NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MICROCOMPUTER PROGRAM DESIGN CONSIDERATIONS
FOR THE NOVICE USER

by

David C. Moore

March 1987

Thesis Advisor: Norman Lyons

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 54	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 94943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
MICROCOMPUTER PROGRAM DESIGN CONSIDERATIONS FOR THE NOVICE USER

12 PERSONAL AUTHOR(S)
Moore, David C.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1987 March	131

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	novice user; computer interface design/ considerations; computer interface

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The purpose of this thesis is to present the issues and considerations related to the development and implementation of a user interface for a microcomputer-based application program. The interface design goal is to enable a novice user to fully utilize all application program functions without prior training or reference to a user's manual.

The results of the empirical evaluation of the user interface are presented together with an analysis in support of the effectiveness of a proposed interface design methodology and interface design considerations.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. Norman Lyons	22b TELEPHONE (Include Area Code) (408) 646-2666 22c OFFICE SYMBOL Code 54Lb

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

Approved for public release; distribution is unlimited

Microcomputer Program Design Considerations for the Novice User

by

David C. Moore
Lieutenant Commander, United States Navy
B.S., Ohio State University, 1976

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1987

ABSTRACT

The purpose of this thesis is to present the issues and considerations related to the development and implementation of a user interface for a microcomputer-based application program. The interface design goal is to enable a novice user to fully utilize all application program functions without prior training or reference to a user's manual.

The results of the empirical evaluation of the user interface are presented together with an analysis in support of the effectiveness of a proposed interface design methodology and interface design considerations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

This author would like to express his appreciation to Professor Norman R. Lyons for the professional guidance and educational insights he provided.

Additionally, this author wishes to thank his wife, Betty, for the support provided during this educational experience.

I. INTRODUCTION

The relatively recent, widespread proliferation of microcomputers into both the home and work place has resulted in a shifting of computer operation and, in some cases, programming tasks, from the traditional realm of trained, professional operators and programmers directly to the end user. Technological advances have reduced the skills necessary to energize and physically communicate with the hardware. However, the process of effectively interfacing with the hardware via the constructs of software of ever increasing complexity, often requires the new user to obtain a detailed working knowledge of a particular software system before the benefits of the system may be realized.

This requirement seems contrary to the conjecture expressed by Coombs and Alty [Ref. 1:p. 3] that the majority of users do not wish to be extensively trained in computing and employers certainly wish to minimize user training costs.

The purpose of this thesis is to develop and evaluate the effectiveness of interface techniques designed to eliminate any user, application-specific training prior to application program use. In order to provide an appreciation for the nature of interface design issues, Chapter 2

presents a review and analysis of interface evolution and the state of current thinking relating to interface design. Chapter 3 details the rationale and anticipated benefits of specific interface design decisions and techniques employed in the development of the research interface. In Chapter 4, the interface evaluation methodology and evaluation results are presented, discussed and analyzed. Finally, Chapter 5 suggests that the concept of including an interface requirements specification into the system design and development process is essential to the production of viable applications for novice users.

The scope of this research was intentionally limited to one application program's interface in order to more fully evaluate the effect of the employed interface. By this action, the empirical evaluation results and ensuing conclusions would not be general in nature and thus avoid a recapitulation of the generalized findings and recommendations currently presented in available literature.

Additional limitations imposed upon the design of the specific interface were based on the fact that the target microcomputer system's hardware consisted of 512 kilobytes of main memory, two 360 kilobyte diskette drives, a monitor, keyboard and printer. Admittedly, this particular hardware configuration precludes evaluation of such technically feasible interfacing approaches as the use of light pens, pressure sensitive screens or voice command. However, the

target system's configuration seems consistent with the assumption that the majority of general purpose microcomputer systems in use share the same general configuration and/or limitations.

II. USER INTERFACE ISSUES

As a result of technological advances in the computer field, a relatively new and immature field of study has arisen to explore principles and methods for better adapting computer systems to meet human needs. This fledgling field has, as yet, no simple title nor well established repertoire of concepts and techniques. The field is frequently referred to as "interface design" and "dialog engineering" [Ref. 2:p. 3].

A. USER INTERFACE EVOLUTION

Prior to the widespread use of time sharing systems, the vast majority of computers were operated in batch mode. As a result of batch processing, end users only indirectly interacted with the computer via operations personnel. Consequently, there was no reason for "user friendly" interfaces since the operators were trained professionals, knowledgeable of the requisite interface procedures.

Although the introduction of time sharing systems, enabling direct user interaction, generated an acknowledged need for "user friendly" interfaces, the pursuit of user interface design attributes was relegated to academia. This relegation was due to the fact that time sharing systems were achieved through the layering of complex and costly

software onto existing, batch oriented minicomputers and mainframes, and hardware and software providers did not find it economically feasible to reconstruct new, coordinated systems for existing machines [Ref. 3:pp. 338-339].

The advent of the microprocessor has had a profound impact on the computer industry. One of the most significant impacts was the dissolution of the long adhered to premise that computers were expensive and should be built with the minimum number of circuits, thus assuring efficiency [Ref. 4:pp. 110-123]. Consequently, it now became both technologically and financially feasible to consider the user's needs in the hardware and software design process.

B. THE PRESENT INTERFACE STATE

With the realization that it was now technically feasible to incorporate interface considerations into the design of a microcomputer system, such diverse professions as educationalists, psychologists and ergonomic specialists began contributing to the area of interface design. However, their findings and recommendations have not produced significant advances in interface design since these non-computer oriented professionals are rarely invited to participate in the design effort. On those occasions when they have become involved in the system design process, their contributions have been somewhat diminished due to a

lack of knowledge and appreciation of the machine's capabilities to make things easier for the user [Ref. 3:p. 339].

Since the mid-1970s there have been many studies and much written with respect to guidelines for the development of effective user interfaces. Unfortunately there is no well defined standard or authority and a fair amount of inconsistency from source to source [Ref. 5:pp. 25-25].

Although there may be inconsistencies between any two given studies, analysis of the various studies in aggregate has allowed later researchers to develop more comprehensive guidelines based upon previous, incomplete studies and the resolution of individual inconsistencies. Table 1 presents a highly generalized summary of desirable, interface attributes identified by Shneiderman [Ref. 6:pp. 216-244]. Gaines and Shaw [Ref 5:pp. 30-44] have taken the process one step farther and proposed more specific, interface design rules. These rules, together with the general interface attributes which they support, are presented in Table 2.

TABLE 1. DESIRABLE INTERFACE ATTRIBUTES

DESIRABLE INTERFACE ATTRIBUTES
1. Easy to learn.
2. Easy to use.
3. Easy to remember.
4. Prompt response times.
5. Reliable.
6. Courteous.
7. Helpful.

TABLE 2. INTERFACE DESIGN RULES

INTERFACE DESIGN RULE	SUPPORTED ATTRIBUTES
Use interface prototype or related system when discussing the interface with users.	Easy to learn/remember
Develop interface using user's model.	Easy to learn/remember
User should dominate computer.	Easy to use
System response/activity should be clear consequence of user's actions.	Easy to learn, helpful, reliable
System should adapt to user's expertise.	Easy to use
Provide for uniformity and consistency.	Easy to learn/remember
Ensure requisite information/memory aids are available to user throughout system.	Easy to use, helpful
User manuals should be based on actual user dialog.	Easy to learn, use and remember, helpful
Train through experience.	Easy to learn/remember
Make immediate, clear responses to inputs.	Courteous, prompt
Validate data on entry.	Reliable, courteous
Provide a reset/abort command.	Easy to use, reliable
Make corrections through re-entry.	Reliable

Although Shneiderman's interface attributes and Gaines' and Shaw's rules provide general direction for interface design, there remains much leeway for system design and programming personnel as to the actual implementation and interpretation of these attributes and rules. Peterson's and Silberschatz's observation seems to concisely sum up

the current state of user interface design:

> Users desire certain obvious properties in a system. The system should be convenient to use, easy to learn, easy to use, reliable, safe, and fast. Of course, these specifications are not very useful in the system design, since there is no general agreement on how to achieve these goals. [Ref. 8:p. 441]

III. RESEARCH INTERFACE DESIGN CONSIDERATIONS

Due to the myriad of possible, interactive computer applications, the specific application program and user group will often dictate the manner and degree of implementation of the generalized guidelines found in literature concerned with interface design.

A. THE APPLICATION PROGRAM

Although this research project is concerned with the user interface, it was deemed necessary to develop an application program with which to interface and to provide direction to the interface development.

The actual methods employed by the application program to satisfy the user's functional requirements are not germane to this research effort. Therefore only a brief description of the program's overall function is provided to establish a frame of reference.

The application program was developed specifically for the accountant of the Army Emergency Relief organization (AER) at Fort Ord, California. AER's function is to provide no interest loans to military personnel (primarily army) who satisfactorily demonstrate a valid need for financial assistance. The accountant's primary function is to record disbursement of the loan, post loan repayments to applicable

loan accounts and general ledger, and advise higher authority of any financial deviations or problems with respect to individual loan accounts. A secondary function requires the AER accountant to provide statistics of varying natures to higher authority upon request. Since a service member may have multiple, concurrent loans, the nominal size of AER's data base is on the order of 1900 to 2100 members and 2900 to 3200 loans. The AER application program basically provides for maintenance of individual loan accounts, general ledger and statistical information.

B. APPLICATION PROGRAM DESIGN THEORY

Much has been, and continues to be, written regarding computer program design and development. While various design and development methodologies are advocated in the literature, all have the expressed goal of producing good, working programs. Unfortunately, it seems as if the majority of methodologies stress design and development of the functional elements of a program with the user interface being of secondary concern. In other words, once the functional aspects of a program have been defined and designed, the interface is designed to fit the functional design structure.

The theory underlying the methodology used in the design and development of this research project is essentially a reversal of current design and development methodologies.

The theory proposes definition and design of the interface prior to, or at least concurrent with, functional design. This development approach is intended to place the interface issue at the forefront. Thus, functional design is driven not only by requirements specifications, but by interface considerations as well. While this approach may increase the difficulty and complexity of functional element design, the actual, internal methods employed are usually of little concern to the user. Assuming the system meets the user's functional specifications, the interface becomes the primary user issue. As noted by Eason and Damodaran with respect to users' perceptions of a computer system:

> It is of little interest to him [the user] that the system is a technical masterpiece, or that it serves another user very well; if it serves his task needs poorly, it stands condemned as a poor system. [Ref. 7:p. 116]

Since the goal of this research is to develop a system requiring no user training prior to use of the application program, interface issues are of paramount concern. In the following sections of this chapter, the issues pertaining to the design and implementation of the research interface are presented and discussed.

C. INTERFACE DESIGN PHILOSOPHY

Traditionally, the design of a "core" program to satisfy the user's functional requirements would be relatively straight forward. The goal is well defined; design the "core" program to perform the specified requirements. Since

the actual workings of this portion of the program are invisible to the user, one need only consider the technical aspects of the task; the user is of secondary concern.

However, the approach taken in the design of the research program requires that "core" related design decisions be made with respect to both the requirements specifications and interface considerations. Since a project's requirement specifications serve as the benchmark against which a program's functionality is assessed, the same approach was used with respect to interface design.

Unlike the requirement specification, which may be stated in such measurable metrics as response times and throughput rates, the interface specification is much more nebulous. The exact meanings of terms such as "easy to use" and "friendly" are highly individualistic and ambiguous. As a result, it is left to the designer or programmer to produce their interpretation of these ambiguous terms.

In order to develop an interface requirements specification, the attributes of a novice computer user were analyzed.

The term "novice user" is assumed to apply to an individual who is not, nor desires to become, an expert in, or familiar with, computer technology, but uses a computer to assist in the performance of assigned tasks. A generally accepted attribute of the novice user is the overall perception of the computer as a tool to assist in the performance

of a task. If the user deems the tool inappropriate for the task at hand or the effort to use the tool exceeds the return, the tool will experience little to no use.

Based on the attributes of a novice user, several assumptions were generated which formed the basis for the formulation of user interface specifications. First, the novice user's interests and aspirations lay outside the computer field and only limited time and effort could be expected to be devoted to mastering the application system. Second, the user would view the resulting system as a means to an end and not an end in itself, thus desiring to minimize time and effort devoted to system operation and output interpretation. Finally, the user would desire immediate answers to questions about the system without lengthy and time consuming reference to user and technical manuals.

As a result of the analysis and assumptions, an interface requirements specification was developed in the form of a questionnaire, against which candidate interface designs were evaluated prior to implementation. The contents of this questionnaire are presented as Table 3.

Only after an interface design idea met the requirements of the interface specification were the technical implementation issues addressed. Basically, the design philosophy was to adapt the program to the needs of the user versus forcing the user to adapt to the needs of the program.

TABLE 3. RESEARCH INTERFACE REQUIREMENTS SPECIFICATION

INTERFACE REQUIREMENTS SPECIFICATIONS	RESPONSE
1. Does the interface contain references, concepts or words words unique to the computer field?	No
2. Does the interface require user inputs/actions which have no identifiable counterpart or rationale in the corresponding manual process?	No No
3. Does the interface contain all necessary information to accomplish the desired operation?	Yes
4. Does the interface require the minimum, necessary user actions to complete the operation?	Yes
5. Is the interface consistent with previously developed interfaces?	Yes
6. Does the interface provide for immediate and positive error detection and correction/recovery?	Yes

D. THE USER COMMAND INTERFACE

Since the target computer system's primary input device was the keyboard, there appeared only three viable command entry modes: a menu system, a command language or a combination of the two. The selection of a menu system for the research interface reflects the observation of Reid that:

> Menus have been recommended for occasional and novice users as they reduce the amount of information the user needs to remember. [Ref. 9:p. 111]

As with many concepts, there are some disadvantages associated with a menu driven system, which, if not handled effectively, can negate the concept's overall usefulness.

The mere fact that the display screen of a computer system encompasses a finite area limits the number of options which may be displayed on a given screen.

If a system offers more options than can be displayed on one screen, it may be tempting to reduce the space occupied by each option description. However, if the option descriptions become too cryptic, the primary advantage of a menu system is lost as the user now must acquire and remember the meaning of each option.

Another alternative would be a system of layered menus, where the selection of an option from the primary or main menu would produce another menu and so on until the menu containing the desired operation was encountered. The main problem associated with this approach is one of navigation. As one progresses through successive menu layers, it becomes difficult to determine one's location in the system relative to a known point of reference, in this case the main menu [Ref. 9:p. 111]. Loss of a frame of reference can disorient and confuse the user, as humans are accustomed to using the space and objects around them for organization and establishment of frames of reference [Ref. 10:pp. 1-3].

The research program has 47 different options. Since all 47 could not be displayed on a single screen without becoming too cryptic, a system was required that preserved the advantages of a menu driven system and avoided the potential disadvantages. The resulting main menu consists

of the 10 general operations depicted in Figure 1, through which all 47 options are accessible. Limiting the main menu to 10 operations provided enough room for non-cryptic operation identification. However, this action necessitated a layering of subordinate menus. To avoid the navigation problem, these subordinate menus are presented as windows or panels on top of the main menu. The intent of this approach is to create the illusion that the user is still in the main menu section of the program, thus preserving the user's frame of reference. Figure 2 shows an example of operation three's subordinate menu. Since many of the available operations use the same input/output displays, there are only six display screens, including the main menu, in the system. Depending upon which option is selected the user will see one of five input/output screens. The only place the user can go from an input/output screen is back to the main menu. Thus there is no navigation problem for the user to contend with; the user is either viewing the main menu or an input/output screen.

E. INTERFACE DIALOG DESIGN

For the purposes of designing the research interface, the term dialog was defined as two-way communication. Stoner notes that two-way communication is a complex process where a receiver provides feedback to the sender of a message [Ref. 11:p. 496-499]. In the case of the research,

	AER Loan and Financial Accounting System MAIN MENU	14 FEB 87

LOAN or ACCOUNT OPERATIONS	OTHER OPERATIONS
1. Enter a NEW Loan.	6. Display/Correct General Statistics
2. Enter a TRANSFER-IN Loan.	7. Display/Correct General Ledger
3. Post Loan Repayments.	8. Post Disbursement of a GRANT
4. View, Change, Correct or Delete.	9. Post Misc Receipts/Disbursements
5. Reenter Accidentally Deleted Loan	10. Print Out a Selected Report

AER Loan Account Statistics

Current	CH-13	Delinqt	Uncoll	Paid-Off	Trans-In	Trans-Out	Accounts	Loans
1857	34	23	14	62	15	12	1954	2017

REMEMBER: Pressing ESC at any time will return you to this display.
Press F1 - F10 for more information about each operation.

Please Enter the Desired Operation Number (press ESC to QUIT)

Figure 1. Research Interface Main Menu Display

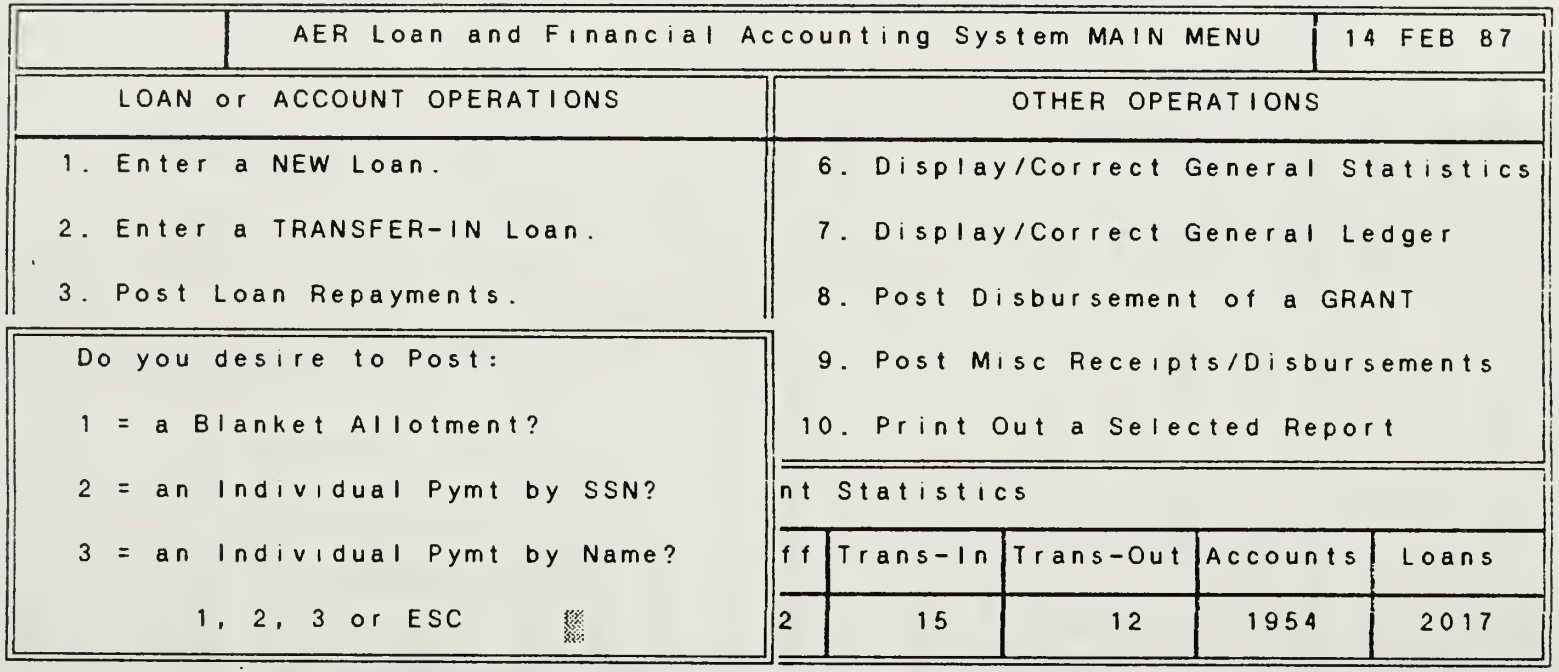


Figure 2. Sub-Menu Display

application program, the user is considered the sender and the program the receiver providing feedback.

When humans receive feedback, there is more involved than simply content. The message is evaluated with respect to the source, read between the lines for hidden meanings, and words interpreted with respect to our understanding of the word. [Ref. 12:pp. 238-246]

Since feedback can convey more than physical message content, a detailed analysis and design of the feedback mechanism, with emphasis on human perceptions and attributes, was seen as a means to convey the image of a "friendly" system to the user.

The primary perception the interface was designed to convey was system servility. By so doing, it was envisioned that the novice user would view the system as a capable and willing servant and not a system requiring user submission.

The resulting system prompts for user actions were simply displayed as requests versus commands. Instead of displaying a message such as: Enter the desired option, the message was displayed as: Please enter the desired operation number. The innocuous inclusion of the word "please" changes the perception of the message from a command to a request, and may even convey the impression of a personable, polite computer.

The other type of system message analyzed was the error message. To maintain the perception of system servility,

error messages of an informative nature were designed to be almost apologetic as opposed to cryptic chastisements. An example of an informational error message is the case where the user requests display of information not held in the system. The system responds with: "I'm sorry, I can't seem to locate the desired account".

Error or abnormal situation messages requiring user action, are presented as a system plea for user assistance. The intended user perception of these messages is that the user is in complete control of a personified system. Figure 3 is depicts the abnormal situation message displayed when the system cannot determine to which loan the payment is to be applied. Figure 4 is the window displayed when a printer fault is detected.

The final type of error response coded into the system consists of a short, audio "beep" when illegal keyboard entry is detected. Whenever a key is depressed, the system immediately analyzes the input to determine compatibility with the type of input field. If it is a valid entry, the character is displayed, otherwise the "beep" sound is produced. The user receives instantaneous feedback and does not waste time and effort entering an entire data string only to be informed after entry that it is an invalid input.

