f shows the send–messages control module. This module provides message sending between TSO users. Several user identifications with no embed blank or comma between the user identifications can be entered for the designations of the message sending. More than one line of the message can be entered and terminated by a null or blank line. No user identification for designation, no message, or pressing an attention button

on terminal when a message is entering will terminate this module. Otherwise, the message is sent to the designations with the valid TSO user identifications by an assembler routine. This routine actually uses a TPUT Macro of the OS/360 Assembler to issue a terminal output to the given userid designation. An echo message of each delivery is replied for its success and failure.

5.2.2.7 Monitor Student Control

Figure 5.1.g shows the monitor–student control module. This module provides facilities to list the log file from a given starting date to an ending date, and provides turning on and off capabilities to continuously monitors the students from looking at the log file in every period of time.

Firstly, the transaction log records can be browsed through and translated into descriptive form by providing the module with the starting and ending dates in the syntax of YY/MM/DD. If the ending date is earlier than the starting date, nothing will be listed. In other case, the whole log file from the day it created will be listed when the starting date is entered a zero. If a starting date is given, it should be between the creation date of the log file and the last date recorded on the log file. Since the log file is usually large to hold at least 5000 records, an efficient search method is adopted to locate the records with the specified date.

The search is started from either the beginning or the end of the log file depending which end is closer to the starting date. The search uses an asynchronous file handling to reduce the search time. Since retrieving a record from a file is slow compared with the computing speed of the CPU,

the asynchronous file handling allows multiple of records to be retrieved concurrently. Conventionally, a record retrieval is handled one after another retrieval. The waiting time between the two records actually accessed is much longer than the accessing time of the records. This waiting time can be eliminated by retrieving multiple records at the same time without waiting one retrieval after another. The synchronization of these asynchronous retrievals is critical. The date comparison in the searching process can be done right immediately after an asynchronous retrieval is arrived.

While the transaction log records are listing, a choice to continue or quit the list is provided by hitting the attention button on the terminal. The total number of transaction log records is displayed at the end of the listing.

Secondly, the monitor–student control module provides facility to continuously monitor the student activities. The monitoring is turned on by calling a background routine which runs simultaneously with the CAPSI program and monitors continuously the log file in every 5 seconds of time interval. The background routine can be turned off by the instructor through a switch of a global variable which is shared between the CAPSI program and the background routine. This background routine will reduce its priority when no record is written on the log file by every 10 minutes and will turn itself off when after half an hour of idle time with no record written on the log file. The synchronization of the turn on and turn off is critical and accomplished by a built–in function checking the completion stage of the background routine. No second times of turn on and turn off is allowed and controlled by the module.

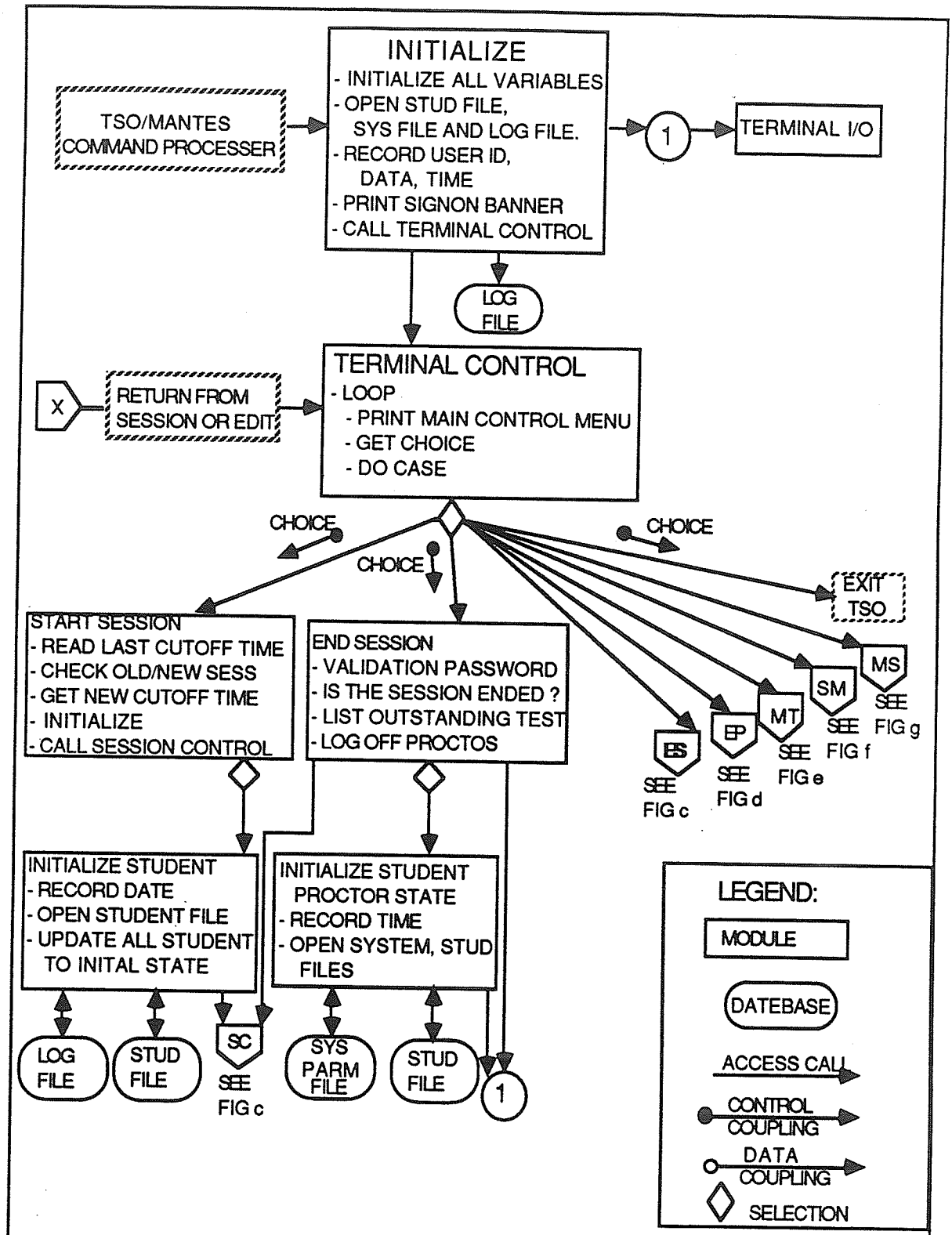

Fig. 5.1.a-g. Structured chart for CAPSI with classroom setting.

(a) Initialization and terminal control modules.

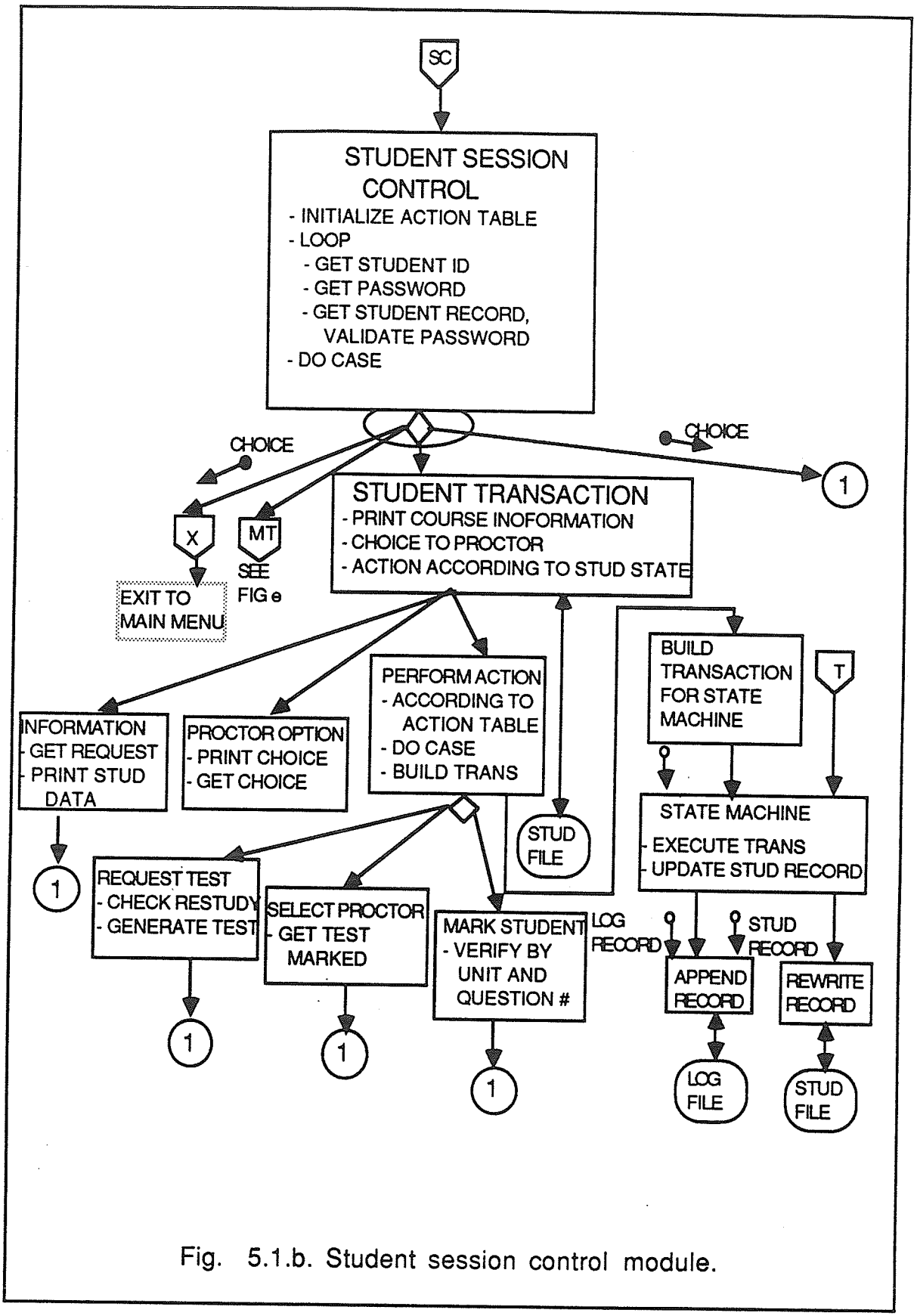

Fig. 5.1.b. Student session control module.

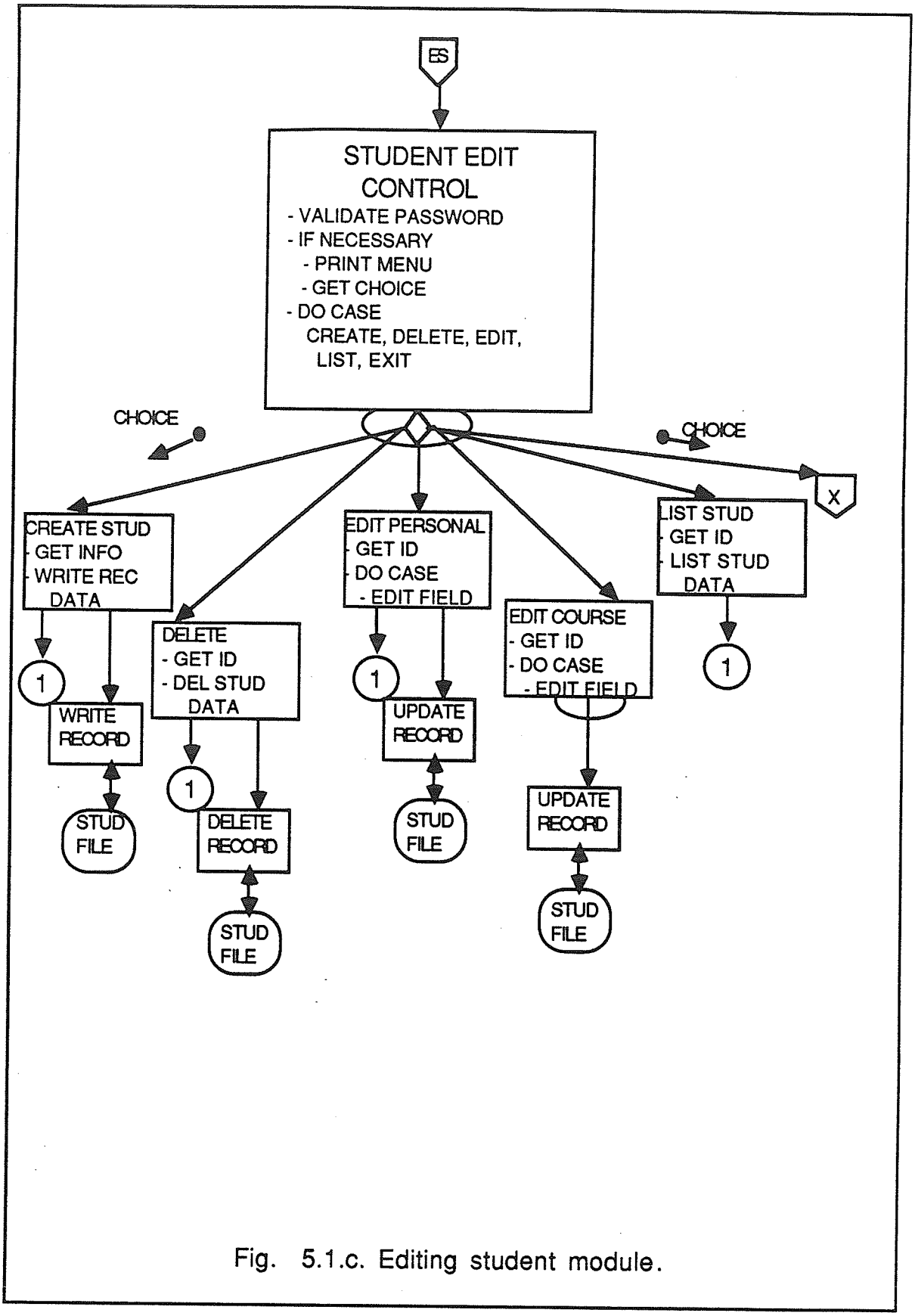

Fig. 5.1.c. Editing student module.

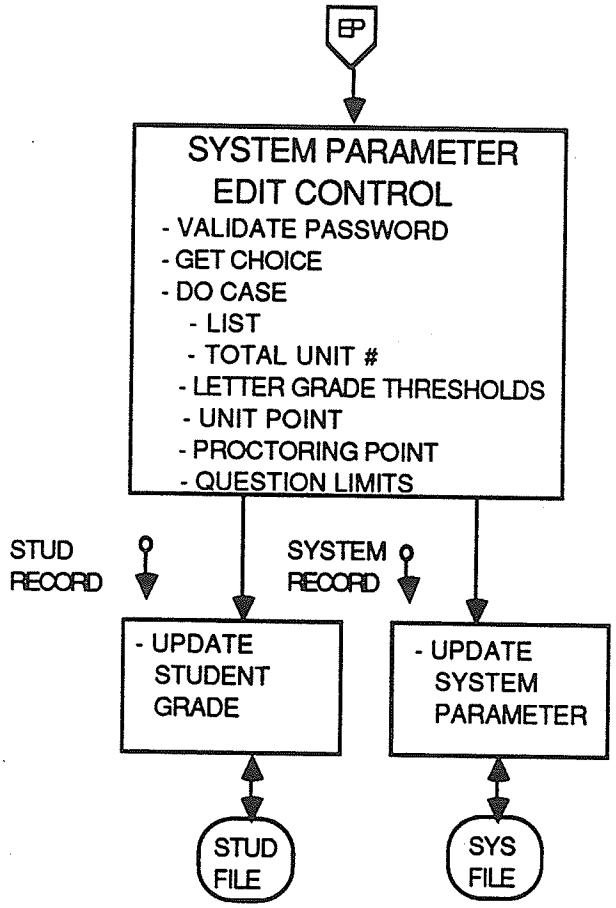

Fig. 5.1.d. Edit system parameter module.

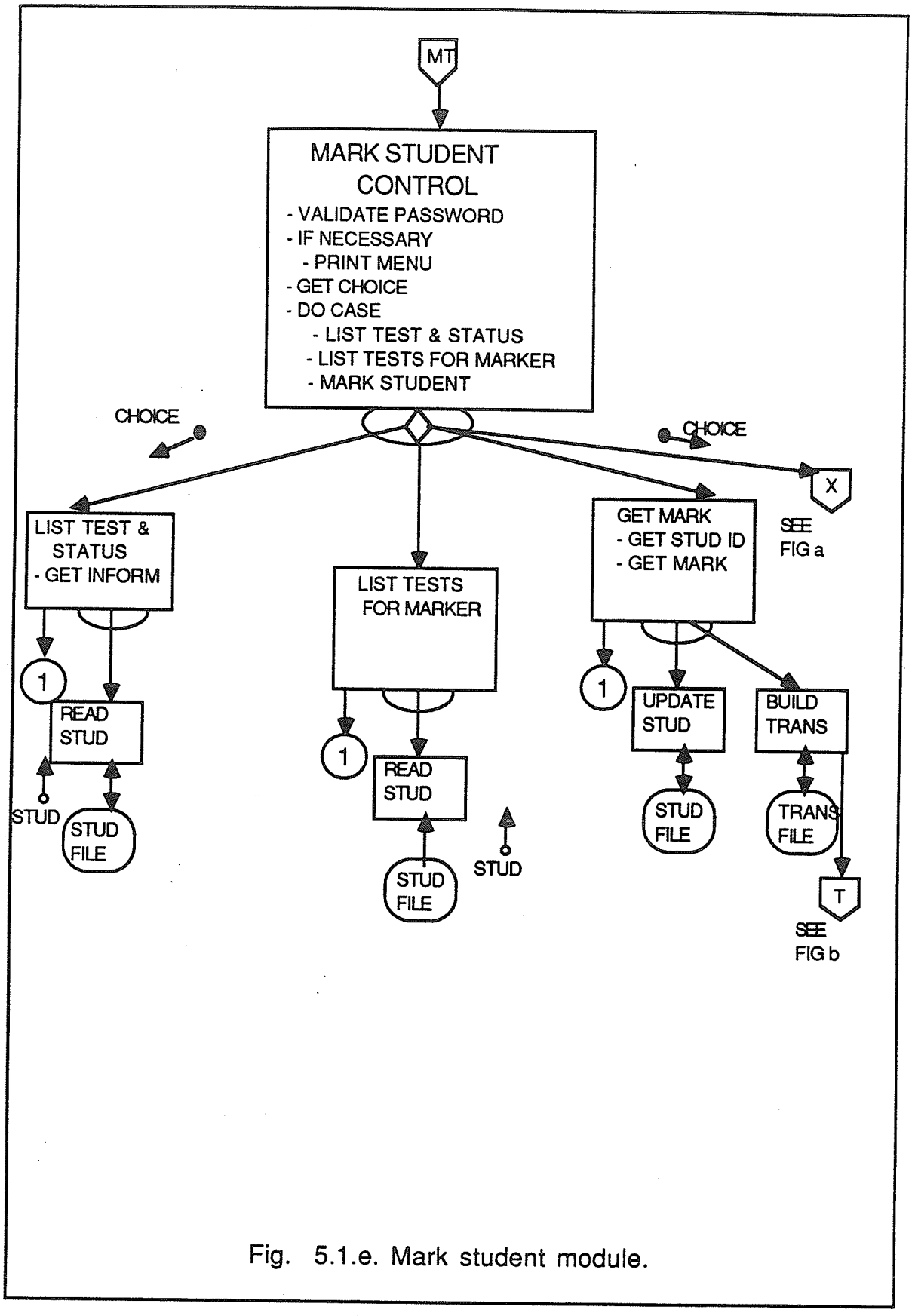

Fig. 5.1.e. Mark student module.

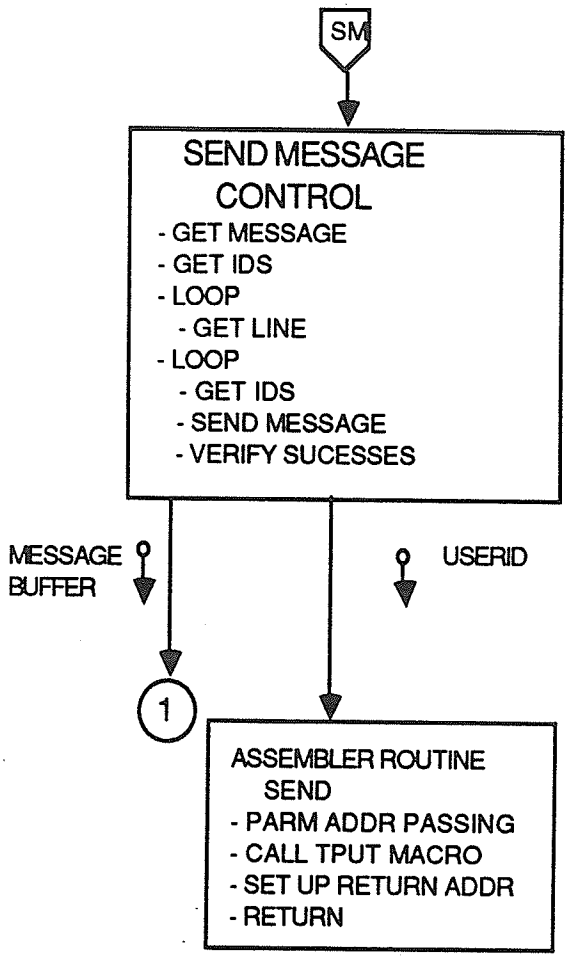

Fig. 5.1.f. Send messages module.

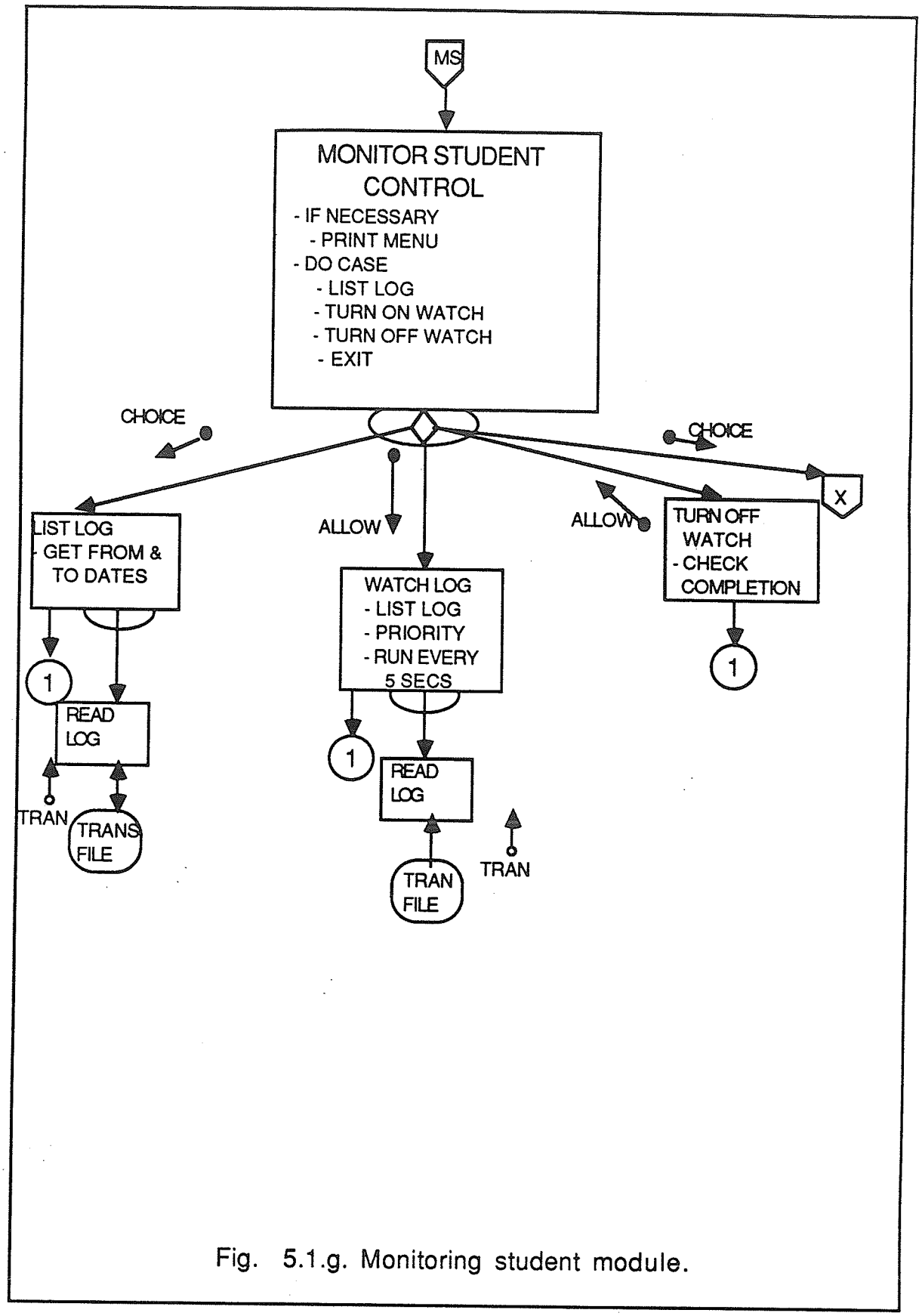

Fig. 5.1.g. Monitoring student module.

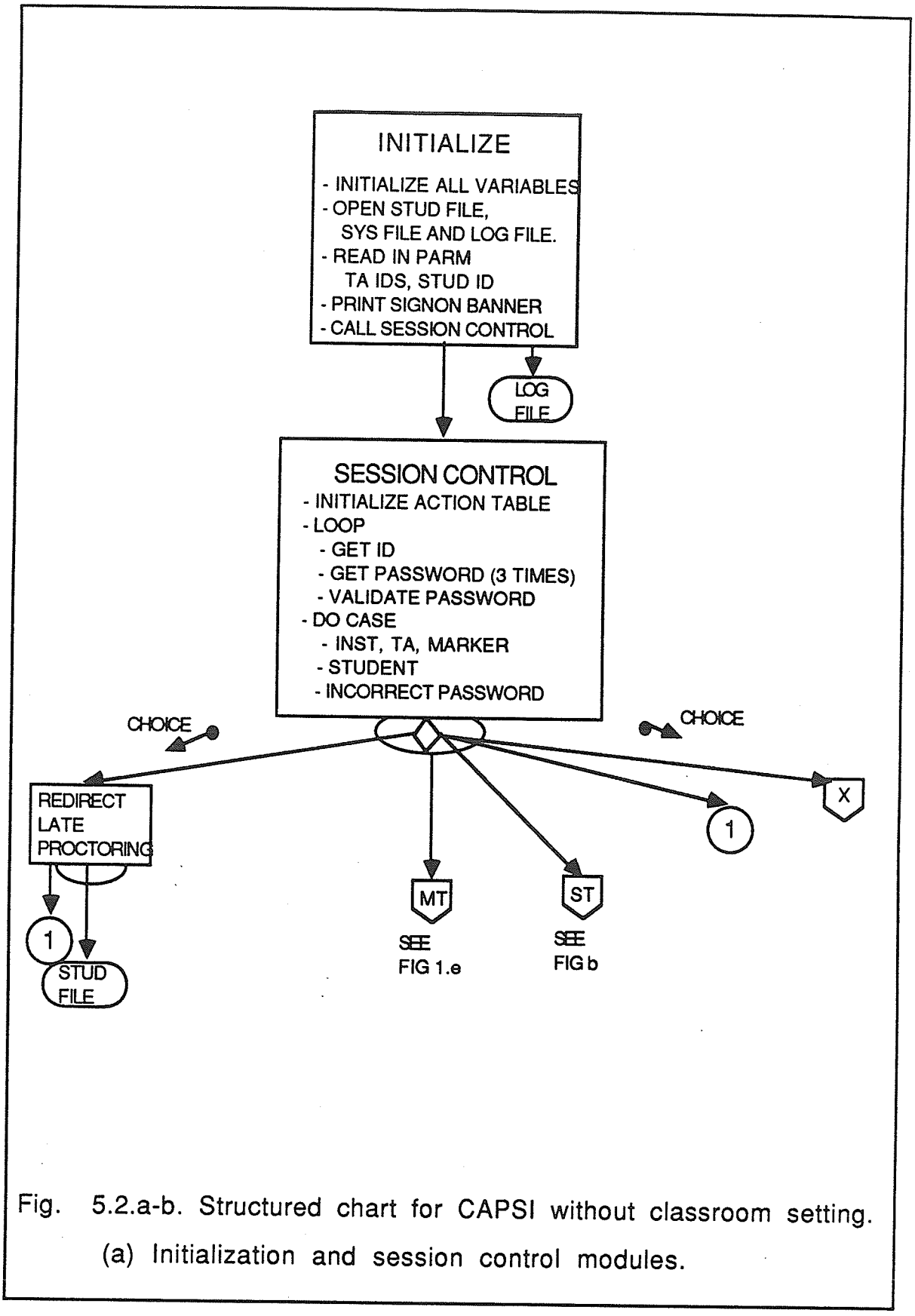

Fig. 5.2.a-b. Structured chart for CAPSI without classroom setting.
(a) Initialization and session control modules.

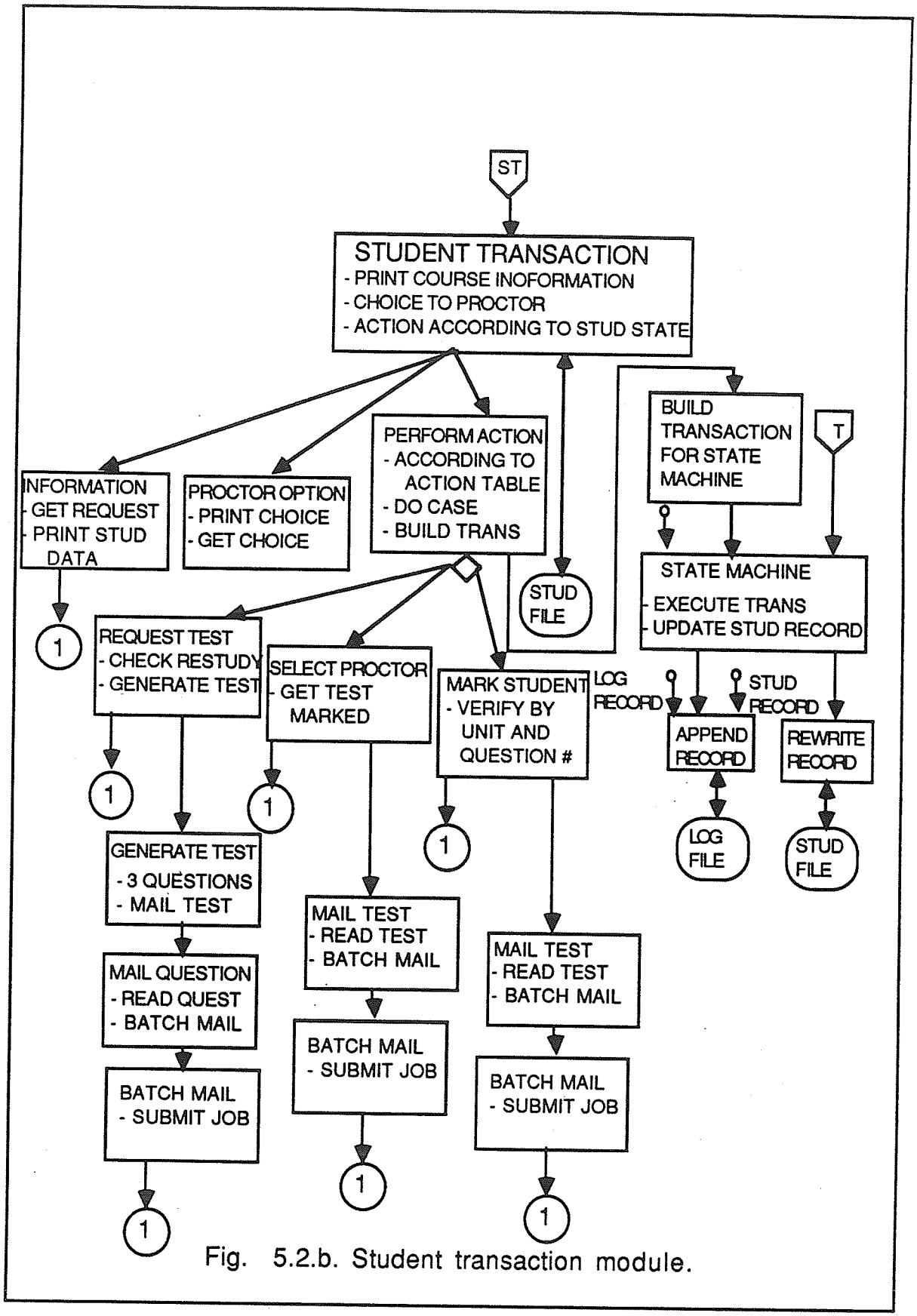

Fig. 5.2.b. Student transaction module.

5.2.3 Program Structure of CAPSI without Classroom Setting

Another small version of CAPSI is designed and used in virtual classroom, without classroom setting. The structured charts are illustrated in Figures 5.2.a and 5.2.b.

5.2.3.1 Initialization and Session Control

As shown in Figure 5.2.a, this control module is a combination of initialization module and session-control module from the version of CAPSI with the classroom setting but having many improvements and modifications. Since this control module can be accessed and called by students, all the unnecessary and privileged modules are eliminated.

The initialization part of this module is much like the initialization module from the version of CAPSI with the classroom setting. Except, this module opens up more internal files for sending and receiving mail, and for retrieving questions from the question bank. Moreover, the user identifications of the teaching assistants, the instructor, and the student invoking this module are passed into the CAPSI through a parameter. The student identification is used as a sender address of the electronic mail. The user identifications of the teaching assistants and instructor are used as the mailing designations of the test when one of these identifications is successively selected as a marker.

The session-control part of this module is much like the

session–control module of the version of CAPSI with classroom setting. Except, three attempts of a valid student identification and three attempts of the associated password of a given valid student identification are allowed in every time this module is invoked; otherwise, the module terminates itself for preventing further illegal attempts. There is an additional function that the instructor can list out all the unmarked students who waited over 24 hours. Finally, this module calls the student–transaction control module.

5.2.3.2 Student Transaction Control

As shown in Figure 5.2.b, the function of the student transaction control is like the one in the version of CAPSI with classroom setting, but the electronic mailing facility is fully implemented into this version of CAPSI.

When a student generates a test, he/she no longer only receives the three question numbers on the terminal, but receives an electronic test paper with the three questions typed onto it. The student then types in the answers of the question onto this electronic test paper. When the student requests the test to be marked, he/she no longer only receives the proctor names and identifications on the terminal, but the answered test will be electronically mailed to the two proctors as well as the instructor. A validation of the question text is made before mailing out the test to the proctors; it avoids the student to change the questions by him/herself.

A student assigned to proctor will receive an answered test through

the electronic mailbox. The proctor then makes comment on the test and calls up the this version of CAPSI to enter the result. After the result is entered, the marked test will be automatically mailed to the student who wrote the test as well as to the instructor for test record keeping.

The integration of the mailing facility is implemented by several modules functioned as mailing out the test questions, mailing out the answered and marked test, and submitting batch job for mailing services.

5.2.4 The Programming Language Employed

IBM's Optimizing PL/I programming language is utilized to implement the CAPSI on the TSO operating system. PL/I (Programming Language I) is the first of the very large and powerful multipurpose programming language used for both the scientific and business problems. In 1964, PL/I was attempted to incorporate many notable features and concepts from Algol 60 (ALGOrithmic Language 1960), COBOL (COmmon Business–Oriented Language), Fortran (FORmula TRANslation), and other earliest languages. Almost all of the conventional and special features of IBM's Optimizing PL/I are employed to develop and implement the CAPSI program.

5.2.4.1 Conventional Features

PL/I provides a fixed set of built–in types (i.e., integer, real, boolean, character, bit) as well as mechanisms for structuring more complex data types starting from the elementary ones (i.e., record structures, arrays,

pointers). The related fields of a student are grouped into a structured record, so the entire record given a name can be used to refer to the whole sets of the student fields.

```
DCL 1 STUDENT,
      2 ID          CHAR(7),
      2 NAME        CHAR(30),
        .
        .
      2 PASSWORD    CHAR(7);
```

Besides the use of pointers, PL/I allows a variable of any kind of data type to be dynamically allocated and free whenever it is required in the program. For example, the unlimited size of the message buffer in send–messages module is implemented in this way of dynamic allocation. Whenever the buffer with the preassumed size is full, a larger size of another buffer is allocated dynamically and then the whole message is copied back from the full buffer to the new buffer by using a temporary buffer.

```
DCL (SIZE,MORE)  INTEGER INIT(100);
DCL BUFFER       CHAR(SIZE) VARYING CONTROLLED;

ALLOC BUFFER;
.
.
if BUFFER is full
   temp=BUFFER;
   SIZE=SIZE+MORE;
   FREE BUFFER;
   ALLOC BUFFER;
   BUFFER=temp;
end if
new buffer with extended size
```

The IBM's PL/I allows an argument string of 100 characters or less to be passed to the PL/I main program while the program is invoked from TSO. In the case of CAPSI, the important information of the TSO user identifications of teaching assistants, instructor, and the student invoking CAPSI are passed into CAPSI.

```
MPSI : PROC(PARM) OPTIONS(MAIN);
  DCL PARM     CHAR(100) VARYING;
  •
  •
  retrieve user identifications from the parameter PARM
```

To pass the parameter string in TSO:

```
CALL 'PSI.LOAD(member name)' '/argument of
                the user identifications'
```

PL/I provides a full set of built-in functions to support all its conventional and special features. The built-in functions that CAPSI used can be classified into arithmetic, string-handling, condition-handling, muti-tasking, and miscellaneous. The miscellaneous built-in functions used are the PLIRETV interfacing with the assembler language, and the DATE and TIME interfacing with the system clock. The string-handling built-in functions simplify the processing of bit and character strings. These functions are used in quite a number of places in CAPSI such as the bit manipulation for the pseudo-random number and the character manipulation for the composition of the electronic test paper.

Finally, only some important and significant conventional PL/I features used in CAPSI are highlighted in above, but many more conventional features used in CAPSI are not described in here.

5.2.4.2 Condition–Handling Feature

When a PL/I program is executed, a number of conditions are detected if they are raised. These conditions may be errors, such as zero–division, or may be conditions that are expected, such as the end of an input file. When a condition is raised, the program will interrupt its flow and will execute an action either from the on–unit provided by the programmer or the implicit action provided by the PL/I. There are condition–handling built–in functions to investigate the cause of the interrupt and to recover from abending the program execution. In this way, the behavior of the program becomes totally predictable even in anomalous situation.

There are several expected conditions in CAPSI are handled such as handling an end of a file, an invalid index key of a record, and the attention button hit on the terminal. When all student records are sequentially browsed through, an end–of–file interrupt occurs and then the end–of–file on–unit handles the situation and terminates the further retrieval of any student record.

```
        ON ENDFILE(STUDFIL) GOTO EOF;
        .
        .
        loop
          retrieve record
        end loop
EOF:
        continue
```

When an invalid student identification is entered for logging onto the CAPSI, the failure of retrieving this non–existent student record from the student file causes the key on–unit to reject the illegal attempt to log on.

The attention condition is raised when the user signals attention at the terminal during interactive processing. If no attention on–unit is provided properly, the program will be terminated and passed control to the terminal. In the case of CAPSI, an attention at the terminal is only accepted to continue or quit a long list of information listing out on the terminal, or to terminate the message sending utility while the message is entering. In other cases, CAPSI just ignores any attention at the terminal to avoid illegal termination and damage of the program.

5.2.4.3 File–Handling Feature

PL/I supports a wide range of data set organizations with several types of data transmission and of accessing methods. First of all, the student file uses the Index file organization using the accessing methods of Indexed Direct Access Method, IDAM, and Indexed Sequential Access Method, ISAM.

```
DCL STUDFIL  ENV(INDEXED KEYLOC(2) KEYLENGTH(7)
     RECSIZE(107) BLKSIZE(107) F BUFFERS(35));
```

The key of the file indexed is seven characters long at the second character position of each record. Each record contains the 107 bytes of student information. Finally, the rest of other files are similar to the above file, but some use Regional file organization or Virtual Storage Access Method, VSAM.

5.2.4.4 Multi-Tasking Feature

PL/I allows some background routines called tasks run simultaneously with the main program. The execution of such concurrent tasks is said to be asynchronous. PL/I provides multi-tasking capabilities to assign priority to each task and to synchronize the execution sequence among tasks. The multi-tasking feature is used in CAPSI to provide monitoring purposes.

```
DCL STAGE          EVENT;
DCL WATCH          TASK;
     •
     •
     CALL monitor_routine TASK(WATCH) EVENT(STAGE)
          PRIORITY(-200);
     •
     executing simultaneously with the monitor routine
     •
     WAIT(STAGE);
     wait until the monitor routine terminates
```

The task and the stage of the task are declared. The monitor routine is

invoked from the main program to execute concurrently with the main program. The main program will wait for the termination of the monitor routine routine if the monitor routine is still executing; otherwise, the main program will continue its execution. PL/I also provides some features to assign priority to, to check the completion of, and to terminate the execution of an executing task.

5.2.4.5 Debugging Feature

PL/I provides a large amount of debugging features in both compilation and execution of program. The Checkout compiler is designed to provide extensive facilities for program checkout and debugging. Excellent error and warning diagnostics are provided at both compile time and execution time. Also, PL/I provides separate compilation to develop large program such as CAPSI. In execution time, specific debugging options can be indicated in the PLIXOPT variable.

```
DCL PLIXOPT  CHAR(50) VARYING STATIC EXTERNAL
        INIT('FLOW,COUNT');
```

For examples, the flow debugging aid produces a tracing output of execution when an error occurs and the count debugging aid produces the statement frequency tables used to compare the program with the original design. The separate compilation and some debugging aids of PL/I give advantages to the development of CAPSI. Furthermore, the PL/I Optimizing compiler is provided to produce efficient object code to improve the performance of a program at execution time considerably. Therefore, CAPSI uses the

optimizing compiler for the final testing and for the production runs.

5.2.4.6 Interfacing with Assembler

PL/I can interface with the low level language OS/360 Assembler Level G by declaring as following:

```
DCL SEND  EXTERNAL OPTIONS(ASM,INTER,RETCODE)
     ENTRY(CHAR(*) VARYING, CHAR(8));
```

This assembler routine send accepts a message of any length and an userid with 8 characters long. This assembler routine is coded in OS/360 Assembler using a TPUT Macro to send a message to the given userid. The PL/I built–in function PLIRETV returns the completion code of the message sending.

5.2.4.7 Interfacing with MANTES File

The Computer Services provides a NAM Access Method (NAMAM) to allow PL/I programs to read and write MANTES file through VSAM. The NAMAM is used for CAPSI to read questions from the question bank file and to read tests from students.

```
DCL TEST  FILE KEYED INPUT RECORD
          ENV(VSAM KEYLOC(0) KEYLENGTH(255));
```

Any file record in the MANTES group can be accessed through the VSAM access method by providing the keys of file name and sequence number of

the record.

5.2.4.8 Interfacing with the System Dynamic Allocation Routine

The Computer Services provides a DYNAM routine which allows PL/I to interface with SVC 99, the system dynamic allocation routines. This DYNAM routine allows system resources to be allocated and free within the PL/I program rather than outside the program through either JCL (Job Control Language) or TSO (Timesharing Option) commands. In fact, CAPSI uses this routine to dynamically create files like electronic mails and batch JCL jobs for postal services in MANTES. The JCL job submission within CAPSI is made possible by writing the JCL commands into the system reserved file INTRDR which interfaces with the JES (Job Entry System). The job submitted is a batch MANTES to execute some MANTES commands for the postal services.

```
DCL DYNAM  EXTERNAL OPTIONS(ASM,INTER,RETCODE)
          ENTRY;
DCL 1  WORK,
       2  WA_LEN   FIXED BIN(31) INIT(2000),
       2  FILLER     CHAR(2000);
  •
  •
  FETCH DYNAM;
  •
  •
  CALL DYNAM(WORK,'ALLOC ','DD=MAIL;', ••
  operands equivalent to DD (Data Definition) statement in JCL ••);
  •
  •
  create mail file
  •
  CALL DYNAM(WORK,'ALLOC ','DD=BATCH;', ••
  •
  •
  create batch job file
  •
  CALL DYNAM(WORK,'ALLOC ','DD=INTRDR;', ••);
  interface to JES
  PUT FILE(INTRDR) EDIT(
    '// JOB ',
    '// EXEC MANTES'
    ••) (SKIP,A);
  job submission through INTRDR system file
```

As a result, all the test deliveries are handled by CAPSI and are transparent to students. In this way, CAPSI facilitates a highly–structured and global form of communications and interactions among students, teaching assistants, and instructor.

5.3 Summary

Firstly, the hardware system on which CAPSI runs is described. It mainly consists an Amdahl mainframe computer and other supportive peripherals. CAPSI implemented in PL/I runs on the TSO environment. Secondly, the evolution of the CAPSI program is briefly presented. Thirdly, the major modules in CAPSI are described through the structured charts of the program. Finally, the major PL/I features and interfaces employed in CAPSI are described in details. In the coming chapter, it discusses the five–year experience and experimental results of CAPSI.

CHAPTER VI

EXPERIMENTAL RESULTS AND DISCUSSIONS

6.1 Subjects Taught

Since 1983 CAPSI has been an on-going project at the University of Manitoba [KiPe88a] and the following courses with the range of 20 to 65 students have been taught:

1. Introduction to Psychology with 2 times on campus and 3 times off campus;
2. Behavior Modification Principles/Applications with 5 times on campus and twice off;
3. Learning Foundations of Psychology with 5 times on campus;
4. Humanistic and Transpersonal Psychology with 6 times on campus; and
5. Experimental Child Psychology with 4 times on campus.

Although only psychology courses have been taught by CAPSI, other disciplines can also be taught. The CAPSI approach is independent of course content but requires an appropriate set of structured course material.

6.2 Selection of Students for the Program

Students are eligible to enroll a university course by fulfilling the requirements of the corresponding department. More recently, a statement is added in the university's general calendar to inform that courses using

CAPSI are computer mediated, and that students have an option to their participation in the computer–mediated courses. Students enrolled in the long–distance courses are considered highly self–motivated, because they need to self-study and hardly see the instructor or even other students in the same course.

6.3 CAPSI for On–Campus Learning

Over the past five years, CAPSI was utilized to teach five on–campus undergraduate courses in total of 22 times to approximately 600 students at the University of Manitoba. Almost all of the courses were taught by Professor J.J. Pear, the Department of Psychology, one by Professor J.H. Whiteley, the Department of Psychology, and one by both of them.

6.4 Experience and Evolution of CAPSI

The earliest version of CAPSI could be run on only one terminal. The courses that used CAPSI took place in regular class schedule with a terminal connected by phone to the university mainframe computer. Student requested test from CAPSI by receiving three random question numbers from a unit. Then the student referred the questions from a list of the study questions on the unit and answered the questions in duplicate using carbon paper for both the instructor and the student to keep a copy. The student required his/her test proctored by receiving the names of two proctors or instructor from CAPSI. Then the student located the two proctors or the instructor and gave each of them a copy of the answers. The proctors marked the test as the same criteria as by the instructor and entered

the results into CAPSI.

Since only one terminal was used in a class, line–up problem happened and complaint aroused in the courses of more than 50 students. One way to solve the line–up problem was to divide the class into two groups and assign each group to a terminal. The disadvantage of this way was the students in a group could not proctor students in the other group. Another way was to assign each terminal to one of two different courses and allows student to work on their courses during the consecutive class periods. The disadvantage was the unfair between students who could and could not make the extra class period.

In the beginning of 1986, a multiuser form of CAPSI was available for students to access both terminals without splitting into groups. However, line–up problem still happened in large classes, especially near the end of the course but the magnitude of the problem was reduced a lot. Furthermore, a third terminal was available mainly for entering the test results immediately by the instructor and teaching assistants. In addition, a new command to print out the students who were supposed to write tests in the class facilitated the supervision of students by the instructor. As well, a new message sending facility was available in CAPSI for instructor to make correction suggestions to students. This multiuser capability also allowed simultaneous delivery of a course to more than one location, and thus facilitated off–campus teaching.

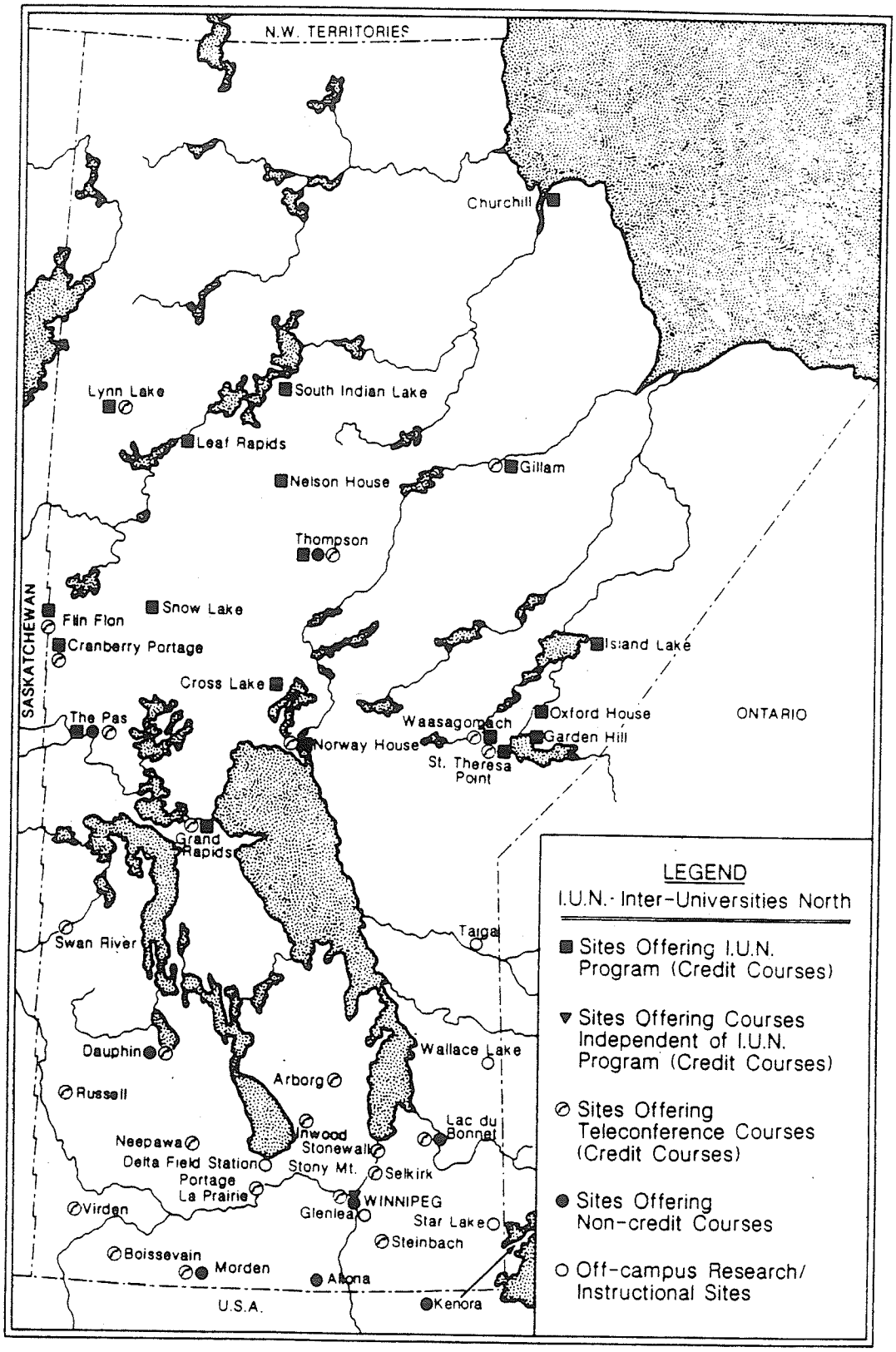

Fig 6.1. The University of Manitoba off–campus instructional sites [From UofM86].

6.5 CAPSI for Off-Campus Learning

Like most other major universities, the University of Manitoba, Winnipeg, Manitoba, offers off-campus courses to people in distant communities as shown in Figure 6.1. With budget cutbacks and high travel rates, an instructor can economically deliver lectures through voice teleconferencing in a number of communities simultaneously.

At the beginning of 1985, a full-year course of the Introduction to Psychology was offered by using CAPSI and voice teleconferencing from Winnipeg to Thompson, a community over 800 km north of the university. The classroom in Thompson contained two phone lines with one accessing an audio-teleconferencing equipment, and the other accessing the CAPSI program in the university. About 20 students and a supervising teaching assistant participated in voice contact with the instructor and in computer contact with the CAPSI program. The instructor marked tests over the phone and entered the results through the terminal in Winnipeg; student proctors marked his/her assigned tests and entered the results through the terminal in Thompson.

Due to the success of the introductory course, a higher level off-campus CAPSI course on the subject of Behavior Modification Principles was offered during the May-June intersession of 1985. The course was taught from Winnipeg to Thompson with the enrollment of eighteen students and to Flin Flon with the enrollment of six students simultaneously. Differences from previous long-distance CAPSI course

were proctors marked tests from the other location over the phone and the instructor could listen and make suggestions or corrections during the marking interchanges.

During the 1986–87 academic year, two half–year off–campus CAPSI courses were offered on the subjects of Behavior Modification Principles and its sequel Behavior Modification Applications in six Manitoba locations: Morden, Lac du Bonnet, St. Boniface, Stonewall, Virden, and Thompson. Unfortunately, 16 out of 60 students registered for the first–term course dropped very early because the necessary computer equipment was not yet set up in the six locations by the time the course began. However, computer equipment was finally set up at all locations within a month and 35 students out of the rest 44 students completed the course successfully. Nevertheless, extra classes were necessary to help students catch up the leakage caused by the equipment confusion. Since no supervisor was available at most locations to monitor the way students wrote their tests, more weight was placed on the mid–term and final examinations to insure high quality of learning.

6.6 Inclusion of Electronic Mailing and Messaging into CAPSI

At the beginning of the 1986–87 second–term, students in off–campus course who had access to either terminals or microcomputers with modems could invoke the CAPSI program at their own schedule and mail their test answers to the instructor for marking by the electronic mailing system of the university computer. About 10 out of 30 students in the course took advantage of this opportunity on a regular basis. The instructor marked and

mailed back the tests with feedback within 24 hours.

At the beginning of the 1987–88 first–term, CAPSI was upgraded to include the capabilities of electronic–mail delivery. In other words, the delivering and receiving unit tests were transparent with respect to the views of students, proctors, teaching assistants, and instructor.

6.7 CAPSI for Virtual Classroom

The upgraded version of CAPSI without any regular class schedule has been utilized to offer courses on–campus as well as off–campus. As far as CAPSI is concerned, there is no distinction between on–campus and off–campus courses because the function of CAPSI is irrelevant to the invoking locations. Since the physical boundary of the classroom vanishes, CAPSI can be used to implement the concept of virtual classroom [Hilt86] in which the physical classroom may be much smaller than the logical classroom with the analogy to the concepts of virtual memory and virtual machine. Ideally, the size of the logical classroom is unlimited. With the existing Canadian and international computer networks, CAPSI can potentially deliver courses to people distributed over large remote areas, such as large provinces or across Canada, and courses in international programs such as the recently announced Commonwealth University intended to provide education to all the Commonwealth countries. Thus, CAPSI can contribute to the formation and operation of virtual campus, such as the Commonwealth University. This generalization is based on the locality principle and can be viewed as an attempt to coordinate local synchronization with global desynchronization in a complex system, as

defined by Kinsner and Pear [KiPe88c].

6.8 Course Statistics

The number of students completing a course appears to vary as a function of the previous CAPSI experience of the students, technical problems of the equipment, and the difficulty of the course material. For example, in an on–campus course of Behavior Modification Principles offered during the first–term of the 1986–87, of 70 students who started the course and wrote at least the first unit test (77 had enrolled), 63 (or 90%) completed the course with at least a grade of "C." In the sequel of the first course, Behavior Modification Applications, of 43 students who started the course (50 had enrolled), 42 (or 98%) completed the course with at least a grade of "A." In an off–campus counterpart to the first course, of 46 students who started the course (59 had enrolled), 36 (or 78%) completed the course with at least a grade of "C." This low percentage of students completing the course was probably due to the technical problem of equipment not available at the beginning of the course. In an off–campus counterpart to the sequel of the first off–campus course, of 21 students who started the course (21 had enrolled), 20 (or 95%) completed it with at least a grade of "B+." From the above examples, the numbers of students completing in the sequels of the on–campus and off–campus courses were increased probably due to the previous experience of the students with the CAPSI method. On the other hand, in a conceptually more difficult course, Learning Foundations of Psychology, taught on campus during the first–term of the 1986–87, of 55 students who started the course (70 had enrolled), 38 (or 69%) completed it with at least a grade of "C." Thus, it is

evident that percentages of students completing a course can vary widely by using the CAPSI method as well as the other methods, and probably for the similar reasons.

6.9 Data Logged by CAPSI

The CAPSI records the interactions of the course, which includes all marking transactions, the type and result of each transaction, and the date and time of transaction. The progress of students in the course and information about the functioning of the course itself are recorded and analyzed.

6.9.1 Student Performance

The first-term on-campus "Humanistic and Transpersonal Psychology" course from 16 September 1987 to 15 December 1987 is recorded [KiPL88]. The test frequencies on units completed and proctor frequencies are plotted against the time of the course. The course had a total of 14 units, and the proctor score for each test marked was half of the unit score for each test completed. In each graph, successful test attempts are indicated by the step crawling, while each unsuccessful test attempt by a cross as voluntary cancel of the test and by a triangle as not passing the test, the word restudy is used instead; the conditional successful test attempts are indicated by a square. The dates and day numbers are placed under the graph.

Note from Figure 6.2 that student #AN started the course during day 7 (22 September), passed units at a high rate, and completed the units

during day 48 (2 November), 5 days after the middle of the course. This student was self–motivated and worked steadily throughout the course. He/she even completed unit 3 and 4 at day 10 (25 September). He/she totally got three conditional passes at the following days: 7 (22 September) for unit 1, day 19 (4 October) for unit 6, and day 21 (6 October) for unit 7. Furthermore, he/she voluntarily canceled unit 11 at day 42 (27 October) and did not get any restudy throughout the course. He/she earned proctor scores as a result of serving as a proctor at a steady pace. He/she quitted proctoring 6 days before the last day of the course. He/she could have earned few more proctor scores, but he/she might spend this period for studying his/her final examination.

As shown in Figure 6.3, student #SL started the course in a later time at day 14 (29 September) and struggled through the unit 1 until day 35 (20 October), 9 days before the middle of the course. He got a restudy at day 14 (29 September) for unit 1. Since then, he/she needed another 21 days in order to conditionally pass the unit 1. This phenomenon could be interpreted that this student needed a longer period to get used to the computerized system or he/she was a person that started his/her work late. Because of his/her late starting, he/she had to work much harder in the second–half of the course in order to get good results. He/she completed the rest of the test units in a fast pace with only a voluntary cancel at day 63 (17 November) for unit 11. As well, he/she earned proctor scores in a very fast pace during the second–half of the course. For example, he/she even earned three proctoring per day three times at the following days: day 56 (10 November), day 78 (2 December), and day 79 (3 December); and earned two proctoring per day at the following two days: day 71 (25 November) and day

82 (7 December). Even though the students #SL and #AN have achieved almost the same scores, the time spreads of learning are different, which produce different qualities of learning.

As shown in Figure 6.4, student #JO started two weeks later, conditionally completed the first unit at day 16 (1 October), and then did no further work in 20 days. Until day 37 (22 October) and day 41 (26 October), he/she completed unit 2 and unit 3 respectively, and then again did no further work in 20 days until day 62 (16 November), only 24 days left before the last day of the course. Finally, he/she completed other eleven unit tests in such a rush within this period of 24 days with one conditional pass of unit 5 at day 64 (18 November), and one voluntary cancel of unit 8 at day 72 (26 November). He/she passed unit tests at a very high rate and just managed to complete all the units by the very end of the course. However, he/she did not earn any proctor points in the course, which was probably due to his/her choice not to proctor, his/her personal reasons, his/her slow pace in advancing the units, or insufficient time to mark tests during the last four weeks of the course. Unfortunately, he/she did not gain or reinforce his/her course knowledge through the process of proctoring.

As shown in Figure 6.5, the class average performance regarding the frequencies of test and proctoring by all students without considering the dropped students is plotted. The diagram indicates how an average student should achieve throughout the course. From the statistics of the course, 61 students registered for the course and 5 students dropped out from the course. When the course started, the number of the unit test generated was more than the number of proctoring until tests generated was more than

the number of proctoring until at day 14 (29 September). From that day, the number of number was far more than the number of unit tests generated. It is because each test requires to be proctored twice by two students. The number of proctoring on the diagram is not exactly twice the number of the tests generated because some tests are marked by teaching assistants or the instructor. The two curves show the steady pace of advancing in the course.

6.9.2 Workload Dynamics

Figure 6.6 shows the frequency of tests marked by teaching assistants or instructor and the frequency of tests marked by student proctors. The two curves crossed at the frequency unit 50 at day 13 (28 September) which indicate that the teaching assistants or instructor marked most of the tests at the first two weeks of the course. This can be explained that the students must pass the units before being selected as proctors. On the other hand, the curves indicate that almost all the marking was done by proctors near the end of the course. It is because students were able to mark more tests as they completed more units and credited more scores from proctoring. Thus, the instructor was very busy in marking tests near the beginning of the term, but had more time to supervise the marking by others and to have more interaction with students as the course went by. The number of tests marked (workload) transited from the teaching assistants and instructor to the students during the course. At the end of the course, the number of tests marked by students is six times of the tests marked by teaching assistants or instructor. It is understandable that a test is marked only once by either a teaching assistant or instructor but twice by two student proctors. The workload transition is dependent on a number of factors such as the

number of students, number of days in a course, and number of teaching assistants. This relation shows the dynamics of the transactions, and provides the foundation for a more formal study of course efficiency optimization.

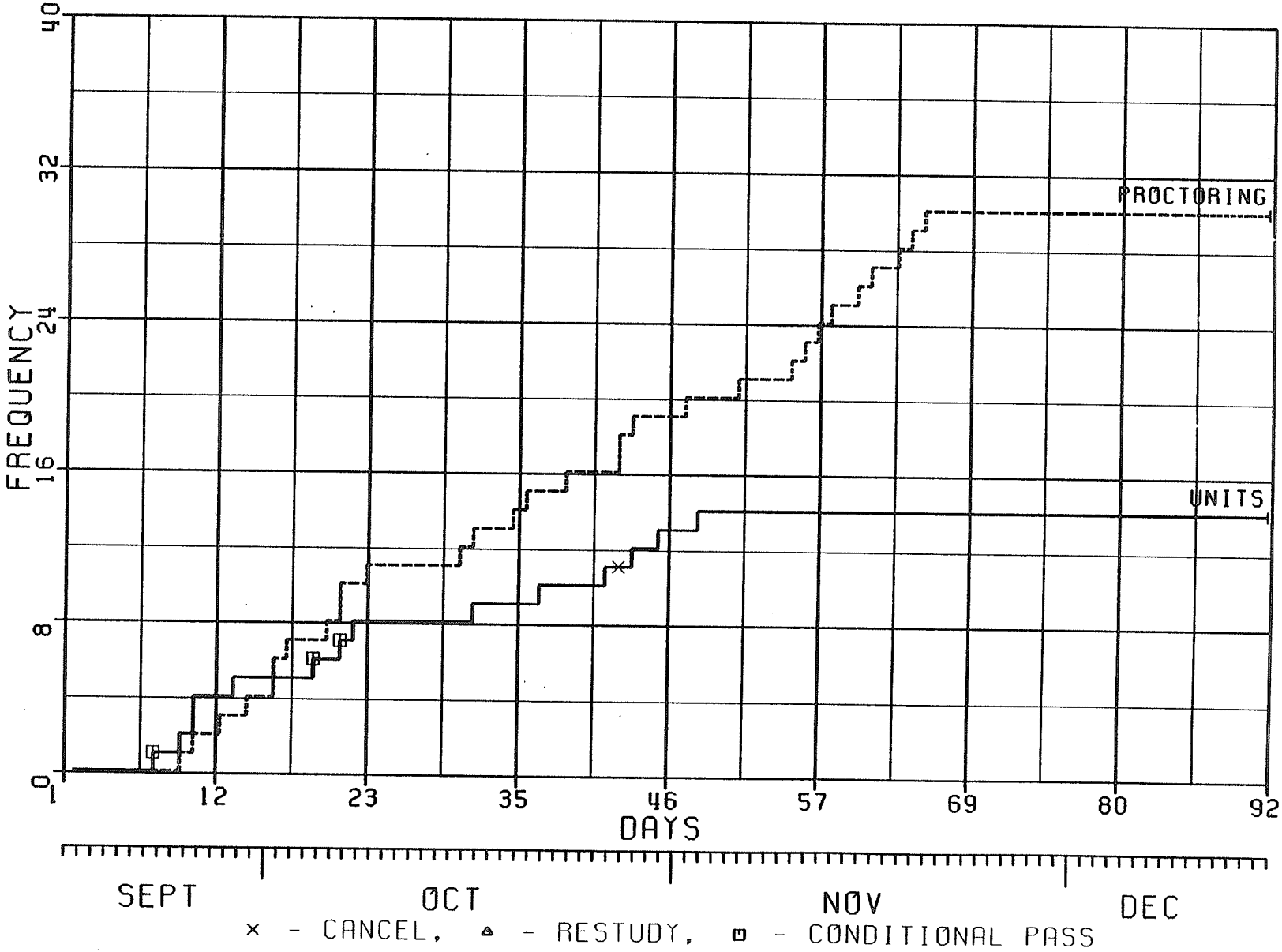

Fig 6.2. Test & proctoring performance of student #AN.

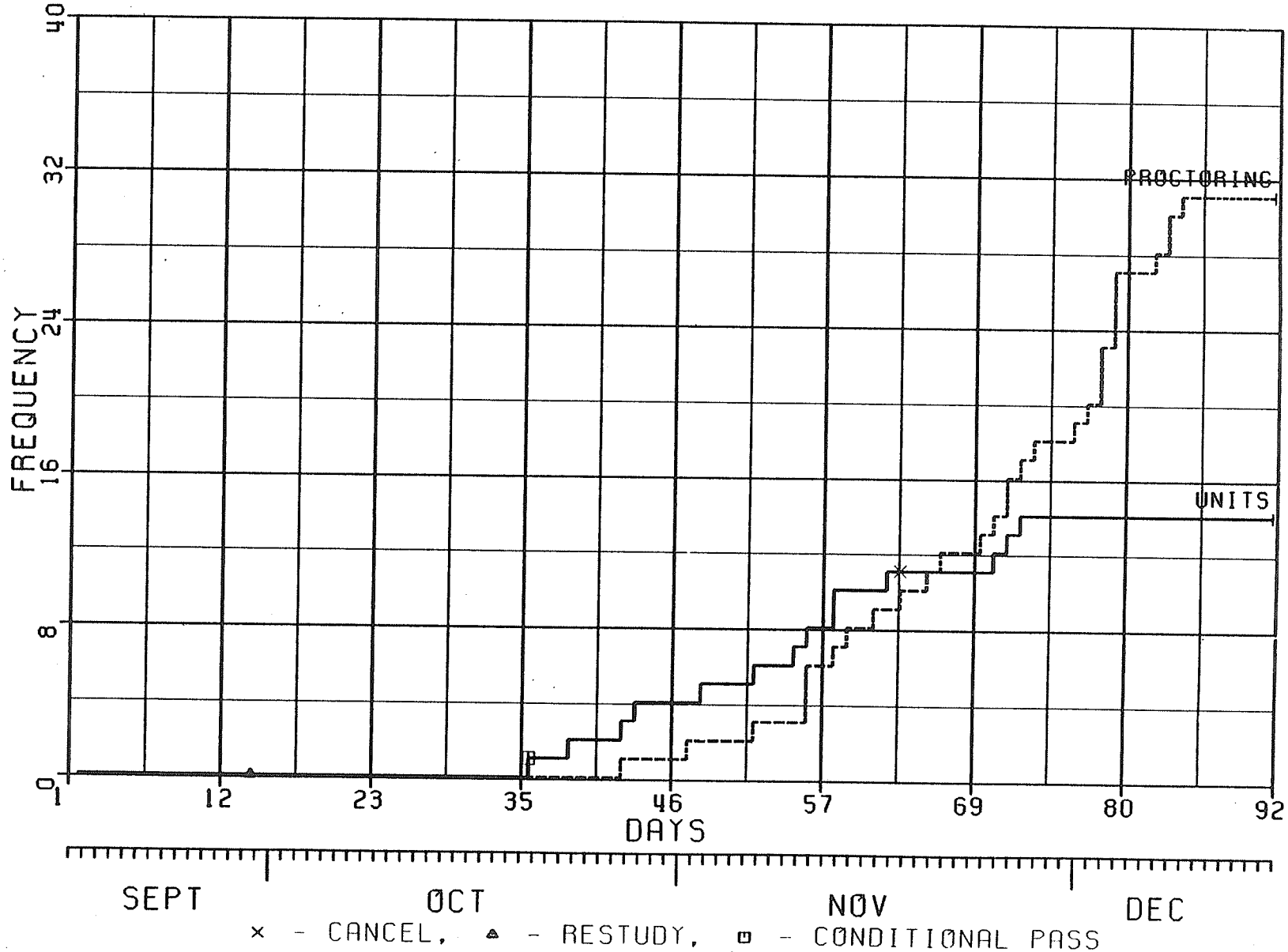

Fig 6.3. Test & proctoring performance of student #SL.

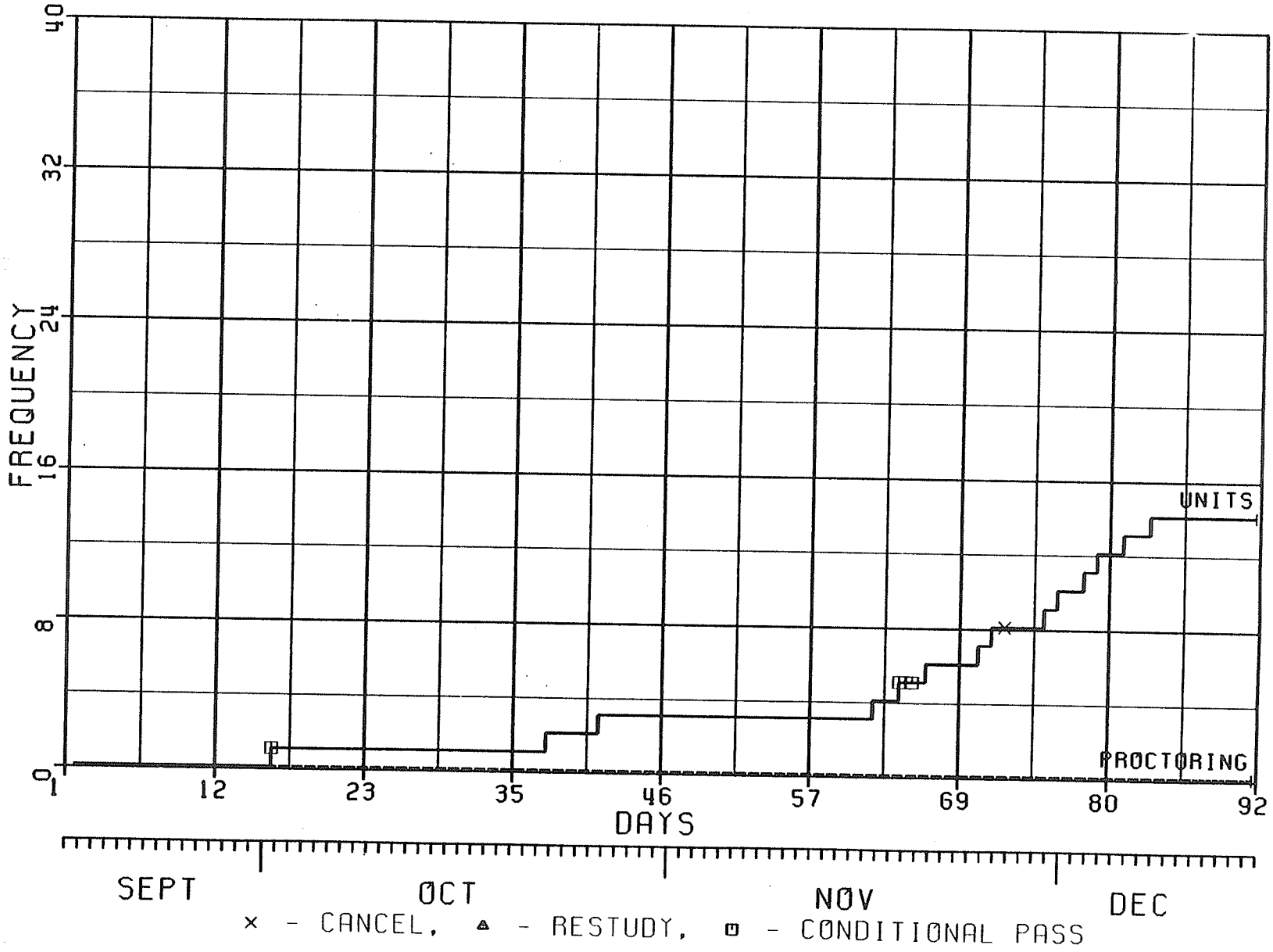

Fig 6.4. Test & proctoring performance of student #JO.

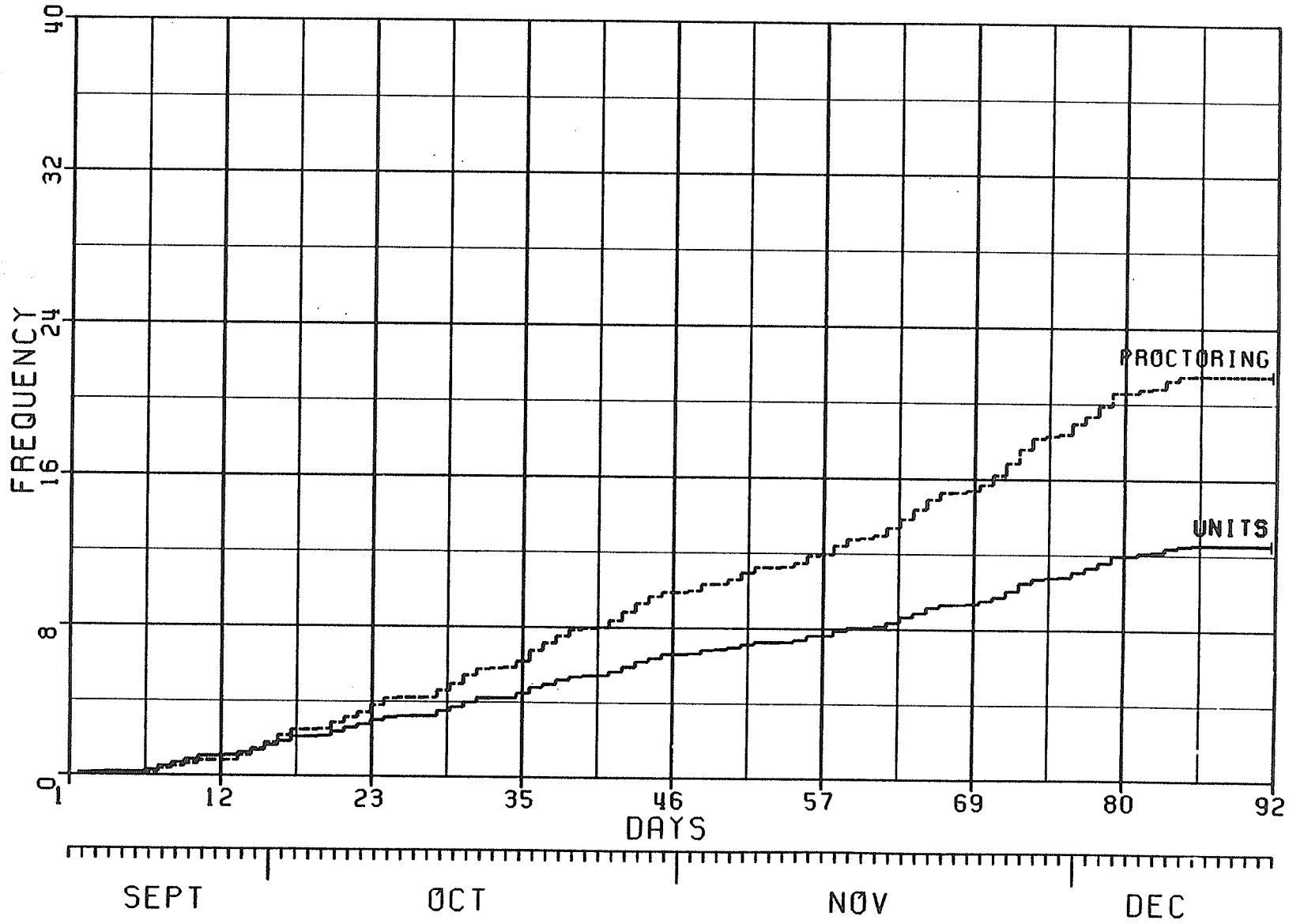

Fig 6.5. Average test & proctoring performance.

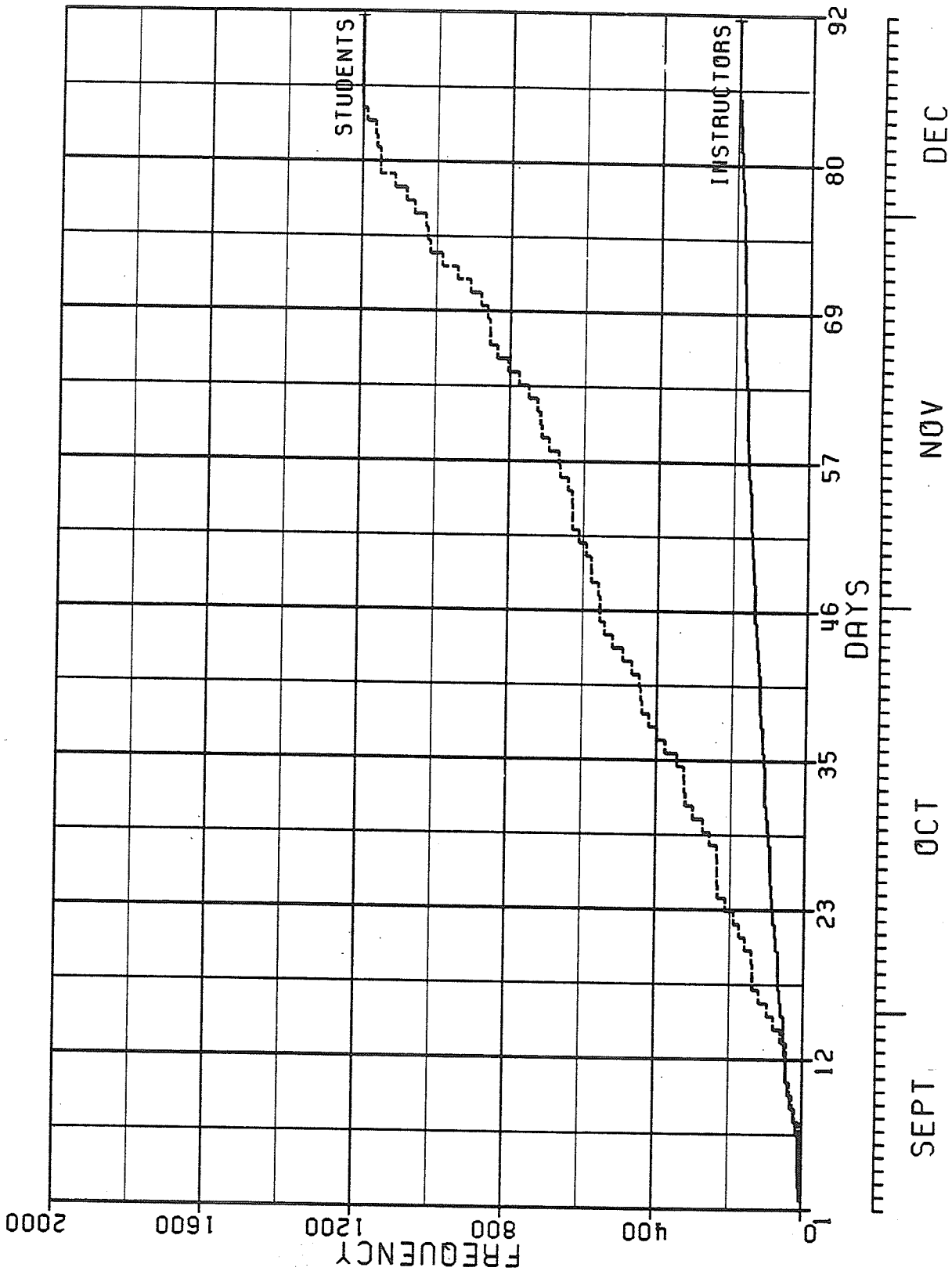

Fig 6.6. Total proctoring performance.

6.10 Problem of Supervision

Unsupervised tests written by students outside the regular class periods raises the possibility of not learning the material well by using unauthorized aid. Several ways are used to deal with the problem of supervision. Firstly, high-weighted and supervised mid-term and final examinations are ensured that students need to learn the test material well in order to pass the examinations. Secondly, the instructor periodically checks the performance of student proctors to ensure that these students learn the material well in order to perform adequately as proctors.

6.11 Student Reactions to the Method

Overall, most CAPSI students appear to react favorably to the teaching method. Some of the positive comments made on a recent CAPSI course evaluated by students taking it with teleconferencing were:

> The self taught aspect of the course allowed me to retain a greater amount of the material discussed in the chapter.
>
> You have to really know the material. If you do all the work you should get a good mark.
>
> It [the PSI method] really makes you understand the material.
>
> I found by teaching myself & rewarding myself for good grades, it kept my interest up and urged me to go further than I had in many other courses.
>
> The chapter tests forced me to learn each chapter well. Therefore I was more ready to write the final exam (and mid-term).

You can get a very good mark — it's guaranteed, if you work hard ! There are no surprises or trick questions. You already have a pass before you write your final exam.

The methods are concise and sensible. I felt I covered and learned the material probably as well or better than any other method I have experienced.

No need for babysitters or to pay for gas. Can do course at own leisure in privacy of own home and still get feedback immediately through computer terminal. Can ask questions through computer terminal vs. correspondence [where you] have to wait a few weeks for a reply.

I've had to work harder than in some other courses, but I'll also end up with a good mark !

Negative comments were mainly focused on technical problems, absence of lectures and class discussions, and the amount of work required for the course. For example,

The weakness of the course was the delay in the arrival of the text book and the delay in the availability of the computer.

Would appreciate a question and answer session with professor at end of each class. A lecture or two would be nice.

I prefer at least some lectures to help explain some concepts.

Not enough class discussion or student/professor interaction.

Would like a change in the system which allows for some discussion of the material.

Maybe a little more lecturing on the course might help.

It seemed to be alot of material to cover in the amount of time.

There is a great deal of reading.

The positive comments consolidate the previous research that PSI is a highly effective teaching method ([ShRS82b], [KuKC79]). Most students with the CAPSI courses completed rate the CAPSI courses are at least as good or better than any other courses using other approaches and feel that they learn better with CAPSI approach than with the lecture approach. As well most CAPSI students consider the self-pacing and proctoring to be the major strengths of CAPSI.

The negative comments point out the deficiencies of CAPSI courses, which should not be difficult to correct. With regard to the computer equipment, it becomes more convenient that most school divisions with the equipment generously permit it to be used for college and university courses in their communities. Moreover, home computers or office computers with permission to be used in course work are other sources of the computer equipment. The course textual material can be immediately delivered by electronic mail once the equipment is available.

Regarding to the absence of lectures, the students should be informed and explained beforehand that lectures will not be given and the reasons for this, because students often tend to equate teaching with lecturing. The reasons lectures are unnecessary and achieved in other means are:

1. CAPSI uses well-defined study objectives with minimized clarification in order to master the material, and
2. students can inquiry and discuss the material with the instructor and other students through computer facilities of mailing and messaging.

If it is really desirable, audio- or video-taped lectures could be mailed to

student and occasional lectures or discussions could be even provided through teleconference equipment.

Regarding to the amount of course work, most students seem to understand that a good deal of work can virtually guarantee a good grade with high quality of learning. Sometimes, it is easy to overestimate the amount of student work, but an instructor with previous CAPSI experience can set a reasonable workload for students in a CAPSI course.

6.12 Costs

The University of Manitoba Computer Services Department provides free of charge computer equipment, computer accounts, and computing time for students in a course to utilize the mainframe computer by the recommendation of the corresponding academic department. As a result, no computer or equipment cost is required to offer any on–campus course.

An off–campus CAPSI course can be offered at no greater cost than a standard teleconference with lecturing approach. CAPSI can be invoked elsewhere in Canada and the United States through computer networks, such as Datapac; the costs of the Datapac connection is assumed by the university. Moreover, no charged long–distance calls to Datapac ports from anywhere within Manitoba are provided by the Manitoba Telephone Systems. Hence, long–distance charges for the use of CAPSI at the university are nil within Manitoba, and nil or minimal outside Manitoba. Even though long–distance calls are charged for the teleconference when it is used in conjunction with CAPSI, the cost is certainly less than the one in

a standard teleconference course because CAPSI course does not require as much lecture and discussion time as in a lecture course.

The CAPSI is also economical because the way student proctors from the course are utilized. Indeed, one instructor, possibly one or more teaching assistants to help out in the course, can teach many more students without sacrificing the quality of learning than any other instructor does in conventional courses. Finally, a cost formula for calculating the total cost of on–campus and off–campus courses using CAPSI is formally defined by Pear and Kinsner [PeKi87].

6.13 Summary

The five–year experience with CAPSI suggests that it is a powerful teaching method with wide generality. Although by far only psychology course has been taught, there is no reason that it could not also be successfully used with other subjects, just as has been the case with PSI. This chapter describes how CAPSI is used for on–campus and off–campus education, how the students perform in courses and react to the CAPSI method, how CAPSI can be generalized for virtual classroom, and how the data obtained in courses is analyzed. In the analysis of data, different performances among students and the dynamics of marking workload between instructor and proctors are discussed. Finally, the cost to offer CAPSI on–campus and off–campus courses is described. In the following last chapter, the conclusions are drawn and recommendations are suggested.

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

Over the past five years, the experience with CAPSI suggests that it is a powerful teaching method with high versatility: it can be used in a number of different ways in a variety of courses. Base on the experience that CAPSI has offered a total of 26 times and taught an approximate total of 700 students at the University of Manitoba, CAPSI shows considerable promise for effectively and economically delivering on–campus and long–distance education. In addition, CAPSI eliminates the spatial and temporal restrictions of both on–campus and off–campus education. CAPSI can be used not only in a physical classroom, but also in a virtual classroom, with no scheduled classes and specific location. Thus, CAPSI can contribute to the formation and operation of virtual campus.

The success of CAPSI is due to the explicit incorporation of engineering concepts and behavioral psychology. In keeping with the engineering approach on which it is based, CAPSI is conceptualized as a modular approach to ideally incorporate other effective devices or procedures that complement to CAPSI. Furthermore, CAPSI founded on the concept of behavioral and applied psychology is rooted in the system of learning–reinforcement theory. In CAPSI, positive reinforcement is immediately, frequently, repeatedly applied to students and proctors throughout the course to achieve and reinforce their successes of learning. As a result, CAPSI brings to the forefront a number of commonalities between engineering, behavioral psychology, and educational psychology

thus indicating the potential for fruitful cooperation between those three fields in order to develop a unified method for teaching and learning in diverse areas.

CAPSI is a technological educational innovation in which technologies are utilized and exploited to the fullest. Engineering is most characterized by the utilization of technology. CAPSI utilizes the communications and computer technologies. Since CAPSI is a highly–structured form of communications, CAPSI exploit the computer communications technology for structuring and facilitating the man–man interactions.

Moreover, the utilization of computer technology assists CAPSI the functions of communications, management, measurement, quality control, and research. With its well–designed research database, CAPSI provides a complete and readily accessible record of all testing and marking interactions in the course. Thus, CAPSI makes possible to thoroughly monitor, analyze, and evaluate a significant portion of the behavior and learning in the course.

In conclusion, CAPSI is a powerful technological educational innovation. Also, CAPSI has demonstrated its novelties and improvements over its ancestor PSI. Realizing its full potential, CAPSI is advancing the technological innovation process of education.

The research done and the conclusions drawn from it are important to education and engineering. CAPSI is the product of the collaboration

between engineering concepts and software engineering to improve the educational instructions especially the personalized instruction of both teaching and learning. Since CAPSI is still young and under development, it has many areas to be further improved and developed. The following areas are recommended to do further research and development:

1. To conduct systematic research on the effectiveness of CAPSI [PeKi87].
2. To formally study of the modelling, parameter estimation, and optimization of the dynamical educational process from the research data obtained in courses [KiPe88a].
3. To study how to enhance the students' learning and long-term retention of the material in a given course; such studies could utilize concepts from control and automata theory, including fuzzy and probabilistic automata, recommended by Kinsner and Pear [KiPe88a].
4. To allow stand-alone implementations of CAPSI [KiPe88a].
5. To exploit all aspects of computer communications technology such as on-line databases, bulletin board systems, and computer conferencing systems [KiPe88a].
6. To develop an authorizing system ([NATA81], [PeKi87]) for generating course material, study objectives, and test questions.
7. To create a knowledge base using the recent approach of knowledge representation and knowledge engineering for intelligent tutors ([Wool87], [PeKi87]) to assist in marking tests.
8. To develop a simple natural language translator to allow international use of CAPSI in projects such as the Commonwealth University and CIDA educational activities.

REFERENCES

[Bena84a] Benaim, M. (1984a). A model for the evaluation of instructional methods. *IEEE Trans. Educ.*, E-27 (2) 105–108.

[Bena84b] Benaim, M. (1984b). P.S.I. versus traditional method criteria for comparison and results (1969–1979). *IEEE Trans. Educ.*, E-27 (1) 41–46.

[BlBu76] Block, J. H. & Burns, R. B. (1976). Mastery learning In L.S. Shulman (Ed.), *Review of research in education* tasca (Ill.): Peacock, 4.

[FHBT76] Friedman, C. P., Hirschi, S., Parlett, M., and Taylor, E. F. (1976). The rise and fall of physics at M.I.T. *American Journal of Physics* 44(3) 204–211.

[Gree71] Green, B. A. (1971). Physics teaching by the Keller Plan at M.I.T. *American Jorunal of Physics* 39(7) 71–81.

[HeKP84a] Herzog, F., Kinsner, W., and Pear, J. J. (1984a). Personalized System of Instruction: User manual. *Technical Report*, Industrial Applications of Microelectronics Centre and Department of Electrical Engineering, University of Manitoba, Winnipeg, Manitoba, 36 pp.

[HeKP84b] Herzog, F., Kinsner, W., and Pear, J. J. (1984b). Personalized System of Instruction: System reference manual. *Technical Report*, Industrial Applications of Microelectronics Centre and Department of Electrical Engineering, University of Manitoba, Winnipeg, Manitoba, 163 pp.

[Hilt86] Hiltz, S. R. (1986). The "virtual classroom": Using computer-mediated communication for university teaching. *Journal of Communication*. 36, 2, 95–104.

[HoUl79] Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Reading (MA): Addison-Wesley, 418 pp.

[Hurs76] Hursh, D. E. (1976). Personalized system of instruction: What do the data indicate? *Journal of Personalized Instruction*, **1**, 91–105.

[JoRu77] Johnson, K. R., & Ruskin, R. S. (1977) *Behavioral instruction: An evaluative review.* Washington, D.C.: American Psychological Association.

[Kell68] Keller, F. S. (1968). "Good–bye, teacher ... ." *Journal of Applied Behavior Analysis*, **1**, 79–89.

[KiPe88a] Kinsner, W. and Pear, J.J. (1988a). Computer–aided personalized system of instruction for the virtual classroom. *Can. J. Educational Communic.*, **17**(1), 21–36.

[KiPe88b] Kinsner, W. and Pear, J.J. (1988b). Computer–aided personalized system of instruction: A method incorportating engineering concepts for higher education in engineering. *Proc. Can. Conf. Engineering Education*, Winnipeg, Manitoba, Canada; May 16–17.

[KiPe88c] Kinsner, W. and Pear, J.J. (1988c). Dynamical Educational System for the Virtual Campus. *Technological Innovation and Human Resources*, **2** (in review).

[KiPL88] Kinsner, W., Pear, J.J., and Leung, Y.M. (1988). Community involvement in higher education through CAPSI. *Proc. Soc. Study Higher Education*, Windsor, Ontario, Canada; June 2–5 (in preparation).

[KuKC79] Kulik, J. A., Kulik, C. C., & Cohen, P. A. (1979). Meta–analysis of outcome studies of Keller's personalized system of instruction. *American Psychologist*, **34**, 307–318.

[KuKS76] Kulik, J. A., Kulik, C.–L. C., & Smith, B. B. (1976). Research on the personalized system of instruction. *Journal of Programmed Learning and Educational Technology*, **13**, 23–30.

[NATA81] A National Authoring Language. (1981). Honeywell Information Systems and National Research Council.

[Page80] Page–Jones, M. (1980). *The Practical Guide to Structured Systems Design*. Englewood Cliffs (NY): Yourdon Press (Prentice Hall), 354 pp.

[PeKi87] Pear, J.J. and Kinsner, W. (1987). Computer–Aided Personalized System of Instruction: An effective and economical method for short– and long–distance education. *Machine–Medicated Learning*.

[Robi76] Robin, A. R. (1976). Behavioral instruction in the college classroom. *Review of Educational Research*, **46**, 313–354.

[Ryan74] Ryan, B. (1974). P.S.I. Keller's personalized system of instruction: An appraisal. *Amer. Psychol. Ass.*.

[ShRS82a] Sherman, J. G., Ruskin, R. S. & Semb, G. B. (eds.) (1982). *The personalized system of instruction: 48 seminal papers*. Lawrence (KS): TRI Publications.

[ShRS82b] Sherman J. G. (1982b). "PSI today," in Sherman, J. G., Ruskin, R. S. & Semb, G. B. (eds.) (1982). *The personalized system of instruction: 48 seminal papers*. Lawrence (KS): TRI Publications.

[SiHa80] Silver, H. and Hanson, J. (1980). *The learning preference inventory: User's manual*. Moorestown (NJ): Hanson Silver and Associates.

[Skin53] Skinner, B. F. (1953). *Scoemce and human behavior*. New York: The Macmillan Company.

[Skin54] Skinner, B. F. (1954). The science of learning and the art of teaching. *Harvard Educational Review*, **24**, 86–97.

[Skin61] Skinner, B. F. (1961). Teaching machines. *Scientific American*, **205**, 90–102.

[Tave76] Taveggia, T. (1976). Personalized instruction: A summary of comparative research, 1967–74. *American Journal of Physics*, 44(11) 1028–1033.

[UofM86] Institutional Statistics Book. (1985–86). Office of Institutional Analysis, The University of Manitoba.

[Wool87] Woolf, B. P. (1987). Representing complex knowledge in an intelligent machine tutor. *Computational Intelligence*, **3**, 45–55.

APPENDIX A

SOURCE CODE FOR CAPSI

WITHOUT CLASSROOM SETTING

```
//PSIJOB JOB'CO=2,F=DAI1,T=30,L=10,I=120','PSI_MAIN',MSGLEVEL=(1,1)
//         EXEC ASMGC
//ASM.SYSIN DD *
*
*   CALL BY PL1
*   CALL SCN(BUFFER);
*      DCL BUFFER    CHAR(*) VARYING;
*
DUMREC1  CSECT
         ENTRY SCN
         DC    C'SCN'
         DC    AL1(5)
SCN      DS    0H
         STM   14,11,12(13)
         BALR  10,0
         USING *,10
         LA    4,SAVEAREA
         ST    13,SAVEAREA+4
         ST    4,8(13)
         LA    13,SAVEAREA
         L     4,0(1)                    GET ADDR OF BUFFER
         LA    6,2(4)                    GET ACTUAL BUFFER ADDR
         LH    4,0(4)                    GET LENGTH OF BUFFER
         TPUT  (6),(4),CONTROL
         L     13,4(13)
         LM    14,11,12(13)
         BR    14
SAVEAREA DC    20F'0'
         END
// EXEC PL1OCL,CSIZE=512K,IS=NIS,STG=NSTG,AP='F(1),INT',X=X,A=A
//PL1.SYSIN DD *
/*   PERSONALIZED SYSTEM INSTRUCTION   */
/*****************************************************************/
/*                                                               */
/*              P S I     MAIN  PROGRAM.                         */
/*                                                               */
/*              AUTHOR           FRANK  HERZOG  IAMC.           */
/*              DATE WRITTEN     JAN. 15  1984.                 */
/*              AUTHOR           MANIX  LEUNG, YIU-MAN.         */
/*              LAST UPDATE      FEB. 15  1984.                 */
/*                               OCT. 15  1985                  */
/*                               SEPT. 1  1987                  */
/*                                                               */
/*****************************************************************/
/*                                                               */
/*     VERSION 3.0                                               */
/*     ALLOWS STUDENTS TO CALL PSI FROM THEIR TSO ACCOUNTS       */
/*     VERSION 2.0                                               */
/*     CHANGES FROM SINGLE USER TO MULTIPLE USERS THAT CAN ACCESS */
/*     ON A SHARED DATA BASE CONCURRENTLY.  THE STUDENT DATABASE */
/*     WAS MODIFIED FROM REGIONAL(1) FILE TO INDEX FILE          */
/*     ORGANIZATION. THE FACILITIES OF MONITORING STUDENT        */
/*     ACTIVITIES AND SENDING MESSAGES ARE ADDED.                */
/*                                                               */
/*****************************************************************/
/*                                                               */
/*                                                               */
/*     THIS PROGRAM PERFORMS THE INTERACTIVE EDITING AND STUDENT */
/*     TRANSACTION CONTROL FUNCTIONS OF THE SYSTEM. THREE FILES  */
/*     ARE USED BY THIS PROGRAM. THE FIRST IS THE SYSTEM PARA-   */
/*     METER FILE , A SINGLE RECORD DIRECT ACCESS FILE,IT CONTAINS */
/*     THE VALUES WHICH DETERMINE THE NUMBER OF UNITS IN THE COURSE*/
/*     ..... ETC. THE SECOND IS THE STUDENT RECORD FILE (STUDFIL) */
/*     IT IS ALSO A DIRECT INDEX ACCESS FILE.  THREE RECORDS ARE */
/*     PREALLOCATED TO STORE INFORMATION RELATED TO THE SPECIAL  */
/*     STUDENT NUMBERS EDIT,INST,TA. (ONLY THE PASSWORD FIELD IS */
/*     USED IN THESE RECORDS.). THE THIRD FILE IS A TRANSACTION  */
/*     LOG FILE WHICH KEEPS A RECORD OF STUDENT TRANSACTIONS AS  */
/*     THEY OCCUR, FOR STATISTICAL PURPOSES. THIS PROGRAM IS     */
/*     DIVIDED INTO ROUTINES WHICH ROUGHLY PARALLEL THE MENU     */
/*     OPTIONS. THE FIRST ROUTINE WHICH IS PERFORMED IS AN       */
/*     INITIALIZATION ROUTINE WHICH SETS UP THE DIRECT INDEX FILE */
/*     TO ACCESS STUDENT RECORDS AND ALSO WRITES A LOG RECORD WITH */
/*     A TRANSACTION TYPE OF 2. AFTER INITIALIZATION , THE MAIN  */
/*     CONTROL MENU ROUTINE IS CALLED. IT IN TURN CALLS SUB-ROUTINE*/
/*     WHICH EDIT STUDENT RECORDS , START SESSION . . . ETC.     */
/*     DEPENDING ON THE CHOICES MADE BY THE USER.                */
/*                                                               */
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/*     GLOBAL DATA DICTIONARY                                    */
/*     ------ ---- ----------                                    */
/*                                                               */
/*     LOGFIL         THIS IS THE LOG FILE. FOR A DESCRIPTION SEE */
/*                    THE SECTION IN THE SYSTEM REFERENCE MANUAL */
/*                    ON DATA FILES.                             */
/*     SYSPFIL        THE SYSTEM PARAMETER FILE- DESCRIBED IN THE */
/*                    REFERENCE MANUAL.                          */
/*     STUDFIL        STUDENT RECORD FILE , DESCRIBED IN THE     */
/*                    SYSTEM REFERENCE MANUAL.                   */
/*     TST_ ...       THESE ARE THE DENOTATIONS FOR TEST STATES, */
/*                    SEE THE SYSTEM REFERENCE MANUAL FOR A      */
/*                    MEANING OF THE CODES.                      */
/*     PST_. . .      THESE ARE THE DENOTATIONS FOR PROCTOR STATES*/
/*     CUTOFF         THIS IS THE TEST CUTOFF TIME LIMIT.        */
/*     CUR_I_TEST     THIS IS THE COUNTER CONTAINING THE NUMBER  */
/*                    OF TESTS CURRENTLY ASSIGNED TO THE         */
/*                    INSTRUCTOR OR TA.                          */
/*     I_TEST_LIM     THIS IS THE LIMIT TO THE NUMBER OF TESTS   */
/*                    WHICH CAN BE ASSIGNED TO THE INSTRUCTOR OR */
/*                    TA.                                        */
/*     CURDATE        A VARIABLE CONTAINING THE CURRENT DATE.    */
/*                                                               */
/*****************************************************************/
```

```
MPSI:PROC(PARM) OPTIONS(MAIN);
   DCL PARM                          CHAR(256) VARYING;
   DCL SYSIN                         FILE STREAM INPUT;
   DCL SYSPRINT                      FILE STREAM OUTPUT PRINT
                        ENV(RECSIZE(132));
   DCL LOGFIL                        FILE RECORD EXCLUSIVE KEYED
          UNBUFFERED ENV(F RECSIZE(42) BLKSIZE(42) REGIONAL(1) );
   DCL SYSPFIL                       FILE RECORD EXCLUSIVE KEYED
          UNBUFFERED ENV(F RECSIZE(285) BLKSIZE(285) REGIONAL(1) );
   DCL STUDFIL                       FILE RECORD KEYED
                        ENV(INDEXED KEYLOC(2) KEYLENGTH(7) RECSIZE(107)
                                    BLKSIZE(107) F BUFFERS(35));
   DCL TESTIN                        FILE KEYED INPUT RECORD
                        ENV(VSAM KEYLOC(0) KEYLENGTH(255));
   DCL MAILIN                        FILE KEYED INPUT RECORD
                        ENV(VSAM KEYLOC(0) KEYLENGTH(255));
   DCL MAILOUT                       FILE STREAM OUTPUT
                        ENV(RECSIZE(255));
   DCL COMMOUT                       FILE STREAM OUTPUT
                        ENV(RECSIZE(72));
   DCL INTRDR                        EXTERNAL FILE STREAM OUTPUT
                        ENV(FB RECSIZE(80) BLKSIZE(6080));
   DCL INTFILE                       FILE EXTERNAL STREAM
                        ENV(RECSIZE(72));
   DCL DYNAM                         EXTERNAL OPTIONS(ASM,INTER,RETCODE)
                                     ENTRY;
   DCL SCN                           EXTERNAL OPTIONS(ASM,INTER)
                                     ENTRY(CHAR(*) VARYING);

   DCL  1 STUD ,
          2 DELKEY                   BIT(8),
          2 ID                       CHAR(7) INIT(' '),
          2 NAME                     CHAR(30) INIT(' '),
          2 PHONE                    CHAR(7) INIT(' '),
          2 FACULTY                  CHAR(2) INIT(' '),
          2 YEAR                     CHAR(2) INIT(' '),
          2 STATUS                   CHAR(3) INIT(' '),
          2 UNIT                     FIXED BIN(15) INIT(0),
          2 Q1                       FIXED BIN(15) INIT(0),
          2 Q2                       FIXED BIN(15) INIT(0),
          2 Q3                       FIXED BIN(15) INIT(0),
          2 TEST                     FIXED DEC(7 ,3) INIT(0),
          2 PROCTOR                  FIXED DEC(7 ,3) INIT(0),
          2 TERM                     FIXED DEC(7 ,3) INIT(0),
          2 EXAM                     FIXED DEC(7 ,3) INIT(0),
          2 TOTAL                    FIXED DEC(7 ,3) INIT(0),
          2 LETTER                   CHAR(2) INIT(' '),
          2 PASSWORD                 CHAR(8) INIT(' '),
          2 PSTATE                   FIXED BIN(15) INIT(0),
          2 TSTATE                   FIXED BIN(15) INIT(0),
          2 SID                      CHAR(7) INIT(' '),
          2 ATIME                    PICTURE '999999' ;
   DCL  1 TSTUD  LIKE STUD;

   DCL  1 SAVE  LIKE STUD;
   DCL  1 SYSP ,
          2 DELKEY                   BIT(8),
          2 MUNIT                    FIXED BIN(15),
          2 UNITL(99)                FIXED BIN(15),
          2 VPASS                    FIXED DEC(7,3),
          2 VPROC                    FIXED DEC(7,3),
          2 LGTHRESH(13)             FIXED DEC(7,3),
          2 SESSDATE                 PICTURE '999999',
          2 SESSTIME                 PICTURE '999999',
          2 SESSCUTOFF               PICTURE '999999',
          2 CUR_I_TEST               FIXED BIN(15),
          2 ASSIGN_TA                FIXED BIN(15),
          2 NLOGREC                  FIXED BIN(15);
   DCL  1 SYST LIKE SYSP;
   DCL 1 LOGREC,
          2 DELKEY                   BIT(8),
          2 TYPE                     CHAR(1),
          2 ID                       CHAR(7),
          2 TS                       FIXED BIN(15),
          2 PS                       FIXED BIN(15),
          2 UNIT                     FIXED BIN(15),
          2 Q1                       FIXED BIN(15),
          2 Q2                       FIXED BIN(15),
          2 Q3                       FIXED BIN(15),
          2 TP                       FIXED DEC(7,3),
          2 PP                       FIXED DEC(7,3),
          2 CTIME                    PICTURE '999999',
          2 SID                      CHAR(7);
   DCL 1 WORK,
          2 WA_LEN                   FIXED BIN(31) INIT(2000),
          2 FILLER                   CHAR(2000);
   DCL (LSTUD,LSTUDMORE)             FIXED BIN(15) INIT(100);
   DCL 1 PINDX(LSTUD)                CONTROLLED,
          2 ID                       CHAR(7),
          2 PROCTOR                  FIXED DEC(7, 3);
   DCL STEND                         FIXED BIN(15);
   DCL (LST,LSTMORE)                 FIXED BIN(15) INIT(200);
   DCL 1 ST(LST)                     CONTROLLED,
          2 ID                       CHAR(7),
          2 NAME                     CHAR(30);
   DCL L_TST                         FIXED BIN(15) INIT(11);
   DCL OUTSTANDING(L_TST)            BIT(1) INIT(
              '0'B,'1'B,'0'B,'1'B,'1'B,'1'B,'0'B,'1'B,'1'B,'1'B,'1'B);
   DCL L_SPEC                        FIXED BIN(15) INIT(5);
   DCL SPEC_ID(L_SPEC)               CHAR(7) INIT(
       'INST','TA','EDIT','ALL','MARKER');
   DCL L_PST                         FIXED BIN(15) INIT(4);
   DCL PSC(L_PST)                    CHAR(13) INIT(
       'INITIAL','AVAILABLE','NOT AVAILABLE','PROCTORING');
   DCL TSC(L_TST)                    CHAR(11) INIT(
       'INITIAL','WRITING','NOT WRITING','NO MARK YET','ONE PASS',
       'ONE FAIL','RESTUDY','0-MARK W/ C','1-PASS /W C','1-FAIL /W C',
```

```
    'O-MRK W/ 2C');
DCL LGLIT(13)                   CHAR(2) INIT('A+','A ','A-',
'B+','B ','B-','C+','C ','C-','D+','D ','D-','F ');
DCL TAS                         FIXED BIN(15) INIT(0);
DCL TA(50)                      CHAR(7) VARYING;
DCL (DATE,TIME,LENGTH,SUBSTR,VERIFY,ABS,MOD,HBOUND)   BUILTIN;
DCL (TRANSLATE, INDEX,UNSPEC,ONSOURCE,REPEAT)         BUILTIN;
DCL (BOOL,STRING,PLIRETV)                             BUILTIN;
DCL TST_I                       FIXED BIN(15) INIT(1);
DCL TST_T                       FIXED BIN(15) INIT(2);
DCL TST_NT                      FIXED BIN(15) INIT(3);
DCL TST_R0                      FIXED BIN(15) INIT(4);
DCL TST_R1                      FIXED BIN(15) INIT(5);
DCL TST_R2                      FIXED BIN(15) INIT(6);
DCL TST_R3                      FIXED BIN(15) INIT(7);
DCL TST_CR0                     FIXED BIN(15) INIT(8);
DCL TST_CR1                     FIXED BIN(15) INIT(9);
DCL TST_CR2                     FIXED BIN(15) INIT(10);
DCL TST_CR3                     FIXED BIN(15) INIT(11);
DCL PST_I                       FIXED BIN(15) INIT(1);
DCL PST_PA                      FIXED BIN(15) INIT(2);
DCL PST_PNA                     FIXED BIN(15) INIT(3);
DCL PST_P                       FIXED BIN(15) INIT(4);
DCL I_TEST_LIM                  FIXED BIN(15) INIT(99);
DCL TRUE                        BIT(1) INIT('1'B);
DCL FALSE                       BIT(1) INIT('0'B);
DCL YN(2)                       CHAR(1) INIT('Y','N');
DCL CQ(2)                       CHAR(1) INIT('C','Q');
DCL RESP                        FIXED BIN(15);
DCL PROF                        CHAR(8) VARYING INIT('PEAR');
DCL PREFIX                      CHAR(8) VARYING;
DCL USERIDS                     CHAR(256) VARYING;
DCL (RECORD,RECORD2)            CHAR(510) VARYING;
DCL LINE                        CHAR(255) VARYING;
DCL RSTYTME                     PICTURE '999999' INIT(100);
DCL TSTWITHIN                   PICTURE '999999' INIT(100);
DCL REDIRHR                     PICTURE '999999' INIT(10000);
DCL CURTIME                     PICTURE '999999';
DCL CUTOFF                      PICTURE '999999';
DCL CURDATE                     CHAR(6);
DCL OUTDATE                     CHAR(8);
DCL UPPERCASE                   CHAR(26) INIT(
'ABCDEFGHIJKLMNOPQRSTUVWXYZ' );
DCL LOWERCASE                   CHAR(26) INIT(
'abcdefghijklmnopqrstuvwxyz' );
DCL (BEL,BS,OFF,ON)             CHAR(1);
 /************ MAIN PROGRAM *************************/
 CALL INIT;
 CALL SESSION_CONTROL;

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

```
 /*     THIS PROCEDURE IS THE LOWEST LEVEL TERMINAL INPUT */
 /*     ROUTINE. THE PARAMETER CIN IS FILLED WITH L       */
 /*     CHARACTERS FROM THE KEYBOARD WHEN THIS ROUTINE IS */
 /*     CALLED. ALL LOWERCASE ALPHABETIC CHARACTERS ARE   */
 /*     TRANSLATED TO UPPERCASE BY THIS ROUTINE.          */
 /*                                                       */
 /*                                                       */
 /*********************************************************/
 TGET:PROC(CIN,L,PROTECT);
    DCL CIN                     CHAR(*);
    DCL L                       FIXED BIN(15);
    DCL PROTECT                 BIT(1);
    DCL T50                     CHAR(50);
    DCL BS52                    CHAR(52);
    DCL C60                     CHAR(60);
    DCL (I,J)                   FIXED BIN(15);
    ON ENDFILE(SYSIN) BEGIN;
       T50=' ';
       CLOSE FILE(SYSIN);
       OPEN FILE(SYSIN);
       PUT FILE(SYSPRINT) SKIP;
       END;
    IF (PROTECT) THEN DO;
       BS52=REPEAT(BS,52);
       C60=REPEAT('**&&@',12);
       J=MOD(RAND,5)+1;
       DO I=1 TO 2;
          PUT FILE(SYSPRINT) EDIT(SUBSTR(C60,I+J,L),
             SUBSTR(BS52,1,L)) (A,A);
       END;
       PUT FILE(SYSPRINT) EDIT(' ',SUBSTR(BS52,1,L+2),': ')
          (A(L),A,A);
       CALL SCN(STRING(OFF));
       END;
    GET FILE(SYSIN) EDIT(T50) (A(50));
    IF (PROTECT) THEN
       CALL SCN(STRING(ON));
    CIN=SUBSTR(T50,1,L);
    CIN=TRANSLATE(CIN,UPPERCASE,LOWERCASE);
 END; /* TGET */

 /*********************************************************/
 /*                                                       */
 /*   THIS PROCEDURE PRINTS THE VALUE OF A PASSWORD       */
 /*   AND ERASES IT FROM THE SCREEN.                      */
 /*                                                       */
 /*********************************************************/
 ECHOPASS:PROC(CIN);
    DCL CIN                          CHAR(*);
    DCL BS8                          CHAR(8);
    BS8=REPEAT(BS,8);
    CALL TPUT('PASSWORD: "'||CIN||'"',1);
    DELAY(1000);
```

```
   PUT FILE(SYSPRINT) EDIT(BS,BS8,'*X*OWWM1',BS8,'OOWWMMX*',BS8,
      'WWM1**@O',BS8,'M1*X@@WW',BS8,'          ')(A);
END; /* ECHOPASS */

/*****************************************************************/
/*                                                               */
/*      THIS PROCEDURE RETURNS A BOOLEAN VALUE TRUE IF           */
/*      THE STUDENT NUMBER PASSED IN PARAMETER CID               */
/*      MATCHES ANY OF THE SPECIAL STUDENT NUMBERS               */
/*      RECOGNIZED BY THE SYSTEM. (IE, INST,EDIT,TA,ALL,         */
/*      MARKER).                                                 */
/*                                                               */
/*****************************************************************/
SPECIAL_ID:PROC(CID) RETURNS(BIT(1));
   DCL CID                    CHAR(7);
   DCL I                      FIXED BIN(15);
   DCL SPEC                   BIT(1);
   SPEC = FALSE;
   DO I=1 TO L_SPEC;
      IF (SPEC_ID(I)=CID) THEN SPEC=TRUE;
   END;
   RETURN(SPEC);
END; /* SPECIAL ID */

/*****************************************************************/
/*                                                               */
/*    THIS PROCEDURE TAKES THE CHARACTERS PASSED IN COUT AND     */
/*    WRITES THEM TO THE TERMINAL AFTER SKIPPING L LINES.        */
/*                                                               */
/*****************************************************************/
TPUT:PROC(COUT,L);
   DCL L                      FIXED BIN(15);
   DCL COUT                   CHAR(*);
   SELECT(L);
   WHEN(1) PUT FILE(SYSPRINT) EDIT(COUT) (COL(1), A);
   WHEN(0) PUT FILE(SYSPRINT) SKIP(L) EDIT(COUT) (COL(1), A);
   OTHERWISE PUT FILE(SYSPRINT) EDIT(COUT) (COL(1), SKIP(L-1), A);
   END;
END; /* TPUT */

/*****************************************************************/
/*                                                               */
/*      THE RAND PROCEDURE GENERATES A RANDOM 32 BIT INTEGER     */
/*      THE PSUEDORANDOM TECHNIQUE USED IS A COMBINATION OF      */
/*      LINEAR CONGRUENTIAL AND SHIFT REGISTER TECHNIQUES.       */
/*                                                               */
/*****************************************************************/
(NOFIXEDOVERFLOW):(NOZERODIVIDE):
RAND:PROC RETURNS(FIXED BIN(31));
   DCL SEED                   FIXED BIN(31) INIT(17) STATIC;
   DCL (T1,T2)                FIXED BIN(31);
   SEED=69069*SEED*TIME;
   UNSPEC(T1)=( (15)'0'B )||SUBSTR(UNSPEC(SEED),1,17);
   T1=T1+SEED;
   T2=(T1*131072)+T1;
   SEED=SEED+T2;
   RETURN(SEED);
END; /* RAND */

/*****************************************************************/
/*                                                               */
/*      THE GENRAND PROCEDURE GENERATES A RANDOM INTEGER IN      */
/*      RANGE 1-LIM WHICH EXCLUDES THE NUMBERS IN Q1,Q2,Q3.      */
/*      IT IS USED TO GENERATE THE QUESTION NUMBERS FOR TESTS    */
/*      AND IN THE PROCTOR SELECTION ALGORITHM WHERE IT IS       */
/*      USED TO SELECT PROCTORS IF THERE ARE MORE THAN ONE       */
/*      WITH THE LOWEST NUMBER OF POINTS.                        */
/*                                                               */
/*****************************************************************/
GENRAND:PROC(LIM,Q1,Q2,Q3) RETURNS(FIXED BIN(15));
   DCL (LIM,Q1,Q2,Q3)         FIXED BIN(15);
   DCL RGEN                   FIXED BIN(15);
   DCL EQ                     BIT(1);
   IF (LIM<2) THEN RETURN(1);
   RGEN=MOD(RAND,LIM)+1;
   IF (LIM=2) THEN RETURN(RGEN);
   EQ=TRUE;
   DO WHILE(EQ);
     IF ((RGEN=Q1)|(RGEN=Q2)) THEN DO;
        IF (RGEN<LIM) THEN
           RGEN=RGEN+1;
        ELSE
           RGEN=1;
        END;
     ELSE EQ=FALSE;
   END;
   RETURN(RGEN);
END; /* GENRAND */

/*****************************************************************/
/*                                                               */
/*   THIS PROCEDURE WRITES OUT THE MESSAGE THAT THE STUDENT      */
/*   NUMBER HAS JUST BEEN MODIFIED OR DELETED.                   */
/*                                                               */
/*****************************************************************/
REC_NO_FOUND:PROC;
   CALL TPUT('ERROR IS FOUND IN ' || STUD.ID, 1);
   CALL TPUT('YOUR USERID HAS JUST BEEN MODIFIED OR', 1);
   CALL TPUT('YOUR STUDENT RECORD HAS JUST BEEN DELETED.', 1);
   CALL TPUT('TRANSACTION CANCELLED.', 1);
   CALL TPUT('PLEASE CONTACT TO YOUR INSTRUCTOR !', 1);
END; /* REC_NO_FOUND */