Although the audio signal alone does not identify the exact error, the accompanying field windows are designed to contain all requisite information to enable the user to

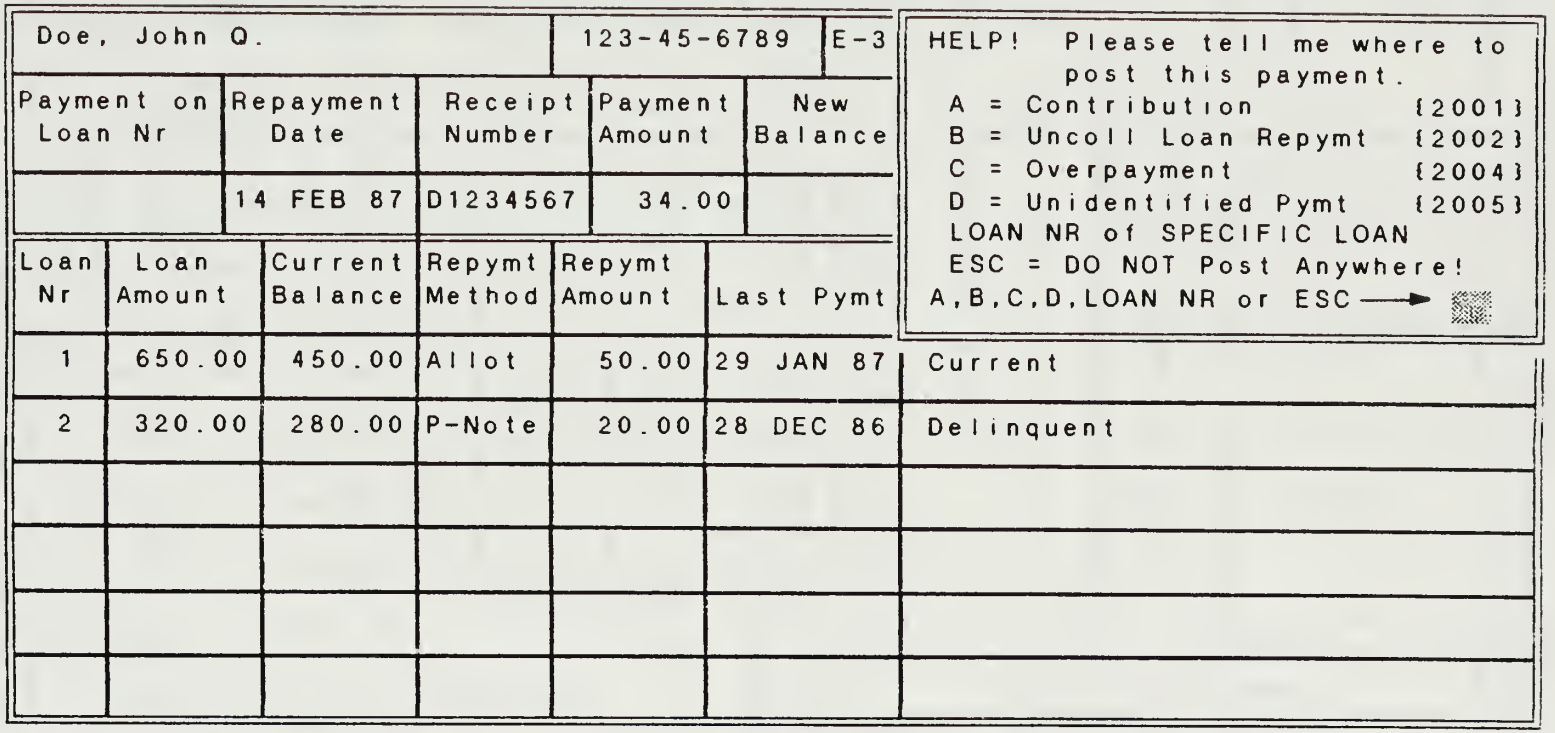
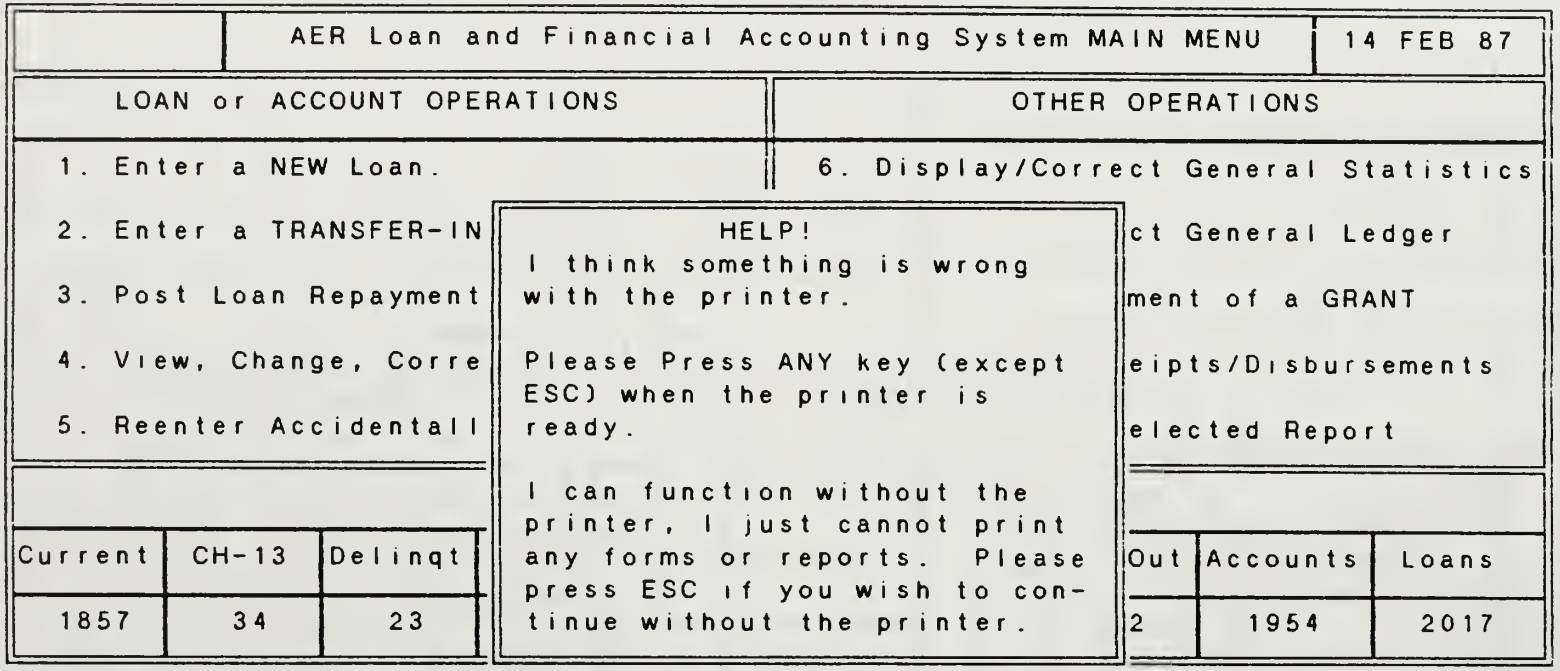


Figure 3. User Assistance Request Display



REMEMBER: Pressing ESC at any time will return you to this display.
Press F1 - F10 for more information about each operation.

Which Operation Number do you desire? (press ESC to QUIT)

Figure 4. System Error Detection Display

determine the necessary input. The audio signal is designed primarily as a courtesy to inform the user of accidentally depressed keys while protecting the system from input type mismatches.

F. THE ESCAPE MECHANISM

Assuming a novice user will probably probe the system during the familiarization process, it was decided to install a mechanism which would immediately halt whatever process the user was doing and return to the main menu. As recommended by Gaines and Shaw:

> Provide a reset command that cleanly aborts the current activity back to a convenient checkpoint. The user should be able at any stage in a transaction to abort it cleanly with a system command that takes him back to a well defined checkpoint as if the transaction had never been initiated. [Ref. 5:p. 42]

The system command selected for the research program was the Esc key. In order to preserve simplicity and limit the amount of system related knowledge required of the user, the Esc key is the only "special function" key the user must remember. To aid the user's retention, many of the system prompts contain reference to the Esc key.

G. ERGONOMIC CONSIDERATIONS

The primary issue in this area was to develop the physical actions necessary for communicating with the system which would not be ambiguous or meaningless to the novice user while not frustrating or impeding the user as more

experience was gained. Analysis of this issue revealed two primary areas warranting in-depth design consideration.

The first area considered was direction of the system. The selection of a menu driven system with its enumerated options seemed a viable method of direction for both the novice and expert. Since the menu identifies the available options, the novice user has all the requisite information available to initiate the desired process. For the user who has gained familiarity with the system, the process of option selection is fast, requiring only those keystrokes necessary to select the option. There are no special keys, complex keystroke sequences, or English-like commands to confuse the novice or slow down the expert. To further ease the selection process, the numeric keypad was placed in the numeric entry mode by the program. While the horizontally arranged, numeric keys across the top of the keyboard remain functional, the numeric keypad allows all necessary operation selection and numeric data entry to be performed from one keyboard location with a minimum of physical movement.

The decision to use numeric option selection codes was influenced by the ability of humans to cognitively process numbers faster (27-39 msec/number) than letters or icons (40-93 msec/item) [Ref. 13:p. 43]. If the user is not an accomplished typist, numeric entry should be easier and quicker than having to search the standard "QWERTY" keyboard for the desired letter.

The other area considered involved the implementation of an on-line assistance facility. In order to provide maximum assistance to the novice user and not impede the expert, help panels or windows describing the purpose or required input field contents are displayed by default. By so doing, the novice user requires no knowledge of a special mechanism to invoke on-line assistance. Since there is no invoking mechanism, there is no change of program mode from the current process, to the assistance mode, then back to the process. Thus the expert user may ignore the assistance display and continue as if the display was not present. An example of an assistance window is presented in Figure 5.

Since the target system's keyboard has a numeric keypad, the system allows numeric entry from the numeric keypad for purely ergonomic reasons of speed and physical ease of data entry. The numeric keys across the top of the keyboard may also be used, however, the physical arrangement of the numeric keypad reduces then time and movement necessary to enter a desired numeric input.

H. DISPLAY COLOR CONSIDERATIONS

Colors in themselves were not seen as an information transmittal medium. Color combinations were selected when necessary to draw user attention. Light, complementary colors were used overall to provide a soothing display. The background is a very light blue, lines are in light yellow

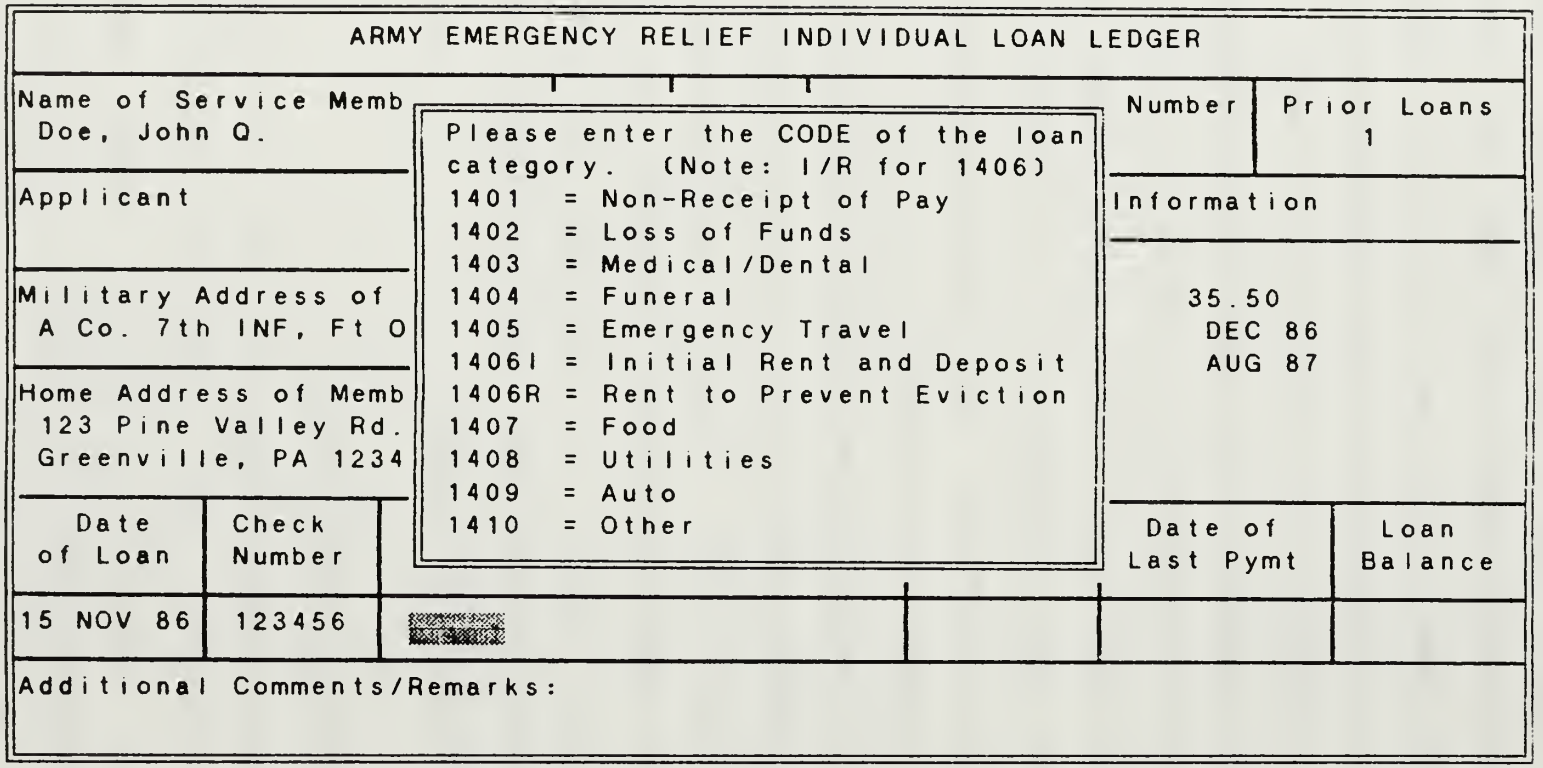


Figure 5. Assist Window Display

and column headings are in white. The assistance windows consist of a red background with white and/or black foreground characters. The choice of red for assistance window backgrounds is not meant to imply an emergency situation, but merely to contrast with the overall blue background and thus draw attention to the window.

I. DESIGN SUMMARY

The purpose of this chapter is not to provide specific interface implementations, as it is realized that the specific application will largely determine the interface structure. Rather, the intent is to propose some basic philosophies that may be useful when designing an interface. A summary of the research interface constructs and Table 1 attributes supported is presented as Table 4.

As previously noted, the primary philosophy behind the majority of the research, interface, design decisions was to adapt the system to the user and not require the user to adapt to arbitrarily defined constructs of the system. It is realized that there are unavoidable constructs to which a user must adapt, such as using the keyboard for communication. However, adherence to this primary philosophy by system designers and programmers should reduce or eliminate the number of arbitrary constructs introduced into the system.

TABLE 4. INTERFACE ATTRIBUTES SUPPORTED BY THE RESEARCH INTERFACE

RESEARCH INTERFACE CONSTRUCT	SUPPORTED ATTRIBUTE
1. Menu command system.	Easy to learn/use/ remember.
2. Sub-menu display overlays.	Easy to use, helpful.
3. Entry type checking upon individual character entry with audio error signal.	Prompt response, reliable.
4. Default display of assistance/instruction windows.	Easy to use, helpful.
5. Content of assistance/instruction/error windows.	Courteous, helpful.
6. Display coloration.	Helpful.
7. No multi/special function keys other than the ESC key.	Easy to learn/use/ remember, reliable.
8. Consistent displays and I/O requirements.	Easy to learn/use/ remember, helpful, reliable.
9. Activation of numeric keypad for option selection and data entry.	Easy to use.
10. Use of ESC key to abort any process at any operation at any time.	Easy to learn/use remember, prompt response, helpful, courteous.

A supporting philosophy or concept suggests a realization by design and programming personnel that the user of the resulting system probably does not have an interest in the computer field and views the system simply as a means or tool to assist in the performance of a task or function. The implication of this concept is that interface

constructs which are meaningful to development personnel, due to their level of computer expertise, may be quite meaningless or confusing to the end user. It is therefore proposed that interface design decisions should be made under the assumption that the user has no knowledge of the computer field and with respect to user perceptions and expectations.

IV. EVALUATION OF THE RESEARCH INTERFACE

In order to assess the validity of the assumptions and theories underlying development of the research interface and the results of their aggregation, it was deemed appropriate to evaluate the resulting interface on novice users. The purpose of this chapter is to present the evaluation methodology and results of the evaluation.

A. EVALUATION METHODOLOGY

The basic methodology required a novice user to attempt ten predefined operations with the application system. Although the application system provides for 47 different operations, many are minor variations of a general operations. The ten operations selected for evaluation were representative of ten general areas. The user was first given a written description of the evaluation procedure and a brief background scenario to establish the interaction environment. The user's performance was then observed, noting actions taken or not taken and problems encountered. Upon completion of the ten operations, the user was given the questionnaire reproduced as Figure 6 to record his impressions and feelings about the evaluation session. The background scenario and performance tasks used for the evaluation process are presented as Appendix A.

EXPERIMENT QUESTIONNAIRE

Please answer the following questions by circling the response which best describes your opinion.

1. I found the color schemes displayed on the computer screen:
 A. Distracting B. Had no real affect C. Helpful D. Very Helpful

2. I found the "Beep" sound when I made a typing error:
 A. Distracting B. Had no real affect C. Helpful D. Very Helpful

3. The overall appearance and layout of the computer screens was:
 A. Distracting B. Had no real affect C. Helpful D. Very Helpful

4. The appearance of the assist windows or panels was:
 A. Distracting B. Had no real affect C. Helpful D. Very Helpful

5. The information contained in the assist windows or panels:
 A. Distracting B. Had no real affect C. Helpful D. Very Helpful

6. The ability to return to the main menu at any time by pressing ESC is:
 A. A bad concept B. Okay in some situations, not all C. No opinion
 D. Reassuring E. Highly reassuring

7. In general, I felt:
 A. The program was very difficult to work with.
 B. The program neither helped or hindered my accomplishment of the various operations.
 C. The program helped in my accomplishment of all the operations.
 D. The program greatly helped in my accomplishment of all the operations.

8. Assuming you are an experienced AER accountant and were given a computer and this program, do you feel you:
 A. Would desire extensive training before using this program?
 B. Would desire some training before using this program?
 C. Would require no training to use this program?

9. I would summarize my feelings about this computer session as:
 A. Frustrating B. Challenging C. No opinion
 D. Satisfying E. Very Satisfying

10. The following is optional, however, any comments or recommendations regarding your session with the program would be greatly appreciated.

Figure 6. Evaluation Session Questionnaire

As previously noted, the development objective of allowing a novice user to use the system without prior training is based on the assumption that the user is familiar with the processes and procedures required for manual accomplishment of the various tasks. In order to maintain the validity of this assumption, evaluation session users were selected from personnel assigned to the installation activity. The intention of limiting the scope of prospective evaluation session users was to increase the probability that the participants would posses enough knowledge of the target user's job functions to allow for a meaningful evaluation of the system interface. The only other user selection criteria was the requirement that participants have no prior experience with a microcomputer based system.

Due to the small size of the installation activity and the restrictions placed on the selection of evaluation session participants, a total of six participated in the interface evaluation. While it may appear that six evaluations are not statistically significant, the extremely high data correlation of the individual results implies further evaluations probably would not have generated significantly different results.

B. EVALUATION SESSION OBSERVATIONS

Aggregate analysis of the observations recorded during the interaction sessions revealed two distinct behavior patterns which resulted in the classification of the users as type A and B.

Although all participants were informed that any actions, short of physical violence, would not damage the computer or the program and were encouraged to experiment, this seemed to have had little impact on their initial actions. Each participant appeared to approach the first task with extreme trepidation. Having correctly determined the option number required for the operation, users were observed to make several false starts before physically selecting the option. Following each aborted keystroke the participant would return to an examination of the main menu. Once the selection was finally made and the input/output screen appeared on the display screen, each participant was observed to display one of two reactions. Users later categorized as type A would immediately begin intense examination of the new display. Type B users would invariably allow themselves an audio and/or physical expression of self satisfaction before turning their attention to the new display.

Having correctly invoked the input screen for the first operation, both user types successfully completed the required input actions and returned to the main menu upon

completion. However, type A users were observed to proceed with the data entry process at a slower pace than type B users. When the audio, error signal was produced, signifying illegal data entry, type B users recovered faster than type A users, and were quicker to correct their mistake and proceed. Type A users responded to the error signal by returning to an intense examination of the display.

All participants exhibited a positive learning curve as inferred by steady increases in task performance speed as the session progressed. Although the sessions were not timed, type B users tended to spend progressively less time evaluating and reacting to each new display screen. Type A users continued methodical examination of each display, with an observable increase in data entry and option selection speeds.

Analysis of the observations seems to suggest definitive characteristics of the two user types. The two type A users appeared uncomfortable with the trial and error approach of operation accomplishment. Much time was spent analyzing the displays as if searching for information which would reduce the risk of the next keystroke. Type A users seemed highly task oriented, resenting anything perceived as barring task accomplishment. If these users experienced any self satisfaction of increased confidence in their abilities to interact with the system, it was not observable.

Type B users seemed to display an entirely different approach to the tasks. They were more prone to experimentation and displayed obvious satisfaction upon successful completion of seemingly trivial tasks. Type B users appeared to develop a familiarity with system constructs and characteristics more rapidly than type A users. While type A users seemed to view each new operation as disjoint from previous operations, type B users tended to recognize and transfer the lessons learned from previous operations. Type B user sessions tended to evolve into a friendly competition between man and machine with the users frequently issuing friendly, verbal challenges to the computer.

C. POST-SESSION QUESTIONNAIRE ANALYSIS

The tabulated responses to the post-session questionnaire (Figure 6) are presented in Table 4. As may be noted, responses to the first six categories relating to interface design constructs were awarded the highest ratings. This positive feedback, coupled with the fact that all participants successfully completed all operations tends to suggest that the interfaces associated with each operation were sufficient to permit accomplishment. The responses to question seven, dealing with overall ease of use, supports the previous six responses in aggregate.

Responses to question eight, concerning prior training desirability, were, initially, the most disturbing, as the

TABLE 5. POST EVALUATION SESSION QUESTIONNAIRE RESPONSE DISTRIBUTION

Response Category	Response Letters from Figure 6 Low ◄——► High				
	A	B	C	D	E
Display Colors	0	0	0	6	*
Audio Error Signal	0	0	0	6	*
Display Format	0	0	0	6	*
Assist Windows	0	0	0	6	*
Window Content	0	0	0	6	*
ESCape Construct	0	0	0	0	6
Ease of Use	0	0	0	6	*
Prior Training Desirability	0	4	2	*	*
Overall Impression	0	1	0	1	4

* signifies no question provided

main objective of this research was the development of an application program requiring no formal user training. The validity of the four responses indicating a desire for training prior to system use was questioned due to the fact that all participants successfully completed all evaluatory operations without prior training. To resolve this apparent dichotomy, the participants were interviewed as to the reasons for their responses.

The interviews disclosed two basic reasons for the responses. First, there was an assumption by the participants that the program had more capabilities than those to which they had been exposed. Thus, prior training would be necessary to enable effective realization of those unknown capabilities. The other reason had to do with the application for which the program was designed. The application program was designed for the organization's accountant. As recommended by Gaines and Shaw [Ref. 5:p. 30], the system was developed to emulate the user's model of the programed functions. As a result many of the interfaces employ accounting terminology and procedures. Though five of the participants had a general knowledge of the account's duties, none were well versed in the specifics of the accounting field. As a result, one underlying reason for the given response was an identified deficiency in the area of accounting. This revelation diminished the usefulness of the overall response for interface evaluation purposes, as one of the assumptions upon which the interface design is based is user knowledge of the functional aspects of the application.

Of the responses to question nine, which requested a subjective judgement of the evaluation session in general, four participants, classified as type B users, considered it very satisfying. Of the two type A users, one judged the session as satisfying and the other as challenging. It was

noted that the individual evaluating the session a challenging, had a particularly difficult time understanding the accounting terminology, requiring frequent explanations by analogy throughout the session. The reasons given as to why a rating of very satisfying was indicated by the type B users, centered around self satisfaction at being able to correctly perform the requested operations. Many remarked upon termination of the evaluation session that once they got started it was easy. For the type B users, the perception of a computer as a complex, hands off machine, to be used only by trained professionals appeared dissolved.

Considering these responses, it seems reasonable to assume the aggregation of the various interface constructs employed, produced an environment conducive for user, task accomplishment and successfully established a master-servant relationship between man and machine respectively.

D. EVALUATION SUMMARY

Due to individual differences, it is extremely difficult, if not impossible, to derive clear-cut classifications which characterize all users, in all circumstances, at all times. Consequently, the categories of type A and B users should be viewed as opposite ends of a continuum. The characteristics and attributes of these extremes are presented in Table 6.

TABLE 6. TYPE A AND B USER CHARACTERISTICS

TYPE A USER CHARACTERISTICS	TYPE B USER CHARACTERISTICS
Highly task oriented. Disregards items not germane to task accomplishment.	Interested and exited by everything. Experiments with various items enroute to task accomplisment.
Each action carefully thought out prior to execution.	Actions more intuitive and impulsive.
Uncomfortable with the new and unfamiliar.	Considers new and unfamiliar as a challenge to be mastered.
Takes error messages personally. Great care taken to avoid repeat of same error.	Error messages viewed as part of learning process.
Views each new task as separate and unrelated to previously completed tasks.	Similarities between new and previously completed tasks quickly identified and used.

The results of the evaluation process are viewed as overall supportive of the assumptions and theories underlying the interface design. User perceptions regarding the program seem consistent with design intent. However, several revelations became apparent during the evaluation process which preclude concluding that the application program, in its present form, can effectively support novice user interaction without some prior training.

In retrospect, it appears the primary, interface development assumption of user familiarity with the requirements of the job, is not the only operative assumption. The fact that the design goal was the development of a system

requiring no user manual or prior training, inherently assumes a user willing to accept the Montessori approach of experience and learning through experimentation and discovery. Task oriented type A users and/or prospective users with neither the time nor inclination for experimentation will essentially render the system useless.

A seemingly minor but serious interface design error lays in the assumption that a user's knowledge of a standard typewriter keyboard could be transferred to the computer's keyboard. It became immediately obvious at the start of the evaluation sessions that the interface contained no provision to inform the user of the requirement to press the return or enter key upon completion of data entry. Although this omission may be easily rectified with additional screen documentation, it serves to illustrate the observation by Gaines and Shaw in that:

> ...it highlights a major pitfall into which we all occasionally fall since the phenomenon of assuming that what we personally know and have experienced is obvious is a common one for all human behaviour. [Ref. 5:p. 30]

Thus it seems imperative that when designing systems for little to no formal user training, extreme and methodical care must be exercised when assessing the validity of assumptions regarding user capabilities.