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

```
/*      THIS PROCEDURE READS ONE STUDENT RECORD GIVEN THE    */
/*      STUDENT NUMBER IN PARAMETER CID. THE PARAMETER FOUND */
/*      IS RETURNED AS A BOOLEAN TRUE VALUE IF THE RECORD    */
/*      CORRESSPONDING TO STUDENT NUMBER CID WAS FOUND.      */
/*                                                           */
/*************************************************************/
READ_STUD:PROC(CID,FOUND);
   DCL CID                  CHAR(7);
   DCL FOUND                BIT(1);
   ON KEY(STUDFIL) FOUND=FALSE;
   FOUND=TRUE;
   READ NOLOCK FILE(STUDFIL) INTO(STUD) KEY(CID);
END; /* READ_STUD */

/*************************************************************/
/*                                                           */
/*   THIS PROCEDURE READS AND WRITES ONE STUDENT RECORD      */
/*   GIVEN THE STUDENT NUMBER IN STUD.ID. THE PARAMETER      */
/*   FOUND IS RETURNED AS A BOOLEAN TRUE VALUE IF THE RECORD */
/*   CORRESSPONDING TO STUDENT NUMBER CID WAS FOUND.         */
/*                                                           */
/*************************************************************/
READ_STUD_LOCK:PROC(FOUND);
   DCL FOUND                BIT(1);
   ON KEY(STUDFIL) BEGIN;
      FOUND=FALSE;
      CALL REC_NO_FOUND;
      END;
   FOUND=TRUE;
   READ FILE(STUDFIL) INTO(STUD) KEY(STUD.ID);
END; /* READ_STUD_LOCK */

/*************************************************************/
/*                                                           */
/*      THIS PROCEDURE REWRITES THE STUDENT RECORD ASSOCI-   */
/*      ATED WITH STUDENT NUMBER PASSED IN CID. FOUND IS THE*/
/*      BOOLEAN VALUE RETURNED TRUE IF THE RECORD WAS FOUND.*/
/*                                                           */
/*************************************************************/
UPDATE_STUD:PROC(CID,FOUND);
   DCL CID                  CHAR(7);
   DCL FOUND                BIT(1);
   ON KEY(STUDFIL) FOUND=FALSE;
   FOUND = TRUE;
   REWRITE FILE(STUDFIL) FROM(STUD) KEY(CID);
END;/* UPDATE_STUD */

/*************************************************************/
/*                                                           */
/*     THIS ROUTINE CLOSES THE FILE OF THE SYSTEM.           */
/*     IF SOME RECORDS ARE LOCKED BY ANOTHER TASK, THEN      */
/*     THE ERROR CONDITION WILL BE RAISED. IF SO, DELAY A    */
/*     WHILE AND TRY TO CLOSE THE FILE AGAIN.                */
/*                                                           */
/*************************************************************/
CLOSE_FILE:PROC(INFILE);
   DCL INFILE               FILE VARIABLE;
   ON ERROR BEGIN;
      DELAY(10);
      GOTO AGAIN;
      END;
AGAIN: CLOSE FILE(INFILE);
END; /* CLOSE_FILE */

/*************************************************************/
/*                                                           */
/*     THIS ROUTINE ACCEPTS A 6 CHARACTER VALUE REPRESENTING */
/*     A TIME AS HHMMSS AND RETURNS THE SAME TIME VALUE WITH */
/*     THE HOURS,MINUTES SECONDS SEPARATED BY COLONS.HH:MM:SS*/
/*                                                           */
/*************************************************************/
SEPARATE:PROC(TIMEARG) RETURNS(CHAR(8));
   DCL TIMEARG              PICTURE '999999';
   RETURN(SUBSTR(TIMEARG,1,2)||':'||SUBSTR(TIMEARG,3,2)||
          ':'||SUBSTR(TIMEARG,5,2));
END; /* SEPARATE */

/*************************************************************/
/*                                                           */
/*  THIS ROUTINE TAKES AN INTEGER AND RETURNS                */
/*  THE DIGITS ONLY.                                         */
/*                                                           */
/*************************************************************/
COMPRESS:PROC(NUM) RETURNS(CHAR(15) VARYING);
   DCL NUM FIXED BIN(15);
   DCL STRNUM CHAR(15) VARYING;
   STRNUM=NUM;
   RETURN(SUBSTR(STRNUM,VERIFY(STRNUM,' ')));
END;

/*************************************************************/
/*                                                           */
/*  THIS ROUTINE TAKES TWO 6 DIGIT DAY-TIME VALUES*/
/*  REPRESENT IN DDHHMM AND RETURNS A VALUE WHICH */
/*  IS THE SUM OF THE DAYS, HOURS AND MINUTES.    */
/*                                                           */
/*************************************************************/
ADDTD:PROC(TD1,TD2) RETURNS(PICTURE'999999');
   DCL (TD1,TD2)            PICTURE '999999';
   DCL (I,CARRY)            FIXED BIN(15);
   DCL (DD,MM,HH)           PICTURE '99';
   I=SUBSTR(TD1,5,2)+SUBSTR(TD2,5,2);
   MM=MOD(I,60);
   CARRY=I/60;
   I=SUBSTR(TD1,3,2)+SUBSTR(TD2,3,2)+CARRY;
```

```
      HH=MOD(I,24);
      CARRY=I/24;
      I=SUBSTR(TD1,1,2)+SUBSTR(TD2,1,2)+CARRY;
      DD=I;
      RETURN(DD||HH||MM);
   END; /* ADDTD */

   /*****************************************************/
   /*                                                   */
   /*  THIS ROUTINE TAKES TWO 6 DIGIT DAY-TIME VALUES*/
   /*  REPRESENT IN DDHHMM AND RETURNS A VALUE,         */
   /*  THE SUBTRACTION OF TD2 FROM TD1 WHICH IS         */
   /*  ALWAYS GREATER THAN TD2.                         */
   /*                                                   */
   /*****************************************************/
   SUBTD:PROC(TD1,TD2) RETURNS(PICTURE'999999');
      DCL (TD1,TD2)          PICTURE '999999';
      DCL (I,CARRY)          FIXED BIN(15);
      DCL (DD,MM,HH)         PICTURE '99';
      I=SUBSTR(TD1,5,2)-SUBSTR(TD2,5,2);
      IF (I<0) THEN DO;
         CARRY=-1;
         I=I+60;
         END;
      ELSE
         CARRY=0;
      MM=I;
      I=CARRY+SUBSTR(TD1,3,2)-SUBSTR(TD2,3,2);
      IF (I<0) THEN DO;
         CARRY=-1;
         I=I+24;
         END;
      ELSE
         CARRY=0;
      HH=I;
      I=CARRY+SUBSTR(TD1,1,2)-SUBSTR(TD2,1,2);
      DD=I;
      RETURN(DD||HH||MM);
   END; /* SUBTD */

/*****************************************************************/
/*                                                               */
/*     THIS PROCEDURE BUILDS AND WRITES A LOG RECORD.THE CID     */
/*     AND TRANS PARAMETERS DEFINE THE SID AND TYPE FIELDS TO    */
/*     BE WRITTEN WHILE THE REST OF THE FIELDS OF THE LOG        */
/*     ARE GLEANED FROM THE CURRENT VALUES IN THE STUDENT        */
/*     RECORD.                                                   */
/*     FOR A DESCRIPTION OF THE LOG RECORD FIELDS SEE THE REF-   */
/*     ERANCE MANUAL.                                            */
/*                                                               */
/*****************************************************************/
LOG:PROC(TRANS,CID);
   DCL TRANS                 CHAR(1);
   DCL CID                   CHAR(7);
   ON KEY(LOGFIL) BEGIN;
      CALL READ_SYSP_LOCK;
      SYSP.NLOGREC=SYSP.NLOGREC-1;
      CALL UPDATE_SYSP;
      CALL TPUT('PLEASE CONTACT TO YOUR INSTRUCTOR !',1);
      CALL TPUT('THE LOG FILE IS FULL AND NO RECORD CAN BE '||
         'WRITTEN ON IT.',1);
      CALL TPUT('THE LOG FILE IS NEEDED TO BE ENLARGED !',1);
      GOTO OUT;
      END;
   CALL READ_SYSP_LOCK;
   SYSP.NLOGREC=SYSP.NLOGREC+1;
   CALL UPDATE_SYSP;
   READ FILE(LOGFIL) INTO(LOGREC) KEY(SYSP.NLOGREC);
   LOGREC.DELKEY=(8)'0'B;
   LOGREC.TYPE=TRANS;
   LOGREC.CTIME=SUBSTR(TIME,1,6);
   LOGREC.SID=CID;
   IF (TRANS='2'|TRANS='3'|TRANS='4'|TRANS='5') THEN DO;
      LOGREC.TS=TST_I;
      LOGREC.PS=PST_I;
      LOGREC.ID=' ';
      LOGREC.UNIT,LOGREC.Q1,LOGREC.Q2,LOGREC.Q3,LOGREC.TP,
         LOGREC.PP=0;
      END;
   ELSE DO;
      LOGREC.TS=STUD.TSTATE;
      LOGREC.PS=STUD.PSTATE;
      LOGREC.ID=STUD.ID;
      LOGREC.UNIT=STUD.UNIT;
      LOGREC.Q1=STUD.Q1;
      LOGREC.Q2=STUD.Q2;
      LOGREC.Q3=STUD.Q3;
      LOGREC.TP=STUD.TEST;
      LOGREC.PP=STUD.PROCTOR;
      END;
   REWRITE FILE(LOGFIL) FROM(LOGREC) KEY(SYSP.NLOGREC);
OUT:;
END; /* LOG */

/*****************************************************************/
/*                                                               */
/*   THIS PROCEDURE IS CALLED WHEN THERE IS A SERIOUS            */
/*   INCONSISTANCY IN THE INTERNAL RECORDS. AN ERROR NUMBER      */
/*   DEFINED IN THE TABLE BELOW IS WRITTEN TO THE TERMINAL.      */
/*                                                               */
/*   ERROR   CAUSE                          LOCATION             */
/*   -----   -----                          --------             */
/*   1       STUDENT NUMBER IN ST ARRAY     SESSION CONTROL,     */
/*           NOT FOUND IN STUDENT FILE.     PROCTOR SELECTION.   */
/*   2       INVALID CASE CHOICE.           SESSION CONTROL,     */
```

```
/*                                                PROCTOR SELECTION   */
/*   3       SECONDARY ID NOT FOUND.              SESSION CONTROL,    */
/*                                                MARK TEST           */
/*   4       STUDENT WRITING TEST AND             SESSION CONTROL     */
/*           PROCTORING AT THE SAME TIME                              */
/*   5       -SAME AS ERROR 2.                    SESSION CONTROL MAIN*/
/*                                                SELECT STATMENT.    */
/*   6       -STUDENT NUMBER NOT FOUND            -STUDENT LOGIN.     */
/*   7       -SAME AS ERROR 6.                    -GET_PROC_STATUS    */
/*   8       -SAME AS ERROR 2.                    -GET_PROC_STATUS    */
/*   9       -SAME AS ERROR 2.                    -GET_MARK           */
/*   10      -SPECIAL STUDENT NUMBER NOT          -GET_EDIT_PASS      */
/*            FOUND.                                                  */
/*   11      -SAME AS ERROR 1.                    -START_SESSION.     */
/*   12      -SAME AS ERROR 1.                    -END SESSION.       */
/*   13      -SAME AS ERROR 6.                    -EDIT_PERSONAL      */
/*   14      -SAME AS ERROR 1.                    -EDIT_PERSONAL      */
/*   15      -SAME AS ERROR 10.                   -EDIT_PERSONAL      */
/*   16      -SAME AS ERROR 10.                   -GET_INST_PASS      */
/*   17      -SAME AS ERROR 6.                    -EDIT_COURSE        */
/*   18      -SAME AS ERROR 1.                    -LIST_STUD          */
/*   19      -SAME AS ERROR 6.                    -SESSION CONTROL,TEST*/
/*                                                 GENERATE.          */
/*   20      -SAME AS ERROR 6.                    -STATE_MACH         */
/*   21      -SAME AS ERROR 5.                    -STATE_MACH         */
/*   22      -SAME AS ERROR 1.                    -REMOV_STUD         */
/*   23      -SAME AS ERROR 1.                    -EDIT_SYSP,CHANGE   */
/*                                                 THRESHOLDS.        */
/*   24      -SAME AS ERROR 6.                    -GENTEST            */
/*   25      -SAME AS ERROR 6.                    -GET_MARK           */
/*                                                                    */
/*********************************************************************/
INTERNAL_ERR:PROC(ENUM);
  DCL ENUM                              FIXED BIN(15);
  DCL COUT                              CHAR(24);
  PUT STRING(COUT) EDIT('***INTERNAL ERROR:',ENUM)(A,F(6,0));
  CALL TPUT(COUT,1);
END; /* INTERNAL_ERR */

 /*******************************************************************/
 /*                                                                 */
 /*    THIS ROUTINE CONTAINS THE COMMON CODE REQUIRED TO            */
 /*    WRITE A PROMPT TO THE TERMINAL, GET A SINGLE CHARACTER*/
 /*    RESPONSE BACK AND MATCH TO A KNOWN SET OF RESPONSES   */
 /*    THE PARAMETER MES CONTAINS THE PROMPT TO BE LISTED.   */
 /*    THE PARAMETER RA IS AN ARRAY OF SINGLE CHARACTER VALID*/
 /*    RESPONSES. RETVAL IS THE RETURN VALUE: AN INDEX INTO  */
 /*    THE RA ARRAY INDICATING WHICH ONE OF THE RESPONSES WAS*/
 /*    TYPED IN. RETVAL IS SET TO 0 IF A BLANK OR RETURN WAS */
 /*    TYPED IN. THE PROGRAM WILL LOOP REQUESTING A RESPONSE */
 /*    UNTIL A VALID ONE IS TYPED IN.                               */
 /*                                                                 */
 /*******************************************************************/
```

```
GET_RESP:PROC(MES,RA,RETVAL,DEFAULT);
   DCL MES                          CHAR(*);
   DCL RA(*)                        CHAR(1);
   DCL (RETVAL,DEFAULT,I,LIM)       FIXED BIN(15);
   DCL CONT                         BIT(1);
   DCL TEMP                         CHAR(1);
   CONT = TRUE;
   RETVAL = 0 ;
   LIM = HBOUND(RA,1);
   DO WHILE(CONT);
      TEMP=' ';
      IF (DEFAULT^=0) THEN
         CALL TPUT(MES||RA(DEFAULT)||BS||BS||': ',1);
      ELSE
         CALL TPUT(MES,1);
      CALL TGET(TEMP,1,FALSE);
      DO I=1 TO LIM;
         IF (RA(I)=TEMP) THEN DO;
            RETVAL=I;
            CONT=FALSE;
            END;
      END;
      IF (TEMP=' ') THEN DO;
         RETVAL=0;
         CONT=FALSE;
         END;
      ELSE DO;
         IF (CONT) THEN DO;
            CALL TPUT('INVALID RESPONSE, TRY AGAIN.',1);
            END;
         END;
   END; /* WHILE */
END; /*  GET RESPONSE */

/*******************************************************************/
/*                                                                 */
/*     THIS PROCEDURE IS EXECUTED BEFORE THE CONTROL MENU          */
/*     IS DISPLAYED. IT PRINTS THE SIGN-ON BANNER AND PRIMES */
/*     THE CURDATE. THE PROCEDURE WRITES A LOG RECORD WITH   */
/*     TRANSACTION TYPE=3                                          */
/*                                                                 */
/*******************************************************************/
INIT:PROC;
   DCL J                            FIXED BIN(15);
   CURDATE= DATE;
   OUTDATE= SUBSTR(CURDATE,1,2)||'/'||SUBSTR(CURDATE,3,2)||
            '/'||SUBSTR(CURDATE,5,2);
   CURTIME=SUBSTR(TIME,1,6);
   CALL TPUT('PERSONALIZED SYSTEM INSTRUCTION.',1);
   CALL TPUT('VERSION 3.0 : SEPT 1,1987',1);
   CALL TPUT(OUTDATE||'    '||SEPARATE(CURTIME),1);
   UNSPEC(BEL)='00101111'B;
```

```
   UNSPEC(OFF)='00001110'B;
   UNSPEC(ON) ='00001111'B;
   UNSPEC(BS) ='00010110'B;
   UNSPEC(WORK.WA_LEN)=BOOL('1'B,UNSPEC(WORK.WA_LEN),'0111'B);
   FETCH DYNAM;
   CALL DYNAM(WORK,'ALLOC ','DD=INTFILE;','PREF='||
     PREFIX||';','DSN=INTFILE VOL=WORK F DSO=PS '||
     'LRECL=72 BLKSIZE=72 TRK PRIM=1 SEC=1 '||
     'UNIT=DISK NEW DEL CDEL;');
   OPEN FILE(STUDFIL) EXCLUSIVE UPDATE DIRECT UNBUFFERED,
      FILE(SYSPFIL) UPDATE DIRECT,
      FILE(LOGFIL) UPDATE DIRECT,
      FILE(SYSIN),
      FILE(SYSPRINT) LINESIZE(132),
      FILE(TESTIN),
      FILE(MAILIN),
      FILE(INTFILE) OUTPUT LINESIZE(72);
   ALLOC PINDX,ST;
   IF (PARM=' ')|(INDEX(PARM,' ')=0) THEN DO;
      CALL TPUT('INVALID PARAMETERS.',1);
      CALL SCN(STRING(BEL));
      STOP;
      END;
   J=INDEX(PARM,' ');
   USERIDS=SUBSTR(PARM,1,J-1);
   PARM=SUBSTR(PARM,J+1);
   J=INDEX(PARM||' ',' ');
   PREFIX=SUBSTR(PARM,1,J-1);
   DO WHILE(USERIDS^='');
      J=INDEX(USERIDS||',',',');
      TA(TAS+1)=SUBSTR(USERIDS,1,J-1);
      IF (J+1>LENGTH(USERIDS)) THEN
         USERIDS='';
      ELSE
         USERIDS=SUBSTR(USERIDS,J+1);
      IF (TA(TAS+1)^=' ') THEN
         TAS=TAS+1;
   END;
   CALL READ_SYSP;
   IF SYSP.SESSDATE^=CURDATE THEN DO;
      CALL LOG('3',CURDATE||' ');
      SYSP.SESSDATE=CURDATE;
      CALL UPDATE_SYSP;
   END;
END; /* INIT */

/*****************************************************************/
/*                                                               */
/*     THIS PROCEDURE REWRITES THE SYSTEM PARAMETER RECORD       */
/*     IT IS CALLED AFTER EVERY MODIFICATION TO A SYSTEM         */
/*     PARAMETER.                                                */
/*                                                               */
/*****************************************************************/

UPDATE_SYSP:PROC;
   REWRITE FILE(SYSPFIL) FROM(SYSP) KEY(0) ;
END; /* UPDATE_SYSP */

/*****************************************************************/
/*                                                               */
/*    THIS PROCEDURE IS USED TO READ THE SYSTEM PARAMETER        */
/*    RECORD.                                                    */
/*                                                               */
/*****************************************************************/
READ_SYSP:PROC;
   READ NOLOCK FILE(SYSPFIL) INTO(SYSP) KEY(0) ;
END; /* READ_SYSP */

/*****************************************************************/
/*                                                               */
/*    THIS PROCEDURE IS USED TO READ AND LOCK THE SYSTEM         */
/*    PARAMETER RECORD.                                          */
/*                                                               */
/*****************************************************************/
READ_SYSP_LOCK:PROC;
   READ FILE(SYSPFIL) INTO(SYSP) KEY(0) ;
END; /* READ_SYSP_LOCK */
   /*************************************************************/
   /*                                                           */
   /*       THIS PROCEDURE CONTAINS THE COMMON CODE REQUIRED    */
   /*       TO PROMPT THE TERMINAL FOR A STUDENT NUMBER AND     */
   /*       THEN RECIEVE A REPLY. PARAMETER STNPRMPT IS THE     */
   /*       INPUT STRING THE ROUTINE WILL WRITE TO THE TER-     */
   /*       MINAL  AS A PROMPT. CID CONTAINS THE STUDENT        */
   /*       NUMBER WHICH IS TYPED IN. FOUND,SPEC,BLANKF ARE     */
   /*       BOOLEAN VALUES RETURNED WHICH INDICATE IF THE       */
   /*       STUDENT NUMBER TYPED IN WAS FOUND IN THE RECORDS    */
   /*       ,WAS A SPECIAL STUDENT NUMBER (LIKE EDIT,INST ..    */
   /*       .. .ETC. )OR WAS TYPED IN AS A BARE RETURN( ALL     */
   /*       BLANK).                                             */
   /*                                                           */
   /*************************************************************/
   GETID:PROC(CID,FOUND,SPEC,BLANKF,STNPRMPT,DEFAULT);
      DCL STNPRMPT                CHAR(*);
      DCL CID                     CHAR(7);
      DCL (FOUND,SPEC,BLANKF,DEFAULT) BIT(1);
      DCL I                       FIXED BIN(15);
      FOUND=FALSE;
      SPEC=FALSE;
      BLANKF=FALSE;
      IF (DEFAULT=TRUE)&(PREFIX=PROF) THEN
         DEFAULT=FALSE;
      IF (DEFAULT=TRUE) THEN
         DO I=1 TO TAS WHILE(DEFAULT);
            IF (PREFIX=TA(TAS)) THEN
               DEFAULT=FALSE;
         END;
```

```
   IF (DEFAULT=TRUE) THEN DO;
      CALL TPUT(STNPRMPT||PREFIX,2);
      CALL TPUT(STNPRMPT,0);
      CALL TGET(CID,7,FALSE);
      IF (CID=' ') THEN
         CID=PREFIX;
      END;
   ELSE DO;
      CALL TPUT(STNPRMPT,2);
      CALL TGET(CID,7,FALSE);
      END;
   IF (CID=' ') THEN DO;
      BLANKF=TRUE;
      RETURN;
      END;
   IF (SPECIAL_ID(CID)) THEN DO;
      SPEC=TRUE;
      RETURN;
      END;
   FOUND=TRUE;
   CALL READ_STUD(CID,FOUND);
END; /* GET ID */

/*****************************************************************/
/*                                                               */
/*        THIS PROCEDURE CONTAINS THE COMMON CODE                */
/*        REQUIRED TO READ IN AN INTEGER FROM THE                */
/*        TERMINAL. IT RETURNS A BINARY VALUE OF THE             */
/*        CHARACTERS TYPED IN AND STORES IN THE                  */
/*        PARAMETER ITEMP.                                       */
/*        THE BOOLEAN VALUES RETURNED IN ERR AND BLANKF          */
/*        INDICATE WHETHER THE VALUE TYPED IN COULD BE           */
/*        CONVERTED CORRECTLY AND IF IT WAS TYPED IN AS          */
/*        BARE RETURN(ALL BLANKS) OR NOT                         */
/*                                                               */
/*****************************************************************/
GET_INT:PROC(ITEMP,ERR,BLANKF);
   DCL ITEMP                  FIXED BIN(15);
   DCL (ERR,BLANKF)           BIT(1);
   DCL TEMPIN                 CHAR(7);
   ON CONVERSION BEGIN;
      ERR = TRUE;
      TEMPIN= '0';
      ITEMP=0;
      ONSOURCE='0';
      END;
   ON SIZE   ERR=TRUE;
   ERR=FALSE;
   BLANKF=FALSE;
   ITEMP=0;
   CALL TGET(TEMPIN,7,FALSE);
   IF (TEMPIN=' ') THEN DO;
         BLANKF=TRUE;
         RETURN;
         END;
(SIZE): ITEMP=TEMPIN;
   END; /* GET INT*/

   /*****************************************************************/
   /*                                                               */
   /*     THIS PROCEDURE LOOKS AT THE TEST,PROCTOR,TERM AND         */
   /*     EXAM POINT FIELDS OF THE CURRENT STUDENT RECORD           */
   /*     FORMING A SUM AND PLACING IT IN THE TOTAL POINTS          */
   /*     FIELD. AFTER THAT A NEW LETTER GRADE IS CALCULATED        */
   /*     AND ASSIGNED DEPENDING ON THE TOTAL POINT VALUE.          */
   /*     THIS PROCEDURE IS CALLED EVERY TIME THE MARKS OF A        */
   /*     STUDENT IS CHANGED OR THE LETTER GRADES THRESHOLDS        */
   /*     ARE CHANGED.                                              */
   /*                                                               */
   /*****************************************************************/
   (NOSIZE):TOTAL_MARKS:PROC;
      DCL I                        FIXED BIN(15);
      CALL READ_SYSP;
      STUD.TOTAL=STUD.TEST+STUD.PROCTOR+STUD.TERM+STUD.EXAM;
      STUD.LETTER=LGLIT(13);
      DO I=13 TO 1 BY -1;
         IF (STUD.TOTAL>=SYSP.LGTHRESH(I)) THEN
            STUD.LETTER=LGLIT(I);
      END;
   END; /* TOTAL MARKS */

   /*****************************************************************/
   /*                                                               */
   /*   THIS IS A SUBROUTINE OF LISTING OUT THE STUDENT             */
   /*   PROCTORS OF THE INPUT STUDENT ID. IF NO STUDENT             */
   /*   PROCTORS FOR HIM IS ASSIGNED, THE STUD_PROC WILL            */
   /*   RETURNS A FALSE. PRINT IS TO TURN ON OR OFF                 */
   /*   THE LISTING OF THE STUDENT PROCTOR.                         */
   /*                                                               */
   /*****************************************************************/
      STUD_PROCTORS:PROC(CID,DIRECT,PRINT,STUD_PROC);
         DCL CID                   CHAR(7);
         DCL (DIRECT,PRINT,STUD_PROC) BIT(1);
         STUD_PROC=FALSE;
         ON ENDFILE(STUDFIL) GOTO EOF;
         CALL CLOSE_FILE(STUDFIL);
         OPEN FILE(STUDFIL) INPUT SEQUENTIAL BUFFERED;
         DO WHILE('1'B);
            READ FILE(STUDFIL) INTO(STUD);
            IF ((STUD.PSTATE=PST_P)&(STUD.SID=CID)) THEN DO;
               IF (PRINT) THEN
                  CALL TPUT('PROCTOR HAS NOT ENTERED RESULTS,
                     ||'USERID: '||STUD.ID||', NAME: '||
                     STUD.NAME,1);
               STUD_PROC=TRUE;
```

```
            END;
         END;
   EOF:  CALL CLOSE_FILE(STUDFIL);
         IF (DIRECT) THEN
            OPEN FILE(STUDFIL) EXCL UPDATE DIRECT UNBUFFERED;
         ELSE DO;
            OPEN FILE(STUDFIL) INPUT SEQUENTIAL BUFFERED;
            READ FILE(STUDFIL) INTO(STUD) KEY(CID);
            END;
      END; /* STUD_PROCTORS */

/*****************************************************************/
/*                                                               */
/*    THIS PROCEDURE LISTS ALL THE UNMARKED STUDENTS             */
/*    WHO WAITED OVER 24 HOURS.                                  */
/*                                                               */
/*****************************************************************/
      REDIRECT:PROC;
         DCL HEADR1                        CHAR(38) INIT(
            'STUD.#  NAME');
         DCL HEADR2                        CHAR(48) INIT(
         'UNIT# QUESTIONS TEST STATE  PROCTOR STATE   TIME');
         DCL CID                           CHAR(7);
         DCL (NOW,ADDED)                   PICTURE '999999';
         CALL TPUT('LISTING OF ALL THE UNMARKED '||
            'STUDENTS WHO PASSED OVER 24 HOURS.',1);
         CALL TPUT(HEADR1||HEADR2,1);
         CALL CLOSE_FILE(STUDFIL);
         OPEN FILE(STUDFIL) INPUT SEQUENTIAL BUF;
         ON ENDFILE(STUDFIL) GOTO EOF;
         ON ATTENTION BEGIN;
            ON ATTENTION SYSTEM;
            CALL GET_RESP('(C)ONTINUE OR (Q)UIT  : ',
               CQ, RESP,1);
            IF (RESP=2) THEN GOTO EOF;
                        ELSE GOTO CONT;
            END;
         DO WHILE('1'B);
   CONT: READ FILE(STUDFIL) INTO(STUD) KEYTO(CID);
            IF (^SPECIAL_ID(STUD.ID)&
               (OUTSTANDING(STUD.TSTATE))) THEN DO;
               ADDED=ADDTD(STUD.ATIME,REDIRHR);
               NOW=SUBSTR(DATE,5,2)||SUBSTR(TIME,1,4);
               IF ((NOW<ADDED)&(SUBTD(ADDED,NOW)>REDIRHR)) THEN
                  NOW=(SUBSTR(NOW,1,2)-1+SUBSTR(ADDED,1,2))||
                     SUBSTR(NOW,3,4);
               IF (NOW>ADDED) THEN
                  CALL ONE_STATUS_OUT(CID,FALSE);
            END;
         END;
   EOF:  ON ATTENTION SYSTEM;
         CALL CLOSE_FILE(STUDFIL);
         OPEN FILE(STUDFIL) EXCL UPDATE DIRECT UNBUF;
      END; /* REDIRECT */

/*****************************************************************/
/*                                                               */
/*   THIS IS A SUBROUTINE OF THE LIST STUDENT STATUS             */
/*   IT CONTAINS THE CODE WHICH FORMATS ONE LINE OF              */
/*   OUTPUT.                                                     */
/*                                                               */
/*****************************************************************/
      ONE_STATUS_OUT:PROC(CID,DIRECT);
         DCL CID                           CHAR(7);
         DCL DIRECT                        BIT(1);
         DCL STUD_PROC                     BIT(1);
         DCL OUT_UNIT                      PICTURE 'ZZ9';
         DCL (OUT_Q1,OUT_Q2,OUT_Q3) PICTURE 'ZZZ9';
         DCL OUT1                          CHAR(38);
         DCL OUT2                          CHAR(49);
         OUT_Q1=ABS(STUD.Q1);
         OUT_Q2=ABS(STUD.Q2);
         OUT_Q3=ABS(STUD.Q3);
         OUT1=STUD.ID||' '||STUD.NAME;
         OUT2=OUT_Q1||OUT_Q2||OUT_Q3||' '||TSC(STUD.TSTATE)||
            ' '||PSC(STUD.PSTATE)||' '||SUBSTR(STUD.ATIME,1,2)||
            '/'||SUBSTR(STUD.ATIME,3,2)||
            ':'||SUBSTR(STUD.ATIME,5,2);
         IF (OUTSTANDING(STUD.TSTATE)) THEN DO;
            OUT_UNIT=STUD.UNIT+1;
            CALL TPUT(OUT1||OUT_UNIT||OUT2,1);
            IF (STUD.TSTATE^=TST_T) THEN DO;
               CALL STUD_PROCTORS(CID,DIRECT,TRUE,STUD_PROC);
               IF (^STUD_PROC) THEN
                  CALL TPUT('PROCTOR SELECTED IS INSTRUCTOR'||
                     ' OR TA.',1);
               END;
            ELSE DO;
               CALL TPUT('STUDENT IS WRITING A TEST.',1);
               END;
            END;
         ELSE DO;
            OUT_UNIT=STUD.UNIT;
            CALL TPUT(OUT1||OUT_UNIT||OUT2,1);
            CALL TPUT('STUDENT HAS NO UNMARKED TEST.',1);
            END;
         CALL TPUT(' ',1);
      END; /* ONE_STATUS_OUT */

/*****************************************************************/
/*                                                               */
/*      THIS IS A SUBROUTINE THAT PROMPTS                        */
/*      THE INSTRUCTOR OR TA FOR THE                             */
/*      NEXT STUDENT NUMBER TO BE MARKED (3 WRONG                */
/*      TRIALS OR A BARE RETURN CAUSES IT TO TERMINATE*/
```

```
/*      ): THEN PROMPTS HIM FOR A MARK IN THE SAME    */
/*      MANNER STUDENT PROCTORS ARE PROMPTED.         */
/*                                                    */
/******************************************************/
    MARK_STUD:PROC;
  ooo  confidential procedure  ooo
    END;  /* MARK_STUD */

     /*****************************************************/
     /*                                                   */
     /*     THIS PROCEDURE IS THE MAIN ROUTINE WHICH      */
     /*     HANDLES STUDENT TRANSACTION PROCESSING.       */
     /*     THIS ROUTINE LOOPS SEVERAL TIMES CALLING      */
     /*     THE STUD_LOGIN SUBROUTINE TO LOGIN THE        */
     /*     STUDENT. NEXT AFTER SIGNING ON THE            */
     /*     NEXT STUDENT THIS ROUTINE LOOKS AT HIS        */
     /*     INTERNAL STATE TO DETERMINE WHICH PROMPTS     */
     /*     ARE PRINTED. THE ARRAY CASE_TBL IS USED TO    */
     /*     DETERMINE WHICH OF THE THREE ALTERNATIVES     */
     /*     OF THE SELECT STATEMENT INTERNAL TO THIS      */
     /*     ROUTINE IS CHOSEN. ONE ALTERNATIVE IS THAT    */
     /*     THE STUDENT HAS NO UNMARKED TEST AND HE       */
     /*     SHOULD BE PROMPTED TO GENERATE THE NEXT ONE   */
     /*     THE NEXT ALTERNATIVE IS TAKEN IF HE HAS       */
     /*     GENERATED A TEST BUT NOT SELECTED PROCTORS.   */
     /*     THE THIRD ALTERNITIVE IS THAT HE HAS BEEN     */
     /*     SELECTED TO PROCTOR SOMEONE ELSES TEST AND    */
     /*     HE MUST NOW ENTER IN THE MARK. THE NULPROC    */
     /*     VARIABLE IS SET TO TRUE IF THE CURRENT        */
     /*     STUDENT HAS BEEN SELECTED AS A PROCTOR BUT    */
     /*     THE TEST HE HAS BEEN ASKED TO PROCTER HAS     */
     /*     ALREADY BEEN MARKED BY THE INSTRUCTOR OR      */
     /*     HAS BEEN CANCELLED. THIS CAUSE THE STUDENT    */
     /*     LOGIN PROCEDURE TO BE SKIPPED THE NEXT TIME   */
     /*     THROUGH THE LOOP AND ALLOWS THE CURRENTLY     */
     /*     LOGGED IN STUDENT TO PERFORM ANOTHER  TYPE    */
     /*     OF TRANSACTION.                               */
     /*                                                   */
     /*****************************************************/
     SESSION_CONTROL:PROC;
        DCL (BLANKF,FOUND,NOTMARKER)        BIT(1);
        DCL CASE_TBL(L_PST,L_TST)           CHAR(1) INIT(
            'T','P','T','P','P','P','T','P','P','P','P',
            'T','P','T','P','P','P','T','P','P','P','P',
            'T','P','T','P','P','P','T','P','P','P','P',
            'M','E','M','M','M','M','M','M','M','M','M');
        DCL CID                             CHAR(7);
        DCL (VALID,PROCEDE,NULL_PROC,NONE) BIT(1);
        NULL_PROC=FALSE;
REPEAT:
        NOTMARKER=TRUE;
        IF (^NULL_PROC) THEN DO;
           CALL STUD_LOGIN(CID,VALID);
           IF ^VALID THEN RETURN;
           IF (CID='INST') THEN DO;
              CALL REDIRECT;
              CALL MARK_STUD;
              NOTMARKER=FALSE;
              END;
           ELSE IF ((CID='TA') | (CID='MARKER')) THEN DO;
              CALL MARK_STUD;
              NOTMARKER=FALSE;
              END;
           ELSE DO;
              CALL PRINT_INFO(FOUND);
              IF (^FOUND) THEN GOTO REPEAT;
              END;
           END;
        IF NOTMARKER THEN DO;
           IF (STUD.PSTATE^=PST_P & ^NULL_PROC) THEN DO;
              CALL GET_PROC_STATUS(FOUND);
              IF (^FOUND) THEN GOTO REPEAT;
              END;
           NULL_PROC=FALSE;
           CALL READ_STUD(STUD.ID,FOUND);
           IF (^FOUND) THEN DO;
              CALL REC_NO_FOUND;
              GOTO REPEAT;
              END;
           SELECT(CASE_TBL(STUD.PSTATE,STUD.TSTATE));
           WHEN('T') CALL ASK_TEST; /*TEST ASK*/
           WHEN('P') CALL SELECT_PROC; /*PROCTOR SELECT */
           WHEN('M') CALL MARK; /*MARK TEST*/
           OTHERWISE CALL INTERNAL_ERR(5); /*ERROR */
           END; /*SELECT*/
           IF (^NULL_PROC) THEN DO;
              CALL TPUT('OK   TRANSACTION COMPLETE.',1);
              CALL TPUT('ENDING AT: '||
                 SEPARATE(SUBSTR(TIME,1,6)),1);
              END;
           ELSE
              GOTO REPEAT;
           END; /* IF */
/* LOGICAL END OF SESSION CONTROL */

        /*************************************/
        /*                                   */
        /*   ASKS FOR TEST GENERATION        */
        /*                                   */
        /*************************************/
        ASK_TEST:PROC;
           DCL TMSG                         CHAR(22) INIT(
           'GENERATE TEST ON UNIT ');
           DCL TMSG2                        CHAR(19) INIT(
           ' (Y)ES,(N)O?     : ');
```

```
            DCL IOUT                        PICTURE 'ZZZ9';
            DCL YN(2)                       CHAR(1) INIT('Y','N');
            DCL TRESP                       FIXED BIN(15);
            PROCEDE = TRUE;
            CALL READ_SYSP;
            IF (STUD.TSTATE=TST_R3) THEN
               CALL RESTUDY_CHECK(PROCEDE);
            IF (PROCEDE) THEN DO;
               IOUT=STUD.UNIT+1;
               CALL GET_RESP(TMSG||IOUT||TMSG2,YN,TRESP,2);
               IF (TRESP=1) THEN DO; /*GEN TEST*/
                  CALL READ_STUD(STUD.ID,FOUND);
                  IF (^FOUND) THEN DO;
                     CALL REC_NO_FOUND;
                     RETURN;
                     END;
                  IF (CASE_TBL(STUD.PSTATE,STUD.TSTATE)^='T')
                     THEN DO;
                     NULL_PROC=TRUE;
                     RETURN;
                     END;
                  CALL GENTEST(FOUND);
                  END;
               END;
         END; /* ASK_TEST */

         /*****************************************/
         /*                                       */
         /*   REQUIRES FOR PROCTORS               */
         /*                                       */
         /*****************************************/
         SELECT_PROC:PROC;
            DCL (PID1,PID2)             CHAR(7);
            DCL TA_NAME                 CHAR(7) VARYING;
            DCL JOBID                   CHAR(8) VARYING;
            DCL WHERE                   CHAR(100) VARYING;
            DCL VALID                   BIT(1);
            DCL CP(2)                   CHAR(1) INIT('C','P');
            DCL PMSG1                   CHAR(56) INIT(
   'DO YOU WANT YOUR TEST (C)ANCELLED OR (P)ROCTORED?     : ');
            DCL PMSG2                   CHAR(60) INIT(
'YOU MUST HAVE YOUR CURRENT TEST PROCTORED BEFORE PROCEEDING.');
            DCL IPROCMSG                CHAR(37) INIT(
            'PROCTOR SELECTED IS INSTRUCTOR OR TA.');
            DCL PS1A                    CHAR(32) INIT(
            'FIRST PROCTOR SELECTED, USERID: ');
            DCL PS2A                    CHAR(33) INIT(
            'SECOND PROCTOR SELECTED, USERID: ');
            DCL PRESP                   FIXED BIN(15);
            DCL (NOW,ADDED)             PICTURE '(6)9';
            DCL CONDITIONAL(L_TST)      BIT(1) INIT(
          '0'B,'0'B,'0'B,'0'B,'0'B,'0'B,'0'B,'1'B,'1'B,'1'B,'1'B);
```

```
IF (STUD.TSTATE=TST_T) THEN DO;
   ADDED=ADDTD(STUD.RTIME,TSTWITHIN);
   NOW=SUBSTR(DATE,5,2)||SUBSTR(TIME,1,4);
   IF (NOW<ADDED)&(SUBTD(ADDED,NOW)>TSTWITHIN) THEN
      NOW=(SUBSTR(NOW,1,2)-1+SUBSTR(ADDED,1,2))||
         SUBSTR(NOW,3,4);
   IF (NOW>=ADDED) THEN DO;
      CALL TPUT('YOUR TEST IS AUTOMATICALLY '||
         'CANCELLED.',1);
      CALL TPUT('YOU HAVE PASSED 1 HOUR OF '||
         'TEST PERIOD.',1);
      CALL TPUT('YOU HAVE TO WAIT FOR 1 HOUR OF'||
         ' RESTUDYING TIME FROM THE TIME YOUR '||
         'TEST EXPIRED.',1);
      CALL STATE_MACH('E',CID);
      NULL_PROC=TRUE;
      RETURN;
   END;
   CALL GET_RESP(PMSG1,CP,PRESP,0);
   CALL READ_STUD(STUD.ID,FOUND);
   IF (^FOUND) THEN DO;
      CALL REC_NO_FOUND;
      RETURN;
      END;
   IF (^(CASE_TBL(STUD.PSTATE,STUD.TSTATE)='P' &
       (STUD.TSTATE=TST_T)))
      THEN DO;
      NULL_PROC=TRUE;
      RETURN;
      END;
   IF (PRESP=1) THEN DO;
      CALL STATE_MACH('C',CID);
      END;
   IF (PRESP=2) THEN DO;
      CALL VALID_IN_ANSWERED(VALID);
      IF ^VALID THEN RETURN;
      CALL FIND_PROCTORS(PID1,PID2,NONE);
      CALL READ_STUD_LOCK(FOUND);
      IF (^FOUND) THEN RETURN;
      IF (^(CASE_TBL(STUD.PSTATE,STUD.TSTATE)='P' &
          (STUD.TSTATE=TST_T)))
         THEN DO;
         NULL_PROC=TRUE;
         UNLOCK FILE(STUDFIL) KEY(STUD.ID);
         RETURN;
         END;
      IF (NONE) THEN DO;
         CALL TPUT('ALL AVAILABLE PROCTORS'
          ||' ARE BUSY. TRY AGAIN LATER.',1);
         UNLOCK FILE(STUDFIL) KEY(STUD.ID);
         END;
      ELSE DO;
         CALL STATE_MACH('I',CID);
```

```
JOBID=STUD.ID;
WHERE=PROF;
IF (PID1='INST') THEN DO;
   CALL TPUT(IPROCMSG,1);
   TA_NAME=TA(SYSP.ASSIGN_TA+1);
   CALL TPUT('TA: '||TA_NAME,1);
   IF TA_NAME^=PROF THEN
      WHERE=WHERE||', '||TA_NAME;
   END;
ELSE DO;
  STUD.ID=PID1;
  CALL READ_STUD_LOCK(FOUND);
  IF (^FOUND) THEN
     CALL INTERNAL_ERR(1);
  NOW=SUBSTR(DATE,5,2)||SUBSTR(TIME,1,4);
  IF ((NOW<STUD.RTIME)&
     (SUBTD(STUD.RTIME,NOW)>ASTYTME)) THEN
     NOW=(SUBSTR(NOW,1,2)-1+
        SUBSTR(STUD.RTIME,1,2))||
        SUBSTR(NOW,3,4);
  IF ((STUD.PSTATE=PST_PA)&
     (STUD.TSTATE^=TST_T)&
     (^((STUD.TSTATE=TST_R3)&
     (NOW<STUD.RTIME))))
     THEN DO;
     CALL STATE_MACH('P',CID);
     CALL TPUT(PS1A||PID1,1);
     WHERE=WHERE||', '||SUBSTR(PID1,1,
        INDEX(PID1||' ',' ')-1);
     STUD.ID=PID2;
     CALL READ_STUD_LOCK(FOUND);
     IF (^FOUND) THEN
        CALL INTERNAL_ERR(1);
     IF ((STUD.PSTATE=PST_PA)&
        (STUD.TSTATE^=TST_T)&
        (^((STUD.TSTATE=TST_R3)&
        (NOW<STUD.RTIME))))
        THEN DO;
        CALL TPUT(PS2A||PID2,1);
        WHERE=WHERE||', '||SUBSTR(PID2,1,
           INDEX(PID2||' ',' ')-1);
        CALL STATE_MACH('P',CID);
        END;
     ELSE DO;
        UNLOCK FILE(STUDFIL) KEY(PID2);
        CALL READ_STUD(PID1,FOUND);
        CALL STATE_MACH('N',PID1);
        CALL TPUT('USERID: '||
           PID1||' IS CANCELLED '||
           'TO BE A PROCTOR.',1);
        WHERE=PROF;
        CALL READ_SYSP_LOCK;
        IF SYSP.CUR_I_TEST<I_TEST_LIM
           THEN DO;
           CALL TPUT(IPROCMSG,1);
           SYSP.CUR_I_TEST=
              SYSP.CUR_I_TEST+1;
           SYSP.ASSIGN_TA=
              MOD(SYSP.ASSIGN_TA+1,TAS);
           TA_NAME=TA(SYSP.ASSIGN_TA+1);
           CALL TPUT('TA: '||TA_NAME,1);
           IF TA_NAME^=PROF THEN
              WHERE=WHERE||', '||TA_NAME;
           END;
        ELSE DO;
           CALL READ_STUD(CID,FOUND);
           CALL STATE_MACH('T',' ');
           CALL TPUT('ALL AVAILABE'||
              ' PROCTORS ARE '||
              'BUSY. TRY AGAIN LATER.',1);
           WHERE='';
           END;
        CALL UPDATE_SYSP;
        END;
     END;
  ELSE DO;
     UNLOCK FILE(STUDFIL) KEY(PID1);
     WHERE=PROF;
     CALL READ_SYSP_LOCK;
     IF (SYSP.CUR_I_TEST<I_TEST_LIM)
        THEN DO;
        CALL TPUT(IPROCMSG,1);
        SYSP.CUR_I_TEST=SYSP.CUR_I_TEST+1;
        SYSP.ASSIGN_TA=
           MOD(SYSP.ASSIGN_TA+1,TAS);
        TA_NAME=TA(SYSP.ASSIGN_TA+1);
        CALL TPUT('TA: '||TA_NAME,1);
        IF TA_NAME^=PROF THEN
          WHERE=WHERE||', '||TA_NAME;
        END;
     ELSE DO;
        CALL READ_STUD(CID,FOUND);
        CALL STATE_MACH('T',' ');
        CALL TPUT('ALL AVAILABLE'||
           ' PROCTORS ARE '||
           'BUSY. TRY AGAIN LATER.',1);
        WHERE='';
        END;
     CALL UPDATE_SYSP;
     END;
  END; /*ELSE*/
END; /*ELSE*/
IF WHERE^='' THEN
   CALL MAIL_OUT(WHERE,JOBID,PREFIX,
      'ANSWERED');
```

```
            END; /*THEN*/
          END;
        ELSE DO; /* PROCTORS ALREADY SELECTED*/
          IF (CONDITIONAL(STUD.TSTATE)) THEN DO;
            CALL VALID_IN_CONDITIONAL(VALID,JOBID,CID,
              WHERE);
            IF ^VALID THEN
              RETURN;
            CALL STATE_MACH('A',STRING(WHERE));
            IF (WHERE^=PROF) THEN
              WHERE=PROF||', '||WHERE;
            CALL MAIL_OUT(WHERE,JOBID,PREFIX,'REVISED');
            END;
          ELSE
            CALL TPUT(PMSG2,1);
          END;
      END; /* SELECT_PROC */

      /*****************************************/
      /*                                       */
      /*   A PROCTOR MARKS A STUDENT           */
      /*                                       */
      /*****************************************/
      MARK:PROC;
        DCL RESP                  FIXED BIN(15);
        DCL SELMSG                CHAR(53) INIT(
'YOU HAVE BEEN SELECTED TO PROCTOR A TEST FOR USERID: ');
        DCL NULMSG1               CHAR(59) INIT(
 'A TEST YOU HAVE BEEN SELECTED TO PROCTOR HAS BEEN CANCELLED' );
        DCL NULMSG2               CHAR(43) INIT(
      'BY THE STUDENT OR MARKED BY THE INSTRUCTOR.');
        DCL MARKSTATE(L_TST)      BIT(1) INIT(
       '0'B,'0'B,'0'B,'1'B,'1'B,'1'B,'0'B,'1'B,'1'B,'1'B,'1'B);
        IF (STUD.SID^=' ') THEN DO;
          SAVE=STUD;
          CALL READ_STUD(STUD.SID,FOUND);
          IF (^FOUND) THEN DO;
              CALL INTERNAL_ERR(3);
              END;
          CALL TPUT(SELMSG||SAVE.SID,1);
          IF (MARKSTATE(STUD.TSTATE)) THEN DO;
          /*  CALL GET_QUES_NUM(BLANKF) */
            BLANKF=FALSE;
            CALL GET_MARK(BLANKF,SAVE.ID,FOUND,RESP,
              MARKSTATE);
            IF (^FOUND) THEN DO;
              STUD=SAVE;
              CALL TPUT(NULMSG1,1);
              CALL TPUT(NULMSG2,1);
              NULL_PROC=TRUE;
              CALL STATE_MACH('N',CID);
              RETURN;
              END;
            IF (^(BLANKF|(RESP=2))) THEN DO;
              STUD=SAVE;
              CALL STATE_MACH('M',CID);
              CALL GET_PROC_STATUS(FOUND);
              IF (^FOUND) THEN RETURN;
              END;
            END;
          ELSE DO;
            STUD=SAVE;
            CALL TPUT(NULMSG1,1);
            CALL TPUT(NULMSG2,1);
            NULL_PROC=TRUE;
            CALL STATE_MACH('N',CID);
            END;
          END;
        ELSE DO;
          CALL TPUT(NULMSG1,1);
          CALL TPUT(NULMSG2,1);
          NULL_PROC=TRUE;
          CALL STATE_MACH('N',CID);
          END;
      END; /* MARK */

      /*************************************************/
      /*                                               */
      /*    THIS PROCEDURE PROMPTS THE TERMINAL FOR    */
      /*    A STUDENT ID AND A PASSWORD AND            */
      /*    RETURNS ONCE A VALID STUDENT NUMBER ,      */
      /*    PASSWORD PAIR HAS BEEN ENTERED. THE        */
      /*    STUDENT NUMBER OF THE JUST LOGGED IN       */
      /*    STUDENT IS RETURN IN THE PARAMETER CID.    */
      /*    PARAMETER VALID INDICATES WHETHER THE      */
      /*    STUDENT NUMBER IS VALID OR NOT.            */
      /*                                               */
      /*************************************************/
      STUD_LOGIN:PROC(CID,VALID);
        ***  confidential procedure  ***
      END; /*STUDENT LOGIN*/

      /*************************************************/
      /*                                               */
      /*    THIS PROCEDURE IS CALLED TO PRINT THE      */
      /*    <DO YOU WISH TO VIEW YOUR CURRENT COURSE*/
      /*    STANDING> PROMPT AND IT DISPLAYS THE       */
      /*    DATA IF THE RESPONSE IS AFFIRMATIVE.       */
      /*                                               */
      /*************************************************/
      PRINT_INFO:PROC(FOUND);
        DCL FOUND                 BIT(1);
        DCL DRESP                 FIXED BIN(15);
        DCL YN(2)                 CHAR(1) INIT('Y','N');
        DCL DMSG                  CHAR(60) INIT(
```

```
'WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O     : ');
          DCL COUT1                  CHAR(61);
          DCL COUT2                  CHAR(40);
          DCL COUT3                  CHAR(38);
          CALL GET_RESP(DMSG,YN,DRESP,2);
          CALL READ_STUD(STUD.ID,FOUND);
          IF (^FOUND) THEN DO;
             CALL REC_NO_FOUND;
             RETURN;
             END;
          IF (DRESP=1) THEN DO;
             PUT STRING(COUT1) EDIT('CURRENT UNIT:',
                 STUD.UNIT,', TEST POINTS:',STUD.TEST,
                 ', PROCTOR POINTS:',STUD.PROCTOR)
                 (A,P'ZZ9',2 (A,F(7,2)));
             CALL TPUT(COUT1,1);
             PUT STRING(COUT2) EDIT('TERM PROJECT:',
                STUD.TERM,', FINAL EXAM:',STUD.EXAM)
                (A,F(7,2));
             CALL TPUT(COUT2,1);
             PUT STRING(COUT3) EDIT('TOTAL POINTS:',
                STUD.TOTAL,', LETTER GRADE: ',STUD.LETTER)
                (A,F(7,2),A,A(2));
             CALL TPUT(COUT3,1);
             END;
       END; /* PRINT INFO */

    /*****************************************************/
    /*                                                   */
    /*    THIS ROUTINE WRITES THE <PROCTOR? > PROMPT     */
    /*    TO THE TERMINAL AND ALLOWS THE STUDENT TO      */
    /*    CHANGE HIS PROCTOR AVAILABLITY STATUS.         */
    /*                                                   */
    /*****************************************************/
       GET_PROC_STATUS:PROC(FOUND);
          DCL YN(2)                  CHAR(1) INIT('Y','N');
          DCL MSG                    CHAR(26) INIT(
          'PROCTOR? (Y)ES,(N)O     : ');
          DCL NOTMSG(4)               CHAR(4) VARYING
             INIT(' NOT','',' NOT','');
          DCL RESP                    FIXED BIN(15);
          DCL FOUND                   BIT(1);
          CALL GET_RESP(MSG,YN,RESP,0);
          CALL READ_STUD(STUD.ID,FOUND);
          IF (^FOUND) THEN DO;
             CALL REC_NO_FOUND;
             RETURN;
             END;
          SELECT(RESP);
          WHEN(1)DO; /* CHANGE TO PA */
             IF (STUD.PSTATE^=PST_PA) THEN DO;
                CALL READ_STUD_LOCK(FOUND);
                IF (^FOUND) THEN RETURN;
                STUD.PSTATE= PST_PA;
                CALL UPDATE_STUD(STUD.ID,FOUND);
                IF (^FOUND) THEN
                   CALL INTERNAL_ERR(7);
                END;
             END;
          WHEN(2) DO; /* CHANGE TO PNA */
             IF ((STUD.PSTATE^=PST_PNA)|
                (STUD.PSTATE^=PST_I)) THEN DO;
                CALL READ_STUD_LOCK(FOUND);
                IF (^FOUND) THEN RETURN;
                STUD.PSTATE=PST_PNA;
                CALL UPDATE_STUD(STUD.ID,FOUND);
                IF (^FOUND) THEN
                   CALL INTERNAL_ERR(7);
                END;
             END;
          WHEN(0) DO; /* LEAVE UNALTERED */
             END;
          OTHERWISE DO;
             CALL INTERNAL_ERR(8);
             RETURN;
             END;
          END; /* SELECT*/
          CALL TPUT('USERID: '||STUD.ID||' IS'||
         NOTMSG(STUD.PSTATE)||' AVAILABLE FOR PROCTORING.',1);
       END; /* GET PROC STATUS */

    /*****************************************************/
    /*                                                   */
    /*    THIS PROCEDURE IS CALLED IF A STUDENT IS IN    */
    /*    THE RESTUDY STATE TO CHECK IF HE HAS COM-      */
    /*    PLETED ALL OF HIS RESTUDY TIME. THE BOOLEAN    */
    /*    ARGUMENT PROCEDE IS RETURNED WITH THE VALUE    */
    /*    FALSE IF THE STUDENT HASNT FINNISHED HIS 60    */
    /*    MINUTES OF RESTUDY TIME.                       */
    /*                                                   */
    /*****************************************************/
       RESTUDY_CHECK:PROC(PROCEDE);
          DCL PROCEDE                BIT(1);
          DCL NOW                    PICTURE '(6)9';
          PROCEDE = TRUE;
          NOW=SUBSTR(DATE,5,2)||SUBSTR(TIME,1,4);
          IF ((NOW<STUD.RTIME)&
             (SUBTD(STUD.RTIME,NOW)>RSTYTME)) THEN
             NOW=(SUBSTR(NOW,1,2)-1+
                SUBSTR(STUD.RTIME,1,2))||
                SUBSTR(NOW,3,4);
          IF (STUD.RTIME>NOW) THEN DO;
             PROCEDE = FALSE;
             CALL TPUT('YOU HAVE NOT COMPLETED 1 HOUR '||
             'OF RESTUDYING TIME.',1);
```

```
      END;
  END; /* RESTUDY CHECK */

/*****************************************************/
/*                                                   */
/*    THIS PROCEDURE GENERATES A TEST ON THE NEXT    */
/*    UNIT AND PRINTS THE NUMBER OF QUESTIONS ON     */
/*    THE TERMINAL.                                  */
/*                                                   */
/*****************************************************/
  GENTEST:PROC(FOUND);
     DCL FOUND                  BIT(1);
     DCL (I,J,TEMP,LIM)         FIXED BIN(15);
     DCL QUESTIONS(3)           FIXED BIN(15) INIT((3)0);
     DCL COUT                   CHAR(72);
     CALL READ_SYSP;
     IF (STUD.UNIT >= SYSP.NUNIT) THEN DO;
        CALL TPUT('YOU HAVE COMPLETED ALL THE UNITS.',1);
        END;
     ELSE DO;
        LIM=SYSP.UNITL(STUD.UNIT+1);
        IF (LIM<3) THEN DO;
          CALL TPUT('TEST CANNOT BE GENERATED.',1);
          CALL TPUT('UNIT MUST HAVE AT LEAST 3 CHOICES'
            ,1);
          CALL TPUT('SEE INSTRUCTOR.',1);
          RETURN;
          END;
        DO I=1 TO 3;
           QUESTIONS(I)=GENRAND(LIM,QUESTIONS(1),
              QUESTIONS(2),QUESTIONS(3));
        END;
        DO I=1 TO 2;
           DO J=2 TO 3;
              IF QUESTIONS(I)>QUESTIONS(J) THEN DO;
                 TEMP=QUESTIONS(I);
                 QUESTIONS(I)=QUESTIONS(J);
                 QUESTIONS(J)=TEMP;
                 END;
           END;
        END;
        CALL READ_STUD_LOCK(FOUND);
        IF (^FOUND) THEN RETURN;
        IF (CASE_TBL(STUD.PSTATE,STUD.TSTATE)^='T')
           THEN DO;
           NULL_PROC=TRUE;
           UNLOCK FILE(STUDFIL) KEY(STUD.ID);
           RETURN;
           END;
        STUD.Q1=QUESTIONS(1);
        STUD.Q2=QUESTIONS(2);
        STUD.Q3=QUESTIONS(3);
        STUD.TSTATE=TST_T;
        CALL UPDATE_STUD(STUD.ID,FOUND);
        IF (^FOUND) THEN CALL INTERNAL_ERR(24);
        PUT STRING(COUT) EDIT('TEST GENERATED ON UNIT:'
        ,STUD.UNIT+1,', AT TIME: ',SEPARATE(SUBSTR(
        TIME,1,6)),', QUESTIONS:',STUD.Q1,',',
        STUD.Q2,',',STUD.Q3)
        (A ,F(4,0),A,A,A,3 (F(4,0),A(1)) );
        CALL TPUT(COUT,1);
        CALL STATE_MACH('T',CID);
        CALL MAIL_QUESTIONS(STRING(STUD.ID),'PSI',
           PREFIX);
     END;
  END; /* GENTEST */

/*****************************************************/
/*                                                   */
/*    THIS PROCEDURE VALIDATES THE INPUT ANSWERED    */
/*    PAPER WHETHER IT IS CORRECT SYNTAX AND         */
/*    RIGHT QUESTIONS.                               */
/*                                                   */
/*****************************************************/
VALID_IN_ANSWERED:PROC(VALID);
   DCL VALID                    BIT(1);
   DCL QUESTIONS(3)             FIXED BIN(15);
   DCL I                        FIXED BIN(15);
   DCL REPEAT                   BUILTIN;
   ON KEY(MAILIN) GOTO EOF;
   ON ENDFILE(MAILIN) GOTO EOF;
   VALID=TRUE;
   IF VALID THEN DO;
      READ FILE(MAILIN) INTO(RECORD) KEY('1+');
      READ FILE(MAILIN) INTO(RECORD);
      LINE=SUBSTR(RECORD,256);
      IF (INDEX(LINE,'To:')^=1)|
         (INDEX(LINE,SUBSTR(STUD.ID,1,
         INDEX(STUD.ID||' ',' ')-1))=0) THEN
         VALID=FALSE;
      END;
   IF VALID THEN DO;
      READ FILE(MAILIN) INTO(RECORD);
      LINE=SUBSTR(RECORD,256);
      IF (INDEX(LINE,'From:')^=1)|
         (INDEX(LINE,'PSI')=0) THEN
         VALID=FALSE;
      END;
   IF VALID THEN DO;
      READ FILE(MAILIN) INTO(RECORD);
      LINE=SUBSTR(RECORD,256);
      IF (INDEX(LINE,'Unit '||COMPRESS(STUD.UNIT+1))=0)|
         (INDEX(LINE,COMPRESS(STUD.Q1)||' '||
         COMPRESS(STUD.Q2)||' '||COMPRESS(STUD.Q3))=0)
         THEN
         VALID=FALSE;
```

```
   END;
 IF VALID THEN DO;
   READ FILE(MAILIN);
   READ FILE(MAILIN) INTO(RECORD);
   ON KEY(TESTIN) RECORD2=REPEAT(' ',254)||
      'CONTACT YOUR INSTRUCTOR '||
      'THAT THE FILE "HEADER" WAS NOT FOUND.';
   READ FILE(TESTIN) INTO(RECORD2) KEY('HEADER');
   IF (SUBSTR(RECORD,256)^=SUBSTR(RECORD2,256)) THEN
      VALID=FALSE;
   END;
 IF VALID THEN DO;
   QUESTIONS(1)=STUD.Q1;
   QUESTIONS(2)=STUD.Q2;
   QUESTIONS(3)=STUD.Q3;
   READ FILE(MAILIN) INTO(RECORD);
   ON KEY(TESTIN) RECORD2=REPEAT(' ',254)||
     'CONTACT YOUR INSTRUCTOR THAT UNIT '||
     COMPRESS(STUD.UNIT+1)||' QUESTION '||
     COMPRESS(QUESTIONS(I))||' WAS NOT FOUND.';
   ON ENDFILE(TESTIN) GOTO EOF2;
   DO I=1 TO 3 WHILE(VALID);
      IF (INDEX(SUBSTR(RECORD,256),'Question '||
         COMPRESS(QUESTIONS(I))||':')^=1) THEN
         VALID=FALSE;
      IF VALID THEN DO;
         READ FILE(TESTIN) INTO(RECORD2)
            KEY('TESTS.U'||COMPRESS(STUD.UNIT+1)||'.Q'
            ||COMPRESS(QUESTIONS(I))||' FIRST');
         READ FILE(MAILIN) INTO(RECORD);
         IF (SUBSTR(RECORD,256)^=SUBSTR(RECORD2,256))
            THEN
            VALID=FALSE;
         END;
      DO WHILE(VALID);
          READ FILE(TESTIN) INTO(RECORD2);
          READ FILE(MAILIN) INTO(RECORD);
          IF (SUBSTR(RECORD,256)^=SUBSTR(RECORD2,256))
             THEN
             VALID=FALSE;
      END;
EOF2: IF VALID THEN DO;
         READ FILE(MAILIN) INTO(RECORD);
         IF (INDEX(SUBSTR(RECORD,256),'Answer '||
            COMPRESS(QUESTIONS(I))||':')^=1) THEN
            VALID=FALSE;
         END;
      IF VALID & (I<3) THEN DO;
         DO WHILE(INDEX(SUBSTR(RECORD,256),'Question '
            ||COMPRESS(QUESTIONS(I+1))||':')^=1);
            READ FILE(MAILIN) INTO(RECORD);
         END;
```

```
         END;
      END; /* DO WHILE */
      IF ^VALID THEN DO;
         CALL TPUT('YOUR TEST UNIT '||COMPRESS(STUD.UNIT
            +1)||' QUESTION '||COMPRESS(QUESTIONS(I-1))||
            ' IS DIFFERENT FROM THE SYSTEM''S.',1);
         CALL TPUT('NO PROCTOR SELECTED.',1);
         VALID=FALSE;
         RETURN;
         END;
      END; /* IF */
   IF ^VALID THEN DO;
EOF: CALL TPUT('POINT TO THE ANSWERED PAPER '||
        'BEFORE YOU EXECUTE THE PSI PROGRAM.',1);
      CALL TPUT('NO PROCTOR SELECTED.',1);
      VALID=FALSE;
      END;
   END; /* VALID_IN_ANSWERED */

/*****************************************************/
/*                                                   */
/*    THIS PROCEDURE SCANS THE STUDENT FILE FOR      */
/*    STUDENTS WHO ARE AVAILABLE FOR PROCTORING      */
/*    THE CURRENTLY LOGGED IN STUDENTS AND SELECTS   */
/*    TWO OF THEM (IF POSSIBLE) TO PROCTOR THE       */
/*    STUDENT. THE PARAMETERS PID1,PID2 ARE USED     */
/*    TO RETURN THE STUDENT NUMBERS OF THE TWO       */
/*    PROCTORS AFTER THEY ARE FOUND. (IF STUDENTS    */
/*    ARE NOT AVAILABLE THE INSTRUCTOR WILL BE       */
/*    SELECTED, AND IF MORE THAN I_TEST_LIM TEST     */
/*    CURRENTLY ASSIGNED TO THE INSTRUCTOR THEN      */
/*    THE RETURN ARGUMENT NONE IS SET TO TRUE. THE   */
/*    FOLLOWING ALGORITHM IS USED TO SELECT          */
/*    PROCTORS: THE ARRAY PINDX USED TO KEEP ALL     */
/*    THAT ARE ELIGIBLE TO PROCTOR THE CURRENT       */
/*    STUDENT. ( A STUDENT IS ELIGABLE IF HE HAS     */
/*    PASSED THE UNIT THE CURRENT STUDENT TEST IS    */
/*    IS ON  AND IF HIS PROCTOR STATE IS PROCTOR     */
/*    AVAILABLE AND HE IS NOT WRITING A TEST OR      */
/*    STILL WAITING OUT HIS 60 MINUTES OF RESTUDY    */
/*    TIME.) A STUDENT IS INSERTED INTO PINDX        */
/*    THROUGH A CALL TO INSRTP SUBROUTINE.IF LESS    */
/*    THAN TWO STUDENTS ARE AVAILABLE THE INST-      */
/*    RUCTOR IS SELECTED. OTHERWISE THE STUDENTS     */
/*    WITH THE LOWEST NUMBER OF PROCTOR POINTS IS    */
/*    SELECTED. ( THE NOT_EQ SUBROUTINE IS USED      */
/*    TO DETERMINE THE NUMBER OF STUDENTS WITH THE   */
/*    LOWEST PROCTOR POINTS.) IF MORE THAN ONE       */
/*    STUDENT HAS THE LOWEST PROCTOR POINTS (OR      */
/*    SECOND LOWEST) THEN THE GENRAND SUBROUTINE     */
/*    IS CALLED TO RANDOMLY SELECT FROM AMONG THEM   */
/*                                                   */
/*****************************************************/
```

```
 FIND_PROCTORS:PROC(PID1,PID2,NONE);
    DCL (NONE,AVAIL)              BIT(1);
    DCL (PID1,PID2)               CHAR(7);
    DCL (J,I,PEND,PEND1)          FIXED BIN(15);
    DCL 1 TEMP LIKE PINDX;
    DCL NOW                       PICTURE '(6)9';
    PEND=0;
    ON ENDFILE(STUDFIL) GOTO EOF;
    CALL CLOSE_FILE(STUDFIL);
    OPEN FILE(STUDFIL) INPUT SEQUENTIAL BUFFERED;
    DO WHILE('1'B);
       READ FILE(STUDFIL) INTO(TSTUD);
       NOW=SUBSTR(DATE,5,2)||SUBSTR(TIME,1,4);
       IF ((NOW<TSTUD.RTIME)&
          (SUBTD(TSTUD.RTIME,NOW)>RSTYTME)) THEN
          NOW=(SUBSTR(NOW,1,2)-1+
             SUBSTR(TSTUD.RTIME,1,2))||
             SUBSTR(NOW,3,4);
       IF ((TSTUD.PSTATE=PST_PA)&(TSTUD.TSTATE^=TST_T)&
          (TSTUD.UNIT>STUD.UNIT)&
          (^((TSTUD.TSTATE=TST_A3)&(NOW<TSTUD.RTIME))))
          THEN
          CALL INSRTP;
    END;
EOF: CALL CLOSE_FILE(STUDFIL);
    OPEN FILE(STUDFIL) EXCL UPDATE DIRECT UNBUFFERED;
    NONE=FALSE;
    IF (PEND<2) THEN DO; /* SELECT INSTR.*/
       CALL READ_SYSP_LOCK;
       IF (SYSP.CUR_I_TEST>=I_TEST_LIM) THEN DO;
          NONE=TRUE;
          END;
       ELSE DO;
          PID1='INST';
          PID2=PID1;
          SYSP.CUR_I_TEST=SYSP.CUR_I_TEST+1;
          SYSP.ASSIGN_TA=MOD(SYSP.ASSIGN_TA+1,TAS);
          END;
       CALL UPDATE_SYSP;
       END;
    ELSE DO;
       PEND1=NOT_EQ(1,PEND);
       I=GENRAND(PEND1,0,0,0);
       PID1=PINDX(I).ID;
       TEMP = PINDX(1);
       PINDX(1)=PINDX(I);
       PINDX(I)=TEMP;
       PEND1=NOT_EQ(2,PEND);
       I=GENRAND(PEND1-1,0,0,0);
       PID2=PINDX(I+1).ID;
       END;
 /*LOGICAL END OF FIND PROCTORS. */
```

```
/*****************************************************/
/*                                                   */
/*    THIS IS A SUBROUTINE OF THE FIND_PROCTORS      */
/*    PROCEDURE WHICH TAKES THE PROCTOR NAME AND     */
/*    PROCTOR VALUE AND INSERTS THE RECORD TO THE    */
/*    PINDX ARRAY IN SUCH AS THAT ALL THE            */
/*    IN PINDX ARE IN ASCENDING ORDER OF PROCTOR     */
/*    POINTS.                                        */
/*                                                   */
/*****************************************************/
  INSRTP:PROC;
     DCL 1 CARRY LIKE PINDX;
     DCL (K,L)                    FIXED BIN(15);
     DCL INSRTD                   BIT(1);
     DCL LBL                      LABEL;
     ON SUBSCRIPTRANGE BEGIN;
        DCL 1 TEMP(LSTUD) LIKE PINDX;
        DCL M FIXED BIN(15);
        TEMP=PINDX;
        LSTUD=LSTUD+LSTUDMORE;
        FREE PINDX;
        ALLOC PINDX;
        DO M=1 TO PEND;
           PINDX(M)=TEMP(M);
        END;
        GOTO LBL;
        END;
     INSRTD=FALSE;
     LBL=A;
     DO K=1 TO PEND WHILE(^INSRTD);
        IF (TSTUD.PROCTOR<PINDX(K).PROCTOR) THEN DO;
        A: DO L=PEND TO K BY -1;
(SUBSCRIPTRANGE): PINDX(L+1)=PINDX(L);
           END;
           PINDX(K)=TSTUD, BY NAME;
           INSRTD=TRUE;
        END;
     END;
     PEND=PEND+1;
     LBL=B;
(SUBRG): B: IF (^INSRTD) THEN PINDX(PEND)=TSTUD, BY NAME;
   END; /* INSRTP*/
```

```
/*****************************************************/
/*                                                   */
/*    THIS IS A SUBROUTINE OF THE FIND_PROCTORS      */
/*    PROCEDURE. IT IS USED TO SCAN THE PINDX        */
/*    ARRAY FOR A CONTIGUOUS SET OF STUDENTS WHICH*/
/*    THE SAME PROCTOR POINTS. SINCE THE PINDX       */
/*    ARRAY BUILT IN SORTED ORDER THIS ROUTINE       */
/*    ACCEPTS TWO ARGUMENTS: A SCAN START POSITION*/
/*    AND END POSITION. THE INDEX OF THE LAST        */
```

```
/*    STUDENT , IN THE RANGE OF THE SCAN, WHO HAS */
/*    SAME AMOUNT OF PROCTOR POINTS AS THE STUDENT*/
/*    INDEXED AT THE START LOCATION IS RETURNED TO*/
/*    TO THE CALLING PROGRAM. THIS INFORMATION IS */
/*    TO DETERMINE IF THERE ARE MORE THAN ONE     */
/*    STUDENT WITH THE LOWEST (OR SECOND LOWEST)  */
/*    PROCTOR POINTS.                             */
/*                                                */
/**************************************************/
  NOT_EQ:PROC(STRT,END) RETURNS(FIXED BIN(15));
     DCL (STRT,END,MARKER)          FIXED BIN(15);
     DCL K                          FIXED BIN(15);
     DCL EQ                         BIT(1);
     EQ=TRUE;
     MARKER=END;
     DO K=STRT+1 TO END WHILE(EQ);
        IF (PINDX(STRT).PROCTOR^=PINDX(K).PROCTOR)
           THEN DO;
           EQ=FALSE;
           MARKER=K-1;
           END;
     END;
     RETURN(MARKER);
  END; /* NOT EQ */
END; /* FIND PROCTORS */

/**************************************************/
/*                                                */
/*   THIS PROCEDURE IS CALLED TO PROMPT PROCTORS  */
/*   BEFORE THEY ENTER THEIR MARK TO ENTER THE    */
/*   UNIT AND QUESTION NUMBERS WRITTEN BY THE     */
/*   STUDENT BEING PROCTORED TO VERIFY THAT HE    */
/*   WROTE THE SAME TEST WHICH WAS GIVEN TO HIM.  */
/*   IF THE PROCTOR RESPONDS WITH A BARE RETURN TO*/
/*   ANY OF THE PROMPTS THE RETURN ARGUMENT-      */
/*   DONTKNOW IS SET TO TRUE. IT ALLOWS ANOTHER   */
/*   3 TRIALS OF ENTERING UNIT NUMBERS.           */
/*                                                */
/**************************************************/
GET_QUES_NUM:PROC(DONTKNOW);
  DCL (BLANKF,NGOOD,EQ,CONT,DONTKNOW)    BIT(1);
  DCL QARRY(3)                           BIT(1);
  DCL (VAL,TGOOD,I,K,TRIAL)              FIXED BIN(15);
  DCL TRY                                FIXED BIN(15) INIT(3);
  DCL VALOUT                             PICTURE 'ZZZ9';
  DONTKNOW=FALSE;
  CONT=TRUE;
  DO I=0 TO TRY WHILE(CONT);
     CALL TPUT('ENTER UNIT NUMBER WRITTEN BY THE'||
                  ' STUDENT     : ',1);
     CALL GET_INT(VAL,NGOOD,BLANKF);
     IF (BLANKF) THEN
        CONT=FALSE;
     ELSE DO;
        IF (NGOOD) THEN
           CALL TPUT('INVALID NUMBER, RE-ENTER.',1);
        ELSE DO;
           IF ((STUD.UNIT+1)^=VAL) THEN DO;
              VALOUT=VAL;
              CALL TPUT('STUDENT DID NOT WRITE TEST'||
                 ' ON UNIT:'||VALOUT,1);
           END ;
           ELSE CONT=FALSE;
        END;
     END;
  END;
END;
QARRY=FALSE;
CONT=TRUE;
TGOOD=0;
IF (BLANKF|(I>TRY)) THEN DO;
   DONTKNOW=TRUE;
   CONT=FALSE;
   END;
TRIAL=0;
DO WHILE(CONT&(TRIAL<=TRY));
   CALL TPUT('ENTER A QUESTION NUMBER WRITTEN BY THE'||
   ' STUDENT     : ',1);
   CALL GET_INT(VAL,NGOOD,BLANKF);
   IF (BLANKF) THEN
      CONT=FALSE;
   ELSE DO;
      IF (NGOOD) THEN
         CALL TPUT('INVALID NUMBER, RE-ENTER.',1);
      ELSE DO;
         TRIAL=TRIAL+1;
         K=0;
         SELECT(VAL);
         WHEN(ABS(STUD.Q1)) K=1;
         WHEN(ABS(STUD.Q2)) K=2;
         WHEN(ABS(STUD.Q3)) K=3;
         OTHERWISE K=0;
         END; /* SELECT */
         IF (K=0) THEN DO;
            VALOUT=VAL;
            CALL TPUT('STUDENT WAS NOT ISSUED '||
            'QUESTION NUMBER:'||VALOUT,1);
            END;
         ELSE DO;
            IF (QARRY(K)) THEN DO;
              VALOUT=VAL;
              CALL TPUT('QUESTION NUMBER:'||VALOUT||
              '  HAS ALREADY BEEN VERIFIED.',1);
              END;
            ELSE DO;
               TRIAL=0;
```

```
                TGOOD=TGOOD+1;
                QARRY(K)=TRUE;
                END;
             END;
          END;
       END;
       IF (TGOOD=3) THEN CONT=FALSE;
    END; /*WHILE*/
    IF (BLANKF|(TRIAL>TRY)) THEN DONTKNOW=TRUE;
  END; /* GET QUES NUM */
END; /* SESSION CONTROL */

/*****************************************************/
/*                                                   */
/*    THIS PROCEDURE VALIDATES THE INPUT MARKED      */
/*    PAPER WHETHER IT IS CORRECT SYNTAX.            */
/*                                                   */
/*****************************************************/
VALID_IN_MARKED:PROC(VALID,JOBID,CID);
  DCL VALID                  BIT(1);
  DCL JOBID                  CHAR(*) VARYING;
  DCL CID                    CHAR(7);
  ON KEY(MAILIN) GOTO EOF;
  ON ENDFILE(MAILIN) GOTO EOF;
  VALID=TRUE;
  IF VALID THEN DO;
     READ FILE(MAILIN) INTO(RECORD) KEY('1+');
     READ FILE(MAILIN) INTO(RECORD);
     LINE=SUBSTR(RECORD,256);
     IF (CID='INST') THEN
        IF (INDEX(LINE,'To:')^=1)|(INDEX(LINE,PROF)=0)
        THEN
        VALID=FALSE;
     ELSE
        JOBID=PREFIX;
  ELSE
     IF (INDEX(LINE,'To:')^=1)|(INDEX(LINE,PROF)=0)
       |(INDEX(LINE,SUBSTR(CID,1,INDEX(CID||' ',' ')-1)
       )=0) THEN
       VALID=FALSE;
     ELSE
        JOBID=SUBSTR(CID,1,INDEX(CID||' ',' ')-1);
  END;
 IF VALID THEN DO;
    READ FILE(MAILIN) INTO(RECORD);
    LINE=SUBSTR(RECORD,256);
    IF (INDEX(LINE,'From:')^=1)|(INDEX(LINE,SUBSTR(
       STUD.ID,1,INDEX(STUD.ID||' ',' ')-1))=0) THEN
       VALID=FALSE;
    END;
 IF VALID THEN DO;
    READ FILE(MAILIN) INTO(RECORD);
    LINE=SUBSTR(RECORD,256);
    IF (INDEX(LINE,'Unit '||COMPRESS(STUD.UNIT+1))=0)|
       (INDEX(LINE,COMPRESS(STUD.Q1)||' '||
       COMPRESS(STUD.Q2)||' '||COMPRESS(STUD.Q3))=0)
       THEN
       VALID=FALSE;
    END;
 IF ^VALID THEN DO;
EOF: CALL TPUT('POINT TO THE MARKED PAPER '||
       'BEFORE YOU EXECUTE THE PSI PROGRAM.',1);
    CALL TPUT('NO RESULT ENTERED.',1);
    VALID=FALSE;
    END;
 END; /* VALID_IN_MARKED */

/*****************************************************/
/*                                                   */
/*    THIS PROCEDURE VALIDATES THE INPUT CONDITIONAL */
/*    PAPER WHETHER IT IS CORRECT SYNTAX.            */
/*                                                   */
/*****************************************************/
VALID_IN_CONDITIONAL:PROC(VALID,JOBID,CID,WHERE);
  DCL VALID                  BIT(1);
  DCL (JOBID,WHERE)          CHAR(*) VARYING;
  DCL CID                    CHAR(7);
  ON KEY(MAILIN) GOTO EOF;
  ON ENDFILE(MAILIN) GOTO EOF;
  VALID=TRUE;
  IF VALID THEN DO;
     READ FILE(MAILIN) INTO(RECORD) KEY('1+');
     READ FILE(MAILIN) INTO(RECORD);
     LINE=SUBSTR(RECORD,256);
     IF (CID='INST') THEN
        IF (INDEX(LINE,'To:')^=1) THEN
        VALID=FALSE;
     ELSE
        JOBID=PREFIX;
  ELSE
     IF (INDEX(LINE,'To:')^=1)|(INDEX(LINE,
        SUBSTR(CID,1,INDEX(CID||' ',' ')-1))=0) THEN
        VALID=FALSE;
     ELSE
        JOBID=SUBSTR(CID,1,INDEX(CID||' ',' ')-1);
  END;
 IF VALID THEN DO;
    READ FILE(MAILIN) INTO(RECORD);
    LINE=SUBSTR(RECORD,256);
    IF (INDEX(LINE,'From:')^=1) THEN
       VALID=FALSE;
    ELSE
       WHERE=SUBSTR(LINE,10);
    END;
 IF VALID THEN DO;
```

```
      READ FILE(MAILIN) INTO(RECORD);
      LINE=SUBSTR(RECORD,256);
      IF (INDEX(LINE,'Unit '||COMPRESS(STUD.UNIT+1))=0)|
         (INDEX(LINE,COMPRESS(STUD.Q1)||' '||
         COMPRESS(STUD.Q2)||' '||COMPRESS(STUD.Q3))=0)
         THEN
         VALID=FALSE;
      END;
   IF ^VALID THEN DO;
EOF: CALL TPUT('POINT TO THE CONDITIONAL PAPER '||
         'BEFORE YOU EXECUTE THE PSI PROGRAM.',1);
      CALL TPUT('NO CONDITIONAL PAPER MAILED.',1);
      VALID=FALSE;
      END;
   END; /* VALID_IN_CONDITIONAL */

/*****************************************************/
/*                                                   */
/*   THIS PROCEDURE IS USED TO ENTER A PROCTOR RE-   */
/*   SULT. THE INPUT ARGUMENT CID CONTAINS THE       */
/*   STUDENT NUMBER OF THE PROCTOR ENTERING THE      */
/*   RESULT. THE INPUT ARGUMENT BLANKF IS SET TO     */
/*   TRUE IF THE PROCTOR RESPONDED WITH A RETURN     */
/*   TO ONE OF THE PROMPTS IN THE GET_QUES_NUM       */
/*   PROCEDURE. IF BLANKF IS SET TO TRUE THE         */
/*   PROCTOR IS ONLY ALLOWED TO ENTER A RESTUDY      */
/*   RESULT (OR NO RESULT AT ALL).                   */
/*                                                   */
/*****************************************************/
  GET_MARK:PROC(BLANKF,CID,FOUND,RESP,PERMIT);
    DCL CID                    CHAR(7);
    DCL (BLANKF,NOPASS,FOUND) BIT(1);
    DCL PERMIT(*)              BIT(1);
    DCL RESP                   FIXED BIN(15);
    DCL MMSG                   CHAR(38) INIT(
    '(P)ASS,(C)ONDITIONAL,(R)ESTUDY?     : ');
    DCL PCR(3)                 CHAR(1) INIT('P','C','R');
    DCL (R1,R2,R3)             FIXED BIN(15);
    DCL OUTVAL                 PICTURE 'ZZZ9';
    DCL QMSG1                  CHAR(20) INIT(
    'WAS QUESTION NUMBER:');
    DCL QMSG2                  CHAR(42) INIT(
    ' COMPLETED SUCCESSFULLY (Y)ES,(N)O?     : ');
    DCL WHERE                  CHAR(100) VARYING;
    DCL JOBID                  CHAR(8) VARYING;
    DCL VALID                  BIT(1);
    DCL MISSED                 CHAR(20) VARYING INIT('');
    NOPASS=BLANKF;
    CALL GET_RESP(MMSG,PCR,RESP,0);
    BLANKF=FALSE;
    CALL READ_STUD_LOCK(FOUND);
    IF (^FOUND) THEN RETURN;
    IF (^PERMIT(STUD.TSTATE)) THEN DO;
      CALL TPUT('USERID: '||STUD.ID||
         'HAS JUST BEEN MARKED, NO RESULT ENTERED.',1);
      FOUND=FALSE;
      UNLOCK FILE(STUDFIL) KEY(STUD.ID);
      RETURN;
      END;
    STUD.Q1=ABS(STUD.Q1);
    STUD.Q2=ABS(STUD.Q2);
    STUD.Q3=ABS(STUD.Q3);
    CALL UPDATE_STUD(STUD.ID,FOUND);
    IF (^FOUND) THEN CALL INTERNAL_ERR(25);
    SELECT(RESP);
    WHEN(1) DO;
       IF (NOPASS) THEN DO;
          CALL TPUT('NOT ALLOWED TO PASS. NO RESULT '||
          ' ENTERED.',1);
          BLANKF=TRUE;
          END;
       ELSE DO;
          CALL VALID_IN_MARKED(VALID,JOBID,CID);
          IF ^VALID THEN DO;
             BLANKF=TRUE;
             RETURN;
             END;
          CALL STATE_MACH('J',CID); /* PASS */
          CALL TPUT('PASS RESULT ENTERED.',1);
          PUT FILE(INTFILE) EDIT('Result:  PASS')(SKIP,A);
          WHERE=PROF||', '||SUBSTR(STUD.ID,1,
             INDEX(STUD.ID||' ',' ')-1);
          CALL MAIL_OUT(WHERE,JOBID,PREFIX,'PASS');
          END;
       END; /*WHEN*/
    WHEN(2) DO;
       IF (NOPASS) THEN DO;
          CALL TPUT('NOT ALLOWED TO "CONDITIONAL". '||
             'NO RESULT ENTERED.',1);
          BLANKF=TRUE;
          END;
       ELSE DO;
          CALL VALID_IN_CONDITIONAL(VALID,JOBID,CID,WHERE);
          IF ^VALID THEN DO;
             BLANKF=TRUE;
             RETURN;
             END;
          CALL STATE_MACH('K',CID); /*CONDITIONAL*/
          CALL TPUT('CONDITIONAL RESULT ENTERED.',1);
          CALL TPUT('ENTER PASS OR RESTUDY LATER.',1);
          PUT FILE(INTFILE) EDIT('Result:  CONDITIONAL')
             (SKIP,A);
          CALL MAIL_OUT(WHERE,JOBID,PREFIX,'CONDITIONAL');
          END;
       END; /*WHEN*/
```

```
   WHEN(3) DO; /* RESTUDY */
      CALL VALID_IN_MARKED(VALID,JOBID,CID);
      IF ^VALID THEN DO;
         BLANKF=TRUE;
         RETURN;
         END;
      OUTVAL = STUD.Q1;
      CALL GET_RESP(QMSG1||OUTVAL||QMSG2,YN,R1,1);
      OUTVAL=STUD.Q2;
      CALL GET_RESP(QMSG1||OUTVAL||QMSG2,YN,R2,1);
      OUTVAL=STUD.Q3;
      CALL GET_RESP(QMSG1||OUTVAL||QMSG2,YN,R3,1);
      CALL READ_STUD_LOCK(FOUND);
      IF (^FOUND) THEN RETURN;
      IF (^PERMIT(STUD.TSTATE)) THEN DO;
         CALL TPUT('USERID: '||STUD.ID||
             'HAS JUST BEEN MARKED, NO RESULT ENTERED.',1);
         FOUND=FALSE;
         UNLOCK FILE(STUDFIL) KEY(STUD.ID);
         RETURN;
         END;
      IF (R1=2) THEN DO;
         MISSED=MISSED||' '||COMPRESS(STUD.Q1);
         STUD.Q1=-STUD.Q1;
         END;
      IF (R2=2) THEN DO;
         MISSED=MISSED||' '||COMPRESS(STUD.Q2);
          STUD.Q2=-STUD.Q2;
          END;
      IF (R3=2) THEN DO;
         MISSED=MISSED||' '||COMPRESS(STUD.Q3);
         STUD.Q3=-STUD.Q3;
         END;
      CALL UPDATE_STUD(STUD.ID,FOUND);
      IF (^FOUND) THEN CALL INTERNAL_ERR(25);
      CALL STATE_MACH('L',CID);
      CALL TPUT('RESTUDY RESULT ENTERED.',1);
      PUT FILE(INTFILE) EDIT('Missed: '||MISSED,
         'Result:  RESTUDY')(SKIP,A);
      WHERE=PROF||', '||SUBSTR(STUD.ID,1,
         INDEX(STUD.ID||' ',' ')-1);
      CALL MAIL_OUT(WHERE,JOBID,PREFIX,'RESTUDY');
      END;
   WHEN(0) DO;
      BLANKF=TRUE;
      END;
   OTHERWISE DO;
      BLANKF=TRUE;
      CALL INTERNAL_ERR(9);
      END;
   END;
END; /* GET MARK */
```

```
/*****************************************************/
/*                                                   */
/*    THIS PROCEDURE MAILS QUESTIONS TO THE          */
/*    STUDENT THROUGH BATCH ELECTRONIC MAIL.         */
/*                                                   */
/*****************************************************/
 MAIL_QUESTIONS:PROC(WHERE,JOBID,PREFIX);
    DCL WHERE                 CHAR(*) VARYING;
    DCL (JOBID,PREFIX)        CHAR(*) VARYING;
    DCL QUESTIONS(3)          FIXED BIN(15);
    DCL I                     FIXED BIN(15);
    DCL REPEAT                BUILTIN;
    CALL DYNAM(WORK,'ALLOC ','DD=MAILOUT;','PREF=PSI;',
       'DSN='||PREFIX||'.*MAIL.*OUT VOL=WORK F DSO=PS '||
       'LRECL=255 BLKSIZE=255 TRK PRIM=10 SEC=10 '||
       'UNIT=DISK NEW CAT CDEL CLOSE;');
    IF (PLIRETV^=0) THEN
       CALL DYNAM(WORK,'ALLOC ','DD=MAILOUT;','PREF=PSI;',
          'DSN='||PREFIX||'.*MAIL.*OUT OLD CDEL CLOSE;');
    OPEN FILE(MAILOUT) LINESIZE(255);
    PUT FILE(MAILOUT) EDIT(
       'To:      '||WHERE,
       'From:    '||JOBID,
       'Subject: '||
       SUBSTR(STUD.ID,1,INDEX(STUD.ID||' ',' ')-1)||' '||
       SUBSTR(OUTDATE,4)||' '||
       SUBSTR(SEPARATE(SUBSTR(TIME,1,6)),1,5)||' Unit '
       ||COMPRESS(STUD.UNIT+1)||' : '||
       COMPRESS(STUD.Q1)||' '||COMPRESS(STUD.Q2)||' '||
       COMPRESS(STUD.Q3),
       ' ')(SKIP,A);
    ON KEY(TESTIN) RECORD=REPEAT(' ',254)||'CONTACT YOUR '
      ||'INSTRUCTOR THAT THE FILE "HEADER" WAS NOT FOUND.';
    READ FILE(TESTIN) INTO(RECORD) KEY('HEADER');
    PUT FILE(MAILOUT) EDIT(SUBSTR(RECORD,256))(SKIP,A);
    QUESTIONS(1)=STUD.Q1;
    QUESTIONS(2)=STUD.Q2;
    QUESTIONS(3)=STUD.Q3;
    ON KEY(TESTIN) RECORD=REPEAT(' ',254)||'CONTACT YOUR'
       ||' INSTRUCTOR THAT UNIT '||COMPRESS(STUD.UNIT+1)||
       ' QUESTION '||COMPRESS(QUESTIONS(I))||
       ' WAS NOT FOUND.';
    ON ENDFILE(TESTIN) GOTO EOF;
    DO I=1 TO 3;
       READ FILE(TESTIN) INTO(RECORD) KEY('TESTS.U'||
          COMPRESS(STUD.UNIT+1)||'.Q'||
          COMPRESS(QUESTIONS(I))||' FIRST');
       PUT FILE(MAILOUT) EDIT('Question '||
          COMPRESS(QUESTIONS(I))||': ',
          SUBSTR(RECORD,256))(SKIP,A);
       DO WHILE('1'B);
          READ FILE(TESTIN) INTO(RECORD);
```

```
            PUT FILE(MAILOUT) EDIT(SUBSTR(RECORD,256))
               (SKIP,A);
         END;
    EOF: PUT FILE(MAILOUT) EDIT('Answer '||
            COMPRESS(QUESTIONS(I))||': ',
            ' ')(SKIP,A);
      END;
      PUT FILE(MAILOUT) EDIT('Comments: ',
         ' ')(SKIP,A);
      CLOSE FILE(MAILOUT);
      CALL BATCH_MAIL_PSI(JOBID,PREFIX);
      CALL TPUT('TEST PAPER IS MAILED FROM '||JOBID||
         ' TO '||WHERE,1);
   END; /* MAIL_QUESTIONS */

/*****************************************************/
/*                                                   */
/*    THIS PROCEDURE MAILS OUT THE FILE MAILIN TO    */
/*    THE STUDENT THROUGH BATCH ELECTRONIC MAIL.     */
/*                                                   */
/*****************************************************/
   MAIL_OUT:PROC(WHERE,JOBID,PREFIX,MSG);
      DCL WHERE                 CHAR(*) VARYING;
      DCL (JOBID,PREFIX)        CHAR(*) VARYING;
      DCL MSG                   CHAR(*);
      CALL DYNAM(WORK,'ALLOC ','DD=MAILOUT;','PREF='||
         PREFIX||';','DSN=*MAIL.*OUT VOL=WORK F DSO=PS '||
         'LRECL=255 BLKSIZE=255 TRK PRIM=10 SEC=10 '||
         'UNIT=DISK NEW CAT CDEL CLOSE;');
      IF (PLIRETV^=0) THEN
         CALL DYNAM(WORK,'ALLOC ','DD=MAILOUT;','PREF='||
            PREFIX||';','DSN=*MAIL.*OUT OLD CDEL CLOSE;');
      OPEN FILE(MAILOUT) LINESIZE(255);
      PUT FILE(MAILOUT) EDIT(
         'To:      '||WHERE,
         'From:    '||JOBID)
         (SKIP,A);
      ON KEY(MAILIN) GOTO EOF;
      ON ENDFILE(MAILIN) GOTO EOF;
      READ FILE(MAILIN) INTO(RECORD) KEY('1+');
      READ FILE(MAILIN) IGNORE(2);
      READ FILE(MAILIN) INTO(RECORD);
      PUT FILE(MAILOUT) EDIT(SUBSTR(RECORD,256)||' '||MSG)
         (SKIP,A);
      CLOSE FILE(INTFILE);
      ON ENDFILE(INTFILE) GOTO EOF1;
      OPEN FILE(INTFILE) INPUT;
      DO WHILE('1'B);
         GET FILE(INTFILE) EDIT(RECORD)(A(72));
         PUT FILE(MAILOUT) EDIT(RECORD)(SKIP,A);
      END;
EOF1: CLOSE FILE(INTFILE);
      OPEN FILE(INTFILE) OUTPUT LINESIZE(72);
      DO WHILE('1'B);
         READ FILE(MAILIN) INTO(RECORD);
         PUT FILE(MAILOUT) EDIT(SUBSTR(RECORD,256))
            (SKIP,A);
      END;
  EOF: CLOSE FILE(MAILOUT);
      CALL BATCH_MAIL(JOBID,PREFIX);
      CALL TPUT(MSG||' TEST PAPER IS MAILED FROM '||JOBID||
         ' TO '||WHERE,1);
   END; /* MAIL_OUT */.

/*****************************************************/
/*                                                   */
/*    THIS PROCEDURE SUBMITS A BATCH MANTES TO       */
/*    MAIL TESTS.                                    */
/*                                                   */
/*****************************************************/
BATCH_MAIL:PROC(JOBID,PREFIX);
   DCL (JOBID,PREFIX)             CHAR(*) VARYING;
   CALL DYNAM(WORK,'ALLOC ','DD=COMMOUT;','PREF='||
      PREFIX||';','DSN=*MAIL.*COMM VOL=WORK F DSO=PS '||
      'LRECL=72 BLKSIZE=72 TRK PRIM=1 SEC=1 '||
      'UNIT=DISK NEW CAT CDEL CLOSE;');
   IF (PLIRETV^=0) THEN
      CALL DYNAM(WORK,'ALLOC ','DD=COMMOUT;','PREF='||
         PREFIX||';','DSN=*MAIL.*COMM OLD CDEL CLOSE;');
   OPEN FILE(COMMOUT) LINESIZE(72);
   PUT FILE(COMMOUT) EDIT(
      ••• confidential MANTES commands •••  )(SKIP,A);
   CLOSE FILE(COMMOUT);
   CALL DYNAM(WORK,'ALLOC ','DD=INTRDR REMOTE=LOCAL '||
      'SYSOUT=A CLOSE SYSOPROG=INTRDR;');
   OPEN FILE(INTRDR) LINESIZE(80);
   PUT FILE(INTRDR) EDIT(
      '//'
      '//'  ••• confidental JCL commands for batch job  •••
      '//'
      (SKIP,A);
   CLOSE FILE(INTRDR);
END; /* BATCH_MAIL */

/*****************************************************/
/*                                                   */
/*    THIS PROCEDURE SUBMITS A BATCH MANTES TO       */
/*    MAIL TESTS BY PSI USERID.                      */
/*                                                   */
/*****************************************************/
BATCH_MAIL_PSI:PROC(JOBID,PREFIX);
   DCL (JOBID,PREFIX)             CHAR(*) VARYING;
   CALL DYNAM(WORK,'ALLOC ','DD=COMMOUT;','PREF=PSI;',
      'DSN='||PREFIX||'.*MAIL.*COMM VOL=WORK F DSO=PS '||
      'LRECL=72 BLKSIZE=72 TRK PRIM=1 SEC=1 '||
```

```
      'UNIT=DISK NEW CAT CDEL CLOSE;');
   IF (PLIRETV^=0) THEN
      CALL DYNAM(WORK,'ALLOC ','DD=COMMOUT;','PREF=PSI;',
         'DSN='||PREFIX||'.*MAIL.*COMM OLD CDEL CLOSE;');
    OPEN FILE(COMMOUT) LINESIZE(72);
    PUT FILE(COMMOUT) EDIT(
       ••• confidential MANTES commands •••  )(SKIP,A);
    CLOSE FILE(COMMOUT);
    CALL DYNAM(WORK,'ALLOC ','DD=INTRDR REMOTE=LOCAL '||
       'SYSOUT=A CLOSE SYSOPROG=INTRDR;');
    OPEN FILE(INTRDR) LINESIZE(80);
    PUT FILE(INTRDR) EDIT(••• confidential JCL commands  •••)
       (SKIP,A);
    CLOSE FILE(INTRDR);
END; /* BATCH_MAIL_PSI */

/*****************************************************/
/*                                                   */
/*    THIS PROCEDURE IS THE STATE MACHINE FOR THE */
/*    THE STUDENT TRANSACTIONS. A TRANSACTION TYPE*/
/*    IS PASSED IN ARGUMENT TRANS AND IT IS USED  */
/*    ALONG WITH THE STUDENTS CURRENT STATE TO    */
/*    GIVE THE STUDENT HIS NEW STATE. THE         */
/*    TRANSACTION IS THEN LOGGED BY A CALL TO THE */
/*    LOG_TRANS ROUTINE.                          */
/*                                                   */
/*****************************************************/
  STATE_MACH:PROC(TRANS,CID);
     DCL CID                          CHAR(7);
     DCL TRANS                        CHAR(1);
     DCL (FOUND,STUD_PROC)            BIT(1);
     CALL READ_SYSP;
     CALL READ_STUD_LOCK(FOUND);
     IF (^FOUND) THEN RETURN;
     SELECT(TRANS);
     WHEN('T') DO; /* GENERATE TEST */
        STUD.TSTATE=TST_T;
        IF (CID^=' ') THEN
           STUD.RTIME=SUBSTR(DATE,5,2)||SUBSTR(TIME,1,4);
        CALL UPDATE_STUD(STUD.ID,FOUND);
        IF (^FOUND) THEN CALL INTERNAL_ERR(20);
        CALL LOG(TRANS,CID);
        END;
     WHEN('C')DO; /* CANCEL TEST */
        STUD.TSTATE=TST_R3;
        STUD.RTIME=ADDTD(SUBSTR(DATE,5,2)||SUBSTR(TIME,1,4),
           RSTYTME);
        CALL UPDATE_STUD(STUD.ID,FOUND);
        IF (^FOUND) THEN CALL INTERNAL_ERR(20);
        CALL LOG(TRANS,CID);
        END;
     WHEN('E')DO; /* TEST EXPIRED */
        STUD.TSTATE=TST_R3;
        STUD.RTIME=ADDTD(STUD.RTIME,TSTWITHIN);
        STUD.RTIME=ADDTD(STUD.RTIME,RSTYTME);
        CALL UPDATE_STUD(STUD.ID,FOUND);
        IF (^FOUND) THEN CALL INTERNAL_ERR(20);
        CALL LOG(TRANS,CID);
        END;
     WHEN('I')DO; /* MARK TEST */
        STUD.TSTATE=TST_R0;
        STUD.RTIME=SUBSTR(DATE,5,2)||SUBSTR(TIME,1,4);
        CALL UPDATE_STUD(STUD.ID,FOUND);
        IF (^FOUND) THEN CALL INTERNAL_ERR(20);
        CALL LOG(TRANS,CID);
        END;
     WHEN('P')DO; /* PROCTOR SELECT */
        STUD.PSTATE=PST_P;
        STUD.SID=CID;
        CALL UPDATE_STUD(STUD.ID,FOUND);
        IF (^FOUND) THEN CALL INTERNAL_ERR(20);
        CALL LOG(TRANS,CID);
        END;
     WHEN('M')DO;
        STUD.PSTATE=PST_PA;
        STUD.PROCTOR=STUD.PROCTOR+SYSP.VPROC;
        CALL TOTAL_MARKS;
        CALL UPDATE_STUD(STUD.ID,FOUND);
        IF (^FOUND) THEN CALL INTERNAL_ERR(20);
        END;
     WHEN('N')DO;
        STUD.PSTATE=PST_PA;
        CALL UPDATE_STUD(STUD.ID,FOUND);
        IF (^FOUND) THEN CALL INTERNAL_ERR(20);
        END;
     WHEN('K')DO; /* CONDITIONAL */
        DCL CPTBL(L_TST)       FIXED BIN(15) INIT(
          TST_I,TST_T,TST_NT,TST_CR0,TST_CR1,TST_CR2,TST_R3,
          TST_CR3,TST_CR1,TST_CR2,TST_CR3);
        STUD.TSTATE=CPTBL(STUD.TSTATE);
        STUD.RTIME=SUBSTR(DATE,5,2)||SUBSTR(TIME,1,4);
        CALL UPDATE_STUD(STUD.ID,FOUND);
        IF (^FOUND) THEN CALL INTERNAL_ERR(20);
        CALL LOG(TRANS,CID);
        END;
     WHEN('R')DO; /* REVISION */
        DCL RPTBL(L_TST)       FIXED BIN(15) INIT(
           TST_I,TST_T,TST_NT,TST_R0,TST_R1,TST_R2,TST_R3,
           TST_R0,TST_R1,TST_R2,TST_CR0);
        STUD.TSTATE=RPTBL(STUD.TSTATE);
        STUD.RTIME=SUBSTR(DATE,5,2)||SUBSTR(TIME,1,4);
        CALL UPDATE_STUD(STUD.ID,FOUND);
        IF (^FOUND) THEN CALL INTERNAL_ERR(20);
        CALL LOG(TRANS,CID);
        END;
     WHEN('J')DO; /* PASS RESULT */
```

```
      DCL PPTBL(L_TST)       FIXED BIN(15) INIT(
         TST_I,TST_I,TST_I,TST_R1,TST_NT,TST_R3,TST_R3,
         TST_CR1,TST_NT,TST_R3,TST_CR1);
      IF ((CID='INST')|(CID='TA')) THEN DO;
         IF (STUD.TSTATE^=TST_T) THEN DO;
            SAVE=STUD;
            CALL STUD_PROCTORS(SAVE.ID,TRUE,FALSE,
               STUD_PROC);
            STUD=SAVE;
            IF ^STUD_PROC & (SYSP.CUR_I_TEST>0) THEN DO;
               CALL READ_SYSP_LOCK;
               SYSP.CUR_I_TEST=SYSP.CUR_I_TEST-1;
               CALL UPDATE_SYSP;
               END;
            END;
         STUD.TSTATE=TST_NT;
         END;
      ELSE DO;
         STUD.TSTATE=PPTBL(STUD.TSTATE);
         END;
      IF (STUD.TSTATE=TST_R3) THEN
         STUD.RTIME=ADDTD(SUBSTR(DATE,5,2)||
            SUBSTR(TIME,1,4),RSTYTME);
      IF (STUD.TSTATE=TST_NT) THEN DO;
         STUD.TEST=STUD.TEST+SYSP.VPASS;
         STUD.UNIT=STUD.UNIT+1;
         CALL TOTAL_MARKS;
         END;
      CALL UPDATE_STUD(STUD.ID,FOUND);
      IF (^FOUND) THEN CALL INTERNAL_ERR(20);
      CALL LOG(TRANS,CID);
      END;
   WHEN('L')DO; /* RESTUDY RESULT */
      DCL SPTBL(L_TST)   FIXED BIN(15) INIT(
         TST_I,TST_T,TST_NT,TST_R2,TST_R3,TST_R3,TST_R3,
         TST_CR2,TST_R3,TST_R3,TST_CR2);
      IF ((CID='INST')|(CID='TA')) THEN DO;
         IF (STUD.TSTATE^=TST_T) THEN DO;
            SAVE=STUD;
            CALL STUD_PROCTORS(SAVE.ID,TRUE,FALSE,
               STUD_PROC);
            STUD=SAVE;
            IF ^STUD_PROC & (SYSP.CUR_I_TEST>0) THEN DO;
               CALL READ_SYSP_LOCK;
               SYSP.CUR_I_TEST=SYSP.CUR_I_TEST-1;
               CALL UPDATE_SYSP;
               END;
            END;
         STUD.TSTATE=TST_R3;
         END;
      ELSE DO;
         STUD.TSTATE=SPTBL(STUD.TSTATE);
            END;
         IF (STUD.TSTATE=TST_R3) THEN
            STUD.RTIME=ADDTD(SUBSTR(DATE,5,2)||
               SUBSTR(TIME,1,4),RSTYTME);
         CALL UPDATE_STUD(STUD.ID,FOUND);
         IF (^FOUND) THEN CALL INTERNAL_ERR(20);
         CALL LOG(TRANS,CID);
         END;
      OTHERWISE DO;
         UNLOCK FILE(STUDFIL) KEY(STUD.ID);
         CALL INTERNAL_ERR(21);
         END;
      END;
   END; /* STATE_MACH */
   /*****************************************************/
   /*                                                   */
   /*       E N D    O F    P R O G R A M               */
   /*                                                   */
   /*****************************************************/
END; /* MAIN PROGRAM */
//LKED.SYSLIB DD
//            DD DSN=SYS2.PLI.OPT.PLITASK,DISP=SHR
//LKED.SYSLMOD DD DSN=eee the location of the production runs eee,DISP=OLD
```