Although the formal evaluation sessions were completed, visits to AER to perform minor maintenance on the production version of the program provided some additional, unexpected observations. The users classified as type B continued to

show great interest in the application program. They were observed probing the various system capabilities and literally, generating pretenses to interact with the program. Requests were made of the accountant, who was to be the primary user, for meaningful data to input. The system was in constant use. This sudden activity was viewed as significant, considering the computer had been present in the organization for over a year as well as several standard, general application software packages. Further investigation revealed that none of the type A users have used or shown any interest in the computer since the evaluation sessions.

The results of the evaluation sessions coupled with the post-evaluation period observations, seem to support the overall success of the research project and the underlying methodology and assumptions presented in Chapter 3.

V. CONCLUSION: APPLICABILITY OF FINDINGS

The overall success of the research interface is attributed, primarily, to the successful incorporation of theories and ideas relevant to human behavior obtained from sources external to the traditional realm of computer science. The development and use of the interface requirements specification then aided in the consistency of application of the theories and ideas. Additionally, by placing the interface requirements specifications on equal footing with the requirements specifications, a system of potentially complex interfaces was reduced to one which invites and encourages the novice user.

It is realized each application program has its own, unique interface requirements, and the applicability of this particular interface requirements specification to other application programs may be questionable. However, the concept of an interface requirements specifications during the design and development process seems a viable process to produce a system that not only satisfies the user's functional requirements, but meets the unstated, psychological and ergonomic needs of its users.

Since computers have moved from the laboratory into the mainstream of human existence, it not only seems logical but

necessary for design and development personnel to augment their computer related knowledge with more in-depth knowledge of the disciplines concerned with the study of human characteristics and attributes of the user.

APPENDIX A

INTERFACE EVALUATION FORMS

The purpose of this experiment is to evaluate a new computer program. You will be asked to perform a series of operations. Your ability to perform the various operations will be observed and noted.

****** IMPORTANT ******

Please understand, your ability or inability to perform the requested operations IS NOT a reflection on reflection on you, but an indication of the effectiveness or ineffectiveness of the program. Remember, it is the program which is being evaluated, NOT you.

Please try and complete each operation without asking for assistance. However, should you find it impossible to proceed without an answer to your question, do not hesitate to ask. Feel free to experiment or when in doubt, try something you think appropriate. Feel free to voice any comments, positive or negative, during the session. This is NOT a timed experiment. You may proceed at your pace. Take all the time you need to comprehend what is presented on the computer's screen. Finally, NOTHING you may do, short of physical violence, will break, blow-up, or otherwise damage either the computer or the program.

BACKGROUND

This program was developed for the Army Emergency Relief (AER) organization's accountant. For the purpose of this experiment, imagine you are that accountant.

The overall function of AER is to provide no-interest loans to military personnel, primarily army, who have a bonafide need for financial assistance. As the accountant, you are not directly involved in the process of loan application or approval. Your duties commence upon approval of the loan.

Once the loan is approved, you establish an Army Emergency Relief Individual Loan Ledger (DA Form 1108). The DA Form 1108 contains information about the individual and is used to record loan repayments and the outstanding loan balance. In addition to keeping the DA Form 1108's up to date, you are responsible for accurately keeping track of all funds associated with your particular AER organization. You keep track of these funds by means of the AER General Ledger. The General Ledger is composed of various accounts, each with its own account code.

Another of your functions as the accountant is to provide information, upon request, about individual loan accounts, loan accounts in general and the General Ledger to other AER personnel as required for the performance of their duties.

Please let me know when you are ready to begin the computer session. If you have any questions about anything please ask.

COMPUTER PROGRAM OPERATIONS

1. SGT Harris has just given you an approved loan package for you to establish a loan account. The package's content are as follows:

Personal Information:	Terry, A. Johnson 471-23-7391 E-4, Active Duty No previous AER loans. 145 S. Treelawn Ave Rusty Spur, Idaho 75634
Duty Station:	A Company, 7th Infantry, Ft Ord, CA
Loan Information:	Loan Amount: $340.00 Allotment Amount: $ 68.00 Reason for Loan: Initial Rent and Deposit Allotment to Start: March 1987 Allotment to Stop: July 1987

   Seeing that all is in order, you sign check number 634152 and give it to SGT Harris for delivery to Johnson.

   Please establish the loan account.

2. SGT Jones is in the process of taking a loan application and asks you to verify that William Q. Tell, SSN: 423-45-1928, has only had one previous AER loan.

   What is your response?

3. The AER officer is on the intercom in a panic, as Col Evans is on the outside line, wanting to know how many personnel assigned to Ft Ord received loans last month.

   What is your response?

4. Going through the mail, you come across a check for $54.23 from the Chapter 13 Bankruptcy Court Trustee for payment on the loan account of Ohso Broke.

   Please apply the repayment.

5. Alfred Martin, 364-29-5647, has just come in as part of his discharge check-out process and wants to pay off the remainder of his loan. He hands you $40.00, says thanks and keep the change. If there is any money left over after applying the repayment to the outstanding loan balance then you must apply the excess money to either General Ledger Account 2001 (Contributions) if the excess money is $5.00 or less, or to Account 2004 (Over Payments).

   Please process this transaction.

6. Another letter contains a check for $100.00 with a note from an individual who was helped by AER several years ago and now, out of financial difficulty, wants to contribute this $100.00 so others may continue to receive the services of AER.

   Please post this contribution to the General Ledger.

7. Beverly Anderson just stopped in to inform you that she just got married and would like her account to reflect her married name of Pruitte.

   Please make the change.

8. You have just been informed that Daniel Washington, 432-74-1423, was involved in a fatal automobile accident over the weekend. Under these circumstances, AER regulations require you to declare all outstanding loan balances of the deceased uncollectible.

   Please update Washington's account.

9. Looking over the last computer print out of the General Ledger, you notice that there is a mistake in the totals. You have traced this mistake to account code 2006 for FEB 87. Instead of entering -23.67 you entered 23.67.

   Please correct this error.

10. How many loans were given out in DEC 86 and what was their total amount?

APPENDIX B

APPLICATION PROGRAM SOURCE CODE

The following, undocumented, application program source code is written in Borland International, Inc., Turbo Pascal™, version 3.0.

Since the application program was not the object of research, but merely a necessary, temporary tool for the researcher, no documentation was deemed necessary.

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

```
File Name: AER.PAS

{$I GLOBAL.AER}
{$I REGISTER.CPU}
{$I CONVERT.PAS}
{$I FILEOPS.PAS}
{$I SCREENIO.PAS}
{$I LEDGER.PAS}
{$I HARDCOPY.PAS}
{$I AERPROCS.PAS}
{$I OVERLAYS.OVR}

begin  { Main Program }
   PortW[$03D8] := $09;  { Set video blink mode off }
   ClrScr; Esc := False;
   KBSB := KBSB or $20;   { Activate Num_Lock }
   Load_Display_Screens_into_Memory;
   UpDate_Loans; if ESC then Exit;
   ESC := True;
   View_Change_or_Delete; { Load overlay procedure }
   ESC := False;
   repeat
      Fill_Field(3,2,CSDate);
      for I := 0 to 6 do Fill_Field(3,I+3,String_Int(Loan_Totals[I],4));
      Fill_Field(3,10,String_Int(Index_Stats.Next_Name_Ptr,4));
      Fill_Field(3,11,String_Int(Loan_Stats.Prev_Record,4));
      KBSB := KBSB or $20;   {Activate Num_Lock }
      repeat
         PF_Key := True;
         Screen_Input(3,13,13);
         if ESC then   { terminate program }
            begin
               KBSB := KBSB and $DF;  { set Num_Lock OFF }
               Close_Files; Exit
            end;
         if Not(PF_Key) then
            begin
               Selection := Integer_Value(Field_Contents(3,13));
               if Not(Selection in [1..10]) then Buzzer
            end
         else
            begin
               Display_Window(3,Selection + 14);
               I := Key_Depressed;
               Display_Screen := Prepared_Screen;
               if I <> 13 then Selection := 0;
               ESC := False
            end
      until Selection in [1..10];
```

File Name: AER.PAS (cont)

```
      if Selection = 1 then Loan_Entry(1)
      else if Selection = 2 then Loan_Entry(2)
      else if Selection = 3 then
         begin
            repeat
               Screen_Input(3,14,14);
               I := Integer_Value(Field_Contents(3,14));
               if Not ((I in [1..3]) or (ESC)) then Buzzer
            until (I in [1..3]) or (ESC);
            if Not ESC then Record_Payments(I)
         end
      else if Selection = 4 then View_Change_or_Delete
      else if Selection = 5 then Loan_Entry(4)
      else if selection = 6 then Display_General_Stats
      else if Selection = 7 then Display_Financials(1)
      else if Selection = 8 then Loan_Entry(3)
      else if Selection = 9 then Display_Financials(2)
      else if (Selection = 10) and (Printer_OK = 0) then
         begin
            repeat
               Screen_Input(3,12,12);
               I := Integer_Value(Field_Contents(3,12));
               if Not ((I in [1..10]) or (ESC)) then buzzer
            until (I in [1..10]) or (ESC);
            if Not ESC then Seek_Records(I)
         end;
      Prepare_Screen(3);
      Display_Screen := Prepared_Screen;
      Correcting := False;
      ESC := False
   until Selection = 13
end.  { Main Program }
```

```
File Name: GLOBAL.AER

const
   Hi_Lite = $40;     { Input field color = black on red }
   Display_Memory = $B800;  { $B000 for monochrome monitors }
   Index_AER = 'Index.AER';
   Accounts_AER = 'Accounts.AER';
   Loans_AER = 'Loans.AER';
   GrdStats_AER = 'GrdStats.AER';
   LEDGER_FRM = 'Ledger.FRM';
   Valid_Month = ' JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC';

type
   Identification_Record = record
      Hash_Case_Nr_Ptr : integer;          { 2 bytes}
      Hash_Name_Ptr : integer;             { 2 bytes}
      Next_Case_Nr_Ptr : integer;          { 2 bytes}
      Previous_Case_Nr_Ptr: integer;       { 2 bytes}
      Next_Name_Ptr: integer;              { 2 bytes}
      Previous_Name_Ptr: integer;          { 2 bytes}
      SSN : real;                          { 6 bytes}
      Name : string[25];                   {26 bytes}
      Grade_and_Status : byte;             { 1 byte }
      Accounts_Ptr : integer;              { 2 bytes}
   end;  { Identification_Record }         {47 bytes}

   Accounting_Record = record
      Acct_Status : byte;                  { 1 byte }
      Loan_Nr : byte;                      { 1 byte }
      Repay_Method : byte;                 { 1 byte }
      Allot_Info : real;                   { 6 bytes}
      Loan_Info : real;                    { 6 bytes}
      Balance_Info : real;                 { 6 bytes}
      Next_Record : integer;               { 2 bytes}
      Prev_Record : integer;               { 2 bytes}
   end;  { Accounting_Record }             {25 bytes}

   Total_Account = record
      Rec_Loc : integer;                   { 2 bytes}
      Loan_Data : Accounting_Record;       {25 bytes}
   end; {Total_Account}

   Entire_Account = array[1..15] of Total_Account;

   Qty_Amount = record
      Qty : integer;                       { 2 bytes}
      Amt : real;                          { 6 bytes}
   end; { Qty_Amount }                     { 8 bytes}
```

File Name: GLOBAL.AER (cont)

```
   AER_Accounts = record
      Entry_Year : byte; {Last digit of applicable year}      {01 bytes}
      AX000 : array[1..6]  of real;  {Account Totals}         {36 bytes}
      A2000 : array[1..10] of Real;  {Receipts}               {60 bytes}
      A3000 : array[9..16] of Real;  {Disbursements}          {48 bytes}
      A6000 : array[17..21] of real; {Loan Balance Summary}   {30 bytes}
      A2QTY : array[1..5] of integer; {Quantity Totals}       {10 bytes}
      A3QTY : array[10..13] of integer; {Quantity Totals}     {08 bytes}
      A6QTY : array[17..19] of integer;  {Quantity Totals}    {06 bytes}
   end; {AER_Accounts }                                      {199 bytes}

   General_Stats = record
      Year : byte;                                           {001 byte }
      Grade_Stats : array[1..2,1..9] of Qty_Amount;          {144 bytes}
      Loan_Cats : array[1..11] of Qty_Amount;                {088 bytes}
      Duty_Station : array[1..3] of Qty_Amount;              {024 bytes}
   end; { General_Stats }                                    {256 bytes}

   scrnline = array[1..160] of byte;
   Scrnarray = array[1..25] of scrnline;
   Screen_Data = record
      Screen_Image : Scrnarray;
      Field_Posits : ScrnLine;
      Window_Info : ScrnLine
   end; {record Screen_Data}

   String3 = string[3];
   String5 = string[5];
   String9 = string[9];
   String11 = string[11];
   String25 = string[25];
   String40 = string[40];
   String80 = string[80];

var
   Index,Index_Stats : Identification_Record;
   Loan,Loan_Stats : Accounting_Record;

   Index_File : file of Identification_Record;
   Loan_File : file of Accounting_Record;
   Stats_File : file of General_Stats;
   Accounts_File : file of AER_Accounts;

   Selection, CurMon, CurDate, Code, I, J : integer;
   Screen : array[1..6] of Screen_Data absolute $6000:0000;
   Display_Screen : scrnarray absolute Display_Memory:$0000;
   Prepared_Screen : ScrnArray;
   Rec_Pos : array[1..15] of integer;
   Stats_Code : array[0..6] of byte;
```

File Name: GLOBAL.AER (cont)

```
   Loan_Totals : array[0..10] of integer;
   PF_Key, Print_On, Correcting, ESC : boolean;
   KBSB : byte absolute $0000:$0417;
   Grade : String3;
   Scan_Code : byte;
   Status : char;
   Date, CSDate : String9;
   Window_Contents : array[1..6,1..130] of String80 absolute $5000:0000;
```

File Name: REGISTER.CPU

```
type
   CPU_Registers = record
      AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : integer
   end;

var
   Regs : CPU_Registers;

Function Key_Depressed : byte;

begin
   if ESC then Exit;
   Regs.AX := 0;intr($16,Regs); Key_Depressed := lo(Regs.AX);
   if lo(Regs.AX) = 27 then ESC := True else ESC := False;
   if hi(Regs.AX) = 78 then Key_Depressed := 13
end;  { Function Key_Depressed }
```

```
File Name: CONVERT.PAS

Function Integer_Value(Str_Val : String40) : integer;

var
   Temp_Int_Val : integer;

begin
   val(Str_Val,Temp_Int_Val,Code);
   if Code = 0 then Integer_Value := Temp_Int_Val
   else Integer_Value := 0
end;  { Function Integer_Value }

Function SSN_Str(Real_SSN : real) : String11;

var
   Temp_Str : String11;
   S1 : integer;

begin
   Str(Real_SSN:9:0,Temp_Str);
   for S1 := 1 to 9 do
      if Temp_Str[S1] = ' ' then Temp_Str[S1] := '0';
   insert('-',Temp_Str,4); insert('-',Temp_Str,7);
   SSN_Str := Temp_Str
end; { Function SSN_Str }

Procedure Split_Date_and_Money(Date_Money : real;
                               var Date_Out : String9;
                               var Money_Amt : real);

var
   Day, Mon, Year, Int_Date : integer;
   Day_Str, Year_Str : string[2];

begin
   Int_Date := trunc(Date_Money);
   Money_Amt := frac(Date_Money) * 10000;
   Year := Int_Date div 512;Str(80 + Year:2,Year_Str);
   Mon := (Int_Date - 512 * Year) div 32;
   Day := Int_Date - Year * 512 - Mon * 32;
   if Day = 0 then Day_Str := '  '
   else Str(Day:2,Day_Str);
   Date_Out := Day_Str+' '+copy(Valid_Month,4*Mon-2,3)+' '+Year_Str
end;   { Procedure Split_Date_and_Money }
```

File Name: CONVERT.PAS (cont)

```
Function Merge_Date_and_Money(Str_Date:String9; Money_Amt:real) : real;

var
   Mon, Day, Year : integer;

begin
   while length(Str_Date) < 9 do insert('0',Str_Date,1);
   Day := Integer_Value(copy(Str_Date,1,2));
   Mon := ((pos(copy(Str_Date,4,3),Valid_Month) + 2) div 4) * 32;
   year := (Integer_Value(copy(Str_Date,8,2)) - 80) * 512;
   Merge_Date_and_Money := Year + Mon + Day + Money_Amt/10000.0
end; { Function Merge_Date_and_Money }

Procedure Extract_Date_Data(In_Date:String9;
                            var Mon_Nr,Int_Date:integer);

var
   Mon : string[3];

begin
   while length(In_Date) < 9 do insert(' ',In_Date,1);
   Mon := copy(In_Date,length(In_Date)-5,length(In_Date)-3);
   Mon_Nr := (pos(Mon,Valid_Month) + 2) div 4;
   Int_Date := round(Merge_Date_and_Money(In_Date,0.0))
end;  { Procedure Extract_Date_Data }

Function Encode_Grade_and_Status(Grd : String3; Stat : char) : byte;

var
   Temp_Code : byte;

begin
   Temp_Code := ord(Grd[3])-48;
   if Grd[1] = 'E' then Temp_Code := $20 or Temp_Code
   else if Grd[1] = 'W' then Temp_Code := $40 or Temp_Code
   else Temp_Code := $80 or Temp_Code;
   if Stat = 'R' then Temp_Code := $10 or Temp_Code;
   Encode_Grade_and_Status := Temp_Code
end; { Function Encode_Grade_and_Status }

Procedure Decode_Grade_and_Status(Code_Val : byte; var Grd : String3;
                                  var Stat : char);
begin
   if Code_Val and $20 = $20 then Grd := 'E-'
   else if Code_Val and $80 = $80 then Grd := '0-' else Grd := 'W-';
   if Code_Val and $10 = $10 then Stat := 'R'
   else Stat := 'A';
   if (Code_Val and $0F) = 0 then Grd := 'UNK'
   else Grd := Grd + chr((Code_Val and $0F) + 48)
end;  { Procedure Decode_Grade_and_Status }
```

File Name: CONVERT.PAS (cont)

```
Procedure Hash(Raw_Value : String25; var Hash_Value : integer;
               var SSN_Hash : boolean);

type
   Ordering_Set = set of char;

var
   Sub_Total, H1, H2, H3 : integer;
   Soc_Sec_Nr : real;

begin
   while pos(' ',Raw_Value) <> 0 do
      delete(Raw_Value,pos(' ',Raw_Value),1);
   while     pos('-',Raw_Value)     <>     0     do
delete(Raw_Value,pos('-',Raw_Value),1);
   Val(Raw_Value,Soc_Sec_Nr,Code);
   if Code = 0 then
      begin
         Hash_Value := (round(frac(Soc_Sec_Nr/10000)*10000) mod 5000)+1;
         SSN_Hash := True; Exit
      end
   else
      begin
         Sub_Total := 0;
         if length(Raw_Value) > 7 then H2 := 7
         else H2 := length(Raw_Value);
         H3 := 102;
         for H1 := 1 to H2 do
            begin
               Sub_Total:=Sub_Total+H3*(Ord(upcase(Raw_Value[H1]))-65);
               H3 := H3 div 2
            end;
         Hash_Value := abs(Sub_Total); SSN_Hash := False
      end
end;   { Procedure Hash }

Function Real_Value(Str_Val : String40) : real;

var
   Temp_Real_Val : real;

begin
   if (Str_Val[4] = '-') and (Str_Val[7] = '-') then
      begin
         delete(Str_Val,4,1); delete(Str_Val,6,1)
      end;
   val(Str_Val,Temp_Real_Val,Code);
   Real_Value := Temp_Real_Val
end;  { Function Real_Value }
```

File Name: CONVERT.PAS (cont)

```
Function String_Real(Real_In : real;String_Size : integer):String11;

var
   Temp_Result : String11;

begin
   Str(Real_In:11:2,Temp_Result);
   if length(Temp_Result) > String_Size then
      repeat
         delete(Temp_Result,1,1)
      until length(Temp_Result) = String_Size;
   String_Real := Temp_Result
end;  { Function String_Real }

Function String_Int(Integer_In, String_Size : integer) : String5;

var
   Temp_Result : String5;

begin
   Str(Integer_In:5,Temp_Result);
   if length(Temp_Result) > String_Size then
      repeat
         delete(Temp_Result,1,1)
      until length(Temp_Result) = String_Size;
   String_Int := Temp_Result
end;  { Function String_Int }

Function Date_Difference(Date1,Date2 : String9) : integer;

var
   Date_Code1, Date_Code2, Mon1, Mon2, Year_Correct : integer;

begin
   Extract_Date_Data(Date1,Mon1,Date_Code1);
   Extract_Date_Data(Date2,Mon2,Date_Code2);
   Year_Correct := abs((Date_Code1 div 512) - (Date_Code2 div 512))*128;
   Date_Difference := (Date_Code1 - Date_Code2 - Year_Correct) div 32
end; { Function Date_Difference }
```

File Name: CONVERT.PAS (cont)

```
Function New_Status(Act : Char; Loan_Rec : Accounting_Record) : byte;

var
   ADiff, PDiff, Inc : integer;   ADate, PDate : string[9];
   T_Real1, T_Real2 : real;

begin
   if Act = 'D' then Inc := -1 else Inc := 1;
   New_Status := Loan_Rec.Acct_Status;
   with Loan_Rec do
      if Acct_Status in [1,3,5,6] then
         Loan_Totals[Acct_Status] := Loan_Totals[Acct_Status] + Inc
      else
         begin
            Split_Date_and_Money(Allot_Info,ADate,T_Real1);
            Split_Date_and_Money(Balance_Info,PDate,T_Real2);
            ADiff := Date_Difference(CSDate,ADate);
            PDiff := Date_Difference(CSDate,PDate);
            if ADiff > 4 then ADiff := 4;  if PDiff > 4 then PDiff := 4;
            if (Acct_Status = 4) and (PDiff > 0) then New_Status := $FF
            else if Acct_Status = 4 then
               Loan_Totals[4] := Loan_Totals[4] + Inc
            else if (Acct_Status=0) and (Adiff > 0) and (PDiff > 0) then
               begin
                  New_Status := 2;
                  Loan_Totals[2] := Loan_Totals[2] + Inc;
                  Loan_Totals[7] := Loan_Totals[7] + Inc
               end
            else if Acct_Status = 0 then
               Loan_Totals[0] := Loan_Totals[0] + Inc
            else
               begin
                  if (Pdiff < 1) or (ADiff < 1) then
                     begin
                        New_Status := 0;
                        Loan_Totals[0] := Loan_Totals[0] + Inc
                     end
                  else
                     begin
                        Loan_Totals[2] := Loan_Totals[2] + Inc;
                        if PDiff > Adiff then
                           Loan_Totals[6+Adiff] := Loan_Totals[6+Adiff]
                                                            + Inc
                        else
                           Loan_Totals[6+Pdiff] := Loan_Totals[6+Pdiff]
                                                            + Inc
                     end
               end
         end
end;  { Function New_Status }
```

```
File Name: FILEOPS.PAS

Function Strings_Equal(Input_String,Record_String : String25) : boolean;

var
   S1, StrLen : integer;
   Str1, Str2 : string[25];

begin
   Str1 := '';Str2 := '';
   if length(Input_String) > length(Record_String) then
      StrLen := length(Record_String)
   else StrLen := length(Input_String);
   for S1 := 1 to StrLen do
      begin
         if Input_String[S1] <> chr(32) then
            Str1 := Str1 + upcase(Input_String[S1]);
         if Record_String[S1] <> chr(32) then
            Str2 := Str2 + upcase(Record_String[S1])
      end;
   if Str1 = Str2 then Strings_Equal := True
   else Strings_Equal := False
end; { Function Strings_Equal }

Procedure Get_Index_Record(Hash_Object:String25; Var Rec_Ptr:integer);

var
   Hash_Val : integer;
   Case_is_the_Key,Record_Located,No_Record : boolean;

begin
   Hash(Hash_Object,Hash_Val,Case_is_the_Key);
   seek(Index_File,Hash_Val);read(Index_File,Index);
   if Case_is_the_Key then
      seek(Index_File,Index.Hash_Case_Nr_Ptr)
   else if Index.Hash_Name_Ptr = 0 then
      begin
         Rec_Ptr := 0; Exit
      end
      else seek(Index_File,Index.Hash_Name_Ptr);
   No_Record := false; Record_Located := False;
   repeat
      read(Index_File,Index);
      if Case_is_the_Key then
         begin
            if SSN_Str(Index.SSN) = Hash_Object then
               Record_Located := True
            else if Index.Next_Case_Nr_Ptr = 0 then
               No_Record := True
            else seek(Index_File,Index.Next_Case_Nr_Ptr)
         end
```