APPENDIX B

SOURCE CODE FOR CAPSI

WITH CLASSROOM SETTING

```
//PSIJOB JOB 'F=DA11,T=20,L=8,I=120','PSI_MAIN',MSGLEVEL=(1,1)
//         EXEC ASMGC
//ASM.SYSIN DD *
*
*  CALL BY PL1
*  CALL SCN(BUFFER);
*     DCL BUFFER    CHAR(*) VARYING;
*
DUMREC1  CSECT
         ENTRY SCN
         DC    C'SCN'
         DC    AL1(5)
SCN      DS    0H
         STM   14,11,12(13)
         BALR  10,0
         USING *,10
         LA    4,SAVEAREA
         ST    13,SAVEAREA+4
         ST    4,8(13)
         LA    13,SAVEAREA
         L     4,0(1)                  GET ADDR OF BUFFER
         LA    6,2(4)                  GET ACTUAL BUFFER ADDR
         LH    4,0(4)                  GET LENGTH OF BUFFER
         TPUT  (6),(4),CONTROL
         L     13,4(13)
         LM    14,11,12(13)
         BR    14
SAVEAREA DC    20F'0'
         END
//         EXEC ASMGC
//ASM.SYSIN DD *
*
*  CALL BY PL1
*  CALL SEND(BUFFER,USERID);
*     DCL BUFFER   CHAR(*) VARYING;
*     DCL USERID   CHAR(8);
*
DUMREC2  CSECT
         ENTRY SEND
         DC    C'SEND'
         DC    AL1(5)
SEND     DS    0H
         STM   14,11,12(13)
         BALR  10,0
         USING *,10
         LA    4,SAVEAREA
         ST    13,SAVEAREA+4
         ST    4,8(13)
         LA    13,SAVEAREA
         L     4,0(1)                  GET ADDRESS OF BUFFER
         L     5,4(1)                  GET ADDRESS OF USERID
         LA    6,2(4)                  GET ACTUAL BUFFER ADDRESS
         LH    4,0(4)                  GET LENGTH OF BUFFER
         TPUT  (6),(4),CONTROL,USERIDL=(5)
         L     13,4(13)
         L     14,12(13)
         LM    0,11,20(13)
         BR    14
SAVEAREA DC    20F'0'
         END
// EXEC PL10CL,CSIZE=512K,IS=NIS,STG=NSTG,AP='F(1),INT',X=X,A=A
//PL1.SYSIN DD *
/*   PERSONALIZED SYSTEM INSTRUCTION   */
/*********************************************************************/
/*                                                                   */
/*              P S I    MAIN  PROGRAM.                              */
/*                                                                   */
/*              AUTHOR           FRANK  HERZOG  IAMC.               */
/*              DATE WRITTEN     JAN. 15  1984.                     */
/*              AUTHOR           MANIX  LEUNG, YIU-MAN.             */
/*              LAST UPDATE      FEB. 15  1984.                     */
/*                               OCT. 15  1985                      */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
/*     VERSION 2.0                                                   */
/*     CHANGES FROM SINGLE USER TO MULTIPLE USERS THAT CAN ACCESS    */
/*     ON A SHARED DATA BASE CONCURRENTLY.  THE STUDENT DATABASE     */
/*     WAS MODIFIED FROM REGIONAL(1) FILE TO INDEX FILE              */
/*     ORGANIZATION. THE FACILITIES OF MONITORING STUDENT            */
/*     ACTIVITIES AND SENDING MESSAGES ARE ADDED.                    */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*     THIS PROGRAM PERFORMS THE INTERACTIVE EDITING AND STUDENT     */
/*     TRANACTION CONTROL FUNCTIONS OF THE SYSTEM. THREE FILES       */
/*     ARE USED BY THIS PROGRAM. THE FIRST IS THE SYSTEM PARA-       */
/*     METER FILE , A SINGLE RECORD DIRECT ACCESS FILE,IT CONTAINS   */
/*     THE VALUES WHICH DETERMINE THE NUMBER OF UNITS IN THE COURSE  */
/*     ..... ETC. THE SECOND IS THE STUDENT RECORD FILE (STUDFIL)    */
/*     IT IS ALSO A DIRECT INDEX ACCESS FILE.  THREE RECORDS ARE     */
/*     PREALLOCATED TO STORE INFORMATION RELATED TO THE SPECIAL      */
/*     STUDENT NUMBERS EDIT,INST,TA. (ONLY THE PASSWORD FIELD IS     */
/*     USED IN THESE RECORDS.). THE THIRD FILE IS A TRANSACTION      */
/*     LOG FILE WHICH KEEPS A RECORD OF STUDENT TRANSACTIONS AS      */
/*     THEY OCCUR, FOR STATISTICAL PURPOSES. THIS PROGRAM IS         */
/*     DIVIDED INTO ROUTINES WHICH ROUGHLY PARALLEL THE MENU         */
/*     OPTIONS. THE FIRST ROUTINE WHICH IS PERFORMED IS AN           */
/*     INITIALIZATION ROUTINE WHICH SETS UP THE DIRECT INDEX FILE    */
/*     TO ACCESS STUDENT RECORDS AND ALSO WRITES A LOG RECORD WITH   */
/*     A TRANSACTION TYPE OF 2. AFTER INITIALIZATION , THE MAIN      */
/*     CONTROL MENU ROUTINE IS CALLED. IT IN TURN CALLS SUB-ROUTINE  */
/*     WHICH EDIT STUDENT RECORDS , START SESSION . . . ETC.         */
/*     DEPENDING ON THE CHOICES MADE BY THE USER.                    */
```

```
/*                                                                    */
/**********************************************************************/
/**********************************************************************/
/*                                                                    */
/*      GLOBAL DATA DICTIONARY                                        */
/*      ------ ---- ----------                                        */
/*                                                                    */
/*      LOGFIL          THIS IS THE LOG FILE. FOR A DESCRIPTION SEE   */
/*                      THE SECTION IN THE SYSTEM REFERENCE MANUAL    */
/*                      ON DATA FILES.                                */
/*      SYSPFIL         THE SYSTEM PARAMETER FILE- DESCRIBED IN THE   */
/*                      REFERENCE MANUAL.                             */
/*      STUDFIL         STUDENT RECORD FILE , DESCRIBED IN THE        */
/*                      SYSTEM REFERENCE MANUAL.                      */
/*      STAT_TBL_END_SESS THIS IS A TABLE OF TEST AND PROCTOR STATE   */
/*                      USED IN THE END SEESION ROUTINE.              */
/*      TST_ ...        THESE ARE THE DENOTATIONS FOR TEST STATES,    */
/*                      SEE THE SYSTEM REFERENCE MANUAL FOR A         */
/*                      MEANING OF THE CODES.                         */
/*      PST_. . .       THESE ARE THE DENOTATIONS FOR PROCTOR STATES  */
/*      CUTOFF          THIS IS THE TEST CUTOFF TIME LIMIT.           */
/*      CUR_I_TEST      THIS IS THE COUNTER CONTAINING THE NUMBER     */
/*                      OF TESTS CURRENTLY ASSIGNED TO THE            */
/*                      INSTRUCTOR OR TA.                             */
/*      I_TEST_LIM      THIS IS THE LIMIT TO THE NUMBER OF TESTS      */
/*                      WHICH CAN BE ASSIGNED TO THE INSTRUCTOR OR    */
/*                      TA.                                           */
/*      CURDATE         A VARIABLE CONTAINING THE CURRENT DATE.       */
/*                                                                    */
/**********************************************************************/
MPSI:PROC(PARM) OPTIONS(MAIN);
   DCL PARM                         CHAR(100) VARYING;
   DCL USERID                       CHAR(8) INIT(PARM);
   DCL SYSIN                        FILE STREAM INPUT;
   DCL SYSPRINT                     FILE STREAM OUTPUT PRINT
                                    ENV(RECSIZE(132));
   DCL LOGFIL                       FILE RECORD EXCLUSIVE KEYED
         UNBUFFERED ENV(F RECSIZE(42) BLKSIZE(42) REGIONAL(1) );
   DCL SYSPFIL                      FILE RECORD EXCLUSIVE KEYED
         UNBUFFERED ENV(F RECSIZE(285) BLKSIZE(285) REGIONAL(1) );
   DCL STUDFIL                      FILE RECORD KEYED
                              ENV(INDEXED KEYLOC(2) KEYLENGTH(7) RECSIZE(107)
         BLKSIZE(107) F BUFFERS(35));
   DCL SCN                          EXTERNAL OPTIONS(ASM,INTER)
                                    ENTRY(CHAR(*) VARYING);
   DCL SEND                         EXTERNAL OPTIONS(ASM,INTER,RETCODE)
                                    ENTRY(CHAR(*) VARYING, CHAR(8));
   DCL  1 STUD ,
          2 DELKEY                  BIT(8),
          2 ID                      CHAR(7) INIT(' '),
          2 NAME                    CHAR(30) INIT(' '),
          2 PHONE                   CHAR(7) INIT(' '),
          2 FACULTY                 CHAR(2) INIT(' '),
          2 YEAR                    CHAR(2) INIT(' '),
          2 STATUS                  CHAR(3) INIT(' '),
          2 UNIT                    FIXED BIN(15) INIT(0),
          2 Q1                      FIXED BIN(15) INIT(0),
          2 Q2                      FIXED BIN(15) INIT(0),
          2 Q3                      FIXED BIN(15) INIT(0),
          2 TEST                    FIXED DEC(7 ,3) INIT(0),
          2 PROCTOR                 FIXED DEC(7 ,3) INIT(0),
          2 TERM                    FIXED DEC(7 ,3) INIT(0),
          2 EXAM                    FIXED DEC(7 ,3) INIT(0),
          2 TOTAL                   FIXED DEC(7 ,3) INIT(0),
          2 LETTER                  CHAR(2) INIT(' '),
          2 PASSWORD                CHAR(8) INIT(' '),
          2 PSTATE                  FIXED BIN(15) INIT(0),
          2 TSTATE                  FIXED BIN(15) INIT(0),
          2 SID                     CHAR(7) INIT(' '),
          2 RTIME                   PICTURE '999999' ;
   DCL  1 TSTUD  LIKE STUD;
   DCL  1 SAVE  LIKE STUD;
   DCL  1 SYSP ,
          2 DELKEY                  BIT(8),
          2 NUNIT                   FIXED BIN(15),
          2 UNITL(100)              FIXED BIN(15),
          2 VPASS                   FIXED DEC(7,3),
          2 VPROC                   FIXED DEC(7,3),
          2 LGTHRESH(13)            FIXED DEC(7,3),
          2 SESSDATE                PICTURE '999999',
          2 SESSTIME                PICTURE '999999',
          2 SESSCUTOFF              PICTURE '999999',
          2 CUR_I_TEST              FIXED BIN(15),
          2 NLOGREC                 FIXED BIN(15);
   DCL  1 SYST LIKE SYSP;
   DCL  1 LOGREC,
          2 DELKEY                  BIT(8),
          2 TYPE                    CHAR(1),
          2 ID                      CHAR(7),
          2 TS                      FIXED BIN(15),
          2 PS                      FIXED BIN(15),
          2 UNIT                    FIXED BIN(15),
          2 Q1                      FIXED BIN(15),
          2 Q2                      FIXED BIN(15),
          2 Q3                      FIXED BIN(15),
          2 TP                      FIXED DEC(7,3),
          2 PP                      FIXED DEC(7,3),
          2 CTIME                   PICTURE '999999',
          2 SID                     CHAR(7);
   DCL LPTR                         FIXED BIN(15) INIT(10);
   DCL  1 LOGS(LPTR) LIKE LOGREC;
   DCL (LSTUD,LSTUDMORE)            FIXED BIN(15) INIT(100);
   DCL  1 PINDX(LSTUD)              CONTROLLED,
          2 ID                      CHAR(7),
          2 PROCTOR                 FIXED DEC(7, 3);
```

```
DCL STEND                       FIXED BIN(15);
DCL (LST,LSTMORE)               FIXED BIN(15) INIT(200);
DCL 1 ST(LST)                   CONTROLLED,
      2 ID                      CHAR(7),
      2 NAME                    CHAR(30);
DCL (LBUF,LBUFMORE)             FIXED BIN(15) INIT(2000);
DCL BUFFER                      CHAR(LBUF) VARYING CONTROLLED;
DCL 1 STAT_TBL_END_SESS,
    2 TSTATE(10)                FIXED BIN(15) INIT(1,8,1,9,9,9,
                                      1,8,9,1),
    2 PSTATE(4)                 FIXED BIN(15) INIT(1,1,1,4);
DCL L_SPEC                      FIXED BIN(15) INIT(5);
DCL SPEC_ID(L_SPEC)             CHAR(7) INIT(
    'INST','TA','EDIT','ALL','MARKER');
DCL PSC(L_PST)                  CHAR(13) INIT(
    'INITIAL','AVAILABLE','NOT AVAILABLE','PROCTORING');
DCL TSC(L_TST)                  CHAR(11) INIT(
    'INITIAL','WRITING','NOT WRITING','NO MARK YET','ONE PASS',
    'ONE FAIL','RESTUDY','OUT. TEST','OUT. MARK','CANCELLED');
DCL LGLIT(13)                   CHAR(2) INIT('A+','A ','A-',
'B+','B ','B-','C+','C ','C-','D+','D ','D-','F ');
DCL (DATE,TIME,LENGTH,SUBSTR,VERIFY,ABS,MOD,HBOUND)   BUILTIN;
DCL (TRANSLATE,INDEX,UNSPEC,ONSOURCE,REPEAT,TRUNC)    BUILTIN;
DCL (STRING,FLOOR,COMPLETION,PRIORITY,PLIRETV)        BUILTIN;
DCL L_TST                       FIXED BIN(15) INIT(10);
DCL TST_I                       FIXED BIN(15) INIT(1);
DCL TST_T                       FIXED BIN(15) INIT(2);
DCL TST_NT                      FIXED BIN(15) INIT(3);
DCL TST_R0                      FIXED BIN(15) INIT(4);
DCL TST_R1                      FIXED BIN(15) INIT(5);
DCL TST_R2                      FIXED BIN(15) INIT(6);
DCL TST_R3                      FIXED BIN(15) INIT(7);
DCL TST_OT                      FIXED BIN(15) INIT(8);
DCL TST_OM                      FIXED BIN(15) INIT(9);
DCL TST_RC                      FIXED BIN(15) INIT(10);
DCL L_PST                       FIXED BIN(15) INIT(4);
DCL PST_I                       FIXED BIN(15) INIT(1);
DCL PST_PA                      FIXED BIN(15) INIT(2);
DCL PST_PNA                     FIXED BIN(15) INIT(3);
DCL PST_P                       FIXED BIN(15) INIT(4);
DCL L_TEST_LIM                  FIXED BIN(15) INIT(5);
DCL TRUE                        BIT(1) INIT('1'B);
DCL FALSE                       BIT(1) INIT('0'B);
DCL YN(2)                       CHAR(1) INIT('Y','N');
DCL CQ(2)                       CHAR(1) INIT('C','Q');
DCL RESP                        FIXED BIN(15);
DCL THENTYM                     PICTURE '999999' INIT(1000);
DCL CURTIME                     PICTURE '999999';
DCL CUTOFF                      PICTURE '999999';
DCL CURDATE                     CHAR(6);
DCL OUTDATE                     CHAR(8);
DCL NUMERIC                     CHAR(10) INIT('0123456789');
DCL UPPERCASE                   CHAR(26) INIT(
'ABCDEFGHIJKLMNOPQRSTUVWXYZ' );
DCL LOWERCASE                   CHAR(26) INIT(
'abcdefghijklmnopqrstuvwxyz' );
DCL (BEL,LF,CR,BS,OFF,ON)       CHAR(1);
DCL TURNOFF                     BIT(1) INIT('1'B);
DCL ALLOW                       BIT(1) INIT('0'B) EXTERNAL;
DCL WATCH                       TASK;
DCL STAGE                       EVENT;
 /************ MAIN PROGRAM *************************/
 CALL INIT;
 CALL TERM_CONTROL;

 /*************************************************************/
 /*                                                           */
 /*     THIS PROCEDURE IS THE LOWEST LEVEL TERMINAL INPUT     */
 /*     ROUTINE. THE PARAMETER CIN IS FILLED WITH L           */
 /*     CHARACTERS FROM THE KEYBOARD WHEN THIS ROUTINE IS     */
 /*     CALLED. ALL LOWERCASE ALPHABETIC CHARACTERS ARE       */
 /*     TRANSLATED TO UPPERCASE BY THIS ROUTINE.              */
 /*                                                           */
 /*                                                           */
 /*************************************************************/
TGET:PROC(CIN,L,PROTECT);
   DCL CIN                      CHAR(*);
   DCL L                        FIXED BIN(15);
   DCL PROTECT                  BIT(1);
   DCL T50                      CHAR(50);
   DCL BS52                     CHAR(52);
   DCL C60                      CHAR(60);
   DCL (I,J)                    FIXED BIN(15);
   ON ENDFILE(SYSIN) BEGIN;
      T50='';
      CLOSE FILE(SYSIN);
      OPEN FILE(SYSIN);
      PUT FILE(SYSPRINT) SKIP;
      END;
   IF (PROTECT) THEN DO;
      BS52=REPEAT(BS,52);
      C60=REPEAT('**&&@',12);
      J=MOD(RAND,5)+1;
      DO I=1 TO 4;
         PUT FILE(SYSPRINT) EDIT(SUBSTR(C60,I+J,L),
            SUBSTR(BS52,1,L)) (A,A);
      END;
      PUT FILE(SYSPRINT) EDIT(' ',SUBSTR(BS52,1,L+2),': ')
         (A(L),A,A);
      CALL SCN(STRING(OFF));
      END;
   GET FILE(SYSIN) EDIT(T50) (A(50));
   IF (PROTECT) THEN
      CALL SCN(STRING(ON));
   CIN=SUBSTR(T50,1,L);
```

```
   CIN=TRANSLATE(CIN,UPPERCASE,LOWERCASE);
END; /* TGET */

/*************************************************************/
/*                                                           */
/*   THIS PROCEDURE PRINTS THE VALUE OF A PASSWORD           */
/*   AND ERASES IT FROM THE SCREEN.                           */
/*                                                           */
/*************************************************************/
ECHOPASS:PROC(CIN);
   DCL CIN                          CHAR(*);
   DCL BS8                          CHAR(8);
   BS8=REPEAT(BS,8);
   CALL TPUT('PASSWORD: "'||CIN||'"',1);
   DELAY(1000);
   PUT FILE(SYSPRINT) EDIT(BS,BS8,'*X*OWWMM',BS8,'OOWWMMX*',BS8,
      'WWMM**@O',BS8,'MM*X@&WW',BS8,'         ')(A);
END; /* ECHOPASS */

/*************************************************************/
/*                                                           */
/*      THIS PROCEDURE RETURNS A BOOLEAN VALUE TRUE IF       */
/*      THE STUDENT NUMBER PASSED IN PARAMETER CID           */
/*      MATCHES ANY OF THE SPECIAL STUDENT NUMBERS           */
/*      RECOGNIZED BY THE SYSTEM. (IE INST,EDIT,TA,ALL,      */
/*      MARKER).                                             */
/*                                                           */
/*************************************************************/
SPECIAL_ID:PROC(CID) RETURNS(BIT(1));
   DCL CID                      CHAR(7);
   DCL I                        FIXED BIN(15);
   DCL SPEC                     BIT(1);
   SPEC = FALSE;
   DO I=1 TO L_SPEC;
      IF (SPEC_ID(I)=CID) THEN SPEC=TRUE;
   END;
   RETURN(SPEC);
END; /* SPECIAL ID */

/*****************************************************************/
/*                                                               */
/*    THIS PROCEDURE TAKES THE CHARACTERS PASSED IN COUT AND     */
/*    WRITES THEM TO THE TERMINAL AFTER SKIPPING L LINES.        */
/*                                                               */
/*****************************************************************/
TPUT:PROC(COUT,L);
   DCL L                        FIXED BIN(15);
   DCL COUT                     CHAR(*);
   SELECT(L);
   WHEN(1) PUT FILE(SYSPRINT) EDIT(COUT) (COL(1), A);
   WHEN(0) PUT FILE(SYSPRINT) SKIP(L) EDIT(COUT) (COL(1), A);
   OTHERWISE PUT FILE(SYSPRINT) EDIT(COUT) (COL(1), SKIP(L-1), A);
   END;
END; /* TPUT */

/*************************************************************/
/*                                                           */
/*     THE RAND PROCEDURE GENERATES A RANDOM 32 BIT INTEGER  */
/*     THE PSUEDORANDOM TECHNIQUE USED IS A COMBINATION OF   */
/*     LINEAR CONGRUENTIAL AND SHIFT REGISTER TECHNIQUES.    */
/*                                                           */
/*************************************************************/
(NOFIXEDOVERFLOW):(NOZERODIVIDE):
RAND:PROC RETURNS(FIXED BIN(31));
   DCL SEED                     FIXED BIN(31) INIT(17) STATIC;
   DCL (T1,T2)                  FIXED BIN(31);
   SEED=69069*SEED*TIME;
   UNSPEC(T1)=( (15)'0'B )||SUBSTR(UNSPEC(SEED),1,17);
   T1=T1+SEED;
   T2=(T1*131072)+T1;
   SEED=SEED+T2;
   RETURN(SEED);
END; /* RAND */

/*************************************************************/
/*                                                           */
/*     THE GENRAND PROCEDURE GENERATES A RANDOM INTEGER IN   */
/*     RANGE 1-LIM WHICH EXCLUDES THE NUMBERS IN Q1,Q2,Q3.   */
/*     IT IS USED TO GENERATE THE QUESTION NUMBERS FOR TESTS */
/*     AND IN THE PROCTOR SELECTION ALGORITHM WHERE IT IS    */
/*     USED TO SELECT PROCTORS IF THERE ARE MORE THAN ONE    */
/*     WITH THE LOWEST NUMBER OF POINTS.                     */
/*                                                           */
/*************************************************************/
GENRAND:PROC(LIM,Q1,Q2,Q3) RETURNS(FIXED BIN(15));
   DCL (LIM,Q1,Q2,Q3)           FIXED BIN(15);
   DCL RGEN                     FIXED BIN(15);
   DCL EQ                       BIT(1);
   IF (LIM<2) THEN RETURN(1);
   RGEN=MOD(RAND,LIM)+1;
   IF (LIM=2) THEN RETURN(RGEN);
   EQ=TRUE;
   DO WHILE(EQ);
     IF ((RGEN=Q1)|(RGEN=Q2)) THEN DO;
        IF (RGEN<LIM) THEN
           RGEN=RGEN+1;
        ELSE
           RGEN=1;
        END;
     ELSE EQ=FALSE;
   END;
   RETURN(RGEN);
END; /*GENRAND*/

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