File Name: FILEOPS.PAS (cont)

```
      else
         begin
            if Strings_Equal(Hash_Object,Index.Name) then
               Record_Located := True
            else if Index.Next_Name_Ptr = 0 then No_Record := True
            else seek(Index_File,Index.Next_Name_Ptr)
         end
   until (No_Record) or (Record_Located);
   if Record_Located then Rec_Ptr := FilePos(Index_File) - 1
   else Rec_Ptr := 0;
end;  { Procedure Get_Index_Record }

Procedure Write_Index_Record;

var
   Temp_Index : Identification_Record;
   Temp_Loan : Accounting_Record;
   Record_Posit, Case_Hash_Val, Name_Hash_Val : integer;
   SSN_String : String11;
   Dummy : boolean;

begin
   SSN_String := SSN_Str(Index.SSN); Temp_Index := Index;
   Get_Index_Record(SSN_String, Record_Posit); {check if record exists}
   if Record_Posit <> 0 then
      begin
         Index.Grade_and_Status := Temp_Index.Grade_and_Status;
         seek(Index_File,Record_Posit); write(Index_File,Index);
         seek(Loan_File,Index.Accounts_Ptr);
         read(Loan_File,Temp_Loan);
         if Temp_Loan.Next_Record <> 0 then
            repeat
               seek(Loan_File,Temp_Loan.Next_Record);
               read(Loan_File,Temp_Loan)
            until Temp_Loan.Next_Record = 0;
         Loan.Prev_Record := FilePos(Loan_File) - 1;
         Temp_Loan.Next_record := Loan_Stats.Next_Record;
         seek(Loan_File,Loan.Prev_Record);
         write(Loan_File,Temp_Loan)
      end
   else                              {record does not exist}
      begin
         Index := Temp_Index; Hash(SSN_String,Case_Hash_Val,Dummy);
         seek(Index_File,Case_Hash_Val); read(Index_File,Temp_Index);
         Index.Previous_Case_Nr_Ptr := Case_Hash_Val;
         Index.Next_Case_Nr_Ptr := Temp_Index.Hash_Case_Nr_Ptr;
         Temp_Index.Hash_Case_Nr_Ptr := Index_Stats.Accounts_Ptr;
         seek(Index_File,Case_Hash_Val); write(Index_File,Temp_Index);
         if Index.Next_Case_Nr_Ptr <> 0 then
```

File Name: FILEOPS.PAS (cont)

```
            begin
               seek(Index_File,Index.Next_Case_Nr_Ptr);
               read(Index_File,Temp_Index);
               Temp_Index.Previous_Case_Nr_Ptr :=
                                              Index_Stats.Accounts_Ptr;
               seek(Index_File,Index.Next_Case_Nr_Ptr);
               write(Index_File,Temp_Index)
            end;
         Index.Accounts_Ptr := Loan_Stats.Next_Record;
         Hash(Index.Name,Name_Hash_Val,Dummy);
         seek(Index_File,Name_Hash_Val); read(Index_File,Temp_Index);
         Index.Previous_Name_Ptr := Name_Hash_Val;
         Index.Next_Name_Ptr := Temp_Index.Hash_Name_Ptr;
         Temp_Index.Hash_Name_Ptr := Index_Stats.Accounts_Ptr;
         seek(Index_File,Name_Hash_Val); write(Index_File,Temp_Index);
         if Index.Next_Name_Ptr <> 0 then
            begin
               seek(Index_File,Index.Next_Name_Ptr);
               read(Index_File,Temp_Index);
               Temp_Index.Previous_Name_Ptr := Index_Stats.Accounts_Ptr;
               seek(Index_File,Index.Next_Name_Ptr);
               write(Index_File,Temp_Index)
            end;
         seek(Index_File,Index_Stats.Accounts_Ptr);
         read(Index_File,Temp_Index);
         Index.Hash_Case_Nr_Ptr := Temp_Index.Hash_Case_Nr_Ptr;
         Index.Hash_Name_Ptr := Temp_Index.Hash_Name_Ptr;
         seek(Index_File,Index_Stats.Accounts_Ptr);
         write(Index_File,Index);
         seek(Loan_File,Loan_Stats.Next_Record);
         read(Loan_File,Temp_Loan);
         Loan.Prev_Record := - Index_Stats.Accounts_Ptr;
         Index_Stats.Accounts_Ptr := Temp_Index.Accounts_Ptr;
         Index_Stats.Previous_Case_Nr_Ptr := Index_Stats.Accounts_Ptr;
         Index_Stats.Next_Name_Ptr := Index_Stats.Next_Name_Ptr + 1
      end;
   seek(Loan_File,Loan_Stats.Next_Record);
   read(Loan_File,Temp_Loan);
   seek(Loan_File,Loan_Stats.Next_Record);
   Loan.Next_Record := 0;
   write(Loan_File,Loan);
   Loan_Stats.Next_Record := Temp_Loan.Next_Record;
   Loan_Stats.Prev_Record := Loan_Stats.Prev_Record + 1;
   seek(Loan_File,0); write(Loan_File,Loan_Stats);
   seek(Index_File,0); write(Index_File,Index_Stats);
   Flush(Index_File); Flush(Loan_File)
end;  { Procedure Write_Index_Record }
```

File Name: FILEOPS.PAS (cont)

```
Procedure Delete_Loan(Loan_Record_Ptr : integer;
                      var Next_Loan_Record : integer);
var
   Temp_Loan : Accounting_Record;

begin
   seek(Loan_File,Loan_Record_Ptr);
   read(Loan_File,Loan);
   Loan.Acct_Status := New_Status('D',Loan);
   Next_Loan_Record := Loan.Next_Record;
   if Loan.Next_Record <> 0 then
      begin
         seek(Loan_File,Loan.Next_Record);
         read(Loan_File,Temp_Loan);
         Temp_Loan.Prev_Record := Loan.Prev_Record;
         seek(Loan_File,Loan.Next_Record);
         write(Loan_File,Temp_Loan)
      end;
   if Loan.Prev_Record < 0 then
      begin
         seek(Index_File,abs(Loan.Prev_Record)); read(Index_File,Index);
         Index.Accounts_Ptr := Loan.Next_Record;
         seek(Index_File,abs(Loan.Prev_Record)); write(Index_File,Index)
      end
   else
      begin
         seek(Loan_File,Loan.Prev_Record);
         read(Loan_File,Temp_Loan);
         Temp_Loan.Next_Record := Loan.Next_Record;
         seek(Loan_File,Loan.Prev_Record);
         write(Loan_File,Temp_Loan)
      end;
   FillChar(Loan,25,0);
   Loan.Acct_Status := $FF;
   Loan.Next_Record := Loan_Stats.Next_Record;
   Loan_Stats.Prev_Record := Loan_Stats.Prev_Record - 1;
   Loan_Stats.Next_Record := Loan_Record_Ptr;
   seek(Loan_File,Loan_Record_Ptr);
   write(Loan_File,Loan);
   seek(Loan_File,0); write(Loan_File,Loan_Stats);
   Flush(Loan_File)
end;  { Procedure Delete Loan }
```

File Name: FILEOPS.PAS (cont)

```
Procedure Delete_Account(Index_Entry_Ptr : integer);

var
   Temp_Index : Identification_Record;   Temp_Loan : Accounting_Record;
   Next_Ptr, Record_Ptr, Case_Hash_Val, Name_Hash_Val : integer;
   SSN_String : String25; Dummy : boolean;

begin
   Str(Index.SSN:9:0,SSN_String); Hash(SSN_String,Case_Hash_Val,Dummy);
   Hash(Index.Name,Name_Hash_Val,Dummy);
   Next_Ptr := Index.Accounts_Ptr;
   repeat Delete_Loan(Next_Ptr,Next_Ptr) until Next_Ptr = 0;
   Temp_Index := Index; Temp_Index.Name := 'EMPTY';
   Temp_Index.Accounts_Ptr := Index_Stats.Accounts_Ptr;
   Index_Stats.Accounts_Ptr := Index_Entry_Ptr;
   Index_Stats.Next_Name_Ptr := Index_Stats.Next_Name_Ptr - 1;
   seek(Index_File,Index_Entry_Ptr); write(Index_File,Temp_Index);
   seek(Index_File,Index.Previous_Case_Nr_Ptr);
   read(Index_File,Temp_Index);
   if Index.Previous_Case_Nr_Ptr = Case_Hash_Val then
      Temp_Index.Hash_Case_Nr_Ptr := Index.Next_Case_Nr_Ptr
   else Temp_Index.Next_Case_Nr_Ptr := Index.Next_Case_Nr_Ptr;
   seek(Index_File,Index.Previous_Case_Nr_Ptr);
   write(Index_File,Temp_Index);
   if Index.Next_Case_Nr_Ptr <> 0 then
      begin
         seek(Index_File,Index.Next_Case_Nr_Ptr);
         read(Index_File,Temp_Index);
         Temp_Index.Previous_Case_Nr_Ptr := Index.Previous_Case_Nr_Ptr;
         seek(Index_File,Index.Next_Case_Nr_Ptr);
         write(Index_File,Temp_Index)
      end;
   seek(Index_File,Index.Previous_Name_Ptr);
   read(Index_File,Temp_Index);
   if Index.Previous_Name_Ptr = Name_Hash_Val then
      Temp_Index.Hash_Name_Ptr := Index.Next_Name_Ptr
   else Temp_Index.Next_Name_Ptr := Index.Next_Name_Ptr;
   seek(Index_File,Index.Previous_Name_Ptr);
   write(Index_File,Temp_Index);
   if Index.Next_Name_Ptr <> 0 then
      begin
         seek(Index_File,Index.Next_Name_Ptr);
         read(Index_File,Temp_Index);
         Temp_Index.Previous_Name_Ptr := Index.Previous_Name_Ptr;
         seek(Index_File,Index.Next_Name_Ptr);
         write(Index_File,Temp_Index)
      end;
   seek(Index_File,0); write(Index_File,Index_Stats);
   Flush(Index_File)
end;  {procedure Delete_Account }
```

```
File Name: SCREENIO.PAS

Procedure Buzzer;      { Produces audio error signal }
begin
   sound(800); delay(100); nosound
end;  { Procedure Buzzer }

Procedure Display_Window(Screen_Nr : integer; Window_Nr : byte);

var
   X,Y,Z, Offset, Window_Ptr : integer;
   Window_Lines : byte;
   DisplayString : String80;

begin
   Window_Ptr := Window_Nr*4 - 3;
   with Screen[Screen_Nr] do
      begin
         Window_Lines := 0;
         Z := Window_Info[Window_Ptr + 3];
         X := Window_Info[Window_Ptr]; Y := Window_Info[Window_Ptr + 1];
         while Window_Lines < Window_Info[Window_Ptr + 2] do
            begin
               DisplayString := Window_Contents[Screen_Nr,Z];
               Offset := (Y - 1)*160 + 2*(X - 1);
               inline(

                  $50/$51/$57/$56/$06/$9C/  {PUSH AX,CX,DI,SI,ES,Flags}
                  $2E/$B8/Display_Memory/   {CS:MOV AX,[Display_Memory]}
                  $50/                      {PUSH AX}
                  $07/                      {POP ES}
                  $8B/$BE/Offset/           {MOV DI,[BP+Offset]}
                  $8D/$B6/DisplayString/    {LEA SI,[BP+DisplayString]}
                  $31/$C9/                  {XOR CX,CX}
                  $36/$8A/$0C/              {SS:MOV CL,[SI]}
                  $46/                      {INC SI}
                  $FC/                      {CLD}
                  $36/$A4/                {L1: SS:MOVSB}
                  $E2/$FC/                  {LOOP L1}
                  $9D/$07/$5E/$5F/$59/$58); {POP Flags,ES,SI,DI,CX,AX}

               Z := Z + 1; Y := Y + 1; Window_Lines := Window_Lines + 1
            end
      end
end;   { Procedure Display_Window }
```

```
File Name: SCREENIO.PAS (cont)

Procedure Prepare_Screen(Screen_Number : integer);

var
   PI, PJ : integer;

begin
   Prepared_Screen := Screen[Screen_Number].Screen_Image;
   PJ := 1;
   with Screen[Screen_Number] do
   repeat
      for PI := 0 to ($7F and Field_Posits[PJ+2]) - 1 do
      if not odd(PI) then
      Prepared_Screen[Field_Posits[PJ+1],Field_Posits[PJ] + PI] := $FF;
      PJ := PJ + 3
   until Field_Posits[PJ] = 0
end;  { Procedure Prepare_Screen  }

Procedure Display_Input_Field(Screen_Num,Fld_Num : integer;
                              var End_Of_Field : integer);

var
   D1, D2, Ypos, Field_End : integer;

begin
   Fld_Num := Fld_Num*3 - 2;
   with Screen[Screen_Num] do
      begin
         D2 := -3;
         gotoXY((Field_Posits[Fld_Num]+1) shr 1,
                                              Field_Posits[Fld_Num+1]);
         repeat
            D2 := D2 + 3; Ypos:=Field_Posits[D2+Fld_Num+1];
            End_Of_Field := Field_Posits[D2+Fld_Num] +
                              ($7F and Field_Posits[D2+Fld_Num+2]) - 1;
            for D1 := Field_Posits[D2+Fld_Num] to End_Of_Field do
               if Odd(D1) then
                  begin
                     if Screen_Image[Ypos,D1] in [32,45] then
                        Display_Screen[Ypos,D1] := Screen_Image[Ypos,D1]
                     else
                        begin
                           Display_Screen[YPos,D1] := $FF;
                           Display_Screen[YPos,D1+1] := Hi_Lite
                        end
                  end
         until Field_Posits[D2+Fld_Num+2] < 127
      end
end;  { Procedure Display_Input_Field }
```

File Name: SCREENIO.PAS (cont)

```
Procedure Screen_Input(Display_Nr:byte; Start_Field, End_Field:integer);

var
   OrigX, OrigY, X_Disp, Y_Disp, Field_Nr, Field_End, Dec_Pt : integer;
   InType : byte;
   Mon : string[4];

function Input_Error : boolean;

var
   InChar : byte;

begin
   Input_Error := True; InChar := lo(Regs.AX);
   if (InType in [65..90]) and (Inchar = 13) and (X_Disp < Field_End+2)
      then Exit;
   if (X_Disp = Field_End + 2) and (InChar <> 13) then Exit;
   if (InType = 36) and (X_Disp = OrigX) and (InChar = 13) then Exit;
   if (InType = 36) and (Not(Inchar in [13,45,46,48..57])) then Exit
   else if  (InType in [78,110]) and (Not(InChar in [13,48..57])) then
      Exit
   else if (InType = 99) then
      begin
         if (Display_Screen[Y_Disp,X_Disp-2] = 54) and
                                    (Not(InChar in [73,82])) then Exit
         else if (Display_Screen[Y_Disp,X_Disp-2] <> 54) and
                                              (InChar <> 13) then Exit
      end
   else if (InType = 85) and (Not(InChar in [13,48..57,65..90])) then
      Exit
   else if (InType = 89) and (Not(InChar in [13,56,57])) then Exit
   else if (InType = 68) and
       (Not(InChar in [48..57,65..71,74,76,77..80,82..86,89])) then Exit
   else if  (InType = 77) and
       (Not(InChar in [65..71,74,76,77..80,82..86,89])) then Exit
   else if (InType = 71) and (Not(InChar in [69,79,87])) then Exit
   else if (InType = 83) and (Not(InChar in [65,82] )) then Exit
   else if (InType = 90) and (Not(InChar in [69,79,82,87])) then Exit
   else if (InType = 82) and (Not(InChar in [65,80])) then Exit
   else if not(Inchar in [13,32..126]) then Exit;
   if (InType in [68,77]) and (Not(InChar in [48..57])) and
                       (Pos(Mon + chr(InChar),Valid_Month) = 0) then Exit
   else Input_Error := False;
end;  { internal function Input_Error }
```

File Name: SCREENIO.PAS (cont)

```
procedure Rub_Out;

begin
   if X_Disp = OrigX then Buzzer
   else with Screen[Display_Nr] do
      begin
         X_Disp := X_Disp - 2;
         if Screen_Image[Y_Disp,X_Disp] in [32,45] then
            X_Disp := X_Disp - 2;
         if Display_Screen[Y_Disp,X_Disp] = 46 then Dec_Pt := 0;
         if Screen_Image[Y_Disp,X_Disp] = 77 then
            delete(Mon,length(Mon),1);
         Display_Screen[Y_Disp,X_Disp] := $FF;
         gotoXY((X_Disp+1) div 2,Y_Disp)
      end
end;  { internal procedure Rub_Out }

procedure Display_Input(InChar : integer);

begin
   if (X_Disp >= Field_End + 2) or ((X_Disp = OrigX) and
                                                (InChar = 13)) then
      begin
         Buzzer; Exit
      end;
   with Screen[Display_Nr] do
      begin
         if InType = 36 then
            begin
               if ((InChar = 45) and (X_Disp <> OrigX)) or
                  ((InChar = 46) and (Dec_Pt <> 0)) or
                  ((X_Disp = Dec_Pt + 6) and (Dec_Pt <> 0)) then
                     begin
                        Buzzer; Exit
                     end
               else
                  if InChar = 46 then Dec_Pt := X_Disp
               else
                  if (X_Disp = Field_End - 6) and (Dec_Pt = 0) then
                     begin
                        Dec_Pt := X_Disp + 2;
                        Display_Screen[Y_Disp,X_Disp] := InChar;
                        Display_Screen[Y_Disp,X_Disp+2] := 46;
                        X_Disp := X_Disp + 4;
                        gotoXY((X_Disp+1) div 2,Y_Disp); Exit
                     end
            end
```

File Name: SCREENIO.PAS (cont)

```
         else if InType = 68 then
            begin
               if not (InChar in [48..57]) then
                  begin
                     if X_Disp = OrigX then
                        begin
                           Display_Screen[Y_Disp,OrigX] := $20;
                           Display_Screen[Y_Disp,OrigX+2] := $20
                        end
                     else if X_Disp = OrigX + 2 then
                        begin
                           Display_Screen[Y_Disp,X_Disp] :=
                                          Display_Screen[Y_Disp,OrigX];
                           Display_Screen[Y_Disp,OrigX] := $30
                        end;
                     X_Disp := OrigX + 6
                  end
               else if ((Display_Screen[Y_Disp,OrigX] = 51) and
                     (Not(Inchar in [48,49]))) or
                     (Display_Screen[Y_Disp,OrigX] in [52..57]) then
                  begin
                     Buzzer;Exit
                  end
            end;
         if Screen_Image[Y_Disp,X_Disp] = 77 then
            Mon := Mon + chr(InChar);
         Display_Screen[Y_Disp,X_Disp] := InChar; X_Disp := X_Disp + 2;
         if Screen_Image[Y_Disp,X_Disp] in [32,45] then
            X_Disp := X_Disp + 2;
         if X_Disp < Field_End + 2 then gotoXY((X_Disp+1) div 2,Y_Disp)
   end
end;  { internal procedure Display_Input }

procedure Clear_Hi_Lite;

var
   C1, C2, C3 : integer;

begin
   if (X_Disp = OrigX) and
         (Screen[Display_Nr].Field_Posits[3*Field_Nr] > 127) then
      begin
         repeat
            Field_Nr := Field_Nr + 1
         until Screen[Display_Nr].Field_Posits[3*Field_Nr] < 128;
         Exit
      end;
```

File Name: SCREENIO.PAS (cont)

```
   if Screen[Display_Nr].Screen_Image[OrigY,OrigX] = 36 then
      with Screen[Display_Nr] do
      begin
         if Dec_Pt = 0 then
            begin
               Display_Screen[OrigY,X_Disp] := 46;
               Dec_Pt := X_Disp
            end;
         C1 := Dec_Pt + 4;
         C2 := Field_End;
         for C3 := OrigX to Field_End do
            if Odd(C3) then Prepared_Screen[OrigY,C3] := $FF;
         for C3 := C1 downto OrigX do
            if Odd(C3) then
               begin
                  if Display_Screen[OrigY,C3] in [45,46,48..57] then
                     Prepared_Screen[OrigY,C2]:=Display_Screen[OrigY,C3]
                  else Prepared_Screen[OrigY,C2] := 48;
                  C2 := C2 - 2
               end
      end
   else
      begin
         for C2 := OrigX to Field_End do
            if Odd(C2) then
               Prepared_Screen[OrigY,C2] := Display_Screen[OrigY,C2]
      end
end;  {internal procedure Clear_Hi_Lite }

begin       { procedure Screen_Input }
   if ESC then Exit;
   Field_Nr := Start_Field;
   repeat
      Prepared_Screen := Display_Screen;
      if Field_Nr > End_Field then Exit;
      with Screen[Display_Nr] do
         if (Field_Posits[160] = 1) and (Window_Info[Field_Nr*4-3] <> 0)
             and (Field_Nr <= 40) and (Not(Correcting)) then
            Display_Window(Display_Nr,Field_Nr);
      with Screen[Display_Nr] do
         begin
            X_Disp := Field_Posits[Field_Nr*3-2];
            OrigX := X_Disp;
            Y_Disp := Field_Posits[Field_Nr*3-1];
            OrigY := Y_Disp
         end;
      Dec_Pt := 0; Mon := ' ';
      Display_Input_Field(Display_Nr,Field_Nr,Field_End);
```

File Name: SCREENIO.PAS (cont)

```
      repeat
         Regs.AX:=$0000; intr($16,Regs);
         if (PF_Key) and (hi(Regs.AX) in [59..68]) then
            begin
               Selection := hi(Regs.AX) - 58;Exit
            end
         else PF_Key := False;
         if (hi(Regs.AX) in [72,75,77,80]) and (Correcting) then
            begin
               Scan_Code := hi(Regs.AX); Exit
            end;
         with Screen[Display_Nr] do
            InType := Screen_Image[Y_Disp,X_Disp];
         if hi(Regs.AX) = 78 then Regs.AX := 13;
         if InType in [68,71,77,82,83,85,90,99,117] then
            Regs.AX := ord(upcase(chr(lo(Regs.AX))));
         if lo(Regs.AX) = 27 then ESC := True
         else if lo(Regs.AX) = 8 then Rub_Out
         else if Input_Error then Buzzer
         else if lo(Regs.AX) <> 13 then Display_Input(lo(Regs.AX))
      until ((lo(Regs.AX) = 13) and (not (Input_Error))) or (ESC);
      if ESC then Exit;
      Clear_Hi_Lite;
      Display_Screen := Prepared_Screen;
      Field_Nr := Field_Nr +1
   until Screen[Display_Nr].Field_Posits[Field_Nr*3-2] = 0;
end;  { Procedure Screen_Input  }

Function Field_Contents(Screen_Number,Field_Nr : integer) : String80;

var
   R1, End_Of_Field, X_Disp, Y_Disp : Integer;
   Input_String : String80;

begin
   if ESC then Exit;
   Input_String := '';
   with Screen[Screen_Number] do
      begin
         X_Disp := Field_Posits[3*Field_Nr - 2];
         Y_Disp := Field_Posits[3*Field_Nr - 1];
         End_Of_Field := X_Disp + ($7F and Field_Posits[3*Field_Nr])-1;
         for R1 := X_Disp to End_Of_Field do
            if (Odd(R1)) and (Display_Screen[Y_Disp,R1] <> $FF) then
               Input_String:=Input_String+chr(Display_Screen[Y_Disp,R1])
      end;
   Field_Contents := Input_String
end;  { Function Field_Contents }
```

File Name: SCREENIO.PAS (cont)

```
Procedure Fill_Field(Display_Nr,Field_Nr:byte; Display_String:String40);

var
   F1,X_Coord : integer;

begin
   if ESC then Exit;
   with Screen[Display_Nr] do
      begin
         F1 := Field_Nr; X_Coord := (Field_Posits[3*F1-2] + 1) shr 1;
         gotoXY(X_Coord,Field_Posits[3*F1 - 1]); write(Display_String)
      end
end; { Procedure Fill_Field }
```

File Name: LEDGER.PAS

```
Procedure Stats_Record_IO(Action : char; LMon : integer;
                                  var Work_Stats : General_Stats);

begin
   if LMon = 0 then
      begin
         seek(Stats_File,12); read(Stats_File,Work_Stats); Exit
      end;
   Seek(Stats_File,LMon mod 12);
   if Action in ['R'] then
      begin
         read(Stats_File,Work_Stats);
         if (lo(CurDate div 512) > Work_Stats.Year) and
                                                (LMon = Curmon) then
            begin
               if LMon = 1 then
                  begin
                     seek(Stats_File,12);
                     write(Stats_File,Work_Stats)
                  end;
               FillChar(Work_Stats,257,0);
               Work_Stats.Year := CurDate div 512;
               Seek(Stats_File,LMon mod 12);
               write(Stats_File,Work_Stats);
            end
      end
   else
      begin
         Seek(Stats_File,LMon mod 12); write(Stats_File,Work_Stats);
         Flush(Stats_File)
      end
end;  { Procedure Stats_Record_IO }
```

File Name: LEDGER.PAS (cont)

```
Procedure Record_General_Stats(Rec_Mon : integer);

var
   Loan_Amt : real;
   CatNDX,R1 : integer;
   LCat : string[5];
   Lgrd : string[3];
   Dusta : string[34];
   Stats_Rec : General_Stats;

begin
   Stats_Record_IO('R',Rec_Mon,Stats_Rec);
   Loan_Amt := Real_Value(Field_Contents(1,20));
   LGrd := Field_Contents(1,2); DuSta := Field_Contents(1,8);
   for R1 := 1 to length(DuSta) do DuSta[R1] := upcase(DuSta[R1]);
   R1 := Integer_Value(copy(Lgrd,3,1));
   with Stats_Rec do
      begin
         if Field_Contents(1,3) = 'R' then
            begin
               Grade_Stats[2,9].Qty := Grade_Stats[2,9].Qty + 1;
               Grade_Stats[2,9].Amt := Grade_Stats[2,9].Amt + Loan_Amt
            end
         else if (Lgrd[1] = 'E') and (R1 <> 0) then
            begin
               Grade_Stats[1,R1].Qty := Grade_Stats[1,R1].Qty + 1;
               Grade_Stats[1,R1].Amt := Grade_Stats[1,R1].Amt + Loan_Amt
            end
         else if (Lgrd[1] = 'W') and (R1 in [1..4]) then
            begin
               Grade_Stats[2,R1].Qty := Grade_Stats[2,R1].Qty + 1;
               Grade_Stats[2,R1].Amt := Grade_Stats[2,R1].Amt + Loan_Amt
            end
         else if R1 in [1..4] then
            begin
               Grade_Stats[2,R1+4].Qty := Grade_Stats[2,R1+4].Qty + 1;
               Grade_Stats[2,R1+4].Amt:=
                                       Grade_Stats[2,R1+4].Amt+Loan_Amt
            end;
         if (pos('ORD',DuSta) <> 0) or (pos('FOCA',DuSta) <> 0) then
            R1 := 1
         else if (pos('DLI',DuSta) <> 0) or (pos('POM',DuSta) <> 0) then
            R1 := 2
         else R1 := 3;
            Duty_Station[R1].Qty := Duty_Station[R1].Qty + 1;
            Duty_Station[R1].Amt := Duty_Station[R1].Amt + Loan_Amt;
         LCat := Field_Contents(1,19);
         CatNDX := Integer_Value(copy(LCat,3,2));
```

File Name: LEDGER.PAS (cont)

```
         if CatNDX in [1..10] then
            begin
               if (CatNDX in [7..10]) or (LCat[5] = 'R') then
                  CatNDX := CatNDX + 1;
               Loan_Cats[CatNDX].Qty := Loan_Cats[CatNDX].Qty + 1;
               Loan_Cats[CatNDX].Amt := Loan_Cats[CatNDX].Amt + Loan_Amt
            end
      end;
   Stats_Record_IO('W',Rec_Mon,Stats_Rec)
end;  { Procedure Record_General_Stats }

Procedure Ledger_Record_IO(Action : char; LMon : integer;
                                   var Work_Account : AER_Accounts);

var
   Prev_Month : AER_Accounts;
   NDX, R1, Ledger_Month : integer;
   A1, A6 : real;

begin
   NDX := LMon;
   if LMon = 0 then
      begin
         seek(Accounts_File,12);
         read(Accounts_File,Work_Account);
         Exit
      end;
   Seek(Accounts_File,LMon mod 12);
   if Action = 'R' then
      begin
         read(Accounts_File,Work_Account);
         if (lo(CurDate div 512) > Work_Account.Entry_Year) and
            (CurMon = LMon) then
               begin
                  if LMon = 1 then
                     begin
                        seek(Accounts_File,12);
                        write(Accounts_File,Work_Account)
                     end;
                  FillChar(Work_Account,199,0);
                  Work_Account.Entry_Year := Curdate div 512;
                  seek(Accounts_File,LMon mod 12);
                  write(Accounts_File,Work_Account)
               end
      end
```

File Name: LEDGER.PAS (cont)

```
   else
      with Work_Account do
         repeat
            AX000[2] := 0;AX000[3] := 0;A2000[7] := 0;
            for R1 := 1 to 6 do A2000[7] := A2000[7] + A2000[R1];
            for R1 := 7 to 10 do AX000[2] := AX000[2] + A2000[R1];
            for R1 := 9 to 16 do AX000[3] := AX000[3] + A3000[R1];
            AX000[4] := AX000[1] + AX000[2] - AX000[3];
            AX000[6] := A3000[10]+A6000[17]-A2000[3]-A6000[18]-
                                    A6000[19]+A6000[20]+A6000[21]+AX000[5];
            Seek(Accounts_File,NDX mod 12);
            write(Accounts_File,Work_Account);
            Flush(Accounts_File);
            if NDX mod 12 <> Curmon mod 12 then
               begin
                  NDX := NDX + 1;
                  A1 := AX000[4]; A6 := AX000[6];
                  seek(Accounts_File,NDX mod 12);
                  read(Accounts_File,Work_Account);
                  AX000[1] := A1; AX000[5] := A6
               end
            else NDX := -1
         until NDX = -1
end;  { Procedure Ledger_Record_IO }

Procedure Ledger(Cat,Item,LDate : integer; PAmt : real);

var
   Posting_Account : AER_Accounts;

begin
   Ledger_Record_IO('R',Ldate,Posting_Account);
   if (Cat = 6) and (Item = 15) then
      begin
         Cat := 3; Item := 10
      end
   else if (Cat = 6) and (Item = 17) then
      begin
         Cat := 2; Item := 3
      end
   else if (Cat = 6) and (Item = 16) then Item := 17;
   with Posting_Account do
      if Cat = 2 then
         begin
            A2000[Item] := A2000[Item] + PAmt;
            if Item in [1..5] then A2QTY[Item] := A2QTY[Item] + 1
         end
```

File Name: LEDGER.PAS (cont)

```
      else if Cat = 3 then
         begin
            A3000[Item] := A3000[Item] + PAmt;
            if Item in [10..13] then A3QTY[Item] := A3QTY[Item] + 1
         end
      else      {Cat = 6}
         begin
            A6000[Item] := A6000[Item] + PAmt;
            if Item in [17..19] then A6QTY[Item] := A6QTY[Item] + 1
         end;
   Ledger_Record_IO('W',LDate,Posting_Account)
end;  { Procedure Ledger}
```

File Name: HARDCOPY.PAS

```
Function Printer_OK : byte;

var
   P1 : byte;

begin
   Prepared_Screen := Display_Screen;
   repeat
      Regs.AX := $0200;
      Regs.DX := 0;
      Intr($17,Regs);
      if hi(Regs.AX) <> 144 then
         if Print_On then
            begin
               Display_Window(6,11);
               P1 := Key_Depressed
            end
   until (hi(Regs.AX) = 144) or (ESC) or (Not(Print_On));
   if hi(Regs.AX) = 144 then
      begin
         Printer_OK := 0;
         Print_On := True
      end
   else if (ESC) or (Not(Print_On)) then
      begin
         Printer_OK := 1;
         Print_On := False
      end;
   Display_Screen := Prepared_Screen

end; { Function Printer_OK }
```

```
File Name: HARDCOPY.PAS (cont)

Function Tab(Spaces : integer) : String25;

var
   T1 : integer;
   Temp_Space : String25;

begin
   Temp_Space := '';
   for T1 := 1 to Spaces do Temp_Space := Temp_Space + ' ';
   Tab := Temp_Space
end;  { function Tab }

Procedure Form_1108;

const
   LCat : array[1..11] of string[25] = ('1401: N/R of Pay',
      '1402: Loss of Funds','1403: Medical/Dental','1404: Funeral',
      '1405: Emergency Travel','1406: Init Rent & Deposit',
      '1406: Rent to Stop Evict.','1407: Food','1408: Utilities',
      '1409: Auto','1410: Other');
var
   F1, LCat_NDX : integer;
   AmtL : real;
   Tb,Dbl_On,Dbl_Off,PStat :char;
   SetTab,ClrTab,UL_On,UL_Off : string[3];
   P10,P15,LCat_Str : string[5];
   P12,Pit : String[6];
   Loan_Amt : string[7];
   Pay_Amt : string[10];
   Line,Line1 : String[88];
   OTH : string[21];
   Grph,Box,BoxX,Act,Ret : string[25];
   OTH1 : string[27];
   Rmks,Rmks1 : string[40];

begin
   LCat_Str := Field_Contents(1,19);
   LCat_NDX := Integer_Value(copy(Lcat_Str,3,2));
   if (LCat_Str[5] = 'R') or (LCat_NDX in [7..10]) then
      LCat_NDX := LCat_NDX + 1;
   AmtL := Real_Value(Field_Contents(1,20)); Str(AmtL:7:2,Loan_Amt);
   Pay_Amt := Field_Contents(1,12) + 'x' +
              String_Int(1+Date_Difference(Field_Contents(1,14),
                                           Field_Contents(1,13)),2);
   Oth := Field_Contents(1,15); Oth1 := Field_Contents(1,16);
   Rmks := Field_Contents(1,23); Rmks1 := Field_Contents(1,24);
   P10 := chr(18); P12 := chr(27)+chr(58); P15 := chr(15);
   SetTab := chr(27)+chr(68); ClrTab := chr(27)+chr(68)+chr(0);
   Tb := chr(9);
```

File Name: HARDCOPY.PAS (cont)

```
   UL_On := chr(27)+chr(45)+chr(1); UL_Off := chr(27)+chr(45)+chr(0);
   Dbl_On := chr(14); Dbl_Off := chr(20);
   Grph := chr(27)+chr(76)+chr(11)+chr(0);
   BoxX := chr(0)+chr(0)+chr(0)+chr(255)+chr(195)+chr(165)+chr(153)+
           chr(153)+chr(165)+chr(195)+chr(255);
   Box := chr(0)+chr(0)+chr(0)+chr(255)+chr(129)+chr(129)+chr(129)+
          chr(129)+chr(129)+chr(129)+chr(255);
   if Field_Contents(1,3) = 'A' then
      begin
         Act := 'ACTIVE' + Grph + BoxX; Ret := 'RETIRED' + Grph + Box
      end
   else
      begin
         Act := 'ACTIVE' + Grph + Box; Ret := 'RETIRED' + Grph + BoxX
      end;
   if length(Oth) = 0 then Oth := '_____________';
   If length(Oth1) = 0 then Oth1 := '____________________';
   Line := '';
   for F1 := 1 to 88 do Line := Line + chr(196);
   Line1 := line;
   write(lst,P12,chr(27)+chr(88)+chr(6)+chr(96));
   write(lst,ClrTab,SetTab,chr(37),chr(44),chr(60),chr(77),
         chr(95),chr(0));
   writeln(lst,P12,chr(218) + Line + chr(191));
   writeln(lst,chr(179),P10,Tab(14),'ARMY EMERGENCY RELIEF INDIVIDUAL
           LOAN LEDGER',P12,Tb,Tb,chr(179));
   insert(chr(194),Line1,31);insert(chr(194),Line1,38);
   insert(chr(194),Line1,54);insert(chr(194),Line1,71);
   write(lst,P12);
   writeln(lst,chr(195),Line1,chr(180));
   writeln(lst,chr(179),P15,' NAME OF SERVICE MEMBER',P12,Tb,chr(179),
           P15,'GRADE',P12,Tb,chr(179),UL_On,P15,Tab(7),'STATUS',Tab(8),
           UL_Off,P12,Tb,chr(179),P15,'SOCIAL SECURITY NUMBER',P12,Tb,
           chr(179),P15,' CASE NUMBER',P12,Tb,chr(179));
   with Index do
      with Loan do
         begin
            write(lst,chr(179),P12,NAME,P12,Tb,chr(179),
                  Field_Contents(1,2),Tb,chr(179),,P15,Act,'     ',Ret,
                  P12,Tb,chr(179),' ',P10,SSN_Str(SSN),P12,Tb,chr(179),
                  P10,Dbl_On,Copy(SSN_Str(SSN),8,4),'/');
            if Loan_Nr < 10 then
               writeln(lst,Loan_Nr:1,Dbl_Off,P12,Tb,chr(179))
            else writeln(lst,Loan_Nr:2,Dbl_Off,P12,Tb,chr(179))
         end;
   Line1 := Line;
   insert(chr(197),Line1,31); insert(chr(197),Line1,38);
   insert(chr(197),Line1,54); insert(chr(197),Line1,71);
   writeln(lst,chr(195),Line1,chr(180));
```

File Name: HARDCOPY.PAS (cont)

```
    writeln(lst,chr(179),P15,
            ' APPLICANT  (If other than Service Member)',P12,Tb,chr(179),
           P15,'RELATION',P12,Tb,chr(179),P15,Tab(6),'REPAYMENT',P12,
            Tb,chr(179),P15,Tab(6),'DELINQUENT',P12,Tb,chr(179),P15,
            Tab(6),'UNCOLLECTIBLE',P12,Tb,chr(179));
    Line1 := copy(Line,1,50);
    insert(chr(197),Line1,16); insert(chr(197),Line1,33);
    delete(line1,50,2);
    writeln(lst,chr(179)',Field_Contents(1,6),Tb,chr(179),P15,
            Field_Contents(1,7),P12,Tb,chr(195),Line1,chr(180));
    Line1 := copy(Line,1,36); insert(chr(193),Line1,31);
    writeln(lst,chr(195),Line1,chr(180),P15,'MONTHLY ALLOTMENT:',P12,Tb,
            chr(179),P15,'DATE ',UL_On,Tab(16),UL_Off,P12,Tb, chr(179),
            P15,'AMOUNT ',P12,UL_On,Tab(12),UL_Off,Tb,chr(179));
    write(lst,ClrTab,SetTab,chr(44),chr(60),chr(77),chr(95),chr(0));
    writeln(lst,chr(179),P15,' MILITARY ADDRESS OF SERVICE MEMBER',P12,
            Tb,chr(179),P15,'AMOUNT ',P12,Pay_Amt:10,Tb,chr(179),P15,
            'AMOUNT ',UL_On,Tab(14),UL_Off,P12,Tb,chr(179),P15,'DA FORM
            1106:', P12,Tb,chr(179));
    writeln(lst,chr(179),Field_Contents(1,8),Tb,chr(179),P15,'START    ',
            P12,Field_Contents(1,13),Tb,chr(179),P15,'LETTERS TO
            BORROWER:',P12,Tb,chr(179),P15,'APPROVED ',P12,UL_On,Tab(10),
            UL_Off,Tb,chr(179));
    Line1 := copy(Line,1,37);
    writeln(lst,chr(195),Line1,chr(180),P15,'STOP     ',P12,
            Field_Contents(1,14),Tb,chr(179),P15,'DATE ',UL_On,Tab(16),
            UL_Off,P12,Tb,chr(179),P15,'DA FORM 1105-3:',P12,Tb,
            chr(179));
    writeln(lst,chr(179),P15,' HOME ADDRESS OF SERVICE MEMBER',P12,Tb,
            chr(179),P15,'OTHER ',OTH,P12,Tb,chr(179),P15,'DATE ',UL_On,
            Tab(16),UL_Off,P12,Tb,chr(179),P15,'POSTED ',P12,UL_On,
            Tab(11),UL_Off,Tb,chr(179));
    writeln(lst,chr(179),Field_Contents(1,9),Tb,chr(179),P15,OTH1,P12,Tb,
            chr(179),P15,'DATE ',UL_On,Tab(16),UL_Off,P12,Tb,chr(179),
            UL_On,Tab(15),UL_Off,Tb,chr(179));
    writeln(lst,chr(179),Field_Contents(1,10),Tb,chr(179),Tb,chr(179),Tb,
            chr(179),Tb,chr(179));
    Line1 := Line;
    insert(chr(194),Line1,11); insert(chr(194),Line1,24);
    insert(chr(193),Line1,38); insert(chr(194),Line1,52);
    insert(chr(193),Line1,54); insert(chr(194),Line1,64);
    insert(chr(193),Line1,71); insert(chr(194),Line1,76);
    writeln(lst,chr(195),Line1,chr(180));
    write(lst,ClrTab,SetTab,chr(17),chr(30),chr(58),chr(70),chr(82),
          chr(95),chr(0),chr(13));
    writeln(lst,chr(179),'    ',P15,'DATE',P12,Tb,chr(179),P15,'CHECK OR
           RECEIPT',P12,Tb,chr(179),Tab(11),P15,'EXPLANATION',  P12,Tb,
           chr(179),' ',P15,'AMOUNT OF LOAN',P12,Tb,chr(179),' ',P15,
           'AMOUNT OF LOAN',P12,Tb,chr(179),'     ',P15,'BALANCE',P12,Tb,
           chr(179));
```

File Name: HARDCOPY.PAS (cont)

```
   writeln(lst,chr(179),Tb,chr(179),P15,'      NUMBER',P12, Tb, chr(179),
           Tb,chr(179),Tb,chr(179),P15,'   REPAYMENTS',P12,Tb,chr(179),
           Tb,chr(179));
   Line1 := Line;
   insert(chr(197),Line1,11); insert(chr(197),Line1,24);
   insert(chr(197),Line1,52); insert(chr(197),Line1,64);
   insert(chr(194),Line1,72); insert(chr(197),Line1,76);
   insert(chr(194),Line1,85);
   writeln(lst,chr(195),Line1,chr(180));
   write(lst,ClrTab,SetTab,chr(17),chr(30),chr(58),chr(70),chr(78),
         chr(82),chr(91),chr(95),chr(0),chr(13));
   writeln(lst,chr(179),Field_Contents(1,17),' ',chr(179),P10,
           Field_Contents(1,18),P12,Tb,chr(179),Lcat[LCat_NDX],Tb,
           chr(179),P10,Field_Contents(1,20):8,P12,Tb,chr(179),Tb,
           chr(179),Tb,chr(179),P10,copy(Loan_Amt,1,4):6,P12,Tb,
           chr(179),P10,copy(Loan_Amt,6,2):2,P12,Tb,chr(179));
   Line1 := Line;
   insert(chr(197),Line1,11); insert(chr(197),Line1,24);
   insert(chr(193),Line1,52); insert(chr(197),Line1,64);
   insert(chr(197),Line1,72); insert(chr(197),Line1,76);
   insert(chr(197),Line1,85);
   writeln(lst,chr(195),Line1,chr(80));
   insert(chr(196),Line1,52); delete(Line1,53,1);
   write(lst,ClrTab,SetTab,chr(17),chr(30),chr(70),chr(78),chr(82),
         chr(91),chr(95),chr(0),chr(13));
   if (length(Rmks) = 40) or (length(Rmks1) = 40) then Pit := P15
   else Pit := P12;
   for F1 := 1 to 19 do
      begin
         if F1 in [1,2] then
            begin
               writeln(lst,chr(179),Tb,chr(179),Tb,chr(179),Pit,Rmks,
                       P12,Tb,chr(179),Tb,chr(179),Tb,chr(179),Tb,
                       chr(179),Tb,chr(179));
               Rmks := Rmks1
            end
         else
             writeln(lst,chr(179),Tb,chr(179),Tb,chr(179),Tb,chr(179),
                     Tb,chr(179),Tb,chr(179)',Tb,chr(179),Tb,chr(179));
         writeln(lst,chr(195),Line1,chr(196),chr(180))
      end;
   writeln(lst,chr(179),Tb,chr(179),Tb,chr(179),Tb,chr(179),Tb,
           chr(179),Tb,chr(179)',Tb,chr(179),Tb,chr(179));
   Line1 := Line;
   insert(chr(193),Line1,11); insert(chr(193),Line1,24);
   insert(chr(196),Line1,52);
   insert(chr(193),Line1,64); insert(chr(193),Line1,72);
   insert(chr(193),Line1,76); insert(chr(193),Line1,85);
   writeln(lst,chr(192),Line1,chr(217));
```

File Name: HARDCOPY.PAS (cont)

```
   writeln(lst,P10,Dbl_On,'DA FORM 1108',Tab(21),
           copy(SSN_Str(Index.SSN),8,11),Dbl_Off,P12);
   for F1 := 1 to 4 do writeln(lst)
end;  { Procedure Form_1108 }

Procedure Print_Header(Header_Ident : integer);

var
   Hdr : string[80];

begin
   if Header_Ident in [1..6] then
      if Header_Ident = 1 then
         Hdr := '            Chapter 13 Loans as of '
      else if Header_Ident = 2 then
         Hdr := '          All Delinquent Loans as of '
      else if Header_Ident = 3 then
         Hdr := ' Uncollectible Loans Awaiting Approval as of '
      else if Header_Ident = 4 then
         Hdr := '             Paid-Off Loans as of '
      else if Header_Ident = 5 then
         Hdr := 'Transfer-In Loans Awaiting 1st Repayment as of '
      else Hdr := '  Transfer-Out Loans Awaiting Approval as of ';
   write(lst,chr(18),chr(13));
   if Header_Ident in [7..9] then
      writeln(lst,Tab(21),(Header_Ident-6):2,
              ' Month Old Delinquent Loans as of ',CSDate)
   else if Header_Ident = 10 then
      writeln(lst,Tab(17),
              'Delinquent Loans More than 3 Months Old as of ',CSDate)
   else writeln(lst,Tab(16),Hdr,CSDate);
   writeln(lst,chr(27),chr(68),chr(0),chr(27),chr(68),chr(11),
           '"*08>G',chr(0));
   writeln(lst,chr(9),chr(9),chr(9),chr(9),chr(9),
           'LOAN  ACCOUNT    LAST');
   writeln(lst,chr(9),'NAME',chr(9),'SSN',chr(9),
          'GRADE STATUS   NR   BALANCE   PAYMENT');
   writeln(lst)
end;  { Procedure Print_Header }

Procedure Print_Report(Loan_Index : integer; Account : Entire_Account);

var
   Grade : string[3];
   S, Tb : char;
   BDate : string[9];
   Balance : real;
   Box : string[15];
```

File Name: HARDCOPY.PAS (cont)

```
begin
   Box := chr(27)+chr(76)+chr(11)+chr(0)+chr(0)+chr(0)+chr(0)+chr(255)+
          chr(129)+chr(129)+chr(129)+chr(129)+chr(129)+chr(129)+
          chr(255);
    write(lst,chr(18),chr(27),chr(68),chr(0),chr(27),chr(68),chr(3),
          chr(30),chr(43),chr(48),chr(57),chr(62),chr(71),chr(0),
          chr(13));
   Tb := chr(9);
   with Index do
      with Account[Rec_Pos[Loan_Index]].Loan_Data do
         begin
            write(lst,Box,Tb,Name,Tb,SSN_Str(SSN));
            Decode_Grade_and_Status(Grade_and_Status,Grade,S);
            if S = 'A' then
               write(lst,Tb,Grade,Tb,'Active')
            else write(lst,Tb,Grade,Tb,'Retired');
            Split_Date_and_Money(Balance_Info,BDate,Balance);
            writeln(lst,Tb,Loan_Nr,Tb,Balance:7:2,Tb,BDate)
         end
end; { Procedure Print_Report }

Procedure Print_General_Ledger(Print_Record : AER_Accounts);

var
   Tb : char;
   P1, P2 : integer;
   Prt_Str : String80;
   Lgr_Fmt : text;

begin
   Tb := chr(9); P2 := 1;
   write(lst,chr(18),chr(27),chr(68),chr(0),chr(27),chr(68),chr(50),
        chr(60),chr(0),chr(13));
   writeln(lst,Tab(25),Field_Contents(5,10));
   assign (Lgr_Fmt,LEDGER_FRM); reset(Lgr_Fmt);
   for P1 := 1 to 46 do with Print_Record do
      begin
         if P1 in [1,3,5,17,19,29,31,33,35,37,45] then writeln(lst)
         else
            begin
               readln(Lgr_Fmt,Prt_Str);
               if P1 in [2,6,20,34] then writeln(lst,Prt_Str)
               else if P1 in [4,18,30,32,36,46] then
                  begin
                     writeln(lst,Prt_Str,Tb,Tb,AX000[P2]:10:2);
                     P2 := P2 + 1
                  end
               else if P1 in [7..11] then
```