```
/*                                                                  */
/*   THIS PROCEDURE WRITES OUT THE MESSAGE THAT THE STUDENT         */
/*   NUMBER HAS JUST BEEN MODIFIED OR DELETED.                      */
/*                                                                  */
/******************************************************************/
REC_NO_FOUND:PROC;
   CALL TPUT('ERROR IS FOUND IN ' || STUD.ID, 1);
   CALL TPUT('YOUR STUDENT NUMBER HAS JUST BEEN MODIFIED OR', 1);
   CALL TPUT('YOUR STUDENT RECORD HAS JUST BEEN DELETED.', 1);
   CALL TPUT('TRANSACTION CANCELLED.', 1);
   CALL TPUT('PLEASE CONTACT TO YOUR INSTRUCTOR !', 1);
END; /* REC_NO_FOUND */

/******************************************************************/
/*                                                                */
/*      THIS PROCEDURE READS ONE STUDENT RECORD GIVEN THE         */
/*      STUDENT NUMBER IN PARAMETER CID. THE PARAMETER FOUND      */
/*      IS RETURNED AS A BOOLEAN TRUE VALUE IF THE RECORD         */
/*      CORRESSPONDING TO STUDENT NUMBER CID WAS FOUND.           */
/*                                                                */
/******************************************************************/
READ_STUD:PROC(CID,FOUND);
   DCL CID                  CHAR(7);
   DCL FOUND                BIT(1);
   ON KEY(STUDFIL) FOUND=FALSE;
   FOUND=TRUE;
   READ NOLOCK FILE(STUDFIL) INTO(STUD) KEY(CID);
END; /* READ_STUD */

/******************************************************************/
/*                                                                */
/*   THIS PROCEDURE READS AND WRITES ONE STUDENT RECORD           */
/*   GIVEN THE STUDENT NUMBER IN STUD.ID. THE PARAMETER           */
/*   FOUND IS RETURNED AS A BOOLEAN TRUE VALUE IF THE RECORD      */
/*   CORRESSPONDING TO STUDENT NUMBER CID WAS FOUND.              */
/*                                                                */
/******************************************************************/
READ_STUD_LOCK:PROC(FOUND);
   DCL FOUND                BIT(1);
   ON KEY(STUDFIL) BEGIN;
      FOUND=FALSE;
      CALL REC_NO_FOUND;
      END;
   FOUND=TRUE;
   READ FILE(STUDFIL) INTO(STUD) KEY(STUD.ID);
END; /* READ_STUD_LOCK */

/******************************************************************/
/*                                                                */
/*      THIS PROCEDURE REWRITES THE STUDENT RECORD ASSOCI-        */
/*      ATED WITH STUDENT NUMBER PASSED IN CID. FOUND IS THE      */
/*      BOOLEAN VALUE RETURNED TRUE IF THE RECORD WAS FOUND.      */
/*                                                                */
```

```
/******************************************************************/
UPDATE_STUD:PROC(CID,FOUND);
   DCL CID                  CHAR(7);
   DCL FOUND                BIT(1);
   ON KEY(STUDFIL) FOUND=FALSE;
   FOUND = TRUE;
   REWRITE FILE(STUDFIL) FROM(STUD) KEY(CID);
END;/* UPDATE_STUD */

/******************************************************************/
/*                                                                */
/*      THIS PROCEDURE REMOVES A STUDENT RECORD FROM THE          */
/*      THE STUDENT FILES. CID CONTAINS THE STUDENT NUMBER        */
/*      TO BE REMOVED, FOUND IS RETURNED TRUE IF THE RECORD       */
/*      TO BE DELETED WAS FOUND.                                  */
/*                                                                */
/******************************************************************/
REMOV_STUD:PROC(CID,FOUND);
   DCL CID                  CHAR(7);
   DCL FOUND                BIT(1);
   ON KEY(STUDFIL) FOUND=FALSE;
   FOUND=TRUE;
   DELETE FILE(STUDFIL) KEY(CID);
END; /* REMOVE_STUD */

/******************************************************************/
/*                                                                */
/*      THIS PROCEDURE WRITES A NEW STUDENT RECORD TO THE         */
/*      FILE. THE STUDENT NUMBER OF THE NEW RECORD IS             */
/*      CONTAINED IN CID. FOUND IS RETURNED WITH THE              */
/*      VALUE TRUE IF A RECORD WITH THIS STUDENT NUMBER           */
/*      ALREADY EXIST.                                            */
/*                                                                */
/******************************************************************/
WRITE_STUD:PROC(CID,FOUND);
   DCL CID                  CHAR(7);
   DCL FOUND                BIT(1);
   ON KEY(STUDFIL) FOUND=TRUE;
   FOUND=FALSE;
   WRITE FILE(STUDFIL) FROM(STUD) KEYFROM(CID);
END; /* WRITE_STUD */

/******************************************************************/
/*                                                                */
/*     THIS ROUTINE CLOSES THE FILE OF THE SYSTEM.                */
/*     IF SOME RECORDS ARE LOCKED BY ANOTHER TASK, THEN           */
/*     THE ERROR CONDITION WILL BE RAISED. IF SO, DELAY A         */
/*     WHILE AND TRY TO CLOSE THE FILE AGAIN.                     */
/*                                                                */
/******************************************************************/
CLOSE_FILE:PROC(INFILE);
   DCL INFILE               FILE VARIABLE;
```

```
   ON ERROR BEGIN;
      DELAY(10);
      GOTO AGAIN;
      END;
AGAIN: CLOSE FILE(INFILE);
END; /* CLOSE_FILE */

/*****************************************************************/
/*                                                               */
/*     THIS ROUTINE ACCEPTS A 6 CHARACTER VALUE REPRESENTING */
/*     A TIME AS HHMMSS AND RETURNS THE SAME TIME VALUE WITH */
/*     THE HOURS,MINUTES SECONDS SEPARATED BY COLONS.HH:MM:SS*/
/*                                                               */
/*****************************************************************/
 SEPARATE:PROC(TIMEARG) RETURNS(CHAR(8));
    DCL TIMEARG                 PICTURE '999999';
    RETURN(SUBSTR(TIMEARG,1,2)||':'||SUBSTR(TIMEARG,3,2)||
          ':'||SUBSTR(TIMEARG,5,2));
 END; /* SEPARATE */

  /*****************************************************/
  /*                                                   */
  /*  THIS ROUTINE TAKES TWO 6 DIGIT TIME VALUES    */
  /*  REPRESENTIN HHMMSS AND RETURNS A VALUE WHICH  */
  /*  IS THE SUM OF THE HOURS AND MINUTES WITH THE  */
  /*  SECONDS SET TO ZERO.                          */
  /*                                                   */
  /*****************************************************/
 ADDTIME:PROC(TIM1,TIM2) RETURNS(PICTURE'999999');
    DCL (TIM1,TIM2)      PICTURE '999999';
    DCL (I,CARRY,IM)     FIXED BIN(31);
    DCL MOD              BUILTIN;
    DCL (MM,HH)          PICTURE '99';
    I=SUBSTR(TIM1,3,2)+SUBSTR(TIM2,3,2);
    IM=MOD(I,60);
    MM=IM;
    CARRY=I/60;
    I=SUBSTR(TIM1,1,2)+SUBSTR(TIM2,1,2);
    HH=MOD(I+CARRY,24);
    RETURN(HH||MM||'00');
 END; /* ADDTIME */

  /*****************************************************/
  /*                                                   */
  /*   THIS ROUTINE TAKES A 6 DIGIT DATE VALUES      */
  /*   REPRESENTING YYMMDD AND RETURNS THE NEXT DATE */
  /*   IN THE GENERAL CALENDAR.                      */
  /*                                                   */
  /*****************************************************/
 INCRDATE:PROC(INDATE) RETURNS(PICTURE'999999');
    DCL INDATE           PICTURE '999999';
    DCL (YY,MM,DD)       PICTURE '99';
    YY=SUBSTR(INDATE,1,2);
    MM=SUBSTR(INDATE,3,2);
    DD=SUBSTR(INDATE,5,2)+1;
    IF DD>SUBSTR('312931303130313130313031',MM*2-1,2) |
       (MM=2 & DD>28 & YY/4>FLOOR(YY/4)) THEN DO;
       DD=1;
       MM=MOD(MM+1,13);
       IF MM=0 THEN DO;
          MM=1;
          YY=MOD(YY+1,100);
          END;
       END;
    RETURN(YY||MM||DD);
    END;

/*****************************************************************/
/*                                                               */
/*     THIS PROCEDURE BUILDS AND WRITES A LOG RECORD.THE CID  */
/*     AND TRANS PARAMETERS DEFINE THE SID AND TYPE FIELDS TO */
/*     BE WRITTEN WHILE THE REST OF THE FIELDS OF THE LOG     */
/*     ARE GLEANED FROM THE CURRENT VALUES IN THE STUDENT     */
/*     RECORD.                                                */
/*     FOR A DESCRIPTION OF THE LOG RECORD FIELDS SEE THE REF-*/
/*     ERANCE MANUAL.                                         */
/*                                                               */
/*****************************************************************/
LOG:PROC(TRANS,CID);
   DCL TRANS                 CHAR(1);
   DCL CID                   CHAR(7);
   ON KEY(LOGFIL) BEGIN;
      CALL READ_SYSP_LOCK;
      SYSP.NLOGREC=SYSP.NLOGREC-1;
      CALL UPDATE_SYSP;
      CALL TPUT('PLEASE CONTACT TO YOUR INSTRUCTOR !',1);
      CALL TPUT('THE LOG FILE IS FULL AND NO RECORD CAN BE '||
          'WRITTEN ON IT.',1);
      CALL TPUT('THE LOG FILE IS NEEDED TO BE ENLARGED !',1);
      GOTO OUT;
      END;
   CALL READ_SYSP_LOCK;
   SYSP.NLOGREC=SYSP.NLOGREC+1;
   CALL UPDATE_SYSP;
   READ FILE(LOGFIL) INTO(LOGREC) KEY(SYSP.NLOGREC);
   LOGREC.DELKEY=(8)'0'B;
   LOGREC.TYPE=TRANS;
   LOGREC.CTIME=SUBSTR(TIME,1,6);
   LOGREC.SID=CID;
   IF (TRANS='2'|TRANS='3'|TRANS='4'|TRANS='5') THEN DO;
      LOGREC.TS=TST_I;
      LOGREC.PS=PST_I;
      LOGREC.ID=USERID;
      LOGREC.UNIT,LOGREC.Q1,LOGREC.Q2,LOGREC.Q3,LOGREC.TP,
         LOGREC.PP=0;
```

```
      END;
   ELSE DO;
      LOGREC.TS=STUD.TSTATE;
      LOGREC.PS=STUD.PSTATE;
      LOGREC.ID=STUD.ID;
      LOGREC.UNIT=STUD.UNIT;
      LOGREC.Q1=STUD.Q1;
      LOGREC.Q2=STUD.Q2;
      LOGREC.Q3=STUD.Q3;
      LOGREC.TP=STUD.TEST;
      LOGREC.PP=STUD.PROCTOR;
      END;
   REWRITE FILE(LOGFIL) FROM(LOGREC) KEY(SYSP.NLOGREC);
OUT:;
END; /* LOG */

/*****************************************************************/
/*                                                               */
/*   THIS PROCEDURE IS CALLED WHEN THERE IS A SERIOUS            */
/*   INCONSISTANCY IN THE INTERNAL RECORDS. AN ERROR NUMBER      */
/*   DEFINED IN THE TABLE BELOW IS WRITTEN TO THE TERMINAL.      */
/*                                                               */
/*   ERROR   CAUSE                        LOCATION               */
/*   -----   -----                        --------               */
/*   1       STUDENT NUMBER IN ST ARRAY   SESSION CONTROL,       */
/*           NOT FOUND IN STUDENT FILE.   PROCTOR SELECTION.     */
/*   2       INVALID CASE CHOICE.         SESSION CONTROL,       */
/*                                        PROCTOR SELECTION      */
/*   3       SECONDARY ID NOT FOUND.      SESSION CONTROL,       */
/*                                        MARK TEST              */
/*   4       STUDENT WRITING TEST AND     SESSION CONTROL        */
/*           PROCTORING AT THE SAME TIME                         */
/*   5      -SAME AS ERROR 2.             SESSION CONTROL MAIN   */
/*                                        SELECT STATMENT.       */
/*   6      -STUDENT NUMBER NOT FOUND     -STUDENT LOGIN.        */
/*   7      -SAME AS ERROR 6.             -GET_PROC_STATUS       */
/*   8      -SAME AS ERROR 2.             -GET_PROC_STATUS       */
/*   9      -SAME AS ERROR 2.             -GET_MARK              */
/*   10     -SPECIAL STUDENT NUMBER NOT   -GET_EDIT_PASS         */
/*           FOUND.                                              */
/*   11     -SAME AS ERROR 1.             -START_SESSION.        */
/*   12     -SAME AS ERROR 1.             -END SESSION.          */
/*   13     -SAME AS ERROR 6.             -EDIT_PERSONAL         */
/*   14     -SAME AS ERROR 1.             -EDIT_PERSONAL         */
/*   15     -SAME AS ERROR 10.            -EDIT_PERSONAL         */
/*   16     -SAME AS ERROR 10.            -GET_INST_PASS         */
/*   17     -SAME AS ERROR 6.             -EDIT_COURSE           */
/*   18     -SAME AS ERROR 1.             -LIST_STUD             */
/*   19     -SAME AS ERROR 6.             -SESSION CONTROL,TEST  */
/*                                         GENERATE.             */
/*   20     -SAME AS ERROR 6.             -STATE_MACH            */
/*   21     -SAME AS ERROR 5.             -STATE_MACH            */
/*   22     -SAME AS ERROR 1.             -REMOV_STUD            */
/*   23     -SAME AS ERROR 1.             -EDIT_SYSP,CHANGE      */
/*                                         THRESHOLDS.           */
/*   24     -SAME AS ERROR 6.             -GENTEST               */
/*   25     -SAME AS ERROR 6.             -GET_MARK              */
/*                                                               */
/*****************************************************************/
INTERNAL_ERR:PROC(ENUM);
  DCL ENUM                              FIXED BIN(15);
  DCL COUT                              CHAR(24);
  PUT STRING(COUT) EDIT('***INTERNAL ERROR:',ENUM)(A,F(6,0));
  CALL TPUT(COUT,1);
END; /* INTERNAL_ERR */

/*****************************************************************/
/*                                                               */
/*    THIS ROUTINE CONTAINS THE COMMON CODE REQUIRED TO          */
/*    WRITE A PROMPT TO THE TERMINAL, GET A SINGLE CHARACTER     */
/*    RESPONSE BACK AND MATCH TO A KNOWN SET OF RESPONSES        */
/*    THE PARAMETER MES CONTAINS THE PROMPT TO BE LISTED.        */
/*    THE PARAMETER RA IS AN ARRAY OF SINGLE CHARACTER VALID     */
/*    RESPONSES. RETVAL IS THE RETURN VALUE: AN INDEX INTO       */
/*    THE RA ARRAY INDICATING WHICH ONE OF THE RESPONSES WAS     */
/*    TYPED IN. RETVAL IS SET TO 0 IF A BLANK OR RETURN WAS      */
/*    TYPED IN. THE PROGRAM WILL LOOP REQUESTING A RESPONSE      */
/*    UNTIL A VALID ONE IS TYPED IN.                             */
/*                                                               */
/*****************************************************************/
GET_RESP:PROC(MES,RA,RETVAL,DEFAULT);
   DCL MES                          CHAR(*);
   DCL RA(*)                        CHAR(1);
   DCL (RETVAL,DEFAULT,I,LIM)       FIXED BIN(15);
   DCL CONT                         BIT(1);
   DCL TEMP                         CHAR(1);
   CONT = TRUE;
   RETVAL = 0 ;
   LIM = HBOUND(RA,1);
   DO WHILE(CONT);
      TEMP=' ';
      IF (DEFAULT^=0) THEN
         CALL TPUT(MES||RA(DEFAULT)||BS||BS||BS||': ',1);
      ELSE
         CALL TPUT(MES,1);
      CALL TGET(TEMP,1,FALSE);
      DO I=1 TO LIM;
         IF (RA(I)=TEMP) THEN DO;
            RETVAL=I;
            CONT=FALSE;
            END;
      END;
      IF (TEMP=' ') THEN DO;
         RETVAL=0;
         CONT=FALSE;
```

```
         END;
      ELSE DO;
         IF (CONT) THEN DO;
            CALL TPUT('INVALID RESPONSE, TRY AGAIN.',1);
            END;
         END;
   END; /* WHILE */
 END; /*  GET RESPONSE */

/*****************************************************************/
/*                                                               */
/*     THIS PROCEDURE IS EXECUTED BEFORE THE CONTROL MENU        */
/*     IS DISPLAYED. IT PRINTS THE SIGN-ON BANNER AND PRIMES     */
/*     THE CURDATE. THE PROCEDURE WRITES A LOG RECORD WITH       */
/*     TRANSACTION TYPE=2                                        */
/*                                                               */
/*****************************************************************/
INIT:PROC;
   DCL E1                          EVENT;
   CURDATE= DATE;
   OPEN FILE(STUDFIL) EXCLUSIVE UPDATE DIRECT UNBUFFERED,
      FILE(SYSPFIL) UPDATE  DIRECT,
      FILE(LOGFIL) UPDATE DIRECT;
   CALL LOG('2',CURDATE||' ') EVENT(E1);
   UNSPEC(BEL)='00101111'B;
   UNSPEC(LF) ='00100101'B;
   UNSPEC(CR) ='00001101'B;
   UNSPEC(OFF)='00001110'B;
   UNSPEC(ON) ='00001111'B;
   UNSPEC(BS) ='00010110'B;
   IF (USERID=' ') THEN DO;
      CALL TPUT('UNIDENTIFIED USER IS NOT ALLOWED TO USE PSI.',1);
      CALL SCN(STRING(BEL));
      WAIT(E1);
      STOP;
      END;
   ALLOC PINDX,ST,BUFFER;
   COMPLETION(STAGE)='1'B;
   OUTDATE= SUBSTR(CURDATE,1,2)||'/'||SUBSTR(CURDATE,3,2)||
            '/'||SUBSTR(CURDATE,5,2);
   CURTIME=SUBSTR(TIME,1,6);
   CALL TPUT('PERSONALIZED SYSTEM INSTRUCTION.',1);
   CALL TPUT('VERSION 2.0 : OCT 15,1985',1);
   CALL TPUT(OUTDATE||'    '||SEPARATE(CURTIME),1);
   CALL TPUT('STARTING INITIALIZATION NOW.',1);
   WAIT(E1);
END; /* INIT */

/*****************************************************************/
/*                                                               */
/*     THIS PROCEDURE REWRITES THE SYSTEM PARAMETER RECORD*/
/*     IT IS CALLED AFTER EVERY MODIFICATION TO A SYSTEM  */
/*     PARAMETER.                                                */
/*                                                               */
/*****************************************************************/
UPDATE_SYSP:PROC;
   REWRITE FILE(SYSPFIL) FROM(SYSP) KEY(0) ;
END; /* UPDATE_SYSP */

/*****************************************************************/
/*                                                               */
/*    THIS PROCEDURE IS USED TO READ THE SYSTEM PARAMETER */
/*    RECORD.                                                    */
/*                                                               */
/*****************************************************************/
READ_SYSP:PROC;
   READ NOLOCK FILE(SYSPFIL) INTO(SYSP) KEY(0) ;
END; /* READ_SYSP */

/*****************************************************************/
/*                                                               */
/*    THIS PROCEDURE IS USED TO READ AND LOCK THE SYSTEM  */
/*    PARAMETER RECORD.                                          */
/*                                                               */
/*****************************************************************/
READ_SYSP_LOCK:PROC;
   READ FILE(SYSPFIL) INTO(SYSP) KEY(0) ;
END; /* READ_SYSP_LOCK */

/*****************************************************************/
/*                                                               */
/*     THIS PROCEDURE IS USED TO DISCOVER WHICH MENU     */
/*     CHOICE WAS MADE. SINCE A USER IS ALLOWED TO ENTER */
/*     MULTIPLE MENU CHOICES ON A LINE WHICH WILL MOVE   */
/*     DOWNWARD THROUGH THE MENU HIERARCHY WITHOUT THE   */
/*     PRINTING OF INTERVENING MENUS THIS ROUTINE        */
/*     EXTRACTS THE FIRST CHOICE FROM THE OTHERS. (THE   */
/*     CHOICES MUST BE SEPARATED BY PERIODS.) THE PARA-  */
/*     METER SOURCE CONTAINS THE SOURCE STRING TYPED IN  */
/*     BY THE USER, THE PARAMETER FIRST IS THE FIRST     */
/*     CHOICE IN THE LINE, THE PARAMETER REST IS THE REST*/
/*     OF THE LINE WITH THE FIRST CHOICE REMOVED AND THE */
/*     BOOLEAN VARIABLE ERR IS USED TO INDICATED A SYNTAX*/
/*     ERROR IN THE INPUT.                                       */
/*                                                               */
/*****************************************************************/
EXTRACT_CHOICE:PROC(SOURCE,FIRST,REST,ERR);
   DCL (SOURCE,FIRST,REST)  CHAR(*);
   DCL ERR                  BIT(1);
   DCL (I,J)                FIXED BIN(15);
   ERR = FALSE;
   FIRST=' ';
   REST=' ';
   IF ( VERIFY(SOURCE,NUMERIC||'. ')^=0) THEN
      ERR = TRUE; /*SOURCE CONTAINS INVALID CHARS*/
   ELSE DO;
```

```
      J=VERIFY(SOURCE,' '); /* J=FIRST NON-BLANK */
      IF (J=0) THEN DO;
         ERR = TRUE; /* SOURCE EMPTY */
         END;
      ELSE DO;
         FIRST = SUBSTR(SOURCE,J);
         I = VERIFY(FIRST,NUMERIC);
         REST = SUBSTR(FIRST,I+1);
         SUBSTR(FIRST,I) = ' ';
         IF (I=1) THEN ERR=TRUE;
      END; /*ELSE */
   END; /* ELSE */
END; /* EXTRACT CHOICE */

/*****************************************************************/
/*                                                               */
/*     THIS PROCEDURE IS THE ROUTINE WHICH DISPLAYS THE MAIN     */
/*     CONTROL MENU AND PERFORMS A CALL TO A SUBROUTINE          */
/*     DEPENDING ON THE MENU CHOICE MADE.                        */
/*                                                               */
/*****************************************************************/
 TERM_CONTROL:PROC;
    DCL L_CHOICE                    FIXED BIN(15) INIT(10);
    DCL (CHOICE,CHOICE_FIRST,CHOICE_REST)
                                    CHAR(L_CHOICE);
    DCL (ERR,CONTINUE)              BIT(1);
    DCL ENTERSTNUM                  CHAR(27) INIT(
    'ENTER STUDENT NUMBER     : ');
    DCL ENTERNEXT                   CHAR(32) INIT(
    'ENTER NEXT STUDENT NUMBER     : ');
    CONTINUE = TRUE;
    DO WHILE(CONTINUE);
       CALL PRINT_MAIN_MENU;
       CALL TGET(CHOICE,L_CHOICE,FALSE);
       CALL EXTRACT_CHOICE(CHOICE,CHOICE_FIRST,CHOICE_REST,ERR);
       IF (ERR) THEN CHOICE_FIRST = 'X';
       SELECT(CHOICE_FIRST);
       WHEN('1') DO; /* START SESSION */
          CALL START_SESSION(CHOICE_REST);
          END;
       WHEN('2') DO; /* END SESSION */
          CALL GET_INST_PASS(ERR);
          IF (ERR) THEN CALL END_SESSION(CHOICE_REST);
          END;
       WHEN('3') DO; /* EDIT STUDENT */
          CALL GET_EDIT_PASS(ERR);
          IF (^ERR) THEN CALL EDIT_STUD(CHOICE_REST);
          END;
       WHEN('4')DO; /* EDIT SYSTEM PARAMETRS. */
          CALL GET_EDIT_PASS(ERR);
          IF (^ERR) THEN CALL EDIT_SYSP(CHOICE_REST);
          END;
       WHEN('5')DO; /* MARK TEST */
          CALL GET_INST_PASS(ERR);
          IF (ERR) THEN CALL MARK_TEST(CHOICE_REST);
          END;
       WHEN('6') DO; /* SEND MESSAGES */
          CALL SEND_MESS(CHOICE_REST);
          END;
       WHEN('7') DO; /* MONITOR STUDENT */
          CALL GET_INST_PASS(ERR);
          IF (ERR) THEN DO;
             CALL MONITOR(CHOICE_REST);
             IF (ALLOW & ^TURNOFF) THEN DO;
                CALL TPUT('WATCHING IS STILL ON.',1);
                WAIT(STAGE);
                CALL WATCH_LOG TASK(WATCH) EVENT(STAGE)
                   PRIORITY(-200);
                END;
             END;
          END;
       WHEN('8') DO; /* RETURN TO TSO */
          CALL LOG('5',DATE||' ');
          CONTINUE = FALSE;
          END;
       OTHERWISE DO; /* INVALID INPUT */
          CALL TPUT('INVALID CHOICE: '||CHOICE,1);
          END;
       END; /* SELECT */
    END; /* WHILE */
/* LOGICAL END OF TERM_CONT */

    /*****************************************************************/
    /*                                                               */
    /*    THIS PROCEDURE SIMPLY LISTS THE MEAN CONTROL MENU          */
    /*    ON THE TERMINAL.                                           */
    /*                                                               */
    /*****************************************************************/
    PRINT_MAIN_MENU:PROC;
       CALL TPUT('1.START SESSION.   2.END SESSION.',1);
       CALL TPUT('3.EDIT STUDENT.    4.EDIT SYSTEM PARAMETERS.',1);
       CALL TPUT('5.MARK TEST.       6.SEND MESSAGES.',1);
       CALL TPUT('7.MONITOR STUDENT. 8.EXIT TO TSO.',1);
       CALL TPUT('ENTER CHOICE==>    : ',1);
    END; /* PRINT MAIN MENU */

    /*****************************************************************/
    /*                                                               */
    /*       THIS PROCEDURE CONTAINS THE COMMON CODE REQUIRED        */
    /*       TO PROMPT THE TERMINAL FOR A STUDENT NUMBER AND         */
    /*       THEN RECIEVE A REPLY. PARAMETER STNPRMPT IS THE         */
    /*       INPUT STRING THE ROUTINE WILL WRITE TO THE TER-         */
    /*       MINAL  AS A PROMPT. CID CONTAINS THE STUDENT            */
    /*       NUMBER WHICH IS TYPED IN. FOUND,SPEC,BLANKF ARE         */
    /*       BOOLEAN VALUES RETURNED WHICH INDICATE IF THE           */
```

```
/*       STUDENT NUMBER TYPED IN WAS FOUND IN THE RECORDS*/
/*       ,WAS A SPECIAL STUDENT NUMBER (LIKE EDIT,INST ..*/
/*       .. .ETC.)OR WAS TYPED IN AS A BARE RETURN( ALL  */
/*       BLANK).                                         */
/*                                                       */
/*******************************************************/
GETID:PROC(CID,FOUND,SPEC,BLANKF,STNPRMPT);
   DCL STNPRMPT                CHAR(*);
   DCL CID                     CHAR(7);
   DCL (FOUND,SPEC,BLANKF)     BIT(1);
   FOUND= FALSE;
   SPEC=FALSE;
   BLANKF=FALSE;
   CALL TPUT(STNPRMPT,2);
   CALL TGET(CID,7,FALSE);
   IF (CID=' ') THEN DO;
      BLANKF=TRUE;
      RETURN;
      END;
   IF (SPECIAL_ID(CID)) THEN DO;
      SPEC=TRUE;
      RETURN;
      END;
   FOUND=TRUE;
   CALL READ_STUD(CID,FOUND);
END; /* GET ID */

/*******************************************************/
/*                                                     */
/*       THIS PROCEDURE CONTAINS THE COMMON CODE       */
/*       REQUIRED TO READ IN AN INTEGER FROM THE       */
/*       TERMINAL. IT RETURNS A BINARY VALUE OF THE    */
/*       CHARACTERS TYPED IN AND STORES IN THE         */
/*       PARAMETER ITEMP.                              */
/*       THE BOOLEAN VALUES RETURNED IN ERR AND BLANKF */
/*       INDICATE WHETHER THE VALUE TYPED IN COULD BE  */
/*       CONVERTED CORRECTLY AND IF IT WAS TYPED IN AS */
/*       BARE RETURN(ALL BLANKS) OR NOT                */
/*                                                     */
/*******************************************************/
GET_INT:PROC(ITEMP,ERR,BLANKF);
   DCL ITEMP                   FIXED BIN(15);
   DCL (ERR,BLANKF)            BIT(1);
   DCL TEMPIN                  CHAR(7);
   ON CONVERSION BEGIN;
      ERR = TRUE;
      TEMPIN= '0';
      ITEMP=0;
      ONSOURCE='0';
      END;
   ON SIZE   ERR=TRUE;
   ERR=FALSE;
   BLANKF=FALSE;
   ITEMP=0;
   CALL TGET(TEMPIN,7,FALSE);
   IF (TEMPIN=' ') THEN DO;
      BLANKF=TRUE;
      RETURN;
      END;
(SIZE): ITEMP=TEMPIN;
   END; /* GET INT*/

   /*****************************************************/
   /*                                                   */
   /*       THIS PROCEDURE IS THE REAL VALUE ANALOG     */
   /*       OF GET_INT. IT INPUTS A REAL VALUE IN       */
   /*       CHARACTER FORM AND RETURNS A FIXED DECIMAL  */
   /*       TRANSLATION IN THE PARAMETER RTEMP. AS      */
   /*       BEFORE THE PARAMETERS ERR, AND BLANKF ARE   */
   /*       RETURNED TO INDICATE THE INPUT OF BAD DATA  */
   /*       OR A BARE RETURN.                           */
   /*                                                   */
   /*****************************************************/
   GET_REAL:PROC(RTEMP,ERR,BLANKF);
      DCL RTEMP                FIXED DEC(7,3);
      DCL (ERR,BLANKF)         BIT(1);
      DCL TEMPIN               CHAR(8);
      ON CONVERSION BEGIN;
         ERR=TRUE;
         TEMPIN='0';
         RTEMP=0;
         ONSOURCE='0';
         END;
      ON SIZE   ERR=TRUE;
      ERR=FALSE;
      BLANKF=FALSE;
      RTEMP=0;
      CALL TGET(TEMPIN,8,FALSE);
      IF (TEMPIN=' ') THEN DO;
         BLANKF=TRUE;
         RETURN;
         END;
(SIZE): RTEMP=TEMPIN;
   END; /* GET REAL */

   /*****************************************************/
   /*                                                   */
   /*    THIS PROCEDURE LOOKS AT THE TEST,PROCTOR,TERM AND */
   /*    EXAM POINT FIELDS OF THE CURRENT STUDENT RECORD  */
   /*    FORMING A SUM AND PLACING IT IN THE TOTAL POINTS */
   /*    FIELD. AFTER THAT A NEW LETTER GRADE IS CALCULATED*/
   /*    AND ASSIGNED DEPENDING ON THE TOTAL POINT VALUE. */
   /*    THIS PROCEDURE IS CALLED EVERY TIME THE MARKS OF A*/
   /*    STUDENT IS CHANGED OR THE LETTER GRADES THRESHOLDS*/
   /*    ARE CHANGED.                                     */
```

```
/*                                                              */
/****************************************************************/
(NOSIZE):TOTAL_MARKS:PROC;
   DCL I                          FIXED BIN(15);
   CALL READ_SYSP;
   STUD.TOTAL=STUD.TEST+STUD.PROCTOR+STUD.TERM+STUD.EXAM;
   STUD.LETTER=LGLIT(13);
   DO I=13 TO 1 BY -1;
      IF (STUD.TOTAL>=SYSP.LGTHRESH(I)) THEN
         STUD.LETTER=LGLIT(I);
   END;
END; /* TOTAL MARKS */

/****************************************************************/
/*                                                              */
/*     THIS IS A LOW LEVEL ROUTINE CALLED FROM MANY  */
/*     PLACES IN THE EDITOR TO PROMPT FOR AND REPLACE*/
/*     AN INTEGER VALUED FIELD IN A RECORD. THE      */
/*     CALLING ROUTINE PLACES A FIELD IDENTIFIER     */
/*     STRING IN THE PARAMETER MSG WHICH IS TACKED ON*/
/*     TO THE PART OF THE PROMPT WHICH IS COMMON FOR */
/*     ALL FIELDS (IE CURRENT VALUE OF ...MSG...,    */
/*     ENTER NEW VALUE OF ....MSG....). THE PROCEDURE*/
/*     LOOPS REQUESTING A REPLACEMENT VALUE UNTIL A  */
/*     VALID ONE IS ENTERED. THE PARAMETER VAL       */
/*     THE BINARY REPRESENTATION OF THE REPLACEMENT  */
/*     VALUE WHICH IS RETURNED TO THE CALLING PRO-   */
/*     CEDURE , RANGE IS THE VALID RANGE ALLOWED FOR */
/*     THE NUMBER BEING TYPED IN (1<= REPLACEMENT    */
/*     VALUE <= RANGE).                              */
/*     BLANKF IS RETURNED TRUE IF A RETURN WAS TYPED */
/*     AS A REPLACEMENT VALUE.                       */
/*                                                   */
/*****************************************************/
FPROMPT_NUMI:PROC(VAL,MSG,RANGE,BLANKF);
   DCL MSG                        CHAR(*);
   DCL (ERR,BLANKF)               BIT(1);
   DCL (VAL,ITEMP,TVAL,RANGE)     FIXED BIN(15);
   DCL ITEMPO                     PICTURE 'ZZZ9';
   ERR= TRUE;
   ITEMPO=VAL;
   CALL TPUT('CURRENT VALUE OF '||MSG||' IS:'||ITEMPO,1);
   DO WHILE(ERR);
      CALL TPUT('ENTER NEW VALUE OF '||MSG||'     : ',1);
      CALL GET_INT(TVAL,ERR,BLANKF);
      IF (^ERR) THEN DO;
         IF (^BLANKF) THEN
            IF ((TVAL<1) | (TVAL>RANGE)) THEN DO;
               CALL TPUT('NUMBER OUT OF RANGE.',1);
               ERR=TRUE;
               END;
            ELSE DO;
               VAL = TVAL;
               END;
         END;
      IF (ERR) THEN CALL TPUT('INVALID NUMBER, RE-ENTER.',1);
   END;
END; /* FPROMPT_NUMI */

/****************************************************************/
/*                                                              */
/*     THIS ROUTINE IS THE ANALOG OF FPROMPT_NUMI       */
/*     EXCEPT THAT IT IS USED TO GET A REPLACEMENT      */
/*     VALUE FOR A REAL NUMBER FIELD. THERE IS NO       */
/*     RANGE PARAMETER.                                 */
/*                                                      */
/********************************************************/
FPROMPT_NUMR:PROC(VAL,MSG,BLANKF);
   DCL (VAL,TVAL)                 FIXED DEC(7,3);
   DCL MSG                        CHAR(*);
   DCL (ERR,BLANKF)               BIT(1);
   DCL RTEMPO                     PICTURE 'ZZZZ9V.999';
   ERR = TRUE;
   RTEMPO=VAL;
   CALL TPUT('CURRENT VALUE OF '||MSG||':'||RTEMPO,1);
   DO WHILE(ERR);
      CALL TPUT('ENTER NEW VALUE OF '||MSG||'     : ',1);
      CALL GET_REAL(TVAL,ERR,BLANKF);
      IF ((^ERR) & (^BLANKF) ) THEN DO;
         VAL=TVAL;
         END;
      IF (ERR) THEN CALL TPUT ('INVALID NUMBER, RE-ENTER.',1);
   END; /* WHILE */
END; /* FPROMPT_NUMR */

/****************************************************************/
/*                                                              */
/*     THIS PROCEDURE IS USED TO PROMPT FOR AND VERIFY    */
/*     THE EDIT PASSWORD WHEN IT IS ENTERED FROM THE      */
/*     TERMINAL                                           */
/*                                                        */
/**********************************************************/
GET_EDIT_PASS:PROC(ERR);
     ®®® confidential procedure ®®®
END; /* GET EDIT PASS */

/****************************************************************/
/*                                                              */
/*     THIS PROCEDURE PROMPTS THE TERMINAL FOR THE        */
/*     INST PASSWORD AND THEN VERIFYS IT. IF THE USER     */
/*     HAS ENTERED THE CORRECT PASSWORD THE AUTH PARA-    */
/*     METER IS SET TO TRUE. (FALSE OTHERWISE).           */
/*                                                        */
/**********************************************************/
```

```
GET_INST_PASS:PROC(AUTH);
    *** confidential procedure ***
END; /* GET INST PASS */

/*************************************************************/
/*                                                           */
/*     THIS PROCEDURE PERFORMS THE START SESSION FUNCTION*/
/*     OF THE MAIN CONTROL MENU. IT DISPLAYS THE CURRENT */
/*     TEST CUTOFF TIME AND ALLOWS THE USER TO CHANGE IT */
/*     AND IT ALLOWS THE USER TO RESET ALL STUDENTS TO   */
/*     THEIR INITIAL STATE (NEW SESSION) OR RETAIN THEIR */
/*     OLD STATUS (CONTINUE OLD SESSION) BEFORE IT       */
/*     INVOKES THE ROUTINE WHICH CONTROLS STUDENT TRANS- */
/*     ACTION HANDLING. IF A NEW SESSION IS CHOSEN ALL   */
/*     EXISTING TESTS WETHER PARTIALLY MARKED OR UNMARKED*/
/*     WILL BE DISCARDED AND ALL PROCTORS WILL BE LOGGED */
/*     OFF.                                              */
/*                                                       */
/*************************************************************/
START_SESSION:PROC(CHOICE);
   DCL CHOICE                  CHAR(*);
   DCL (ERR,TLOOP,FOUND)       BIT(1);
   DCL (CCURTIME,COFFTIME)     CHAR(8);
   DCL NEWCUTOFF               CHAR(8) INIT(' ');
   DCL YN(2)                   CHAR(1) INIT('Y','N');
   DCL COMSG                   CHAR(52) INIT(
   'DO YOU WISH TO ALTER CUT-OFF TIME? (Y)ES,(N)O     : ');
   DCL CN(2)                   CHAR(1) INIT('C','N');
   DCL CNMSG                   CHAR(52)INIT(
   '(C)ONTINUE LAST SESSION OR START A (N)EW ONE?     : ');
   DCL RESP                    FIXED BIN(15);
   DCL (START,NOW,STOP)        PICTURE '(12)9';
   DCL (SAVE_DATE,SAVE_TIME,SAVE_OFF) PICTURE '999999';
   DCL E1                              EVENT;
   CURDATE=DATE;
   CURTIME=SUBSTR(TIME,1,6);
   CALL READ_SYSP;
   SAVE_DATE=SYSP.SESSDATE;
   SAVE_TIME=SYSP.SESSTIME;
   SAVE_OFF=SYSP.SESSCUTOFF;
   IF (SYSP.SESSTIME>SYSP.SESSCUTOFF) THEN
      STOP=(SYSP.SESSDATE+1)||SYSP.SESSCUTOFF;
   ELSE
      STOP=SYSP.SESSDATE||SYSP.SESSCUTOFF;
   START=SYSP.SESSDATE||SYSP.SESSTIME;
   NOW=CURDATE||CURTIME;
   IF (START<=NOW) & (NOW<STOP) THEN DO;
      CUTOFF=SYSP.SESSCUTOFF;
      CALL TPUT('THIS IS A CONTINUATION OF LAST SESSION.',1);
      END;
   ELSE DO;
      CUTOFF=ADDTIME(CURTIME,20000);
      CALL TPUT('A NEW SESSION HAS NOT BEEN STARTED.',1);
      END;
   CCURTIME=SEPARATE(CURTIME);
   COFFTIME=SEPARATE(CUTOFF);
   CALL TPUT('CURRENT TIME IS: '||CCURTIME,1);
   CALL TPUT('CURRENT CUT-OFF TIME IS: '||COFFTIME,1);
   CALL GET_RESP(COMSG,YN,RESP,2);
   IF (RESP=1) THEN DO;
      TLOOP=TRUE;
      DO WHILE(TLOOP);
        CALL TPUT('ENTER NEW CUT-OFF (HH:MM:SS) TIME     : ',1);
         CALL TGET(NEWCUTOFF,8,FALSE);
         IF (VERIFY(NEWCUTOFF,' ')=0) THEN DO;
            TLOOP = FALSE; /*STOP INPUT LOOP*/
            END;
         ELSE DO;
            CALL PARSE_TIME(NEWCUTOFF,ERR);
            IF (^ERR) THEN DO;
               CUTOFF=SUBSTR(NEWCUTOFF,1,2)||
                      SUBSTR(NEWCUTOFF,4,2)||
                      SUBSTR(NEWCUTOFF,7,2);
               IF (SYSP.SESSTIME>CUTOFF) THEN
                  CALL TPUT('CUT-OFF DATE '||
                     SEPARATE_DATE(INCRDATE(SYSP.SESSDATE))||
                     ' '||'AND TIME '||SEPARATE(CUTOFF)||
                     ' ARE ASSIGNED TO SYSTEM.',1);
               ELSE
                  CALL TPUT('CUT-OFF DATE '||
                     SEPARATE_DATE(SYSP.SESSDATE)||
                     ' '||'AND TIME '||SEPARATE(CUTOFF)||
                     ' ARE ASSIGNED TO SYSTEM.',1);
               TLOOP=FALSE;
            END;
         END; /* ELSE*/
      END; /* WHILE */
   END; /*IF */
   ERR = FALSE;
   CALL READ_SYSP_LOCK;
   IF SYSP.SESSCUTOFF^=SAVE_OFF THEN DO;
      CALL TPUT('WARNING: SESSION CUT-OFF TIME HAS JUST BEEN '||
         'CHANGED,',1);
      CALL TPUT('THE ABOVE CUT-OFF TIME IS ASSIGNED TO THE '||
         'SYSTEM.',1);
      END;
   SYSP.SESSDATE=CURDATE;
   SYSP.SESSTIME=CURTIME;
   SYSP.SESSCUTOFF=CUTOFF;
   CALL UPDATE_SYSP;
   CALL GET_RESP(CNMSG,CN,RESP,1);
   IF (RESP=2) THEN DO; /* START NEW SESSION. */
      CALL LOG('3',CURDATE||' ') EVENT(E1);
      CALL READ_SYSP_LOCK;
      SYSP.CUR_I_TEST=0;
```

```
      CALL UPDATE_SYSP;
      ON ENDFILE(STUDFIL) GOTO EOF;
      CALL CLOSE_FILE(STUDFIL);
      OPEN FILE(STUDFIL) UPDATE SEQUENTIAL BUFFERED;
      DO WHILE('1'B);
         READ FILE(STUDFIL) INTO(STUD);
         STUD.TSTATE=TST_I;
         STUD.PSTATE=PST_I;
         STUD.RTIME=0;
         REWRITE FILE(STUDFIL) FROM(STUD);
      END;
EOF:  CALL CLOSE_FILE(STUDFIL);
      OPEN FILE(STUDFIL) EXCLUSIVE UPDATE DIRECT UNBUFFERED;
      CALL TPUT('ALL STUDENTS SET TO INITIAL STATE.',1);
      WAIT(E1);
   END; /* IF */

   COFFTIME = SEPARATE(CUTOFF);
   CALL TPUT('TEST ISSUE CUT-OFF TIME WILL BE: '||COFFTIME,1);
   CALL TPUT('SWITCH TERMINAL TO FULL DUPLEX NOW.',1);
   CALL SESSION_CONTROL;
   /*  LOGICAL END OF START SESSION */

   /*****************************************************/
   /*                                                   */
   /*    THIS PROCEDURE IS CALLED FROM START SESSION AFTER*/
   /*    A NEW CUTOFF TIME IS ENTERED TO VERIFY THAT THE  */
   /*    TIME TYPED IN A CORRECT FORMAT. THE TIME VALUE TO*/
   /*    BE VERIFIED IS PASSED IN TIMEARG WHILST THE      */
   /*    BOOLEAN VALUE ERR IS RETURNED INDICATING IF AN   */
   /*    ERROR WAS DISCOVERED.                            */
   /*                                                   */
   /*****************************************************/
   PARSE_TIME:PROC(TIMEARG,ERR);
      DCL TIMEARG                CHAR(8);
      DCL ERR                    BIT(1);
      DCL TIME                   CHAR(8) VARYING;
      TIME=TIMEARG;
      IF (INDEX(TIME,':')=0) THEN DO;
         IF (SUBSTR(TIME,2)=' ') THEN
            TIME='0'||TIME;
         SUBSTR(TIME,3)=':00:00';
         END;
      ELSE DO;
         IF (SUBSTR(TIME,2,1)=':') THEN
            TIME='0'||TIME;
         IF (INDEX(SUBSTR(TIME,4),':')=0) THEN DO;
            IF (SUBSTR(TIME,5)=' ') THEN
               TIME=SUBSTR(TIME,1,3)||'0'||SUBSTR(TIME,4);
               SUBSTR(TIME,6)=':00';
            END;
         ELSE DO;
            IF (SUBSTR(TIME,5,1)=':') THEN
               TIME=SUBSTR(TIME,1,3)||'0'||SUBSTR(TIME,4);
            IF (SUBSTR(TIME,8)=' ') THEN
               TIME=SUBSTR(TIME,1,6)||'0'||SUBSTR(TIME,7);
            END;
         END;
      ERR=FALSE;
      ERR = CHECK_DIG(SUBSTR(TIME,1,2),'24');
      ERR = ERR | CHECK_DIG(SUBSTR(TIME,4,2),'60');
      ERR = ERR | CHECK_DIG(SUBSTR(TIME,7,2),'60');
      ERR = ERR | (SUBSTR(TIME,3,1)^=':') |
                  (SUBSTR(TIME,6,1)^=':');
      IF (ERR) THEN
         CALL TPUT('INVALID TIME: '||TIMEARG,1);
      ELSE
         TIMEARG=TIME;
      CHECK_DIG:PROC(DIG2,LIM) RETURNS(BIT(1));
         DCL (DIG2,LIM)      CHAR(2);
         DCL ERR             BIT(1);
         ERR = FALSE;
         IF (VERIFY(DIG2,NUMERIC)=0 ) THEN DO;
            IF (DIG2 >= LIM ) THEN DO;
               ERR = TRUE;
               END;
            END;
         ELSE ERR= TRUE;
         RETURN(ERR);
      END; /* CHECK_DIG */
   END; /*PARSE TIME*/
END; /* START SESSION */

/*****************************************************/
/*                                                   */
/*    THIS PROCEDURE PERFORMS THE END SESSION FUNCTION  */
/*    OF THE MAIN CONTROL MENU. THE ROUTINE GOES THROUGH*/
/*    ALL THE STUDENT RECORDS PRINTING A LIST OF STUDENT*/
/*    WHO HAVE A TEST WHICH HAS NOT BEEN COMPLETELY     */
/*    MARKED. ALSO ALL STUDENTS WHO HAVE INDICATED THEY */
/*    ARE AVAILABLE FOR PROCTORING ARE SET TO THE NOT   */
/*    AVAILABLE STATE.                                  */
/*                                                   */
/*****************************************************/
END_SESSION:PROC(CHOICE);
   DCL OUT_UNIT               PICTURE 'ZZ9';
   DCL (OUT_Q1,OUT_Q2,OUT_Q3) PICTURE 'ZZZ9';
   DCL CHOICE                 CHAR(*);
   DCL OUTST                  FIXED BIN(15);
   DCL E1                     EVENT;
   DCL FOUND                  BIT(1);
   CALL READ_SYSP;
   IF SYSP.SESSTIME=SYSP.SESSCUTOFF THEN
      CALL TPUT('THIS SESSION HAS ALREADY BEEN ENDED.',1);
   ELSE DO;
```

```
      CURTIME=SUBSTR(TIME,1,6);
      CALL READ_SYSP_LOCK;
      SYSP.SESSTIME=CURTIME;
      SYSP.SESSCUTOFF=CURTIME;
      SYSP.CUR_I_TEST=0;
      CALL UPDATE_SYSP;
      CALL TPUT('STUDENTS WHICH HAVE UNMARKED TESTS.',1);
      CALL LOG('4',DATE||' ') EVENT(E1);
      ON ENDFILE(STUDFIL) GOTO EOF;
      CALL CLOSE_FILE(STUDFIL);
      OPEN FILE(STUDFIL) UPDATE SEQUENTIAL BUFFERED;
      DO WHILE('1'B);
         READ FILE(STUDFIL) INTO(STUD);
         OUTST=STAT_TBL_END_SESS.TSTATE(STUD.TSTATE);
         STUD.PSTATE=STAT_TBL_END_SESS.PSTATE(STUD.PSTATE);
         IF ((OUTST=TST_OT)|(OUTST=TST_OM)) THEN DO;
            OUT_UNIT = STUD.UNIT+1;
            OUT_Q1   = ABS(STUD.Q1);
            OUT_Q2   = ABS(STUD.Q2);
            OUT_Q3   = ABS(STUD.Q3);
          CALL TPUT('STUDENT: '||STUD.NAME||', STUDENT#: '
            ||STUD.ID||', UNIT#: '||OUT_UNIT||', QUESTIONS: '||
            OUT_Q1||','||OUT_Q2||','||OUT_Q3||', AT TIME: '||
            SEPARATE(STUD.RTIME),1);
            END; /* IF */
         ELSE DO;
            STUD.RTIME=0;
            END;
         REWRITE FILE(STUDFIL) FROM(STUD);
      END;
 EOF: CALL CLOSE_FILE(STUDFIL);
      OPEN FILE(STUDFIL) EXCLUSIVE UPDATE DIRECT UNBUFFERED;
      WAIT(E1);
      END;
END; /* END SESSION */

/*************************************************************/
/*                                                           */
/*     THIS PROCEDURE DISPLAYS THE EDIT STUDENT SUB-MENU     */
/*     AND THEN CALLS THE REQUIRED SUBROUTINE TO PERFORM     */
/*     THE MENU CHOICE MADE.                                 */
/*                                                           */
/*************************************************************/
EDIT_STUD:PROC(CHOICE);
   DCL CHOICE                    CHAR(*);
   DCL (CONTINUE,SKIPIN,ERR)     BIT(1);
   DCL (FIRST,REST)              CHAR(L_CHOICE);
   SKIPIN = FALSE;
   IF (CHOICE^= ' ') THEN DO;
      SKIPIN=TRUE;
      END;
   CONTINUE = TRUE;
   DO WHILE(CONTINUE);
      IF (^SKIPIN) THEN DO;
         CALL PRINT_MAIN_EDIT_MENU;
         CALL TGET(CHOICE,L_CHOICE,FALSE);
         IF (TURNOFF & (CHOICE=' ')) THEN RETURN;
         END;
      SKIPIN=FALSE ;
      CALL EXTRACT_CHOICE(CHOICE,FIRST,REST,ERR);
      IF (ERR) THEN FIRST='X';
      SELECT(FIRST);
      WHEN('1') DO; /* CREATE */
         CALL CREAT_STUD(REST);
         END;
      WHEN('2') DO; /*DELETE*/
         CALL DELETE_STUD(REST);
         END;
      WHEN('3') DO; /* EDIT PERSONAL */
         CALL EDIT_PERSONAL(REST);
         END;
      WHEN('4') DO; /* EDIT COURSE */
         CALL EDIT_COURSE(REST);
         END;
      WHEN('5') DO; /* LIST STUD */
         CALL LIST_STUD(REST);
         END;
      WHEN('0','6') DO; /* EXIT TO MAIN MENU */
          CONTINUE = FALSE;
          END;
      OTHERWISE DO; /* BAD CHOICE */
         CALL TPUT('INVALID CHOICE: '||CHOICE,1);
         END;
      END; /*SELECT */
      CALL TPUT('ENDING AT: '||SEPARATE(SUBSTR(TIME,1,6)),1);
END; /* DO LOOP */