File Name HARDCOPY.PAS (cont)

```
                writeln(lst,Prt_Str,Tb,A2QTY[P1-6]:4,Tb,
                        A2000[P1-6]:10:2)
             else if P1 = 40 then
                writeln(lst,Prt_Str,Tb,A2QTY[3]:4,Tb,A2000[3]:10:2)
             else if P1 in [12..16] then
                writeln(lst,Prt_Str,Tb,Tb,A2000[P1-6]:10:2)
             else if P1 in [22..25] then
                 writeln(lst,Prt_Str,Tb,A3QTY[P1-12]:4,Tb,
                         A3000[P1-12]:10:2)
             else if P1 = 38 then
                writeln(lst,Prt_Str,Tb,A3QTY[10]:4,Tb,A3000[10]:10:2)
             else if P1 in [21,26..28] then
                writeln(lst,Prt_Str,Tb,Tb,A3000[P1-12]:10:2)
             else if P1 in [41,42] then
                 writeln(lst,Prt_Str,Tb,A6QTY[P1-23]:4,Tb,
                         A6000[P1-23]:10:2)
             else if P1 in [43,44] then
                writeln(lst,Prt_Str,Tb,Tb,A6000[P1-23]:10:2)
             else writeln(lst,Prt_Str,Tb,A6QTY[17]:4,Tb,
                          A6000[17]:10:2)
          end
      end;
   Close(Lgr_Fmt);
   for P1 := 1 to 20 do writeln(lst)
end;  { Procedure Print_General_Ledger }
```

File Name: AERPROCS.PAS

```
Function Valid_Account_Code(Account_Code : String5) : boolean;

begin
   if (Integer_Value(copy(Account_Code,1,4)) - 2000 in [1..6,8..10]) or
      (Integer_Value(copy(Account_Code,1,4)) - 3008 in [1..8]) or
      (Integer_Value(copy(Account_Code,1,4)) - 6014 in [1..7]) then
            Valid_Account_Code := True
   else
      begin
         Valid_Account_Code := False;
         Buzzer
      end
end; { Function Valid_Account_Code }
```

File Name: AERPROCS.PAS (cont)

```
Procedure Display_Account_Ident(Disp_Nr : integer);

begin
   with Index do
      begin
         Decode_Grade_and_Status(Grade_and_Status,Grade,Status);
         Fill_Field(Disp_Nr,1,Name);
         Fill_Field(Disp_Nr,2,SSN_Str(SSN));
         Fill_Field(Disp_Nr,3,Grade);
         if Disp_Nr <> 4 then
            if Status = 'A' then Fill_Field(Disp_Nr,4,'Active ')
            else Fill_Field(Disp_Nr,4,'Retired')
      end
end;  { Procedure Display_Account_Ident}

Procedure Display_Loans(Disp_Nr, Start_Field, Disp_Start : integer;
                        Account : Entire_Account);

var
   LDate, BDate, ADate : String9;
   D1 : integer;
   Loan_Amt, Balance, Allot_Amt : real;
   Loan_Status : array[0..6] of string[32];

begin
   D1 := Disp_Start;
   Loan_Status[2] := 'Delinquent                          ';
   Loan_Status[3] := 'Uncollectible (not yet approved)';
   Loan_Status[4] := 'Paid-Off.  Holding for 30 Days. ';
   Loan_Status[5] := 'Transfer-In.  Awaiting 1st Pymt.';
   Loan_Status[6] := 'Transfer-Out.  Awaiting MANCOR. ';
   repeat
      with Account[Rec_Pos[D1]].Loan_Data do
         begin
            Loan_Status[0] := 'Current                             ';
            Fill_Field(Disp_Nr,Start_Field,String_Int(Loan_Nr,2));
            split_Date_and_Money(Loan_Info,LDate,Loan_Amt);
            Fill_Field(Disp_Nr,Start_Field+1,String_Real(Loan_Amt,7));
            Split_Date_and_Money(Balance_Info,BDate,Balance);
            Fill_Field(Disp_Nr,Start_Field+2,String_Real(Balance,7));
            if Repay_Method and $7F <> 0 then
               Fill_Field(Disp_Nr,Start_Field+3,' CH-13')
            else if Repay_Method = 0 then
               Fill_Field(Disp_Nr,Start_Field+3,'Allot')
            else Fill_Field(Disp_Nr,Start_Field+3,'P-Note');
            Split_Date_and_Money(Allot_Info,ADate,Allot_Amt);
```

File Name: AERPROCS.PAS (cont)

```
            if Acct_Status = 1 then
                  Fill_Field(Disp_Nr,Start_Field+4,'Various')
            else
               Fill_Field(Disp_Nr,Start_Field+4,
                                          String_Real(Allot_Amt,7));
            if abs(Loan_Amt - Balance) < 0.001 then
               BDate := 'None Yet ';
            Fill_Field(Disp_Nr,Start_Field+5,BDate);
            if (Acct_Status = 0) and (abs(Loan_Amt - Balance) < 0.001)
                and (trunc(Allot_Info + 32.0) - CurDate > 0) then
                  Loan_Status[0] := 'Repayments to start '+
                                                    copy(ADate,4,9)
            else if Acct_Status = 1 then
               Loan_Status[1] := 'CH-13 at '+String_Int(Repay_Method,3)+
                                                ' cents on the dollar';
            Fill_Field(Disp_Nr,Start_Field+6,Loan_Status[Acct_Status]);
            Start_Field := Start_Field + 7; D1 := D1 + 1
         end;
      until (D1 = Disp_Start + 5) or (Rec_Pos[D1] = 0)
end; { Procedure Display_Loans }

Procedure Get_Account(Key_Value : String25; var Nr_of_Loans : integer;
                      var Account : Entire_Account);
var
   Record_File_Position : integer;

begin
   Nr_of_Loans := 0;
   Get_Index_Record(Key_Value,Record_File_Position);
   if Record_File_Position <> 0 then
      begin
         FillChar(Account,405,0); FillChar(Rec_Pos,30,0);
         FillChar(Stats_Code,7,0);
         seek(Loan_File,Index.Accounts_Ptr);
         repeat
            read(Loan_File,Loan);
            Nr_of_Loans := Nr_of_Loans + 1;
            Account[Loan.Loan_Nr].Loan_Data := Loan;
            Account[Loan.Loan_Nr].Rec_Loc := FilePos(Loan_File) - 1;
            Stats_Code[Loan.Acct_Status] := Stats_Code[Loan.Acct_Status]
                                                            + 1;
            Rec_Pos[Nr_of_Loans] := Loan.Loan_Nr;
            seek(Loan_File,Loan.Next_Record)
         until Loan.Next_Record = 0
      end
end;  { Procedure Get_Account }
```

File Name: AERPROCS.PAS (cont)

```
Procedure Loan_Entry(Entry_Type : integer);

var
   Cat : String5;
   L1, L2, WMon, LCat, Mon_Diff : integer;
   ADate,LDate,BDate : String9;
   Account : Entire_Account;

begin
   repeat
      Prepare_Screen(1);
      Display_Screen := Prepared_Screen;
      if Entry_Type = 3 then
         begin
            gotoXY(3,17);write('Date of');
            gotoXY(2,18);write('  Grant  ');
            gotoXY(50,17);write('Grant ');
            Screen_Input(1,1,4);
            Screen_Input(1,8,8);
            Screen_Input(1,17,20); if ESC then Exit
         end
      else if Entry_Type in [1,2] then
         begin
            Screen_Input(1,1,4); if ESC then Exit;
            Get_Account(Field_Contents(1,4),L1,Account);
            repeat
               Screen_Input(1,5,5);if ESC then Exit;
               L2 := Integer_Value(Field_Contents(1,5))
            until L2 in [0..14];
            if L1 <> 0 then
               if L2 < Rec_Pos[L1] then
                  begin
                     L2 := Rec_Pos[L1];
                     Fill_Field(1,5,String_Int(L2,2))
                  end;
            L2 := L2 + 1;
            Screen_Input(1,6,20); if ESC then Exit;
            if Entry_Type = 1 then
               begin
                  Fill_Field(1,21,'None Yet ');
                  Fill_Field(1,22,Field_Contents(1,20));
                  Screen_Input(1,23,24)
               end
            else Screen_Input(1,21,24)
         end
      else
         begin
            gotoXY(66,4); write('Old Loan Nr ');
```

File Name: AERPROCS.PAS (cont)

```
            Screen_Input(1,1,4); if ESC then Exit;
            Get_Account(Field_Contents(1,4),L1,Account);
            repeat
               Screen_Input(1,25,25); if ESC then Exit;
               L2 := Integer_Value(Field_Contents(1,25))
            until L2 in [1..15];
            if L1 <> 0 then
               repeat
                  if Account[L2].Rec_Loc <> 0 then L2 := L2 + 1
               until (Account[L2].Rec_Loc = 0) or (L2 = 15);
            Fill_Field(1,25,String_Int(L2,1));
            Screen_Input(1,11,13); Screen_Input(1,17,17);
            Screen_Input(1,20,22)
         end;
      if ESC then Exit;
      gotoXY(5,2);TextBackground(Red);TextColor(White);
      write('Please VERIFY information.  Press ',
            chr(17),'┘ if correct or ESC to stop entry.');
      TextBackground(Blue);TextColor(Black);
      repeat
         if ESC then Exit
      until Key_Depressed = 13;
      FillChar(Index,47,0);FillChar(Loan,25,0);
      with Index do
         begin
            Name := Field_Contents(1,1);
            Grade := Field_Contents(1,2);
            Status := Field_Contents(1,3);
            Grade_and_Status := Encode_Grade_and_Status(Grade,Status);
            SSN := Real_Value(Field_Contents(1,4))
         end;
      with Loan do
         begin
            Loan_Info := Real_Value(Field_Contents(1,20));
            LDate := Field_Contents(1,17);
            Extract_Date_Data(LDate,WMon,Code);
            if Entry_Type = 1 then Ledger(3,10,WMon,Loan_Info)
            else if Entry_Type = 3 then Ledger(3,11,WMon,Loan_Info);
            if Entry_Type <> 3 then
               begin
                  Loan_Nr := L2;
                  Loan_Info := Merge_Date_and_Money(LDate,Loan_Info);
                  Allot_Info := Real_Value(Field_Contents(1,12));
                  ADate := Field_Contents(1,13);
                  Balance_Info := Real_Value(Field_Contents(1,22));
                  if Entry_Type = 1 then BDate := LDate
                  else BDate := Field_Contents(1,21);
```

File Name: AERPROCS.PAS (cont)

```
                  if Entry_Type = 2 then
                     begin
                        Extract_Date_Data(BDate,WMon,Code);
                        Ledger(6,16,WMon,Balance_Info)
                     end;
                  if Entry_Type = 2 then Acct_Status := 5
                  else Acct_Status := 0;
                  Balance_Info :=
                              Merge_Date_and_Money(BDate,Balance_Info);
                  if Field_Contents(1,11) = 'A' then Repay_Method := 0
                  else Repay_Method := $80;
                  Allot_Info := Merge_Date_and_Money(ADate,Allot_Info);
                  Acct_Status := New_Status('A',Loan)
               end
         end;   { with Loan do }
        if Entry_Type in [1,2,4] then Write_Index_Record;
        if Entry_Type in [1,3] then Record_General_Stats(WMon);
        if (Entry_Type in [1,2]) and (Printer_OK = 0) then Form_1108;
   Until lo(Regs.AX) = 27
end;  { Procedure Loan_Entry }

Procedure Record_Payments(Entry_Mode : integer);

var
   R1, LoanNr, Field, PMon, Nr_Loans : integer;
   Match_Found : boolean;
   PDate : string[9];
   Rcpt_Nr : String[8];
   Allot_Amt, Payment : real;
   Account : Entire_Account;

procedure Post(Loan_Num : integer; New_Balance : real);

begin
   if ESC then Exit;
   Display_Screen := Prepared_Screen;
   with Account[Loan_Num].Loan_Data do
      begin
         Acct_Status := New_Status('D',Account[Loan_Num].Loan_Data);
         if New_Balance = 0.0 then
            Acct_Status := 4
         else if Acct_Status <> 1 then Acct_Status := 0;
         Balance_Info := Merge_Date_and_Money(PDate,New_Balance);
         Acct_Status := New_Status('A',Account[Loan_Num].Loan_Data)
      end;
   seek(Loan_File,Account[Loan_Num].Rec_Loc);
   write(Loan_File,Account[Loan_Num].Loan_Data);
   Display_Loans(4,12,1,Account)
end;  { internal procedure Apply_to_Loan }
```

File Name: AERPROCS.PAS (cont)

```
procedure Apply_Payment(Loan_Num : integer);

var
   LDate, BDate : string[9];
   V1 : integer;
   Balance, New_Balance, Ledger_Amt : real;
   Answer : string[2];
   Transaction_Complete : boolean;

begin
   if ESC then Exit;
   if Loan_Num <> 0 then
      begin
         Prepared_Screen := Display_Screen;
         Fill_Field(4,4,String_Int(Loan_Num,2)); gotoXY(48,2);
         write('Press ',chr(17),'┘ if Loan Nr ',Loan_Num:2,' is the');
         gotoXY(48,3); write('  Correct Loan.');
         gotoXY(48,5); write('If incorrect, press any other');
         gotoXY(48,6); write('  key to select correct loan.');
         if Key_Depressed <> 13 then Loan_Num := 0;
         Display_Screen := Prepared_Screen;
         if ESC then exit
      end;
   if Loan_Num = 0 then
      begin
         repeat
            Loan_Num := 0; Screen_Input(4,10,10); if ESC then Exit;
            Answer := Field_Contents(4,10);
            Answer[1] := upcase(Answer[1]);
            Fill_Field(4,10,'  '); Loan_Num := Integer_Value(Answer);
            if Loan_Num <> 0 then
               if Account[Loan_Num].Rec_Loc = 0 then Loan_Num := 0
         until (Answer[1] in ['A'..'C']) or (Loan_Num <> 0);
         if Answer = 'A' then Ledger(2,1,PMon,Payment)
         else if Answer = 'B' then Ledger(2,2,PMon,Payment)
         else if Answer = 'C' then Ledger(2,4,PMon,Payment);
         if Answer[1] in ['A'..'C'] then Exit
      end;
   repeat
      Fill_Field(4,4,String_Int(Loan_Num,2));
      Transaction_Complete := True;
      with Account[Loan_Num].Loan_Data do
         begin
            Split_Date_and_Money(Balance_Info,Date,Balance);
            New_Balance := Balance - Payment;
            if New_Balance <= 0.001 then Fill_Field(4,8,'    0.00')
            else Fill_Field(4,8,String_Real(New_Balance,7));
            Prepared_Screen := Display_Screen;
```

File Name: AERPROCS.PAS (cont)

```
            if New_Balance >= -0.001 then
               begin
                  if New_Balance < 0.001 then New_Balance := 0.0;
                  Post(Loan_Num,New_Balance);
                  Ledger(2,3,PMon,Payment)
               end
            else
               begin
                  gotoXY(48,2);
                  if Balance < 0.001 then
                     begin
                        write('Loan Paid Off.  Should I apply');
                        gotoXY(48,3);
                        write('the ',Payment:7:2,' repayment to:');
                        Ledger_Amt := Payment; Payment := 0.0
                     end
                  else
                     begin
                        Payment := Balance;
                        write('Applying ',Payment:7:2,
                              ' to Loan. Should');
                        gotoXY(48,3);
                        write('I apply remaining ',
                              Abs(New_Balance):7:2,' to:');
                        Ledger_Amt := Abs(New_Balance)
                     end;
                  repeat
                     V1 := 0; Screen_Input(4,11,11); if ESC then Exit;
                     Answer := Field_Contents(4,11);
                     V1 := Integer_Value(Answer);
                     if V1 <> 0 then
                        if Account[V1].Rec_Loc = 0 then V1 := 0
                  until (Answer[1] in ['A','B']) or (V1 <> 0);
                  gotoXY(48,2);
                  write('                                    ');
                  gotoXY(48,3);
                  write('                                    ');
                  Fill_Field(4,11,'  ');
                  Prepared_Screen := Display_Screen;
                  Post(Loan_Num,0.00);
                  if Payment <> 0.0 then Ledger(2,3,PMon,Payment);
                  if Answer[1] = 'A' then Ledger(2,1,PMon,Ledger_Amt)
                  else if Answer[1] = 'B' then
                     Ledger(2,4,PMon,Ledger_Amt)
                  else
                     begin
                        Transaction_Complete := False;
                        Loan_Num := V1;
                     end
```

File Name: AERPROCS.PAS (cont)

```
                end { if New_Balance < 0.001 }
            end {with Account do}
      until Transaction_Complete
end; { internal procedure Apply_Payment}

begin              { Main Body Record_Payments }
      PDate := ''; Rcpt_Nr := '';
      repeat
         Prepare_Screen(4);
         Display_Screen := Prepared_Screen;
         if Entry_Mode = 1 then
            begin
               Fill_Field(4,5,PDate); Fill_Field(4,6,Rcpt_Nr)
            end;
         if Entry_Mode in [1,2] then Field := 2
         else Field := 1;
         if Field_Contents(4,5) = '' then Screen_Input(4,5,6);
         if ESC then Exit;
         PDate := Field_Contents(4,5); Rcpt_Nr := Field_Contents(4,6);
         Extract_Date_Data(PDate,PMon,R1);
         Screen_Input(4,7,7); if ESC then Exit;
         Payment := Real_Value(Field_Contents(4,7));
         Screen_Input(2,Field,Field); if ESC then Exit;
         Get_Account(Field_Contents(4,Field),Nr_Loans,Account);
         if Nr_Loans <> 0 then
            begin
               Display_Account_Ident(4); Display_Loans(4,12,1,Account);
               Match_Found := False;
               R1 := 0;
               repeat
                  R1 := R1 + 1;
                  with Account[Rec_Pos[R1]].Loan_Data do
                     begin
                        Split_Date_and_Money(Allot_Info,Date,Allot_Amt);
                        if abs(Allot_Amt - Payment) < 0.001 then
                           begin
                              Match_Found := True; Apply_Payment(Loan_Nr)
                           end;
                        if ESC then exit
                     end
               until (Match_Found) or (R1 = Nr_Loans);
               if Not (Match_Found) then Apply_Payment(0);
               if ESC then Exit
            end  {if Nr_Loans <> 0 }
         else
            begin
               repeat
                  Screen_Input(4,9,9); if ESC then Exit;
```

File Name: AERPROCS.PAS (cont)

```
                R1 := Integer_Value(Field_Contents(4,9))
              until R1 in [1..5];
              Fill_Field(4,9,'  ');
              Ledger(2,R1,PMon,Payment)
           end;
        gotoXY(48,2);write('Press:');
        gotoXY(49,4);write('  ',chr(17),'┘ to post another payment');
        gotoXY(51,6);write('ESC to return to main menu')
     until Key_Depressed = 27
end; { Procedure Record_Payments }

Procedure Display_Financials(Mode : integer);

type
   String4 = string[4];
   Input_Set = set of 1..4;

var
   Disp_Acct : AER_Accounts;
   Valid_Input : Input_Set;
   WSDate, Test_Date : String9;
   Acct_Code : string[4];
   D1, TMon, WMon, Acct_Cat, Acct_Item, Copt : integer;


procedure Total_Financials;

var
   Temp_Fin : AER_Accounts;
   End_Month, T1, T2, T3 : integer;
   A2 : array[1..10] of real;
   A3 : array[9..16] of real;
   A6 : array[17..21] of real;
   A2Q : array[1..5] of integer;
   A3Q : array[10..13] of integer;
   A6Q : array[17..19] of integer;
   AX : array[1..6] of real;

begin
   if CurMon = 1 then
      begin
         T3 := 0; End_Month := 12;
      end
   else
      begin
         T3 := 1; End_Month := CurMon
      end;
   Ledger_Record_IO('R',T3,Disp_Acct);
```

File Name: AERPROCS.PAS (cont)

```
   for T1 := 2 to End_Month do with Disp_Acct do
      begin
         for T2 := 1 to 10 do A2[T2] := A2000[T2];
         for T2 := 9 to 16 do A3[T2] := A3000[T2];
         for T2 := 17 to 21 do A6[T2] := A6000[T2];
         for T2 := 1 to 5 do A2Q[T2] := A2QTY[T2];
         for T2 := 10 to 13 do A3Q[T2] := A3QTY[T2];
         for T2 := 17 to 19 do A6Q[T2] := A6QTY[T2];
         for T2 := 1 to 6 do AX[T2] := AX000[T2];
         Ledger_Record_IO('R',T1,Disp_Acct);
         for T2 := 1 to 10 do A2000[T2] := A2000[T2] + A2[T2];
         for T2 := 9 to 16 do A3000[T2] := A3000[T2] + A3[T2];
         for T2 := 17 to 21 do A6000[T2] := A6000[T2] + A6[T2];
         for T2 := 1 to 5 do A2QTY[T2] := A2QTY[T2] + A2Q[T2];
         for T2 := 10 to 13 do A3QTY[T2] := A3QTY[T2] + A3Q[T2];
         for T2 := 17 to 19 do A6QTY[T2] := A6QTY[T2] + A6Q[T2];
         for T2 := 1 to 6 do AX000[T2] := AX000[T2] + AX[T2]
      end  {with Disp_Acct }
end;  { internal Procedure Total_Financials }

procedure Write_Accounts;

begin
   with Disp_Acct do
      begin
         gotoXY(30,4);write(AX000[1]:10:2);
         for I := 1 to 10 do
            if I in [1..5] then
               begin
                  gotoXY(24,4+I); write(A2QTY[I]:4);
                  gotoXY(30,4+I); write(A2000[I]:10:2)
               end
            else
               begin
                  gotoXY(30,4+I);write(A2000[I]:10:2)
               end;
         gotoXY(30,15);write(AX000[2]:10:2);
         gotoXY(30,17);write(AX000[5]:10:2);
         gotoXY(24,18);write(A3QTY[10]:4);
         gotoXY(30,18);write(A3000[10]:10:2);
         gotoXY(24,19);write(A6QTY[17]:4);
         gotoXY(30,19);write(A6000[17]:10:2);
         gotoXY(24,20);write(A2QTY[3]:4);
         gotoXY(30,20);write(A2000[3]:10:2);
         gotoXY(24,21);write(A6QTY[18]:4);
         gotoXY(30,21);write(A6000[18]:10:2);
         gotoXY(24,22);write(A6QTY[19]:4);
         gotoXY(30,22);write(A6000[19]:10:2);
         gotoXY(30,23);write(A6000[20]:10:2);
```

File Name: AERPROCS.PAS (cont)

```
         gotoXY(30,24);write(A6000[21]:10:2);
         gotoXY(30,25);write(AX000[6]:10:2);
         for I := 9 to 16 do
            if I in [10..13] then
               begin
                  gotoXY(64,I-5); write(A3QTY[I]:4);
                  gotoXY(70,I-5); write(A3000[I]:10:2)
               end
            else
               begin
                  gotoXY(70,I-5);write(A3000[I]:10:2)
               end;
         gotoXY(70,12);write(AX000[3]:10:2);
         gotoXY(70,13);write(AX000[4]:10:2);
         gotoXY(77,24)
      end   {with Main_Accounts}
end;  {internal procedure Write_Accounts}

begin
   WSDate := CSDate; Copt := 0; WMon := CurMon;
   if Mode = 2 then Valid_Input := [1,2] else Valid_Input := [1..4];
   repeat
      if ((Copt <> 7) and (Mode = 1)) or (Mode = 2) then
         begin
            Prepare_Screen(5); Display_Screen := Prepared_Screen;
            Fill_Field(5,10,'GENERAL LEDGER FOR MONTH OF '
                                                + copy(WSDate,4,6));
            Ledger_Record_IO('R',WMon,Disp_Acct);
            Write_Accounts;
            repeat
               Screen_Input(5,4-Mode,4-Mode); if ESC then Exit;
               Copt := Integer_Value(Field_Contents(5,4-Mode));
               if Not(Copt in Valid_Input) then Buzzer
            until Copt in Valid_Input;
         end;
      if (Copt = Mode) or (Copt = 7) then
         begin
            Copt := Mode;
            Screen_Input(5,1,1); if ESC then Exit;
            Test_Date := '    ' + Field_Contents(5,1);
            Extract_Date_Data(Test_Date,TMon,D1);
            Code := Date_Difference(CSDate,Test_Date);
            if (Not(Code in [0..11])) or (D1 > CurDate) then
               begin
                  Display_Window(6,8);
                  if Key_Depressed = 27 then Exit
                  else Display_Screen := Prepared_Screen
               end
```

File Name: AERPROCS.PAS (cont)

```
            else
               begin
                  WSDate := Test_Date;
                  WMon := TMon;
                  Ledger_Record_IO('R',WMon,Disp_Acct)
               end
         end;
      if ((Mode = 2) and (Copt = 1)) or ((Mode = 1) and (Copt = 3)) then
         begin
            repeat
               Screen_Input(5,6-Mode,6-Mode); if ESC then Exit;
               Acct_Code := Field_Contents(5,6-Mode)
            until Valid_Account_Code(Acct_Code);
            Acct_Cat := Integer_Value(Acct_Code[1]);
            Acct_Item := Integer_Value(copy(Acct_Code,3,2));
            if Mode = 2 then
               begin
                  Screen_Input(5,8,8); if ESC then Exit;
                  Ledger(Acct_Cat,Acct_Item,WMon,
                                       Real_Value(Field_Contents(5,8)));
                  Ledger_Record_IO('R',WMon,Disp_Acct)
               end
            else
               begin
                  if Acct_Cat = 6 then
                     begin
                        if Acct_Item = 16 then Acct_Item := 17
                        else if Acct_Item = 15 then
                           begin
                              Acct_Cat := 3; Acct_Item := 10
                           end
                        else if Acct_Item = 17 then
                           begin
                              Acct_Cat := 2; Acct_Item := 3
                           end
                     end;
                  if ((Acct_Cat = 2) and (Acct_Item in [1..5])) or
                     ((Acct_Cat = 3) and (Acct_Item in [10..13])) or
                     ((Acct_Cat = 6) and (Acct_Item in [17..19])) then
                     with Disp_Acct do
                        begin
                           Screen_Input(5,6,6);if Esc then Exit;
                           if Acct_Cat = 2 then
                              A2QTY[Acct_Item] :=
                                       Integer_Value(Field_Contents(5,6))
                           else if Acct_Cat = 3 then
                              A3QTY[Acct_Item] :=
                                       Integer_Value(Field_Contents(5,6))
```

```
                        else
                           A6QTY[Acct_Item] :=
                                 Integer_Value(Field_Contents(5,6))
                     end;
                  Screen_Input(5,7,7); if ESC then Exit;
                  with Disp_Acct do
                     if Acct_Cat = 2 then
                        A2000[Acct_Item] :=
                                    Real_Value(Field_Contents(5,7))
                     else if Acct_Cat = 3 then
                        A3000[Acct_Item] :=
                                    Real_Value(Field_Contents(5,7))
                     else
                        A6000[Acct_Item] :=
                                   Real_Value(Field_Contents(5,7));
                  Ledger_Record_IO('W',WMon,Disp_Acct);
                  Ledger_Record_IO('R',WMon,Disp_Acct)
               end
         end  { if Mode = 2 }
      else if (Mode = 1) and (Copt = 2) then
         begin
            Display_Window(5,3); gotoXY(45,1);
            if CurMon <> 1 then
               write('01 JAN ',(80 + CurDate div 512):2,' To ',CSDate)
            else
               write('01 JAN ',(79 + CurDate div 512):2,' To 31 DEC ',
                     (79 + CurDate div 512):2);
            Total_Financials; Write_Accounts;
            repeat
               Screen_Input(5,9,9); if ESC then Exit;
               Copt := Integer_Value(Field_Contents(5,9));
               if Not(Copt in [1,2]) then Buzzer;
               if (Copt = 2) and (Printer_OK = 0) then
                  Print_General_Ledger(Disp_Acct)
            until Copt = 1;
            Copt := 7
         end
      else if (Mode = 1) and (Copt = 4) and (Printer_OK = 0) then
         Print_General_Ledger(Disp_Acct)
   until ESC
end; { Procedure Display_Financials }
```

File Name: AERPROCS.PAS (cont)

```
Procedure Display_General_Stats;

var
   WSDate, Test_Date : String9;
   D1, Copt, TMon, WMon : integer;
   Disp_Stats : General_Stats;

procedure Write_Grade;

var
  W1, W2, Tot_Nr : integer;
  Tot_Amt : real;

begin
   Tot_Nr := 0; Tot_Amt := 0.0;
   gotoXY(8,5);
   for W1 := 1 to 2 do
      for W2 := 1 to 9 do with Disp_Stats.Grade_Stats[W1,W2] do
         begin
            gotoXY(8,whereY);write(Qty:4);
            gotoXY(13,whereY);writeln(Amt:10:2);
            Tot_Nr := Tot_Nr + Qty;
            Tot_Amt := Tot_Amt + Amt
         end;
   gotoXY(8,23); write(Tot_Nr:4);gotoXY(13,23); write(Tot_Amt:10:2)
end;  {internal procedure Write_Grade }

procedure Write_Loan_Cats;

var
   W1, Tot_Nr : integer;
   Tot_Amt : real;

begin
   Tot_Nr := 0; Tot_Amt := 0.0;
   gotoXY(45,5);
   for W1 := 1 to 11 do
      with Disp_Stats do
         begin
            gotoXY(45,whereY);
            write(Loan_Cats[W1].Qty:4); gotoXY(50,whereY);
            writeln(Loan_Cats[W1].Amt:10:2);
            Tot_Nr := Tot_Nr + Loan_Cats[W1].Qty;
            Tot_Amt := Tot_Amt + Loan_Cats[W1].Amt;
            if W1 = 5 then
               begin
                  gotoXY(45,whereY);
                  write((Loan_Cats[6].Qty + Loan_Cats[7].Qty):4);
```

File Name: AERPROCS.PAS (cont)

```
                  gotoXY(50,whereY);
                  writeln((Loan_Cats[6].Amt + Loan_Cats[7].Amt):10:2)
               end
         end;
   gotoXY(45,17); write(Tot_Nr:4);gotoXY(50,17); write(Tot_Amt:10:2)
end;  { internal procedure Write_Loan_Cats }

procedure Write_Duty_Stations;

var
   W1, Tot_Nr : integer;
   Tot_Amt : real;

begin
   Tot_Nr := 0; Tot_Amt := 0.0; gotoXY(45,21);
   for W1 := 1 to 3 do with Disp_Stats.Duty_Station[W1] do
      begin
         gotoXY(45,whereY);
         write(Qty:4); gotoXY(50,whereY); writeln(Amt:10:2);
         Tot_Nr := Tot_Nr + Qty;
         Tot_Amt := Tot_Amt + Amt
      end;
   gotoXY(45,24); write(Tot_Nr:4); gotoXY(50,24); write(Tot_Amt:10:2)
end;  {internal procedure Write_Duty_Station}

procedure Apply_Change(Chg_Cat : integer; Chg_Ident : String3);

var
   A1, A2, Quantity : integer;
   Amount : real;

begin
   Screen_Input(6,6,6); if ESC then Exit;
   Quantity := Integer_Value(Field_Contents(6,6));
   Screen_Input(6,7,7); if ESC then Exit;
   Amount := Real_Value(Field_Contents(6,7));
   if Chg_Cat = 3 then
      begin
         A1 := 2;
         A2 := Integer_Value(copy(Chg_Ident,3,1));
         if Chg_Ident[1] = 'E' then A1 := 1
         else if Chg_Ident[1] = 'O' then A2 := A2 + 4
         else if Chg_Ident[1] = 'R' then A2 := 9;
         Disp_Stats.Grade_Stats[A1,A2].Qty := Quantity;
         Disp_Stats.Grade_Stats[A1,A2].Amt := Amount
      end
```

File Name: AERPROCS.PAS (cont)

```
   else if Chg_Cat = 4 then
      begin
         A1 := Integer_Value(copy(Chg_Ident,1,2));
         if (Chg_Ident[3] = 'R') or (A1 in [7..10]) then A1 := A1 + 1;
         Disp_Stats.Loan_Cats[A1].Qty := Quantity;
         Disp_Stats.Loan_Cats[A1].Amt := Amount
      end
   else
      begin
         A1 := Integer_Value(Chg_Ident[1]);
         Disp_Stats.Duty_Station[A1].Qty := Quantity;
         Disp_Stats.Duty_Station[A1].Amt := Amount
      end;
   Stats_Record_IO('W',WMon,Disp_Stats)
end;  { internal procedure Apply_Change }

procedure Total_Stats;

var
   T1, T2, End_Mon : integer;
   Temp : General_Stats;

begin
   if CurMon = 1 then
      begin
         Stats_Record_IO('R',0,Disp_Stats); End_Mon := 12
      end
   else
      begin
         Stats_Record_IO('R',1,Disp_Stats); End_Mon := CurMon
      end;
   for T1 := 2 to End_Mon do
      begin
         Stats_Record_IO('R',T1,Temp);
         for T2 := 1 to 9 do with Disp_Stats.Grade_Stats[1,T2] do
            begin
               Qty := Qty + Temp.Grade_Stats[1,T2].Qty;
               Amt := Amt + Temp.Grade_Stats[1,T2].Amt
            end;
         for T2 := 1 to 9 do with Disp_Stats.Grade_Stats[2,T2] do
            begin
               Qty := Qty + Temp.Grade_Stats[2,T2].Qty;
               Amt := Amt + Temp.Grade_Stats[2,T2].Amt
            end;
         for T2 := 1 to 11 do with Disp_Stats.Loan_Cats[T2] do
            begin
               Qty := Qty + Temp.Loan_Cats[T2].Qty;
               Amt := Amt + Temp.Loan_Cats[T2].Amt
            end;
```

```
         for T2 := 1 to 3 do with Disp_Stats.Duty_Station[T2] do
            begin
               Qty := Qty + Temp.Duty_Station[T2].Qty;
               Amt := Amt + Temp.Duty_Station[T2].Amt
            end
      end
end;  { internal Procedure Total_Stats }

procedure Print_Stats;

var
   P1 : integer;

begin
   if Printer_OK = 0 then
      begin
         Prepared_Screen := Display_Screen;
         Display_Window(6,10);
         Regs.AX := $0500;intr($05,Regs);
         for P1 := 1 to 40 do writeln(lst)
      end
end;  { internal procedure Print_Stats }

begin
   WSDate := CSDate; WMon := CurMon; Copt := 0;
   Prepare_Screen(6);
   repeat
      if Copt <> 7 then
         begin
            Prepare_Screen(6); Display_Screen := Prepared_Screen;
            gotoXY(45,1);clrEol; write('MONTH OF ',copy(WSDate,4,9));
            Prepared_Screen := Display_Screen;
            Stats_Record_IO('R',WMon,Disp_Stats);
            Write_Grade; Write_Loan_Cats; Write_Duty_Stations;
            repeat
               Screen_Input(6,1,1); if ESC then Exit;
               Copt := Integer_Value(Field_Contents(6,1));
               if Not(Copt in [1..6]) then Buzzer
            until Copt in [1..6]
         end;
      if Copt in [1,7] then
         begin
            Copt := 1;
            Screen_Input(5,1,1); if ESC then Exit;
            Test_Date := '    ' + Field_Contents(5,1);
            Extract_Date_Data(Test_Date,TMon,D1);
            Code := Date_Difference(CSDate,Test_Date);
```

```
            if (Not(Code in [0..11])) or (D1 > CurDate) then
               begin
                  Display_Window(6,8);
                  if Key_Depressed = 27 then Exit
                  else Display_Screen := Prepared_Screen
               end
            else
               begin
                  WSDate := Test_Date; WMon := TMon
               end
         end
      else if Copt = 2 then
         begin
            gotoXY(45,1);
            if CurMon > 1 then
               write('01 JAN ',((Curdate div 512) + 80):2,' to ',CSDate)
            else
               write('01 JAN ',((Curdate div 512) + 79):2,' to ',
                     '31 DEC ',((Curdate div 512) + 79):2);
            Total_Stats;
            Write_Grade; Write_Loan_Cats; Write_Duty_Stations;
            repeat
               Screen_Input(6,9,9); if ESC then Exit;
               Copt := Integer_Value(Field_Contents(6,9));
               if Not(Copt in [1,2]) then Buzzer;
               if Copt = 2 then Print_Stats
            until Copt = 1;
            Copt := 7
         end
      else if Copt = 3 then
         begin
            Screen_Input(6,3,3);
            Apply_Change(3,Field_Contents(6,3));If ESC then Exit;
            Write_Grade
         end
      else if Copt = 4 then
         begin
            Screen_Input(6,2,2);
            Apply_Change(4,Field_Contents(6,2)); if Esc then Exit;
            Write_Loan_Cats
         end
      else if Copt = 5 then
         begin
            Screen_Input(6,4,4);
            Apply_Change(5,Field_Contents(6,4)); if ESC then Exit;
            Write_Duty_Stations
         end
      else if Copt = 6 then Print_Stats
   until Copt = 8
end;  { Procedure Display_General_Stats }
```

File Name: AERPROCS.PAS (cont)

```
Procedure Seek_Records(Mode_Control : integer);

var
   S1, S2, Line, Current_Ptr, Nr_Loans, Total_Tgts, Diff : integer;
   PDiff, ADiff : integer;
   Stat_Acct : byte;
   ADate, BDate : string[9];
   Amt : real;
   Account : Entire_Account;

begin
   if Loan_Totals[Mode_Control] = 0 then exit;
   Current_Ptr := 1; Total_Tgts := 0; Line := 1;
   if Mode_Control in [7..10] then Stat_Acct := 2
   else Stat_Acct := Mode_Control;
   repeat
      Seek(Index_File,Current_Ptr); read(Index_File,Index);
      if Index.Name <> 'EMPTY' then
         begin
            Get_Account(SSN_Str(Index.SSN),Nr_Loans,Account);
            if Stats_Code[Stat_Acct] <> 0 then
               for S1 := 1 to Nr_Loans do
                  with Account[Rec_Pos[S1]].Loan_Data do
                     begin
                        if Mode_Control in [7..10] then
                           begin
                              Split_Date_and_Money(Balance_Info,
                                                   BDate,Amt);
                              Split_Date_and_Money(Allot_Info,
                                                   Adate,Amt);
                              PDiff := Date_Difference(CSDate,BDate);
                              ADiff := Date_Difference(CSDate,ADate);
                              if PDiff > ADiff then Diff := ADiff
                              else Diff := PDiff;
                              if Diff > 4 then Diff := 4
                           end
                        else Diff := 0;
                        if ((Stat_Acct = Acct_Status) and (Diff = 0)) or
                           ((Acct_Status = 2) and (Diff in [1..4])) then
                           begin
                              Total_Tgts := Total_Tgts + 1;
                              if Line = 1 then
                                 begin
                                    Print_Header(Mode_Control);
                                    Line := 6
                                 end;
                              Print_Report(S1,Account);
                              Line := Line + 1;
```

File Name: AERPROCS.PAS (cont)

```
                              if Line = 60 then
                                 begin
                                    for S2 := 1 to 7 do
                                       writeln(lst);
                                    Line := 1
                                 end
                           end
                     end  {with Account[S1] do}
         end; {if Index.Name <> 'EMPTY'}
      Current_Ptr := Current_Ptr + 1
   until (Total_Tgts=Loan_Totals[Mode_Control]) or (Current_Ptr=5001);
   if Line > 1 then
      while Line < 67 do
         begin
            writeln(lst);
            Line := Line + 1
         end
end; { Procedure Seek_Records }
```

File Name: OVERLAYS.OVR

```
Overlay procedure Close_Files;

begin
   close(Index_File);
   close(Loan_File);
   close(Stats_File);
   close(Accounts_File)
end;  { procedure Close_Files  }

Overlay Procedure Load_Display_Screens_into_Memory;

var
   FormFile : file of Screen_Data;
   Windows : text;
   L1, L2, L3 : integer;
   Screen_Ident : string[2];
   File_Name : string[14];

begin
   if ESC then Exit;
   Assign(FormFile,'FORMS.DTA'); reset(FormFile); L1 := 0;
   while not EOF(FormFile) do
      begin
         seek(FormFile,L1);
         L1 := L1 + 1; read(FormFile,Screen[L1])
      end;
```

File Name: OVERLAYS.OVR (cont)

```
   close(FormFile);
   for L2 := 1 to L1 do
      begin
         if Screen[L2].Field_Posits[160] = 1 then
            begin
               Str(L2,Screen_Ident);
               File_Name := 'WINDOW' + Screen_Ident + '.DTA';
               assign(Windows,File_Name); reset(Windows); L3 := 1;
               while not eof(Windows) do
                  begin
                     readln(Windows,Window_Contents[L2,L3]);
                     L3 := L3 + 1
                  end;
               close(Windows)
            end;
      end
end;  { Procedure Load_Display_Screens_into_Memory }

Overlay Procedure UpDate_Loans;

var
   U1, U2, U3, Nr_Accounts_Read, Nr_Recs : integer;
   Temp_Real : real;
   Temp_Status : byte;
   Diskette_In_Drive : boolean;

begin
   Assign(Index_File,Index_Aer);
   Prepared_Screen := Display_Screen;
   repeat
      {$I-} reset(Index_File) {$I+};
      Diskette_In_Drive := (IOResult = 0);
      if Not(Diskette_In_Drive) then
         begin
            ClrScr; gotoXY(17,10);
            write('I cannot seem to find the "B: Drive Diskette.');
            gotoXY(10,12);
            write('Please verify that the "B: Drive" diskette is in ',
                  'the B Drive. ');
            gotoXY(15,15);
            write('Press any key when the problem has been corrected.');
            repeat
            until KeyPressed
         end
   until Diskette_In_Drive;
   Display_Screen := Prepared_Screen;
   Assign(Loan_File, Loans_AER); reset(Loan_File);
   Assign(Stats_File,GrdStats_AER); reset(Stats_File);
   Assign(Accounts_File,Accounts_AER); reset(Accounts_File);
```

File Name: OVERLAYS.OVR (cont)

```
   read(Index_File,Index_Stats);
   read(Loan_File,Loan_Stats);
   Nr_Recs := Loan_Stats.Prev_Record;
   Print_On := True; Correcting := False;
   Prepare_Screen(3); Display_Screen := Prepared_Screen;
   repeat
      Screen_Input(3,2,2); if ESC then Exit;
      CSDate := Field_Contents(3,2)
   until length(CSDate) = 9;
   Extract_Date_Data(CSDate,CurMon,CurDate);
   Regs.AX := $2B00; Regs.CX := 1900 + Integer_Value(copy(CSDate,8,2));
   Regs.DX := CurMon*100 + integer_Value(copy(CSDate,1,2));
   intr($21,Regs);
   I := Printer_OK;
   ESC := False;
   Textbackground(White);textcolor(Red+Blink);
   gotoXY(3,2);write('Working!');
   Textbackground(blue); Textcolor(white);
   FillChar(Loan_Totals,22,0);
   Boot_Up := True;
   U1 := 0; Nr_Accounts_Read := 0;
   repeat
      U1 := U1 + 1;
      seek(Loan_File,U1); read(Loan_File,Loan);
      with Loan do
         if Acct_Status <> $FF then
            begin
               Nr_Accounts_Read := Nr_Accounts_Read + 1;
               Temp_Status := New_Status('A',Loan);
                  if (Acct_Status = 4) and (Temp_Status = $FF) then
                     begin
                        if (Prev_Record < 0) and (Next_Record = 0) then
                           begin
                              seek(Index_File,abs(Prev_Record));
                              read(Index_File,Index);
                              Delete_Account(abs(Prev_Record))
                           end
                        else Delete_Loan(U1,U3)
                     end
                  else if Acct_Status <> Temp_Status then
                     begin
                        Acct_Status := Temp_Status;
                        seek(Loan_File,U1);
                        write(Loan_File,Loan)
                     end
         end {if Acct_Status <> $FF}
   until (U1 = 5000) or (Nr_Accounts_Read = Nr_Recs);
   Boot_Up := False;
   gotoXY(3,2); write('          ')
end;  { Procedure UpDate_Loans }
```

File Name: OVERLAYS.OVR (cont)

```
Overlay Procedure View_Change_or_Delete;

const
   Header : array[1..8] of String[20] = ('  View an Account',
                                         ' Record Chapter 13',
                                         'Record Uncollectible',
                                         'Record Transfer-Out',
                                         'Delete Paid Off Loan',
                                         'Delete Transfer-Out',
                                         'Delete Uncollectible',
                                         'Correct Loan/Account');
   Descr : array[6..7] of string[14] =
                                    ('Uncollectible.','Transfer-Out.');

var
   Account : Entire_Account;
   Index_Hold : Identification_Record;
   File_Key : string[25];
   Fld, S1, S2, S3, S4, NDX, Action,
   NrLoans, LoanNr, Percent, WMon : integer;
   StrIn : string[3];
   UncDate : String9;
   InReal : real;
   Key_Hit : byte;

begin
   if ESC then Exit;
   Key_Hit := 1;
   repeat
      Prepare_Screen(2); Display_Screen := Prepared_Screen;
      if Key_Hit <> 13 then
         begin
            repeat
               Screen_Input(2,8,8); if ESC then Exit;
               Fld := Integer_Value(Field_Contents(2,8));
               if Not (Fld in [1,2]) then Buzzer
            until Fld in [1,2];
            Fill_Field(2,8,'   ');
            repeat
               Screen_Input(2,9,9); if ESC then Exit;
               Action := Integer_Value(Field_Contents(2,9));
               if Not (Action in [1..8]) then Buzzer
            until Action in [1..8];
            Fill_Field(2,9,' ')
         end;
      gotoXY(60,2); write(Header[Action]);
      Screen_Input(2,3-Fld,3-Fld); if ESC then Exit;
      File_Key := Field_Contents(2,3-Fld);
      Get_Account(File_Key,NrLoans,Account);
```