/*************************************************************/
/*                                                           */
/*    THIS IS A SUBROUTINE OF THE EDIT_STUD PRO-             */
/*    CEDURE. IRT SIMPLY PRINTS OUT THE MENU .               */
/*                                                           */
/*************************************************************/
PRINT_MAIN_EDIT_MENU:PROC;
   CALL TPUT('<3.>1 CREATE STUDENT.        '||
             '<3.>2 DELETE STUDENT.',1);
   CALL TPUT('<3.>3 MODIFY PERSONAL DATA. '||
             '<3.>4 MODIFY COURSE DATA.',1);
   CALL TPUT('<3.>5 LIST STUDENT.          '||
             '<3.>6 RETURN TO MAIN MENU.',1);
   CALL TPUT('ENTER CHOICE==>    : ',1);
END; /* PRINT MAIN EDIT MENU */

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

```
/*     THIS ROUTINE PERFORMS THE STUDENT RECORD      */
/*     CREATION FUNCTION OF THE STUDENT EDITOR.      */
/*     IT GOES INTO A LOOP , PROMPTING FOR A NEW     */
/*     STUDENT NUMBER (FOR THE NEW RECORD) AND ALL   */
/*     THE STUDENT PERSONAL DATA FIELDS. THE COURSE  */
/*     DATA FIELDS ARE ALL SET TO 0 AND THEN THE NEW */
/*     RECORD IS WRITTEN. THE PROCEDURE CONTINUES    */
/*     LOOPING UNTIL A BARE RETURN IS ENTERED AS A   */
/*     NEW STUDENT NUMBER.                           */
/*                                                   */
/*****************************************************/
CREAT_STUD:PROC(CHOICE);
   DCL CHOICE                  CHAR(*);
   DCL (CONT,ERR,FOUND)        BIT(1);
   DCL CID                     CHAR(7);
   CONT = TRUE;
   DO WHILE(CONT);
      ERR= FALSE;
      CALL TPUT('ENTER NEW STUDENT NUMBER      : ',2);
      CALL TGET(CID,7,FALSE);
      IF (CID = ' ') THEN DO;
         CONT = FALSE;
         ERR  = TRUE;
         END;
      ELSE DO;
         IF (SPECIAL_ID(CID)) THEN DO;
            CALL TPUT('STUDENT#: '||CID||' IS RESERVED'||
            ' FOR INTERNAL USE.',1);
            ERR = TRUE;
            END;
         ELSE DO;
            FOUND = FALSE;
            CALL READ_STUD(CID,FOUND);
            IF (FOUND) THEN DO;
               CALL TPUT('STUDENT RECORD ALREADY EXISTS,'||
               ' CANNOT BE CREATED.',1);
               ERR = TRUE;
               END;
            END;
      END; /* ELSE */
      IF (CONT &(^ERR)) THEN DO;
         CALL TPUT('STUDENT NAME(UP TO 30 CHARACTERS)'||
              '      : ',1);
         CALL TGET(STUD.NAME,30,FALSE);
         CALL TPUT('PHONE NUMBER(7 CHARACTERS)     : ',1);
         CALL TGET(STUD.PHONE,7,FALSE);
         CALL TPUT('FACULTY CODE (2 CHARACTERS)    : ',1);
         CALL TGET(STUD.FACULTY,2,FALSE);
         CALL TPUT('YEAR AT UNIVERSITY(2 CHARACTERS)'||
              '      : ',1);
         CALL TGET(STUD.YEAR,2,FALSE);
         CALL TPUT('STATUS IN COURSE(3 CHARACTERS)     : ',
              1);
         CALL TGET(STUD.STATUS,3,FALSE);
         CALL TPUT('STUDENT PASSWORD     : ',1);
         CALL TGET(STUD.PASSWORD,8,TRUE);
         CALL ECHOPASS(STUD.PASSWORD);
         STUD.UNIT,STUD.TEST,STUD.PROCTOR,STUD.TERM,
         STUD.EXAM,STUD.TOTAL,STUD.Q1,STUD.Q2,STUD.Q3=0;
         STUD.RTIME=0 ;
         STUD.LETTER,STUD.SID=' ';
         STUD.TSTATE=TST_I;
         STUD.PSTATE=PST_I;
         STUD.ID=CID;
         CALL WRITE_STUD(CID,FOUND);
      END; /* IF*/
   END; /*WHILE*/
END; /* CREATE STUDENT */

/*****************************************************/
/*                                                   */
/*     THIS PROCEDURE REMOVES STUDENT RECORDS FROM   */
/*     FILE. THE PROCEDURE LOOPS , DELETING STUDENT  */
/*     RECORDS UNTIL A BARE RETURN IS ENTERED AS A   */
/*     STUDENT NUMBER.                               */
/*                                                   */
/*****************************************************/
DELETE_STUD:PROC(CHOICE);
   DCL CHOICE                  CHAR(*);
   DCL CID                     CHAR(7);
   DCL (FOUND,ERR,BLANKF)      BIT(1);
   DCL MSG                     CHAR(27) INIT(
      'DO YOU WANT THIS STUDENT#: ');
   DCL MSG2                    CHAR(40) INIT(
      ' TO BE (D)ELETED OR (M)AINTAINED?     : ');
   DCL DM(2)                   CHAR(1) INIT('D','M');
   DCL RESP                    FIXED BIN(15);
   CALL GETID(CID,FOUND,ERR,BLANKF,ENTERSTNUM);
   IF (FOUND & (^ERR) & (^BLANKF)) THEN DO;
      CALL GET_RESP(MSG||CID||MSG2,DM,RESP,2);
      IF (RESP=1) THEN DO;
         CALL REMOV_STUD(CID,FOUND);
         CALL TPUT('STUDENT#: '||CID||' WAS DELETED FROM '||
              'FILES.',1);
         END;
      ELSE DO;
         CALL TPUT('STUDENT#: '||CID||' WAS MAINTAINED.',1);
         END;
      END;
   ELSE IF ((^FOUND) & (^BLANKF)) THEN DO;
     CALL TPUT('STUDENT#: '||CID||' NOT FOUND.',1);
     END;
     ELSE IF (ERR) THEN
       CALL TPUT(CID||' IS A SPECIAL RESERVED STUDENT#',1);
    /* LOGICAL END OF DELETE STUD. */
```

```
   END; /* DELETE STUD */

   /*********************************************************/
   /*                                                       */
   /*   THIS PROCEDURE GENERATES THE MENU SHOWING THE       */
   /*   PERSONAL DATA FIELDS OF THE STUDENT RECORDS         */
   /*   WHICH MAY BE ALTERED.THE PROCEDURE LOOPS GETING     */
   /*   THE STUDENT NUMBER OF THE NEXT STUDENT RECORD TO    */
   /*   BE ALTERED (UNTIL A BARE RETURN IS ENTERED).        */
   /*   AFTER A MENU CHOICE IS MADE THE USER IS PROMPTED    */
   /*   FOR THE REPLACEMENT VALUE OF THE SPECIFIC FIELD     */
   /*   TO BE ALTERED. THE CHANGING OF PASSWORDS IS MUCH    */
   /*   THE SAME AS THE REPLACEMENT OF OTHER FIELDS         */
   /*   EXCEPT THAT IF IT IS THE INST OR EDIT PASSWORD      */
   /*   BEING ALTERED THEN THE USER IS FIRST PROMPTED TO    */
   /*   ENTER THE EXISTING INST PASSWORD BEFORE HE IS       */
   /*   ALLOWED TO ENTER A NEW PASSWORD.                    */
   /*                                                       */
   /*********************************************************/
     EDIT_PERSONAL:PROC(CHOICE);
        DCL CHOICE              CHAR(*);
        DCL (FIRST,REST)        CHAR(L_CHOICE);
        DCL (FOUND,SPEC,BLANKF,CONTINUE,SKIPIN,ERR,AUTH)
                                BIT(1);
        DCL (CID,NEWID)         CHAR(7);
        DCL TEMPIN              CHAR(30);
        DCL I                   FIXED BIN(15);
        SKIPIN = FALSE;
        IF (CHOICE ^= ' ') THEN SKIPIN=TRUE;
        CONTINUE=TRUE;
REPEAT: DO WHILE(CONTINUE);
           ERR=FALSE;
           CALL GETID(CID,FOUND,SPEC,BLANKF,ENTERSTNUM);
           IF (BLANKF) THEN DO;
              CONTINUE=FALSE;
              ERR = TRUE;
              END;
           ELSE DO;
             IF (^FOUND & ^SPEC) THEN DO;
                ERR=TRUE;
                CALL TPUT('STUDENT#: '||CID||' NOT FOUND.',1);
                END;
             ELSE DO;
                IF (^SKIPIN) THEN DO;
                   CALL TPUT('<3.3.>1 NAME.              '||
                             '<3.3.>2 STUDENT NUMBER.',1);
                   CALL TPUT('<3.3.>3 PHONE NUMBER.      '||
                             '<3.3.>4 FACULTY CODE.',1);
                   CALL TPUT('<3.3.>5 YEAR AT UNIVERSITY. '||
                             '<3.3.>6 STATUS IN COURSE.',1);
                   CALL TPUT('<3.3.>7 PASSWORD.',1);
                   CALL TPUT('ENTER CHOICE==>   : ',1);
                   CALL TGET(CHOICE,L_CHOICE,FALSE);
                   END; /*THEN*/
                SKIPIN=FALSE;
                CALL EXTRACT_CHOICE(CHOICE,FIRST,REST,ERR);
                IF (^ERR & SPEC & (FIRST^='7')) THEN DO;
                    ERR=TRUE;
                    CALL TPUT('INVALID CHOICE, ONLY '||
                     'PASSWORDS MAY BE MODIFIED FOR '||
                     'SPECIAL STUDENT NUMBERS.',1);
                   END;
                END;
             END;
          END;
       IF (^ERR) THEN DO;
          SELECT(FIRST);
          WHEN('1') DO; /* NAME */
             CALL TPUT('CURRENT VALUE OF NAME FIELD: '||
              STUD.NAME,1);
             CALL TPUT('ENTER NEW NAME     : ',1);
             CALL TGET(TEMPIN,30,FALSE);
             IF (TEMPIN^=' ') THEN DO;
                CALL READ_STUD_LOCK(FOUND);
                IF (^FOUND) THEN GOTO REPEAT;
                STUD.NAME=TEMPIN;
                CALL UPDATE_STUD(CID,FOUND);
                IF (^FOUND) THEN CALL INTERNAL_ERR(13);
                END;
             END; /*WHEN */
          WHEN('2') DO; /* STUDENT ID.*/
             CALL TPUT('CURRENT VALUE OF STUDENT NUMBER: '
              ||STUD.ID,1);
             CALL TPUT('ENTER NEW NUMBER(7 CHARACTERS)'||
                  '     : ',1);
             CALL TGET(TEMPIN,7,FALSE);
             IF (TEMPIN ^=' ') THEN DO;
                IF (SPECIAL_ID(SUBSTR(TEMPIN,1,7))) THEN
                   CALL TPUT('STUDENT#: '||SUBSTR(TEMPIN,
                   1,7)||' RESERVED FOR INTERNAL USE.',1);
                ELSE DO;
                   CALL READ_STUD(SUBSTR(TEMPIN,1,7),
                      FOUND);
                   IF (FOUND) THEN DO;
                      CALL TPUT('A STUDENT RECORD WITH '||
                      'THIS ID ALREADY EXISTS.',1);
                       END;
                   ELSE DO;
                      CALL READ_STUD(CID,FOUND);
                      IF (^FOUND) THEN
                          CALL INTERNAL_ERR(14);
                      STUD.ID=TEMPIN;
                      CALL WRITE_STUD(STUD.ID,FOUND);
                      CALL REMOV_STUD(CID,FOUND);
                      IF (^FOUND) THEN
                         CALL INTERNAL_ERR(13);
```

```
            END; /* ELSE */
          END; /*ELSE */
       END; /*IF*/
END; /* WHEN */
WHEN('3') DO; /* PHONE */
   CALL TPUT('CURRENT VALUE OF PHONE NUMBER: '
   ||STUD.PHONE,1);
   CALL TPUT('ENTER NEW VALUE(7 CHARACTERS)'||
        '      : ',1);
   CALL TGET(TEMPIN,7,FALSE);
   IF (TEMPIN^=' ') THEN DO;
      CALL READ_STUD_LOCK(FOUND);
      IF (^FOUND) THEN GOTO REPEAT;
      STUD.PHONE=TEMPIN;
      CALL UPDATE_STUD(CID,FOUND);
      IF (^FOUND) THEN CALL INTERNAL_ERR(13);
      END;
   END;
WHEN('4')DO; /* FACULTY CODE */
   CALL TPUT('CURRENT VALUE OF FACULTY CODE: '
   ||STUD.FACULTY,1);
   CALL TPUT('ENTER NEW VALUE(2 CHARACTERS)'||
        '      : ',1);
   CALL TGET(TEMPIN,2,FALSE);
   IF (TEMPIN^=' ') THEN DO;
      CALL READ_STUD_LOCK(FOUND);
      IF (^FOUND) THEN GOTO REPEAT;
      STUD.FACULTY=TEMPIN;
      CALL UPDATE_STUD(CID,FOUND);
      IF (^FOUND) THEN CALL INTERNAL_ERR(13);
      END;
   END; /* WHEN */
WHEN('5')DO; /* YEAR AT UNIV. */
   CALL TPUT('CURRENT VALUE OF YEAR AT UNIV.: '
   ||STUD.YEAR,1);
   CALL TPUT('ENTER NEW VALUE(2 CHARACTERS)'||
        '      : ',1);
   CALL TGET(TEMPIN,2,FALSE);
   IF (TEMPIN^=' ') THEN DO;
      CALL READ_STUD_LOCK(FOUND);
      IF (^FOUND) THEN GOTO REPEAT;
      STUD.YEAR=TEMPIN;
      CALL UPDATE_STUD(CID,FOUND);
      IF (^FOUND) THEN CALL INTERNAL_ERR(13);
      END;
   END; /*WHEN */
WHEN('6') DO; /* STATUS IN COURSE */
   CALL TPUT('CURRENT VALUE OF STATUS IN'||
   ' COURSE: '||STUD.STATUS,1);
   CALL TPUT('ENTER NEW VALUE(3 CHARACTERS)'||
        '      : ',1);
   CALL TGET(TEMPIN,3,FALSE);
   IF (TEMPIN^=' ') THEN DO;
                  CALL READ_STUD_LOCK(FOUND);
                  IF (^FOUND) THEN GOTO REPEAT;
                  STUD.STATUS=TEMPIN;
                  CALL UPDATE_STUD(CID,FOUND);
                  IF (^FOUND) THEN CALL INTERNAL_ERR(13);
                  END;
               END; /*WHEN*/
            WHEN('7') DO; /* PASSWORD */
               ◆◆◆ confidential procedure ◆◆◆
               END; /*WHEN*/
            OTHERWISE DO;
               CALL TPUT('INVALID CHOICE: '||CHOICE,1);
               END;
            END; /* OUTER SELECT */
         END; /* IF */
      END; /* DO WHILE */
   END; /* EDIT PERSONAL */
/* LOGICAL END OF EDIT PERSONAL */

   /*****************************************************/
   /*                                                   */
   /*    THIS PROCEDURE IS CALLED IF THE EDIT COURSE */
   /*    DATA SELECTION WAS MADE FROM THE EDITOR MENU*/
   /*    IT DISPLAYS A MENU OF THE COURSE DATA FIELDS*/
   /*    WHICH CAN BE ALTERED  AND THEN ALLOWS       */
   /*    TO POSITION TO A FIELD TO BE REPLACED. THIS */
   /*    PROCEDURE THEN MAKES USE OF A SUBROUTINE    */
   /*    WHICH CONTAINS THE CODE FOR FIELD REPLACE-  */
   /*    MENT(COM_R_EDIT). THIS SUBROUTINE LOOPS     */
   /*    ASKING FOR THE STUDENT NUMBER OF THE NEXT   */
   /*    RECORD TO BE MODIFIED. THE FIELD TO BE      */
   /*    MODIFIED IS THE ONE PREVIOUSLY CHOSEN       */
   /*    THROUGH THE MENU.                           */
   /*                                                   */
   /*****************************************************/
   EDIT_COURSE:PROC(CHOICE);
      DCL CHOICE                  CHAR(*);
      DCL (FIRST,REST)       CHAR(L_CHOICE);
      DCL TEMPIN                  CHAR(10);
      DCL CID                     CHAR(7);
      DCL (SKIPIN,CONTINUE,ERR,FOUND,SPEC,BLANKF) BIT(1);
      DCL ITEMPO                  PICTURE 'ZZZ9';
      DCL ITEMPI                  FIXED BIN(15);
      DCL RTEMPI                  FIXED DEC(7,3);
      DCL CLOOP                   BIT(1);
      DCL SUNIT                   FIXED BIN(15);
      DCL IDERRMSG                CHAR(25) INIT(
      ' IS NOT A VALID STUDENT#.');
      SKIPIN=FALSE;
      IF (CHOICE^=' ') THEN SKIPIN=TRUE;
      CONTINUE=TRUE;
      DO WHILE(CONTINUE);
```

```
   ERR=FALSE;
   IF (^SKIPIN) THEN DO;
      CALL TPUT('<3.4.>1 CURRENT UNIT.    '||
                '<3.4.>2 TEST POINTS.',1);
      CALL TPUT('<3.4.>3 PROCTOR POINTS. '||
                '<3.4.>4 TERM POINTS.',1);
      CALL TPUT('<3.4.>5 FINAL EXAM.      '||
                '<3.4.>6 RETURN TO EDIT MENU.',1);
      CALL TPUT('ENTER CHOICE==>    : ',1);
      CALL TGET(CHOICE,L_CHOICE,FALSE);
      IF (TURNOFF & (CHOICE=' ')) THEN RETURN;
      END;
   SKIPIN=FALSE;
   CALL EXTRACT_CHOICE(CHOICE,FIRST,REST,ERR);
   IF (^ERR) THEN DO;
      SELECT(FIRST);
      WHEN('1') DO; /*HIGHEST UNIT*/
         DCL HMSG                  CHAR(21) INIT(
         'HIGHEST UNIT REACHED.');
         CLOOP=TRUE;
         BLANKF=FALSE;
REPEAT:  DO WHILE(CLOOP);
         CALL GETID(CID,FOUND,SPEC,BLANKF,ENTERNEXT);
         IF ((SPEC|(^FOUND)) & (^BLANKF) ) THEN DO;
            CALL TPUT(CID||DERRMSG,1);
            END;
         ELSE DO;
         IF (BLANKF) THEN
            CLOOP=FALSE;
         ELSE DO;
            CALL READ_SYSP;
            CALL FPROMPT_NUMI(STUD.UNIT,HMSG,
                     SYSP.NUNIT,BLANKF);
            IF (^BLANKF) THEN DO;
               SUNIT=STUD.UNIT;
               CALL READ_STUD_LOCK(FOUND);
               IF (^FOUND) THEN GOTO REPEAT;
               STUD.UNIT=SUNIT;
               STUD.TSTATE=TST_I;
               STUD.PSTATE=PST_I;
               CALL UPDATE_STUD(CID,FOUND);
               IF (^FOUND) THEN CALL INTERNAL_ERR(17);
               END;
            END; /* ELSE */
         END; /* ELSE */
         END; /* WHILE */
         END; /*WHEN*/
      WHEN('2') DO; /* TEST POINTS */
         DCL TMSG                  CHAR(12) INIT(
         'TEST POINTS');
         CALL COM_R_EDIT(STUD.TEST,TMSG);
         END; /* WHEN */
      WHEN('3') DO; /* PROCTOR POINTS */
         DCL PMSG                  CHAR(14) INIT(
         'PROCTOR POINTS');
         CALL COM_R_EDIT(STUD.PROCTOR,PMSG);
         END;
      WHEN('4') DO; /* TERM POINTS */
         DCL RMSG                  CHAR(11) INIT(
         'TERM POINTS');
         CALL COM_R_EDIT(STUD.TERM,RMSG);
         END;
      WHEN('5') DO;
         DCL EMSG                  CHAR(11) INIT(
         'EXAM POINTS');
         CALL COM_R_EDIT(STUD.EXAM,EMSG);
         END;
      WHEN ('0','6') DO;
         CONTINUE=FALSE;
         END;
      OTHERWISE DO;
         CALL TPUT('INVALID CHOICE: '||CHOICE,1);
         END;
      END; /*SELECT*/
   END; /*IF*/
   ELSE DO;
      CALL TPUT('INVALID CHOICE: '||CHOICE,1);
      END;
   END; /* WHILE */
/* LOGICAL END OF EDIT COURSE */

 /*****************************************************/
 /*                                                   */
 /*    THIS PROCEDURE CONTAINS THE COMMON CODE        */
 /*    REQUIRED BY THE EDIT COURSE PROCEDURE TO       */
 /*    MODIFY THE COURSE DATA FIELDS. RVAL IS THE     */
 /*    ARGUMENT WHICH ISOLATES THE FIELD TO BE        */
 /*    MODIFIED WHILE MSG IS THE CHARACTER STRING     */
 /*    TACKED ON TO THE COMMON PART OF THE PROMPTS    */
 /*    WHICH IDENTIFY TO THE USER THE FIELD BEING     */
 /*    EDITED. THE PROCEDURE LOOPS, POSITIONING TO    */
 /*    A NEW STUDENT RECORD UNTIL A BARE RETURN IS    */
 /*    ENTERED FOR A STUDENT NUMBER.                  */
 /*                                                   */
 /*****************************************************/
 COM_R_EDIT:PROC(RVAL,MSG);
    DCL MSG                          CHAR(*);
    DCL (RVAL,REAL)                  FIXED DEC(7,3);
    DCL (CLOOP,BLANKF,FOUND)         BIT(1);
    CLOOP=TRUE;
    BLANKF=FALSE;
REPEAT: DO WHILE(CLOOP);
      CALL GETID(CID,FOUND,SPEC,BLANKF,ENTERNEXT);
      IF ((SPEC|(^FOUND))&(^BLANKF)) THEN DO;
         CALL TPUT(CID||DERRMSG,1);
```

```
            END;
         ELSE DO;
            IF (BLANKF) THEN
               CLOOP=FALSE;
            ELSE DO;
               CALL FPROMPT_NUMR(RVAL,MSG,BLANKF);
               IF (^BLANKF) THEN DO;
                  REAL=RVAL;
                  CALL READ_STUD_LOCK(FOUND);
                  IF (^FOUND) THEN GOTO REPEAT;
                  RVAL=REAL;
                  CALL TOTAL_MARKS;
                  CALL UPDATE_STUD(CID,FOUND);
                  IF (^FOUND) THEN CALL INTERNAL_ERR(17);
                  END;
               END;
            END;
         END; /*WHILE*/
      END; /* COM_A_EDIT */
   END; /* EDIT COURSE */

/*************************************************************/
/*                                                           */
/*    THIS PROCEDURE PERFORMS THE LIST FUNCTION OF THE       */
/*    STUDENT RECORD EDITOR. IT LOOPS ASKING FOR A           */
/*    STUDENT NUMBER TO BE ENTERED. A LIST OF DATA FOR       */
/*    THAT STUDENT IS THEN PRINTED. IF <ALL> IS THE          */
/*    STUDENT NUMBER ENTERED THEN THE PROCEDURE LISTS        */
/*    ALL THE STUDENTS ON FILE. IF A BARE RETURN IS          */
/*    THE STUDENT NUMBER ENTERED THEN THE PROCEDURE          */
/*    RETURNS CONTROL TO EDIT_STUD.                          */
/*                                                           */
/*************************************************************/
   LIST_STUD:PROC(CHOICE);
      DCL CHOICE                    CHAR(*);
      DCL CID                       CHAR(7);
      DCL (FOUND,SPEC,BLANKF,CONTINUE) BIT(1);
      DCL HEADR1                    CHAR(45) INIT(
      'STUD.#  NAME                          PHONE ');
      DCL HEADR2                    CHAR(46) INIT(
      ' FAC YR STAT UNIT TEST PROCTOR    TERM    EXAM');
      DCL HEADR3                    CHAR(14) INIT(
      '   TOTAL GRADE');
      DCL REC                       FIXED BIN(15);
      DCL OUTVAL                    PICTURE 'ZZZZ9';
      CONTINUE = TRUE;
      DO WHILE(CONTINUE);
         CALL GETID(CID,FOUND,SPEC,BLANKF,ENTERSTNUM);
         IF (BLANKF) THEN
            CONTINUE = FALSE;
         ELSE DO;
            IF ((^FOUND) & (CID^='ALL')  ) THEN DO;
               CALL TPUT(CID||' IS A SPECIAL OR NON-EXISTANT'
                  ||' STUDENT NUMBER. DATA CANNOT BE LISTED.',1);
               END;
            ELSE DO;
               CALL TPUT(HEADR1||HEADR2||HEADR3,1);
               IF (CID='ALL') THEN DO;
                  CALL CLOSE_FILE(STUDFIL);
                  OPEN FILE(STUDFIL) INPUT SEQUENTIAL BUF;
                  ON ENDFILE(STUDFIL) GOTO EOF;
                  ON ATTENTION BEGIN;
                     ON ATTENTION SYSTEM;
                     CALL GET_RESP('(C)ONTINUE OR (Q)UIT  : ',
                        CQ,RESP,1);
                     IF (RESP=2) THEN GOTO OUT;
                              ELSE GOTO CONT;
                     END;
                  REC=0;
                  DO WHILE('1'B);
                  CONT: READ FILE(STUDFIL) INTO(STUD);
                        IF (^SPECIAL_ID(STUD.ID)) THEN DO;
                           REC=REC+1;
                           CALL ONE_OUT;
                           END;
                     END;
                  EOF: ON ATTENTION SYSTEM;
                       OUTVAL=REC;
                       CALL TPUT('TOTAL# OF STUDENTS: '||OUTVAL,1);
                  OUT: CALL CLOSE_FILE(STUDFIL);
                       OPEN FILE(STUDFIL) EXCL UPDATE DIRECT UNBUF;
                       END;
                  ELSE CALL ONE_OUT;
               END; /*ELSE*/
            END; /*ELSE*/
         END; /*WHILE*/
      /* LOGICAL END OF LIST STUD */

/*************************************************************/
/*                                                           */
/*   THIS IS A SUBROUTINE OF THE LIST_STUD PROCEDURE.        */
/*   IT CONTAINS THE CODE WHICH FORMATS ONE LINE OF          */
/*   OUTPUT.                                                 */
/*                                                           */
/*                                                           */
/*************************************************************/
      ONE_OUT:PROC;
         DCL COUT                   CHAR(104);
         PUT STRING(COUT) EDIT(STUD.ID,STUD.NAME,STUD.PHONE,
            STUD.FACULTY,STUD.YEAR,STUD.STATUS,STUD.UNIT,
            STUD.TEST,STUD.PROCTOR,STUD.TERM,STUD.EXAM,
            STUD.TOTAL,STUD.LETTER)
            (5(A,X(1)),A,F(3,0),5(F(8,3)),X(3),A(2));
         CALL TPUT(COUT,1);
      END; /* ONE OUT */
   END; /* LIST_STUD */
```

```
END; /*EDIT STUD*/

/*************************************************************/
/*                                                           */
/*     THIS PROCEDURE PERFORMS THE FUNCTION OF THE EDIT*/
/*     SYSTEM PARAMETER CHOICE FROM THE MAIN CONTROL   */
/*     MENU.                                             */
/*                                                           */
/*************************************************************/
 EDIT_SYSP:PROC(CHOICE);
    DCL CHOICE                  CHAR(*);
    DCL (FIRST,REST)            CHAR(L_CHOICE);
    DCL (CONTINUE,SKIPIN,ERR,BLANKF,FOUND) BIT(1);
    DCL I                       FIXED BIN(15);
    DCL (THRSHLOOP,FIRSTIME)     BIT(1);
    DCL COUT                     CHAR(87);
    SKIPIN = FALSE;
    IF (CHOICE^=' ') THEN SKIPIN=TRUE;
    CONTINUE=TRUE;
    DO WHILE(CONTINUE);
       IF (^SKIPIN) THEN DO;
          CALL TPUT('<4.>1 VEIW SYSTEM PARAMETERS.        '||
                    '<4.>2 ENTER TOTAL NUMBER OF UNITS.',1);
          CALL TPUT('<4.>3 ENTER LETTER GRADE THRESHOLDS. '||
                    '<4.>4 ENTER VALUE OF UNIT PASSING.',1);
          CALL TPUT('<4.>5 ENTER VALUE OF PROCTORING.     '||
                    '<4.>6 ENTER QUESTION NUMBER LIMITS.',1);
          CALL TPUT('<4.>7 RETURN TO MAIN MENU.',1);
          CALL TPUT('ENTER CHOICE==>    : ',1);
          CALL TGET(CHOICE,L_CHOICE,FALSE);
          IF (TURNOFF & (CHOICE=' ')) THEN RETURN;
          END;
       CALL EXTRACT_CHOICE(CHOICE,FIRST,REST,ERR);
       IF (ERR) THEN FIRST='X';
       SKIPIN=FALSE;
       BLANKF=FALSE;
       SELECT(FIRST);
       WHEN('1') DO;
          CALL READ_SYSP;
          PUT STRING(COUT) EDIT('TOTAL NUMBER OF UNITS:',
             SYSP.NUNIT,', VALUE OF UNIT PASSING:',SYSP.VPASS,
             ', VALUE OF PROCTORING:',SYSP.VPROC)
             (A,F(3,0),A,F(8,3),A,F(8,3));
          CALL TPUT(COUT,1);
          CALL TPUT('LETTER GRADE THRESHOLDS:',1);
          PUT FILE(SYSPRINT) EDIT((LGLIT(I),SYSP.LGTHRESH(I)
              DO I=1 TO 13))
              (SKIP,COL(1),5 (A(2),X(1),F(8,3),X(4)));
          CALL TPUT('UNIT QUESTION NUMBER LIMITS:',1);
          PUT FILE(SYSPRINT) EDIT((I,SYSP.UNITL(I) DO I=1 TO
              SYSP.NUNIT))
              (SKIP,COL(1),5 (P'ZZ9.',P'ZZ9',X(5)) );
          END;
       WHEN('2')DO; /*ENTER TOTAL UNITS*/
          DCL T2MSG                        CHAR(21) INIT(
          'TOTAL NUMBER OF UNITS');
          SYST.NUNIT=SYSP.NUNIT;
          CALL FPROMPT_NUMI(SYST.NUNIT,T2MSG,100,BLANKF);
          IF (^BLANKF & (SYST.NUNIT^=SYSP.NUNIT)) THEN DO;
             CALL READ_SYSP_LOCK;
             SYSP.NUNIT=SYST.NUNIT;
             CALL UPDATE_SYSP;
             END;
          END;
       WHEN('3') DO; /* LETTER GRADE THRESHOLD. */
          DCL L3MSG                        CHAR(11) INIT(
          ' THRESHOLD.');
          THRSHLOOP=TRUE;
          FIRSTIME=TRUE;
          DO WHILE(THRSHLOOP);
             ERR=FALSE;
             IF (FIRSTIME) THEN DO;
                CALL TPUT('<4.>1  A+ <4.>2  A  <4.>3  A-',
                1);
                CALL TPUT('<4.>4  B+ <4.>5  B  <4.>6  B-',
                1);
                CALL TPUT('<4.>7  C+ <4.>8  C  <4.>9  C-',
                1);
                CALL TPUT('<4.>10 D+ <4.>11 D  <4.>12 D-'
                ||' <4.>13 F',1);
                END;
             CALL TPUT('ENTER LETTER CHOICE(1-13)==>    : ',1);
             CALL GET_INT(I,ERR,BLANKF);
             FIRSTIME=FALSE;
             IF (BLANKF) THEN THRSHLOOP=FALSE;
             ELSE DO;
                IF (^ERR) THEN DO;
                   IF ((I>13) | (I<1)) THEN DO;
                      CALL TPUT('CHOICE OF OPTION'||
                      ' OUT OF RANGE(1-13)',1);
                      ERR=TRUE;
                      END;
                   ELSE DO;
                      SYST.LGTHRESH(I)=SYSP.LGTHRESH(I);
                      CALL FPROMPT_NUMR(SYST.LGTHRESH(I),LGLIT(I)
                      ||L3MSG,BLANKF);
                      IF (^BLANKF) &
                         (SYST.LGTHRESH(I)^=SYSP.LGTHRESH(I))
                         THEN DO;
                         CALL READ_SYSP_LOCK;
                         SYSP.LGTHRESH(I)=SYST.LGTHRESH(I);
                         CALL UPDATE_SYSP;
                         ON ENDFILE(STUDFIL) GOTO EOF;
                         CALL CLOSE_FILE(STUDFIL);
                         OPEN FILE(STUDFIL) UPDATE SEQL BUF;
```

```
            DO WHILE('1'B);
               READ FILE(STUDFIL) INTO(STUD);
               CALL TOTAL_MARKS;
               REWRITE FILE(STUDFIL) FROM(STUD);
            END;
         EOF: CALL CLOSE_FILE(STUDFIL);
          OPEN FILE(STUDFIL) EXCL UPDATE DIRECT UNBUF;
            END;
          END;
        END;
       IF (ERR) THEN CALL TPUT('INVALID ENTRY.',1);
       END;
   END; /*WHILE*/
   END; /*WHEN*/
WHEN('4')DO; /*ENTER VALUE OF UNIT PASSING*/
   DCL U4MSG                    CHAR(12) INIT(
   'UNIT PASSING');
   SYST.VPASS=SYSP.VPASS;
   CALL FPROMPT_NUMR(SYST.VPASS,U4MSG,BLANKF);
   IF (^BLANKF & (SYST.VPASS^=SYSP.VPASS)) THEN DO;
      CALL READ_SYSP_LOCK;
      SYSP.VPASS=SYST.VPASS;
      CALL UPDATE_SYSP;
      END;
   END;
WHEN('5') DO;
   DCL P5MSG                    CHAR(11) INIT(
   ' PROCTORING');
   SYST.VPROC=SYSP.VPROC;
   CALL FPROMPT_NUMR(SYST.VPROC,P5MSG,BLANKF);
   IF (^BLANKF & (SYST.VPROC^=SYSP.VPROC)) THEN DO;
      CALL READ_SYSP_LOCK;
      SYSP.VPROC=SYST.VPROC;
      CALL UPDATE_SYSP;
      END;
   END;
WHEN('6') DO;
   DCL (ITEMPI,J)               FIXED BIN(15);
   DCL ITEMPO                   PICTURE 'ZZ9';
   BLANKF=FALSE;
   DO WHILE(^BLANKF);
      CALL READ_SYSP;
      ERR=FALSE;
      CALL TPUT('ENTER UNIT NUMBER     : ',1);
      CALL GET_INT(ITEMPI,ERR,BLANKF);
      IF ((^ERR) &(^BLANKF)) THEN DO;
         IF ((ITEMPI>0)&(ITEMPI<=SYSP.NUNIT)) THEN DO;
            J=SYSP.UNITL(ITEMPI);
            ITEMPO=ITEMPI;
           CALL FPROMPT_NUMI(J,'UNIT QUESTION LIMIT '||
            'FOR UNIT('||ITEMPO||')',999,BLANKF);
            IF (J<3) THEN DO;
               CALL TPUT('UNIT MUST HAVE AT LEAST 3'||
                 ' QUESTIONS.',1);
               ERR=TRUE;
               END;
            IF ((^BLANKF) & (^ERR)) THEN  DO;
               CALL READ_SYSP_LOCK;
               SYSP.UNITL(ITEMPI)=J;
               CALL UPDATE_SYSP;
               END;
            END;
            ELSE DO;
               ERR=TRUE ;   END;
            END;
         IF (ERR) THEN
            CALL TPUT('UNIT NUMBER IS OUT OF RANGE',1);
         END; /* WHILE */
         END; /*WHEN*/
       WHEN('0','7') DO; /* EXIT TO CONTROL MENU */
         CONTINUE=FALSE;
         END;
       OTHERWISE DO;
         ERR=TRUE;
         CALL TPUT('INVALID CHOICE: '||CHOICE,1);
         END;
       END; /* SELECT */
       CALL TPUT('ENDING AT: '||SEPARATE(SUBSTR(TIME,1,6)),1);
    END; /*WHILE*/
  END; /* EDIT SYSP. */

/*************************************************************/
/*                                                           */
/*   THIS IS A SUBROUTINE OF LISTING OUT THE STUDENT         */
/*   PROCTORS OF THE INPUT STUDENT ID. IF NO STUDENT         */
/*   PROCTORS FOR HIM IS ASSIGNED, THE STUD_PROC WILL        */
/*   RETURNS A FALSE. PRINT IS TO TURN ON OR OFF             */
/*   THE LISTING OF THE STUDENT PROCTOR.                     */
/*                                                           */
/*************************************************************/
    STUD_PROCTORS:PROC(CID,DIRECT,PRINT,STUD_PROC);
       DCL CID                  CHAR(7);
       DCL (DIRECT,PRINT,STUD_PROC) BIT(1);
       STUD_PROC=FALSE;
       ON ENDFILE(STUDFIL) GOTO EOF;
       CALL CLOSE_FILE(STUDFIL);
       OPEN FILE(STUDFIL) INPUT SEQUENTIAL BUFFERED;
       DO WHILE('1'B);
          READ FILE(STUDFIL) INTO(STUD);
          IF ((STUD.PSTATE=PST_P)&(STUD.SID=CID)) THEN DO;
             IF (PRINT) THEN
                CALL TPUT('PROCTOR HAS NOT ENTERED RESULTS, '
                   ||'STUDENT#: '||STUD.ID||', NAME: '||
                   STUD.NAME||', AT TIME: '||
                   SEPARATE(STUD.RTIME),1);
```

```
            STUD_PROC=TRUE;
            END;
      END;
EOF:  CALL CLOSE_FILE(STUDFIL);
      IF (DIRECT) THEN
         OPEN FILE(STUDFIL) EXCL UPDATE DIRECT UNBUFFERED;
      ELSE DO;
         OPEN FILE(STUDFIL) INPUT SEQUENTIAL BUFFERED;
         READ FILE(STUDFIL) INTO(STUD) KEY(CID);
         END;
   END; /* STUD_PROCTORS */

/*************************************************************/
/*                                                           */
/*      THIS PROCEDURE IS THE ONE INVOKED AFTER THE          */
/*      MARK TEST CHOICE FROM THE MAIN CONTROL MENU IS       */
/*      SELECTED. IT IS ALSO INVOKED FROM THE                */
/*      SESSION_CONTROL PROCEDURE WHEN EVER THE              */
/*      INSTRUCTOR OR TA SIGNS ON AS <MARKER> .              */
/*                                                           */
/*************************************************************/
 MARK_TEST:PROC(CHOICE);
   DCL CHOICE                          CHAR(*);
   DCL (CONTINUE,SKIPIN,ERR)           BIT(1);
   DCL (FIRST,REST)                    CHAR(L_CHOICE);
   DCL HEADR1                          CHAR(38) INIT(
      'STUD.#  NAME');
   DCL HEADR2                          CHAR(50) INIT(
      'UNIT# QUESTIONS TEST STATE    PROCTOR STATE   TIME');
   SKIPIN = FALSE;
   IF (CHOICE^= ' ') THEN DO;
      SKIPIN=TRUE;
      END;
   CONTINUE = TRUE;
   DO WHILE(CONTINUE);
      IF (^SKIPIN) THEN DO;
         CALL TPUT('<5.>1 LIST TEST AND STATUS. '||
                   '<5.>2 LIST TESTS FOR MARKER.',1);
         CALL TPUT('<5.>3 MARK STUDENT.         '||
                   '<5.>4 RETURN FROM MARKING.',1);
         CALL TPUT('ENTER CHOICE==>    : ',1);
         CALL TGET(CHOICE,L_CHOICE,FALSE);
         IF (TURNOFF & (CHOICE=' ')) THEN RETURN;
         END;
      SKIPIN=FALSE ;
      CALL EXTRACT_CHOICE(CHOICE,FIRST,REST,ERR);
      IF (ERR) THEN FIRST='X';
      SELECT(FIRST);
      WHEN('1') DO; /* LIST PROCTOR AND TEST */
         CALL LIST_PROC_TEST(REST);
         END;
      WHEN('2') DO; /* LIST TESTS FOR MARKER */
         CALL LIST_MARKER(REST);
         END;
      WHEN('3') DO; /* MARKING */
         CALL MARK_STUD(REST);
         END;
      WHEN('0','4') DO; /* EXIT */
          CONTINUE = FALSE;
          END;
      OTHERWISE DO; /* BAD CHOICE */
         CALL TPUT('INVALID CHOICE: '||CHOICE,1);
         END;
      END; /*SELECT */
      CALL TPUT('ENDING AT: '||SEPARATE(SUBSTR(TIME,1,6)),1);
   END; /* DO LOOP */

/*************************************************************/
/*                                                           */
/*    THIS PROCEDURE PERFORMS THE LIST STATUS AND TEST       */
/*    OF THE STUDENT RECORD. IT LOOPS ASKING FOR A           */
/*    STUDENT NUMBER TO BE ENTERED. A LIST OF DATA FOR       */
/*    THAT STUDENT IS THEN PRINTED. IF <ALL> IS THE          */
/*    STUDENT NUMBER ENTERED THEN THE PROCEDURE LISTS        */
/*    ALL THE STUDENTS ON FILE. IF A BARE RETURN IS          */
/*    THE STUDENT NUMBER ENTERED THEN THE PROCEDURE          */
/*    RETURNS CONTROL TO THE MARKING MENU.                   */
/*************************************************************/
    LIST_PROC_TEST:PROC(CHOICE);
      DCL CHOICE                    CHAR(*);
      DCL CID                       CHAR(7);
      DCL (FOUND,SPEC,BLANKF,CONTINUE) BIT(1);
      DCL OUTST                     FIXED BIN(15);
      CONTINUE = TRUE;
      DO WHILE(CONTINUE);
         CALL GETID(CID,FOUND,SPEC,BLANKF,ENTERSTNUM);
         IF (BLANKF) THEN
            CONTINUE = FALSE;
         ELSE DO;
           IF ((^FOUND) & (CID^='ALL')  ) THEN DO;
              CALL TPUT(CID||' IS A SPECIAL OR NON-EXISTANT'
              ||' STUDENT NUMBER. DATA CANNOT BE LISTED.',1);
             END;
           ELSE DO;
              IF (CID='ALL') THEN DO;
                 CALL TPUT('LISTING OF ALL THE UNMARKED '||
                    'STUDENTS.',1);
                 CALL TPUT(HEADR1||HEADR2,1);
                 CALL CLOSE_FILE(STUDFIL);
                 OPEN FILE(STUDFIL) INPUT SEQUENTIAL BUF;
                 ON ENDFILE(STUDFIL) GOTO EOF;
                 ON ATTENTION BEGIN;
                    ON ATTENTION SYSTEM;
                    CALL GET_RESP('(C)ONTINUE OR (Q)UIT  : ',
                       CQ, RESP,1);
```

```
                IF (RESP=2) THEN GOTO EOF;
                            ELSE GOTO CONT;
                END;
             DO WHILE('1'B);
          CONT: READ FILE(STUDFIL) INTO(STUD) KEYTO(CID);
                OUTST=STAT_TBL_END_SESS.TSTATE(
                   STUD.TSTATE);
                IF (^SPECIAL_ID(STUD.ID)&
                   ((OUTST=TST_OT)|(OUTST=TST_OM))) THEN
                   CALL ONE_STATUS_OUT(CID,FALSE);
             END;
          EOF: ON ATTENTION SYSTEM;
             CALL CLOSE_FILE(STUDFIL);
             OPEN FILE(STUDFIL) EXCL UPDATE DIRECT UNBUF;
             END;
           ELSE DO;
             CALL TPUT(HEADR1||HEADR2,1);
             CALL ONE_STATUS_OUT(CID,TRUE);
             END;
           END; /*ELSE*/
         END; /*ELSE*/
      END; /*WHILE*/
   END; /* LIST_PROC_TEST */

/*********************************************************/
/*                                                       */
/*   THIS IS A SUBROUTINE OF THE LIST STUDENT STATUS     */
/*   IT CONTAINS THE CODE WHICH FORMATS ONE LINE OF      */
/*   OUTPUT.                                             */
/*                                                       */
/*********************************************************/
   ONE_STATUS_OUT:PROC(CID,DIRECT);
      DCL CID                  CHAR(7);
      DCL DIRECT               BIT(1);
      DCL STUD_PROC            BIT(1);
      DCL OUT_UNIT             PICTURE 'ZZ9';
      DCL (OUT_Q1,OUT_Q2,OUT_Q3) PICTURE 'ZZZ9';
      DCL OUT1                 CHAR(38);
      DCL OUT2                 CHAR(49);
      DCL OUTST                FIXED BIN(15);
      OUT_Q1=ABS(STUD.Q1);
      OUT_Q2=ABS(STUD.Q2);
      OUT_Q3=ABS(STUD.Q3);
      OUT1=STUD.ID||' '||STUD.NAME;
      OUT2=OUT_Q1||OUT_Q2||OUT_Q3||' '||TSC(STUD.TSTATE)||
         ' '||PSC(STUD.PSTATE)||' '||SEPARATE(STUD.RTIME);
      OUTST=STAT_TBL_END_SESS.TSTATE(STUD.TSTATE);
      IF ((OUTST=TST_OT)|(OUTST=TST_OM)) THEN DO;
         OUT_UNIT=STUD.UNIT+1;
         CALL TPUT(OUT1||OUT_UNIT||OUT2,1);
         IF (STUD.TSTATE^=TST_T) THEN DO;
            CALL STUD_PROCTORS(CID,DIRECT,TRUE,STUD_PROC);
            IF (^STUD_PROC) THEN
               CALL TPUT('PROCTOR SELECTED IS INSTRUCTOR'||
                  ' OR TA.',1);
            END;
         ELSE DO;
            CALL TPUT('STUDENT IS WRITING A TEST.',1);
            END;
         END;
      ELSE DO;
         OUT_UNIT=STUD.UNIT;
         CALL TPUT(OUT1||OUT_UNIT||OUT2,1);
         CALL TPUT('STUDENT HAS NO UNMARKED TEST.',1);
         END;
   END; /* ONE_STATUS_OUT */

/*********************************************************/
/*                                                       */
/*   THIS IS A SUBROUTINE OF LISTING OF THE STUDENTS     */
/*   THAT HAS NO STUDENT PROCTORS AND WERE ASSIGNED      */
/*   TO TA OR INSTRUCTOR FOR MARKING.                    */
/*                                                       */
/*********************************************************/
   LIST_MARKER:PROC(CHOICE);
      DCL CHOICE               CHAR(*);
      DCL CID                  CHAR(7);
      DCL STUD_PROC            BIT(1);
      DCL OUT_UNIT             PICTURE 'ZZ9';
      DCL (OUT_Q1,OUT_Q2,OUT_Q3) PICTURE 'ZZZ9';
      DCL OUTST                FIXED BIN(15);
      CALL TPUT('INSTRUCTOR OR TA IS SELECTED.',2);
      CALL TPUT(HEADR1||HEADR2,1);
      CALL CLOSE_FILE(STUDFIL);
      OPEN FILE(STUDFIL) INPUT SEQUENTIAL BUFFERED;
      ON ENDFILE(STUDFIL) GOTO EOF;
      ON ATTENTION BEGIN;
         ON ATTENTION SYSTEM;
         CALL GET_RESP('(C)ONTINUE OR (Q)UIT  : ',CQ,
            RESP,1);
         IF (RESP=2) THEN GOTO EOF;
                     ELSE GOTO CONT;
         END;
      DO WHILE('1'B);
   CONT: READ FILE(STUDFIL) INTO(STUD) KEYTO(CID);
         OUTST=STAT_TBL_END_SESS.TSTATE(
            STUD.TSTATE);
         IF (^SPECIAL_ID(STUD.ID)&
            ((OUTST=TST_OT)|(OUTST=TST_OM))) THEN DO;
            CALL STUD_PROCTORS(CID,FALSE,FALSE,STUD_PROC);
            IF (^STUD_PROC) THEN DO;
               OUT_Q1=ABS(STUD.Q1);
               OUT_Q2=ABS(STUD.Q2);
               OUT_Q3=ABS(STUD.Q3);
               OUT_UNIT=STUD.UNIT+1;
```

```
            CALL TPUT(STUD.ID||' '||STUD.NAME||OUT_UNIT||
               OUT_Q1||OUT_Q2||OUT_Q3||' '||
               TSC(STUD.TSTATE)||' '||PSC(STUD.PSTATE)||
               ' '||SEPARATE(STUD.RTIME),1);
            END;
         END;
      END; /* WHILE */
   EOF: ON ATTENTION SYSTEM;
      CALL CLOSE_FILE(STUDFIL);
      OPEN FILE(STUDFIL) EXCLUSIVE UPDATE DIRECT UNBUFFERED;
    END; /* LIST_MARKER */

/*************************************************************/
/*                                                           */
/*      THIS IS A SUBROUTINE THAT GOES INTO A LOOP           */
/*      PROMPTING THE INSTRUCTOR OR TA FOR THE               */
/*      NEXT STUDENT NUMBER TO BE MARKED                     */
/*      ( A BARE RETURN CAUSE IT TO TERMINATE). AND          */
/*      THEN PROMPTS HIM FOR A MARK IN THE SAME MANNER*/
/*      STUDENT PROCTORS ARE PROMPTED.                       */
/*                                                           */
/*************************************************************/
    MARK_STUD:PROC(CHOICE);
        @@@ confidential procedure @@@
     END;  /* MARK_STUD */
  END; /* MARK_TEST */

    /*************************************************************/
    /*                                                           */
    /*   THIS PROCEDURE PROVIDES MESSAGES SENDING                */
    /*   BETWEEN USERS. A USER LIST (I.E. MORE THAN              */
    /*   1 USER) CAN BE ENTERED WITH THE SYNTAX OF               */
    /*   NO BLANK EMBBED AND COMMA BETWEEN USER NAMES.*/
    /*   THE USER LIST WILL MAINTAIN IN THE SYSTEM               */
    /*   UNTIL THE PROGRAM RETURNS TO TSO. THE                   */
    /*   INCORRECT USER NAME OR THE USER WAS NOT                 */
    /*   SIGNED ON WILL BE DELETED FROM THE LIST.                */
    /*   NO USER NAME OR NO MESSAGE WILL RETURN FROM             */
    /*   PROGRAM. IN ORDER TO SEND MESSAGE IT NEEDS              */
    /*   A NULL OR BLANK LINE AT THE LAST LINE.                  */
    /*                                                           */
    /*************************************************************/
SEND_MESS:PROC(CHOICE);
   DCL CHOICE                    CHAR(*);
   DCL USER                      CHAR(8);
   DCL COLLECTIDS                CHAR(256) VARYING;
   DCL (USERIDS,FROM)            CHAR(256) VARYING
                                 STATIC INIT('');
   DCL TEMPIN                    CHAR(256);
   DCL (I,J,K)                   FIXED BIN(15);
   DCL CONT                      BIT(1);
   CONT=TRUE;
   DO WHILE(CONT);
                                 I=0;
                                 DO UNTIL(INDEX(TRIM(TEMPIN)||',',' ')=0);
                                    I=I+1;
                                    IF (I>1) THEN
                                       CALL TPUT('INVALID SYNTAX, RE-ENTER.',1);
                                    CALL TPUT('SEND TO  : '||USERIDS,1);
                                    CALL TPUT('SEND TO  : ',0);
                                    CALL TGETL(TEMPIN,256);
                                    TEMPIN=TRANSLATE(TEMPIN,UPPERCASE,LOWERCASE);
                                 END;
                                 IF (TEMPIN^=' ') THEN
                                    USERIDS=TRIM(TEMPIN);
                                 IF (USERIDS='') THEN RETURN;
                                 CALL TPUT('SEND FROM  : '||FROM,0);
                                 CALL TPUT('SEND FROM  : ',0);
                                 CALL TGETL(TEMPIN,256);
                                 IF (TEMPIN^=' ') THEN
                                    FROM=TRIM(TEMPIN);
                                 ON ATTENTION BEGIN;
                                    BUFFER='';
                                    CALL TPUT('SEND MESSAGES CANCELLED.',1);
                                    GOTO OUT;
                                    END;
                                 ON STRINGSIZE BEGIN;
                                    DCL TEMP              CHAR(LBUF) VARYING;
                                    TEMP=SUBSTR(BUFFER,1,K);
                                    LBUF=LBUF+LBUFMORE;
                                    FREE BUFFER;
                                    ALLOC BUFFER;
                                    BUFFER=TEMP;
                                    GOTO AGAIN;
                                    END;
                                 BUFFER='';
                                 CALL TPUT('M: ',0);
                                 CALL TGETL(TEMPIN,256);
                                 DO WHILE(TEMPIN^=' ');
                                    K=LENGTH(BUFFER);
(STRINGSIZE):AGAIN:BUFFER=BUFFER||TRIM(TEMPIN)||CR||LF;
                                    CALL TPUT('M: ',0);
                                    CALL TGETL(TEMPIN,256);
                                 END;
                              OUT: ON ATTENTION SYSTEM;
                                 IF (BUFFER^='') THEN DO;
                                    BUFFER=BEL||CR||LF||BUFFER;
                                    COLLECTIDS='';
                                    DO WHILE(USERIDS^='');
                                       J=INDEX(USERIDS||',',',');
                                       USER=SUBSTR(USERIDS,1,J-1);
                                       IF (J+1>LENGTH(USERIDS)) THEN
                                          USERIDS='';
                                       ELSE
                                          USERIDS=SUBSTR(USERIDS,J+1);
```

```
         IF (USER^=' ') THEN DO;
            CALL SEND(CR||LF||LF||'MESSAGE FROM: ('||
               TRIM(USERID)||')  '||FROM||'       ',USER);
            CALL SEND(BUFFER||CR||LF,USER);
            IF (PLIRETV^=0) THEN
               CALL TPUT('MESSAGE CANNOT SEND TO USER: '
                  ||USER,1);
            ELSE DO;
               CALL TPUT('MESSAGE SENT TO USER: '||USER,
                  1);
               COLLECTIDS=COLLECTIDS||TRIM(USER)||',';
               END;
            END;
      END;
      USERIDS=SUBSTR(COLLECTIDS,1,LENGTH(COLLECTIDS)-1);
      END;
   ELSE CONT=FALSE;
END;

/*************************************************/
/*                                               */
/*   THIS SUBROUTINE WILL TRIM OFF THE           */
/*   TRAILING BLANKS OF A STRING                 */
/*                                               */
/*************************************************/
TRIM:PROC(IN) RETURNS(CHAR(256) VARYING);
   DCL IN              CHAR(*);
   DCL I               FIXED BIN(15);
   DO I=LENGTH(IN) TO 1 BY -1
      UNTIL(SUBSTR(IN,I,1)^=' ');
   END;
   RETURN(SUBSTR(IN,1,I));
END; /* TRIM */

/*************************************************************/
/*                                                           */
/*    THIS PROCEDURE IS THE LOWEST LEVEL TERMINAL INPUT      */
/*    ROUTINE. THE PARAMETER CIN IS FILLED WITH L            */
/*    CHARACTERS FROM THE KEYBOARD WHEN THIS ROUTINE IS      */
/*    CALLED.                                                */
/*                                                           */
/*************************************************************/
TGETL:PROC(CIN,L);
   DCL CIN                    CHAR(*);
   DCL T256                   CHAR(256);
   DCL L                      FIXED BIN(15);
   ON ENDFILE(SYSIN) BEGIN;
       T256=' ';
       CLOSE FILE(SYSIN);
       OPEN FILE(SYSIN);
       PUT FILE(SYSPRINT) SKIP;
       END;
   GET FILE(SYSIN) EDIT(T256) (A(256));
       CIN=SUBSTR(T256,1,L);
   END; /* TGETL */
END; /* SEND_MESS */

/*************************************************/
/*                                               */
/*   THIS PROCEDURE GETS A LOG RECORD FROM LOG   */
/*   FILE AND KEEPS TRACK OF THE SESSION DATE.   */
/*   IT RETURNS THE INDEX NUMBER OF NEXT LOG     */
/*   RECORD.                                     */
/*                                               */
/*************************************************/
GET_LOG:PROC(LOGNUM,LOGDATE);
   DCL LOGNUM                      FIXED BIN(15);
   DCL LOGDATE                     PICTURE '999999';
   READ NOLOCK FILE(LOGFIL) INTO(LOGREC) KEY(LOGNUM);
   LOGNUM=LOGNUM+1;
   IF (LOGREC.DELKEY^=(8)'1'B)&(LOGREC.TYPE='1'|
     LOGREC.TYPE='2'|LOGREC.TYPE='3'|LOGREC.TYPE='4'|
     LOGREC.TYPE='5') THEN
      LOGDATE=LOGREC.SID;
END; /* GET_LOG */

/*****************************************************************/
/*                                                               */
/*     THIS ROUTINE ACCEPTS A 6 CHARACTER VALUE REPRESENTING     */
/*     A DATE AS YYMMDD AND RETURNS THE SAME DATE VALUE WITH     */
/*     THE YEARS,MONTHS,DAYS SEPARATED BY SLASH. YY/MM/DD        */
/*                                                               */
/*****************************************************************/
      SEPARATE_DATE:PROC(DATEARG) RETURNS(CHAR(8));
         DCL DATEARG              PICTURE '999999';
         RETURN(SUBSTR(DATEARG,1,2)||'/'||SUBSTR(DATEARG,3,2)||
                '/'||SUBSTR(DATEARG,5,2));
      END; /* SEPARATE_DATE */

      /*************************************************/
      /*                                               */
      /*   THIS PROCEDURE DECODES A LOG RECORD INTO    */
      /*   DESCRIPTIVE FORM AND HANDLES THE SPECIAL    */
      /*   LOG RECORD TYPE (I.E. 1,2,3,4)              */
      /*                                               */
      /*************************************************/
      ONE_OUT:PROC(LOGDATE);
         DCL LOGDATE              PICTURE '999999';
         DCL FOUND                BIT(1);
         DCL COUT                 CHAR(122);
         ON CONVERSION BEGIN;
            GOTO OUT;
            END;
         IF (LOGREC.TYPE='1'|LOGREC.TYPE='2'|LOGREC.TYPE='3'|
            LOGREC.TYPE='4'|LOGREC.TYPE='5') THEN DO;
```

```
   PUT STRING(COUT) EDIT(TYPE(LOGREC.TYPE)||' '||
      LOGREC.ID||' '||SEPARATE_DATE(LOGDATE),
      SEPARATE(LOGREC.CTIME))(A,X(81),A);
   CALL TPUT(COUT,1);
   END;
ELSE DO;
   PUT STRING(COUT) EDIT(TYPE(LOGREC.TYPE),LOGREC.ID,
      NAME(LOGREC.ID),TSC(LOGREC.TS),PSC(LOGREC.PS),
      LOGREC.UNIT,LOGREC.Q1,LOGREC.Q2,LOGREC.Q3,
      LOGREC.TP,LOGREC.PP,SEPARATE(LOGREC.CTIME))
      (5(A,X(1)),F(3,0),3 F(4,0),2 F(8,3),X(1),A);
   CALL TPUT(COUT,1);
   IF (LOGREC.TYPE='J'|LOGREC.TYPE='K'|
      LOGREC.TYPE='L') THEN DO;
      CALL TPUT('MARKED BY        '||LOGREC.SID||' '||
         NAME(LOGREC.SID),1);
      END;
   END;
OUT:;

/*************************************************/
/*                                               */
/*   THIS SUBROUTINE RETURNS A NAME              */
/*   FROM HIS/HER STUDENT ID.                    */
/*                                               */
/*************************************************/
NAME:PROC(ID) RETURNS(CHAR(30));
   DCL ID                          CHAR(7);
   DCL (LOW,MID,HI)                FIXED BIN(15);
   LOW=1;
   HI=STEND;
   DO WHILE(LOW<=HI);
      MID=(LOW+HI)/2;
      IF (ST(MID).ID=ID) THEN
         RETURN(ST(MID).NAME);
      IF (ST(MID).ID>ID) THEN
         HI=MID-1;
      ELSE
         LOW=MID+1;
   END; /* WHILE */
   RETURN('UNKNOWN STUDENT');
END; /* NAME */

/*************************************************/
/*                                               */
/*   THIS SUBROUTINE RETURNS A DESCRIPTIVE       */
/*   FORM OF LOG TYPE BY DECODING THE LOG        */
/*   TYPE.                                       */
/*                                               */
/*************************************************/
TYPE:PROC(C) RETURNS(CHAR(16));
   DCL C                        CHAR(1);
   SELECT(C);
      WHEN('1') RETURN('CREATE LOG');
      WHEN('2') RETURN('START PSI');
      WHEN('3') RETURN('NEW SESSION');
      WHEN('4') RETURN('END SESSION');
      WHEN('5') RETURN('END PSI');
      WHEN('T') RETURN('GENERATE TEST');
      WHEN('C') RETURN('CANCEL TEST');
      WHEN('I') RETURN('WANT TEST MARKED');
      WHEN('P') RETURN('PROCTOR SELECTED');
      WHEN('J') RETURN('PASS');
      WHEN('K') RETURN('CONDITIONAL PASS');
      WHEN('L') RETURN('RESTUDY');
      OTHERWISE RETURN('UNKNOWN TYPE');
      END;
   END; /* TYPE */
END; /* ONE_OUT */

/*************************************************/
/*                                               */
/*   THIS SUBROUTINE CONTINOUSLY MONITORS THE    */
/*   STUDENT ACTIVITIES ACCORDING TO THE TRANSACTIONS */
/*   ONTO THE LOG FILE. IT WILL WATCH THE LOG FILE IN */
/*   IN EVERY 5 SECONDS OF TIME INTERVAL.        */
/*   THIS SUBROUTINE CAN BE STOPPED BY TURNING OFF */
/*   THE WATCHING FROM THE MONITOR MENU.         */
/*   THIS SUBROUTINE WILL START WATCHING THE STUDENT */
/*   ACTIVITIES FROM LAST CALL OR FROM THE SESSION */
/*   OF THE CURRENT DATE IF THIS SUBROUTINE IS FIRST */
/*   CALLED RIGHT AFTER PSI IS EXECUTED. THIS    */
/*   SUBROUTINE WILL REDUCE ITS PRIORITY WHEN    */
/*   NO RECORD WRITTEN ON THHE LOG FILE BY EVERY */
/*   10 MINUTES AND TURN ITSELF OFF WHEN AFTER HALF */
/*   AN HOUR OF IDLE TIME.                       */
/*                                               */
/*************************************************/
WATCH_LOG:PROC;
   DCL WHERE                FIXED BIN(15) EXTERNAL;
   DCL ALLOW                BIT(1) EXTERNAL;
   DCL LOGDATE              PICTURE '999999';
   DCL (HH1,MM1,HH2,MM2)    PICTURE '99';
   DCL (WAIT,TIMES)         FIXED BIN(15);
   DCL HEADR1               CHAR(56) INIT(
      'TYPE             STUD.#  STUDENT NAME');
   DCL HEADR2               CHAR(64) INIT(
'TEST STATE PROCTOR STATE UNIT   QUESTIONS    TEST PROCTOR   TIME');
   CALL TPUT('CONTINUE TO WATCH STUDENT ACTIVITIES.',1);
   CALL TPUT('',1);
   HH1=SUBSTR(TIME,1,2);
   MM1=SUBSTR(TIME,3,2);
   TIMES=0;
   DO WHILE(ALLOW);
      CALL READ_SYSP;
```

```
      IF (WHERE<=SYSP.NLOGREC) THEN DO;
         HH1=SUBSTR(TIME,1,2);
         MM1=SUBSTR(TIME,3,2);
         TIMES=0;
         CALL SCN(STRING(BEL));
         IF (SYSP.NLOGREC-WHERE > 10) THEN
            CALL TPUT(HEADR1||HEADR2,1);
         DO WHILE(WHERE<=SYSP.NLOGREC);
            CALL GET_LOG(WHERE,LOGDATE);
            CALL ONE_OUT(LOGDATE);
         END;
         CALL TPUT('',1);
         END;
       HH2=SUBSTR(TIME,1,2);
       MM2=SUBSTR(TIME,3,2);
       WAIT=(HH2-HH1)*60+MM2-MM1;
       IF (WAIT>=10) THEN DO;
          CALL SCN(STRING(BEL));
          TIMES=TIMES+1;
          IF (TIMES=3) THEN DO;
             CALL TPUT('NO STUDENT ACTIVITIES WITHIN HALF '
               ||'AN HOUR.',1);
             CALL TPUT('WATCHING IS AUTOMATICALLY'
               ||' TURN OFF.',1);
             ALLOW='0'B;
             END;
          ELSE DO;
             CALL TPUT('NO STUDENT ACTIVITIES WITHIN 10 '||
                'MINUTES.',1);
             IF (TIMES=2) THEN
                CALL TPUT('WATCHING WILL BE TURN OFF'
                   ||' AFTER 10 MINUTES.',1);
             PRIORITY(WATCH)=-50;
             CALL SCN(STRING(BEL));
             HH1=SUBSTR(TIME,1,2);
             MM1=SUBSTR(TIME,3,2);
             END;
          CALL TPUT('',1);
          END;
        DELAY(5000);
    END;
END; /* WATCH_LOG */

/*********************************************************/
/*                                                       */
/*   THIS PROCEDURE PROVIDES FACILITIES TO LIST          */
/*   THE LOG FILE FROM A GIVEN START DATE TO AN          */
/*   END DATE AND CONTINUOUSLY MONITORS THE STUDENTS     */
/*   FROM LOOKING AT THE LOG FILE CONTINUOUSLY.          */
/*                                                       */
/*********************************************************/
 MONITOR:PROC(CHOICE);
    DCL CHOICE                          CHAR(*);
    DCL (CONTINUE,SKIPIN,ERR)           BIT(1);
    DCL (FIRST,REST)                    CHAR(10);
    DCL HEADR1                          CHAR(56) INIT(
       'TYPE            STUD.#  STUDENT NAME');
    DCL HEADR2                          CHAR(64) INIT(
'TEST STATE PROCTOR STATE UNIT   QUESTIONS    TEST PROCTOR   TIME');
    DCL LOGDATE                         PICTURE '999999';
    DCL WHERE                           FIXED BIN(15) EXTERNAL;
    DCL DUMP                            BIT(1) STATIC INIT('0'B);
    DCL GETNAME                         BIT(1) STATIC INIT('0'B);
    ON SUBSCRIPTRANGE BEGIN;
       DCL 1 TEMP(LST) LIKE ST;
       DCL M FIXED BIN(15);
       TEMP=ST;
       LST=LST+LSTMORE;
       FREE ST;
       ALLOC ST;
       DO M=1 TO STEND-1;
          ST(M)=TEMP(M);
       END;
       GOTO AGAIN;
       END;
    SKIPIN = FALSE;
    IF( CHOICE^= ' ')THEN DO;
       SKIPIN=TRUE;
       END;
    CONTINUE = TRUE;
    DO WHILE(CONTINUE);
       IF(^SKIPIN) THEN DO;
          CALL TPUT('<7.>1 LIST LOG FILE.      '||
                    '<7.>2 WATCH STUDENT ACTIVITIES.',1);
          CALL TPUT('<7.>3 TURN OFF WATCHING.  '||
                    '<7.>4 RETURN TO MAIN MENU.',1);
          CALL TPUT('ENTER CHOICE==>    : ',1);
          CALL TGET(CHOICE,L_CHOICE,FALSE);
          IF (TURNOFF & (CHOICE=' ')) THEN RETURN;
          END;
       SKIPIN=FALSE ;
       CALL EXTRACT_CHOICE(CHOICE,FIRST,REST,ERR);
       IF(ERR)THEN FIRST='X';
       IF (^GETNAME) THEN DO;
          GETNAME='1'B;
          ON ENDFILE(STUDFIL) GOTO EOF;
          CALL CLOSE_FILE(STUDFIL);
          OPEN FILE(STUDFIL) INPUT SEQL BUFFERED;
          STEND=0;
          DO WHILE('1'B);
             READ FILE(STUDFIL) INTO(STUD);
             STEND=STEND+1;
(SUBRG): AGAIN: ST(STEND)=STUD, BY NAME;
          END;
      EOF: CALL CLOSE_FILE(STUDFIL);
          OPEN FILE(STUDFIL) EXCL UPDATE DIRECT UNBUF;
```

```
   END;
SELECT(FIRST);
WHEN('1') DO; /* LIST LOG FILE */
   CALL LIST_LOG(REST);
   END;
WHEN('2') DO; /* WATCH LOG FILE */
   IF (^DUMP) THEN DO;
      DUMP='1'B;
      CALL READ_SYSP;
      WHERE=FIND_LOG_REC(CURDATE,SYSP.NLOGREC,1,-1);
      CALL TPUT(HEADR1||HEADR2,1);
      ON ATTENTION BEGIN;
         ON ATTENTION SYSTEM;
         CALL GET_RESP('(C)ONTINUE OR (Q)UIT  : ',
            CQ,RESP,1);
         IF (RESP=2) THEN GOTO OUT;
                     ELSE GOTO CONT;
         END;
      DO WHILE(WHERE<=SYSP.NLOGREC);
         CALL GET_LOG(WHERE,LOGDATE);
         CALL ONE_OUT(LOGDATE);
      CONT:;
      END;
      END;
   IF (TURNOFF) THEN
      IF COMPLETION(STAGE) THEN DO;
         ALLOW='1'B;
         CALL WATCH_LOG TASK(WATCH) EVENT(STAGE)
            PRIORITY(-200);
         CALL TPUT('START WATCHING STUDENT '||
           'ACTIVITIES.',1);
         TURNOFF='0'B;
         END;
      ELSE
         CALL TPUT('WATCHING IS TURNING OFF, '||
            'TRY TURN ON LATER.',1);
   ELSE
      IF COMPLETION(STAGE) THEN DO;
         IF (ALLOW) THEN
            CALL TPUT('WATCHING IS TURNING ON, '||
               'BE PATIENT.',1);
         ELSE DO;
            CALL TPUT('NO STUDENT ACTIVITIES WITHIN '||
               'HALF AN HOUR, WATCHING IS TURN OFF BY '
               ||'ITSELF.',1);
            TURNOFF='1'B;
            END;
         END;
      ELSE
         CALL TPUT('WATCHING IS ALREADY ON.',1);
   OUT: ON ATTENTION SYSTEM;
   END;
   WHEN('3') DO; /* TURN OFF WATCHING */
      IF (^TURNOFF) THEN DO;
         IF (^COMPLETION(STAGE)) THEN DO;
            CALL TPUT('TURNING OFF WATCHING.',1);
            ALLOW='0'B;
            END;
         ELSE
            CALL TPUT('NO STUDENT ACTIVITIES WITHIN '||
               'HALF AN HOUR, WATCHING IS TURN OFF BY '||
               'ITSELF.',1);
         TURNOFF='1'B;
         END;
      ELSE
         IF (^COMPLETION(STAGE)) THEN
            CALL TPUT('WATCHING IS TURNING OFF, '||
               'SECOND TURN OFF IS REJECTED.',1);
         ELSE
            CALL TPUT('WATCHING IS ALREADY OFF.',1);
      END;
   WHEN('0','4') DO; /* EXIT */
       CONTINUE = FALSE;
       END;
   OTHERWISE DO; /* BAD CHOICE */
      CALL TPUT('INVALID CHOICE: '||CHOICE,1);
      END;
   END; /*SELECT */
   CALL TPUT('ENDING AT: '||SEPARATE(SUBSTR(TIME,1,6)),1);
END; /* DO LOOP */

/*************************************************************/
/*                                                           */
/*   THIS SUBROUTINE ASKS FOR A STARTING DATE AND            */
/*   AN ENDING DATE IN ORDER TO LIST OUT THE LOG             */
/*   RECORDS. THE DATE SYNTAX IS YY/MM/DD WHICH IS           */
/*   FROM 00/01/01 TO 99/12/31. IF THE END DATE IS           */
/*   SMALLER THAN THE START DATE, NOTHING WILL BE            */
/*   LISTED. THE DEFAULT OF END DATE IS THE CURRENT          */
/*   DATE WHICH IS THE LAST DATE RECORDED ON THE LOG         */
/*   FILE.                                                   */
/*                                                           */
/*************************************************************/
   LIST_LOG:PROC(CHOICE);
     DCL CHOICE                          CHAR(*);
     DCL (STRT_DATE,END_DATE,LOGDATE) PICTURE '999999';
     DCL MSGSTART                        CHAR(44) INIT(
        'LIST LOG STARTING AT (YY/MM/DD) DATE      : ');
     DCL MSGEND                          CHAR(42) INIT(
        'LIST LOG ENDING AT (YY/MM/DD) DATE      : ');
     DCL (LOGNUM,REC)                    FIXED BIN(15);
     DCL OUTVAL                          PICTURE 'ZZZZZ9';
     CALL TPUT('CURRENT DATE IS: '||
        SEPARATE_DATE(CURDATE),1);
     CALL PROMPT_DATE(MSGSTART,0,STRT_DATE);
```

```
      IF (STRT_DATE=0) THEN RETURN;
      CALL PROMPT_DATE(MSGEND,CURDATE,END_DATE);
      CALL READ_SYSP;
      LOGNUM=FIND_LOG(STRT_DATE,END_DATE,CURDATE);
      REC=0;
      CALL GET_LOG(LOGNUM,LOGDATE);
      ON ATTENTION BEGIN;
         ON ATTENTION SYSTEM;
         CALL GET_RESP('(C)ONTINUE OR (Q)UIT  : ',CQ,RESP,1);
         IF (RESP=2) THEN GOTO OUT;
                     ELSE GOTO CONT;
         END;
      CALL TPUT(HEADR1||HEADR2,1);
      DO WHILE(STRT_DATE<=LOGDATE & END_DATE>=LOGDATE &
         LOGNUM<=SYSP.NLOGREC+1);
         REC=REC+1;
         CALL ONE_OUT(LOGDATE);
   CONT: CALL GET_LOG(LOGNUM,LOGDATE);
      END;
      OUTVAL=REC;
      CALL TPUT('LOG RECORD LISTED:'||OUTVAL,1);
OUT: ON ATTENTION SYSTEM;
   OUTVAL=SYSP.NLOGREC+1;
   CALL TPUT('TOTAL LOG RECORD:'||OUTVAL,1);

   /*****************************************************/
   /*                                                   */
   /*   THIS SUBROUTINE WILL PROMPT FOR A DATE          */
   /*   AND VERIFY THE SYNTAX. SYNTAX IS YY/MM/DD.      */
   /*                                                   */
   /*****************************************************/
      PROMPT_DATE:PROC(MSG,INDATE,OUTDATE);
         DCL MSG                        CHAR(*);
         DCL (INDATE,OUTDATE)           PICTURE '999999';
         DCL (TLOOP,ERR)                BIT(1);
         DCL TEMPIN                     CHAR(8);
         TLOOP=TRUE;
         DO WHILE(TLOOP);
            IF (INDATE=0) THEN
               CALL TPUT(MSG,1);
            ELSE DO;
               CALL TPUT(MSG||SEPARATE_DATE(INDATE),1);
               CALL TPUT(MSG,0);
               END;
            CALL TGET(TEMPIN,8,FALSE);
            IF (VERIFY(TEMPIN,' ')=0) THEN DO;
               TLOOP=FALSE;
               OUTDATE=INDATE;
               END;
            ELSE DO;
               CALL PARSE_DATE(TEMPIN,ERR);
               IF (^ERR) THEN DO;
                  OUTDATE=SUBSTR(TEMPIN,1,2)||
                     SUBSTR(TEMPIN,4,2)||SUBSTR(TEMPIN,7,2);
                  TLOOP=FALSE;
                  END;
               ELSE
                  CALL TPUT('INVALID DATE SYNTAX, RE-ENTER.'
                     ,1);
               END;
            END; /* WHILE */
         END; /* PROMPT_DATE */

/*****************************************************/
/*                                                   */
/*   THIS SUBROUTINE VERIFIES THAT THE DATE TYPED IN */
/*   A CORRECT FORMAT. THE DATE VALUE TO BE VERIFIED */
/*   IS PASSED IN DATEARG WHILE THE ERR INDICATING   */
/*   AN ERROR WAS DISCOVERED.                        */
/*                                                   */
/*****************************************************/
      PARSE_DATE:PROC(DATEARG,ERR);
         DCL DATEARG                    CHAR(8);
         DCL ERR                        BIT(1);
         DCL DATE                       CHAR(8) VARYING;
         DATE=DATEARG;
         IF (INDEX(DATE,'/')=0) THEN DO;
            IF (SUBSTR(DATE,2)=' ') THEN
               DATE='0'||DATE;
            SUBSTR(DATE,3)='/01/01';
            END;
         ELSE DO;
            IF (SUBSTR(DATE,2,1)='/') THEN
               DATE='0'||DATE;
            IF (INDEX(SUBSTR(DATE,4),'/')=0) THEN DO;
               IF (SUBSTR(DATE,5)=' ') THEN
                  DATE=SUBSTR(DATE,1,3)||'0'||SUBSTR(DATE,4);
                  SUBSTR(DATE,6)='/01';
               END;
            ELSE DO;
               IF (SUBSTR(DATE,5,1)='/') THEN
                  DATE=SUBSTR(DATE,1,3)||'0'||SUBSTR(DATE,4);
               IF (SUBSTR(DATE,8)=' ') THEN
                  DATE=SUBSTR(DATE,1,6)||'0'||SUBSTR(DATE,7);
               END;
            END;
         ERR=FALSE;
         ERR = CHECK_DIG(SUBSTR(DATE,1,2),'99');
         ERR = ERR | CHECK_DIG(SUBSTR(DATE,4,2),'13')
                   | '00'=SUBSTR(DATE,4,2);
         ERR = ERR | CHECK_DIG(SUBSTR(DATE,7,2),'32')
                   | '00'=SUBSTR(DATE,7,2);
         ERR = ERR | (SUBSTR(DATE,3,1)^='/')
                   | (SUBSTR(DATE,6,1)^='/');
         IF (ERR) THEN
```

```
         CALL TPUT('INVALID DATE: '||DATEARG,1);
      ELSE
         DATEARG=DATE;
      CHECK_DIG:PROC(DIG2,LIM) RETURNS(BIT(1));
         DCL (DIG2,LIM)      CHAR(2);
         DCL ERR             BIT(1);
         ERR = FALSE;
         IF (VERIFY(DIG2,NUMERIC)=0 ) THEN DO;
            IF (DIG2 >= LIM ) THEN DO;
               ERR = TRUE;
               END;
             END;
          ELSE ERR= TRUE;
          RETURN(ERR);
       END; /* CHECK_DIG */
   END; /*PARSE DATE*/

   /*************************************************/
   /*                                               */
   /*   THIS SUBROUTINE FINDS THE INDEX OF THE      */
   /*   LOG RECORD WITH THE STARTING DATE OF        */
   /*   'START'. THIS SUBROUTINE WILL CHECK FOR     */
   /*   THE RANGE OF 'START' DATE. IT SHOULD BE     */
   /*   BETWEEN THE CREATION DATE OF LOG FILE       */
   /*   AND THE LAST DATE USED ON THE LOG FILE.     */
   /*   IF NOT, IT RETURNS 0.                       */
   /*                                               */
   /*************************************************/
  FIND_LOG:PROC(START,END,CURDATE) RETURNS(FIXED BIN(15));
      DCL (START,END,CURDATE) PICTURE '999999';
      DCL (CREATDATE,MIDDATE) PICTURE '999999';
      IF (START>END) THEN RETURN(0);
      READ NOLOCK FILE(LOGFIL) INTO(LOGREC) KEY(0);
      CREATDATE=LOGREC.SID;
      IF ^(CREATDATE<START & START<=CURDATE) THEN
         RETURN(0);
      ELSE DO;
         MIDDATE=(CREATDATE+CURDATE)/2;
         IF (START<=MIDDATE) THEN
            RETURN(FIND_LOG_REC(START,1,SYSP.NLOGREC,1));
         ELSE
            RETURN(FIND_LOG_REC(START,SYSP.NLOGREC,1,-1));
         END;
   END; /* FIND_LOG */
END; /* LIST_LOG */

/*************************************************/
/*                                               */
/*   THIS SUBROUTINE SCANS THE LOG FILE          */
/*   FROM THE BEGINNING OR FROM THE END          */
/*   UNTIL THE GIVEN DATE IS FOUND. IF           */
/*   THE GIVEN DATE IS AT THE LOWER PART         */
/*   OF THE LOG FILE, IT WILL START WITH         */
/*   THE BEGINNING OF THE LOG, OTHERWISE         */
/*   FROM THE END TO THE BEGINNING. THIS         */
/*   SUBROUTINE USES ASYNCHROUS FILE             */
/*   HANDLING TO REDUCE THE SEARCH TIME.         */
/*   WHEN INCR IS 1, INDICATES THE SEARCH        */
/*   IS FROM BEGINNING, -1 INDICATES FROM        */
/*   END TO THE BEGINNING.                       */
/*                                               */
/*************************************************/
FIND_LOG_REC:PROC(DATE,FROM,END,INCR)
   RETURNS(FIXED BIN(15));
   DCL DATE                  PICTURE '999999';
   DCL (FROM,END,INCR)       FIXED BIN(15);
   DCL (I,J,PTR)             FIXED BIN(15);
   DCL TYPE                  CHAR(1);
   DCL E(LPTR)               EVENT;
   IF LPTR > SYSP.NLOGREC THEN
      PTR=SYSP.NLOGREC+1;
   ELSE
      PTR=LPTR;
   DO I=FROM TO END BY INCR*PTR;
      DO J=1 TO PTR;
         READ NOLOCK FILE(LOGFIL) INTO(LOGS(J))
            KEY(I+(J-1)*INCR) EVENT(E(J));
      END;
      DO J=1 TO PTR;
         WAIT(E(J));
         TYPE=LOGS(J).TYPE;
         IF (TYPE='2'|TYPE='3'|TYPE='4'|TYPE='5') THEN DO;
            IF (INCR>0 & LOGS(J).SID>=DATE) THEN
               RETURN(I+(J-1)*INCR);
            IF (INCR<0 & LOGS(J).SID<DATE) THEN
               RETURN(FIND_LOG_REC(DATE,I+(J-1)*INCR,
                  FROM,1));
            END;
      END; /* WHILE */
   END;
   DO J=I-INCR*PTR TO END BY INCR;
     READ NOLOCK FILE(LOGFIL) INTO(LOGREC) KEY(J);
      TYPE=LOGREC.TYPE;
      IF (TYPE='2'|TYPE='3'|TYPE='4'|TYPE='5') THEN DO;
         IF (INCR>0 & LOGREC.SID>=DATE) THEN
            RETURN(J);
         IF (INCR<0 & LOGREC.SID<DATE) THEN
            RETURN(FIND_LOG_REC(DATE,J,FROM,1));
         END;
      END; /* WHILE */
      RETURN(0);
   END; /* FIND_LOG_REC */
END; /* MONITOR */

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

```
      /*                                                    */
      /*     THIS PROCEDURE IS THE MAIN ROUTINE WHICH       */
      /*     HANDLES STUDENT TRANSACTION PROCESSING.        */
      /*     THIS ROUTINE LOOPS CALLING THE STUD_LOGIN      */
      /*     SUBROUTINE TO LOGIN THE NEXT STUDENT. (IF      */
      /*     THE INST,TA,OR MARKER SIGNS ON AS THE NEXT     */
      /*     STUDENT THEN CONTROL IS TRANSFERED OUT OF      */
      /*     THIS ROUTINE TO THE MAIN CONTROL MENU OR       */
      /*     MARK TEST SUBROUTINE). AFTER SIGNING ON THE    */
      /*     NEXT STUDENT THIS ROUTINE LOOKS AT HIS         */
      /*     INTERNAL STATE TO DETERMINE WHICH PROMPTS      */
      /*     ARE PRINTED. THE ARRAY CASE_TBL IS USED TO     */
      /*     DETERMINE WHICH OF THE THREE ALTERNATIVES      */
      /*     OF THE SELECT STATEMENT INTERNAL TO THIS       */
      /*     ROUTINE IS CHOSEN. ONE ALTERNATIVE IS THAT     */
      /*     THE STUDENT HAS NO UNMARKED TEST AND HE        */
      /*     SHOULD BE PROMPTED TO GENERATE THE NEXT ONE    */
      /*     THE NEXT ALTERNATIVE IS TAKEN IF HE HAS        */
      /*     GENERATED A TEST BUT NOT SELECTED PROCTORS.    */
      /*     THE THIRD ALTERNITIVE IS THAT HE HAS BEEN      */
      /*     SELECTED TO PROCTOR SOMEONE ELSES TEST AND     */
      /*     HE MUST NOW ENTER IN THE MARK. THE NULPROC     */
      /*     VARIABLE IS SET TO TRUE IF THE CURRENT         */
      /*     STUDENT HAS BEEN SELECTED AS A PROCTOR BUT     */
      /*     THE TEST HE HAS BEEN ASKED TO PROCTER HAS      */
      /*     ALREADY BEEN MARKED BY THE INSTRUCTOR OR       */
      /*     HAS BEEN CANCELLED. THIS CAUSE THE STUDENT     */
      /*     LOGIN PROCEDURE TO BE SKIPPED THE NEXT TIME    */
      /*     THROUGH THE LOOP AND ALLOWS THE CURRENTLY      */
      /*     LOGGED IN STUDENT TO PERFORM ANOTHER  TYPE     */
      /*     OF TRANSACTION.                                */
      /*                                                    */
      /******************************************************/
      SESSION_CONTROL:PROC;
         DCL (BLANKF,FOUND,NOTMARKER)       BIT(1);
         DCL CASE_TBL(L_PST,L_TST)          CHAR(1) INIT(
            'T','P','T','P','P','P','T','P','P','T',
            'T','P','T','P','P','P','T','P','P','T',
            'T','P','T','P','P','P','T','P','P','T',
            'M','E','M','M','M','M','M','E','M','M');
         DCL CID                            CHAR(7);
         DCL (SLOOP,PROCEDE,NULL_PROC,NONE)     BIT(1);
         SLOOP=TRUE;
         NULL_PROC=FALSE;
REPEAT:  DO WHILE(SLOOP);
            NOTMARKER=TRUE;
            IF (^NULL_PROC) THEN DO;
               CALL STUD_LOGIN(CID);
               IF ((CID='INST') | (CID='TA')) THEN
                  SLOOP = FALSE;
               ELSE IF (CID='MARKER') THEN DO;
                  CALL MARK_TEST(' ');
                  NOTMARKER=FALSE;
                  END;
               ELSE DO;
                  CALL PRINT_INFO(FOUND);
                  IF (^FOUND) THEN GOTO REPEAT;
                  END;
               END;
            IF (SLOOP&NOTMARKER) THEN DO;
               IF (STUD.PSTATE^=PST_P & ^NULL_PROC) THEN DO;
                  CALL GET_PROC_STATUS(FOUND);
                  IF (^FOUND) THEN GOTO REPEAT;
                  END;
               NULL_PROC=FALSE;
               CALL READ_STUD(STUD.ID,FOUND);
               IF (^FOUND) THEN DO;
                  CALL REC_NO_FOUND;
                  GOTO REPEAT;
                  END;
               SELECT(CASE_TBL(STUD.PSTATE,STUD.TSTATE));
               WHEN('T') CALL ASK_TEST; /*TEST ASK*/
               WHEN('P') CALL SELECT_PROC; /*PROCTOR SELECT */
               WHEN('M') CALL MARK; /*MARK TEST*/
               OTHERWISE CALL INTERNAL_ERR(5); /*ERROR */
               END; /*SELECT*/
               IF (^NULL_PROC) THEN DO;
                  CALL TPUT('OK   TRANSACTION COMPLETE.',1);
                  CALL TPUT('ENDING AT: '||
                     SEPARATE(SUBSTR(TIME,1,6)),1);
                  END;
            END; /* IF */
         END; /*WHILE*/
/* LOGICAL END OF SESSION CONTROL */

      /*****************************************/
      /*                                       */
      /*   ASKS FOR TEST GENERATION            */
      /*                                       */
      /*****************************************/
      ASK_TEST:PROC;
         DCL TMSG                       CHAR(22) INIT(
         'GENERATE TEST ON UNIT ');
         DCL TMSG2                      CHAR(19) INIT(
         ' (Y)ES,(N)O?     : ');
         DCL IOUT                       PICTURE 'ZZZ9';
         DCL YN(2)                      CHAR(1) INIT('Y','N');
         DCL (NOW,STOP)                 PICTURE '(12)9';
         DCL TRESP                      FIXED BIN(15);
         PROCEDE = TRUE;
         CALL READ_SYSP;
         IF (STUD.TSTATE=TST_R3)|(STUD.TSTATE=TST_RC) THEN
            CALL RESTUDY_CHECK(PROCEDE);
         IF (SYSP.SESSTIME>SYSP.SESSCUTOFF) THEN
            STOP=(SYSP.SESSDATE+1)||SYSP.SESSCUTOFF;
```

```
            ELSE
               STOP=SYSP.SESSDATE||SYSP.SESSCUTOFF;
            NOW=DATE||SUBSTR(TIME,1,6);
            IF (NOW>STOP) THEN DO;
               PROCEDE=FALSE;
               CALL TPUT('TEST CUT-OFF TIME REACHED,',1);
               CALL TPUT('NO MORE TESTS WILL BE '||
               'ISSUED FOR THE REMAINDING CLASS.',1);
               END;
            IF (PROCEDE) THEN DO;
               IOUT=STUD.UNIT+1;
               CALL GET_RESP(TMSG||IOUT||TMSG2,YN,TRESP,2);
               IF (TRESP=1) THEN DO; /*GEN TEST*/
                  CALL READ_STUD(STUD.ID,FOUND);
                  IF (^FOUND) THEN DO;
                     CALL REC_NO_FOUND;
                     RETURN;
                     END;
                  IF (CASE_TBL(STUD.PSTATE,STUD.TSTATE)^='T')
                     THEN DO;
                     NULL_PROC=TRUE;
                     RETURN;
                     END;
                  CALL GENTEST(FOUND);
                  END;
               END;
         END; /* ASK_TEST */

         /*************************************/
         /*                                   */
         /*   REQUIRES FOR PROCTORS           */
         /*                                   */
         /*************************************/
         SELECT_PROC:PROC;
            DCL (PID1,PID2)            CHAR(7);
            DCL CP(2)                  CHAR(1) INIT('C','P');
            DCL PMSG1                  CHAR(56) INIT(
   'DO YOU WANT YOUR TEST (C)ANCELLED OR (P)ROCTORED?     : ');
            DCL PMSG2                  CHAR(60) INIT(
'YOU MUST HAVE YOUR CURRENT TEST PROCTORED BEFORE PROCEEDING.');
            DCL IPROCMSG               CHAR(37) INIT(
            'PROCTOR SELECTED IS INSTRUCTOR OR TA.');
            DCL PS1A                   CHAR(34) INIT(
            'FIRST PROCTOR SELECTED, STUDENT#: ');
            DCL PS2A                   CHAR(35) INIT(
            'SECOND PROCTOR SELECTED, STUDENT#: ');
            DCL PRESP                  FIXED BIN(15);
            DCL (NOW,STOP)             PICTURE '(12)9';
            IF (STUD.TSTATE=TST_T)|(STUD.TSTATE=TST_OT) THEN DO;
               CALL GET_RESP(PMSG1,CP,PRESP,0);
               CALL READ_STUD(STUD.ID,FOUND);
               IF (^FOUND) THEN DO;
                  CALL REC_NO_FOUND;
                  RETURN;
                  END;
               IF (^(CASE_TBL(STUD.PSTATE,STUD.TSTATE)='P' &
                (STUD.TSTATE=TST_T|STUD.TSTATE=TST_OT)))
               THEN DO;
               NULL_PROC=TRUE;
               RETURN;
               END;
            IF (PRESP=1) THEN DO;
               CALL STATE_MACH('C',CID);
               END;
            IF (PRESP=2) THEN DO;
               CALL FIND_PROCTORS(PID1,PID2,NONE);
               CALL READ_STUD_LOCK(FOUND);
               IF (^FOUND) THEN DO;
                  CALL REC_NO_FOUND;
                  RETURN;
                  END;
               IF (^(CASE_TBL(STUD.PSTATE,STUD.TSTATE)='P' &
                (STUD.TSTATE=TST_T|STUD.TSTATE=TST_OT)))
                  THEN DO;
                  NULL_PROC=TRUE;
                  UNLOCK FILE(STUDFIL) KEY(STUD.ID);
                  RETURN;
                  END;
               IF (NONE) THEN DO;
                  CALL TPUT('ALL AVAILABLE PROCTORS'
                   ||' ARE BUSY. TRY AGAIN LATER.',1);
                  UNLOCK FILE(STUDFIL) KEY(STUD.ID);
                  END;
               ELSE DO;
                  CALL STATE_MACH('I',CID);
                  IF (PID1='INST') THEN DO;
                     CALL TPUT(IPROCMSG,1);
                     END;
                  ELSE DO;
                     STUD.ID=PID1;
                     CALL READ_STUD_LOCK(FOUND);
                     IF (^FOUND) THEN
                        CALL INTERNAL_ERR(1);
                     IF (SYSP.SESSDATE<DATE)&((STUD.RTIME>0) &
                        (STUD.RTIME<=TWENTYM)) THEN
                        STOP=(SYSP.SESSDATE+1)||STUD.RTIME;
                     ELSE
                        STOP=SYSP.SESSDATE||STUD.RTIME;
                     NOW=DATE||SUBSTR(TIME,1,6);
                     IF ((STUD.PSTATE=PST_PA)&
                        (STUD.TSTATE^=TST_T)&
                        (STUD.TSTATE^=TST_OT)&
                        (^((STUD.TSTATE=TST_R3)&
                       (NOW<STOP))))
                       THEN DO;
```

```
      CALL STATE_MACH('P',CID);
      CALL TPUT(PS1A||PID1||
           ', NAME: '||STUD.NAME,1);
      STUD.ID=PID2;
      CALL READ_STUD_LOCK(FOUND);
      IF (^FOUND) THEN
         CALL INTERNAL_ERR(1);
      IF (SYSP.SESSDATE<DATE)&((STUD.RTIME>0
         )&(STUD.RTIME<=TWENTYM)) THEN
         STOP=(SYSP.SESSDATE+1)||STUD.RTIME;
      ELSE
         STOP=SYSP.SESSDATE||STUD.RTIME;
      IF ((STUD.PSTATE=PST_PA)&
         (STUD.TSTATE^=TST_T)&
         (STUD.TSTATE^=TST_OT)&
         (^((STUD.TSTATE=TST_R3)&
         (NOW<STOP))))
         THEN DO;
         CALL TPUT(PS2A||PID2||
           ', NAME: '||STUD.NAME,1);
         CALL STATE_MACH('P',CID);
         END;
      ELSE DO;
         UNLOCK FILE(STUDFIL) KEY(PID2);
         CALL READ_STUD(PID1,FOUND);
         CALL STATE_MACH('N',PID1);
         CALL TPUT('STUDENT#: '||
            PID1||' IS CANCELLED '||
            'TO BE A PROCTOR.',1);
         CALL READ_SYSP_LOCK;
         IF SYSP.CUR_I_TEST<I_TEST_LIM
            THEN DO;
            CALL TPUT(IPROCMSG,1);
            SYSP.CUR_I_TEST=
               SYSP.CUR_I_TEST+1;
            END;
         ELSE DO;
            CALL READ_STUD(CID,FOUND);
            CALL STATE_MACH('T',CID);
            CALL TPUT('ALL AVAILABE'||
                ' PROCTORS ARE '||
                'BUSY. TRY AGAIN LATER.',1);
            END;
         CALL UPDATE_SYSP;
         END;
      END;
   ELSE DO;
      UNLOCK FILE(STUDFIL) KEY(PID1);
      CALL READ_SYSP_LOCK;
      IF (SYSP.CUR_I_TEST<I_TEST_LIM)
         THEN DO;
         CALL TPUT(IPROCMSG,1);
         SYSP.CUR_I_TEST=SYSP.CUR_I_TEST+1;
                  END;
               ELSE DO;
                  CALL READ_STUD(CID,FOUND);
                  CALL STATE_MACH('T',CID);
                  CALL TPUT('ALL AVAILABLE'||
                      ' PROCTORS ARE '||
                      'BUSY. TRY AGAIN LATER.',1);
                  END;
               CALL UPDATE_SYSP;
               END;
            END; /*ELSE*/
         END; /*ELSE*/
      END; /*THEN*/
   END;
  ELSE DO; /* PROCTORS ALREADY SELECTED*/
      CALL TPUT(PMSG2,1);
      END;
END; /* SELECT_PROC */

/*************************************/
/*                                   */
/*   A PROCTOR MARKS A STUDENT       */
/*                                   */
/*************************************/
MARK:PROC;
   DCL SELMSG                CHAR(61) INIT(
'YOU HAVE BEEN SELECTED TO PROCTOR A TEST FOR STUDENT NUMBER: ');
   DCL NULMSG1               CHAR(53) INIT(
'A TEST YOU HAVE BEEN SELECTED TO PROCTOR HAS BEEN CAN' );
   DCL NULMSG2               CHAR(50) INIT(
'CELLED BY THE STUDENT OR MARKED BY THE INSTRUCTOR.');
   DCL MARKSTATE(L_TST)      BIT(1) INIT(
   '0'B,'0'B,'0'B,'1'B,'1'B,'1'B,'0'B,'0'B,'1'B,'0'B);
   IF (STUD.SID^=' ') THEN DO;
      SAVE=STUD;
      CALL READ_STUD(STUD.SID,FOUND);
      IF (^FOUND) THEN DO;
          CALL INTERNAL_ERR(3);
          END;
      CALL TPUT(SELMSG||SAVE.SID,1);
      CALL TPUT('STUDENT NAME:'||STUD.NAME,1);
      IF (MARKSTATE(STUD.TSTATE)) THEN DO;
         CALL GET_QUES_NUM(BLANKF);
         CALL GET_MARK(BLANKF,SAVE.ID,FOUND,MARKSTATE);
         IF (^FOUND) THEN DO;
            STUD=SAVE;
            CALL TPUT(NULMSG1||NULMSG2,1);
            NULL_PROC=TRUE;
            CALL STATE_MACH('N',CID);
            RETURN;
            END;
         IF (^BLANKF) THEN DO;
```

```
            STUD=SAVE;
            CALL STATE_MACH('M',CID);
            CALL GET_PROC_STATUS(FOUND);
            IF (^FOUND) THEN RETURN;
            END;
         END;
      ELSE DO;
         STUD=SAVE;
         CALL TPUT(NULMSG1||NULMSG2,1);
         NULL_PROC=TRUE;
         CALL STATE_MACH('N',CID);
         END;
      END;
   ELSE DO;
      CALL TPUT(NULMSG1||NULMSG2,1);
      NULL_PROC=TRUE;
      CALL STATE_MACH('N',CID);
      END;
END; /* MARK */

 /*****************************************************/
 /*                                                   */
 /*    THIS PROCEDURE PROMPTS THE TERMINAL FOR        */
 /*    A STUDENT NUMBER AND A PASSWORD AND            */
 /*    RETURNS ONCE A VALID STUDENT NUMBER ,          */
 /*    PASSWORD PAIR HAS BEEN ENTERED. THE            */
 /*    STUDENT NUMBER OF THE JUST LOGGED IN           */
 /*    STUDENT IS RETURN IN THE PARAMETER CID.        */
 /*                                                   */
 /*****************************************************/
 STUD_LOGIN:PROC(CID);
      @@@ confidential procedure @@@
 END; /*STUDENT LOGIN*/

 /*****************************************************/
 /*                                                   */
 /*    THIS PROCEDURE IS CALLED TO PRINT THE          */
 /*    <DO YOU WISH TO VIEW YOUR CURRENT COURSE*/
 /*    STANDING> PROMPT AND IT DISPLAYS THE           */
 /*    DATA IF THE RESPONSE IS AFFIRMATIVE.           */
 /*                                                   */
 /*****************************************************/
 PRINT_INFO:PROC(FOUND);
    DCL FOUND                BIT(1);
    DCL DRESP                FIXED BIN(15);
    DCL YN(2)                CHAR(1) INIT('Y','N');
    DCL DMSG                 CHAR(60) INIT(
'WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O     : ');
    DCL COUT1                CHAR(103);
    DCL COUT2                CHAR(38);
    CALL GET_RESP(DMSG,YN,DRESP,2);
    CALL READ_STUD(STUD.ID,FOUND);
    IF (^FOUND) THEN DO;
       CALL REC_NO_FOUND;
       RETURN;
       END;
    IF (DRESP=1) THEN DO;
       PUT STRING(COUT1) EDIT('CURRENT UNIT:',
        STUD.UNIT,', TEST POINTS:',STUD.TEST,
       ', PROCTOR POINTS:',STUD.PROCTOR,', TERM PROJECT:',
       STUD.TERM,', FINAL EXAM:',STUD.EXAM)
       (A,P'ZZ9',4 (A,F(7,2)));
       CALL TPUT(COUT1,1);
       PUT STRING(COUT2) EDIT('TOTAL POINTS:',
       STUD.TOTAL,', LETTER GRADE: ',STUD.LETTER)
       (A,F(7,2),A,A(2));
       CALL TPUT(COUT2,1);
       END;
 END; /* PRINT INFO */

/*****************************************************/
/*                                                   */
/*    THIS ROUTINE WRITES THE <PROCTOR? > PROMPT     */
/*    TO THE TERMINAL AND ALLOWS THE STUDENT TO      */
/*    CHANGE HIS PROCTOR AVAILABLITY STATUS.         */
/*                                                   */
/*****************************************************/
 GET_PROC_STATUS:PROC(FOUND);
    DCL YN(2)                CHAR(1) INIT('Y','N');
    DCL MSG                  CHAR(26) INIT(
    'PROCTOR? (Y)ES,(N)O     : ');
    DCL NOTMSG(4)             CHAR(4) VARYING
       INIT(' NOT','',' NOT','');
    DCL RESP                  FIXED BIN(15);
    DCL FOUND                 BIT(1);
    CALL GET_RESP(MSG,YN,RESP,0);
    CALL READ_STUD(STUD.ID,FOUND);
    IF (^FOUND) THEN DO;
       CALL REC_NO_FOUND;
       RETURN;
       END;
    SELECT(RESP);
    WHEN(1)DO; /* CHANGE TO PA */
       IF (STUD.PSTATE^=PST_PA) THEN DO;
          CALL READ_STUD_LOCK(FOUND);
          IF (^FOUND) THEN RETURN;
          STUD.PSTATE= PST_PA;
          CALL UPDATE_STUD(STUD.ID,FOUND);
          IF (^FOUND) THEN
             CALL INTERNAL_ERR(7);
          END;
       END;
    WHEN(2) DO; /* CHANGE TO PNA */
       IF ((STUD.PSTATE^=PST_PNA)|
          (STUD.PSTATE^=PST_I)) THEN DO;
```

```
         CALL READ_STUD_LOCK(FOUND);
         IF (^FOUND) THEN RETURN;
         STUD.PSTATE=PST_PNA;
         CALL UPDATE_STUD(STUD.ID,FOUND);
         IF (^FOUND) THEN
            CALL INTERNAL_ERR(7);
         END;
      END;
   WHEN(0) DO; /* LEAVE UNALTERED */
      END;
   OTHERWISE DO;
      CALL INTERNAL_ERR(8);
      RETURN;
      END;
   END; /* SELECT*/
   CALL TPUT('STUDENT#: '||STUD.ID||' IS'||
  NOTMSG(STUD.PSTATE)||' AVAILABLE FOR PROCTORING.',1);
END; /* GET PROC STATUS */

/*****************************************************/
/*                                                   */
/*    THIS PROCEDURE IS CALLED IF A STUDENT IS IN    */
/*    THE RESTUDY STATE TO CHECK IF HE HAS COM-      */
/*    PLETED ALL OF HIS RESTUDY TIME. THE BOOLEAN    */
/*    ARGUMENT PROCEDE IS RETURNED WITH THE VALUE    */
/*    FALSE IF THE STUDENT HASNT FINNISHED HIS 10    */
/*    MINUTES OF RESTUDY TIME.                       */
/*                                                   */
/*****************************************************/
RESTUDY_CHECK:PROC(PROCEDE);
   DCL PROCEDE                BIT(1);
   DCL (NOW,STOP)             PICTURE '(12)9';
   PROCEDE = TRUE;
   IF (SYSP.SESSDATE<DATE) &
      ((STUD.RTIME>0)&(STUD.RTIME<=TWENTYM)) THEN
      STOP=(SYSP.SESSDATE+1)||STUD.RTIME;
   ELSE
      STOP=SYSP.SESSDATE||STUD.RTIME;
   NOW=DATE||SUBSTR(TIME,1,6);
   IF (STOP>NOW) THEN DO;
      PROCEDE = FALSE;
      CALL TPUT('YOU HAVE NOT COMPLETED 10 '||
      'MINUTES OF RESTUDYING TIME.',1);
      END;
END; /*RESTUDY CHECK */

/*****************************************************/
/*                                                   */
/*    THIS PROCEDURE GENERATES A TEST ON THE NEXT    */
/*    UNIT AND PRINTS IT ON THE TERMINAL.            */
/*                                                   */
/*****************************************************/
GENTEST:PROC(FOUND);
   DCL FOUND                  BIT(1);
   DCL (TQ1,TQ2,TQ3,LIM)      FIXED BIN(15);
   DCL COUT                   CHAR(72);
   CALL READ_SYSP;
   IF (STUD.UNIT >= SYSP.NUNIT) THEN DO;
      CALL TPUT('YOU HAVE COMPLETED ALL THE UNITS.',1);
      END;
   ELSE DO;
      TQ1=TQ2=TQ3=0 ;
      LIM=SYSP.UNITL(STUD.UNIT+1);
      IF (LIM<3) THEN DO;
        CALL TPUT('TEST CANNOT BE GENERATED.',1);
        CALL TPUT('UNIT MUST HAVE AT LEAST 3 CHOICES'
          ,1);
        CALL TPUT('SEE INSTRUCTOR.',1);
        RETURN;
        END;
      TQ1=GENRAND(LIM,TQ1,TQ2,TQ3);
      TQ2=GENRAND(LIM,TQ1,TQ2,TQ3);
      TQ3=GENRAND(LIM,TQ1,TQ2,TQ3);
      CALL READ_STUD_LOCK(FOUND);
      IF (^FOUND) THEN DO;
         CALL REC_NO_FOUND;
         RETURN;
         END;
      IF (CASE_TBL(STUD.PSTATE,STUD.TSTATE)^='T')
         THEN DO;
         NULL_PROC=TRUE;
         UNLOCK FILE(STUDFIL) KEY(STUD.ID);
         RETURN;
         END;
      STUD.Q1=TQ1;
      STUD.Q2=TQ2;
      STUD.Q3=TQ3;
      STUD.TSTATE=TST_T;
      CALL UPDATE_STUD(STUD.ID,FOUND);
      IF (^FOUND) THEN CALL INTERNAL_ERR(24);
      PUT STRING(COUT) EDIT('TEST GENERATED ON UNIT:'
      ,STUD.UNIT+1,', AT TIME: ',SEPARATE(SUBSTR(
      TIME,1,6)),', QUESTIONS:',TQ1,',',TQ2,',',TQ3)
      (A ,F(4,0),A,A,A,3 (F(4,0),A(1)) );
      CALL TPUT(COUT,1);
      CALL STATE_MACH('T',CID);
   END;
END; /* GENTEST */

/*****************************************************/
/*                                                   */
/*    THIS PROCEDURE SCANS THE STUDENT FILE FOR      */
/*    STUDENTS WHO ARE AVAILABLE FOR PROCTORING      */
/*    THE CURRENTLY LOGGED IN STUDENTS AND SELECTS   */
/*    TWO OF THEM (IF POSSIBLE) TO PROCTOR THE       */
/*    STUDENT. THE PARAMETERS PID1,PID2 ARE USED     */
```

```
/*    TO RETURN THE STUDENT NUMBERS OF THE TWO    */
/*    PROCTORS AFTER THEY ARE FOUND. (IF STUDENTS */
/*    ARE NOT AVAILABLE THE INSTRUCTOR WILL BE    */
/*    SELECTED, AND IF MORE THAN I_TEST_LIM TEST  */
/*    CURRENTLY ASSIGNED TO THE INSTRUCTOR THEN   */
/*    THE RETURN ARGUMENT NONE IS SET TO TRUE. THE*/
/*    FOLLOWING ALGORITHM IS USED TO SELECT       */
/*    PROCTORS: THE ARRAY PINDX USED TO KEEP ALL  */
/*    THAT ARE ELIGIBLE TO PROCTOR THE CURRENT    */
/*    STUDENT. ( A STUDENT IS ELIGABLE IF HE HAS  */
/*    PASSED THE UNIT THE CURRENT STUDENT TEST IS */
/*    IS ON  AND IF HIS PROCTOR STATE IS PROCTOR  */
/*    AVAILABLE AND HE IS NOT WRITING A TEST OR   */
/*    STILL WAITING OUT HIS 10 MINUTES OF RESTUDY */
/*    TIME.) A STUDENT IS INSERTED INTO PINDX     */
/*    THROUGH A CALL TO INSRTP SUBROUTINE.IF LESS */
/*    THAN TWO STUDENTS ARE AVAILABLE THE INST-   */
/*    RUCTOR IS SELECTED. OTHERWISE THE STUDENTS  */
/*    WITH THE LOWEST NUMBER OF PROCTOR POINTS IS */
/*    SELECTED. ( THE NOT_EQ SUBROUTINE IS USED   */
/*    TO DETERMINE THE NUMBER OF STUDENTS WITH THE*/
/*    LOWEST PROCTOR POINTS.) IF MORE THAN ONE    */
/*    STUDENT HAS THE LOWEST PROCTOR POINTS (OR   */
/*    SECOND LOWEST) THEN THE GENRAND SUBROUTINE  */
/*    IS CALLED TO RANDOMLY SELECT FROM AMONG THEM*/
/*                                                */
/**************************************************/
  FIND_PROCTORS:PROC(PID1,PID2,NONE);
     DCL (NONE,AVAIL)              BIT(1);
     DCL (PID1,PID2)               CHAR(7);
     DCL (J,I,PEND,PEND1)          FIXED BIN(15);
     DCL 1 TEMP LIKE PINDX;
     DCL (NOW,STOP)                PICTURE '(12)9';
     PEND=0;
     ON ENDFILE(STUDFIL) GOTO EOF;
     CALL CLOSE_FILE(STUDFIL);
     NOW=DATE||SUBSTR(TIME,1,6);
     OPEN FILE(STUDFIL) INPUT SEQUENTIAL BUFFERED;
     DO WHILE('1'B);
        READ FILE(STUDFIL) INTO(TSTUD);
        IF (SYSP.SESSDATE<DATE) &
           ((TSTUD.RTIME>0)&(TSTUD.RTIME<=TWENTYM)) THEN
           STOP=(SYSP.SESSDATE+1)||TSTUD.RTIME;
        ELSE
           STOP=SYSP.SESSDATE||TSTUD.RTIME;
        IF ((TSTUD.PSTATE=PST_PA)&(TSTUD.TSTATE^=TST_T)
            &(TSTUD.TSTATE^=TST_OT) & (TSTUD.UNIT>
            STUD.UNIT)&(^((TSTUD.TSTATE=TST_R3)&
            (NOW<STOP)))) THEN
            CALL INSRTP;
     END;
EOF: CALL CLOSE_FILE(STUDFIL);
     OPEN FILE(STUDFIL) EXCL UPDATE DIRECT UNBUFFERED;
     NONE=FALSE;
     IF (PEND<2) THEN DO; /* SELECT INSTR.*/
        CALL READ_SYSP_LOCK;
        IF (SYSP.CUR_I_TEST>=I_TEST_LIM) THEN DO;
           NONE=TRUE;
           END;
        ELSE DO;
           PID1='INST';
           PID2=PID1;
           SYSP.CUR_I_TEST=SYSP.CUR_I_TEST+1;
           END;
        CALL UPDATE_SYSP;
        END;
     ELSE DO;
        PEND1=NOT_EQ(1,PEND);
        I=GENRAND(PEND1,0,0,0);
        PID1=PINDX(I).ID;
        TEMP = PINDX(1);
        PINDX(1)=PINDX(I);
        PINDX(I)=TEMP;
        PEND1=NOT_EQ(2,PEND);
        I=GENRAND(PEND1-1,0,0,0);
        PID2=PINDX(I+1).ID;
        END;
  /*LOGICAL END OF FIND PROCTORS. */

/**************************************************/
/*                                                */
/*    THIS IS A SUBROUTINE OF THE FIND_PROCTORS   */
/*    PROCEDURE WHICH TAKES THE PROCTOR NAME AND  */
/*    PROCTOR VALUE AND INSERTS THE RECORD TO THE */
/*    PINDX ARRAY IN SUCH AS THAT ALL THE         */
/*    IN PINDX ARE IN ASCENDING ORDER OF PROCTOR  */
/*    POINTS.                                     */
/*                                                */
/**************************************************/
  INSRTP:PROC;
     DCL 1 CARRY LIKE PINDX;
     DCL (K,L)                     FIXED BIN(15);
     DCL INSRTD                    BIT(1);
     DCL LBL                       LABEL;
     ON SUBSCRIPTRANGE BEGIN;
        DCL 1 TEMP(LSTUD) LIKE PINDX;
        DCL M FIXED BIN(15);
        TEMP=PINDX;
        LSTUD=LSTUD+LSTUDMORE;
        FREE PINDX;
        ALLOC PINDX;
        DO M=1 TO PEND;
           PINDX(M)=TEMP(M);
        END;
        GOTO LBL;
```

```
            END;
          INSRTD=FALSE;
          LBL=A;
          DO K=1 TO PEND WHILE(^INSRTD);
             IF (TSTUD.PROCTOR<PINDX(K).PROCTOR) THEN DO;
             A: DO L=PEND TO K BY -1;
(SUBSCRIPTRANGE): PINDX(L+1)=PINDX(L);
                END;
                PINDX(K)=TSTUD, BY NAME;
                INSRTD=TRUE;
             END;
          END;
          PEND=PEND+1;
          LBL=B;
(SUBRG): B: IF (^INSRTD) THEN PINDX(PEND)=TSTUD, BY NAME;
        END; /* INSRTP*/

      /*******************************************************/
      /*                                                     */
      /*    THIS IS A SUBROUTINE OF THE FIND_PROCTORS        */
      /*    PROCEDURE. IT IS USED TO SCAN THE PINDX          */
      /*    ARRAY FOR A CONTIGUOUS SET OF STUDENTS WHICH*/
      /*    THE SAME PROCTOR POINTS. SINCE THE PINDX         */
      /*    ARRAY BUILT IN SORTED ORDER THIS ROUTINE         */
      /*    ACCEPTS TWO ARGUMENTS: A SCAN START POSITION*/
      /*    AND END POSITION. THE INDEX OF THE LAST          */
      /*    STUDENT , IN THE RANGE OF THE SCAN, WHO HAS */
      /*    SAME AMOUNT OF PROCTOR POINTS AS THE STUDENT*/
      /*    INDEXED AT THE START LOCATION IS RETURNED TO*/
      /*    TO THE CALLING PROGRAM. THIS INFORMATION IS */
      /*    TO DETERMINE IF THERE ARE MORE THAN ONE          */
      /*    STUDENT WITH THE LOWEST (OR SECOND LOWEST)  */
      /*    PROCTOR POINTS.                                  */
      /*                                                     */
      /*******************************************************/
        NOT_EQ:PROC(STRT,END) RETURNS(FIXED BIN(15));
          DCL (STRT,END,MARKER)          FIXED BIN(15);
          DCL K                          FIXED BIN(15);
          DCL EQ                         BIT(1);
          EQ=TRUE;
          MARKER=END;
          DO K=STRT+1 TO END WHILE(EQ);
             IF (PINDX(STRT).PROCTOR^=PINDX(K).PROCTOR)
                THEN DO;
                EQ=FALSE;
                MARKER=K-1;
                END;
          END;
          RETURN(MARKER);
        END; /* NOT EQ */
      END; /* FIND PROCTORS */
```

```
      /*******************************************************/
      /*                                                     */
      /*   THIS PROCEDURE IS CALLED TO PROMPT PROCTORS       */
      /*   BEFORE THEY ENTER THEIR MARK TO ENTER THE         */
      /*   UNIT AND QUESTION NUMBERS WRITTEN BY THE          */
      /*   STUDENT BEING PROCTORED TO VERIFY THAT HE         */
      /*   WROTE THE SAME TEST WHICH WAS GIVEN TO HIM.       */
      /*   IF THE PROCTOR RESPONDS WITH A BARE RETURN TO*/
      /*   ANY OF THE PROMPTS THE RETURN ARGUMENT-           */
      /*   DONTKNOW IS SET TO TRUE. ALLOW ANOTHER            */
      /*   3 TRIALS OF ENTERING UNIT NUMBERS.                */
      /*                                                     */
      /*******************************************************/
    GET_QUES_NUM:PROC(DONTKNOW);
      DCL (BLANKF,NGOOD,EQ,CONT,DONTKNOW)    BIT(1);
      DCL QARRY(3)                           BIT(1);
      DCL (VAL,TGOOD,I,K,TRIAL)              FIXED BIN(15);
      DCL TRY                                FIXED BIN(15) INIT(3);
      DCL VALOUT                             PICTURE 'ZZZ9';
      DONTKNOW=FALSE;
      CONT=TRUE;
      DO I=0 TO TRY WHILE(CONT);
         CALL TPUT('ENTER UNIT NUMBER WRITTEN BY THE'||
                   ' STUDENT     : ',1);
         CALL GET_INT(VAL,NGOOD,BLANKF);
         IF (BLANKF) THEN
            CONT=FALSE;
         ELSE DO;
            IF (NGOOD) THEN
               CALL TPUT('INVALID NUMBER, RE-ENTER.',1);
            ELSE DO;
               IF ((STUD.UNIT+1)^=VAL) THEN DO;
                  VALOUT=VAL;
                  CALL TPUT('STUDENT DID NOT WRITE TEST'||
                    ' ON UNIT:'||VALOUT,1);
               END ;
               ELSE CONT=FALSE;
            END;
         END;
      END;
      QARRY=FALSE;
      CONT=TRUE;
      TGOOD=0;
      IF (BLANKF|(I>TRY)) THEN DO;
         DONTKNOW=TRUE;
         CONT=FALSE;
         END;
      TRIAL=0;
      DO WHILE(CONT&(TRIAL<=TRY));
         CALL TPUT('ENTER A QUESTION NUMBER WRITTEN BY THE'||
         ' STUDENT     : ',1);
         CALL GET_INT(VAL,NGOOD,BLANKF);
         IF (BLANKF) THEN
```