File Name: OVERLAYS.OVR (cont)

```
      if NrLoans <> 0 then
         begin
            Display_Account_Ident(2); Display_Loans(2,10,1,Account);
            if Not(Action in [1,8]) then
               begin
                  repeat
                     LoanNr := 0;
                     Screen_Input(2,6,6); if ESC then Exit;
                     StrIn := Field_Contents(2,6);
                     if StrIn <> 'ALL' then
                        begin
                           LoanNr := Integer_Value(StrIn);
                           if Not(LoanNr in [1..15]) then
                              begin
                                 Buzzer; LoanNr := 0
                              end
                           else if Account[LoanNr].Rec_Loc = 0 then
                              begin
                                 Buzzer; LoanNr := 0
                              end
                        end
                  until (StrIn = 'ALL') or (LoanNr <> 0);
                  if NrLoans = 1 then StrIn := 'ALL';
                  if StrIn = 'ALL' then
                     begin
                        S1 := 1; LoanNr := 0
                     end
                  else
                     begin
                        S1 := 0;
                        repeat
                           S1 := S1 + 1
                        until Rec_Pos[S1] = LoanNr
                     end;
                  Fill_Field(2,6,'    ')
               end;  { if Action <> 1 }
            if Action = 2 then   {record ch-13 }
               begin
                  repeat
                     Screen_Input(2,5,5); if ESC then Exit;
                     Percent := Integer_Value(Field_Contents(2,5))
                  until Percent in [0..100];
                  Fill_Field(2,5,'    ');
                  repeat
                     with Account[Rec_Pos[S1]].Loan_Data do
                        begin
                           S4 := New_Status('D',
                                       Account[Rec_Pos[S1]].Loan_Data);
                           Acct_Status := 1;
```

```
                        S4 := New_Status('A',
                                    Account[Rec_Pos[S1]].Loan_Data);
                        Repay_Method := Percent
                     end;
                  Display_Loans(2,10,1,Account);
                  S1 := S1 + 1
               until (Rec_Pos[S1] = 0) or (Rec_Pos[S1-1] = LoanNr);
            end  { if Action = 2 }
         else if Action = 3 then    { record uncollectible}
            repeat
               S4 := New_Status('D',Account[Rec_Pos[S1]].Loan_Data);
               Account[Rec_Pos[S1]].Loan_Data.Acct_Status := 3;
               Display_Loans(2,10,1,Account);
               S4 := New_Status('A',Account[Rec_Pos[S1]].Loan_Data);
               S1 := S1 + 1
            until (Rec_Pos[S1] = 0) or (Rec_Pos[S1-1] = LoanNr)
         else if Action = 4 then     { record transfer-out}
            repeat
               S4 := New_Status('D',Account[Rec_Pos[S1]].Loan_Data);
               Account[Rec_Pos[S1]].Loan_Data.Acct_Status := 6;
               Display_Loans(2,10,1,Account);
               S4 := New_Status('A',Account[Rec_Pos[S1]].Loan_Data);
               S1 := S1 + 1
            until (Rec_Pos[S1] = 0) or (Rec_Pos[S1-1] = LoanNr)
         else if Action in [5..7] then
            begin
               if Action = 5 then NDX := 4
               else if Action = 6 then NDX := 3
               else NDX := 6;
               gotoXY(1,21);
               if (StrIn='ALL') and (NrLoans <> Stats_Code[NDX]) then
                  Write('Sorry, I can only delete accounts when ',
                        'ALL loans are declared ',Descr[Action])
               else
            if (Account[Rec_Pos[S1]].Loan_Data.Acct_Status<>NDX) then
                  write('Sorry, Loan ',Rec_Pos[S1]:2,
                        ' has not yet been declared ',Descr[Action],
                        '  I cannot delete it.')
               else
                  begin
                     if Action in [6,7] then
                        begin
                           gotoXY(1,22);
                           if StrIn = 'ALL' then
                              write('Date Account Approved ',
                                    Descr[Action])
                           else
                              write('Date Loan ',Rec_Pos[S1]:2,
                                    ' Approved ',Descr[Action]);
```

File Name: OVERLAYS.OVR (cont)

```
                        Screen_Input(4,52,52); if ESC then Exit;
                        UncDate := Field_Contents(4,52);
                        Extract_Date_Data(UncDate,WMon,Code);
                        S2 := S1;
                        repeat
                           with Account[Rec_Pos[S2]].Loan_Data do
                              Split_Date_and_Money(Balance_Info,
                                                   Date,InReal);
                           Ledger(6,25-Action,Wmon,InReal);
                           S2 := S2 + 1
                        until (Rec_Pos[S2] = 0) or
                                        (Rec_Pos[S2-1] = LoanNr);
                     end;
                  if StrIn = 'ALL' then
                        Delete_Account(FilePos(Index_File) - 1)
                  else Delete_Loan(Account[LoanNr].Rec_Loc,Code);
                  Get_Account(File_Key,NrLoans,Account);
                  Prepare_Screen(2);
                  Display_Screen := Prepared_Screen;
                  gotoXY(60,2); write(Header[Action]);
                  if NrLoans <> 0 then
                     begin
                        Display_Account_Ident(2);
                        Display_Loans(2,10,1,Account)
                     end;
                  gotoXY(5,21);
                  if StrIn = 'ALL' then
                     write('Account ',File_Key,
                           ' has been removed from my memory.')
                  else
                     write('Loan Nr ',LoanNr:2,
                           ' has been removed from my memory.')
               end
            end
         else if Action = 8 then
            begin
               KBSB := KBSB and $DF;
               gotoXY(6,22); write(chr(24)); gotoXY(1,23);
               write('Use ',chr(27),' ',chr(26),
                     ' keys to select item to correct.');
               gotoXY(6,24); write(chr(25)); Correcting := True;
               Prepared_Screen := Display_Screen;
               repeat
                  S2 := Key_Depressed;
               until (hi(Regs.AX) in [72,75,77,80]) or (ESC);
               if ESC then Exit;
               S1 := 1;
               repeat
                  Scan_Code := 0;
```

File Name: OVERLAYS.OVR (cont)

```
                    Screen_Input(2,S1,S1); if ESC then Exit;
                    if ((Scan_Code = 72) and (whereY = 2)) or
                       ((Scan_Code = 75) and (whereX < 8)) or
                       ((Scan_Code = 77) and (whereX in [38,51])) or
                       ((Scan_Code = 80) and (S1 > NrLoans*7 + 3)) then
                          begin
                             Buzzer;
                             Display_Screen := Prepared_Screen
                          end
                    else if Scan_Code in [72,75,77,80] then
                       begin
                          Display_Screen := Prepared_Screen;
                          if (Scan_Code = 72) and (S1 > 15) then
                             S1 := S1 - 7
                          else if Scan_Code = 72 then S1 := 1
                          else if (Scan_Code = 80) and (S1 > 10) then
                             S1 := S1 + 7
                          else if Scan_Code = 80 then S1 := 11
                          else if Scan_Code = 75 then S1 := S1 - 1
                          else S1 := S1 + 1
                       end;
                 until Not(Scan_Code in [72,75,77,80]);
                 if S1 < 5 then
                    begin
                       Index_Hold := Index;
                       Index_Hold.Name := Field_Contents(2,1);
                       Index_Hold.SSN :=
                                        Real_Value(Field_Contents(2,2));
                       StrIn := Field_Contents(2,3);
                       UncDate := Field_Contents(2,4);
                       with Index_Hold do
                          Grade_and_Status :=
                             Encode_Grade_and_Status(StrIn,UncDate[1]);
                       Delete_Account(FilePos(Index_File) - 1);
                       Index := Index_Hold;
                       for S2 := 1 to NrLoans do
                          begin
                             Loan := Account[Rec_Pos[S2]].Loan_Data;
                             Write_Index_Record;
                             S3 := New_Status('A',Loan)
                          end
                    end
                 else with Account[Rec_Pos[(S1-2) div 7]].Loan_Data do
                    begin
                       S2 := (S1-2) mod 7;
                       S4 := Acct_Status;
                       S3 := New_Status('D',
                             Account[Rec_Pos[(S1-2) div 7]].Loan_Data);
```

File Name: OVERLAYS.OVR (cont)

```
                        if S2 = 2 then
                          Split_Date_and_Money(Loan_Info,UncDate,InReal)
                        else if S2 = 3 then
                           Split_Date_and_Money(Balance_Info,UncDate,
                                                              InReal)
                        else if S2 = 4 then
                           begin
                              StrIn := Field_Contents(2,S1);
                              if StrIn[1] = 'A' then
                                 begin
                                    Repay_Method := 0;
                                    S4 := 0
                                 end
                              else if StrIn[1] = 'P' then
                                 begin
                                    Repay_Method := $80;
                                    S4 := 0
                                 end
                           end
                        else if S2 = 5 then
                           Split_Date_and_Money(Allot_Info,UncDate,
                                                              InReal)
                        else if S2 = 6 then
                           begin
                              Split_Date_and_Money(Balance_Info,UncDate,
                                                             InReal);
                              UncDate := Field_Contents(2,S1);
                              Balance_Info :=
                                    Merge_Date_and_Money(UncDate,InReal)
                           end;
                        if S2 in [2,3,5] then
                           begin
                             InReal := Real_Value(Field_Contents(2,S1));
                              if S2 = 2 then
                                 Loan_Info :=
                                    Merge_Date_and_Money(UncDate,Inreal)
                              else if S2 = 3 then
                                 begin
                                    if InReal = 0.0 then S4 := 4
                                    else if (Inreal > 0.0) and
                                                          (S4 = 4) then
                                       S4 := 0;
                                    Balance_Info :=
                                    Merge_Date_and_Money(UncDate,Inreal)
                                 end
                              else
                                 Allot_Info :=
                                    Merge_Date_and_Money(UncDate,Inreal)
                           end;
```

File Name: OVERLAYS.OVR (cont)

```
                        Acct_Status := S4;
                        Acct_Status := New_Status('A',
                              Account[Rec_Pos[(S1-2) div 7]].Loan_Data);
                        seek(Loan_File,
                                Account[Rec_Pos[(S1-2) div 7]].Rec_Loc);
                        write(Loan_File,
                              Account[Rec_Pos[(S1-2) div 7]].Loan_Data);
                        flush(Loan_File)
                     end;
                  Get_Account(SSN_Str(Index.SSN),NrLoans,Account);
                  Display_Account_Ident(2);
                  Display_Loans(2,10,1,Account);
                  gotoXY(1,22); ClrEol; gotoXY(1,23);
                  ClrEol; gotoXY(1,24);
                  ClrEol; Correcting := False;
                  KBSB := KBSB or $20
               end;  { if Action = 8 }
            S1 := 0;
            if Action in [2..4] then
               repeat
                  S1 := S1 + 1;
                  seek(Loan_File,Account[Rec_Pos[S1]].Rec_Loc);
                  write(Loan_File,Account[Rec_Pos[S1]].Loan_Data)
               until Rec_Pos[S1+1] = 0
         end { if NrLoans <> 0 }
      else
         begin
            gotoXY(14,21);
            write('Sorry, I do not appear to have the ',
                  'requested account.')
         end;
      gotoXY(5,23);
      write('Press ',chr(17),'┘ to continue the same operation (',
             Header[Action],').');
      gotoXY(5,25);
      write('Press any other key to select another operation ',
            '(ESC to Exit).');
      Key_Hit := Key_Depressed;
   until Key_Hit = 27
end;  { Procedure View_Change_or_Delete }
```

APPENDIX C

APPLICATION PROGRAM DISPLAY SCREEN DESIGN SOURCE CODE

The following, undocumented, application program source code is written in Borland International, Inc., Turbo Pascal™, version 3.0.

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

File Name: FORMDRAW.PAS

```
type
   scrnline = array[1..160] of byte;
   Scrnarray = array[1..25] of scrnline;
   Screen_Data = record
      Screen_Image : Scrnarray;
      Field_Posits : ScrnLine;
      Window_Info : ScrnLine
   end; {record Screen_Data}
   String80 = string[80];
   CPU_Registers = record
      AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : integer
   end;

var
   Regs : CPU_Registers;
   Screen : Screen_Data;
   Window_Data : array[1..25,1..25] of String80;
   Temp_String : String80;
   Temp_Window_Info : scrnline;
   scrn : scrnarray absolute $B800:$0000; {$B000 for monochrome}
   Formfile : file of Screen_Data;
   Windows : Text;
   I, I1, I2, J, K, L : integer;
   Diff,   Display_Memory,   Lines_of_Windows,   scrnr,Nr_of_Screens   :
integer;    Entry_Pt,Width,Xpos,Ypos, Last : byte;
   Opt : char;
   Delete, Change, New_Screen, Screen_Mode : boolean;
   scrnr_str : string[2];

Procedure Screen_Draw(Mode : boolean);

var
   Fore, Back : byte;
   Attribute_Only : boolean;

begin
   Fore := $0F; Back := $00; PortW[$03D8] := $09;
   Attribute_Only := False;
   repeat
      I := whereX; J := WhereY; Regs.AX := $0000; intr($16,regs);
      with regs do
         if lo(AX) in [16,17,32..255] then
            begin
               if not Attribute_Only then scrn[J,2*I-1] := lo(AX);
               scrn[J,2*I] := Back or Fore;
               I := I + 1;
               Last := lo(AX)
            end
```

File Name: FORMDRAW.PAS (cont)

```
        else if lo(AX) = 1 then
           begin
              if Attribute_Only then Attribute_Only := False
              else Attribute_Only := True
           end
        else if lo(AX) = 2 then
           begin
              for J := 1 to 25 do
                 for I := 1 to 80 do
                    scrn[J,I*2] := (Scrn[J,I*2] and $0F) or Back;
              J := 1; I := 1;
              gotoXY(I,J)
           end
        else if (lo(AX) = 19) and (Change) and (Mode) then
           begin
              Screen.Field_Posits[I1] := 2*whereX - 1;
              Screen.Field_Posits[I1+1] := whereY;
              I1 := I1 + 2
           end
        else if (lo(AX) = 19) and (Change) and (not (Mode)) then
           begin
              Screen.Window_Info[I1] := whereX;
              Screen.Window_Info[I1+1] := whereY
           end
        else if (lo(AX) = 5) and (Change) and (Mode) then
           begin
              with Screen do
                 Field_Posits[I1] := 2*whereX - Field_Posits[I1-2];
              I1:=I1 + 1
           end
        else if (lo(AX) = 5) and (Change) and (not (Mode)) then
           begin
              with Screen do
                 Window_Info[I1+2] := J - Window_Info[I1+1] + 1;
              width := whereX
           end
        else if (lo(AX) = 4) and (Change) and ( not (mode)) then
           with Screen do
              begin
                 Window_Info[I1] := 0;
                 Window_Info[I1+1] := 0;
                 Window_Info[I1+2] := 0;
                 Window_Info[I1+3] := 0;
                 Delete := True;
                 Exit
              end
```

```
      else if (lo(AX) = 3) and (Change) and (Mode) then
         begin
            with Screen do
               Field_Posits[I1] :=
                           $80 or (2*whereX - Field_Posits[I1 - 2]);
            I1 := I1 + 1
         end
      else if (hi(AX) = 72) and (J <> 1) then J := J - 1
      else if (hi(AX) = 80) and (J <> 25) then J := J + 1
      else if (hi(AX) = 75) and (I <> 1) then I := I - 1
      else if (hi(AX) = 77) and (I <> 80) then I := I + 1
      else if hi(AX) = 71 then
         begin
            I := 1;
            J := 1
         end
      else if hi(AX) = 79 then I := 80
      else if hi(AX) = 73 then J := 1
      else if hi(AX) = 81 then J := 25
      else if (hi(AX) = 28) then I := 1
      else if hi(AX) = 14 then
         begin
            scrn[J,2*I-1] := $20;
            I := I - 1
         end
      else if hi(AX) = 94 then
         begin
            if Not Attribute_Only then scrn[J,2*I-1] := Last;
            scrn[J,2*I] := Back or Fore;
            I := I + 1
         end
      else if hi(AX) = 59 then Back := $00
      else if hi(AX) = 60 then Back := $10
      else if hi(AX) = 61 then Back := $20
      else if hi(AX) = 62 then Back := $30
      else if hi(AX) = 63 then Back := $40
      else if hi(AX) = 64 then Back := $50
      else if hi(AX) = 65 then Back := $60
      else if hi(AX) = 66 then Back := $70
      else if hi(AX) = 104 then Fore := $00
      else if hi(AX) = 105 then Fore := $01
      else if hi(AX) = 106 then Fore := $02
      else if hi(AX) = 107 then Fore := $03
      else if hi(AX) = 108 then Fore := $04
      else if hi(AX) = 109 then Fore := $05
      else if hi(AX) = 110 then Fore := $06
      else if hi(AX) = 111 then Fore := $07
      else if hi(AX) = 112 then Fore := Fore and $07
```

File Name: FORMDRAW.PAS (cont)

```
         else if hi(AX) = 113 then Fore := Fore or $08
         else if hi(AX) = 67 then Back := Back and $70
         else if hi(AX) = 68 then Back := Back or $80
         else if hi(AX) = 96 then
            begin
               J := J + 1;
               gotoXY(I-1,J);
               if Not Attribute_Only then scrn[J,2*I-1] := Last;
               scrn[J,2*I] := Back or Fore
            end;
      gotoXY(I,J)
   until lo(Regs.AX) = 27
end; { Internal Procedure Screen_Draw }

procedure Display_Window(Xcoord,Ycoord:byte;DisplayString : String80);

var
   X,Y, Offset : integer;

begin
   X := Xcoord; Y := Ycoord;
   Offset := (Y - 1)*160 + 2*(X - 1);
      inline(
         $50/$51/$57/$56/$06/$9C/              {PUSH AX,CX,DI,SI,ES,Flags}

         $B8/$00/$B8/                          {MOV AX, B800 }
         $50/                                  {PUSH AX}
         $07/                                  {POP ES}
         $8B/$BE/Offset/                       {MOV DI,[BP+Offset]}
         $8D/$B6/DisplayString/                {LEA SI,[BP+DisplayString]}
         $31/$C9/                              {XOR CX,CX}
         $36/                                  {SS:}
         $8A/$0C/                              {MOV CL,[SI]}
         $46/                                  {INC SI}
         $FC/                                  {CLD}
         $36/$A4/                         {L1: SS:MOVSB}
         $E2/$FC/                              {LOOP L1}
         $9D/$07/$5E/$5F/$59/$58)              {POP Flags,ES,SI,DI,CX,AX}
end;    { Internal Procedure Display_Window }


begin  { Main Program }
   assign(FormFile,'FORMS.DTA'); New_Screen := False;
   {$I-} reset(FormFile) {$I+};
   if IOresult <> 0 then
      begin
         rewrite(FormFile); FillChar(Screen.Field_Posits,160,0);
         FillChar(Screen.Window_Info,160,0);scrnr := 1
      end
```

File Name: FORMDRAW.PAS (cont)

```
   else
      begin
         clrscr;
         Nr_of_Screens := FileSize(FormFile);
         writeln('Number of Screens in FORMS.DTA: ',Nr_of_Screens);
         write('Screen, Window or Quit (S, W or Q) ');readln(opt);
         if opt in ['q','Q'] then
            begin
               close(Formfile);exit
            end;
         if opt in ['S','s'] then Screen_Mode := True
         else Screen_Mode := False;
         write('Screen # to bring up ');readln(scrnr);
         if (scrnr > Nr_of_Screens) and (Screen_Mode) then
             begin
                writeln('New Screen.  Screen number is ',
                        Nr_of_Screens + 1);
                scrnr := Nr_of_Screens + 1;New_Screen := True
             end
         else
            if (scrnr > Nr_of_Screens) and (not (Screen_Mode)) then
               exit;
          if Not New_Screen then
            begin
               write('Change control settings? ');read(opt);
               if (opt = 'y') or (opt = 'Y') then Change := True
               else Change := False;
               clrscr;
               seek(FormFile,scrnr-1);
               read(FormFile,Screen);
               if (Change) and (Screen_Mode) then
                  FillChar(Screen.Field_Posits,160,0)
            end
         else FillChar(Screen,4000,0);
      end;
   Scrn := Screen.Screen_Image;
   if Screen_Mode then
      begin
         I1 := 1; gotoXY(1,1); Screen_Draw(Screen_Mode);
         Screen.Screen_Image := Scrn
      end
   else
      begin
         for I:= 1 to 25 do
            for J := 1 to 20 do
            Window_Data[I,J]:= 'Empty';
         J := 1;
         Str(scrnr,scrnr_str);
```

```
        Temp_String := 'Window' + scrnr_str + '.DTA';
        assign(Windows,Temp_String);
        {$I-} reset(Windows) {$I+};
        if IOresult <> 0 then
           begin
              rewrite(Windows);FillChar(Screen.Window_Info,160,0)
           end
        else while not eof(Windows) do
           begin
              if Screen.Window_Info[J*4-1] <> 0 then
                 for I:= 1 to Screen.Window_Info[J*4-1] do
                    readln(Windows,Window_Data[J,I]);
              J := J + 1
           end;
        repeat
           Delete := False;
           scrn := Screen.Screen_Image;
           gotoXY(1,25); write('Window Number ?  (0 to exit) ');
           read(I1);
           if I1 <> 0 then
              begin
                 I2 := I1; I1 := I1*4 - 3; gotoXY(20,12);
                 Temp_Window_Info := Screen.Window_Info;
                 if Screen.Window_Info[I1] <> 0 then
                    for I := 1 to Screen.Window_Info[I1+2] do
                       with Screen do
                           Display_Window(Window_Info[I1],
                                          Window_Info[I1+1]+I-1,
                                          Window_Data[I2,I]);
                 Screen_Draw(Screen_Mode);
                 if Not(Delete) then
                    begin
                       Window_Data[I2,1] := '';  K := 1;
                       I := Screen.Window_Info[I2*4-2];
                       repeat
                          Window_Data[I2,K] := '';
                          J := (Screen.Window_Info[I2*4-3] shl 1) - 1;
                          L := J;
                          repeat
                             Window_Data[I2,K] := Window_Data[I2,K] +
                                                  chr(scrn[I,J]) +
                                                  char(scrn[I,J+1]);
                             J := J + 2
                          until (scrn[I,J-2] in [186,187,188]) and
                                (J-2 > L);
                          I := I + 1;
                          K := K + 1
                       until scrn[I-1,J-2] = 188;
```

File Name: FORMDRAW.PAS (cont)

```
                        Screen.Window_Info[4*I2-1] := K - 1
                     end
               end;
            Entry_Pt := 1;
            for I := 1 to 40 do with screen do
               if Window_Info[4*I-1] <> 0 then
                  begin
                     Window_Info[4*I] := Entry_Pt;
                     Entry_Pt := Window_Info[4*I-1] + Window_Info[4*I]
                  end
         until I1 = 0
      end;
   clrscr;
   write('Save to File ? (Y/N) ');
   read(Opt);
    if (Not (Screen_Mode)) and (upcase(Opt) = 'Y') then
      begin
         rewrite(Windows);
         for I := 1 to 25 do
            if Screen.Window_Info[4*I-1] <> 0 then
               for J := 1 to Screen.Window_Info[4*I-1] do
                  writeln(Windows,Window_Data[I,J]);
         close(Windows);
         Screen.Field_Posits[160] := 1
      end;
   if upcase(Opt) = 'Y' then
      begin
         if New_Screen then
            Seek(FormFile,FileSize(FormFile))
         else
            seek(FormFile,scrnr-1);
         write(formfile,Screen)
      end;
   close(FormFile);
   clrscr
end. { Main Program }
```

LIST OF REFERENCES

1. Coombs, M. J. and Alty, J. L. (Eds), Computing Skills and the User Interface, Academic Press, Inc., 1981.

2. Sime, M. E. and Coombs, M. J. (Eds), Designing for Human-Computer Communication, Academic Press, Inc., 1983.

3. James, E. B., "The User Interface: How We May Compute", In Coombs, M. J. and Alty, J. L. (Eds), Computing Skills and the User Interface, Academic Press, Inc., 1981.

4. Sutherland, I. E. and Mead, C. A., "Microelectronics and Computer Science", In Scientific American (eds) Microelectronics, W. H. Freeman, 1977.

5. Gaines, B. R. and Shaw, M. L. G., "Dialog Engineering", In Sime, M. E. and Coombs, M. J. (Eds), Designing for Human-Computer Communication, Academic Press, Inc., 1983.

6. Shneiderman, B., Software Psychology, Winthrop Publishers, Inc., 1980.

7. Eason, K. D. and Damodaran, L., "The Needs of the Commercial User", In Coombs, M. J. and Alty, J. L. (Eds), Computing Skills and the User Interface, Academic Press, Inc., 1981.

8. Peterson, J. L. and Silberschatz, A., Operating System Concepts, 2d ed., Addison-Wesley Publishing Company, Inc., 1985.

9. Reid, P., "Work Station Design, Activities and Display Techniques", In Monk, A. (ed), Fundamentals of Human-Computer Interaction, Academic Press, Inc., 1984.

10. Bolt, R. A., The Human Interface Where People and Computers Meet, Lifetime Learning Publications, 1984.

11. Stoner, J. A. F., Management, 2d ed., Prentice-Hall, Inc., 1982.

12. Sayles, L. R. and Strauss, G., Human Behavior in Organizations, Prentice-Hall, Inc., 1966.

13. Card, S. K., Moran, T. P. and Newell, A., The Psychology of Human-Computer Interaction, Lawrence Erlbaum Associates, Inc., 1983.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Dr. Tung X. Bui, Code 54Bd Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943-5000	1
4.	Dr. T. R. Sivasankaran, Code 54Sj Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943-5000	1
5.	Computer Technology Programs, Code 37 Naval Postgraduate School Monterey, California 93943-5000	1
6.	LCDR D. C. Moore 3206 Westbourne Dr. Antioch, California 94123	4

Thesis
M762
c.1

Moore
Microcomputer program
design considerations for
the novice user.