```
         CONT=FALSE;
      ELSE DO;
         IF (NGOOD) THEN
            CALL TPUT('INVALID NUMBER, RE-ENTER.',1);
         ELSE DO;
            TRIAL=TRIAL+1;
            K=0;
            SELECT(VAL);
            WHEN(ABS(STUD.Q1)) K=1;
            WHEN(ABS(STUD.Q2)) K=2;
            WHEN(ABS(STUD.Q3)) K=3;
            OTHERWISE K=0;
            END; /* SELECT */
            IF (K=0) THEN DO;
               VALOUT=VAL;
               CALL TPUT('STUDENT WAS NOT ISSUED '||
               'QUESTION NUMBER:'||VALOUT,1);
               END;
            ELSE DO;
               IF (QARRY(K)) THEN DO;
                 VALOUT=VAL;
                 CALL TPUT('QUESTION NUMBER:'||VALOUT||
                 '  HAS ALREADY BEEN VERIFIED.',1);
                 END;
               ELSE DO;
                  TRIAL=0;
                  TGOOD=TGOOD+1;
                  QARRY(K)=TRUE;
                  END;
               END;
            END;
         END;
         IF (TGOOD=3) THEN CONT=FALSE;
      END; /*WHILE*/
      IF (BLANKF|(TRIAL>TRY)) THEN DONTKNOW=TRUE;
   END; /* GET QUES NUM */
END; /* SESSION CONTROL */

/*******************************************************/
/*                                                     */
/*   THIS PROCEDURE IS USED TO ENTER A PROCTOR RE-     */
/*   SULT. THE INPUT ARGUMENT CID CONTAINS THE         */
/*   STUDENT NUMBER OF THE PROCTOR ENTERING THE        */
/*   RESULT. THE INPUT ARGUMENT BLANKF IS SET TO       */
/*   TRUE IF THE PROCTOR RESPONDED WITH A RETURN       */
/*   TO ONE OF THE PROMPTS IN THE GET_QUES_NUM         */
/*   PROCEDURE. IF BLANKF IS SET TO TRUE THE           */
/*   PROCTOR IS ONLY ALLOWED TO ENTER A RESTUDY        */
/*   RESULT (OR NO RESULT AT ALL).                     */
/*                                                     */
/*******************************************************/
 GET_MARK:PROC(BLANKF,CID,FOUND,PERMIT);

 DCL CID                      CHAR(7);
 DCL (BLANKF,NOPASS,FOUND)    BIT(1);
 DCL PERMIT(*)                BIT(1);
 DCL MMSG                     CHAR(43) INIT(
 '(P)ASS,(C)ONDITIONAL PASS,(R)ESTUDY?     : ');
 DCL PCR(3)                   CHAR(1) INIT('P','C','R');
 DCL (RESP,R1,R2,R3)          FIXED BIN(15);
 DCL OUTVAL                   PICTURE 'ZZZ9';
 DCL QMSG1                    CHAR(20) INIT(
 'WAS QUESTION NUMBER:');
 DCL QMSG2                    CHAR(42) INIT(
 ' COMPLETED SUCCESSFULLY (Y)ES,(N)O?     : ');
 NOPASS=BLANKF;
 CALL GET_RESP(MMSG,PCR,RESP,0);
 BLANKF=FALSE;
 CALL READ_STUD_LOCK(FOUND);
 IF (^FOUND) THEN RETURN;
 IF (^PERMIT(STUD.TSTATE)) THEN DO;
    CALL TPUT('STUDENT#: '||STUD.ID||
       'HAS JUST BEEN MARKED, NO RESULT ENTERED.',1);
    FOUND=FALSE;
    UNLOCK FILE(STUDFIL) KEY(STUD.ID);
    RETURN;
    END;
 STUD.Q1=ABS(STUD.Q1);
 STUD.Q2=ABS(STUD.Q2);
 STUD.Q3=ABS(STUD.Q3);
 CALL UPDATE_STUD(STUD.ID,FOUND);
 IF (^FOUND) THEN CALL INTERNAL_ERR(25);
 SELECT(RESP);
 WHEN(1) DO;
    IF (NOPASS) THEN DO;
       CALL TPUT('NOT ALLOWED TO PASS. NO RESULT '||
       ' ENTERED.',1);
       BLANKF=TRUE;
       END;
    ELSE DO;
       CALL STATE_MACH('J',CID); /* PASS */
       CALL TPUT('PASS RESULT ENTERED.',1);
       END;
    END; /*WHEN*/
 WHEN(2) DO;
    IF (NOPASS) THEN DO;
       CALL TPUT('NOT ALLOWED TO PASS. NO RESULT '||
       'ENTERED.',1);
       BLANKF=TRUE;
       END;
    ELSE DO;
       CALL STATE_MACH('K',CID); /*CONDITIONAL PASS*/
       CALL TPUT('CONDITIONAL PASS ENTERED.',1);
       END;
    END; /*WHEN*/
 WHEN(3) DO; /* RESTUDY */
```

```
      OUTVAL = STUD.Q1;
      CALL GET_RESP(QMSG1||OUTVAL||QMSG2,YN,R1,1);
      OUTVAL=STUD.Q2;
      CALL GET_RESP(QMSG1||OUTVAL||QMSG2,YN,R2,1);
      OUTVAL=STUD.Q3;
      CALL GET_RESP(QMSG1||OUTVAL||QMSG2,YN,R3,1);
      CALL READ_STUD_LOCK(FOUND);
      IF (^FOUND) THEN RETURN;
      IF (^PERMIT(STUD.TSTATE)) THEN DO;
         CALL TPUT('STUDENT#: '||STUD.ID||
            'HAS JUST BEEN MARKED, NO RESULT ENTERED.',1);
         FOUND=FALSE;
         UNLOCK FILE(STUDFIL) KEY(STUD.ID);
         RETURN;
         END;
      IF (R1=2) THEN
         STUD.Q1=-STUD.Q1;
      IF (R2=2) THEN
          STUD.Q2=-STUD.Q2;
      IF (R3=2) THEN
         STUD.Q3=-STUD.Q3;
      CALL UPDATE_STUD(STUD.ID,FOUND);
      IF (^FOUND) THEN CALL INTERNAL_ERR(25);
      CALL STATE_MACH('L',CID);
      END;
   WHEN(0) DO;
      BLANKF=TRUE;
      END;
   OTHERWISE DO;
      BLANKF=TRUE;
      CALL INTERNAL_ERR(9);
      END;
   END;
END; /* GET MARK */

/*******************************************************/
/*                                                     */
/*    THIS PROCEDURE IS THE STATE MACHINE FOR THE */
/*    THE STUDENT TRANSACTIONS. A TRANSACTION TYPE*/
/*    IS PASSED IN ARGUMENT TRANS AND IT IS USED  */
/*    ALONG WITH THE STUDENTS CURRENT STATE TO    */
/*    GIVE THE STUDENT HIS NEW STATE. THE         */
/*    TRANSACTION IS THEN LOGGED BY A CALL TO THE */
/*    LOG_TRANS ROUTINE.                          */
/*                                                     */
/*******************************************************/
STATE_MACH:PROC(TRANS,CID);
   DCL CID                          CHAR(7);
   DCL TRANS                        CHAR(1);
   DCL (FOUND,STUD_PROC)             BIT(1);
   CALL READ_SYSP;
   CALL READ_STUD_LOCK(FOUND);
   IF (^FOUND) THEN RETURN;
   SELECT(TRANS);
   WHEN('T') DO; /* GENERATE TEST */
      STUD.TSTATE=TST_T;
      STUD.RTIME=SUBSTR(TIME,1,6);
      CALL UPDATE_STUD(STUD.ID,FOUND);
      IF (^FOUND) THEN CALL INTERNAL_ERR(20);
      CALL LOG(TRANS,CID);
      END;
   WHEN('C')DO; /* CANCEL TEST */
      STUD.TSTATE=TST_R3;
      STUD.RTIME=ADDTIME((SUBSTR(TIME,1,6)),TWENTYM);
      CALL UPDATE_STUD(STUD.ID,FOUND);
      IF (^FOUND) THEN CALL INTERNAL_ERR(20);
      CALL LOG(TRANS,CID);
      END;
   WHEN('I')DO; /* MARK TEST */
      STUD.TSTATE=TST_R0;
      CALL UPDATE_STUD(STUD.ID,FOUND);
      IF (^FOUND) THEN CALL INTERNAL_ERR(20);
      CALL LOG(TRANS,CID);
      END;
   WHEN('P')DO; /* PROCTOR SELECT */
      STUD.PSTATE=PST_P;
      STUD.SID=CID;
      STUD.RTIME=SUBSTR(TIME,1,6);
      CALL UPDATE_STUD(STUD.ID,FOUND);
      IF (^FOUND) THEN CALL INTERNAL_ERR(20);
      CALL LOG(TRANS,CID);
      END;
   WHEN('M')DO;
      STUD.PSTATE=PST_PA;
      STUD.PROCTOR=STUD.PROCTOR+SYSP.VPROC;
      CALL TOTAL_MARKS;
      CALL UPDATE_STUD(STUD.ID,FOUND);
      IF (^FOUND) THEN CALL INTERNAL_ERR(20);
      END;
   WHEN('N')DO;
      STUD.PSTATE=PST_PA;
      CALL UPDATE_STUD(STUD.ID,FOUND);
      IF (^FOUND) THEN CALL INTERNAL_ERR(20);
      END;
   WHEN('J','K')DO; /* PASS RESULT */
      DCL PPTBL(L_TST)       FIXED BIN(15) INIT(
      TST_I,TST_I,TST_I,TST_R1,TST_NT,TST_R3,TST_R3,
      TST_OT,TST_OM,TST_RC);
      IF ((CID='INST')|(CID='TA')) THEN DO;
         IF (STUD.TSTATE^=TST_T) THEN DO;
            SAVE=STUD;
            CALL STUD_PROCTORS(SAVE.ID,TRUE,FALSE,
               STUD_PROC);
            STUD=SAVE;
            IF ^STUD_PROC & (SYSP.CUR_I_TEST>0) THEN DO;
```

```
            CALL READ_SYSP_LOCK;
            SYSP.CUR_I_TEST=SYSP.CUR_I_TEST-1;
            CALL UPDATE_SYSP;
            END;
         END;
      STUD.TSTATE=TST_NT;
      END;
   ELSE DO;
      STUD.TSTATE=PPTBL(STUD.TSTATE);
      END;
   IF (STUD.TSTATE=TST_R3) THEN DO;
      STUD.RTIME=ADDTIME((SUBSTR(TIME,1,6)),
       TWENTYM); END;
   IF (STUD.TSTATE=TST_NT) THEN DO;
      STUD.TEST=STUD.TEST+SYSP.VPASS;
      STUD.UNIT=STUD.UNIT+1;
      CALL TOTAL_MARKS;
      END;
   CALL UPDATE_STUD(STUD.ID,FOUND);
   IF (^FOUND) THEN CALL INTERNAL_ERR(20);
   CALL LOG(TRANS,CID);
   END;
WHEN('L')DO; /* RESTUDY RESULT */
   DCL SPTBL(L_TST)   FIXED BIN(15) INIT(
   TST_I,TST_T,TST_NT,TST_R2,TST_R3,TST_R3,TST_R3,
   TST_OT,TST_OM,TST_RC);
   IF ((CID='INST')|(CID='TA')) THEN DO;
      IF (STUD.TSTATE^=TST_T) THEN DO;
         SAVE=STUD;
         CALL STUD_PROCTORS(SAVE.ID,TRUE,FALSE,
            STUD_PROC);
         STUD=SAVE;
         IF ^STUD_PROC & (SYSP.CUR_I_TEST>0) THEN DO;
            CALL READ_SYSP_LOCK;
            SYSP.CUR_I_TEST=SYSP.CUR_I_TEST-1;
            CALL UPDATE_SYSP;
            END;
         END;
      STUD.TSTATE=TST_R3;
      END;
   ELSE DO;
      STUD.TSTATE=SPTBL(STUD.TSTATE);
      END;
   IF (STUD.TSTATE=TST_R3) THEN DO;
      STUD.RTIME=ADDTIME((SUBSTR(TIME,1,6)),
       TWENTYM); END;
   CALL UPDATE_STUD(STUD.ID,FOUND);
   IF (^FOUND) THEN CALL INTERNAL_ERR(20);
   CALL LOG(TRANS,CID);
   END;
OTHERWISE DO;
   UNLOCK FILE(STUDFIL) KEY(STUD.ID);
   CALL INTERNAL_ERR(21);
      END;
   END;
  END; /* STATE_MACH */
  /*****************************************************/
  /*                                                   */
  /*       E N D    O F    P R O G R A M               */
  /*                                                   */
  /*****************************************************/
 END; /* TERM CONTROL */
END; /* MAIN PROGRAM */
//LKED.SYSLIB DD DSN=SYS2.PLI.OPT.PLITASK,DISP=SHR
//            DD DSN=SYS1.PLIBASE,DISP=SHR
//LKED.SYSLMOD DD DSN=*** location of the product ***,DISP=OLD
```

APPENDIX C

EXAMPLES OF CAPSI

IN CLASSROOM SETTING

```
C: lo g=.jpear.psi-r            -- Start PSI from mantes
C: execute psi

PERSONALIZED SYSTEM INSTRUCTION.
VERSION 2.0 : OCT 15,1985
86/06/04    14:32:53
STARTING INITIALIZATION NOW.

1.START SESSION.   2.END SESSION.
3.EDIT STUDENT.    4.EDIT SYSTEM PARAMETERS.
5.MARK TEST.       6.SEND MESSAGES.
7.MONITOR STUDENT. 8.EXIT TO TSO.
ENTER CHOICE==>    : 3.5        -- Position directly to list student
ENTER EDIT PASSWORD : <cr>

STARTING AT: 14:33:03
ENTER STUDENT NUMBER     : all
STUD.#  NAME                          PHONE   FAC YR STAT UNIT TEST PROCTOR    TERM    EXAM   TOTAL GRADE
1       STUDENT NAME 1                2699620 01 86 BSC  0    0.000   0.000   0.000   0.000   0.000
2       STUDENT NAME 2                4771111 07 85 XXX  0    0.000   0.000   0.000   0.000   0.000
3       STUDENT NAME 3                0002222 02 85 ###  0    0.000   0.000   0.000   0.000   0.000
4       NAME 4                        1234567 10 80 &&&  0    0.000   0.000   0.000   0.000   0.000
5       STUDENT 5                     0000000 00 00 000  0    0.000   0.000   0.000   0.000   0.000
6       LONG NAME..................   8888888 88 88 888  0    0.000   0.000   0.000   0.000   0.000
7                                                        0    0.000   0.000   0.000   0.000   0.000
8                                                        0    0.000   0.000   0.000   0.000   0.000
9                                                        0    0.000   0.000   0.000   0.000   0.000
TOTAL# OF STUDENTS:     9       -- The total number of students
ENTER STUDENT NUMBER     : <cr>
ENDING AT: 14:33:26

<3.>1 CREATE STUDENT.        <3.>2 DELETE STUDENT.
<3.>3 MODIFY PERSONAL DATA.  <3.>4 MODIFY COURSE DATA.
<3.>5 LIST STUDENT.          <3.>6 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 3
```

```
ENTER STUDENT NUMBER     : edit  -- Change EDIT password
<3.3.>1 NAME.               <3.3.>2 STUDENT NUMBER.
<3.3.>3 PHONE NUMBER.       <3.3.>4 FACULTY CODE.
<3.3.>5 YEAR AT UNIVERSITY. <3.3.>6 STATUS IN COURSE.
<3.3.>7 PASSWORD.
ENTER CHOICE==>    : 7
ENTER INSTRUCTOR PASSWORD     : <cr>   -- Need INST password to change EDIT password
                                -- Current  INST password was blank
STARTING AT: 14:33:53
CURRENT VALUE OF
PASSWORD: "         "
ENTER NEW PASSWORD    : editpass  -- New value of EDIT password
PASSWORD: "EDITPASS"

ENTER STUDENT NUMBER     : inst  -- Change INST password
<3.3.>1 NAME.               <3.3.>2 STUDENT NUMBER.
<3.3.>3 PHONE NUMBER.       <3.3.>4 FACULTY CODE.
<3.3.>5 YEAR AT UNIVERSITY. <3.3.>6 STATUS IN COURSE.
<3.3.>7 PASSWORD.
ENTER CHOICE==>    : 7
ENTER INSTRUCTOR PASSWORD     : <cr>   -- Need INST password to change INST password

STARTING AT: 14:34:23
CURRENT VALUE OF
PASSWORD: "         "
ENTER NEW PASSWORD    : INSTpass  -- Upper or lower case can be used
PASSWORD: "INSTPASS"

ENTER STUDENT NUMBER     : ta
<3.3.>1 NAME.               <3.3.>2 STUDENT NUMBER.
<3.3.>3 PHONE NUMBER.       <3.3.>4 FACULTY CODE.
<3.3.>5 YEAR AT UNIVERSITY. <3.3.>6 STATUS IN COURSE.
<3.3.>7 PASSWORD.
ENTER CHOICE==>    : 7

CURRENT VALUE OF                -- No INST password is needed to change TA/student password
PASSWORD: "         "
ENTER NEW PASSWORD    : tapass
PASSWORD: "TAPASS   "
```

```
ENTER STUDENT NUMBER     : <cr>
ENDING AT: 14:35:04

<3.>1 CREATE STUDENT.        <3.>2 DELETE STUDENT.
<3.>3 MODIFY PERSONAL DATA. <3.>4 MODIFY COURSE DATA.
<3.>5 LIST STUDENT.          <3.>6 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 4

<3.4.>1 CURRENT UNIT.    <3.4.>2 TEST POINTS.
<3.4.>3 PROCTOR POINTS. <3.4.>4 TERM POINTS.
<3.4.>5 FINAL EXAM.      <3.4.>6 RETURN TO EDIT MENU.
ENTER CHOICE==>    : 5            -- Change a student's exam points

ENTER NEXT STUDENT NUMBER     : 2   -- Give a student with a new exam points
CURRENT VALUE OF EXAM POINTS:    0.000
ENTER NEW VALUE OF EXAM POINTS     : 75.0

ENTER NEXT STUDENT NUMBER     : 5   -- Give another student with a new exam points
CURRENT VALUE OF EXAM POINTS:    0.000
ENTER NEW VALUE OF EXAM POINTS     : 69.34
ENTER NEXT STUDENT NUMBER     : <cr>   -- Return to the edit course menu

<3.4.>1 CURRENT UNIT.    <3.4.>2 TEST POINTS.
<3.4.>3 PROCTOR POINTS. <3.4.>4 TERM POINTS.
<3.4.>5 FINAL EXAM.      <3.4.>6 RETURN TO EDIT MENU.
ENTER CHOICE==>    : <cr>
ENDING AT: 14:36:11

<3.>1 CREATE STUDENT.        <3.>2 DELETE STUDENT.
<3.>3 MODIFY PERSONAL DATA. <3.>4 MODIFY COURSE DATA.
<3.>5 LIST STUDENT.          <3.>6 RETURN TO MAIN MENU.
ENTER CHOICE==>    : <cr>

1.START SESSION.    2.END SESSION.
3.EDIT STUDENT.     4.EDIT SYSTEM PARAMETERS.
5.MARK TEST.        6.SEND MESSAGES.
7.MONITOR STUDENT. 8.EXIT TO TSO.
ENTER CHOICE==>    : 4
ENTER EDIT PASSWORD : EDITpass
```

```
STARTING AT: 14:36:22
<4.>1 VEIW SYSTEM PARAMETERS.         <4.>2 ENTER TOTAL NUMBER OF UNITS.
<4.>3 ENTER LETTER GRADE THRESHOLDS. <4.>4 ENTER VALUE OF UNIT PASSING.
<4.>5 ENTER VALUE OF PROCTORING.      <4.>6 ENTER QUESTION NUMBER LIMITS.
<4.>7 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 1

TOTAL NUMBER OF UNITS:  0, VALUE OF UNIT PASSING:   0.000, VALUE OF PROCTORING:  0.000
LETTER GRADE THRESHOLDS:
A+     0.000     A      0.000     A-     0.000     B+     0.000     B      0.000
B-     0.000     C+     0.000     C      0.000     C-     0.000     D+     0.000
D      0.000     D-     0.000     F      0.000
UNIT QUESTION NUMBER LIMITS:      -- Current total unit number is 0
ENDING AT: 14:36:25

<4.>1 VEIW SYSTEM PARAMETERS.         <4.>2 ENTER TOTAL NUMBER OF UNITS.
<4.>3 ENTER LETTER GRADE THRESHOLDS. <4.>4 ENTER VALUE OF UNIT PASSING.
<4.>5 ENTER VALUE OF PROCTORING.      <4.>6 ENTER QUESTION NUMBER LIMITS.
<4.>7 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 2              -- Change total unit number

CURRENT VALUE OF TOTAL NUMBER OF UNITS IS:   0
ENTER NEW VALUE OF TOTAL NUMBER OF UNITS     : 10
ENDING AT: 14:36:36

<4.>1 VEIW SYSTEM PARAMETERS.         <4.>2 ENTER TOTAL NUMBER OF UNITS.
<4.>3 ENTER LETTER GRADE THRESHOLDS. <4.>4 ENTER VALUE OF UNIT PASSING.
<4.>5 ENTER VALUE OF PROCTORING.      <4.>6 ENTER QUESTION NUMBER LIMITS.
<4.>7 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 6              -- Assign question limit to an unit

ENTER UNIT NUMBER      : 1
CURRENT VALUE OF UNIT QUESTION LIMIT FOR UNIT(  1) IS:   3
ENTER NEW VALUE OF UNIT QUESTION LIMIT FOR UNIT(  1)     : 5

ENTER UNIT NUMBER      : 2
CURRENT VALUE OF UNIT QUESTION LIMIT FOR UNIT(  2) IS:   3
ENTER NEW VALUE OF UNIT QUESTION LIMIT FOR UNIT(  2)     : 5
```

```
ENTER UNIT NUMBER      : 3
CURRENT VALUE OF UNIT QUESTION LIMIT FOR UNIT(  3) IS:   3
ENTER NEW VALUE OF UNIT QUESTION LIMIT FOR UNIT(  3)     : 5

ENTER UNIT NUMBER      : <cr>
ENDING AT: 14:36:54

<4.>1 VEIW SYSTEM PARAMETERS.         <4.>2 ENTER TOTAL NUMBER OF UNITS.
<4.>3 ENTER LETTER GRADE THRESHOLDS. <4.>4 ENTER VALUE OF UNIT PASSING.
<4.>5 ENTER VALUE OF PROCTORING.      <4.>6 ENTER QUESTION NUMBER LIMITS.
<4.>7 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 3              -- Change letter grade thresholds

<4.3.>1  A+ <4.3.>2  A  <4.3.>3  A-
<4.3.>4  B+ <4.3.>5  B  <4.3.>6  B-
<4.3.>7  C+ <4.3.>8  C  <4.3.>9  C-
<4.3.>10 D+ <4.3.>11 D  <4.3.>12 D- <4.3.>13 F

ENTER LETTER CHOICE(1-13)==>    : 1
CURRENT VALUE OF A+ THRESHOLD.:    0.000
ENTER NEW VALUE OF A+ THRESHOLD.      : 100

ENTER LETTER CHOICE(1-13)==>    : 2
CURRENT VALUE OF A  THRESHOLD.:    0.000
ENTER NEW VALUE OF A  THRESHOLD.      : 95

ENTER LETTER CHOICE(1-13)==>    : 3
CURRENT VALUE OF A- THRESHOLD.:    0.000
ENTER NEW VALUE OF A- THRESHOLD.      : 90

ENTER LETTER CHOICE(1-13)==>    : <cr>
ENDING AT: 14:37:59

<4.>1 VEIW SYSTEM PARAMETERS.         <4.>2 ENTER TOTAL NUMBER OF UNITS.
<4.>3 ENTER LETTER GRADE THRESHOLDS. <4.>4 ENTER VALUE OF UNIT PASSING.
<4.>5 ENTER VALUE OF PROCTORING.      <4.>6 ENTER QUESTION NUMBER LIMITS.
<4.>7 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 1
```

```
TOTAL NUMBER OF UNITS: 10, VALUE OF UNIT PASSING:   0.000, VALUE OF PROCTORING:  0.000
LETTER GRADE THRESHOLDS:
A+  100.000    A    95.000    A-   90.000    B+    0.000    B     0.000
B-    0.000    C+    0.000    C     0.000    C-    0.000    D+    0.000
D     0.000    D-    0.000    F     0.000
UNIT QUESTION NUMBER LIMITS:
  1.  5       2.  5       3.  5       4.  3       5.  3
  6.  3       7.  3       8.  3       9.  3      10.  3
ENDING AT: 14:38:02

<4.>1 VEIW SYSTEM PARAMETERS.         <4.>2 ENTER TOTAL NUMBER OF UNITS.
<4.>3 ENTER LETTER GRADE THRESHOLDS. <4.>4 ENTER VALUE OF UNIT PASSING.
<4.>5 ENTER VALUE OF PROCTORING.      <4.>6 ENTER QUESTION NUMBER LIMITS.
<4.>7 RETURN TO MAIN MENU.
ENTER CHOICE==>    : <cr>

1.START SESSION.    2.END SESSION.
3.EDIT STUDENT.     4.EDIT SYSTEM PARAMETERS.
5.MARK TEST.        6.SEND MESSAGES.
7.MONITOR STUDENT. 8.EXIT TO TSO.
ENTER CHOICE==>    : 1              -- Starting a new session

A NEW SESSION HAS NOT BEEN STARTED.
CURRENT TIME IS: 14:38:08
CURRENT CUT-OFF TIME IS: 16:38:00
DO YOU WISH TO ALTER CUT-OFF TIME? (Y)ES,(N)O      : y-- Default is "No"
ENTER NEW CUT-OFF (HH:MM:SS) TIME      : 18:00:00
CUT-OFF DATE 86/06/04 AND TIME 18:00:00 ARE ASSIGNED TO SYSTEM.
(C)ONTINUE LAST SESSION OR START A (N)EW ONE?      : n-- Default is "Continue"
ALL STUDENTS SET TO INITIAL STATE.
TEST ISSUE CUT-OFF TIME WILL BE: 18:00:00
SWITCH TERMINAL TO FULL DUPLEX NOW.

ENTER STUDENT NUMBER      : 1     -- It is ready to handle student transactions
ENTER PASSWORD      : wrongpass
INVALID
PASSWORD: "WRONGPAS"
ENTER PASSWORD      : <cr>          -- Re-enter password, the current password is blank
```

```
STARTING AT: 14:38:53
WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O      : y -- Default is "No"
CURRENT UNIT:  0, TEST POINTS:   0.00, PROCTOR POINTS:   0.00, TERM PROJECT:   0.00, FINAL EXAM:   0.00
TOTAL POINTS:   0.00, LETTER GRADE: B+
PROCTOR? (Y)ES,(N)O      : y
STUDENT#: 1        IS AVAILABLE FOR PROCTORING.
GENERATE TEST ON UNIT    1 (Y)ES,(N)O?      : y -- Default is "No"
TEST GENERATED ON UNIT:   1, AT TIME: 14:39:02, QUESTIONS:   2,   5,   1
OK   TRANSACTION COMPLETE.
ENDING AT: 14:39:02

ENTER STUDENT NUMBER      : 1
ENTER PASSWORD      : <cr>

STARTING AT: 14:39:15
PROCTOR? (Y)ES,(N)O      : <cr>   -- Default is nothing
STUDENT#: 1        IS AVAILABLE FOR PROCTORING.
DO YOU WANT YOUR TEST (C)ANCELLED OR (P)ROCTORED?      : p  -- Default is nothing
PROCTOR SELECTED IS INSTRUCTOR OR TA.
OK   TRANSACTION COMPLETE.
ENDING AT: 14:39:26

ENTER STUDENT NUMBER      : 8
ENTER PASSWORD      : <cr>         -- Current password is blank

STARTING AT: 14:39:34
WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O      : y
CURRENT UNIT:  0, TEST POINTS:   0.00, PROCTOR POINTS:   0.00, TERM PROJECT:   0.00, FINAL EXAM:   0.00
TOTAL POINTS:   0.00, LETTER GRADE: B+
PROCTOR? (Y)ES,(N)O      : y
STUDENT#: 8        IS AVAILABLE FOR PROCTORING.
GENERATE TEST ON UNIT    1 (Y)ES,(N)O?      : y
TEST GENERATED ON UNIT:   1, AT TIME: 14:39:41, QUESTIONS:   2,   4,   5
OK   TRANSACTION COMPLETE.
ENDING AT: 14:39:42

ENTER STUDENT NUMBER      : marker   -- Marking without going out of this session
ENTER PASSWORD      : instpass    -- TA or INST password can be used
```

```
STARTING AT: 14:39:57
<5.>1 LIST TEST AND STATUS. <5.>2 LIST TESTS FOR MARKER.
<5.>3 MARK STUDENT.         <5.>4 RETURN FROM MARKING.
ENTER CHOICE==>    : 1

ENTER STUDENT NUMBER      : all    -- List all who are writing a test or proctor(s) assigned
LISTING OF ALL THE UNMARKED STUDENTS.
STUD.#  NAME                          UNIT# QUESTIONS TEST STATE    PROCTOR STATE   TIME
1       STUDENT NAME 1                  1   2   5   1 NO MARK YET AVAILABLE       14:39:02
PROCTOR SELECTED IS INSTRUCTOR OR TA.
8                                       1   2   4   5 WRITING     AVAILABLE       14:39:41
STUDENT IS WRITING A TEST.

ENTER STUDENT NUMBER      : 5
STUD.#  NAME                          UNIT# QUESTIONS TEST STATE    PROCTOR STATE   TIME
5       STUDENT 5                       0   0   0   0 INITIAL     INITIAL         00:00:00
STUDENT HAS NO UNMARKED TEST.

ENTER STUDENT NUMBER      : 0
0       IS A SPECIAL OR NON-EXISTANT STUDENT NUMBER. DATA CANNOT BE LISTED.

ENTER STUDENT NUMBER      : <cr>
ENDING AT: 14:40:28

<5.>1 LIST TEST AND STATUS. <5.>2 LIST TESTS FOR MARKER.
<5.>3 MARK STUDENT.         <5.>4 RETURN FROM MARKING.
ENTER CHOICE==>    : 2           -- List all who are writing a test or TA/INST is proctor

INSTRUCTOR OR TA IS SELECTED.
STUD.#  NAME                          UNIT# QUESTIONS TEST STATE    PROCTOR STATE   TIME
1       STUDENT NAME 1                  1   2   5   1 NO MARK YET AVAILABLE       14:39:02
8                                       1   2   4   5 WRITING     AVAILABLE       14:39:41
ENDING AT: 14:40:36

<5.>1 LIST TEST AND STATUS. <5.>2 LIST TESTS FOR MARKER.
<5.>3 MARK STUDENT.         <5.>4 RETURN FROM MARKING.
ENTER CHOICE==>    : 3           -- Mark student
```

```
ENTER STUDENT NUMBER TO BE MARKED      : 1
STUDENT#: 1       , CURRENT TEST QUESTIONS:      2     5      1, ON UNIT:    1
(P)ASS,(C)ONDITIONAL PASS,(R)ESTUDY?      : p -- Default is nothing
PASS RESULT ENTERED.

ENTER STUDENT NUMBER TO BE MARKED      : <cr>
ENDING AT: 14:40:54

<5.>1 LIST TEST AND STATUS. <5.>2 LIST TESTS FOR MARKER.
<5.>3 MARK STUDENT.         <5.>4 RETURN FROM MARKING.
ENTER CHOICE==>    : <cr>

ENTER STUDENT NUMBER      : ta     -- Get out of this session, TA/INST can be used
ENTER PASSWORD      : tapass

STARTING AT: 14:42:14
1.START SESSION.   2.END SESSION.
3.EDIT STUDENT.    4.EDIT SYSTEM PARAMETERS.
5.MARK TEST.       6.SEND MESSAGES.
7.MONITOR STUDENT. 8.EXIT TO TSO.
ENTER CHOICE==>    : 2             -- End current session
ENTER INSTRUCTOR PASSWORD      : instpass

STARTING AT: 14:42:24
STUDENTS WHICH HAVE UNMARKED TESTS.
STUDENT:                    , STUDENT#: 8       , UNIT#:    1, QUESTIONS:     2,    4,    5, AT TIME: 14:39:41

1.START SESSION.   2.END SESSION.
3.EDIT STUDENT.    4.EDIT SYSTEM PARAMETERS.
5.MARK TEST.       6.SEND MESSAGES.
7.MONITOR STUDENT. 8.EXIT TO TSO.
ENTER CHOICE==>    : 3
ENTER EDIT PASSWORD : editpass

STARTING AT: 14:42:39
<3.>1 CREATE STUDENT.       <3.>2 DELETE STUDENT.
<3.>3 MODIFY PERSONAL DATA. <3.>4 MODIFY COURSE DATA.
<3.>5 LIST STUDENT.         <3.>6 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 1
```

```
ENTER NEW STUDENT NUMBER      : 0   -- Create a new student
STUDENT NAME(UP TO 30 CHARACTERS): student # 0
PHONE NUMBER(7 CHARACTERS)     : 7777777
FACULTY CODE (2 CHARACTERS)     : 03
YEAR AT UNIVERSITY(2 CHARACTERS)     : 84
STATUS IN COURSE(3 CHARACTERS)     : xxx
STUDENT PASSWORD     : Opass
PASSWORD: "OPASS    "
ENTER NEW STUDENT NUMBER      : <cr>
ENDING AT: 14:43:19

<3.>1 CREATE STUDENT.       <3.>2 DELETE STUDENT.
<3.>3 MODIFY PERSONAL DATA. <3.>4 MODIFY COURSE DATA.
<3.>5 LIST STUDENT.         <3.>6 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 2

ENTER STUDENT NUMBER     : 9
DO YOU WANT THIS STUDENT#: 9       TO BE (D)ELETED OR (M)AINTAINED?     : d  -- Default is "Maintained"
STUDENT#: 9       WAS DELETED FROM FILES.
ENDING AT: 14:43:31

<3.>1 CREATE STUDENT.       <3.>2 DELETE STUDENT.
<3.>3 MODIFY PERSONAL DATA. <3.>4 MODIFY COURSE DATA.
<3.>5 LIST STUDENT.         <3.>6 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 3

ENTER STUDENT NUMBER     : 1     -- Make changes on this student
<3.3.>1 NAME.               <3.3.>2 STUDENT NUMBER.
<3.3.>3 PHONE NUMBER.       <3.3.>4 FACULTY CODE.
<3.3.>5 YEAR AT UNIVERSITY. <3.3.>6 STATUS IN COURSE.
<3.3.>7 PASSWORD.
ENTER CHOICE==>    : 1            -- Change student name

CURRENT VALUE OF NAME FIELD: STUDENT NAME 1
ENTER NEW NAME      : new name 1
```

```
ENTER STUDENT NUMBER     : 1
<3.3.>1 NAME.                <3.3.>2 STUDENT NUMBER.
<3.3.>3 PHONE NUMBER.        <3.3.>4 FACULTY CODE.
<3.3.>5 YEAR AT UNIVERSITY. <3.3.>6 STATUS IN COURSE.
<3.3.>7 PASSWORD.
ENTER CHOICE==>    : 7            -- Change password

CURRENT VALUE OF
PASSWORD: "         "
ENTER NEW PASSWORD    : 1pass
PASSWORD: "1PASS    "

ENTER STUDENT NUMBER     : <cr>
ENDING AT: 14:44:41

<3.>1 CREATE STUDENT.        <3.>2 DELETE STUDENT.
<3.>3 MODIFY PERSONAL DATA. <3.>4 MODIFY COURSE DATA.
<3.>5 LIST STUDENT.          <3.>6 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 4

<3.4.>1 CURRENT UNIT.    <3.4.>2 TEST POINTS.
<3.4.>3 PROCTOR POINTS. <3.4.>4 TERM POINTS.
<3.4.>5 FINAL EXAM.      <3.4.>6 RETURN TO EDIT MENU.
ENTER CHOICE==>    : 5            -- Change final exam points

ENTER NEXT STUDENT NUMBER     : 2
CURRENT VALUE OF EXAM POINTS:    0.000
ENTER NEW VALUE OF EXAM POINTS     : 10

ENTER NEXT STUDENT NUMBER     : 7
CURRENT VALUE OF EXAM POINTS:    0.000
ENTER NEW VALUE OF EXAM POINTS     : 5
ENTER NEXT STUDENT NUMBER     : <cr>

<3.4.>1 CURRENT UNIT.    <3.4.>2 TEST POINTS.
<3.4.>3 PROCTOR POINTS. <3.4.>4 TERM POINTS.
<3.4.>5 FINAL EXAM.      <3.4.>6 RETURN TO EDIT MENU.
ENTER CHOICE==>    : <cr>
ENDING AT: 14:45:29
```

```
<3.>1 CREATE STUDENT.        <3.>2 DELETE STUDENT.
<3.>3 MODIFY PERSONAL DATA. <3.>4 MODIFY COURSE DATA.
<3.>5 LIST STUDENT.          <3.>6 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 5

ENTER STUDENT NUMBER      : ALL
STUD.#  NAME                          PHONE  FAC YR STAT UNIT TEST PROCTOR    TERM    EXAM   TOTAL GRADE
0       STUDENT # 0                   7777777 03 84 XXX  0    0.000   0.000   0.000   0.000   0.000
1       NEW NAME 1                    2699620 01 86 BSC  1    0.000   0.000   0.000   0.000   0.000   B+
2       STUDENT NAME 2                4771111 07 85 XXX  0    0.000  75.000   0.000  10.000  85.000   B+
<break>
(C)ONTINUE OR (Q)UIT   : q        -- After hitting a break key, default is "Continue"
TOTAL# OF STUDENTS:     9
ENTER STUDENT NUMBER      : <cr>
ENDING AT: 14:45:50

<3.>1 CREATE STUDENT.        <3.>2 DELETE STUDENT.
<3.>3 MODIFY PERSONAL DATA. <3.>4 MODIFY COURSE DATA.
<3.>5 LIST STUDENT.          <3.>6 RETURN TO MAIN MENU.
ENTER CHOICE==>    : <cr>

1.START SESSION.    2.END SESSION.
3.EDIT STUDENT.     4.EDIT SYSTEM PARAMETERS.
5.MARK TEST.        6.SEND MESSAGES.
7.MONITOR STUDENT. 8.EXIT TO TSO.
ENTER CHOICE==>    : 5            -- Same as MARKER logging in during a session
ENTER INSTRUCTOR PASSWORD      : instpass

STARTING AT: 14:45:55
<5.>1 LIST TEST AND STATUS. <5.>2 LIST TESTS FOR MARKER.
<5.>3 MARK STUDENT.          <5.>4 RETURN FROM MARKING.
ENTER CHOICE==>    : 2

INSTRUCTOR OR TA IS SELECTED.
STUD.#  NAME                          UNIT# QUESTIONS TEST STATE    PROCTOR STATE   TIME
                                        1   2   4   5 WRITING     INITIAL         14:39:41
ENDING AT: 14:46:09
```

```
<5.>1 LIST TEST AND STATUS. <5.>2 LIST TESTS FOR MARKER.
<5.>3 MARK STUDENT.         <5.>4 RETURN FROM MARKING.
ENTER CHOICE==>    : 3

ENTER STUDENT NUMBER TO BE MARKED     : 8
STUDENT#: 8       , CURRENT TEST QUESTIONS:     2     4     5, ON UNIT:    1
(P)ASS,(C)ONDITIONAL PASS,(R)ESTUDY?     : c
CONDITIONAL PASS ENTERED.

ENTER STUDENT NUMBER TO BE MARKED     : <cr>
ENDING AT: 14:46:23

<5.>1 LIST TEST AND STATUS. <5.>2 LIST TESTS FOR MARKER.
<5.>3 MARK STUDENT.         <5.>4 RETURN FROM MARKING.
ENTER CHOICE==>    : <cr>

1.START SESSION.    2.END SESSION.
3.EDIT STUDENT.     4.EDIT SYSTEM PARAMETERS.
5.MARK TEST.        6.SEND MESSAGES.
7.MONITOR STUDENT.  8.EXIT TO TSO.
ENTER CHOICE==>    : 6

SEND TO   : jpear04                -- Enter userid list
SEND FROM  : nobody                -- A personal identification
M: this is a message...
M: continue...
M: <cr>                            -- Use <cr> or blank line to send message
MESSAGE SENT TO USER: JPEAR04

SEND TO   : jpear,pear             -- Over-write the default string by re-type the userid list
SEND FROM  : nobody                -- Default is the string before
M: another message                 -- Hit a break key will terminate the sending facility
M: <cr>
MESSAGE SENT TO USER: JPEAR
MESSAGE CANNOT SEND TO USER: PEAR   -- Userid is incorrect or not signed on

SEND TO   : JPEAR
SEND FROM  : nobody
M: <cr>                            -- Terminate the message sending facility
```

```
1.START SESSION.    2.END SESSION.
3.EDIT STUDENT.     4.EDIT SYSTEM PARAMETERS.
5.MARK TEST.        6.SEND MESSAGES.
7.MONITOR STUDENT.  8.EXIT TO TSO.
ENTER CHOICE==>     : 7
ENTER INSTRUCTOR PASSWORD      : instpass

STARTING AT: 14:48:12
<7.>1 LIST LOG FILE.      <7.>2 WATCH STUDENT ACTIVITIES.
<7.>3 TURN OFF WATCHING.  <7.>4 RETURN TO MAIN MENU.
ENTER CHOICE==>     : 1

CURRENT DATE IS: 86/06/04
LIST LOG STARTING AT (YY/MM/DD) DATE       : 0   -- List the whole log file
LIST LOG ENDING AT (YY/MM/DD) DATE       : 86/06/04-- Default is the current date

TYPE              STUD.#  STUDENT NAME  TEST STATE PROCTOR STATE UNIT   QUESTIONS    TEST PROCTOR   TIME
CREATE LOG                86/06/04                                                                  12:46:08
START PSI         JPEAR   86/06/04                                                                  14:32:53
NEW SESSION       JPEAR   86/06/04                                                                  14:38:27
GENERATE TEST     1       NEW NAME 1    WRITING     AVAILABLE       0   2   5   1   0.000   0.000 14:39:02
WANT TEST MARKED  1       NEW NAME 1    NO MARK YET AVAILABLE       0   2   5   1   0.000   0.000 14:39:26
GENERATE TEST     8                     WRITING     AVAILABLE       0   2   4   5   0.000   0.000 14:39:42
PASS              1       NEW NAME 1    NOT WRITING AVAILABLE       1   2   5   1   0.000   0.000 14:40:51
MARKED BY         INST
END SESSION       JPEAR   86/06/04                                                                  14:42:25
CONDITIONAL PASS  8                     NOT WRITING INITIAL
MARKED BY         INST
LOG RECORD LISTED:      9
TOTAL LOG RECORD:       9
ENDING AT: 14:48:36

<7.>1 LIST LOG FILE.      <7.>2 WATCH STUDENT ACTIVITIES.
<7.>3 TURN OFF WATCHING.  <7.>4 RETURN TO MAIN MENU.
ENTER CHOICE==>     : 1

CURRENT DATE IS: 86/06/04
LIST LOG STARTING AT (YY/MM/DD) DATE       : 86/06/04 -- List log records between two dates
LIST LOG ENDING AT (YY/MM/DD) DATE       : 86/06/04
```

```
TYPE             STUD.#  STUDENT NAME  TEST STATE PROCTOR STATE UNIT   QUESTIONS    TEST PROCTOR   TIME
CREATE LOG               86/06/04                                                                  12:46:08
START PSI        JPEAR   86/06/04                                                                  14:32:53
NEW SESSION      JPEAR   86/06/04                                                                  14:38:27
GENERATE TEST    1       NEW NAME 1    WRITING     AVAILABLE       0   2   5   1   0.000   0.000 14:39:02
WANT TEST MARKED 1       NEW NAME 1    NO MARK YET AVAILABLE       0   2   5   1   0.000   0.000 14:39:26
<break>
(C)ONTINUE OR (Q)UIT  : q          -- After hitting a break key, default is "Continue"
TOTAL LOG RECORD:     9            -- The total number of records in log file
ENDING AT: 14:48:59

<7.>1 LIST LOG FILE.     <7.>2 WATCH STUDENT ACTIVITIES.
<7.>3 TURN OFF WATCHING. <7.>4 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 2

TYPE             STUD.#  STUDENT NAME  TEST STATE PROCTOR STATE UNIT   QUESTIONS    TEST PROCTOR   TIME
CREATE LOG               86/06/04                                                                  12:46:08
START PSI        JPEAR   86/06/04                                                                  14:32:53
NEW SESSION      JPEAR   86/06/04                                                                  14:38:27
GENERATE TEST    1       NEW NAME 1    WRITING     AVAILABLE       0   2   5   1   0.000   0.000 14:39:02
WANT TEST MARKED 1       NEW NAME 1    NO MARK YET AVAILABLE       0   2   5   1   0.000   0.000 14:39:26
GENERATE TEST    8                     WRITING     AVAILABLE       0   2   4   5   0.000   0.000 14:39:42
PASS             1       NEW NAME 1    NOT WRITING AVAILABLE       1   2   5   1   0.000   0.000 14:40:51
MARKED BY        INST
END SESSION      JPEAR   86/06/04                                                                  14:42:25
CONDITIONAL PASS 8                     NOT WRITING INITIAL
MARKED BY        INST
START WATCHING STUDENT ACTIVITIES.
ENDING AT: 17:19:33

<7.>1 LIST LOG FILE.     <7.>2 WATCH STUDENT ACTIVITIES.
<7.>3 TURN OFF WATCHING. <7.>4 RETURN TO MAIN MENU.
ENTER CHOICE==>    : <cr>

CONTINUE TO WATCH STUDENT ACTIVITIES.

INVALID CHOICE:                    -- <cr> is not allowed because the watching is on
ENDING AT: 17:19:43
```

```
<7.>1 LIST LOG FILE.      <7.>2 WATCH STUDENT ACTIVITIES.
<7.>3 TURN OFF WATCHING. <7.>4 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 4            -- Need to type in the exact number for selection
ENDING AT: 17:19:56

WATCHING IS STILL ON.

1.START SESSION.    2.END SESSION.
3.EDIT STUDENT.     4.EDIT SYSTEM PARAMETERS.
5.MARK TEST.        6.SEND MESSAGES.
7.MONITOR STUDENT. 8.EXIT TO TSO.
ENTER CHOICE==>    : 1

CONTINUE TO WATCH STUDENT ACTIVITIES.

A NEW SESSION HAS NOT BEEN STARTED.
CURRENT TIME IS: 17:20:07
CURRENT CUT-OFF TIME IS: 19:20:00
DO YOU WISH TO ALTER CUT-OFF TIME? (Y)ES,(N)O      : N-- Default is "No"
(C)ONTINUE LAST SESSION OR START A (N)EW ONE?      : C-- Default is "Continue"
TEST ISSUE CUT-OFF TIME WILL BE: 19:20:00
SWITCH TERMINAL TO FULL DUPLEX NOW.

ENTER STUDENT NUMBER      : 4
ENTER PASSWORD      : <cr>
STARTING AT: 17:20:15
PROCTOR? (Y)ES,(N)O      : y
STUDENT#: 4        IS AVAILABLE FOR PROCTORING.
GENERATE TEST ON UNIT     1 (Y)ES,(N)O?      : Y
TEST GENERATED ON UNIT:    1, AT TIME: 17:20:19, QUESTIONS:    5,    4,    3
OK   TRANSACTION COMPLETE.
ENDING AT: 17:20:20

ENTER STUDENT NUMBER      : 6
ENTER PASSWORD      : <cr>
STARTING AT: 17:20:27
PROCTOR? (Y)ES,(N)O      : y
STUDENT#: 6        IS AVAILABLE FOR PROCTORING.
GENERATE TEST ON UNIT     1 (Y)ES,(N)O?      : Y
TEST GENERATED ON UNIT:    1, AT TIME: 17:20:34, QUESTIONS:    5,    2,    3
```

```
OK   TRANSACTION COMPLETE.
ENDING AT: 17:20:35
                                -- The messages from watching in every 5 seconds
GENERATE TEST    4       NAME 4          WRITING     AVAILABLE    0   5   4   3   0.000   0.000 17:20:20
GENERATE TEST    6                       WRITING     AVAILABLE    0   5   2   3   0.000   0.000 17:20:35

ENTER STUDENT NUMBER     : TA
ENTER PASSWORD     : <cr>       -- Wrong password is rejected

ENTER STUDENT NUMBER     : ta
ENTER PASSWORD     : tapass

STARTING AT: 17:20:52
1.START SESSION.   2.END SESSION.
3.EDIT STUDENT.    4.EDIT SYSTEM PARAMETERS.
5.MARK TEST.       6.SEND MESSAGES.
7.MONITOR STUDENT. 8.EXIT TO TSO.
ENTER CHOICE==>    : 7
ENTER INSTRUCTOR PASSWORD     : instpass

STARTING AT: 17:21:02
<7.>1 LIST LOG FILE.     <7.>2 WATCH STUDENT ACTIVITIES.
<7.>3 TURN OFF WATCHING. <7.>4 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 3          -- Turn off watching

TURNING OFF WATCHING.
ENDING AT: 17:21:08

<7.>1 LIST LOG FILE.     <7.>2 WATCH STUDENT ACTIVITIES.
<7.>3 TURN OFF WATCHING. <7.>4 RETURN TO MAIN MENU.
ENTER CHOICE==>    : 3          -- Do not allow second turn off

WATCHING IS ALREADY OFF.
ENDING AT: 17:21:18

<7.>1 LIST LOG FILE.     <7.>2 WATCH STUDENT ACTIVITIES.
<7.>3 TURN OFF WATCHING. <7.>4 RETURN TO MAIN MENU.
ENTER CHOICE==>    : <cr>       -- <cr> is allowed because the watching is off
```

```
1.START SESSION.    2.END SESSION.
3.EDIT STUDENT.     4.EDIT SYSTEM PARAMETERS.
5.MARK TEST.        6.SEND MESSAGES.
7.MONITOR STUDENT.  8.EXIT TO TSO.
ENTER CHOICE==>     : 1

THIS IS A CONTINUATION OF LAST SESSION.
CURRENT TIME IS: 17:22:07
CURRENT CUT-OFF TIME IS: 19:20:00
DO YOU WISH TO ALTER CUT-OFF TIME? (Y)ES,(N)O      : N-- Default is "No"
(C)ONTINUE LAST SESSION OR START A (N)EW ONE?      : C-- Default is "Continue"
TEST ISSUE CUT-OFF TIME WILL BE: 19:20:00
SWITCH TERMINAL TO FULL DUPLEX NOW.

ENTER STUDENT NUMBER      : 3
ENTER PASSWORD      : <cr>
STARTING AT: 17:23:36
WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O      : y
CURRENT UNIT:  0, TEST POINTS:   0.00, PROCTOR POINTS:   0.00, TERM PROJECT:   0.00, FINAL EXAM:   0.00
TOTAL POINTS:   0.00, LETTER GRADE:
PROCTOR? (Y)ES,(N)O      : y
STUDENT#: 3        IS AVAILABLE FOR PROCTORING.
GENERATE TEST ON UNIT    1 (Y)ES,(N)O?      : y
TEST GENERATED ON UNIT:   1, AT TIME: 17:23:36, QUESTIONS:   3,   1,   4
OK   TRANSACTION COMPLETE.
ENDING AT: 17:23:37

ENTER STUDENT NUMBER      : 3
ENTER PASSWORD      : <cr>
STARTING AT: 17:23:37
WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O      : N
PROCTOR? (Y)ES,(N)O      : <cr>
STUDENT#: 3        IS AVAILABLE FOR PROCTORING.
DO YOU WANT YOUR TEST (C)ANCELLED OR (P)ROCTORED?      : p
FIRST PROCTOR SELECTED, STUDENT#: 8       , NAME:
SECOND PROCTOR SELECTED, STUDENT#: 1       , NAME: STUDENT NAME 1
OK   TRANSACTION COMPLETE.
ENDING AT: 17:23:38
```

```
ENTER STUDENT NUMBER      : 3
ENTER PASSWORD      : <cr>
STARTING AT: 17:23:38
WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O      : N
PROCTOR? (Y)ES,(N)O      : <cr>
STUDENT#: 3        IS AVAILABLE FOR PROCTORING.
YOU MUST HAVE YOUR CURRENT TEST PROCTORED BEFORE PROCEEDING.
OK   TRANSACTION COMPLETE.
ENDING AT: 17:23:38

ENTER STUDENT NUMBER      : 1
ENTER PASSWORD      : 1pass
STARTING AT: 17:23:39
WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O      : N
YOU HAVE BEEN SELECTED TO PROCTOR A TEST FOR STUDENT NUMBER: 3
STUDENT NAME:STUDENT NAME 3
ENTER UNIT NUMBER WRITTEN BY THE STUDENT      : 1
ENTER A QUESTION NUMBER WRITTEN BY THE STUDENT      : 3
ENTER A QUESTION NUMBER WRITTEN BY THE STUDENT      : 1
ENTER A QUESTION NUMBER WRITTEN BY THE STUDENT      : 7
STUDENT WAS NOT ISSUED QUESTION NUMBER:    7
ENTER A QUESTION NUMBER WRITTEN BY THE STUDENT      : 8
STUDENT WAS NOT ISSUED QUESTION NUMBER:    8
ENTER A QUESTION NUMBER WRITTEN BY THE STUDENT      : 4
(P)ASS,(C)ONDITIONAL PASS,(R)ESTUDY?      : p
PASS RESULT ENTERED.
PROCTOR? (Y)ES,(N)O      : y
STUDENT#: 1        IS AVAILABLE FOR PROCTORING.
OK   TRANSACTION COMPLETE.
ENDING AT: 17:23:40

ENTER STUDENT NUMBER      : 8
ENTER PASSWORD      : <cr>
STARTING AT: 17:23:41
WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O      : N
YOU HAVE BEEN SELECTED TO PROCTOR A TEST FOR STUDENT NUMBER: 3
STUDENT NAME:STUDENT NAME 3
ENTER UNIT NUMBER WRITTEN BY THE STUDENT      : 2
STUDENT DID NOT WRITE TEST ON UNIT:    2
ENTER UNIT NUMBER WRITTEN BY THE STUDENT      : 2   -- 3 extra trials
```

```
STUDENT DID NOT WRITE TEST ON UNIT:   2
ENTER UNIT NUMBER WRITTEN BY THE STUDENT      : 4
STUDENT DID NOT WRITE TEST ON UNIT:   4
ENTER UNIT NUMBER WRITTEN BY THE STUDENT      : 5
STUDENT DID NOT WRITE TEST ON UNIT:   5
(P)ASS,(C)ONDITIONAL PASS,(R)ESTUDY?      : p -- Do not accept, wrong guess
NOT ALLOWED TO PASS. NO RESULT  ENTERED.
OK   TRANSACTION COMPLETE.
ENDING AT: 17:23:41

ENTER STUDENT NUMBER      : 8
ENTER PASSWORD      : <cr>
STARTING AT: 17:23:42
WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O      : N
YOU HAVE BEEN SELECTED TO PROCTOR A TEST FOR STUDENT NUMBER: 3
STUDENT NAME:STUDENT NAME 3
ENTER UNIT NUMBER WRITTEN BY THE STUDENT      : 1
ENTER A QUESTION NUMBER WRITTEN BY THE STUDENT      : 7  -- Wrong guess
STUDENT WAS NOT ISSUED QUESTION NUMBER:   7
ENTER A QUESTION NUMBER WRITTEN BY THE STUDENT      : 8
STUDENT WAS NOT ISSUED QUESTION NUMBER:   8
ENTER A QUESTION NUMBER WRITTEN BY THE STUDENT      : 5
STUDENT WAS NOT ISSUED QUESTION NUMBER:   5
(P)ASS,(C)ONDITIONAL PASS,(R)ESTUDY?      : c
NOT ALLOWED TO PASS. NO RESULT ENTERED.
OK   TRANSACTION COMPLETE.
ENDING AT: 17:23:42

ENTER STUDENT NUMBER      : 8
ENTER PASSWORD      : <cr>
STARTING AT: 17:25:43
WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O      : N
YOU HAVE BEEN SELECTED TO PROCTOR A TEST FOR STUDENT NUMBER: 3
STUDENT NAME:STUDENT NAME 3
ENTER UNIT NUMBER WRITTEN BY THE STUDENT      : <cr>  -- Do not accept, no guess
(P)ASS,(C)ONDITIONAL PASS,(R)ESTUDY?      : p
NOT ALLOWED TO PASS. NO RESULT ENTERED.
OK   TRANSACTION COMPLETE.
ENDING AT: 17:25:43
```

```
ENTER STUDENT NUMBER      : 8
ENTER PASSWORD      : <cr>
STARTING AT: 17:25:44
WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O      : N
YOU HAVE BEEN SELECTED TO PROCTOR A TEST FOR STUDENT NUMBER: 3
STUDENT NAME:STUDENT NAME 3
ENTER UNIT NUMBER WRITTEN BY THE STUDENT      : <cr>
(P)ASS,(C)ONDITIONAL PASS,(R)ESTUDY?      : r
WAS QUESTION NUMBER:    3 COMPLETED SUCCESSFULLY (Y)ES,(N)O?      : n
WAS QUESTION NUMBER:    1 COMPLETED SUCCESSFULLY (Y)ES,(N)O?      : Y
WAS QUESTION NUMBER:    4 COMPLETED SUCCESSFULLY (Y)ES,(N)O?      : Y
PROCTOR? (Y)ES,(N)O      : y
STUDENT#: 8        IS AVAILABLE FOR PROCTORING.
OK    TRANSACTION COMPLETE.
ENDING AT: 17:25:44

ENTER STUDENT NUMBER      : 3
ENTER PASSWORD      : <cr>
WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O      : N
PROCTOR? (Y)ES,(N)O      : <cr>
STUDENT#: 3        IS AVAILABLE FOR PROCTORING.
YOU HAVE NOT COMPLETED 20 MINUTES OF RESTUDYING TIME.
OK    TRANSACTION COMPLETE.
ENDING AT: 17:25:44

ENTER STUDENT NUMBER      : 7
ENTER PASSWORD      : <cr>
STARTING AT: 17:25:45
WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O      : N
PROCTOR? (Y)ES,(N)O      : y
STUDENT#: 7        IS AVAILABLE FOR PROCTORING.
GENERATE TEST ON UNIT     1 (Y)ES,(N)O?      : y
TEST GENERATED ON UNIT:    1, AT TIME: 17:25:45, QUESTIONS:    2,    3,    4
OK    TRANSACTION COMPLETE.
ENDING AT: 17:25:46

ENTER STUDENT NUMBER      : 7
ENTER PASSWORD      : <cr>
```

```
STARTING AT: 17:28:46
WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O     : N
PROCTOR? (Y)ES,(N)O      : <cr>
STUDENT#: 7        IS AVAILABLE FOR PROCTORING.
DO YOU WANT YOUR TEST (C)ANCELLED OR (P)ROCTORED?     : c
OK   TRANSACTION COMPLETE.
ENDING AT: 17:28:47

ENTER STUDENT NUMBER     : 7
ENTER PASSWORD      : <cr>
STARTING AT: 17:28:47
WANT TO VIEW YOUR CURRENT COURSE STANDING? (Y)ES,(N)O     : N
PROCTOR? (Y)ES,(N)O      : <cr>
STUDENT#: 7        IS AVAILABLE FOR PROCTORING.
YOU HAVE NOT COMPLETED 20 MINUTES OF RESTUDYING TIME.
OK   TRANSACTION COMPLETE.
ENDING AT: 17:28:48

ENTER STUDENT NUMBER     : ta
ENTER PASSWORD      : tapass
STARTING AT: 17:30:52
1.START SESSION.    2.END SESSION.
3.EDIT STUDENT.     4.EDIT SYSTEM PARAMETERS.
5.MARK TEST.        6.SEND MESSAGES.
7.MONITOR STUDENT.  8.EXIT TO TSO.
ENTER CHOICE==>     : 8            -- Return to mantes

C:
```